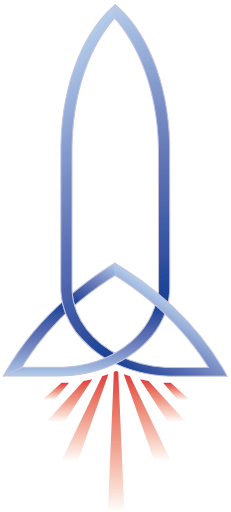
SpaceWire-2011

Proceedings of the 4th International SpaceWire Conference

San Antonio 2011

Editors: Steve Parkes, Allison Bertrand, Martin Suess,
Glenn Rakow
Editorial Assistant: Lisa Rodway



Space
Technology
Centre
University of Dundee

SpaceWire-2011
Proceedings of International SpaceWire Conference
San Antonio 2011

ISBN: 978-0-9557196-3-9

Preface

These proceedings contain the papers presented at the 2011 International SpaceWire Conference, held in San Antonio, Texas, USA between 8 and 10 November, 2011. The International SpaceWire Conference aims to bring together SpaceWire product designers, hardware engineers, software engineers, system developers, mission specialists and academics interested in and working with SpaceWire, to share the latest ideas and developments related to SpaceWire technology.

SpaceWire is a spacecraft on-board communication network designed to connect together instruments, mass-memory, processors, downlink telemetry, and other on-board sub-systems. It offers high-speed, low-power, simplicity, relatively low implementation cost, and architectural flexibility making it ideal for many space missions. Since the SpaceWire standard was published in January 2003, it has been adopted by ESA, NASA, JAXA and RosCosmos, and is being widely used on scientific, Earth observation, commercial and other spacecraft. High-profile missions using SpaceWire include: Gaia, Sentinels 1, 2, 3 and 5 precursor, Bepi-Colombo, James Webb Space Telescope, GOES-R, Lunar Reconnaissance Orbiter and Astro-H.

The conference provides a forum for the exchange of experiences with the application of SpaceWire, and for the exploration of new ideas and technologies related to SpaceWire. It also allows presentations on the latest test and development equipment, chips and IP cores, and software associated with SpaceWire.

This year the conference is hosted by Southwest Research Institute (SwRI), one of the oldest and largest independent, nonprofit, applied research and development (R&D) organizations in the United States. Founded in 1947, with headquarters in San Antonio, Texas, SwRI provides contract research and development services to industrial and government clients. SwRI consists of 11 technical divisions that offer multidisciplinary, problem-solving services in a variety of areas in engineering and the physical sciences. Southwest Research Institute (SwRI) spacecraft avionics have an excellent performance record, having flown on over 50 missions without a single on-orbit failure. Space Missions using SwRI spacecraft avionics include IMAGE, SWIFT, Deep Impact, Orbital Express, Kepler, and MMS.

The Conference Chairpersons would like to acknowledge the support and hard work of many of the individuals who made the International SpaceWire Conference 2011 possible. First, we thank the authors and keynote speakers for their high-quality contributions. We express our gratitude to the members of the Technical Committee for their assistance in the review process. We thank all the people supporting the conference at SWRI, the Space Technology Centre at the University of Dundee, the European Space Agency (ESA) and National Aeronautics and Space Administration (NASA).

The Conference Chairpersons,

Allison Bertrand, *Southwest Research Institute, USA*
Steve Parkes, *Space Technology Centre, University of Dundee, UK*
Martin Suess, *European Space Agency, The Netherlands*
Glenn Rakow, *National Aeronautics and Space Administration, USA*

Technical Committee

Allison Bertrand, South West Research Institute, USA

Barry Cook - 4 Links Ltd., UK

Stephane Davy – Syderal, Switzerland

Omar Emam - Astrium, UK

Wahida Gasti - ESA, The Netherlands

Daniel Gilley - Lockheed Martin, USA

Alain Girard - Thales Alenia Space, France

Viacheslav Grishin- Submicron PLC, Russia

Sev Gunes-Lasnet - Astrium, France

Omar Haddad - Dell, USA

Hiroki Hihara - NEC, Japan

Christophe Honvault - ESA

Torbjörn Hult - RUAG Space, Sweden

Jørgen Ilstad - ESA, The Netherlands

Paul Jaffe - Naval Research Laboratory, USA

David Jameux- ESA, The Netherlands

Gerald Kempf - RUAG Space, Austria

Clifford Kimmery - Honeywell Inc., USA

Alexander Kisin - MEI, USA

Robert Klar - South West Research Institute, USA

Jim Lux - NASA JPL, USA

Patrick McGuirk - PnP Innovations, USA

Peter Mendham - Scisys Ltd., UK

Masaharu Nomachi - University of Osaka, Japan

Olivier Notebaert - Astrium SAS, France

Steve Parkes - University of Dundee, Scotland, UK

Manuel Prieto - Alcala University, Spain

Glenn Rakow - NASA GSFC, USA

Paul Rastetter - Astrium GmbH, Germany

Derek Schierlmann - Naval Research Laboratory, USA

Alan Senior - SEA, UK

Yuriy Sheynin - St. Petersburg State University of Aerospace Instrumentation, Russia

Tatiana Solokhina - ELVEES, Russia

Martin Suess - ESA, The Netherlands

Tadayuki Takahashi - JAXA, Japan

Raffaele Vitulli - ESA, The Netherlands

Takahiro Yamada - JAXA/ISAS, Japan

Tuesday 8 November

Keynote Presentation

BIOGRAPHY

Last Revised October 2011

Mr. BRET G. DRAKE
Deputy Chief Architect, Human Spaceflight Architecture Team,
Exploration Missions and Systems Office,
NASA Johnson Space Center

Mr. Drake is currently leading the future Mission Planning and Analysis activities for the Exploration Missions and Systems Office at NASA's Johnson Space Center. For the past several years Mr. Drake has led the Agency in the design and analysis of human exploration mission approaches beyond low-Earth Orbit including missions to the Moon, Near-Earth Objects, and Mars. Mr. Drake has been involved in various agency strategic planning activities for NASA's exploration efforts for over twenty years including the NASA 90-day study and the White House Synthesis group, Integrated Space Plan, Exploration Systems Architecture Study, and the Review of Human Space Flight Plans Committee (aka Augustine Committee).

Previously, Mr. Drake served as Chief of the Advanced Missions & System Design Office for the Constellation Program and Interim Program Manager for the Lunar Prospector mission at NASA Headquarters and has served as the project lead for many design efforts at the Johnson Space Center. Mr. Drake graduated from the University of Texas at Austin with a Bachelor of Science degree in 1984 with a degree in Aerospace Engineering.

NASA'S HUMAN EXPLORATION PLANS & ARCHITECTURE

Bret G. Drake

Abstract

During the past few years the direction for future human exploration beyond low-Earth orbit has undergone revision and a less destination specific framework has emerged. This strategy, referred to as a Capability Driven Framework, is based on the idea of an ever expanding human presence beyond low-Earth orbit in terms of duration and distance from the Earth. It is based on evolving capabilities which are utilized after operational experience has been established from completing less demanding missions. In theory, the Capability Driven Framework enables multiple destinations and provides increased flexibility, greater cost effectiveness, and sustainability. This presentation will provide an overview of the Capability Driven Framework which is NASA's approach towards developing a robust human spaceflight program that is sustainable over long spans of time. Understanding future exploration needs consistent with the Capability Driven Framework will help guide research activities on the International Space Station, identifying key technology needs, and establishing future collaboration with international partners, academia, and industry, which are essential in maintaining progressive cadence of missions that ultimately land humans on Mars.

Networks and Protocols 1

A GENERALIZED APPROACH TO PLUG-AND-PLAY NETWORK ATTACHED STORAGE USING SPACEWIRE

Session: SpaceWire Networks and Protocols

Long Paper

Paul B. Wood, Sue A. Baldor, Dan Goes, Allison R. Bertrand

Southwest Research Institute®, 6220 Culebra Road, San Antonio, Texas 78238

E-mail: paul.wood@swri.org, sue.baldor@swri.org, dan.goes@swri.org, allison.bertrand@swri.org

ABSTRACT

This paper describes a generalized approach to defining a protocol for Plug-and-Play (PnP) Network Attached Storage (NAS) using SpaceWire. A key concern in the design of a PnP device is the presence of competing standards from the U.S. and the European Union for PnP on SpaceWire (SpW) networks. Providing adaptation between these protocols would allow an end device such as an NAS to operate on a network conforming to either standard. To validate the adaptation layer, a test bed is being developed. The test bed will include a simulated NAS connected on a SpaceWire network. The network will also include producer and consumer nodes to store and retrieve data from the simulated NAS as a challenge task.

1 SPACE PNP UBIQUITY – THE FUTURE IS NOW!

Plug-and-Play (PnP) technologies have been widely adopted for terrestrial computing applications. Over the last two decades, terrestrial PnP technology has transitioned from the sarcastically monikered "Plug and Pray" to the ubiquitous Universal Serial Bus (USB) that allows the vast majority of devices to be connected to a typical workstation with no special actions required on the user's part. These capabilities commonly extend to servers and associated devices on local area networks, a scope comparable with a typical spacecraft (S/C) data system configuration.

Two PnP protocol options for SpaceWire (SpW) are currently maturing – one defined by the Air Force Research Laboratory (AFRL) and one developed by the European Space Agency (ESA). From a practical perspective, we wanted to address the concern that developers and managers face with regards to selecting a specific technology to target for PnP-enabled SpW networking. Our approach to solving this problem has been to abstract the PnP protocol layer such that application programs can interact with either PnP protocol without a need to be aware of unique aspects of either protocol. This enables the goal of developing portable applications that can be reused without modification.

We selected Network Attached Storage (NAS) as a challenge task to use for experiments on our adaptation layer. An NAS device can accept data from many sources and provide data to many sinks. The use of an NAS device varies from a common mission design pattern where mission processors typically include local storage to buffer data prior to transmission to the spacecraft processor. With an NAS device, the data server portion of the spacecraft is separated from the mission and S/C processors. Delineating the data storage as a separate device on the network allows for more flexible network design (including variations in which devices produce data and which devices consume data).

2 RECONCILING DISPARATE STANDARDS

In view of the desire to build an end device that can operate in networks constructed according to either of the SpW-PnP standards, we first sought to understand how these standards are similar and different. The Spacecraft Onboard Interface Services (SOIS)/SpaceWire-PnP and Space Plug-and-Play Architecture (SPA)/SPA-SpaceWire (SPA-S) standards have similar goals and, not surprisingly, they share a number of characteristics in common. These characteristics are listed in Table 1 and include such things as mechanisms to adapt to network changes, Electronic Data Sheets (EDS) allowing devices to self-describe their capabilities, and a central database of capabilities and services available by devices (once discovered) on the network. Table 2 shows areas of difference. Differences range from policy items (International Traffic in Arms Regulations [ITAR] restrictions) to unique data protocols (e.g., Remote Memory Access Protocol [RMAP]) to the maturity of a reference implementation. Note that our work is limited to areas of the standards that affect implementation and operation (e.g., device discovery differences).

Table 1. SOIS/SpaceWire-PnP and SPA/SPA-S Similarities

Attribute	Description
Network identification	Devices connected to networks are given unique network identifiers.
Network changes	Network is able to adapt to topology and composition changes as they occur.
Data sheets	Both use an Electronic Data Sheet (EDS) format with similarities between the two formats.
Capability database	A central service is in charge of maintaining lookup of capabilities provided and services requested by registered devices.
Self-description	Devices self-describe capabilities and needs through their EDS.
Device dictionaries	Standardized virtual device information repositories (Common Data Dictionary and Dictionary of Terms) are used, and evidence that these two dictionaries will align is given.
Code generation	Automated code generation for communicating with devices based on EDS is used.

Table 2. SOIS/SpaceWire-PnP and SPA/SPA-S Differences

Attribute	SOIS/ SpaceWire-PnP	SPA/SPA-S
Source of standard	Primarily coming from ESA	Primarily coming from AFRL
ITAR restriction	Non-ITAR restricted	Some ITAR restrictions
Reference implementation	Reference implementation currently unavailable (but slated for release in August 2011)	Implementation based on earlier standard available (Satellite Data Model [SDM]), implementation based on new standard is now or will soon be available (SPA Services Manager [SSM])
Relationship to Plug-and-Play	SpaceWire-PnP is a realization of the subnetwork portion of the standard and compliant with the standard, but not built into the core of the standard itself	Plug-and-Play concepts and functions built into the foundation of SPA
Device discovery	"Active nodes" on network initiate device discovery	Network managers (SPA managers) initiate device discovery
System management messages	System messages (for discovery, configuration, etc.) are RMAP-based	System messages have their own proprietary packet format over SpaceWire
Network sub-types	Different requirements of devices based on composition of network (level 1 vs. level 2)	Single network level without requiring different features of devices based on network composition
Node types	Distinction between passive and active nodes for ability to initiate commands	Distinction between architectural components of the network (e.g. sub-network managers vs. endpoints) in ability to control network

Table 3 shows a minimal set of candidate functions needed to perform PnP adaptation. Functions are provided to initialize the adaptation layer and configure the device. Ideally, initialization will be able to self-discover the underlying PnP protocol; however, at this time, the Application Program Interface (API) takes an argument that determines whether the network is SPA-S or SpW-PnP. Device configuration hides the unique aspects of the two underlying PnP protocols. For a SPA network, this means replying to the probe message — sent by the SPA Lookup Service — with a message containing the component's Universally Unique Identifier (UUID) and the UUID for its Extensible Transducer Electronic Data Sheet (xTEDS). For an SpW-PnP network, this means contacting the device identification service for a device's configuration. In SPA, the lookup service must be interrogated to establish the connection. In SpW-PnP, the sources and sinks must work through the network manager to establish the connection. Functions in the adaptation API are also provided to make connections, identify when connections are ready, and send and receive data across either type of network (e.g., using the lookup service vs. the network manager).

Table 3. Minimal API for Adaptation

Function	Description
spnp_init(network type)	Configures the SpaceWire abstraction library.
spnp_device_configure(device id, network type)	Configures the device for the network type.
spnp_device_connect(device id)	Establishes a connection between the device and a data-service on the network. This connection may be bi-directional.
spnp_device_release(connection handle)	Releases an existing connection.
spnp_device_data_avail(connection handle)	Callback to notify the device that data from the service it's connected to is ready.
spnp_device_data_recv(connection handle, options, receive buffer)	Receives data from the connected service.
spnp_device_data_send(connection handle, options, transmit buffer)	Sends device data to the connected service.

Applications must respond automatically to the system messages of both PnP protocols. This should be handled by the adaptation layer. For example, a request may come in from an SpW-PnP network as an NMS_READ_NETWORK_ID message. Since this message is specific to SpW-PnP, the adaptation layer will know that it (the adaptation layer) needs to respond with the NMS_READ_NETWORK_ID indication containing the specific SpW-PnP network ID. Similarly, if a SPA network status request message comes in on a SPA network, the adaptation layer must respond with a SPA network status reply message.

3 DATA STORAGE CHALLENGE TASK

Given an abstraction layer, a suitable challenge task was needed to validate the approach. The idea of a Network Attached Storage device was selected as a good challenge task, since storage devices in the form of solid-state recorders are used on many spacecraft. Terrestrial Network Attached Storage devices are complex devices that need to support a richer set of capabilities than that needed for space applications. We have begun to develop concepts for a protocol for such a device. Some of the capabilities an NAS protocol must address include:

- Reporting configuration and status information
- Setting configuration values
- Reading and writing data

The protocol is organized according to requests and responses.

Table 4. NAS Protocol Requests

Request	Data Elements
Information probe	• General configuration information • Bad block information • Error Detection and Correction (EDAC) information • Metrics information
Import Bad Block Map	Bit map of bad blocks
Write Data	• Address • Count (blocks) • Data
Read Data	• Address • Count (blocks)

Table 5. NAS Protocol Responses

Response	Data Elements
Configuration/Status Data	Configuration/Status items/value pairs • Power mode • Block size • Memory size • Bank count
Bad Block Map	Bit map of bad blocks
EDAC Data	EDAC counters
Metrics Data	Memory metrics values
Requested Data	• Count (blocks) • Data values

The information probe/response is shown in Figure 1. This figure shows the request format including an 8-bit field for the requested data. The probed device responds with requested data. In the figure, the format of a response to an information probe (the Configuration/Status Item/Value Response) is shown. Figure 1 also shows the write request protocol. The request consists of an Information Probe that includes the request code. Figure 2 shows the data read request protocol. Several fields are large to allow for device growth. In addition to the large size of the address and data fields, the reserved bits will allow for the extension of the protocol in the future for additional capability. This is indicated by the most significant bit in the request and response protocol words.

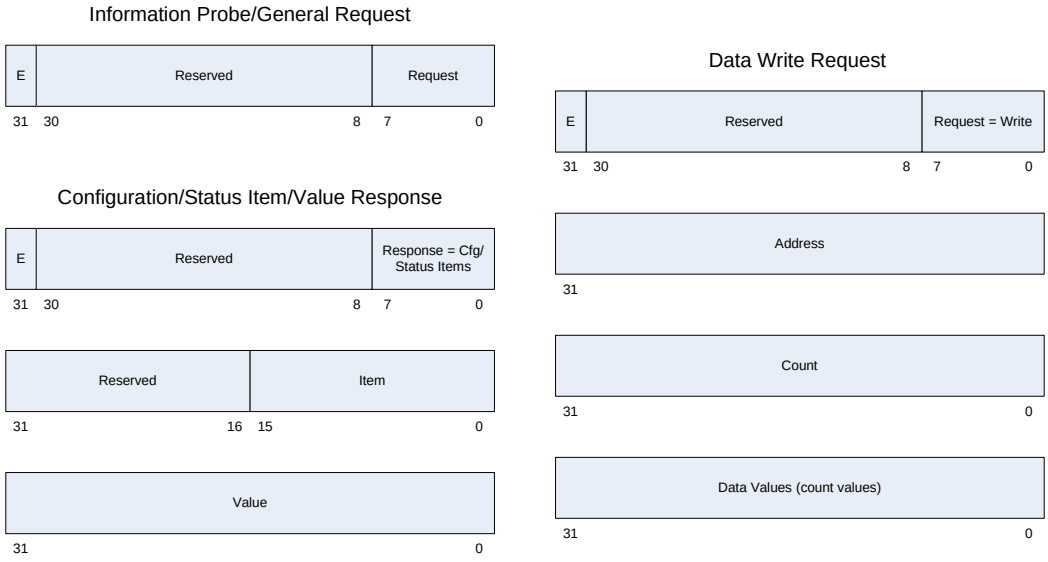


Figure 1. Information Probe/Response and Write Request Protocol

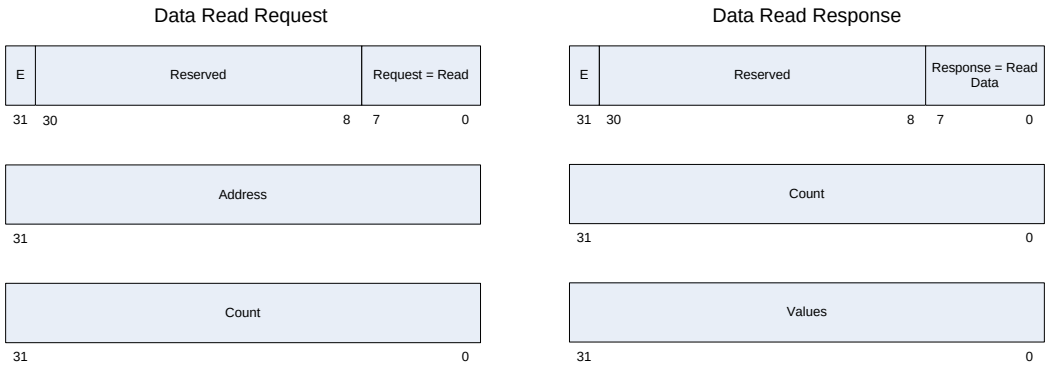


Figure 2. Data Read Request Protocol

4 BUILDING A TEST BED

Figure 3 shows the organization of the test bed. The test bed consists of several workstations connected through a pair of STAR-Dundee SpW routers. Separate workstations are dedicated for the simulated producer, consumer, and NAS functions. All workstations use the Ubuntu variant of the Linux Operating System. Each machine has a custom SpW interface (I/F) board and a corresponding driver. One of the core capabilities of these boards is a Field-Programmable Gate Array (FPGA) implementation of RMAP. An optional system can be incorporated by connecting the SpW routers via a USB. These connections are used to configure and monitor the routers.

Having the producer, consumer, and NAS use the same configuration allowed us to focus on the concerns of the adaptation layer and SpW protocol specifics. Later extensions to this work could include demonstrating the adaptation layer on other platforms, particularly embedded systems to ensure a truly universal interface. We hope our experience with this adaptation layer can contribute to work on the standards.

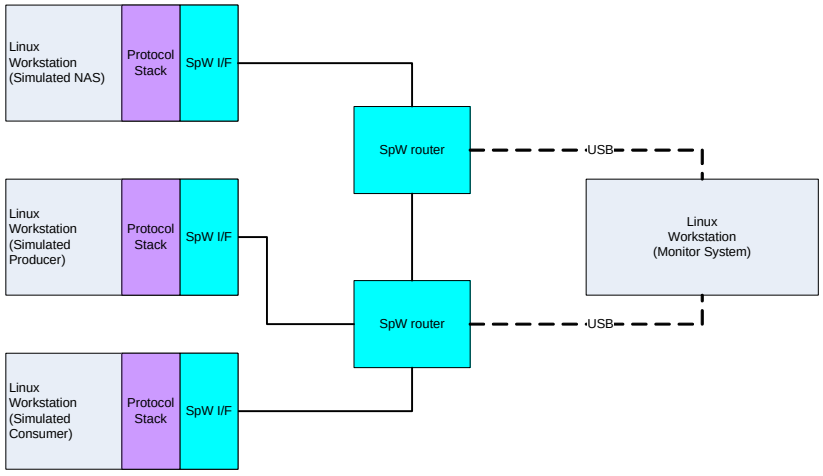


Figure 3. Test Bed

Figure 4 shows the protocol stack for the test bed. The positioning of the adaptation layer can clearly be seen between the specific protocols and the higher levels. Depending on the sophistication of the adaptation layer, it could be built to work integrated with the operating system and file system layers or to bypass them. The benefits of the two approaches are greater portability on the one hand and lower overhead and a potentially simpler interface on the other.

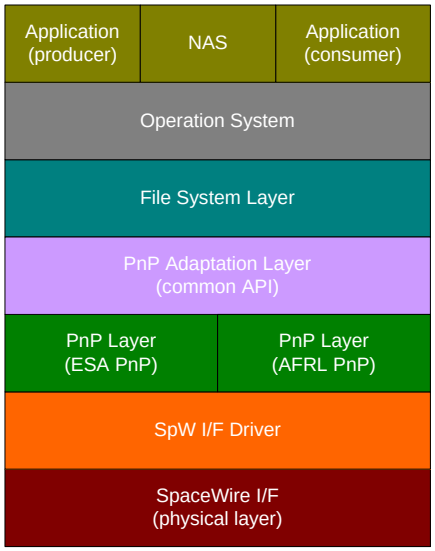


Figure 4. Protocol Layers

The raw communication level has been achieved at this time. Application programs for the producer and consumer have been written to communicate directly to each other over the SpW I/F without any PnP elements present. The I/F operates at nearly 5 Mbits/s without data loss. This data rate is not significantly affected regardless of whether the data passes through one or both of the SpW routers. The producer and consumer applications will be modified to work with the adaptation layer and read/write to the simulated NAS device. A simulated NAS device is being produced as well. We plan to run experiments on the test bed to evaluate the impact of the adaptation layer on efficiency, ease of use, etc.

5 CONCLUSION

Building an adaptation layer should be possible. Some areas of incompatibility may require changes to the standards to reconcile effectively (i.e., to eliminate special protocol unique knowledge in the adaptation layer). Work is progressing on the test bed; and, through the use of reference implementations for the two standards, a practical evaluation of an adaptation layer can be achieved.

6 REFERENCES

1. Space Technology Centre, "SpaceWire-PnP Protocol Definition," University of Dundee, Dundee, Scotland, UK, Draft A, Issue 2.1, September 16, 2009.

2. American Institute of Aeronautics and Astronautics, "Standards Development Guidebook for Space Plug and Play Architecture," March 2011.

3. American Institute of Aeronautics and Astronautics, "Space Plug-and-Play Architecture Standard: Networking," March 2011.

4. Albert Ferrer Florit, Steve Parkes, and Peter Mendham, "SpaceWire Plug-and-Play: A Roadmap," in Proceedings of the 2nd International SpaceWire Conference, Nara, Japan, 2008.

A SOFTWARE ADAPTATION LAYER FOR SUPPORTING MULTIPLE SPACEWIRE PLUG AND PLAY STANDARDIZATIONS

Session: SpaceWire Networks and Protocols

Long Paper

Sue A. Baldor, Paul B. Wood, Allison R. Bertrand, Dan Goes

Southwest Research Institute®, 6220 Culebra Road, San Antonio, Texas 78238

E-mail: sue.baldor@swri.org, paul.wood@swri.org, allison.bertrand@swri.org, dan.goes@swri.org

ABSTRACT

Over the past few years, two standards for SpaceWire enabled Plug-and-Play (PnP) have emerged, one from the Air Force Research Laboratory (AFRL) and the other from the European Space Agency (ESA). This multiplicity hinders the software designer's ability to build reusable software that can work with either PnP platform. Fortunately, these SpaceWire-enabled PnP protocols share enough in common that a unifying interface may be defined. With a suitable abstraction layer, application software will be portable across protocols, reducing the costs of deployment of the different protocols and facilitating the evaluation of protocols against actual application usage patterns.

1 INTRODUCTION

Abstraction is a paradigm that has been used in computer science and software engineering for decades to provide a common interface to differing networking protocols or hardware. Utilizing this interface, programmers may readily port their applications to multiple target platforms, greatly reducing the development costs to support the different target environments.

As standards for Plug-and-Play (PnP)-enabled SpaceWire networks continue to be developed, the concept of abstraction can be applied to these protocols. Two standards presently exist for PnP with SpaceWire: Space Plug-and-Play Architecture (SPA) and SpaceWire-PnP (SpW-PnP). By exploiting the similarities and handling the differences at the lower level, we can create a common interface that assists both software and spacecraft designers in assembling functional networks.

In this investigation, we apply this methodology to define a common Application Programming Interface (API) for supporting SPA—with SpaceWire support—and the SpW-PnP standards.

2 COMPARISON OF TWO PLUG-AND-PLAY STANDARDS

At first glance, one may see little commonality between the core services described in these standards. However, if we look at the network as a whole we can find

similarities in their operations. Though the service decomposition may be different for each, we can see that similar operations are performed on each network type.

2.1 CORE SERVICES

In SpW-PnP, the term service is used more to describe the capabilities offered by the network than to describe a traditional software service. This stems from the standard's philosophy of largely leaving implementation details out. There are considered to be four core services within the SpW-PnP standard: the Device Identification, Network Management, Link Configuration, and Router Configuration services. Additionally, the standard envisions support capability services that offer higher-level capabilities. Currently, two capability services are described: the Remote Memory Access Protocol (RMAP) Data Source and the RMAP Data Sink.

Table 1. SpaceWire-PnP Services

Core Services
Device Identification Service: The Device Identification Service acts as a central repository where one can query information about a device such as its status and provided capability services. The parameters stored for each device include the device's identity, type, status, PnP level of service, and aforementioned capabilities. With the Level 2 PnP network, an additional field is defined in the device status for the owner identity. Device errors may also be reported through this service.
Network Management Service: The Network Management Service permits access to parameters which are required for the discovery of devices and network topology. In a Level 1 network, the single active node also takes the role of the management service. When multiple active nodes are introduced with the Level 2 network, some mechanism must be introduced to arbitrate conflicts between potential device owners. In such a network, each active node has a corresponding Network Manager. Using an additional parameter for the node priority, conflicts in node ownership are resolved by a vote based on this priority; and, in the event of a tie, the port number.
Link Configuration Service: The Link Configuration Service provides a mechanism for the links of a device to be queried and configured. This service will configure the link rates for each SpaceWire link on all connected devices. Additionally, status information including the current link rate and port activity may be queried.
Router Configuration Service: The Router Configuration Service permits the features of each SpaceWire router to be queried and configured. Foremost, the service is responsible for configuring the routing tables for each SpaceWire router under the active node's ownership. Additionally, the routing control parameters of the router are configured, including the watchdog timer, arbitration, and time code settings. The status of all router settings may be queried through this service.
Optional Capability Services
RMAP Data Source: The Data Source service allows a device to disclose data in a standard way on an RMAP interface. Both RMAP targets and initiators may be described and configured through the capability service. A target data source provides an interface for RMAP reads, while an initiator data source provides an interface for RMAP writes. Only one component may use the data source at a time. Optionally, a target data source service may support a queue for pending read operations.
RMAP Data Sink: Data sinks operate in a similar manner to data sources, but in the opposite data direction. Target sinks provide an interface for RMAP writes, while initiator sinks provide an interface for RMAP reads.

The core services of a SPA network, known as the SPA Services Manager (SSM), extend beyond network discovery and configuration. The core services in a SPA network with SPA-SpaceWire (SPA-S) infrastructure provide for topology discovery and device discovery as well as providing mechanisms by which components may locate desired data services and establish a connection to them. Four core services are necessary in the SPA network: the Central Addressing Service (CAS), the SPA Lookup Service, the SPA-X subnet manager – a SPA-S manager in the case of our SpaceWire network, and the SPA Local manager.

Table 2. SPA (with SPA-S Subnet) Core Services

Core Services
Central Addressing Service: The Central Addressing Service is responsible for issuing blocks of addresses to subnet managers. When a SPA system starts up, the subnet managers request a block of addresses from the CAS. The CAS responds with a block of addresses for the requesting subnet manager to use. The CAS must reside on a node that contains a SPA Local Manager.
SPA Lookup Service: The SPA Lookup Service serves as a central repository for information about the components on the network. This information includes the Extensible Transducer Electronic Data Sheet (xTEDS) for a device and its unique ID. Within the xTEDS, a device will describe its configuration and any data services that it may provide. Components may subscribe to particular data services through the Lookup Service. There is a single active instance of the Lookup Service on a SPA network.
SPA Subnet Manager: The complexity of a SPA network is largely hidden within the SPA subnet managers. The subnet manager performs topology discovery for the subnet it is responsible for. The manager detects new components added to the subnetwork and assigns them a unique address. Subnet managers are also responsible for routing packets to the components that are under its influence. On a SpaceWire subnetwork, there would reside at least two SPA managers: a SpaceWire manager and a local SPA manager. The SpaceWire subnetwork manager is responsible for (managing) the connected SpaceWire resources. The SPA Local Manager provides an interface between the SpaceWire manager and the other SPA services.

2.2 NETWORK DISCOVERY AND CONFIGURATION

In a SPA-S network, the SpaceWire subnetwork manager is responsible for network topology discovery. When the network is first powered on, the subnetwork manager registers itself with the CAS and acquires a block of addresses that it may allocate to its subnetwork components [2]. Concurrently, the subnetwork manager iteratively probes its network for connected components. When the manger receives its assigned block of addresses, it will allocate an address for each component and notify each in turn of its address [3].

Components in a SPA network are each responsible for their own xTEDS configuration document. During the time the subnetwork manager is probing components on its network, each component replies with its Universally Unique Identifier (UUID) and its xTEDS UUID. The subnet manager relays this information to the SPA Lookup Service. Following the topology discovery process, the Lookup Service will submit queries for a component's xTEDS to the managing subnet manger. The manager in turn forwards this request to the component. The component responds with a message containing its xTEDS which is forwarded to the Lookup Service [2].

While the details of network topology discovery are largely missing from the SpW-PnP draft, papers by the standard's authors have given methods for how it would be accomplished. Within the SpW-PnP framework, there is the notion of an active node. Active nodes perform network discovery for their section of the network. Passive nodes do not have any knowledge of the network topology and must be assigned an active node owner. Active nodes are also responsible for configuring and managing the passive nodes that they own [1]. The active node discovers the devices connected to it by iteratively querying each active link that it can find. Simulations have indicated that breadth-first searches are most efficient for network discovery [4].

In an SpW-PnP network, device configuration is performed during the device discovery process. However, this does not export the xTEDS documents. It is conceivable that xTEDS support could be accomplished using the data source and sink capability services described in the standard [3].

2.3 Data Transactions

When a component registers with the SPA Lookup Service, any data services it supports are included with its xTEDS document. When a component wishes to subscribe to such a data service, the component contacts the Lookup Service to query for the type of data it wishes to consume. The Lookup Service will provide a route to the component or components providing the target data service. From the list, the component will choose to subscribe to one or more providers. Generally, the component will request the subscription from the Lookup Service who manages subscriptions so that subscribers can be notified when a provider leaves the network. As data becomes available, the provider transmits directly to the subscriber.

Within an SpW-PnP network, information about a device's capability services – data sinks or data sources – is read by the Device Identification Service. A component desiring to receive or publish a particular data product may determine a matching service by inquiring through the active node. The component will establish an RMAP connection with the source/sink using the capability service's protocol. After the connection is established, normal RMAP transactions may be used to read or write the data.

2.4 Common Elements

After considering the services used by each protocol, elements common to the two standards become clear. These similarities between SPA and SpW-PnP were compiled to aid in the process of defining an abstraction layer. Table 3 describes some of the commonalities between the standards as they are deduced from services and pertain to the mechanics of the network.

Table 3. PnP Abstraction API

Similarities	Description
Data sheets	Both standards are committed to an Electronic Data Sheet (EDS) format, with similarities between the two formats.
Capability database	In both, there is a central service in charge of maintaining a lookup of capabilities provided and services requested by registered devices.
Device dictionaries	There exists in both a standardized virtual device information repository (Common Data Dictionary and Dictionary of Terms), and there is evidence that these two dictionaries will align.
Network identification	In both networks, devices connected to the network are given unique network identifiers.
Network changes	Both networks are able to adapt to topology and composition changes as they occur.

3 PLUG-AND-PLAY ABSTRACTION

Considering the elements in common between these two standards, we can begin to define an abstraction layer capable of communicating with networks conforming to either standard.

3.1 PROPOSED ARCHITECTURE AND API

Our proposed approach to the problem of supporting these standards involves a layered software architecture with an API for applications. The API exploits the commonality we have defined between the PnP standards while the complexity and areas of divergence may be hidden in the lower levels of the architecture. Figure 1 displays the software hierarchy for the adaptation architecture.

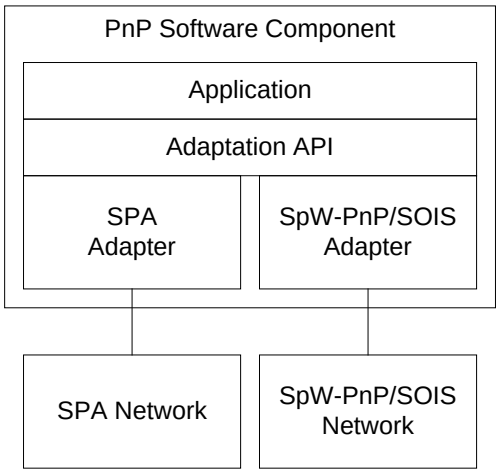


Figure 1. Adaptation Software Hierarchy

Applications invoke the adaptation API directly to communicate with other nodes on the network. The API will invoke functions from either the SPA adapter or the SpW-PnP adapter based on the type of PnP network the software resides on or is targeting. Figure 2 depicts how an application – a Network Attached Storage (NAS) application in this case – might support communications with both a SPA/SPA-S network and an SpW-PnP network.

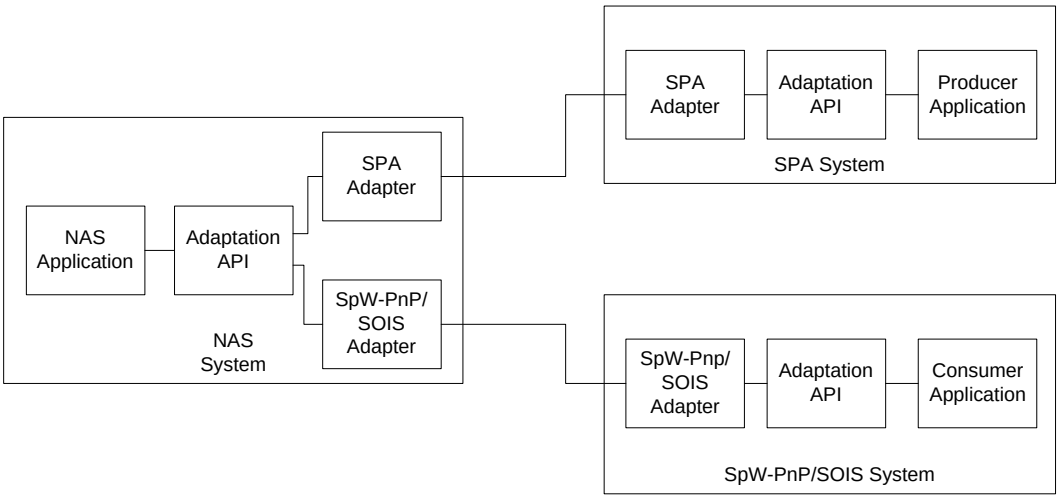


Figure 2. Adaptation Network Example

In the API, we propose supporting the major functions a component must perform when connecting to a Plug-and-Play network. These functions include component discovery, component configuration, data services connection, and data transmission between components. In defining this API, we assume that SpW-PnP will have a mechanism in place that supports exporting an xTEDS for a device. This assumption is necessary to choose a single document format for component configuration. Table 4 lists our proposed functions for the PnP abstraction. We will describe in greater detail the implementations of these functions in subsequent sections.

Table 4. PnP Abstraction API

Function	Description
spnp_init	Configures the SpaceWire abstraction library.
spnp_configure	Configures the component for the network type.
spnp_connect	Establishes a connection between the device and a data service on the network.
spnp_release	Terminates an existing connection.
spnp_data_avail	Callback to notify the component that data from the service it is connected to is ready.
spnp_data_recv	Receives data from the connected service or component.
spnp_data_send	Sends device data to the connected service or component.

3.2 Network Discovery and Device Configuration

The adaptation layer must comply with the messaging requirements for each standard during the phase of device (and on a larger scale, network) discovery. This can happen at two points in the lifetime of the network – during network power-up and when a device is newly connected to the network. To the application, this process is abstracted within the *spnp_configure* function. Each network adapter must respond to the specific protocol of the network it is configured to operate with. In the case of a SpaceWire-enabled SPA network, this requires waiting for and responding to the probe message. Once the subnetwork manager receives its allocation of addresses, the manager will issue an address to the component. The software must wait for this address assignment. Similarly, the SpW-PnP adapter must support the active node's query at which time the address of the device is set.

Following address assignment, the *spnp_configure* function will initiate any additional device configuration. In the case of a SPA network, this involves forwarding its xTEDS to the Lookup Services through the subnetwork manager. In SpW-PnP the mechanism for exporting xTEDS is ambiguous. We assume that either a capability service or higher-level service – such as the Spacecraft Onboard Interface Services (SOIS) Device Virtualization Service – will support xTEDS export.

3.3 DATA TRANSACTIONS

In our API, we group data transactions into four separate operations: connect, send, receive, and release. The *spnp_connect* will establish a connection between components for either data receipt or transmission. The connection process covers two phases: 1) finding an acceptable data service provider and 2) establishing the connection to the provider. For a SPA network, this would involve finding and then negotiating a connection with the Lookup Service. In an SpW-PnP network, this requires querying the Device Identification Service for a compatible data sink or source and then negotiating a connection with the node that provides the data service.

Once the connection has been established, the application may begin to receive or send data. If the application is receiving data, it must wait for the *spnp_data_avail* callback. The adapter software will invoke the callback when the data service has data ready for transmission. Data receipt and transmission are performed within the *spnp_data_recv* and *spnp_data_send* functions. During *spnp_data_send*, the SPA adapter would need to format all data in a SPA data packet prior to sending the SpaceWire packet. Similarly, the SpW-PnP adapter would need to format an RMAP packet for the data write. For *spnp_data_recv*, the respective adapters would strip off packet headers – the SPA data packet header for SPA networks or the RMAP header for SpW-PnP networks – before handing the data to the application.

4 CONCLUSION

With multiple Plug-and-Play standards emerging for SpaceWire, it is important to find common ground between them for the sake of interoperability. Having an abstraction layer which takes advantage of these common elements would be helpful to application and spacecraft designers seeking to incorporate SpaceWire elements into their Plug-and-Play network. We were able to analyze the two standards in search of commonalities, and used our findings to define an API for adapting applications to be used by either SPA or SpaceWire-PnP.

5 REFERENCES

1. Space Technology Centre, " SpaceWire-PnP Protocol Definition," University of Dundee, Dundee, Scotland, UK, Draft A, Issue 2.1, September 16, 2009.

2. American Institute of Aeronautics and Astronautics, " Standards Development Guidebook for Space Plug and Play Architecture," March 2011.

3. American Institute of Aeronautics and Astronautics, "Space Plug-and-Play Architecture Standard: Networking," March 2011.

4. Albert Ferrer Florit, Steve Parkes, and Peter Mendham, "SpaceWire Plug-and-Play: A Roadmap," in Proceedings of the 2nd International SpaceWire Conference, Nara, Japan, 2008.

NEW TECHNIQUE FOR SPACEWIRE NETWORK DISCOVERY

Session: Networks and Protocols

Long Paper

Kody D. Mason, Justin W. Enderle

Sandia National Laboratories, PO Box 5800, Albuquerque, NM 87185-0968

E-mail: kmason@sandia.gov, jwender@sandia.gov

ABSTRACT

Early techniques used to discover the topology of a dynamic SpaceWire network have typically relied on prior knowledge of some protocol implementation. Systematically generated request messages, when responded to by each routing switch or end-node, facilitated discovery. The challenge today, however, is to discover and map network topology without relying on any one protocol implementation - or even any SpaceWire protocol. By exploiting the design of SpaceWire routing switches, discovery is possible on dynamic, heterogeneous SpaceWire networks using the concept and technique of looping messages back to oneself. Exploring the advantages and implications of such a viable technique may lead to a new standard for network discovery.

1 NETWORKS AND NODES

Using the terms and definitions from the European Cooperation for Space Standardization (ECSS) Glossary, and building upon the SpaceWire foundation [1], the notion of a dynamic SpaceWire network is one in which the links between routing switches and nodes can be added or removed in a *Plug-n-Play* like fashion. When links between routing switches are manipulated, the topology of the SpaceWire network changes. When links between nodes and routing switches are manipulated, packet sources and destinations appear and disappear.

This paper will begin by differentiating between *Network Discovery* and *Node Discovery*. The former involves the systematic probing for SpaceWire routing switches, and the latter involves polling switches for links to nodes, and then identifying such.

When probing for routing switches, early network discovery techniques typically relied on each routing switch's configuration port to respond to identification requests to confirm the routing switch's presence. A request packet was typically dispatched to the configuration port, and a response packet provided confirmation of existence.

This same request/response approach was generally used for node identification as well. Dispatching one or more requests to an active link (which might be node or

another routing switch) could produce a response if a node was present and it understood the protocol.

Recent proposals, such as the SpaceWire PnP Protocol Definition Draft [2], put forth basic *Service Definitions* for *device identification*, *network management*, and *link* and *router* configuration. This paper will blur the boundaries between Network Discovery and routing switch configuration. Link configuration (particularly speed) is assumed to be automatic or take place prior to physical link connection.

2 NEW PROBING TECHNIQUE

Per the SpaceWire PnP Draft [2], "SpaceWire does not offer a standard mechanism for detecting the topology of a network." One aim of this paper is to propose such a standard.

The new probing technique involves a shift away from the request/response model. Rather than dispatching a request to some possible physical-path-address on the network, and awaiting a response from a packet receiving/processing/replying entity, a single packet is addressed with a round-trip physical-path-address that will essentially "loop" through a possible routing switch and be returned to the originator with all path-addressing bytes removed along the way out and back.

Perhaps the best way to visualize this technique is to think of the SpaceWire routing switch as a "roundabout" intersection with a vehicle (packet) both entering and exiting the roundabout at the same point.

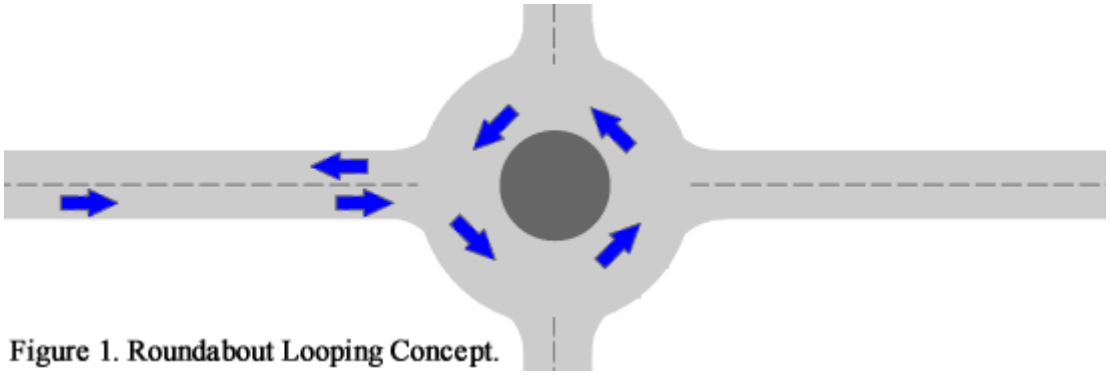


Figure 1. Roundabout Looping Concept.

The significance to the probing entity is that if it receives the recognizable payload portion of a packet back, then that round-trip physical address is valid in most cases.

To more explicitly reiterate this technique, consider a node acting as a probing entity connected to routing switch A's port five. Switch A's port three links to switch B's port two, and switch B's port four links to switch C's port one. Therefore, the physical-path-address from the probe to switch C is "34", and the return path is "125".

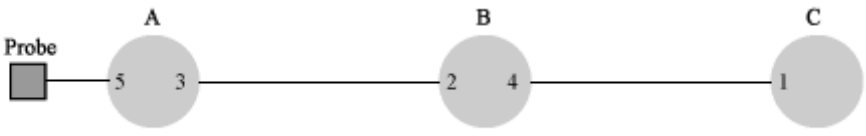


Figure 2. Example of Round-Trip Physical-Path *Addressing*: *"34125"*

By addressing a probe packet, [PACKET], with "34125", then the probe node will receive back [PACKET] after it *loops* through switch C. Notionally, switch C's port one (1) is the "turn-around point" or the "turn-around port."

The Network Discovery process is typically breadth-first. General practice is to begin probing one link (or "hop") from the probing node, then as routing switches are discovered, a new list of potentially viable physical-paths is generated for one hop beyond that. Probing can be stopped when the hop count reaches a point where the new potentially viable list yields no results.

2.1 BREADTH-FIRST PROBING

Recall that SpaceWire physical-path-addressing uses addresses in the range of one to thirty-one (1-31.) A probing entity can discover its own port number on its routing switch with a single-byte physical address preceding its probe packet payload. From Figure 2, the packet containing "5[PACKET]", when written, will cause "[PACKET]" to be read back.

Round-trip physical-path-addresses are always an odd number of bytes. The iteration technique, when generating the potential list of addresses for the next hop count, involves inserting different pairs of port numbers just before the turn-around point of each known round-trip-address at the previous hop.

For example, if the list of known round-trip-addresses for hop number two (hop #2) was simply "325", then the initial potential list for hop number three (hop #3) would be:

a) 3<u>11</u>25
b) 3<u>12</u>25
c) 3<u>13</u>25
d) 3<u>14</u>25
e) 3<u>15</u>25
f) 3<u>16</u>25
g) 3<u>17</u>25
h) 3<u>18</u>25
i) ...
lll) 3<u>88</u>25

where the maximum port to be probed is either thirty-one (31) or an implementation-defined maximum. From the list above, the maximum port to be probed for is eight (8.) However, generating possible round-trip-addresses is subject to certain pitfalls (see section 2.3.)

2.2 BASIC ROUTER IDENTIFICATION REQUIREMENT

Discovering physical-path-addresses that indicate a potentially valid round-trip path through a routing switch is the first step in mapping a network topology. In order to be able to accurately create a topology map, some unique indicator must be available to identify routing switch instances in order to distinguish newly discovered switches from ones previously discovered through other physical paths.

Since the SpaceWire routing switch design has a configuration component, a request for a router ID is one method of routing switch identification. A potential *best practice* for hardware designers is to allow a hardware component to be used to set a unique default ID per router (not unlike the purpose of a "MAC" Address for an Ethernet "PHY".)

Another option for identification involves using the Remote Memory Access Protocol (RMAP) [3] to read an identification number or string from a non-volatile memory location. The SpaceWire PnP Draft [2] proposes something even more advanced.

2.3 POTENTIAL PITFALLS

Probing in the manner described above is subject to several pitfalls. These pitfalls fall into three basic categories: discovery logic, routing switch design, and node robustness.

2.3.1 COINCIDENTAL RETURN PATHS (DISCOVERY LOGIC PITFALL)

As mentioned above, the receipt of a recognizable probe *payload* does not guarantee that the round-trip physical-path-address actually looped at a turn-around point in a routing switch. There is a chance that, by coincidence, two separate return paths coming back from the routing switch are identical except for the turn-around port, itself. In this case, the fact that two probe packets (with identical outbound paths) successfully made their way back to the probing entity is the clue necessary to identify this situation and trigger further analysis. One return path completes the loop-back through the routing switch, but the other return path flows through different links back to the probing entity.

Referencing Figure 3, two probe packets addressed as "12115" and "12415" will both be returned to the probe entity. Likewise, two others addressed as "42115" and "42415" will also. When only the turn-around port is different in the round-trip path-addresses, the coincidental path should be discarded. Determining which one should be discarded requires confirming the identity of the switch one hop prior to the suspected turn-around point. In the case of the "12115" and "12415" pair, confirming that the identity returned by addressing "1" (switch B) matches that returned by "124" (also switch B) is required to know that "12415" is the one to keep, and "12115" is the one to discard.

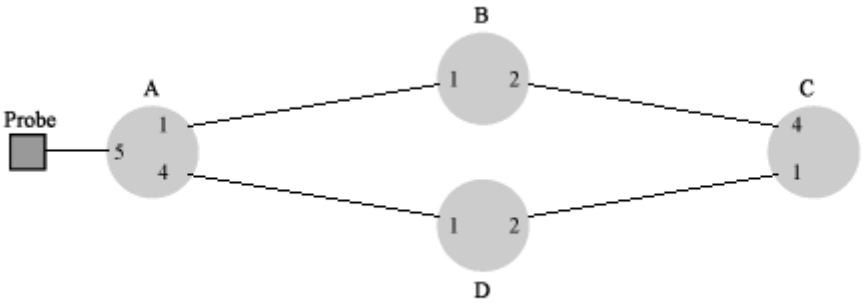


Figure 3. Coincidental Return Paths.

2.3.2 ECHOING (DISCOVERY LOGIC PITFALL)

In the course of generating the list of potentially viable round-trip paths at the next hop count, care must be taken not to "echo" back and forth between two routing

switches. For example, as depicted in Figure 4, if a discovered round-trip path is "34125", then the temptation to probe for "34**14**125" should be avoided.

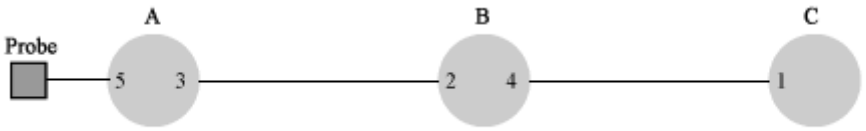


Figure 4. Example of Echoing: "3414125"

2.3.3 NEVER BACKWARDS (DISCOVERY LOGIC PITFALL)

Even more general than the echoing pitfall is the condition when generating the list produces any next hop round-trip path-address where the next outbound port (at the next hop) matches the previous turn-around point. As an example, consider the new potential path of "34**1x**125" (where 'x' is anything.) As long as the bolded '1' matches the '1' in "125", the route will bring the packet backwards (closer to the probe.)

2.3.4 INACTIVE OR NON-EXISTENT PORTS (ROUTING SWITCH DESIGN PITFALL)

As a probing entity transmits its discovery packets across the network, routing switches will invariably receive packets physically addressed to ports that are not active, or do not even exist. Depending on the routing switch design, an attempt to remove the next physical-address-byte and write the remaining packet to such a port could cause a router lockup. One *best practice* for a SpaceWire routing switch design is to always silently drop packets destined for inactive or non-existent ports.

2.3.5 BUFFER LIMITATIONS (ROUTING SWITCH DESIGN PITFALL)

So far, little has been mentioned regarding the contents of the probe packet payload – the bytes that find their way back to the probing entity indicating that a potentially valid round-trip address was discovered. The issue at hand is not so much what the probe packet *payload* contents is, but rather how large it is.

Using the roundabout analogy presented earlier, suppose that a large truck is pulling three large trailers as it attempts to circum-navigate the roundabout. Before the third trailer enters the roundabout from the side street, suppose the front of the truck runs into it. The SpaceWire routing switch design may limit the number of bytes that can be buffered while a packet is retrieved from a port and then written back to it. To minimize the likelihood of such an occurrence, very small *payloads* should be used in the probe packets.

Note: Since the number of bytes which have to loop through the routing switch include both the return-path portion of the address and the *payload*, then the buffer size used in the routing switch design is the key to determining the maximum number of "hops" that can be discovered with this technique.

2.3.6 PACKET PARSING ERRORS (NODE ROBUSTNESS PITFALL)

This new technique for Network Discovery can create a manageable "storm" of probe packets on the SpaceWire network. The *blast intervals* and delays between packet transmissions are easily configurable within the probing entity; however, the effects of all these physical-path-addressed probe packets on nodes could be problematic.

As potentially viable probe packets find their way across links from routing switches to nodes, the nodes may encounter bytes from the physical-path-address or from the probe packet *payload* contents. These bytes may fall where a SpaceWire protocol byte is expected. Nodes have the potential of misinterpreting these packets (if they appear to be a recognized SpaceWire protocol), or in other cases, nodes may fail to disregard these packets (if they appear to be an unknown or unsupported protocol.)

Although on the surface, this new Network Discovery technique appears to introduce the risk of node failures, it actually can have the opposite effect. By requiring this discovery technique to be used during the design and testing of routing switches and nodes, the entire network can be tested for a higher level of reliability and robustness before final implementation.

2.4 COMPLETING NETWORK DISCOVERY

When the probing process is completed, a *results table* will contain all valid round-trip physical-path-addresses and corresponding router identities. Multiple rows may be found for any router identity signifying multiple paths to the router. At this point, a logical addressing scheme can be used to compile route tables. These tables can be generated with any desired regional addressing supplement. Note that section 2.6 contains a method for consistent logical address assignment based on the concept of affinity.

Routing switches may be partially configured now. Specifically, switch-to-switch logical address routes may be inserted into all *route tables*. Node Discovery is now possible using either physical-path or (routing switch level) logical addressing combined with (node level) physical addressing.

Finally, the *results table* can be used to dynamically visualize the network. Depicted are the probing entity (blue), and routing switches from two separate vendors (red, and green.) Presumably, the identification of routing switches may have involved more than one technique (per section 2.2.)

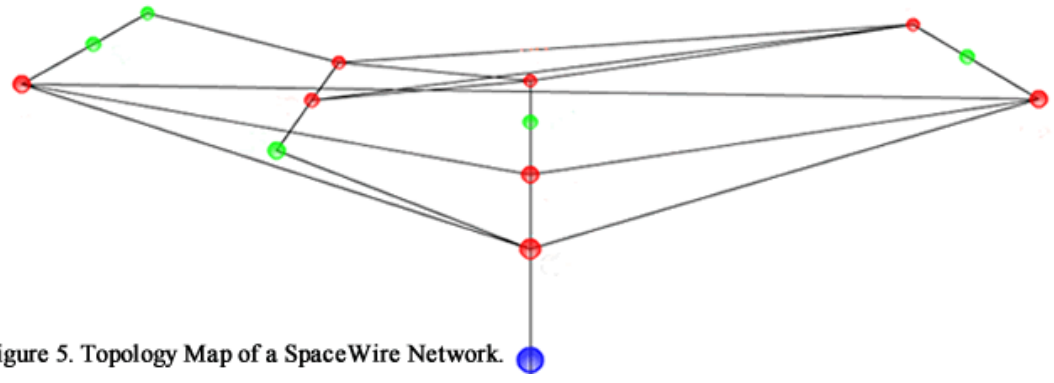


Figure 5. Topology Map of a SpaceWire Network.

2.5 POLLING FOR NODES

The process of *Node Discovery* involves the systematic polling of nodes for management information. Node Discovery requires that each node receive and process a request packet, then respond.

As of the writing of this paper, the authors are unaware of an adopted standard in the SpaceWire community to address Node Discovery in a multi-vendor, heterogeneous SpaceWire network.

A proposal to adopt an Internet standard, such as the Simple Network Management Protocol (SNMP), could remedy the situation. Specifically, adoption of SNMPv1 [4] as a SpaceWire-supported protocol with a minimal required implementation of the "System" group from RFC-1213 [5] could enable standardized Node Discovery as well as provide a single technique for routing switch and end-node identification. Such adoption may be consistent with one of the aims of the SpaceWire PnP Draft [2] to "leverage existing technologies as much as possible."

2.6 LOGICAL ADDRESS ASSIGNMENTS – AFFINITY

The notion of *affinity* (of a SpaceWire logical address to a particular switch or node) can be borrowed from the *plug-n-play* behaviour of many computers and personal computing devices. Consider how portable storage devices or serial communications devices are often managed when they are attached to a computer:

For example, upon the first attachment of a USB modem to a personal computer (PC), the USB *plug-n-play* device manager will determine the device type and serial number of the modem. If this specific device is not listed within a registry, then it is assigned the next unused "COM" port and added to the registry. In the future, each time the device is subsequently attached, its registry information is used to re-assign the same "COM" port as before, so the device has an affinity to a particular port number. The rationale for this behaviour is that humans will naturally remember which COM port is which over time, and humans will want consistency in assignments.

Another example of affinity is the manner in which Dynamic Host Control Program (DHCP) servers typically assign Internet Protocol (IP) addresses. When a request for an IP address is made, most DHCP servers will attempt to re-assign one that was last used by the requesting MAC if that IP address is not already in use.

This same notion applies to dynamic *plug-n-play* SpaceWire networks. When a new routing switch or node is discovered, the probe entity can assign the next unused logical address for the region. If the probe has a means to persistently save the identity of the discovered switch or node, along with its newly assigned logical address, then subsequent re-discoveries of the same entity can result in consistent logical address re-assignment.

3 SUMMARY

The techniques described above for *Network Discovery* and *Node Discovery* are indeed different. While the request/response type of discovery technique is required for node discovery, the benefits of using round-trip physical-path-addressed SpaceWire packets to discover routing switches are many. Chief among them is not relying on packet processing entities to support (understand) one or more SpaceWire protocols. Essentially, if a routing switch has active links on the network, and it is functioning with a unique identity, then it can be discovered and mapped through its switch-to-switch links.

4 REFERENCES

1 ECSS-E-ST-50-12C, European Cooperation for Space Standardization, "SpaceWire – Links, nodes, routers and networks", 31 July 2008, 15-22.

2 SpW-PnP-PD, Space Technology Centre, School of Computing, University of Dundee, "SpaceWire PnP Protocol Definition, Draft A Issue 21" 16 September 2009, 16-17, 41.

3 ECSS-E-ST-50-52C, European Cooperation for Space Standardization, "SpaceWire – Remote memory access protocol", 5 February 2010, 13-15.

4 J. Case et al., RFC-1157, "A Simple Network Management Protocol (SNMP)", May 1990, 2-33.

5 K. McCloghrie et al., RFC-1213, "Management Information Base for Network Management of TCP/IP-based internets: MIB-II", March 1991, 10-13.

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

TOWARDS SPACEWIRE PLUG-AND-PLAY ECSS STANDARD

Session: Networks and Protocols

Long Paper

David Jameux

Company European Space Agency / European Space Technology Centre,

Keplerlaan 1, 2201 AZ Noordwijk ZH (The Netherlands)

E-mail: david.jameux@esa.int

ABSTRACT

The SpaceWire working group on Plug-And-Play have drafted a protocol specification to allow network discovery and the detection of configuration changes in the network. The objective of using these techniques is to support rapid integration of future spacecraft that are using SpaceWire networks. In this paper, we recall the need for formalising and breadboarding the current draft standard for SpaceWire Plug-And-Play as well as the features that a demonstrator for such breadboard should exhibit. We also explain how this issue should be tackled through the ESA/TRP activity "Network Discovery Protocols". We discuss the capabilities of SpaceWire Plug-And-Play on an example of complex on-board data systems architecture and we describe the steps still to be taken in order to prepare for the standardisation of the SpaceWire Plug-And-Play protocol by the appropriate ECSS Working Group.

1 BACKGROUND

Through several years of standardisation and technology development activities, the European Space Agency (ESA) have prepared the SpaceWire technology that allows embarking high speed data networks on board spacecraft. This new technology has become widely adopted not only by ESA missions but also by other agencies and industries.

The SpaceWire standard [1] defines the aspects of a highly flexible and capable communication system which roughly correspond to the physical and data-link layers of the ISO Open Systems Interconnection (OSI) basic reference model. The standard also defines a number of features which fit into the network layer of this model. Whilst following the standard does ensure a certain degree of interoperability, which is further extended by the protocol identification mechanism [2] and the SpaceWire standard protocol suite ([3], [4]), SpaceWire networks must still be designed, constructed, and configured carefully for a given application, usually requiring customised software and/or hardware.

The lack of standardisation for simple configuration tasks required on almost all SpaceWire networks limits the level of interoperability which may exist between devices and software, and the extent to which both hardware and software can be re-used between different applications.

The SpaceWire working group on Plug-And-Play (PnP), consisting in European and US experts from industry as well space agencies, have drafted a protocol specification to allow network discovery and the detection of configuration changes in the network [7]. The objective of using these techniques is to support rapid integration of future spacecraft subsystems that are using SpaceWire networks.

In its latest version, the draft SpaceWire PnP protocol is based on the syntax and synchronisation rules of the SpaceWire Remote Memory Access Protocol (RMAP) [3]. This draft protocol specification is quite advanced but, in view of its standardisation in the frame of the European Cooperation for Space Standardisation (ECSS), it must be completed and validated through breadboarding, verification, and demonstration.

This is currently being done in the frame of the "Network Discovery Protocols" Research & Development (R&D) contract kicked off in October 2011. This contract is funded under the ESA Technology Research Programme (TRP).

The overall goal of this activity is to further define design, breadboard, test, and validate a SpaceWire Plug-And-Play protocol, and produce the related documentation. To this purpose, such a protocol will first be designed and described in detail. Then, a SpaceWire Plug-And-Play test bed will be built up mainly from existing SpaceWire equipment. The necessary functions to support the PnP protocol will be implemented in firmware and/or in software. Functional tests and overall demonstration will be performed, assessing the usefulness and deriving recommendations for improvements.

2 OBJECTIVES

SpaceWire does not offer a standard mechanism for detecting the topology of a network, or what devices are attached to it. Nor does it offer a standard mechanism for configuring the various aspects of a SpaceWire network, such as links and switches. SpaceWire also lacks standard features to assist detection or configuration beyond the network, in the service domain. It is the aim of the SpaceWire-PnP protocol to add these features, within the scope of what is practical.

The first objective of this activity is to design a SpaceWire Plug-And-Play protocol that fulfils all these needs and to describe it in a form as close as possible to current ECSS writing rules in order to prepare for later standardisation of this protocol at European level.

2.1 BASIC PRINCIPLES OF PLUG-AND-PLAY

The aim of SpaceWire-PnP (Plug-And-Play) is to provide standardised, interoperable mechanisms for performing key functions associated with SpaceWire networks. The term 'Plug-And-Play' originates from the commercial electronics market where a range of techniques were developed to improve the user experience of device integration. From the perspective of a space user, application of the term 'Plug-And-Play' indicates that it should be possible to interface two or more arbitrary devices without the need for configuration. Plug-And-Play generally involves two key aspects:

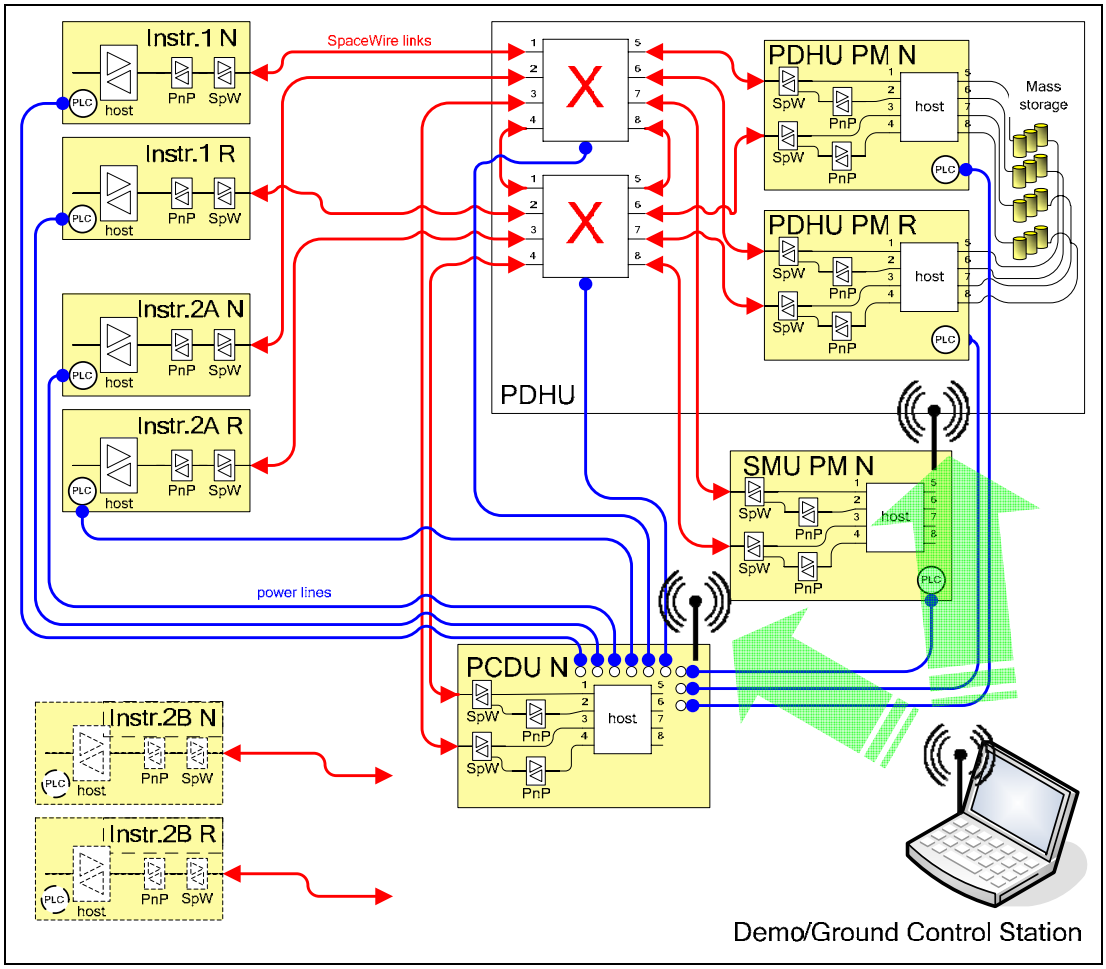


Figure 1 - Example of configuration to demonstrate SpW PnP features

1. Automatic discovery and configuration of hardware and software systems in response to changes in physical interfacing or availability, including whilst the system is running ('hot plugging'); in other words, the capability to detect any connection or disconnection of Plug-And-Play enabled devices.

2. Detection, registration, and configuration of the services that a newly connected Plug-And-Play enabled device provides; as well as detection and de-registration of the services that a newly disconnected Plug-And-Play enabled device was providing.

2.2 Full Compatibility with the Current SpaceWire Standard Suite

The overall goal of the SpaceWire-PnP standard is interoperability at the Network Level as defined in [1]. As such, SpaceWire-PnP should provide services to discover, identify and configure the features of a SpaceWire network, as covered by the next revision of the SpaceWire standard [5], plus a few more corresponding to only the most common use cases.

SpaceWire-PnP should not require devices to support more of the SpaceWire standard than is required to achieve their objectives: if something is optional in the SpaceWire standard, SpaceWire-PnP should not require that it be implemented.

2.3 VALIDATION

The second objective of this activity is to implement and test the SpaceWire network discovery and configuration capabilities of the Plug-And-Play protocol and techniques over a real SpaceWire network. This will be done through testing of each protocol feature whenever possible, and through demonstration of the overall capabilities of the protocol. From the testing and demonstration, recommendations for improvements will be derived which will support the process of standardisation of the SpaceWire Plug-And-Play protocol.

For the validation of these new features at breadboard level, the test setup should be mainly based on existing SpaceWire equipment modified and upgraded with the Plug-And-Play capabilities. Figure 1 shows an example of configuration to demonstrate SpW PnP features.

3 EXAMPLE

Figure 2 shows a representative architecture for on-board data systems as well as space-to-ground telecommunication.

3.1 LINEAR NETWORKS

We consider now Figure 2 in which the SpaceWire link in dotted lines between the two central switches (S2 and S4) is not connected, because the rate of the data potentially flowing between these two switches is, in the worst case, lower than the maximum SpaceWire data rate allowed for the given on-board data systems architecture. Assuming that the network is discovered from each of the Payload Data Handling Units (PDHU) and according to the algorithm baselined for the SpW PnP protocol ([7], [12]), the resulting networks explored, before node merging phase, is shown in Figure 3.

The first conclusion that we can draw from this network exploration is that networks #4 and #7 are identical, as well as networks #3 and #8. The second conclusion is that, although the on-board data systems network shown in Figure 2 seems very complex, there is actually no loop in the explored networks. The second phase of the network discovery algorithm (merging nodes or networks) is therefore not required in this case. This on-board data system is in fact made of six linear networks. The SpaceWire Plug-And-Play service can then proceed with the discovery and configuration of the services provides by each of the terminal nodes in each network.

The configuration of logical addresses for these six networks is straightforward – since they contain no loop – and can be fully handled by the SpaceWire Plug-And-Play service. The six networks being independent, the same logical address may be assigned to the four SpaceWire interfaces of the PDHU, which might reduce the complexity of the applications running on other terminal nodes (instruments and Spacecraft Management Unit – SMU). The same applies to the two SpaceWire interfaces of the SMU. For the same purpose, it is also possible to assign the same logical address to the Nominal and Redundant SpaceWire interfaces of each instrument, provided that the switching tables in each switch is carefully designed.

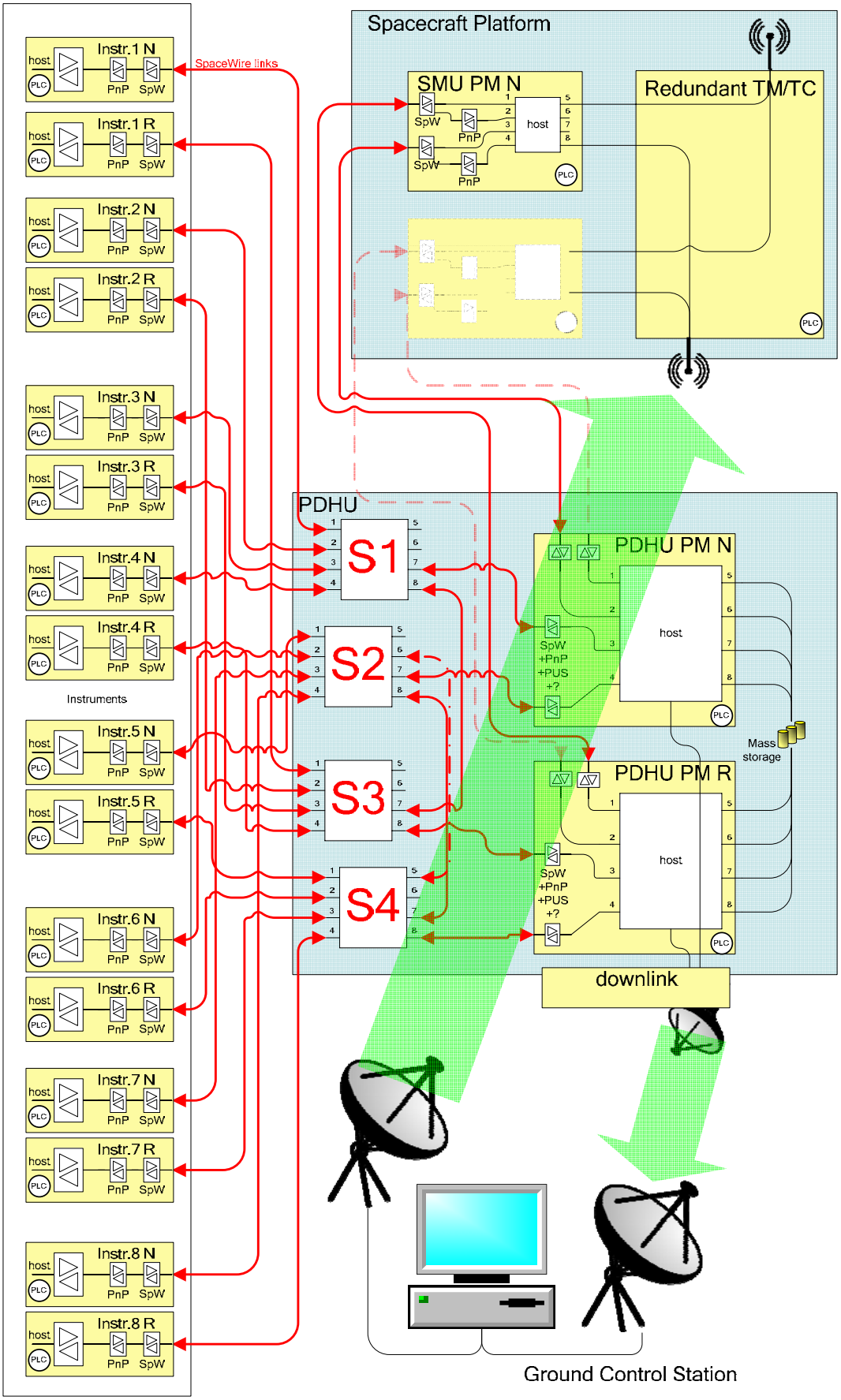


Figure 2 - Example of representative data systems architecture

With this scheme, it is even possible to allow Group Adaptive Routing (GAR, as defined in [1]) between the two switches to enable SpaceWire-level automatic Failure Detection, Isolation, and Recovery (FDIR).

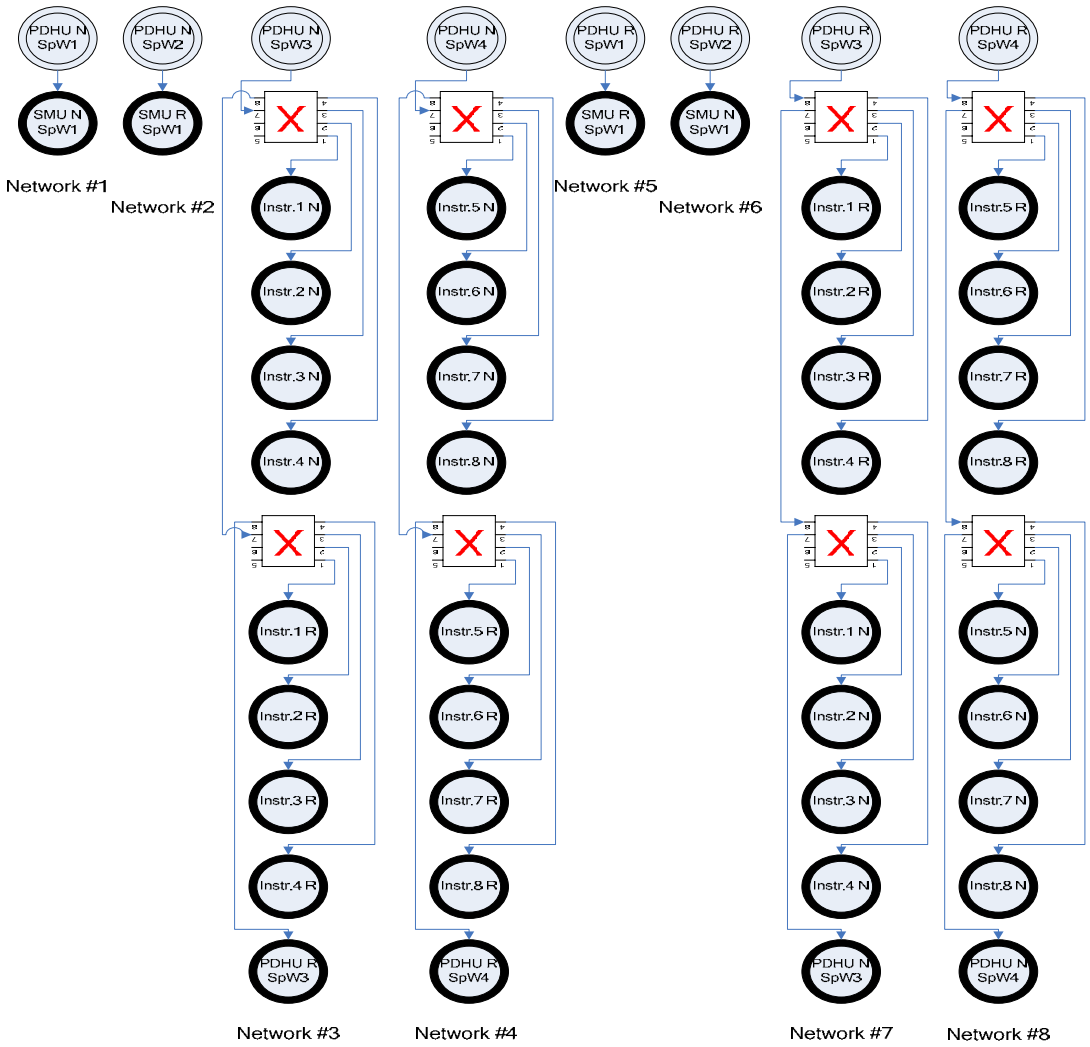


Figure 3 - Exploration of Linear SpaceWire Networks

3.2 Non-Linear Networks with Simple Loops

We now assume that a second link is connecting the two central switches (the SpaceWire link in dotted lines between switches S2 and S4 in Figure 2 is now connected) in order to accommodate more data rate between these two switches. As shown in Figure 4, a simple loop is introduced in network #4/7. Since this loop involves only two switches, the node merging phase of the network discovery algorithm is straightforward and the assignment of logical addresses can follow the same pattern as described for the previous case (linear network).

3.3 Non-Linear Networks with Complex Loops

If we want to increase even more the possibility of using redundant paths in case of failure, we can connect switches S1 and S2 together via an additional SpaceWire link,

as well as switches S3 and S4. This introduces a complex loop and increases significantly the number of possible paths from one terminal node to another, e.g. from an instrument to the PDHU.

An illustration is provided in Figure 5. A reasonable network discovery algorithm would now consider this physical network as only one logical network, assigning different logical addresses to each of the terminal nodes, and therefore to different SpaceWire interfaces of the same spacecraft unit (e.g. the PDHU), although this might not be the preferred option for the system spacecraft designer.

This advocates for the SpaceWire Plug-And-Play services to be complemented with some tools allowing Computer Aided Design (CAD) of SpaceWire networks.

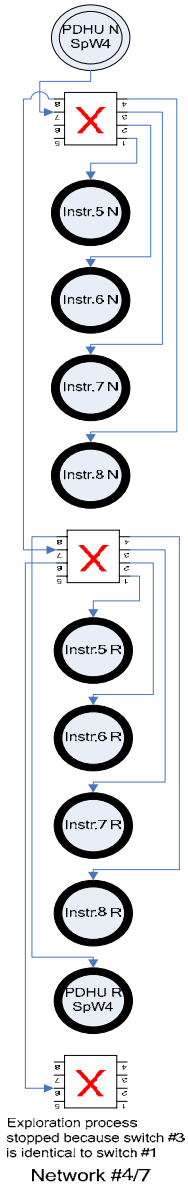


Figure 4 - Exploration of Non-Linear SpaceWire Networks with Simple Loops

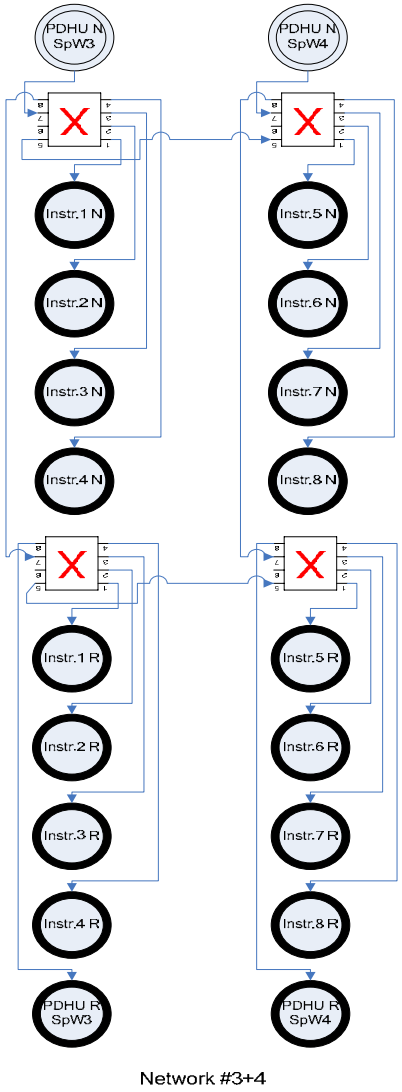


Figure 5 - Exploration of Non-Linear SpaceWire Networks with Complex Loops

4 CONCLUSION

Once designed, formally verified, breadboarded, and validated, the Plug-And-Play services and protocol presented in this paper will be handed over to the SpaceWire Working Group for endorsement. They will then be subject to formal standardisation by the European Cooperation for Space Standardisation (ECSS).

This paper also showed the need for the SpaceWire Plug-And-Play services to be complemented with some tools allowing Computer Aided Design (CAD) of SpaceWire networks. Such tool should be specified by the SpaceWire Working Group. Its breadboarding and validation could possibly be supported by ESA R&D activities.

5 REFERENCES

1. ECSS-E-ST-50-12C, “SpaceWire – Links, nodes, routers”, 31 July 2008

2. ECSS-E-ST-50-51C, “SpaceWire protocol identification”, 5 February 2010

3. ECSS-E-ST-50-52C, “SpaceWire - Remote memory access protocol”, 5 February 2010

4. ECSS-E-ST-50-53C, “SpaceWire - CCSDS packet transfer protocol”, 5 February 2010

5. David Jameux, “SpaceWire Evolutions”, International SpaceWire Conference, San Antonio, November 2011

6. Martin Süß, “SpaceWire Standard Evolution”, International SpaceWire Conference, Nara, November 2008

7. Peter Mendham, SpaceWire-PnP Protocol Definition, Draft A Issue 2.1, 16th September 2009

8. Peter Mendham, Albert Ferrer Florit, Steve Parkes, “SpaceWire Plug-And-Play: A Roadmap”, International SpaceWire Conference, Nara, November 2008,

9. “SpaceWire Plug-and-Play: Fault-Tolerant Network Management for Arbitrary Network Topologies”, International SpaceWire Conference, September 2007, Albert Ferrer Florit, Martin Süss

10. ESA & NASA, “ESA and NASA requirements on SpaceWire PnP”, March 2007,

11. “SpaceWire Plug-and-Play: An Early Implementation and Lessons Learned”, AIAA Infotech@Aerospace 2007 Conference and Exhibit, May 2007, Barry M Cook and C Paul H Walker, 4Links LtdBarry Cook, Paul Walker, “PnP aspects, 4Links contribution”, 8th SpaceWire Working Group, ESTEC, January 2007

12. Albert Ferrer Florit, “PnP aspects, ESA contribution”, 8th SpaceWire Working Group, ESTEC, January 2007

13. “PnP aspects, 4Links contribution”, 8th SpaceWire Working Group, January 2007, Barry Cook, Paul Walker, 4Links Ltd.

PERFORMANCE OF SPACEWIRE PLUG-AND-PLAY PROTOCOLS

Session: SpaceWire Networks and Protocols

Short Paper

Robert A. Klar, Dan Goes, Paul B. Wood, and Sue A. Baldor

Southwest Research Institute®, 6220 Culebra Road, San Antonio, Texas 78238

E-mail: robert.klar@swri.org, dan.goes@swri.org, paul.wood@swri.org, and sue.baldor@swri.org

ABSTRACT

Historically, the integration of spacecraft systems has been an expensive proposition because it requires much dedicated time. Plug-and-Play (PnP) describes a mechanism by which devices can be discovered and configured automatically to be ready for use soon after they are inserted into a system. Although PnP is already ubiquitous in terrestrial computing, it has not yet become well established in spacecraft systems. Application of PnP to spacecraft systems provides much promise for reducing integration efforts.

Since first standardized, SpaceWire has gained widespread popularity for use in spacecraft systems because of its simple circuitry, low power consumption, and high link speeds. In 2007, a working group developed an initial proposal for adding PnP capabilities to SpaceWire. Based on this work, two different proposals emerged and are now under consideration for standardization. The first, "Space Plug-and-Play Avionics – SpaceWire" (SPA-S) was submitted to the American Institute of Aeronautics and Astronautics (AIAA). The second, "SpaceWire-PnP Protocol Definition," was submitted to the European Cooperation for Space Standardization (ECSS). In this paper, we characterize the expected performance of these protocols for network discovery and identify some factors that could influence performance.

1 INTRODUCTION

Historically, spacecraft integration has been both a time-consuming and expensive proposition. A key challenge has been to quickly establish communication pathways between a myriad of spacecraft components in order to establish proper data flow. A part of the difficulty lies in the fact that many spacecraft are purposed for a particular mission and consequently have unique combinations of sensors, actuators, and processors.

Recent years have seen a significant push for a reduction in cost and duration of spacecraft integration efforts. The U.S. Department of Defense has funded a series of initiatives for Operationally Responsive Space (ORS) aimed at decreasing the cost of creating space assets and increasing the speed of deployment. In April 2007, a report was submitted to the Congressional Armed Services Committee which broadly defined ORS as "assured space power focused on timely satisfaction of Joint Force Commanders' needs" [1]. The report breaks down responsiveness into tiers, with the

goal for delivery of capabilities requiring existing technologies on the order of days-to-weeks. In addition to the U.S., Europe has also expressed interest in improving the responsiveness of the space enterprise [2]. An important element of improving responsiveness is the development of better technologies.

Plug-and-Play (PnP) is one technology which offers some promise for reducing integration effort. The term PnP is often used to describe a mechanism by which devices can be discovered and configured automatically soon after they are inserted into a system. It is already ubiquitous in terrestrial computing, and efforts are well underway to apply it effectively to spacecraft systems.

In 2007, a small working group developed an initial proposal for adding PnP capabilities to SpaceWire. From this initial effort, two proposed standard protocols emerged: Space Plug-and-Play Architecture - SpaceWire (SPA-S) and SpaceWire-PnP. Each proposed standard has a slightly different set of services and benefits.

In this paper, we highlight some of the dissimilarities between the proposed standards with emphasis on network discovery and device configuration. In addition to describing some of the protocol features, we provide some analysis of the expected performance of each.

2 COMPARISON OF PLUG-AND-PLAY PROTOCOLS

SPA-S and SpaceWire-PnP provide slightly different approaches to accomplishing network discovery and device configuration on a SpaceWire network.

2.1 SPA-S

Space Plug-and-Play Architecture (SPA) is a collection of standards to facilitate rapid development, integration, and testing of spacecraft. SPA was introduced by the Air Force Research Laboratory (AFRL) and later investigated by collaboration with many other government and industry partners [3]. A SPA reference implementation was implemented in software by a group at the Utah State University Space Dynamics Lab. SPA allows a network of sensors, actuators, and processors to self-organize regardless of the topology and composition of the network. SPA-S provides a subnetwork specification for SPA with SpaceWire as the physical layer.

With the SPA, network discovery is driven by network managers that live at the border of two adjacent subnetworks. For instance, a network manager might bridge a SpaceWire subnetwork (SPA-S) and a local subnetwork (SPA-L). Other core SPA services are attached to these local subnetworks.

Because a SpaceWire subnetwork can be connected to multiple subnetworks, several network managers can coexist on the same subnetwork. Every network manager performs network discovery for itself, determining a path to each element of the subnetwork to which it is connected. Network managers do not take ownership of nodes. Instead, they simply learn the location of each node and pass down addressing and identification from the core SPA services to them.

2.2 SPACEWIRE-PNP

Another proposed standard, SpaceWire-PnP, was developed by the University of Dundee and submitted for standardization to the European Cooperation for Space Standardization (ECSS) [4]. A prototype implementation is currently under development by SciSys and will be used to evaluate the protocol. SpaceWire-PnP includes the following services: device identification, network management, link configuration, and router configuration.

The device identification and network management services provide the support needed by SpaceWire-PnP for network discovery. At the heart of the network management service is the concept of active nodes. When active nodes come online, they discover the nodes on the network by doing a breadth-first search. Active nodes gain "ownership" of passive nodes as they are discovered. Complex networks may have more than one active node.

The SpaceWire-PnP provides two support levels: Level-1) Managed Networks and Level-2) Open Networks. In a Managed Network, network designers ensure that there is no competition between active nodes for ownership of passive nodes. In an Open Network, multiple active nodes vie for ownership of passive nodes; a resolution algorithm is used to eliminate conflicts.

3 PERFORMANCE

Network discovery for both SPA-S and SpaceWire-PnP depend on a breadth-first search algorithm. Each network manager or active node must search the entire subnetwork. Thus, expected performance is $O(N+L)$, where N is the number of nodes on the network and L is the number of links.

For both protocols, specific timing requirements have not been levied on devices. This makes comparison of timing between the protocols difficult without evaluating particular implementations. Experimental research is needed to realistically evaluate performance. Southwest Research Institute (SwRI®) is currently conducting ongoing experimental research to evaluate implementations of these protocols.

Performance will be influenced by several implementation factors:

- Device Protocol Support.

  The message format for SpaceWire-PnP is based on the Remote Memory Access Protocol (RMAP). Since many devices today support a hardware core implementation of RMAP, these could be adapted to support SpaceWire-PnP. Since the protocol uses command-response messaging, hardware support would improve speed.

  To comply with SPA-S, an end node must only keep a routing path to a Subnet Manager (SM-s). Nevertheless, since routing messages through the SM-s can overload it, it is desirable for end nodes to cache routes to other nodes that they communicate with often.

- Network Topology.

  A larger network will take longer to map than a smaller one. Timing delays for an Open Network will be less controlled than a Managed Network.

4 CONCLUSION

Protocols for adding plug-and-play capability to SpaceWire have started to mature. As we move forward to adopt these implementations for use on missions, we must keep a cautious eye on performance. Performance will likely be influenced much by the support included within SpaceWire devices for these emerging protocols.

5 REFERENCES

1. Sega, R. M., and Cartwright, J. E., "Plan for Operationally Responsive Space," Department of Defense, April 17, 2007.

2. Remuss, Nina-Louisa, "Responsive Space for Europe," European Space Policy Institute Report 22, February 2010.

3. "Space Plug-and-Play Architecture Standard – SpaceWire Subnet Adaptation," American Institute of Aeronautics and Astronautics, 2010, *DRAFT*.

4. Mendham, P., Florit, A. F., and Parkes, S., "Spacewire-PnP Protocol Definition," Space Technology Centre, University of Dundee, September 16, 2009, *DRAFT*.

Networks and Protocols 2

CCSDS TIME DISTRIBUTION OVER SPACEWIRE

SpaceWire Networks and Protocols

Sandi Habinc, Marko Isomäki, Daniel Hellström
Aeroflex Gaisler, Kungsgatan 12, SE-411 19 Göteborg, Sweden
E-mail: sandi@gaisler.com, marko@gaisler.com, daniel@gaisler.com

ABSTRACT

Aeroflex Gaisler has developed in collaboration with the European Space Agency (ESA) an initial protocol for the transmission and synchronization of CCSDS Unsegmented Code (CUC) time in SpaceWire networks. The working name of the protocol is "SpaceWire - CCSDS Unsegmented Code Transfer Protocol" (CUCTP).

Aeroflex Gaisler has developed under indirect funding from the European Space Agency (ESA) a new IP core that implements the SpaceWire - CCSDS Unsegmented Code Transfer Protocol named SPWCUC, providing automatic SpaceWire Time-Code transmission and reception, and automatic CUCTP packet reception. It also provides support for CUCTP packet transmission.

The CUCTP protocol and its implementation is a first iteration to solve some of the time distribution problems that exist in SpaceWire networks. Additional work is to be performed both on the specification side as well as on the implementation side before a standard protocol can be established. This paper provides the background to the work and it discusses the current draft solution, with an outlook on what needs to done in the future.

1. BACKGROUND

Time synchronization in spacecraft is becoming increasingly important. For example instruments & navigational on-board resources can now be combined for establishing scientific observations and therefore need to be well synchronized in time.

Traditionally time synchronization has been done via dedicated signals or via deterministic on-board buses (e.g. MIL-STD-1553 or OBDH). With the advent of SpaceWire point-to-point links and router switches being used for critical control functions, the need for accurate time synchronization via this network has arisen.

The SpaceWire protocol provides rudimentary time-code transmission, but lacks support for automatic time message distribution and time synchronization. It has no means for handling latency (delays) and jitter caused by routing or drift caused by unstable oscillators.

2. TIME IN SPACEWIRE NETWORKS – A PROBLEM DEFINITION

The SpaceWire (SpW) standard ECSS-E-ST-50-12C is currently being proposed to be used as well for critical real-time control applications. A missing element is a coherent and accurate means of time message distribution and time synchronization.

Ongoing work is focused on these two aspects, with the direct benefit of being useful for these critical real-time control applications as well as for any type of mission requiring highly accurate time distribution over a SpW network. The aim s to address some of the main time distribution issues that are common to many types of networks or buses and to develop a solution specifically for the SpW network that allows controlling time distribution latency (delay), jitter and drift as defined below.

The standard specifies a Time-Code character that is propagated throughout a SpW network and is used for time distribution. The Time-Code character has the highest transmission priority and is broadcast through the complete network via one or many router switches or directly via point-to-point links.

The transmission time of a Time-Code character is at least 14 transmission clock periods (ESC + data character), which is multiplied by the number of links that the Time-Code has to traverse from the time source to the destination. This introduces a minimum delay or latency of 7 us for each link at 2 Mbit/s transfer rate. This defines the time distribution delay.

Although the time Time-Code character has priority over other characters defined in the protocol, its transmission on a link can be seen as asynchronous with respect to the on-going transmitted character stream. Thus, the actual time of the Time-Code transmission depends on whatever is being transmitted at the SpW link at the time of the Time-Code transmission request. The delay between this Time-Code request and the actual transmission is equal to the time left to complete the transmission of the on-going character. The difference between the shortest and longest time left depends on the character being sent and is in the range of 10 transmission clock periods. Thus for a 2 Mbit/s transfer rate the achievable accuracy for a point-to-point link is in the range of 5 us. The problem is compounded when multiple router switches have to be passed in a network, each router switch contributing to the uncertainty. This defines the jitter.

The SpW network is asynchronous, i.e. there is no common clock signal being distributed for the communication, with each node being responsible for its own clock. This means that the local clocks run independently and can exhibit different stability. The variation between the different clocks (be that oscillators or crystals) will lead to drift and mismatches over time. For example, a SpW node might be clocked by an oscillator that not only a slight frequency offset and may experience also frequency variations over time. This will lead to an increasing difference between the times kept by two nodes in a system. This describes the drift.

3. CURRENT PROTOCOL FORMAT

The current SpaceWire - CCSDS Unsegmented Code Transfer Protocol (CUCTP) packet conforms to the ECSS SpaceWire standard. It contains the CCSDS Unsegmented Code (CUC) field. The CUC field is fixed to a P-Field (possibly extended) and 7-byte T-Field. The T-Field of the CUC format comprises two parts: the coarse time part and the fine time part. The former is in this case a 32-bit counter counting integer number of seconds. The latter is in this case a 24-bit counter counting fractions of seconds, from 2^{-1} down to 2^{-24}. The CUCTP packet is being used for transmitting time-information, it is however not used for transmitting the actual time synchronization events, for which SpaceWire Time-Codes are being used instead.

4. CURRENT PROTOCOL OPERATION

CUCTP provides synchronization between Elapsed Time (ET) counters in the local node and remote nodes, by means of SpaceWire Time-Codes and SpaceWire packets.

SpaceWire Time-Codes are continuously transmitted from a master node to all slave nodes. The transmission of the Time-Codes is synchronized with the local ET counter in the master node. The six bits of the Time-Code time-information correspond to six bits of the local ET counter (its exact mapping being programmable by means of register access). The ET bits with lower weights than the six bits mapped to the Time-Code time-information bits are all zero at time of Time-Code transmission.

When a Time-Code is received in a slave node, the Time-Code time-information is first verified to be an increment of the previously received time-information. The event of the Time-Code reception is assumed to occur synchronously with the local ET counter in the slave node.

Additionally, whenever the Time-Code time-information wraps from 0x3F to 0x00 it is possible to synchronize the ET bits that have a higher weight than the bits mapped to the Time-Code time-information bits. This is performed whenever a new CUCTP packet has been received preceding the reception of the Time-Code with the wrapping time-information. If no such packet has been received, then the synchronization will be as described above, but with an increment of the ET bits with the higher weight.

To summarize, ET bits mapped to the Time-Code time-information bits and ET bits with lower weight are checked for every Time-Code received; whilst ET bits with higher weight are checked whenever the time-information is wrapping. ET bits with lower weight can be offset from the all zero value. ET bits with the lowest weight can be ignored to form a window of tolerance.

5. CURRENT PROTOCOL IMPLEMENTATION

The SpaceWire - CCSDS Unsegmented Code Transfer Protocol interface IP core, named SPWCUC, operates in an AMBA bus system. The AMBA bus is used for configuration, control and status handling. The interface is tightly coupled with Aeroflex Gaisler’s CCSDS Time Manager (GRCTM) and SpaceWire codec with AHB Interface and RMAP target (GRSPW2) IP cores.

The IP core has already been integrated in the RASTA (Reference Avionics System Testbench Activity), and has been delivered to SciSys and Astrium for usage in activities related to CCSDS Spacecraft Onboard Interface Services (SOIS).

6. OUTLOOK

The current CUCTP protocol implementation does solve some of the problems related to time distribution in SpaceWire networks, but there is still some work to be done.

The CUCTP protocol is currently under review and modifications are being foreseen, possibly using an RMAP based approach. Also different methods to counteract latency, jitter and long term drift are being considered for further work. The goal is to include CCSDS based time distribution in the ECSS SpaceWire protocol standards.

7. REFERENCES

1. SpaceWire - Links, Nodes, Routers and Networks, ECSS-E-ST-50-12C
2. SpaceWire - SpaceWire protocol identification, ECSS-E-ST-50-51C
3. SpaceWire - Remote memory access protocol, ECSS-E-ST-50-52C
4. Time Code Formats, CCSDS 301.0-B-3, www.ccsds.org
5. RASTA Interface Control Document (ICD) - Software, TEC-EDD/2007.32/GF
6. RASTA Interface Control Document (ICD) - Hardware, TEC-EDD/2007.31/GF
7. GRLIB IP Library User's Manual, Aeroflex Gaisler, www.gaisler.com
8. GRLIB IP Core User's Manual, Aeroflex Gaisler, www.gaisler.com

THE QUANTITATIVE ANALYSIS AND RESEARCH OF SPACEWIRE DELAY JITTER

Network and Protocols

Short Paper

Chen Xiaomin, Hou Jianru, Cao Song, Sun Huixian

Center for Space Science and Applied Research, Chinese Academy of Sciences

E-mail: Chenxm@cssar.ac.cn , houjianru1985@gmail.com ,

caosong@cssar.ac.cn, shxian@cssar.ac.cn

1 INTRODUCTION:

Delay jitter is the key parameter to reflect the network transmission performance, which measures the difference between the maximum transmission delay and the minimum transmission delay from end to end. For large bandwidth traffic flow, greater delay jitter requires larger cache for sending and receiving. If the maximum transmission delay is too long, the real-time transmission performance of the network will fall, causing the bus performance degradation. Delay jitter performance has especially obvious impact on the quality of images and videos transmission with high bandwidth. Meanwhile, the highly real-time control services also have higher requirements on delay jitter performance of the bus. Currently, all kinds of satellites are equipped with more and more images-payload. Parameters like Delay jitter are always of concern to system designers.

Through theoretical calculations and modeling simulations, this essay carried out quantitative analysis and research for the delay jitter of the SpaceWire under specific application scenarios. Theoretical calculations get the delay jitter under particular scenarios by theoretical derivation. Modeling simulations, on the other hand, established simulation model by Opnet, and obtained the maximum transmission delay and minimum transmission delay by simulation. In this way, it is possible to calculate the delay jitter, qualitatively and quantitatively. By comparison, we obtain the parameters which have key impact on delay jitter. Recommendations and methods to improve the delay jitter are given by analyzing the conclusions. The research results of this article can provide a reference for the SpaceWire design to build a low delay jitter SpaceWire network.

2 CHARACTERISTICS OF TRANSMISSION SERVICE OF THE ON-BOARD DATA NETWORK

For accurately analyzing the delay jitter performance of SpaceWire network, the characteristics of transmission service of the on-board data network must be firstly clarified. Based on the requirements for the parameters such as bandwidth, real-time performance (delay jitter), and data reliability of the transmission service stream, the services can be classified into 3 types, as shown in Table 1.

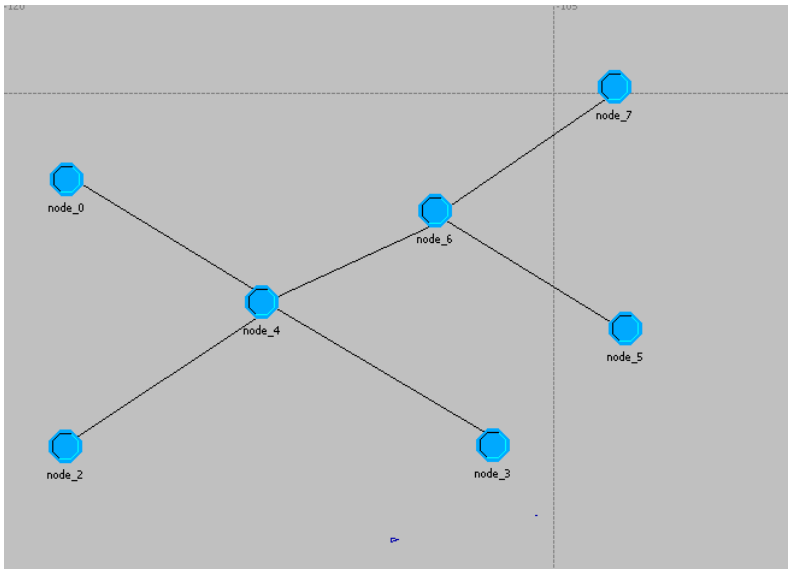
Table 1 Transmission Service Types of the On-board Data Network

Service Type	Bandwidth	Real-time Performance (Delay Jitter)	Data Reliability
Control service	low	average to high	high
High real-time data service	high	high	low
Low real-time data service	average	low	average

3 SIMULATION AND ANALYSIS

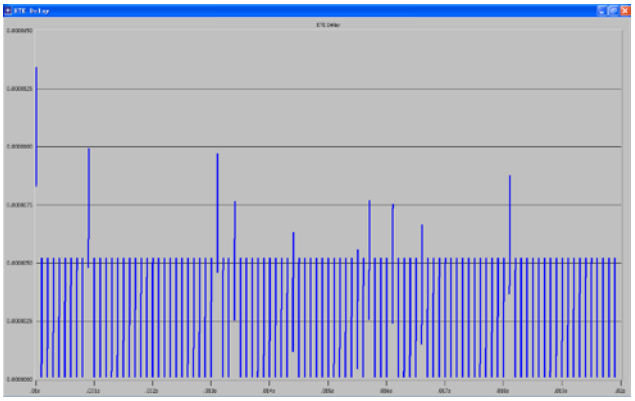
We first build a chain topology. Low-speed device node_0, high-speed equipment node_2 and node_5 are connected to the 4-port router node_4 and node_6. Node_7 is connected to a 4-port router which is a hot module (such as CPU, mass storage). Routers are connected to form a chain topology. Peripheral nodes, which are node_2 and node_5, send data flow fn1 and fn2. Fn1 is the controlling data stream which is low-speed and low real-time. Fn2 is the data stream sent by high-speed device.

Using Opnet Software to create the following model



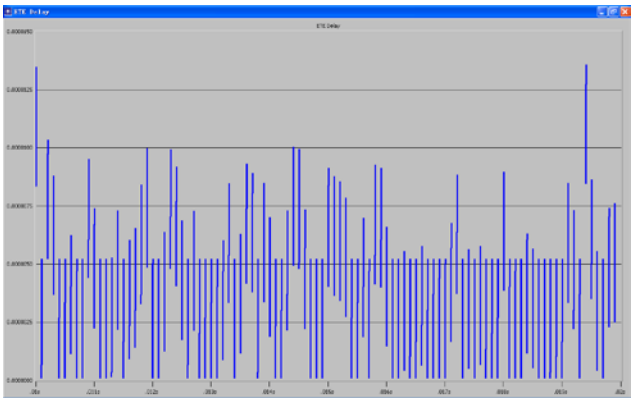
In this model, node_0, node_2 and node_5 are source nodes. Intervals between packets can be set to a variety of functions distribution. Here we set the intervals of the node_0 as constant distribution, and set the intervals of the node_2 and node_5 as random distribution, which are uniform patterns. In this case, when node_0 is counted, the packet intervals are consistent so that they are easy to compare. Meanwhile, node_2 and node_5 are sending packets randomly so that it is easy to manufacture collisions.

Node_4 and node_6 are routing nodes. Node_7 only counts the end-to-end delay when node_0 is sending packets. In order to see more directly the situation during the simulation, when packets are blocked causing the end-to-end delay increases, we set node_0 to send packets at the simulation time of 0.01s, node_2 at 0.009998s and node_5 at 0.009999s. Each of the three nodes sends packets of 1024 bit. First, we set the sending interval of node_0 is constantly 0.0001, and it send 100 packets in total. The sending intervals of node_2 and node_5 are random numbers between 0.0001 and 0.0002. Simulation gets statistics as following:



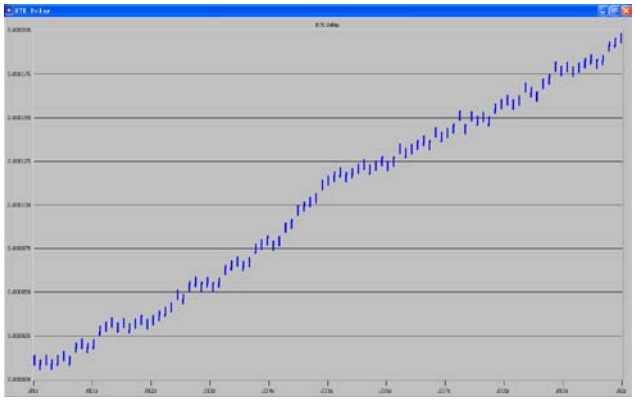
Statistics in the chart are the end-to-end delays. Every line consists of 128 dots each of which represents an N-char delay. In this simulation, the start time of end-to-end delay records from the creation of the package, and the ending time records from when each N-char is received. It leads end-to-end delay of each N-char increase. The highest point of each vertical line is the end-to-end delay of the packet. The simulation time of the task is 0.02s in total, but according to the simulation results, some packets' delay will be longer than average due to congestion. The first packet's delay is particularly long. This is deliberately made when setting the packet's sending time, which is in line with the expectation.

In above case, sending speed of node_5 is equal to node_0. Despite the first package reflects relatively long delay, the overall delay is quite steady. In our plan, node_5 is high-speed device. When node_5 sending according to the function which is randomly distributed from 0.00001 to 0.00002 (that is 10 times larger than the above), we get the following results.

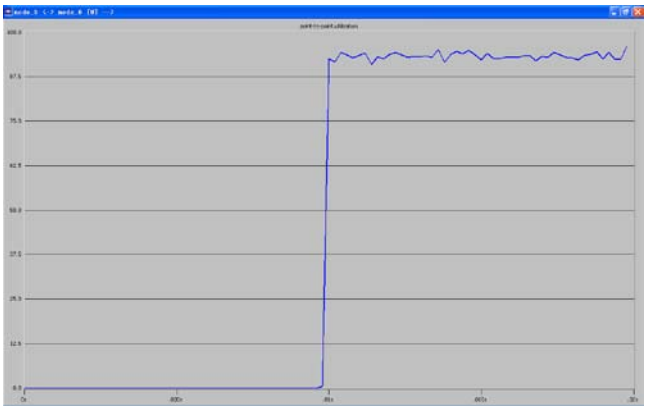


This shows that without high-level routing protocols, high-speed devices can cause interference with high real-time device, thus greatly increasing the end-to-end delay

of it. In extreme cases, node_5 sends packets according to the function which is randomly distributed from 0.000005 to 0.000006 (it close to the total capacity of the link). We get the following results:



It shows that end-to-end delay increases. This is because node_5 occupies almost all links so that node_6 is always in collision. The routing program used random selection for collision, while end-to-end delay records from the creation of the package, leads this result. We can see link utilization of node_5-> node_6:



4 IMPROVEMENT OF SPACEWIRE END-TO-END DELAY

In the simulation we can see, real-time data may not arrive on time due to the lack of high-level agreements. If only for extreme cases described in the end, priority rotation of the routes can solve the problem. But if we want to effectively control the delay jitter of a source, here are two options:

4.1 Division of Priority

When a packet with high priority enters the route, sending of the low-priority packet is immediately stopped and replaced with high-priority packets. Low-priority packet is put in cache and waits until high-priority data is finished.

4.2 Division of time fragment

Routing behaviour can be divided into multiple time fragments. Each fragment is allocated to different routing ports fairly. This is easy for implement, but will extend end-to-end delay of all packets.

5 CONCLUSION

SpaceWire bus standard is still expanding and improving. It is playing a more and more important role in analyzing the characteristics of the SpaceWire network transmission delay. Since SpaceWire technology has been successfully applied in a number of space missions, it is hopefully to become future bus standard of in-orbit. However, analysis and research on the characteristics of its network delay is still not enough.

This article analyzes the factors that influence the data stream delay characteristics of SpaceWire network under the typical topology structure based on an OPNET model. It carries out the quantitative and qualitative analysis on the delay jitter performance under different conditions, thereby providing a universally applicable method for designing the SpaceWire network and also the guidance for the design in the aspect of improving the delay jitter.

REFERENCE

1. Guo Lin, Cao Song, Chen Xiaomin. Research on The Delay Jitter Performance of SpaceWire, Network for Space Applications, IEEE ICCDA 2010

NETWORK MANAGEMENT AND FDIR FOR SPACEWIRE NETWORKS

Session: SpaceWire networks and protocols

Long Paper

David Jameux

European Space Technology Centre,

Keplerlaan 1, 2200 AG Noordwijk (The Netherlands)

E-mail: david.jameux@esa.int

ABSTRACT

Through several years of standardisation and technology development activities, ESA has prepared the SpaceWire technology that allows embarking high speed data networks on board spacecraft. This new technology has become widely adopted not only by ESA missions but also by other agencies and industries. However, some evolutions of the SpaceWire standard have been proposed by the SpaceWire Working Group.

The working group identified shortcomings of the current SpaceWire protocol stack in terms of network management and FDIR. This issue was already addressed several times within the frame of ESA funded R&D activities. First, the "Unified On-Board Processor Architecture for Spacecraft Avionics, Payload Processing and Data Handling" (UNIONICS) GSTP contract investigated the possibilities of task migration over a distributed SpaceWire network. Then, the TRP contract "Multi-processor On-board System for Robotic Exploration" (MOSREM) consolidated the concept by applying it to the most demanding application in terms of space on-board computing, i.e. space robotics. Recently, the GSTP contract "Modular Architecture for Robust Computing" (MARC) allowed proposing some FDIR scheme based on SpaceWire backplane networks.

These techniques are highly promising but they need to be harmonised and breadboarded prior to their eventual standardisation because they will be adopted by the SpaceWire community only if they are backwards compatible, i.e. if they can operate with existing SpaceWire devices.

This will be done in the frame of the ESA/TRP "Network management and FDIR for SpaceWire networks" to be kicked off in July 2011.

In this paper, we recall the need for the design of SpaceWire networking protocol to address the issue of network management and FDIR as well as the improvements foreseen to be developed, breadboarded and documented in ECSS standardisation format through the ESA/TRP activity "Network management and FDIR for SpaceWire networks". We inform about the achievements of the project team [in August 2011] and describe the steps still to be taken in order to prepare for the revision of the SpaceWire standard by the appropriate ECSS Working Group.

SPACEWIRE NETWORK PACKET ERROR HANDLING

Session: SpaceWire Networks and Protocol

Long Paper

Christopher T. Dailey

Dell Services Federal Government, Fairfax, Virginia, USA

E-mail: Christopher.T.Dailey@nasa.gov

Michael W. Pagen

MEI Technologies, Inc., Seabrook, Maryland, USA

E-mail: Michael.W.Pagen@nasa.gov

ABSTRACT

Packet error handling is an essential aspect for reliable, fault tolerant SpaceWire (SpW) networks. Without packet error handling, some faults in a subsystem could propagate through a SpW network, disrupting other packets or possibly the entire network. Due to SpW's unbounded packet size and wormhole routing, these faults must be mitigated at the network level. The packet error handling logic was revised in the National Aeronautics and Space Administration (NASA) Goddard Space Flight Center (GSFC) developed SpW Router Field Programmable Gate Array (FPGA) to automatically preclude packet fault propagation by adding logic on top of the SpW protocol which has been tested and performs as intended.

1 INTRODUCTION

SpW is becoming commonly used for communication networks between and within spacecraft subsystems. This is the case for the Magnetospheric Multiscale (MMS) Mission SpW network, which uses the NASA GSFC developed SpW Remote Memory Access Protocol (RMAP) and Node cores as well as Router FPGAs to connect subsystems within each of the four MMS spacecraft.

The GSFC SpW Router FPGA consists of a multi-port non-blocking routing switch. Multiple SpW Nodes and Routers are typically connected together to implement a SpW network. While SpW networks can have any topological form, including loops, SpW traffic on spacecraft typically resembles a funnel shape. On networks like these, such as MMS, some paths only carry packets to and from one node while other paths are shared, carrying packets from one or more sources to one or more destinations.

Shared paths can propagate faults when one source or destination fails such that a packet takes too long, either temporarily or (more likely) indefinitely to wormhole through the shared path. Note that brief stalls are normal consequences of packet funneling which contribute to packet latency through a network and are not faults. When a packet stalls for too long, other packets that need to use the shared path(s) are precluded from doing so, effectively propagating the fault in one board or subsystem over the network to other boards within the subsystem and/or to other subsystems.

A packet can take too long to wormhole through a shared path due to:

- A fault in the source that increases the packet size to be too long or infinite
- A fault in a source that stops sending a packet for too long (without ending the packet with an End Of Packet (EOP) or Error End of Packet (EEP) and without increasing its intended size)
- A fault in a destination that starts receiving a packet then stops for too long

The packet error handling logic was revised in the NASA GSFC SpW Router FPGA by the Code 561 Flight Data Systems & Radiation Effects Branch for, and funded by, the MMS mission to automatically preclude packet fault propagation. The packet error handling logic added to the GSFC Router FPGA exists on top of and transparent to the SpW protocol and assumes the SpW router(s) in a network are properly designed such that that radiation upsets or faults within the router(s) cannot credibly cause a packet to take too long to wormhole through a shared path.

2 SPACECRAFT SPW TRAFFIC

On spacecraft SpW networks, SpW nodes and routers are used to move command and telemetry packets between and within several subsystems. Below, a generic network topology shows command packets funneling-out from the processor in Figure 1 while telemetry packets flow back to the processor for processing and downlink in Figure 2.

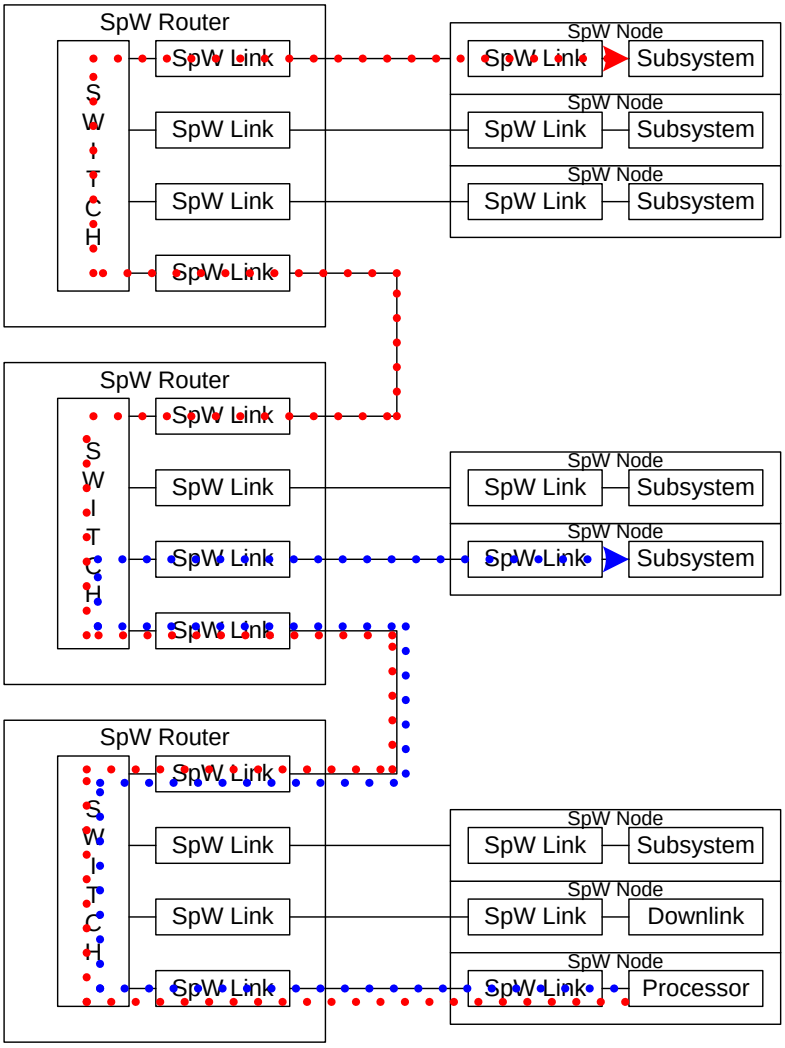


Figure 1: Generic Command Packet Flows

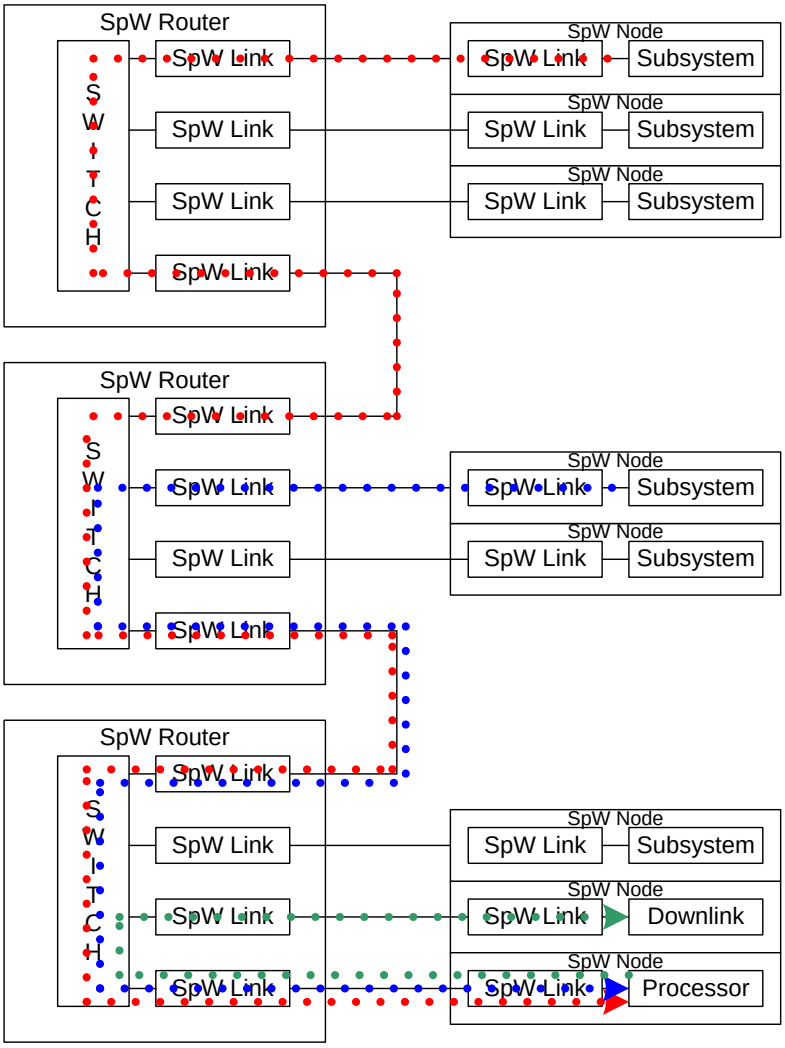


Figure 2: SC FSW Processed Telemetry Packet Flows

The command and telemetry paths within the SpW network have shared paths. A fault in any one of the subsystems that causes packets to take too long to wormhole through the shared paths would block communication to/from the others, propagating the fault.

3 PACKET ERROR HANDLING

Packet error handling is provided in the GSFC Router FPGA and not in the GSFC Node core as its purpose is to prevent fault propagation on shared paths. A node and the link it connects to comprise a dedicated path in a SpW network. Therefore a fault in a node's board or subsystem only affects that board or subsystem, provided the fault is not propagated. However, inside a router, the routing switch can have shared paths which, if blocked, can propagate faults.

3.1 NETWORK LEVEL

Faults that leave wormholes open on shared paths have to be mitigated at the network level, specifically the network switch boundaries, as these cannot always be mitigated in other levels. This is a consequence of SpW's unbounded packet size and wormhole routing.

Since packets can be of any size, there is no time limit on wormhole routing in the SpW standard. Therefore, a wormhole that remains open does not violate any of the "rules" for the physical, signal, character, exchange, packet or network levels. In fact the rules are followed in order for the wormhole to remain open. The link remains in the RUN state but is only passing NULL characters (stalled packet) or is also passing bytes of data (infinite packet) but not an EOP or EEP.

Mitigating at the transport or application levels, such as automated (watchdog) or operator initiated resets or power cycles, may or may not stop fault propagation.

If a fault was due to a transient problem then resetting or power cycling may allow operations to be restarted, provided the cause of the problem was cleared by the reset or power cycle. When faults propagate, it can be difficult to determine the cause and therefore difficult to know what to reset or power cycle. Also, resetting or power cycling is typically best left as a last resort as doing so can erase status information, making troubleshooting more difficult, and usually has significant mission impacts.

If the fault is persistent, then the stuck-open wormhole problem would re-occur after the reset or power cycle, re-propagating the fault. Resetting or power cycling, even repeatedly, would not stop fault propagation in this case.

Thus it is best to mitigate stuck-open wormhole faults at the network level, above the SpW protocol and below any transport or application level services, such as RMAP.

3.2 STUCK-OPEN WORMHOLE MITIGATIONS

Packet error handling to prevent fault propagation from stuck-open wormholes is implemented via two types of limit checks: maximum packet size and packet timeout. If either limit is exceeded, the packet is truncated with an EEP so that the wormhole is closed. Closing the wormhole allows subsequent packets to pass through the path that was stuck, thereby limiting the fault to its source or destination in the SpW network and precluding a fault with this packet from propagating to other packets.

The maximum packet size limit is selectable from several values including an option to disable this limit check (allow packets of any size), via writing registers within the routers. When enabled, the number of bytes in each packet is counted as the packet bytes traverse the routing switch. If the byte count exceeds the limit, the routing switch logic performs the packet error handling steps below.

The packet timeout limit is also selectable from several values including an option to disable this limit check, via writing registers within the routers. When enabled, a timer starts as packet bytes traverse the routing switch. If a timeout occurs, it is due to either a fault in the source or destination. If the empty flag of the source First-In First-Out (FIFO) is asserted for longer than the timeout value, then a fault has occurred in the source of the packet. If the destination FIFO's full flag is asserted for longer than the timeout value, then a fault has occurred in the destination of the packet. Depending on which occurred, the routing switch logic performs the packet error handling steps below.

The packet error handling steps are:

- Disconnect the path between the source and destination switch ports inside the router
    - Subsequent packets from other sources can then arbitrate for the destination port, thereby precluding fault propagation from faults in packet sources
- Discard the remainder of the packet from the source port by draining the source FIFO until an EOP or EEP is found
    - This may never complete if the source is sending an infinitely long packet or has stalled sending a packet
    - If the source completes sending the packet (with an EOP or EEP) then subsequent packets from this source can arbitrate for destination ports in the routing switch
        - This would be the case if the fault did not occur in the source or the source was able to recover from the fault
- Truncate the packet at the destination port by appending an EEP
    - If the destination port has failed to read the packet before the timeout value then the destination port is marked as failed and any subsequent packets requesting this port will be discarded
        - This allows any subsequent packets arriving through the source port to arbitrate for other destination ports, thereby precluding fault propagation from faults in packet destinations
        - If the fault in the destination is fixed such that its destination port FIFO in the router is read then the destination port's fail flag is automatically cleared and the destination port can resume receiving new packets
- Set the appropriate error status

3.3 Network Switch Boundaries

Packet error handling should be performed as close as possible to the cause of the fault. This can be done by only performing packet error handling at the network switch boundaries, where packets begin or end their wormhole paths through one or more routers. Packet error handling could be performed at intermediate points inside the network switch boundaries but doing so is unlikely to be beneficial.

Packet error handling can be enabled or disabled for each switch port in each router individually so that it is only applied to the network switch boundaries. This is necessary to preclude destination timeout packet error handling from occurring in intermediate routers along a packet's wormhole path. Otherwise, destination timeouts

could disable destination ports in intermediate routers, which would be a form of fault propagation, rather than just the destination port in the last router.

3.4 VERIFICATION AND VALIDATION

The packet error handling logic has been verified through simulation and inadvertently validated in integration and test of MMS. Software sent a command packet with a logical address that opened a wormhole from the processor to a node which was mis-configured and did not accept the packet. This stalled the packet and resulted in a destination port timeout in the (properly configured) router port, which disconnected the wormhole at the destination node. Subsequent packets sent by software to other subsystems were then able to pass through the router.

The configuration fault was not propagated, which provided for quicker troubleshooting as a communication failure was reported. Had the packet error handling logic not been present (and enabled), software would not have been able to send any packets to any other subsystems after sending the command packet. The problem likely would have been reported as the whole SpW network locked-up and finding root cause would have taken longer.

4 CONCLUSION

This paper described the packet error handling enhancements added to the GSFC SpW Router and how these are applied to the MMS SpW network. Rationale for performing this error handling at the network level, on top of the SpW protocol and not at higher levels, as well as at network switch boundaries was also discussed. The packet error handling logic has been verified though simulation and validated in integration and test.

Components 1

LOW MASS SPACEWIRE

Session: Components

Short Paper

Gilles Rouchaud

AXON' CABLE S.A.S. - Route de Chalons-en Champagne - 51210 Montmirail - France.

Jorgen Ilstad

ESTEC, Keplerlaan 1 + P.O Box 299 - 2200 AG Noordwijk ZH - The Netherlands.

Florent Mettendorff

AXON' CABLE LTD -AXON' AGORA - Admiralty park- Rosyth - Dunfermline-Fife KY11 2YW – United Kingdom

E-mail: g.rouchaud@axon-cable.com, Jorgen.Ilstad@esa.int, f.mettendorff@axon-cable.co.uk

1 INTRODUCTION

This presentation concerns the **Low Mass SpaceWire** project, reference **A0/1-6214/09**, in cooperation with the European Space Agency. There is an existing standard for SpaceWire and its reference is ECSS-ST-50-12C [1].

It would appear that this existing SpaceWire standard is too oriented to detailing the cable construction when the reality of space applications require greater focus to be given to the physical and electrical properties and particularly to the length and the flexibility of the SpaceWire cable assembly (Link).

A possible consequence is that long SpaceWire links may be too lossy, while short connections may be more rigid and heavier than necessary. Another possibility is that the SpaceWire solution may sometimes be discarded by users as being too simplistic and having too many physical and electrical limitations compared to what they need for their application.

This project provides an opportunity to review the SpaceWire standard with the primary objective of reducing its mass by half. The approach is in three steps: development, manufacture and test of a Low Mass SpaceWire cable. The same or equivalent performance levels shall be maintained for the cable assembly.

2 DEVELOPMENT

This project started by the definition of additional physical and electrical parameters for the Low Mass SpaceWire cable plus a review of the existing requirements. This information was compiled in a Requirements Specification Document (RSD). Among these a specification for insertion losses have been added and the skew has been reduced.

The preliminary design of the cable was based on this RSD and shows at this stage a few potential solutions.

Depending on the forthcoming test results, the main construction changes could be :

- the removal of the overall shield covering the four inner shielded twisted pairs or overall and inner shields in contact

- the silver plated copper shields to be replaced by silver plated aluminium

- full cable shielding and termination through bulkhead connector/backshell interface, unlike the current inner shields of the two signal pairs on each side left floating.

- Use of non-twisted sub-miniature coaxial cables instead of shielded twisted pairs as a potentially very interesting and flexible solution for short lengths. This technology also could allow the forming of the cable into a ribbon shape.

- Polyimide material for the outer insulation instead of PFA for improved irradiation behaviour

The calculations indicate these changes should make the required mass reduction possible (preliminary manufacturing reached 55g/m and 32g/m for the subminiature coaxial cable assembly, as opposed to typically 80g/m for standard SpaceWire).

A second objective of this project was to investigate the possibility of an existing, alternative matched impedance connector to the current rectangular micro-miniature Micro-D) connector defined in the standard.

As a result of the survey, two types of connectors show potential for a SpaceWire application:

- one developped by an Axon competitor with NASA (4-way twinax)

- one developped by Axon with CNES (AXOMACH)

Other configurations of the microminiature connectors (circular or with additional EMI protection) are also under consideration. And a nanominiature potential alternative (albeit without matched impedance) is being investigated.

As the project progressed, an interesting potential innovation in the cable properties was identified. The idea was to explore the feasibility of using a slightly conductive

material as an outer insulation of the cable to support applications exposed to space and therefore requiring improved eletrostatic discharge (ESD) performance.

An investigation revealed that this matter goes beyond the frame of the current Low Mass SpaceWire project, and it was agreed with ESA that it should be potentially considered as a separate project by itself.

3 MANUFACTURE AND TESTS

At the time of writing this short paper, the manufacture of four different Low Mass SpaceWire candidate cables was still on-going.

The quite radical changes in the design imposed some optimisation to the existing manufacturing processes. But the few issues encountered during the first production have now all been overcome and processes mastered.

The four cable samples in manufacture shall serve as test vehicles to undergo the Qualification Test Plan (QTP) recorded in a Test Specification Document and validated by ESA.

The tests concerned will cover the physical, electrical and mechanical aspects required for the Low Mass SpaceWire.

Moreover an innovative test from ESA for **conductive susceptibility** measurement will be included. The purpose of this test is to characterize the robustness of a Low Mass SpaceWire cable assembly against external EMI disturbances. The procedure was proposed by ESA along with a suggested test equipment scenario. Some test components were manufactured in partnership between Axon and Astrium in France.

4 SPACEWIRE SPECIFICATIONS UPDATE

Once the tests are completed and the conclusions drawn from the results, in collaboration with ESA and one of the original authors of the existing SpaceWire standard (Steve Parkes), Axon has to prepare:

- a SpaceWire Cable specification and a PID

a draft, revised issue of the ESCC3902/003 standard [2] (just the cable only specification of the Low Mass SpaceWire (not terminated to connectors)) was created during the development phase to help the future ECSS update mentioned previously. This specification retains the 2 existing conductor solutions (AWG26 and AWG28) and includes some new lightweight variants.

- a SpaceWire Standardisation document

The intention is to update the sections relating to the cable specification in the SpaceWire standard ECSS-ST-50-12 eventually.

5 CONCLUSION

Despite the significant challenge launched by ESA to reduce by half the mass of the existing SpaceWire cable, the development phase has shown that, in theory, it is achievable using new or modified designs.

Furthermore, the new media proposed will be better featured and have a higher margin of performance in relation to the proposed new wording of the standard.

6 REFERENCES

1. European Space Agency - ECSS Secretariat, ECSS-E-ST-50-12C, Space engineering, SpaceWire – Links, nodes, routers and networks , 31st of July 2008, 129 pages.

2. European Space Agency, ESCC Detail Specification No. 3902/003 issue 2, CABLE, "SPACEWIRE", ROUND, QUAD USING SYMMETRIC CABLES, FLEXIBLE, -200 TO +180 OC, June 2008, 24 pages.

IMPLEMENTATION ASPECTS OF THE PHYSICAL LAYER IN SPACEWIRE

Session: Components

Short Paper

Wahida Gasti, Jorgen Ilstad, Farid Guettache, Giorgio Magistrati

ESTEC / European Space Agency, Noordwijk , Netherlands

E-mail: wahida.gasti@esa.int, farid.guettache@esa.int, Giorgio.magistrati@esa.int, jorgen.ilstad@esa.int

ABSTRACT

This paper will focus on the following three areas of the SpW physical layer:

What are the pros and cons of discrete vs. integrated implementation of EIA/TIA 644 LVDS transceivers in flight units?
A trade-off between the two possible implementations of SpW transceivers in on-board equipment will be presented. The advantages or disadvantages of an implementation using discrete LVDS transceivers external to the ASIC/ FPGA versus an integrated LVDS transceiver solution embedded in the ASIC/FPGA will be looked at closely. Key criteria considered are: application (inside or intra units), fault voltage susceptibility, robustness against ESD, redundancy and cross strapping aspects, risks and associated effects in case of failures etc.

How are fail safe requirements defined in ECSS-E-ST-50-12 verified?
The verification of the fail safe requirements defined in section 6.2 of the ECSS-E-ST050-12C is analyzed in particular for implementations where the LVDS transceivers are embedded in an ASIC/FPGA.

How does common mode voltage drift affect communication integrity?
Results from tests related to signal and communication integrity in presence of a common mode voltage difference between units will be presented. The two LVDS transceiver options, embedded vs. discrete transceivers, as discussed above, will conclude the paper. The tests results include SpaceWire components with embedded LVDS transceivers and discrete (external) LVDS transceivers with both nominal and extended common mode voltage ranges.

Introduction

The SpW physical layer is based on differential signal transmission. ANSI/EIA/TIA-644a LVDS is the technology used for the physical layer. LVDS as differential transmission mode presents many advantages over single ended signaling by allowing high data rate, low power consumption, immunity to noise and low EMI, but on the other side it requires extra care to be taken in the design to preserve the symmetry of the differential signal, to keep the impedance matching and to deliver at the receiving end the required quality of the signal. In the beginning mostly discrete circuits from various manufacturers has been used. Nowadays with increased complexity of SpW based designs like router or SoC, the trend is to integrate LVDS transceivers in the ASIC. The same is also true for the FPGA circuits, and the most recent include LVDS transceivers. This option at first glance seems attractive mainly to save area on the PCB. Interoperability between different implementations requires compliance to ANSI/TIA/EIA-664a in the development of the circuit and to follow recommended practices and design guidelines in the design of the application (PCB), the objective is to preserve the integrity of the signal. In this paper our interest will focus mainly on integrated LVDS transceivers in particular those embedded in the Atmel SpaceWire 10X router. The performances of embedded transceivers will be compared to those of the discrete circuits. In the first part of the paper the main advantages to use integrated drivers versus discrete will be outlined. In the second part and starting from SpW standard failsafe requirements, it is addressed how to proceed for verification in the case of integrated transceivers and finally in the last part how common mode drift will affect the signal integrity.

1-Pros and cons of discrete versus integrated implementation of LVDS transceivers in flight units

The trade-offs between the two possible implementation of LVDS transceivers in on-board equipment is presented in the current paragraph (embedded LVDS drivers in FPGA/ASIC and discrete LVDS transceivers).We will focus on the following criteria's:

General signal consideration

-Signal integrity, power-thermal, ESD etc,

And more specific to space

-Redundancy and cross-trapping and risk associated in case of failure

1-PCB issues:

When placing LVDS drivers on PCB the recommended stub maximum lengths is 2.5 cm. It is still possible to use stub with trace lengths longer than 2.5 cm, but transmission can suffer from problems like ringing, overshoot, undershoot, stair step waveforms crosstalk and reflections. If an LVDS transceiver is integrated into an FPGA/ASIC the possibilities of placement on the PCB and close to the backplane connector are limited and depend on several factors among them:

- The I/O density in the FPGA
- Size of the board i.e. distance between component and connector
- PCB layers
- Component placement density
- Signal rate i.e. transition time of the rising and falling edges of the signal.

These factors have a direct an impact on the signal integrity. High density of embedded I/O and component on the board can make the PCB design more difficult and more complex in particular if we want to preserve the signal quality. However embedded signal drivers free up space to implement other components and enable also to overcome the CMOS connection length limiting factor
Discrete I/O enables the placement of drivers on the PCB for optimised distance to the connectors and the board designer will have less hassle about impedance matching along the transmission tracks. In case embedded drivers is the only solution, LVDS repeater can be used to overcome the limitation

2-The I/O load capacitance of an FPGA can be higher than for a discrete I/O (approximately double) A higher capacitance tend to lower the transmission line impedance, and narrow the available noise margin (example in the SN65LVDS31circuit the input capacitance is 3 pF, and for Actel serie RTAX-S/SL Radiation-Tolerant FPGAs the input capacitance is 10 pF)

3-Power: using discrete I/O buffers, which are dissipating devices, helps dissipate heat to keep the ASIC/FPGA cooler (Atmel router consume 3.6 w, at 200Mbits/s with all ports active). We can add flexibility when selecting discrete LVDS drivers due to the fact that drivers with appropriate power supply can be selected.

4-ESD: Discrete I/O have often higher tolerance against ESD (10kV) than FPGA/ASIC I/O (2-2.5 kV)

5-Environmental noise: the receiver is usually connected to a harness which can collect noise and static electricity from the environment and discrete drivers may enable to isolate the FPGA/ASIC from this noise

6-Gain from the ASIC rad-tolerant design which is not always the case for discrete circuit

7-Discrete I/O devices often use larger technology (example 0.35 um) than ASIC/FPGA which make them more robust to high voltage and current and more immune to noise and EMI. But ASIC/FPGA have more resources (in terms of available transistors) to make additional protection in the design

8- Cross-strapping and error propagation
In the ECSS-E-ST-50-14C it is stated regarding cross-trapping:

> *In case of signal cross-strapping, no single failure of either interface circuit shall propagate to the other one*

Two cases should be considered: warm redundancy and cold redundancy. In warm redundancy and in operation the LVDS driver is always active and the cable can be considered never disconnected from the driver and the receiver input is never shorted together. In cold redundancy the link is not always active and when the driver output is in high impedance the receiver can be considered disconnected, when the transmission line is not shielded (case PCB traces) it can act like antenna and collect noise, in the case of a shielded cable the risk is lowered. Discrete circuits offer more flexibility than ASIC/FPGA as it is always possible to power-off or disable the

redundant link. Even in case of failure of the LVDS transceiver the error has little chance to propagate beyond the transceiver to reach other stages of the system. A failure of an embedded transceiver could expose the whole ASIC/FPGA to the risk of being damaged

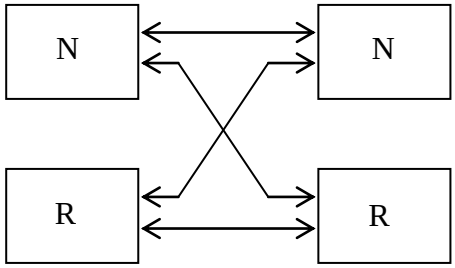


Figure 1: Full cross-strapping of nominal and redundant on-board units.

2-How are fail safe requirements defined in ECSS-E-ST-50-12 verified?

The paragraph 6.2 (Failsafe operation of LVDS) of the ECSS-E-ST-50-12 [1] state the following:

> ***A*** *When any of the following fault conditions occur, the receiver outputs shall not oscillate and shall be locked to high logic level provided that a noise threshold of 10mV is not exceeded at the receiver input.*
>
> 1. *Driver not powered.*
> 2. *Driver disabled.*
> 3. *Driver not connected to receiver*
> 4. *Receiver inputs open circuit (i.e. cable or wire in cable disconnected).*
> 5. *Receiver inputs shorted together*
>
> **B** *When the driver is not powered its output should be high impedance i.e. > 100 kΩ.*
>
> ***C*** *When the receiver is not powered its input should be high impedance i.e. > 100 kΩ.*

For embedded LVDS transceivers [6,8] it is difficult to test the case a) because the receiver output are not accessible for measurement and is connected directly to the SpW codec. If we inject continuously one of the sequence outlined below:

- Start the SpW link - short together the receiver input - restart the SpW link.
- Start the SpW link - disconnect receiver inputs – restart the SpW link.

The exchanged packets between the two endpoint of the link can be recorded and analysed later. Stored packet will help to detect any impact of the faulty operations on the receiver inputs side.

In theorie to detect receiver oscillation the power supply consumption could be monitored for slight variations.

Simulation using PSpice and IBIS models is an alternative way that can be used to analyse fail safe conditions and to verify the data sheet specs.

The starting sequence of the SpW protocol seems robust enough to cope with the faulty condition mentioned in the SpW standard because it constitute a reel barrier against these faulty conditions and it seems to supersedes the fail-safety of LVDS drivers. A continuous stream of error free characters (NULL and application data) is the absolute condition to maintain the link alive. In the event of spurious transitions on a transceiver the likelihood of this being determined as valid data is very small due to the odd parity scheme, causing the link to disconnect if not obeyed, and the fact that the data has to match certain values to be accepted by the node. The latter case is especially true when higher level protocols are used.

3-How does common mode voltage drift affect communication integrity?

The illustrated test set-up in figure 2 is a SpW connection channel between an Atmel router using embedded MH1-RT LVDS transceivers and an SMCS116 [13] evaluation board using the Aeroflex LVDS transceivers for it physical layer

We have performed common mode variation test using DC signal as well AC signal and monitoring the signal integrity through SpW startup sequence (disconnections, parity errors). The common of Atmel router SpW interface has been subjected to a voltage variation.

For the DC test we noticed that common mode difference that +/- 1V is conceivable and has not impact on data integrity on the SpaceWire link. Further tests revealed that common mode voltage shifts up to 1.4V did not cause data corruption with subsequent disconnect.

For The AC conducted susceptibility test a sinusoidal waveform with a maximum amplitude of 1 V (2 V pkpk) in a frequency range of 50kHZ – 100MHz has been injected and the SpW link has been monitored for degradation of performances at each test frequency. The SpW communication link was operated at two different data rates: 10 Mbit/s and 100 Mbit/s and has been kept active during the whole test. The results are summarized in the table 1, we can notice that there was no perturbation of the connection for the injected disturbance signal at frequencies lower than 100MHz but at 100 MHz the shield start to loose its effectiveness and from certain level of signal amplitude errors start to appear on the SpW link. The shield seems appropriate to remove any intrusion of common mode fluctuations.

Frequency (MHz)	Test results
0.05	Ok no disconnection
0.1	Ok no disconnection
0.3	Ok no disconnection
1	Ok no disconnection

3	Ok no disconnection
10	Ok no disconnection
30	Ok no disconnection
100	Ok up to slightly lower than the max level of 1V

Table 1: Summary of results from AC common mode test

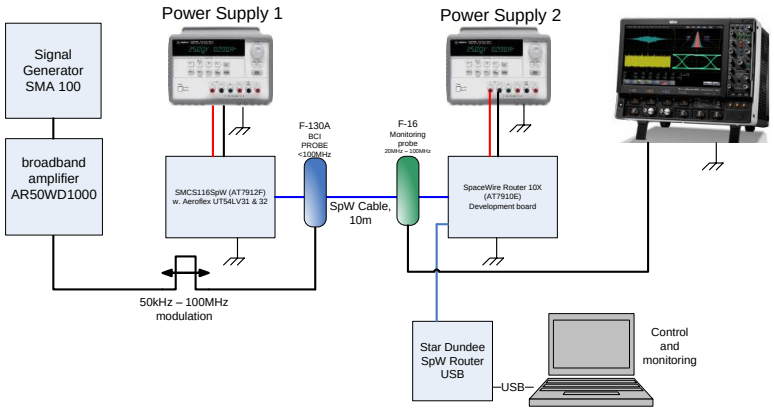


Figure 2: EMC AC Conducted susceptibility test.

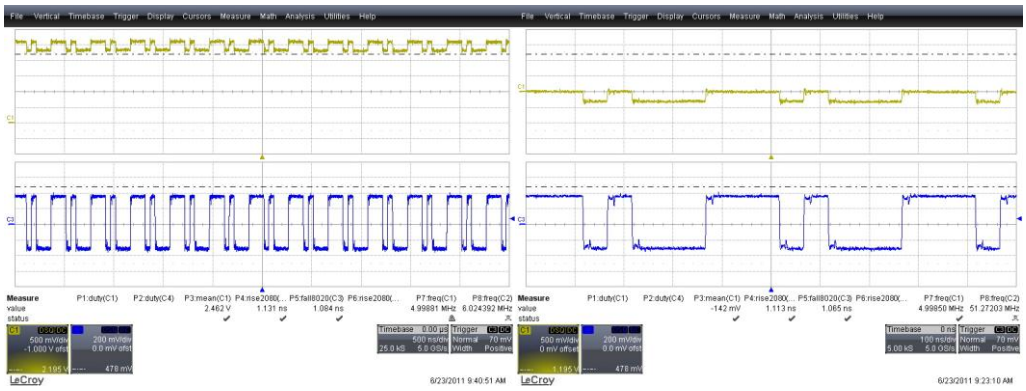


Figure 3: Oscilloscope plots from DC common mode voltage shift.

C1 = Input A of Aeroflex Receiver (Strobe of Port 7).
C2 = Differential Voltage at Aeroflex Receiver Inputs (A-B).Delta V from 0 up to 1.4 V (.2,.8,1, 1.2, 1.4 V) have
been over-imposed.. No errors (disconnection or parity) occurs
until the DeltaV reaches 1.5 V

C1 = Input A of Aeroflex Receiver (Strobe of Port 7).
C2 = Differential Voltage at Aeroflex Receiver Inputs (A-B).
Delta V from 0 down to -1.4 V (-.4,-.6, -.8, -1, -1.2 V) have been over-imposed.
No errors (disconnection or parity) occurs until the DeltaV reaches -1.4 V !!!

Conclusion

The LVDS standard TIA-EIA-644 requirements are defined with sufficient margins that even if fully compliance is not achieved, the LVDS transceivers is able to deliver and decode signals with the required quality level.

Embedded transceivers and discrete circuits have pros and cons, the selection is application dependant, signal integrity is an important metric but at the end it is up to the designer to perform the trade-off.

References

1. ECSS-E-ST-50-12C Spacewire -Links, nodes, routers and networks
2. ANSI/TIA/EIA-644-A-2001 Electrical Characteristics of Low Voltage Differential Signaling (L VDS) Interface Circuits
3. ECSS-E-10-03A testing
4. ECSS-E-ST-20-07C Electromagnetic compatibility
5. ECSS-E-ST-50-14C Spacecraft discrete interfaces
6. Technical Note on LVDS in MH1RT Asic Technology 4 November 2011 ATMEL
7. Test on LVDS Components at ESA 29 June 2011 LVDS-Day TEC-ED, TEC-EDD, TEC-EDP
8. AT7910E SpW-10X SpaceWire Router Datasheet 7796-AERO-07/09
9. UT54LVDS031LV/E Low Voltage 3.3 V Quad Driver Aeroflex Datasheet Dec 2008
10. SMCS116SpW User manual Astrium
11. Data sheet SN65LVDS32 T.I
12. AN1194 Failsafe Biasing of LVDS Interfaces Dec 2001
13. Using LVDS for Actel's Axcelerator® and RTAX-S/SL Devices Actel A.N
14. Virtex-E High Performance Differential Solutions: Low Voltage Differential Signalling (LVDS)
15. Using LVDS in APEX 20KE Devices Altera
16. LVDS owner manual Nation Semi-conductor

NASA-GSFC REMOTE MEMORY ACCESS PROTOCOL TARGET IP CORE

Session: Components

Short Paper

Omar A. Haddad

Dell Services Federal Government/NASA GSFC Code 561

Greenbelt MD 20771

E-mail: omar.a.haddad@nasa.gov

ABSTRACT

The Magnetospheric Multiscale mission (MMS) was the first Goddard Space Flight Center (GSFC) project to adopt Remote Memory Access Protocol (RMAP) over SpaceWire. Adopting RMAP on several SpaceWire-enabled boards of the mission reduces the effort required of board and FPGA designers to implement the SpaceWire interface. It also simplifies the flight software because fewer SpaceWire protocols need to be supported. This paper describes the features of RMAP that GSFC implemented, discusses development and testing issues encountered, and summarizes the end results of implementing RMAP over SpaceWire on MMS.

1 THE ADVANTAGE OF RMAP

Like any network, SpaceWire has layers of communication. Figure 1 below shows the layers implemented on MMS, in accordance with [1]. RMAP resides in the packet layer.

Figure 1

The advantage of implementing RMAP on a board that is considered a target, peripheral, or end point is illustrated in Figure 2 below. While the User Logic block is design-specific, all other blocks are provided to the board/FPGA designers as reference designs which require no development. This is a significant advantage over past designs where the board/FPGA designers needed to dive deeper into the lower layers in order to properly design, debug, and test the SpaceWire interface.

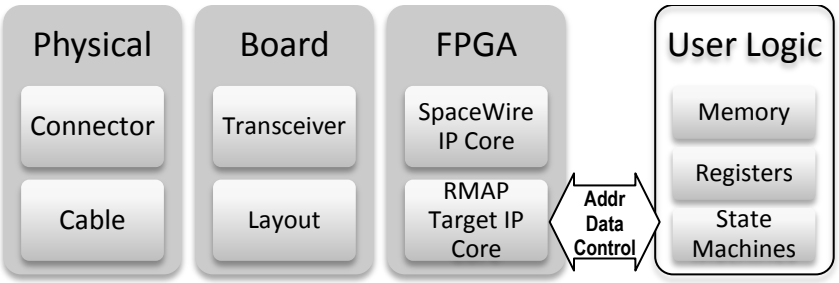


Figure 2

1.1 HIDING THE PACKET LAYER

With RMAP as the packet layer, the user logic connects to a simple address/data/control bus and maps the functionality and control of the board into an address space that can be accessed by the network host. Handling packet parsing, validation, error handling, and recovery is done by the RMAP Target IP core and doesn't need to be thoroughly understood by the board designer.

1.2 REINVENTING THE WHEEL

Since each RMAP-enabled board is given the same reference blocks for their SpaceWire interfaces, the design only needs to be reviewed once. The software interface to all the RMAP enabled boards can use a common element that handles RMAP packets. The hardware and flight software reference designs can be reused from mission to mission without having to 'reinvent the wheel' for each one.

2 RMAP FEATURES IMPLEMENTED

The RMAP standard [2] spells out the partial implementation of the RMAP standard. This section discusses the features of RMAP standard that were implemented by the GSFC RMAP Target IP core along with some non-standard features that were found to be useful.

Read and Write Commands – The GSFC RMAP target supports all variations of read and write commands. The verified write commands are limited to a length of 4 data bytes. This is long enough for critical 32-bit registers to be safely written to and doesn't require a large internal buffer to hold data. The RMW command was not implemented in the GSFC RMAP target core because the MMS mission only had one RMAP initiator, thus, obviating the need for the RMW command.

Event Signalling – Reference [3] provides a description of the *event signalling* feature which was later left out of reference [2]. With this feature, a read command triggers a user-defined event which must complete before the response is generated and returned to the requestor. This feature was used by MMS to request and capture 'freshly' sampled analog telemetry. The read command would trigger an ADC conversion event. Upon completion, the result was returned to the requestor so that stale data is never used.

Bypass Port – For RMAP-enabled target devices that wish to support additional SpaceWire packet formats, the GSFC RMAP target core implements a bypass port that diverts non-RMAP packets past the RMAP target core. The bypass port is bidirectional and uses the logical address in the packet header to determine how to route packets.

SpaceWire Device Register Interface – The GSFC RMAP target core has a special register interface that can be connected the SpaceWire IP Core which establishes communication over the SpaceWire link/cable. This allows the network host to use RMAP commands to read SpaceWire status and configure the SpaceWire operation of that device.

Back-end Timeout – The GSFC RMAP target core was written such that access to the user's address space hand-shake with the user's logic to know when the read/write transaction had completed. Response packets can then be generated. However, if the user's logic never ends the transaction, the RMAP core would wait forever and effectively lock-up. To avoid this scenario, the RMAP core has a programmable timeout counter. If the RMAP core encounters a timeout while waiting for the user logic to respond, the transaction is aborted and an error status is returned to the RMAP initiator/host, if requested.

3 DEVELOPMENT AND TESTING ISSUES

The GSFC RMAP target core was developed and tested as an in-house effort. This section describes some of the challenges and issues that were overcome.

3.1 CRC ALGORITHM CONFUSION

The RMAP format contains two CRC fields, one for the header and one for the data. To assist with calculating the CRC fields properly, reference [3] contained an appendix with sample VHDL and C code that calculates the CRC. The sample code provided contained errors; so the first implementation of the GSFC RMAP target core did not generate the CRC field correctly. This error in reference [3] was corrected in reference [2], however, the error was not discovered in the GSFC core until it was used with third party software that generated the CRC differently.

3.2 LESSONS LEARNED

With so many users of the GSFC RMAP IP core, it became evident that the specification documentation for the core had to be very detailed and clearly written. There were many instances where users required the specification to be updated to contain the information they needed such as data field byte order, transaction latencies, and handling of multiple event signalling transactions.

3.3 COMPLIANCE

Another NASA mission, Astro-H, used the GSFC RMAP IP core to interface to a component built by JAXA. Astro-H performed RMAP standard compliance testing on the GSFC RMAP IP core and found it to be 100% compliant with the mandatory functionality.

4 RESULTS

Implementing standardized interfaces often comes with unnecessary overhead that makes a design less optimal in resource usage. This cost is incurred in the hopes that it is outweighed by the advantages that it buys. Therefore, it is important to assess the benefits of using RMAP on MMS.

4.1 REUSE

In terms of reuse, the concept of using RMAP on target devices that can be fully controlled by address mapped logic was very successful. Not only has MMS taken advantage of this, but multiple other GSFC missions are now using the RMAP target IP core and it is anticipated that many future GSFC missions will also adopt it. The

IP core was used on multiple FPGAs by multiple agencies without requiring individual modification. Even elements of the verification environment were reused from one test bench to another.

4.2 SIMPLIFICATION

Sharing a reference circuit design and FPGA IP core worked out very well and made the effort of designing a SpaceWire RMAP-enabled board less complex. Each board designer received a sample test bench tailored for RMAP-enabled boards. This test bench was used as a starting point for board/FPGA designers to use when exercising their user (board-specific) logic.

4.3 STANDARDIZATION

The use of RMAP enabled the MMS flight software to standardize on how SpaceWire nodes are controlled. This reduces software complexity and increases reliability. Designing a RMAP target IP core that is standard compliant was a task of reasonable effort that has paid dividends many times over. The standardization of the SpaceWire protocol has been also been an advantage in selecting and using lab test equipment with multiple RMAP-enabled devices.

4.4 SUFFICIENCY OF RMAP

Although the adoption of RMAP was a positive experience on MMS, it was determined that RMAP alone is not enough to meet the architectural needs of MMS. The reason for this is that RMAP requires that target nodes do not 'speak' unless 'spoken' to. For this reason, it was necessary to implement another SpaceWire packet protocol that allows for target nodes to freely forward data as it becomes available. Although potential RMAP solutions were considered, they were ruled out due to the complexity of fault detection and recovery.

5 REFERENCES

1. European Cooperation For Space Standardiziation, "SpaceWire – Links, nodes, routers, and networks", ECSS-E-50-12A, January 24, 2003, 27-35.

2. European Cooperation For Space Standardiziation, "SpaceWire protocols", ECSS-E-ST-50-11, July 2008.

3. European Cooperation For Space Standardiziation, "Remote memory access protocol", ECSS-E-50-11 Draft F, December 4, 2006.

1553 TO SPACEWIRE BRIDGE

Session: SpaceWire Components

Short Paper

Jennifer Larsen

Aeroflex Colorado Springs

4350 Centennial Blvd. Colorado Springs, CO 80907

E-mail: *Jennifer.Larsen@aeroflex.com*

ABSTRACT

The 1553 to SpaceWire Bridge allows devices, which are compliant to MIL-STD-1553[2], to access and communicate on a SpaceWire (SpW) bus. This bridge allows existing instruments to be used within a system where the main data bus has been updated to SpaceWire. MIL-STD-1553B messages are decoded and translated into ECSS-E-ST-50-12C[4] compliant messages and communicated over the SpW bus. The bridge device also translates SpW messages into 1553 messages for full duplex data transfer.

1 SPACEWIRE TO 1553 BRIDGE ARCHITECTURE

For SpaceWire to be designed into future missions a bridge from SpaceWire to 1553 and vice-versa is necessary. This bridge requires a large buffer memory to handle the 1MHz 1553 operating frequency verses the relatively high operational frequency of SpaceWire of 2 to 400Mbps as defined in SpaceWire Standard ECSS-E-ST-50-12C. The 1553 to SpW Bridge consumes a small percentage of available SpW bandwidth.

A notional block diagram is presented in Figure 1. The concept of the bridge is to allow older instruments to be integrated into a new system where the backplane has been updated to SpaceWire. The integral blocks of the 1553 to SpW Bridge include:

- SpaceWire Physical Interface
- A and B 1553 channels
- 1553 control bits
- SpW and 1553 message decode
- Required buffer memory

The bridge provides one full duplex ECSS-E-ST-50-12C compliant node interface. This node contains transmit and receive FIFOs used to buffer data being sent within the SpW network. The transmit FIFO takes translated data from the 1553 interface

and transmits it to an external node. Conversely the SpW receive FIFO accepts data from an external node and passes it to the buffer memory and message decode, then to the 1553 interface.

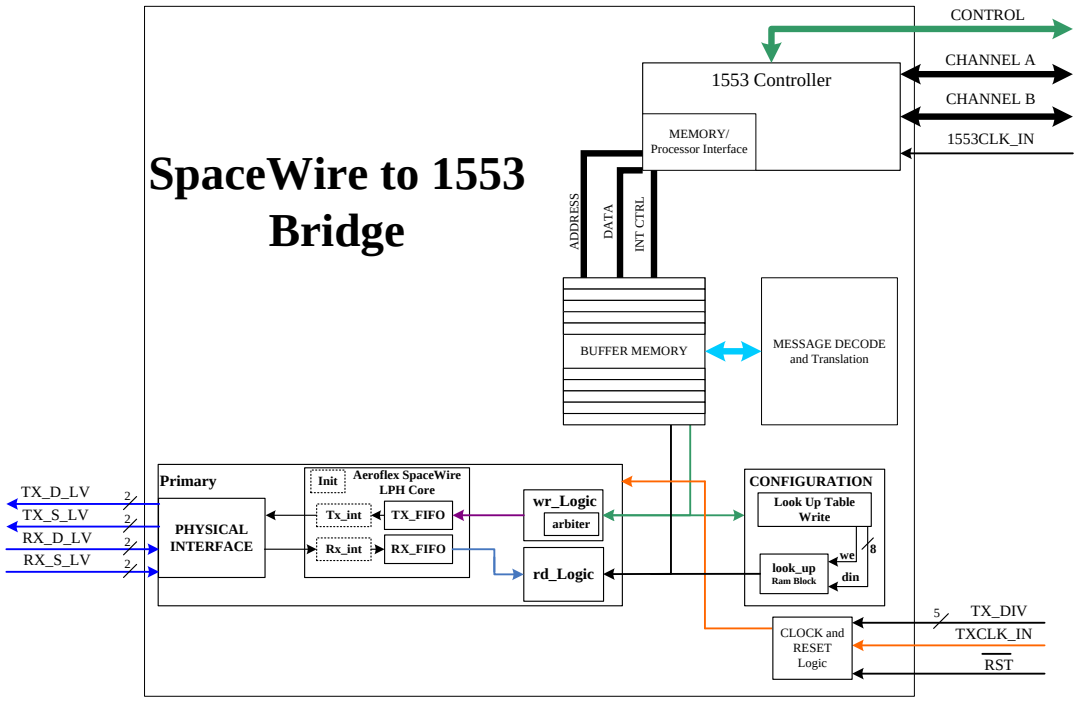


Figure 1. Notional Block Diagram

2 MESSAGE DECODE

The SpW to 1553 Bridge is capable of translating and transferring data to and from the 1553 interface as a Remote Terminal (RT) which comprises the electronics necessary to transfer data between the 1553 data bus and the external node. The Bridge also translates and transfers data from a Bus Controller (BC) which sends commands that direct the flow of data on the 1553 data bus.

There are a few key differences between 1553 and SpW data transfers, see Table 1.

Table 1. 1553 and SpW Protocol Differences

Parameter	1553	SpaceWire
Data Rate	1 MHz	up to 400Mbps
Word Length	20 bits	User Defined
Data Bits / Word	16 bits	User Defined
Message Length	Maximum of 32 data words	User Defined
Transmission Technique	Half-duplex	Full-Duplex
Protocol	Command/response	User Defined
Bus Control	Single or Multiple	Point-to-Point

The Bridge will take 1553 messages using the command and response format and transfer the 1553 messages to a RMAP ECSS-S-ST-50-52C command[5]. The information transfer formats of MIL-STD-1553 specifically the BC-RT and RT-BC

commands are mapped to RMAP Write and Read commands. The 1553 20-Bit command words contain information such as Sync, RT address, Transmit/Receive, Subaddress/Mode, Word Count, and Parity. The 20bit Status words contain Sync, RT address, error information, Service Request, Command Received, Acceptance, Terminal Flag, and Parity.

A BC-RT command coming from the MIL-STD-1553B instrument to the SpaceWire bus is decoded as illustrated in figures 2A and B below.

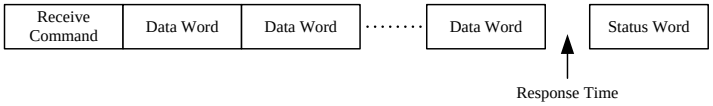


Figure 2A. MIL-STD-1553 BC-RT Information Transfer Format

	Target SpW Address	---	Target SpW Address
Target Logical Address	Protocol Identifier 0x01	Packet type, Command, Source Path Address Length	Key
Reply Address	Reply Address	Reply Address	Reply Address
Reply Address	Reply Address	Reply Address	Reply Address
Reply Address	Reply Address	Reply Address	Reply Address
Source Logical Addresses	Transaction Identifier MSB	Transaction Identifier LSB	Extended Write Address
Write Address MSB	Write Address	Write Address	Write Address LSB
Data Length MSB	Data Length	Data Length LSB	Header CRC
DATA	DATA	DATA	DATA
DATA	DATA	DATA	DATA
DATA	Data CRC	EOP	

Figure 2B. RMAP Write Command

3 EXAMPLE

Assume a 1553 instrument wanted to send the following BC-RT messages, referencing figure 3 and the Aeroflex 1553 Product Handbook. The Bridge device would ensure that the COMMAND WORD, minus the SYNC and Parity bits, are placed in the first data bit of a RMAP Write command. Depending on the overall network topology the RMAP packet will look similar to the figure 4.

Please note that the data length bytes in the RMAP Write command (Figure 2B) have been set to accommodate for the 48-bit (0x30) 1553 BC-RT information transfer.

Converting from binary 1553 messages to Hex SpW RMAP commands: (minus the SYNC and parity bits)

- Command Word: 00001000 00100010 = 0x08 0x22
- Data Word 1: 01101101 10100010 = 0x6D 0xA2
- Data Word 2: 00100001 00100000 = 0x21 0x20
- Status Word: not part of the RMAP Write packet, this bit stream will be part of a Write Reply command from the RT to the BC

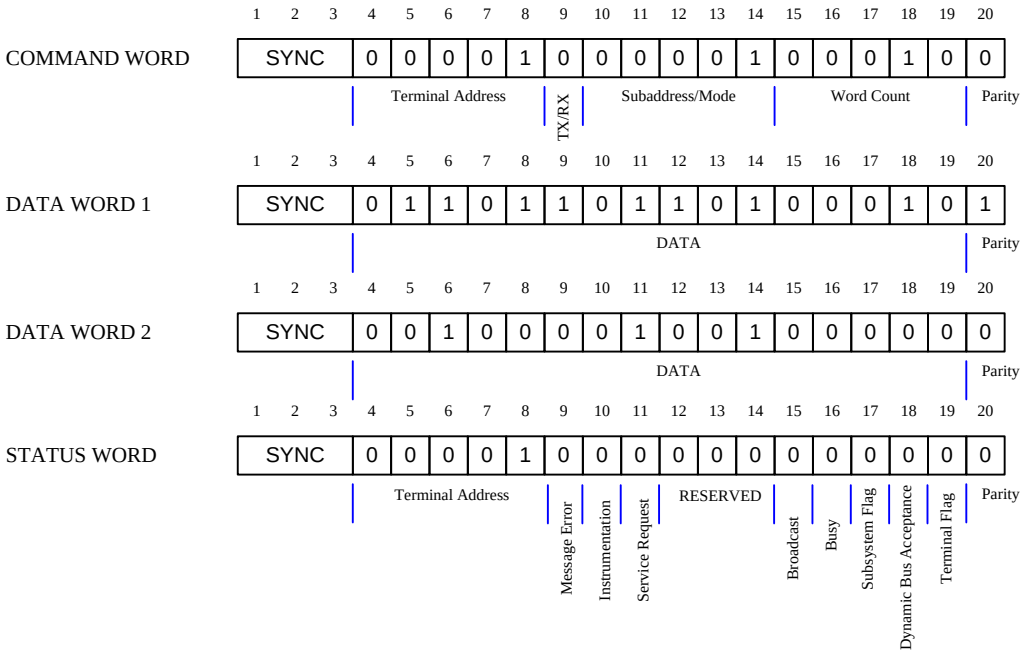


Figure 3. 1553 BC-RT

	Target SpW Address	---	Target SpW Address
Target Logical Address	Protocol Identifier 0x01	Packet type, Command, Source Path Address Length	Key
Reply Address	Reply Address	Reply Address	Reply Address
Reply Address	Reply Address	Reply Address	Reply Address
Reply Address	Reply Address	Reply Address	Reply Address
Source Logical Addresses	Transaction Identifier MSB	Transaction Identifier LSB	Extended Write Address
Write Address MSB	Write Address	Write Address	Write Address LSB
0x00	0x00	0x30	Header CRC
0x08	0x22	0x6D	0xA2
0x21	0x20	EOP	

Figure 4. SpW RMAP Write

4 CONCLUSION

The 1553[2] to SpW[4] Bridge will allow devices/instruments compliant to MIL-STD-1553 to access and communicate on a SpaceWire bus. Even with the differences between the two data bus standards, RMAP[5] commands can be used to bridge information from a SpaceWire bus to a 1553 bus. This Bridge device provides a solution that translates between 1553 and the SpaceWire busses.

5 REFERENCES

1. Aeroflex Colorado Springs, “1553 Product Handbook,” October 1992.

2. Military Standard, “Aircraft Internal Time Division Command/Response Multiplex Data Bus MIL-STD-1553B,” (Notice 2), September 1978

3. IEEE P1355, “Standard for Heterogeneous InterConnect (HIC) IEEE 1355-1995,” Conference Title, Location, June 12, 1996.

4. ESA Publications Division, “SpaceWire Standard Document ECSS-E-ST-50-12C,” The Netherlands, July 30, 2008.

5. ESA Publications Division, “Remote Memory Access Protocol (RMAP) ECSS-S-ST-50-52C,” The Netherlands, February 2010.

Wednesday 9 November

Missions and Applications 1

SOLAR PROBE PLUS AND SPACEWIRE: VIRTUAL SPACECRAFT BUS

Session: SpaceWire Missions and Applications

Long Paper

Alan A. Mick, Joseph R. Hennawy, Christopher J. Krupiarz, Horace Malcom

The Johns Hopkins University Applied Physics Laboratory

11100 Johns Hopkins Road, Laurel, MD 20143

E-mail: Alan.Mick@jhuapl.edu Joseph.Hennawy@jhuapl.edu, Christopher.Krupiarz@jhuapl.edu, Horace.Malcom@jhuapl.edu

ABSTRACT

The Solar Probe Plus (SPP) mission will explore the Sun's corona, one of the last unexplored regions of the solar system. The spacecraft will carry a complement of instruments closer to the Sun than any spacecraft has ever ventured. The mission concept calls for a minimum perihelion of 9.5 solar radii over an extended campaign of in-situ and simultaneous remote observations.

To meet the power, mass, fault management and electromagnetic interference constraints of the mission, the SPP spacecraft architecture uses SpaceWire as the primary data communication interface. SpaceWire has been widely used for payload data-handling on more than 30 space missions and includes many desirable features, such as integrated time code distribution, more than adequate through put, and a flexible network configuration that supports the Solar Probe avionics architecture. This paper describes the Solar Probe Plus SpaceWire architecture in detail with a focus on the development of a transaction protocol and schedule that meets the deterministic requirements for the Solar Probe Plus avionics control functions.

1 THE SOLAR PROBE PLUS MISSION

The Solar Probe Plus (SPP) mission targets the fundamental processes and dynamics that characterize the Sun's corona and outwardly expanding solar wind and energetic particles. Physics of the corona and inner heliosphere connect the activity of the Sun to the environment and technological infrastructure of the Earth. For more than 50 years, the questions of why the solar corona is so much hotter than the photosphere, and how the solar wind is accelerated have puzzled scientists. Remote and global observations have made dramatic discoveries of the phenomenology but still no consistent, physics-based, first-principles approach can explain coronal temperature inversion or solar wind origin. The answers to these questions can only be obtained through local, in-situ measurements of the solar wind down in the corona.

The SPP mission explores the inner region of the heliosphere in great detail through *in-situ* and remote sensing observations of the magnetic field, plasma, and accelerated particles in that region. SPP travels much closer to the Sun than any other spacecraft in order to repeatedly obtain *in-situ* and remotely sensed coronal magnetic field, plasma and energetic particle observations in the region of the Sun that generates the solar wind, between a minimum perihelion of 9.5 solar radii (Rs) and at least out through 55 Rs. The perihelion, over the solar equator, must be within the corona so that the spacecraft passes through the location where acceleration processes are theorized to occur. The direct plasma, magnetic field, and energetic particle observations from SPP will allow testing of and discrimination among the broad range of theories and models that describe the Sun's coronal magnetic field, the heating and

acceleration of the solar wind, and the generation, acceleration, and propagation of energetic particles. By making direct, *in-situ* measurements of the region where the solar wind is created and where some of the most hazardous solar energetic particles are energized, Solar Probe Plus will make fundamental contributions to our ability to characterize and forecast the dynamics of the heliosphere and its radiation environment, an environment in which future space explorers will live and work.

The primary science goal of the Solar Probe Plus mission is to determine the structure and dynamics of the Sun's coronal magnetic field, understand how the solar corona and wind are heated and accelerated, and determine what mechanisms accelerate and transport energetic particles. The SPP mission will achieve this by identifying and quantifying the basic plasma physical processes at the heart of the heliosphere. The primary SPP mission science goal defines three overarching science objectives as follows:

- Trace the flow of energy that heats and accelerates the solar corona and solar wind.
- Determine the structure and dynamics of the plasma and magnetic fields at the sources of the solar wind.
- Explore mechanisms that accelerate and transport energetic particles.

NASA selected five science investigations to achieve the answers to these long posed questions. The Fields Experiment (FIELDS) will make direct measurements of electric and magnetic fields and waves, Poynting flux, absolute plasma density and electron temperature, spacecraft floating potential and density fluctuations, and radio emissions. The Solar Wind Electrons Alphas and Protons (SWEAP) Investigation will count the most abundant particles in the solar wind -- electrons, protons and helium ions -- and measure their properties such as velocity, density, and temperature. The Integrated Science Investigation of the Sun (ISIS) makes observations of energetic electrons, protons and heavy ions that are accelerated to high energies (10s of keV to ~100 MeV) in the Sun's atmosphere and inner heliosphere, and correlates them with solar wind and coronal structures. The Wide-field Imager for Solar PRobe (WISPR) will take images of the solar corona and inner heliosphere. The telescope will also provide images of the solar wind, shocks and other structures as they approach and pass the spacecraft. This investigation complements the other instruments on the spacecraft providing direct measurements by imaging the plasma the other instruments sample. In addition to the instrument payload, NASA also selected an Observatory Scientist (OS) investigation - Heliospheric origins with Solar Probe Plus (HeliOSPP) - to address the SPP science objectives using the SPP system of measurements. The OS provides theoretical input and independent advice to maximize the scientific return from the mission.

The observational campaign consists of 24 perihelion passes inside of 35 R_S over ~7 years with gradually decreasing perihelia. The very first orbit will bring SPP closer to the Sun than any other mission has ever been. 19 perihelia will be within 20 R_S and yield 961 hours of observations. The final three perihelia will be at 9.5 R_S, within the region where the crucial acceleration processes are theorized to occur. Over the course of these observation, Solar Probe Plus will spend a total of 961 hours inside 20 R_S, 434 hours inside 15 R_S, and 30 hours inside 10 R_S. Because of the mission's timing towards the end of one solar cycle and the peak of the next, the solar wind will be sampled in all of its various modalities - slow, fast, variable, transient - as it evolves with rising solar activity toward an increasingly complex structure.

2 THE SOLAR PROBE PLUS SPACECRAFT

The Solar Probe Plus spacecraft, shown below to the left, is three-axis-stabilized. Stabilization and attitude control are effected through reaction wheels with thrusters used for momentum dumping. The most prominent feature is a large, flat, ceramic-

coated carbon-carbon shield, the Thermal Protection System (TPS), which is necessitated by the near Sun environment. At minimum perihelion the solar flux will be roughly 512 times that which is encountered in Earth orbit. The TPS protects the instruments and spacecraft from exposure to the flux. The only components that extend outside the TPS umbra during the solar encounter are the solar arrays, the FIELDS instrument's antennas and SWEAP's Faraday Cup. The solar arrays stay within the TPS's penumbra, and thus are partially protected, while the antennas and the cup are both fully exposed.

Since the solar arrays are necessarily exposed to a higher than typical solar flux during encounter, an active cooling system is required. This cooling system consists of water cooled solar array substrates and a mechanical pump loop which transfers heat from the solar arrays to radiators located under the TPS. Active cooling requires mass and consumes power, and thus the arrays must be minimally exposed, which in turn limits power.

3 SOLAR PROBE AVIONICS

The Solar Probe Plus avionics design is driven by several critical factors: low mass, low power and the need to keep the thermal protection shield pointed towards the sun, particularly during perihelion science operations. During this period solar pressure torque (due to the offset between the center of pressure and center of mass) tends to offpoint the TPS from the sun, which would expose the spacecraft to the full solar flux. Attitude control is very important and must be maintained and reestablished quickly even through a severe fault that would induce a processor reset or permanent failure.

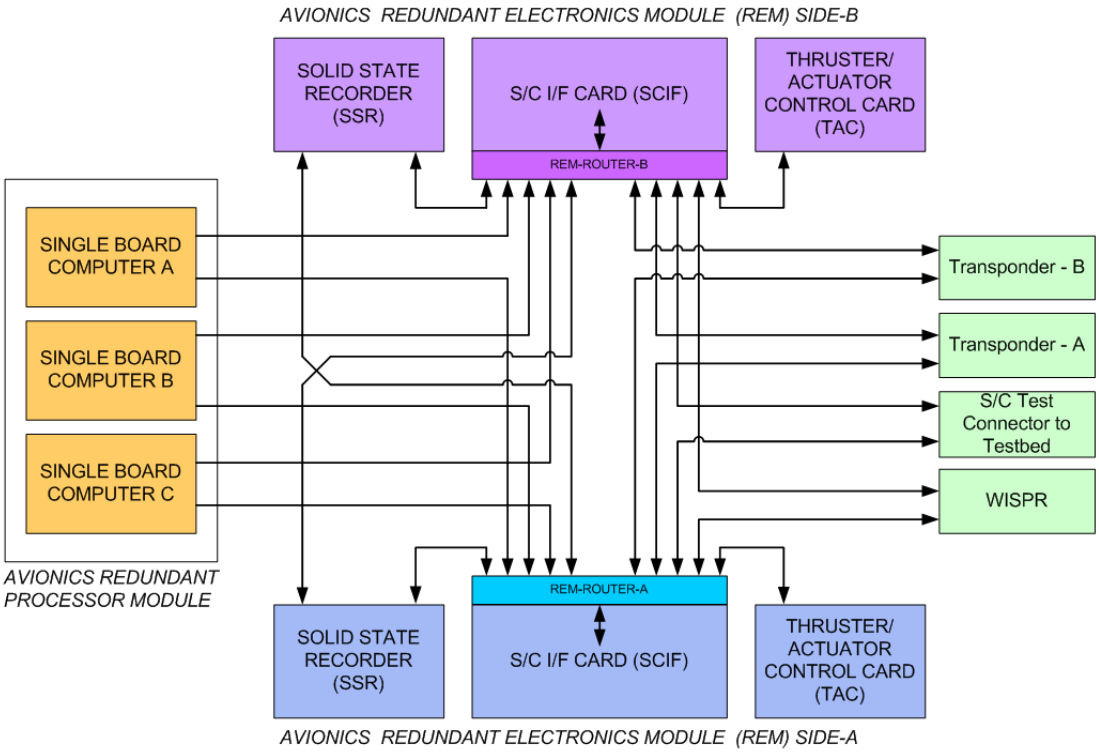


Figure 1: Solar Probe Plus Avionics and SpaceWire Network [4]

For this reason, a hot spare processor was desired and a three processor configuration was chosen: a Prime, a Hot Spare, and a Warm Spare. This configuration is shown in figure 1, labeled the Avionics Redundant Processor Module.

One of the three single board computers (SBC) serves as the primary C&DH and G&C processor (the SBC-Prime) and the other two serve as the hot and warm spares. The SBC-Prime drives one of two redundant Avionics strings consisting of a SpaceWire router, a spacecraft interface card (SCIF), a solid state recorder (SSR), and a thruster / actuator control card (TAC). The other string is normally powered off. The transponders (part of the Telecommunications subsystem) and the SSRs are cross strapped so that either one can be reached from the other side. This allows these two components to be powered on and used regardless of which redundant string is being used. The two strings are labeled Avionics Redundant Electronics Module (Side-A and Side-B) in figure 1.

Although a data interconnect such as 1553 could have been used to link the SBCs and the two strings, SpaceWire was chosen instead for two reasons. First, measurements made on previous 1553 bus implementations have detected emissions that would interfere with SPP science magnetometer measurements. Second, 1553 does not have adequate bandwidth for all data transfers; additional point-to-point serial links would have been required to off load the 1553 bus for SPP.

4 SPACEWIRE-D

Because of low tolerance for magnetic contamination SpaceWire was selected as the primary data interface for the major spacecraft components. SpaceWire has been used on more than 30 missions for payload data handling [3], and at about 4 Mbps, the over-all data throughput requirements for Solar Probe are not very demanding. A relatively low signaling rate of 20 to 30 MHz was deemed sufficient to handle this volume and desirable in order to simplify FPGA development. However, SpaceWire does not directly address the deterministic delivery of information within responsive time constraints for avionics control applications. As the principle avionics data interconnect, this became the driving requirement for the avionics and data handling network. In order to meet this requirement SpaceWire-D, a deterministic data handling protocol for SpaceWire (D for Deterministic), was considered.

In their paper *SpaceWire-D*, Parkes, et. al. provide a succinct summary of the characteristics that make deterministic data delivery over SpaceWire problematic [3]. SpaceWire networks employ an asynchronous data delivery protocol with varying packet sizes and worm-hole routing. With worm-hole routing, the leading bytes of a SpaceWire packet determines its route through the network using path or logical addressing. As a packet arrives and the addressing information is determined by the router, the packet is switched to the output port right away. The size of the packet is unknown until the end of packet is signaled at the end of the data stream. Storing and forwarding of packets is not a feature of SpaceWire routers. While this reduces the amount of buffer memory required for a router and simplifies its implementation, if an output port is already in use by another packet, the incoming packet is left distributed across the network path from the router back to the source for an indeterminate amount of time.

On the receiving end, a node may not be ready to accept the full packet and, once again, the packet may be blocked along its full path and prevent the use of network resources efficiently and deterministically. In order to ensure deterministic data delivery and throughput, SpaceWire traffic must be carefully controlled and the basic, standard SpaceWire implementation does not allow for this to be easily done.

SpaceWire-D is a proposed protocol standard for deterministic data delivery over SpaceWire being developed under the auspices of the European Space Agency. It is a higher level protocol layered over standard SpaceWire utilizing the Remote Memory Access Protocol (RMAP) and the SpaceWire time code distribution facility. Since it makes use only of standard SpaceWire elements without requiring any changes at

lower levels, it can be implemented as part of any standard SpaceWire network using any standard SpaceWire components.

To control the network interactions and prevent conflicting use of network resources, SpaceWire-D utilizes the technique of time-division multiplexing. SpaceWire's time code signaling capability is used to synchronize network interactions by establishing a 64 time-slot schedule. In the simplest case, within each slot one specific node (the initiator) is given full control of the network, and all other nodes remain passive. The initiator uses the RMAP protocol to initiate and complete one read or one write to the target. Since there are no conflicting transactions on the network, any arbitrary node may be the target.

A more complex and efficient schedule may be developed by allowing multiple initiators within a particular time slot as long as they confine interactions to a subset of targets that cannot create conflicting transactions on the network. It is also possible to allow for transactions that span more than one time slot, as long as the schedule can be accommodated without conflicts.

The SpaceWire-D protocol provides a robust, general purpose means of assuring deterministic quality of service within a standard SpaceWire network. However, in considering it for the Solar Probe Plus spacecraft, several concerns were noted. The protocol requires strict partitioning of data into uniform segments to keep each transfer within a relatively short and uniform time division. The current standard recommends a maximum of 256 bytes. This would require the development of a segmentation service over the SpaceWire-D level to partition and reassemble application data. In some cases, such as FPGA based devices that utilize "memory mapped" control registers, this may prove problematic. The recommended size of the data units would lead to a relatively short time slot duration which would result in interrupt frequencies on the order of tens of thousands per second. This was considered too burdensome. Because of these considerations, it was decided to use the basic principles underlying the SpaceWire-D protocol, but to use a less restrictive and more application specific strategy that would mitigate the concerns.

5 SOLAR PROBE PLUS ADAPTATION OF SPACEWIRE-D CONCEPTS

The basic mechanisms underlying the SpaceWire-D protocol were adapted for use on the Solar Probe Plus spacecraft in order to provide deterministic data delivery for the avionics control applications along with sufficient throughput for data handling applications. The SpaceWire time code distribution function is used to synchronize data transfers over the network and the RMAP protocol is employed to allow explicit control over the transaction sequences. Transactions are sequenced in order to prioritize the critical avionics control functions. Relatively small, uniform transaction sizes are used, but application transactions generally are not segmented. A simple schedule is used with only one initiator, which has complete control over the entire network. While some minor exceptions are allowed to these practices, adoption of them allows the spacecraft transactions to proceed deterministically and responsively without conflict.

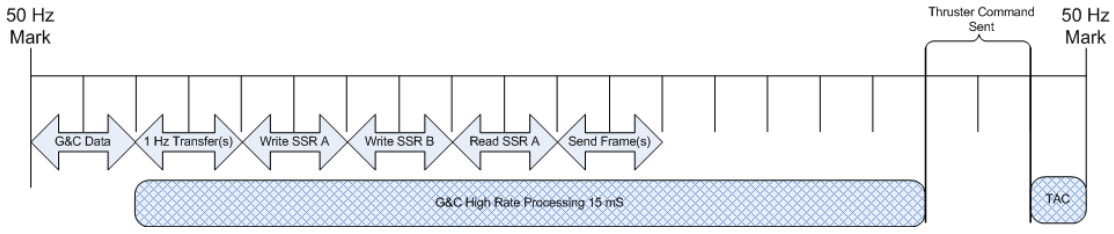


Figure 2: 50 Hz / 20 ms Control Frame Transaction Divisions

Transactions are divided into several classes: 50 Hz guidance and control, 1 Hz command and telemetry, writing to mass storage (both solid state recorders A and B), reading from mass storage (either solid state recorder A or B), and sending telemetry frames to the transponder. These transaction classes are illustrated in figure 2, which shows how they are distributed within a 50 Hz / 20 ms control frame.

Each class is roughly the equivalent of the "slots" or "channels" employed by SpaceWire-D. Each has a characteristic target or set of targets. There are, however, significant differences. The SpaceWire time codes are used to generate 50 Hz interrupts at 20 ms intervals – subdividing the 50 Hz interval is not done. This keeps the interrupt loading low. The divisions within the 50 Hz frame are maintained sequentially, but are not allocated to rigorous time divisions and do not have to conform to rigorous size constraints. Actual execution of a transaction may be optional, such as the case when there is nothing to write to the SSR. In this case, the schedule advances to the next transaction without delay – thus sending a telemetry frame to the transponder, if needed, may "replace" reading and writing to the solid state recorders when there is no need for storage operations.

The ability to flexibly advance the transaction schedule within the 50 Hz frame is possible because, unlike the more general SpaceWire-D, there is one and only one transaction initiator on the entire network, the primary single board computer (SBC-P) running the C&DH application. SpaceWire-D (when adhering to a simple schedule) assigns each slot to any one of a number of transaction initiators and the assigned initiator may then communicate with any target on the network. This allows, within the constraints of the schedule, direct transfers between any two nodes on the network. But because the SBC-P is the only initiator in the SPP configuration, all transactions must pass through the SBC-P. Direct node-to-node transactions that do not pass through the SBC-P are therefore not allowed; they must be implemented as two discrete transactions with the SBC-P serving as the intermediary. This restriction did not create any problems for SPP, since virtually all transactions pass through the SBC-P as a matter of course.

In a sense, this makes the SBC-P the "bus controller" of a "virtual spacecraft bus" implemented over SpaceWire. Only the "bus controller" needs to have explicit knowledge of the schedule. All other nodes simply respond to transactions initiated by the "bus controller". Since the "bus controller" can skip transactions that are optional under various operating modes, only one schedule is needed regardless of current activity. It also simplifies the implementation by eliminating the need to distribute the schedule across several components.

As with SpaceWire-D, the RMAP protocol is generally used for transfers over the SpaceWire network. This has several advantages. RMAP is a robust and reliable protocol that can be implemented in hardware. Several implementations are commercially available. It provides positive verification of successful transaction completion. Three basic operations are available: read, write and read-modify-write. The read and write operations allow the "bus controller" to initiate transfers in either direction. The read-modify-write operation, which is useful for bit operations on control registers, is generally not used.

The SPP network must accommodate a range of devices, both with and without processors. In order to maintain a uniform approach and implementation across these devices, a simple "common buffer transfer" method has been adopted. This consists of two attributes, a buffer size and the buffer itself. For the write operation the initiator knows the buffer size a priori and transmission of the size is intrinsic to the RMAP protocol. When needed, the size may be included as the first word in the transmitted information, so that it is available to the receiver without any modification

or special interaction with the RMAP protocol. For the read operation, the size of the information to be read is not known a priori, and thus two read operations are performed, one to obtain the size of the buffer and the other for the buffer itself. Since these conventions are fairly simple, ad hoc modifications for particular transactions are feasible, for instance, when dealing with a more complicated mass storage device or a transponder. The read and the write buffers are completely separate memory areas; thus for any one component, two buffers are necessary, one for reading and one for writing.

Four general categories of transactions have been established each consisting of one or more common buffer transfers. These are the 50 Hz G&C transactions, the 1 Hz command and telemetry transactions, the SSR read and write transactions, and the transponder telemetry frame transactions.

The guidance and control (G&C) avionics application has priority over all other applications' transactions. The G&C processing uses a 50 Hz control loop schedule which determined the SPP 20 ms processing cycle. At the beginning of each 20 ms "frame," sensor data is transferred over the space wire network and is used to determine the spacecraft attitude and correct for any deviation by issuing a command to actuate the reaction wheels or thrusters. All of the available sensor data must be collected from the SCIF component within four milliseconds at the beginning of the 20 ms control frame and the actuator commands must be received by the TAC 1 ms before the end of the frame. This leaves 15 ms available for the calculation of the actuator command. Our current best estimate is that transfer of the G&C data takes less than 2 ms and G&C processing takes 7.5 ms, which results in a very comfortable margin.

The 1 Hz transactions transfer spacecraft status (such as current mission elapsed time) and commands to specific spacecraft components and receive from each component engineering telemetry. Thus the SBC-P "bus controller" visits the read and the write buffers for each active component in each second, once to write status and commands and once to read telemetry. There may be more than one component resident on a particular SpaceWire node; in this case separate sets of transfer buffers are used for each component.

There are approximately 25 separate, active components on the spacecraft; since two 1 Hz operations are performed for each, this gives about 50 transactions, which fits well with the 50 Hz -"bus schedule" established for G&C performance. However, several of these transactions, specifically those that return science data from the instruments, are much larger than the others. In order to keep the 1 Hz transaction slot more or less uniform in duration from one frame to the next, the longer transactions have been arbitrarily segmented and the shorter transactions have been combined to roughly balance the longer segments. Thus each of the 1 Hz transaction slots may optionally contain several transactions, currently one to three per slot. As the telemetry from each component becomes more defined, this schedule will be altered to keep the 1 Hz transactions balanced across the 50 frames.

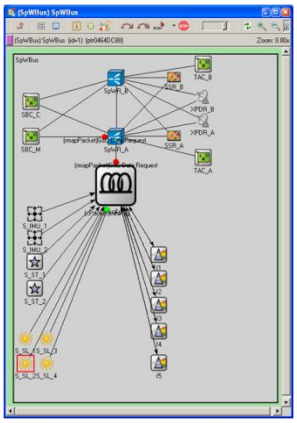
The frequency of the SSR read and write transactions is dependent on the circumstance and operational mode of the spacecraft. As science and engineering data is received by the SBC-P from the instruments and spacecraft components, it is queued for transfer to the SSR. When enough data has accumulated, it is written redundantly to both SSRs. In a like manner, during downlink a queue of telemetry frames is maintained by the SBC-P, which are sent to the transponder. As this queue is depleted, more data is read from one of the two redundant SSRs and used to format more telemetry frames for transfer. Since the rate at which science and telemetry is collected, and the rate at which data is downlinked are variable, the rate at which these

transactions occur is also variable. However, these rates never exceed the capacity provided by performing up to two SSR writes and one SSR read per 50 Hz frame.

During downlink, the SBC-P is responsible for sending telemetry frames to the transponder at a rate sufficient to keep up with the (variable) transmission rate. At the highest transmission rate (1 Mbps) this requires sending an average of a little over two telemetry frames per 50 Hz division. The schedule allows up to four telemetry frames to be sent per division. Typically, two to three frames are sent per division, at the highest rate.

Because the transponder's transmission rate may vary independently and without the knowledge of the SBC-P, the transponder sends a frame request message to the SBC-P asynchronously to the transaction schedule. This is the only exception to the rule that the SBC-P initiates all transfers. In order to keep frame requests relatively infrequent, several frames are requested at once, based on a low-water mark for the number of frames remaining to be transmitted. The low water mark is three frames remaining and the number of frames requested is four, giving a seven frame circular buffer in the transponder. At the highest transmission rate, this allows a frame request to be easily satisfied within the 50 Hz division following its receipt. The asynchronous request allows dynamically adjusted transmission rates to be easily accommodated with loose coupling between the SBC-P "bus controller" and the transponder.

6 DISCRETE EVENT MODEL



In order to verify and help design the "virtual bus schedule" for the network, a discrete event model, displayed to the left, was developed using the OmNet++ open-source simulator. A router component was created that models the characteristic SpaceWire worm-hole routing. SpaceWire and RMAP protocol overheads and scheduling inefficiencies are accounted for dynamically, with more realistic results than could be obtained through a static, analytical process. Various operational modes (e.g., data collection during encounter or downlinking science from the mass storage) may be run, and can be combined and shuffled into various "what if" scenarios. Link rates, data rates and response times may be altered to perform margin analysis and optimization. Iterative modeling helped to develop and validate the initial detailed transaction schedule, and will aid in further refining it as the Solar Probe Plus design progresses and more definitive estimates of telemetry and performance are developed. The model will be useful for designing and refining system and integration tests, and will help us understand and predict spacecraft behavior in advance.

7 CONCLUSION

The Solar Probe Plus mission utilizes SpaceWire as its principle on-board data transfer interconnect. While many missions have used SpaceWire for payload data-handling, Solar Probe Plus also uses it for its avionics applications, in which timely, deterministic data delivery is important. Using the basic SpaceWire-D concepts of time divisions based on the SpaceWire time codes and using RMAP as a transport layer protocol, a "virtual spacecraft bus" was developed that allowed the primary C&DH processor to serve as the "bus controller" and schedule transactions in order to achieve the required response times and throughput in a deterministic manner. A discrete event model that realistically simulates the protocols and the transaction schedule was used to validate and refine the concept, and will be used as the design and development work progresses.

8 REFERENCES AND WORKS CITED

[1] *Solar Probe+: Report of the Science and Technology Definition Team*, NASA/TM—2008–214161, National Aeronautics and Space Administration, Goddard Space Flight Center, Greenbelt, MD (2008). http://solarprobe.gsfc.nasa.gov/SolarProbe+Web.pdf

[2] *Solar Probe+ Mission Engineering Study Report*, NNN06AA01C, National Aeronautics and Space Administration, Heliophysics Division (2008). http://lws.gsfc.nasa.gov/documents/solar_probe/Solar_Probe+_Mission_Engineering_Study_Report.pdf

[3] S. Parkes, A. Ferrer, S. Mills and A. Mason, *SpaceWire-D: Deterministic Data Delivery with SpaceWire*, International SpaceWire Conference, St Petersburg, Russia, June 2010. http://2010.spacewire-conference.org/proceedings/Papers/Standardisation/Parkes1.pdf

[4] D. Rodriguez and R. Nichols, *Solar Probe Plus Mission Definition Review: Avionics*, October 4, 2011.

LA-UR- 11-04814

Approved for public release;
distribution is unlimited.

Title:	SpaceWire in the Joint Architecture Standard
Author(s):	Leonard Burczyk, Justin W. Enderle, Daniel Gallegos, Paul S. Graham, Richard D. Hunt, Jeffrey L. Kalb, David S. Lee, Jacob E. Leemaster, John M. Michel, and Justin L. Tripp
Intended for:	International SpaceWire Conference 2011

SPACEWIRE IN THE JOINT ARCHITECTURE STANDARD

Session: SpaceWire missions and applications

Long Paper

Leonard Burczyk[1], Justin W. Enderle[2], Daniel Gallegos[2], Paul S. Graham[1], Richard D. Hunt[2], Jeffrey L. Kalb[2], David S. Lee[2], Jacob E. Leemaster[2], John M. Michel[1], and Justin L. Tripp[1]

[1]*Los Alamos National Laboratory, Los Alamos NM 87545*

[2]*Sandia National Laboratories, Albuquerque, NM 87185*

E-mail: rdhunt@sandia.gov, jtripp@lanl.gov

ABSTRACT

The Joint Architecture Standard (JAS) is a joint project between Los Alamos National Laboratory and Sandia National Laboratories to provide a common processing and communication infrastructure upon which to more quickly develop payload sensing and processing capabilities. JAS offers a flexible, scalable, and reliable solution to space-based processing for our customer's applications. This standardized architecture is a modular design that allows for rapid prototyping and provides faster system integration and testing that reduces development and integration time and costs. The adaptable architecture meets a wide range of performance requirements including: throughput speeds; reliability; Size, Weight and Power (SWaP) reduction; and mechanical and electrical interfaces. The architecture also allows for evolving design changes while minimizing impacts to established interfaces.

The primary capability enabling technologies in JAS are packet-switched network connectivity and reconfigurable computing. The fundamental technology of packet-switched networks in JAS are serial interconnects. Because JAS has a broad range of data rate requirements and has the added challenge of providing reliable command, control and data handling in a space environment, this architecture has employed two network tiers connected using Consultative Committee for Space Data Systems (CCSDS) and European Cooperation for Space Standardization (ECSS) communication protocol standards. One of these tiers is driven by high performance gigabit-per-second class communication for high bandwidth sensors and data processing. The other tier is driven by reliable command and control that can also support moderate data transfer rates. SpaceWire is an excellent candidate and is the serial interconnect of choice for the latter tier.

BACKGROUND

The JAS hardware architecture defines several standard hardware *nodes* connected through a minimal number of serial and discrete interconnects. Each of these nodes provides a fundamental capability such that a set of them can be combined to form the basis of a payload data processing system.

The JAS node types are shown in Figure 1. The Configuration and Host Interface Node (CH) provides the interface between the payload and Host Platform. This node contains a radiation-hardened processor to allow it to reliably boot and operate when power is applied to the payload. It runs the application software used for configuring, controlling and monitoring the payload and provides the interfaces to connect the payload to the ground system. The Non-volatile Mass Storage Node (NV) contains non-volatile memory for storing applications and data. The SDRAM Mass Memory Node (SD) contains a large amount of fast and dense memory. It provides a temporary storage capability for processing nodes to manipulate payload data as well as being a communication buffer between nodes. The Reconfigurable Processing Node (RP) and Reconfigurable Sensor Interface Node (RS) contain a large reconfigurable logic device, such as the Xilinx Virtex 5, that can be configured to run hardware applications or a soft-core CPU that runs software applications. They are general purpose, high performance processing nodes intended to process payload data. The network interface node (NI) provides network connectivity for slower sensors and lower performance critical processing. The RS and NI nodes contain an I/O interface that allows the development of program-specific interface boards to connect to payload hardware devices. The intent of these nodes is to provide an interface to sensors and perform any necessary pre-processing of their data prior to passing it to other RP, CH or NI nodes for final processing and downlink.

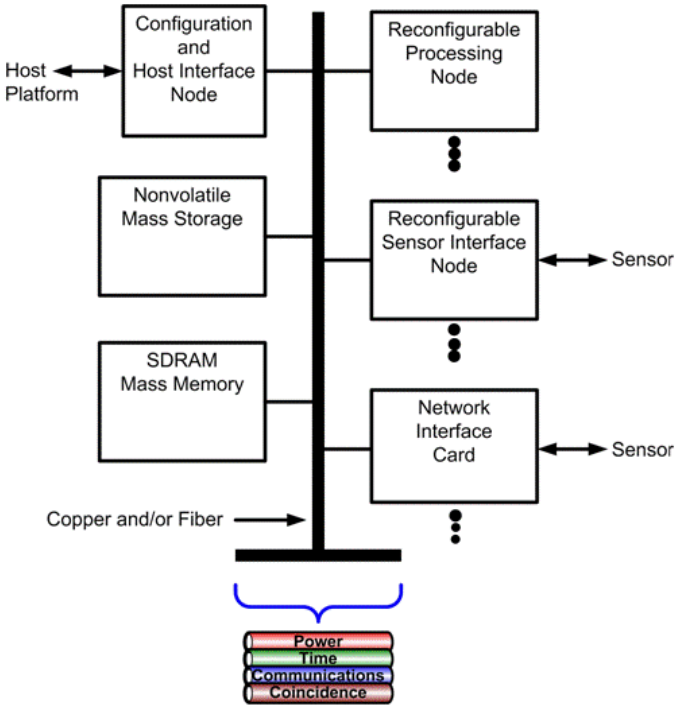


Figure 1: JAS Node Architecture

The number and type of nodes to use in a JAS based payload are determined by system requirements. Complex custom backplanes are eliminated by having a minimal number of physical interconnects between nodes which allows this node-based architecture to scale to most applications. SpaceWire can be used for routing payload command and state-of-health data as well as moderate bandwidth mission data. For high bandwidth requirements, additional networks based on Peripheral Component Interconnect Express (PCIe), Serial RapidIO (SRIO) or custom fast serial can be used.

Each JAS node contains a field-programmable gate array (FPGA) that provides a set of common functions to the node. This FPGA is referred to as the System Monitoring and Communications (SMAC) device. As shown in Figure 2, the SMAC provides a standard set of physical interfaces for communicating with devices both on and off the

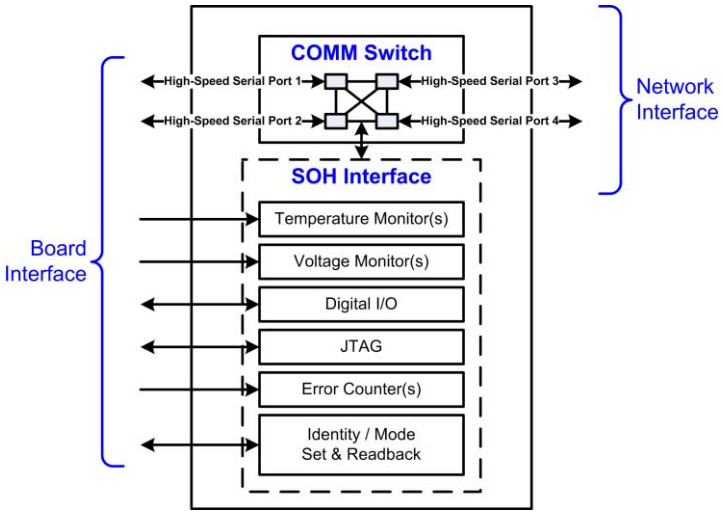


Figure 2: System Monitor and Communications Device

node. It includes a scalable SpaceWire network router with at least 5 ports and a suite of serial and parallel I/O interfaces.

The SMAC runs a suite of firmware intellectual property (IP) that provides a standard set of services. SpaceWire communication is provided by router and endpoint cores designed by NASA Goddard. A Remote Memory Access Protocol (RMAP) core provides a common interface to read and write to memory-mapped peripherals connected to the SMAC. Standardizing on RMAP as a communication protocol reduces the number of protocols that must be supported to communicate with hardware peripherals connected to JAS-based payloads. In addition to the I/O interfaces, RMAP is also used to communicate with other standard devices on JAS nodes such as Point-of-Load (POL) power converters and EEPROM storage devices containing Intelligent Platform Management Interface (IPMI)-based node identification records. This storage format defines items like node capabilities, product and firmware versions, and a unique device identifier. By accessing this information over the SpaceWire network using RMAP, the configuration host node can gather detailed information about each node using a standard protocol.

The SMAC may contain additional features as well. A SpaceWire broadcast capability can be used by any endpoint to deliver a single SpaceWire message to a variable number of other nodes in an efficient manner. This broadcast capability can be used in conjunction with SpaceWire time-codes to achieve coarse-grained time synchronization between nodes without the use of discrete signals. RMAP-accessible SelectMAP and JTAG interfaces provide remote configuration and debugging of Xilinx FPGAs over SpaceWire. This library of services will continue to grow as JAS evolves and the SMAC is a versatile and critical component in standardizing the JAS architecture.

JAS COMMUNICATION PROTOCOLS

There are a number of communication protocols being used on a JAS Spacewire network. Table 1 shows a list of these protocols and their Protocol ID (PID) values. The RMAP and RDDP

Protocol Name	***Value***
Remote Memory Access Protocol (RMAP)	*1*
Reliable Data Delivery Protocol (RDDP)	*238*
JAS Packet Protocol (JPP)	*240*
Goddard Memory Access Protocol (GMAP)	*241*
JAS RDDP (JRDDP)	*242*
Time Protocol	*243*
Broadcast	*245*
Broadcast	*246*

Table 1: SpaceWire Network Protocols

protocols are being used from the defined set of ECSS SpaceWire standard protocols [3]. The others were developed for JAS and assigned values in the user-defined range.

Software applications built on JAS will use a service oriented architecture based on the CCSDS Spacecraft Onboard Interface Services (SOIS) standard [4]. This standard specifies a layered architecture for communicating with devices and applications over serial data links. A representation of the CCSDS SOIS architecture showing the JAS data links and protocols is shown in Figure 3. Services can be implemented as hardware (FPGAs) or as software based on the needs of the application.

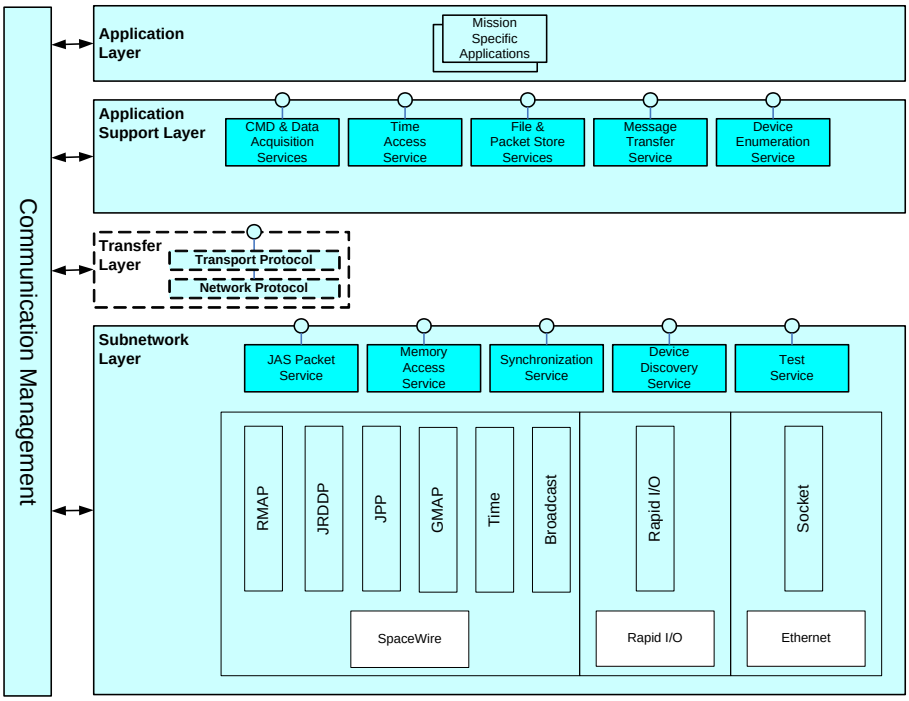


Figure 3: CCSDS SOIS with JAS Protocols

SpaceWire is used by JAS for payload command and control as well as low-to-moderate rate mission data routing. Applications will use one of three fundamental communication protocols for sending data over SpaceWire: the Remote Memory Access Protocol (RMAP); the JAS Reliable Data Delivery Protocol (JRDDP); or the JAS Packet Protocol (JPP). Other SpaceWire protocols are used for specific JAS functions such as router configuration or broadcasting time between the nodes.

RMAP is used to access remote memory based devices across a SpaceWire data link. This protocol is implements the standard managed by the ECSS committee [5]. The protocol itself supports three primary operations: read, write and read-modify-write. While this is not a reliable delivery protocol in that it will not retransmit the commands if there is an error, it does support the ability to notify the sender that the operation was successful. It has the capability of supporting write operations that both verify the data prior to writing it as well as acknowledge that the data was written.

The JAS Reliable Data Delivery Protocol (JRDDP) is a reliable packet transmission protocol used for guaranteed data delivery between two applications over a SpaceWire data link. It is based on the RDDP protocol created by NASA for the GOES-R program and has been modified to make it more flexible so it can meet the

needs of JAS payloads [6]. JRDDP consists of essentially two parts, a sender and a receiver. The sender accepts data from a user application, segments the data into smaller pieces in accordance with user-defined Maximum Transmission Unit (MTU) of SpaceWire, packetizes it for transmission, and then sends it over the data link. Transmission includes a closed-loop acknowledgement packet that is returned to the sender to confirm correct delivery of the packet to the remote application. The receiver accepts SpaceWire packets read from the network and reassembles them to create the original data message. Once reassembled, the data is delivered to the receiving application in the identical form as originally sent. If any errors occur in this transmission, timers will expire on the transmission side, and the sender will try and resend the data for a user-definable number of times.

The JAS Packet Protocol (JPP) provides the capability to send a JAS data packet over a SpaceWire data link. It is a best-effort protocol that provides little error checking and no retransmission capabilities. As such, JPP requires little processing overhead which also makes it easy to implement in hardware or for testing purposes. JPP supports sending a single JAS packet within a single SpaceWire packet. The maximum JAS packet size is 64Kbytes. Since JAS packets contain a Cyclic Redundancy Check (CRC) as part of their definition, this CRC can be used to check the integrity of the JAS packet by receiving applications. The CRC combined with a packet sequence counter, provide the tools necessary for reliable data transfer. In the future, if JAS continues to use this protocol, a segmentation capability will be added for the case that a maximum SpaceWire MTU size is enforced.

The Goddard Memory Access Protocol (GMAP) is used specifically to configure the Goddard SpaceWire routers, and the GMAP packet format expands on the SpaceWire defined packet. There are three GMAP functions: GMAP Write, GMAP Read, and GMAP Read Response. When sending a GMAP Read request to a Goddard router, the GMAP protocol inserts a variable length reply address field into the packet which the router copies byte-for-byte to the address field of the Read Response packet. The router will also insert a SpaceWire protocol ID into the response packet as well as the final return address byte. The return address and protocol ID allow the Read Response packet to be routed to the node that originated the read request and processed by GMAP protocol service. GMAP writes occur without any response from the router so there are no additional capabilities in the router to support this function.

The Time Protocol is used to send an absolute time message from one node to another within a payload. Typically, a single node will maintain the reference time and broadcast it out to all other nodes. This protocol is intended to be used along with SpaceWire timecode packets which are used as the low-latency epoch for telling the receiving nodes that the previously delivered time is valid. This protocol is also intended to be used with the broadcast protocol to enable a coarse-grained time distribution solution for payloads. Discrete hardware signals between nodes can also be used to implement a time distribution solution if more precise timing is required.

The Broadcast Protocol provides the capability for a single node to send a SpaceWire packet to any number of nodes in the system. It uses two different SpaceWire protocol IDs to accomplish this. The combination of the two packet types handles the broadcasting of the packet to all SpaceWire routers and endpoints while eliminating any duplicate deliveries.

JAS provides standard computing and data services for a wide range of sensor systems from small, simple networks to large, complex networks consisting of dozens of nodes. To support this flexibility, the network is divided into subnets at each router and a two-byte SpaceWire regional address is utilized. The first address byte delivers a packet to a specific router, and the second byte delivers the packet to a specific endpoint or application attached to that router. By knowing the topology of the network, routing tables can be established which will delete the first regional address byte from the SpaceWire packets intended for its neighbors, leaving only the application logical address when the packet reaches the intended router. This regional addressing scheme allows remote nodes to communicate with all other nodes using only standard SpaceWire routing, but without the need to assign separate logical addresses for each unique endpoint, which would quickly overrun the available logical address space.

To establish the topology and routing tables, JAS implements two options. The first option is a manual process and requires a priori knowledge of the network. Details of the topology, which includes the physical addressing paths to each router, and the individual routing tables, can be uplinked to the CH node. Using physical addressing along with either the GMAP or RMAP protocol, the CH node loads all the routing tables. The second option uses a network discovery algorithm which, by using physical addressing and polling each port of a router, establishes the topology. Then, a routing algorithm establishes the routing tables. This auto-discovery method allows for a quick way to establish the SpaceWire network as nodes are added or deleted giving the system a level of plug-and-play capability.

JAS DATA FORMATS

The JAS architecture is designed to be a collection of nodes interconnected through a peer-to-peer network topology based on SpaceWire. Transferring data across the network requires a packetized data format. A logical choice was to use a data format based on the CCSDS Space Packet Protocol Standard [1] and ECSS Packet Utilization Standard (PUS) [2]. A combination of these standards were used to create the JAS command and telemetry packet formats (JAS Packets) shown in Figure 4 and Figure 5. The structure of JAS packets are identical to those defined for a CCSDS/PUS packet with the optional fields defined or additional fields added as needed by JAS. The source and destination Application Identifiers (APIDs) are used to describe the generating and receiving entities for the packet. The Transaction_ID can be used as a sequence counter by applications that wish to maintain a list of outstanding

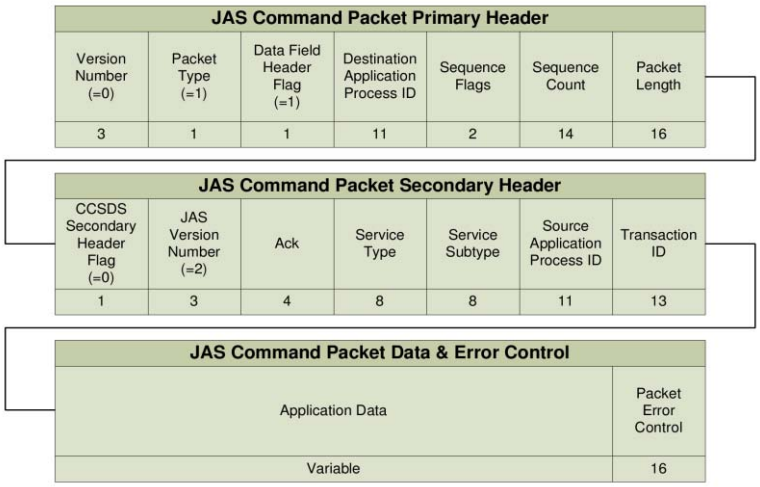


Figure 4: JAS Command Packet

command requests in which a telemetry response is expected. The last two important fields are the Service Type and Service Subtype fields. These are used to identify the packet data contents and follow the services that are described in the PUS specification. The only other modification to the standards was to replace the PUS packet version number with the JAS packet version number in the secondary header. We needed a method to track changes to the JAS packet format and we had no intentions of changing it even if there are future changes to the PUS standard.

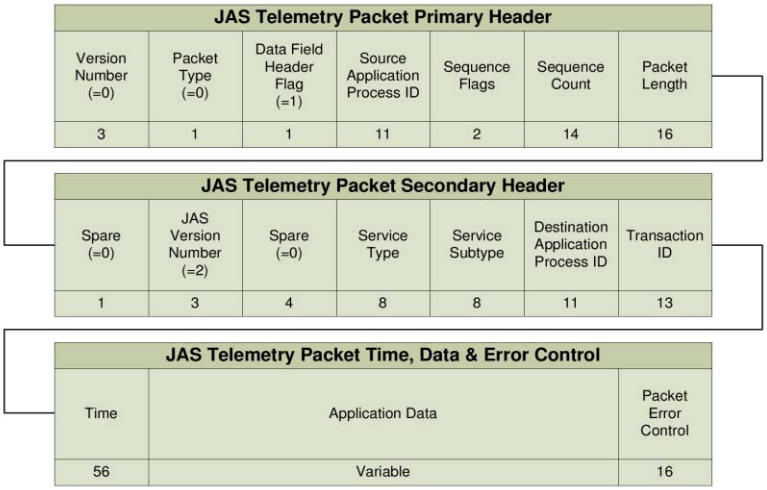


Figure 5: JAS Telemetry Packet

JAS SERVICES

Using JAS packets as the data format for communicating between applications across a SpaceWire network, a set of services were defined to identify the data contents and format. The JAS packet services are based on the PUS service concept. There are a set of standard services described within PUS and these can be used if appropriate but they are targeted primarily for communication between the payload and ground system. A set of additional on-board services is needed for communication between payload applications. Table 2 shows a subset of these additional services, defined for JAS, along with a brief description of each. The service numbers were chosen based on the user-definable range expressed within the PUS specification. Within each service, subtypes were also defined as needed. For example, Table 3 shows two service subtypes defined for the File Access Service. These subtypes correspond to a command packet for requesting the contents of a remote file system and the associated telemetry packet that contains the response.

JAS services provide a straightforward interface to develop applications and provide for a set of standard services as well as the capability to create sets of mission specific services. At the current stage of JAS development, these services are still evolving as the functional aspects of the nodes evolve. The intent is that there will be fixed set of services targeted to JAS

Service Number	***Description***
128	*Device Access Service*
129	*File Access Service*
130	*Platform Management Service*
131	*Time Management Service*
132	*Sensor X Service*
133	*Sensor Y Service*
134	*Test Service*

Table 2: JAS Packet Services

common functions, such as the file system service, and there will be a reusable set of services for program-specific applications.

File Access Service						
Service Type	***Service Subtype***	***Subtype Description***	***Cmd***	***Tlm***	***Service Parameters***	***Data Types and Description***
129	*1*	*File System Directory Listing Request*	*X*		*File_System_ID*	*File_System_ID is an unsigned integer that identifies the file system. It is assumed there is only the root directory in the file systems for JAS so a directory identifier is not required*
129	*2*	*File System Directory Listing Report*		*X*	*File_System_ID, Directory_List*	*File_System_ID is an unsigned integer that identifies the file system. Directory_List is a null-terminated string which is contains a list of each file and attributes. Each file is a record separated by a '|' (pipe) character and ends with a '\n' (newline) character. The fields and format of a single entry would look like "file_name|size|modification_time|create_time\n".*

Table 3: File Service Subtypes

CONCLUSION

To date, networks ranging from two standalone nodes, a configuration host node and a network interface node, to a network of a dozen nodes including multiple reconfigurable processing nodes in a VPX chassis have been demonstrated. Nodes from both Los Alamos and Sandia have been combined and interconnected with SpaceWire for the command and control network. SpaceWire nodes can both be directed from either simulated ground control or networked CH nodes. The rich protocol support and extensibility has made SpaceWire an excellent candidate for reliable communication and is the serial interconnect of choice for reliable command and control at moderate data transfer rates.

REFERENCES

1. The Consultative Committee for Space Data Systems (CCSDS), “Space Packet Protocol”, CCSDS 133.0-B-1, September 2003

2. European Cooperation for Space Standardization (ECSS), “Ground Systems and Operations – Telemetry and Telecommand Packet Utilisation”, ECSS-70-41A, 30 January 2003

3. European Cooperation for Space Standardization (ECSS), “SpaceWire Protocols”, ECSS-E-ST-50-11C Draft 1.3, July 2008

4. The Consultative Committee for Space Data Systems (CCSDS), “Spacecraft Onboard Interface Services”, CCSDS 850.0-G-1.1, May 2010

5. European Cooperation for Space Standardization (ECSS), “SpaceWire – Remote Memory Access Protocol”, ECSS-E-ST-50-52C, February 2010

6. Sandia National Laboratories, “Joint Architecture Standard Reliable Data Delivery Protocol”, JRDDP-001 Version D

7. Sandia National Laboratories, “Joint Architecture Standard Communication Services Specification”, JAS-CSS-00001 Version C Draft 4

IMPLEMENTATION OF THE SOCWIRE PROTOCOL (SOCP) WITHIN THE DYNAMIC RECONFIGURABLE PROCESSING MODULE

Session: SpaceWire Missions and Applications

Long Paper

Frank Bubenhagen, Holger Michel, Harald Michalik, Björn Fiethe

IDA TU Braunschweig, Hans-Sommer-Str.66, D-38106 Braunschweig, Germany

Björn Osterloh

DSI GmbH, Otto-Lilienthal-Straße 1, 28199 Bremen, Germany

Wayne Sullivan, Alex Wishart

Astrium Ltd, Gunnels Wood Road, Stevenage, Herts, UK SG1 2AS3

Jørgen Ilstad

European Space Agency, ESTEC, Keplerlaan 1, Noordwijk ZH, Netherlands

E-mail: bubenhagen@ida.ing.tu-bs.de

ABSTRACT

Future space missions require high-performance on-board processing capabilities and a high degree of flexibility. State of the art radiation tolerant SRAM-based FPGAs with large gate count provide an attractive solution for in-flight dynamic reconfigurability. With these devices an advanced System-on-Chip (SoC) can be implemented. However, the system reliability and qualification has to be guaranteed in the harsh space environment. Previous papers introduced [1] SoCWire as a fault tolerant high-speed SpaceWire based Network-on-Chip (NoC) solution, and [2] the Dynamic Reconfigurable Processing Module (DRPM), a hardware platform within which the SoCWire on-chip communication network is applied. In this paper the new SoCWire Protocol (SoCP) implementation for such a SoCWire network is presented. The protocol is inspired by RMAP, but adapted to the requirements for on-chip data processing chains and considerably simplified to limit resource consumption.

1 INTRODUCTION

Currently, the Dynamic Reconfigurable Processing Module (DRPM), a flexible processing system which provides full support for in-flight dynamic partial reconfiguration of hardware, is in development. This work is done under ESA contract [3] in collaboration with Astrium Limited. For control and monitoring tasks of such a module a program running on a standard processor is sufficient, since these tasks imply only a low amount of data and are relatively infrequent, e.g. interpret and react to telecommands, collect the module status on a regular basis, and provide a housekeeping report. With a software upload these tasks can also easily be adapted.

Quite different is the situation with the huge amounts of data generated by payload instruments. Since the resolution of sensors providing e.g. image or spectrometer data has increased rapidly over years and downlink bandwidth is still limited, this data has to be processed in real-time, which exceeds the processing capabilities of all available space qualified CPUs. Using FPGAs with dedicated logic to implement specific hardware processing cores has become common practice in the space business. Particularly SRAM based FPGAs provide, with their reconfiguration capabilities, new power and resource efficient ways to implement hardware processing cores. Of course extra measures against radiation induced single events have to be taken into account with SRAM based FPGAs, but various mitigation strategies are known to solve this for specific applications. With several processing cores in a chain a macro processing pipeline can be created.

Data frames can be passed through these blocks and the functionality of the processing blocks can be adapted during runtime. For the transmission of the data frames between the sensor interfaces, the processing blocks and the data memories, a network-on-chip infrastructure is used which effectively passes the data through the network. Such a network based approach provides a safe way to isolate a processing block, which is under reconfiguration, from the rest of the system, which may be still in operation. In several papers, e.g. [1], we have shown that SoCWire is an effective and viable solution for this problem. The pure SoCWire network enables the data transmission between several SoCWire nodes within the on-chip network providing the lower communication system layers up to packet level. In reconfigurable applications, further protocol is required to define a set of transactions for the higher level communication between the different nodes. Since every node in the network needs a handler for this protocol, its FPGA resource utilization should be minimal to leave space for the actual processing cores.

2 DYNAMIC RECONFIGURABLE PROCESSING MODULE

Primary objective of the DRPM study is to provide a development environment, which will demonstrate the feasibility of reconfigurable FPGA technology for flight programmes. Therefore, the module is primarily equipped with devices and interfaces, for which space qualified versions are available.

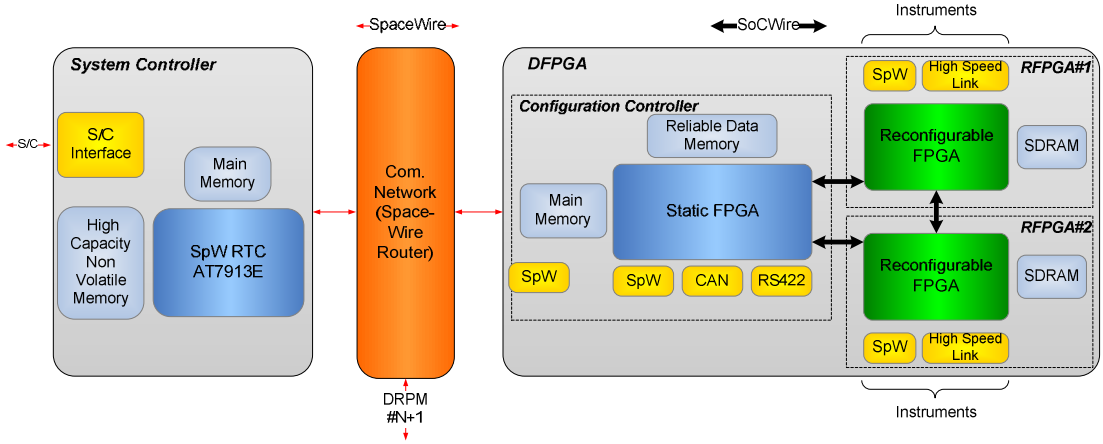


Figure 1 DRPM architecture

2.1 General DRPM Architecture

The DRPM architecture shown in Figure 1 is a modular concept and consists of at least three components: (i) System Controller, (ii) SpaceWire Router and (iii) Dynamically reconfigurable FPGA (DFPGA) module. The function of the SpaceWire router is to interconnect all sub modules and provide expandability to additional DFPGAs or additional DRPMs. With this modular concept processing capacity can simply be extended by adding further processing modules or hardware redundancy can simply be achieved by adding additional DFPGAs. The System Controller controls and supervises the overall DRPM. For these tasks it features a fault-tolerant LEON based CPU, the SpaceWire RTC ASIC (AT7913E). This CPU already incorporates SpaceWire based RMAP (Remote Memory Access Protocol) interfaces for communication with the Spacecraft. Attached to this processor is a high-capacity non-volatile memory for secure storage of all basic and partial configuration bit files required for the DFPGA(s).

2.2 DFPGA Architecture

The DFPGA (Figure 2) is the actual processing unit within the DRPM architecture. Therefore it has one or two Reconfigurable FPGAs (RFPGAs) and a Configuration Controller for overall flow control and managing of the RFPGAs' configuration. This configuration management not only has to provide the currently required configuration bitfiles, but also has to take care of SEU (Single Event Upset) accumulation within the SRAM based FPGAs and to mitigate these effects. The Configuration Controller features a LEON3 processor, several low and medium rate interfaces, interface to a large local reliable data memory, and a data and configuration interface to the RFPGAs. The data interface to each RFPGA is implemented with a 16bit SoCWire network interface.

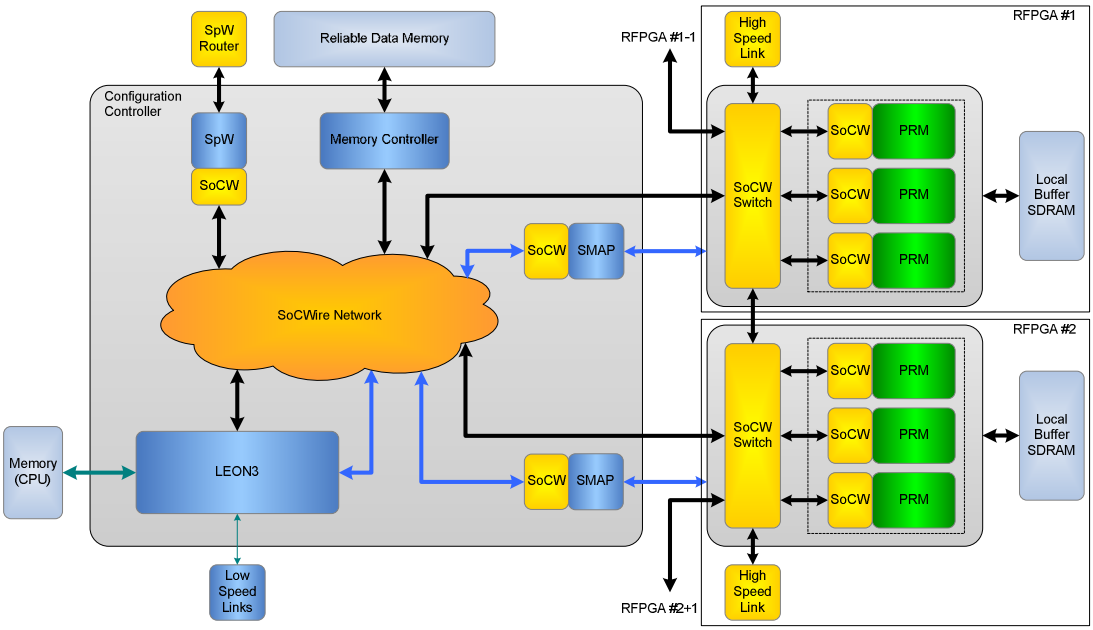


Figure 2 DFPGA architecture

The RFPGA is a module equipped with a reconfigurable Xilinx Virtex 4 FPGA device, local buffer memory, and high speed interfaces. The dedicated logic of the Virtex 4 is divided into a small static area and one or several dynamic areas, called

Partial Reconfigurable Areas (PRAs), which host the dynamically interchangeable Partial Reconfigurable Modules (PRMs). A SoCWire switch within the static area connects to the different PRMs, to the high speed interfaces and to the SoCWire interface between Configuration Controller and RFPGA.

3 SoCWire

Whereas the different subunits of the DRPM architecture are interconnected by a classical SpaceWire network, the SpaceWire based System-on-Chip Wire (SoCWire) network architecture is used on the SRAM based FPGA to interconnect several on-chip processing cores and for interfacing the RFPGAs to the Configuration Controller. The interconnection to the off-chip SpaceWire network is supported by specific SpaceWire to SoCWire bridges. SoCWire has been developed as a NoC architecture which is able to connect PRMs to a host system with the capability to isolate these PRMs logically and physically from the host system [1], [2]. Unlike ESA's SpaceWire standard [4], SoCWire uses a synchronous parallel interface instead of an asynchronous serial interface since it is intended to work in a synchronous on-chip environment. Beside the parallel data lines, there are additional lines for a parity bit, a data control flag and a valid signal. The achievable data rates in the SoCWire network depend on the application and the available FPGA resources.

A SoCWire link is always a point to point connection of two CODECs with receiver and transmitter interface. The simplest SoCWire connection consists of two nodes, see Figure 3(a). SoCWire switches with a configurable number of ports can be instantiated to create a complete network. To keep the switch small in terms of logic resources, a simple path addressing scheme and a simple round robin scheduling algorithm are implemented.

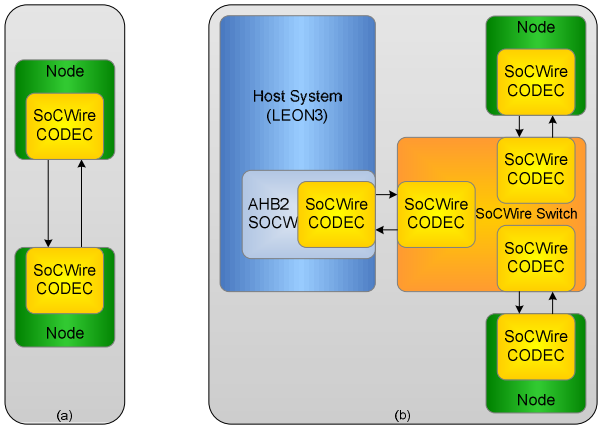


Figure 3 SoCWire network options (a) node to node, and (b) network based on switches

As pointed out in Figure 3(b), the data transfer in a SoCWire network is controlled and supervised by a host system. Typically the host system consists of a LEON processor. As a bridge between the AMBA based processor bus and the SoCWire network, the AHB SoCWire Bridge (AHB2SOCW) was developed [5]. To provide high data rates with low processor involvement, the AHB master of the bridge is controlled by two Direct Memory Access (DMA) engines. The bridge supports 16bit and 32bit SoCWire networks, whereas two 16bit words are combined into one 32bit word to support the LEON3 native AMBA data width and achieve the highest performance in combination with the DMA controller.

4 SOCWIRE PROTOCOL (SOCP)

Interconnected SoCWire CODECs represent the physical link within a SoCWire network with protocols defined up to the packet level. With this level data transmission between two nodes is possible, but a node also has to know how to interpret the meaning of the received data packets. For our network application a more generic support by typical transactions is needed, which are defined in SoCP.

4.1 REQUIREMENTS

The typical SoCWire network is shown in Figure 3(b). A processor driven host system controls and manages the data transfer between the nodes and itself. In a scientific instrument, for example, one node would be an interface to an image sensor and the second node would be a processing module providing e.g. some filter functionality, implemented in dedicated hardware. The image sensor has to know where in the network the filter is located and that it is allowed to send the acquired sensor data to the filter module. After the specific data processing by the filter module, it will send the processed data to the host processor, which will finally buffer and format the data for transmission via the spacecraft interface to ground.

Since the specific processing nodes are hardware implementations, the protocol handler has to be implemented in hardware as well. Furthermore, since every node requires its own protocol handler instantiation, also the logic resources for it are required multiple times. Therefore, the hardware protocol handler's resource usage should be as minimal as possible to leave space for the actual PRMs.

To support this requirement the following limitations and constraints apply to the SoCP implementation: (i) the number of switches in a network path is limited to three, (ii) port 0 of a switch must be the route to the host system, (iii) data width of the SoCWire network is either 16 or 32bit and (iv) the maximum number of ports in a SoCWire switch is limited to 16.

4.2 IMPLEMENTATION

The SoCP handler is placed between SoCWire CODEC and processing core. These three units form a single node in a SoCWire network and are realized within the DRPM context as PRMs (Figure 4). SoCP has to reply to requests, supply the processing core with data, and provide registers to set and read parameters from the processing core. The command and reply format is inspired by the Remote Memory Access Protocol (RMAP) for SpaceWire networks [6]. Like RMAP, SoCP is used to configure a network and to control and supervise nodes. But whereas RMAP is also used for remote memory accesses to SpaceWire nodes, the processing nodes within a SoCWire network typically process the data on consecutive data blocks. One processed data block is in the context of a macro pipeline directly transmitted to subsequent processing nodes.

The hardware implementation of the SoCP also supports a limited number of configurable user registers. With these registers it is possible to adjust parameters of the processing core during runtime, which avoids the need of reconfiguring the PRM for each change of a parameter. Additionally the processing core can provide some status information, which the processor could query.

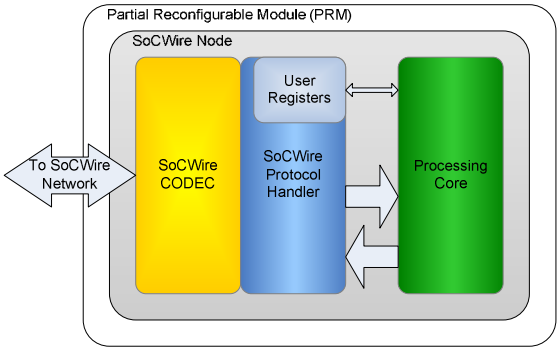


Figure 4 PRM/Node in a SoCWire network

Table 1 shows the transaction types defined in SoCP. *Process and Reply* packets are created and consumed by the host system. A packet is sent to a node, processed by it and sent back to the processor. A variant of this transaction type is that there may be nodes that are bridges to external interfaces, e.g. SpaceWire or SelectMAP Xilinx configuration interfaces. The data to be transmitted is then consumed by the node's sending process, but a reply is given to indicate the processor that the transmission was successful. The SocP does not support autonomous error handling, so e.g. an retry mechanism for missing packets would have to be implemented in the software protocol handler running on the host system.

Transaction Type	**Description**
Process and Reply	Packet generated by host system, processed by a node with reply sent back to host system
Write Register and Reply	Write a register (Reply is sent by node)
Read Register and Reply	Read a register (Reply is sent by node)
Streaming Data Transmission	Streaming data from a source to a destination node
Plug and Play Init Message	Initialization message sent by a node

Table 1 SoCP transaction types

The transactions *Write Register* and *Read Register* are intended to support the processing core with parameters or allow the processing core to provide some status information. Each core has some fixed registers, up to four optional read, and up to four optional write user registers. The width of the registers depends on the configured SoCWire width, i.e. 16 or 32bit. *Streaming Data Transmission* is the transaction for use cases as already described in the example above. A data packet from a node is directly passed to another processing node. Since the source node needs to know where it has to send the data, i.e. where the destination node in the SoCWire network is located, a set of 3 forward addresses is stored within the SoCP core fixed register set. These registers have to be set up by the host system in advance, since the host system is in charge of the current configuration status of the required PRMs and their location within the network. After the destination node has processed the payload data, it will forward the data further to the addresses stored in its register set. The transaction type *Plug and Play Init Message* is intended to inform the host system of a successful reconfiguration of a PRM. Once the new PRM starts running and the SoCWire connection is established the SoCP core generates a message which will be sent via pre-defined routing to the host system.

The two main packet definitions for the transactions are depicted in Figure 5. The packets start with up to three addresses. These are the SoCWire path addresses which

define the route of the packet from source to destination node. Each time the packet passes a SoCWire switch on its way through the network, the heading address is deleted. Packets reaching the target node start with the *Hardware ID* of the node. Therefore, the number of address fields required for a request packet depends on the number of switches en route. Since the host system is always accessible through port 0, all reply packets to the host system must have three address fields filled with zero regardless of the number of switches they have to pass. The SoCP handler running on the host system's processor (in software) has to remove potential zero words. The *Hardware ID* is unique for every node and has to be set up by a generic value during the instantiation of the IP core. The host system then inserts the *Hardware ID* of the target node, which processes the data only when it has verified the ID successfully. The target's reply packets will contain the same ID, so that the software protocol handler knows from which node the reply packet comes from. In case of streaming data packets, the source node of the packet stream has to send the destination *Hardware ID*, which needs to be set up in advance by the host system.

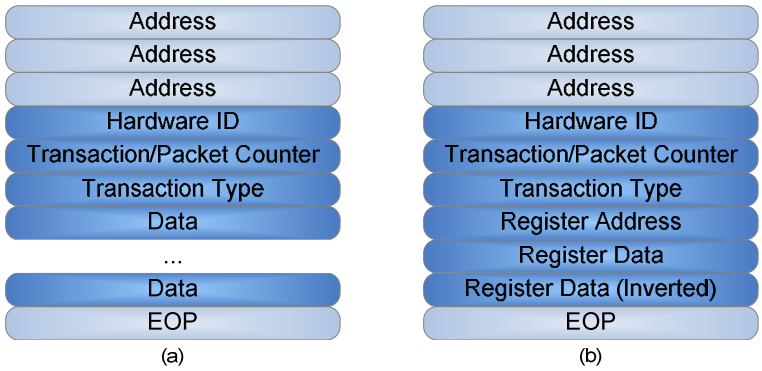


Figure 5 Packet definitions for Process Request/Reply and Stream (a), and for Register Read/Write Request/Reply (b)

The *Transaction/Packet Counter* is a sequence counter incremented with every packet sent by the host system or by the source node of a Streaming packet. The target node will use the same value within the Reply packet, so that the host system can finally identify to which Request the Reply belongs to or whether a sequence of Streaming packets is correct. The *Transaction Type* identifies the type of packet (see Table 1). With a reply also some error conditions will be reported to the host system within this field. Compared to the RMAP terminology the *Transaction/Packet Counter* corresponds to the Transaction ID and the *Transaction Type* corresponds to the Instruction ID [6]. After the *Transaction Type* either up to 2064 data bytes are sent in case of Process and Streaming packets (see Figure 5a), or a *Register Address* followed by the *Register Data* is sent in case of Register packets (see Figure 5b). Finally, every packet is terminated by an *EOP* token, complying to SpaceWire.

Error detection on protocol level is realized via parity bits in the *Hardware ID*, *Transaction/Packet Counter*, *Transaction Type* and *Register Address* field. Within each of these fields bit 15 is the parity bit. The *Register Data* is protected by an inverted copy of the register value to be read or written. Routing errors are detected by verifying the *Hardware ID*. Only when the *Hardware ID* of the target node is identical with ID in the packet, the data will be processed. Process and Streaming data cannot be protected by the SoCP core, since this would require large buffers within the core, which contradicts the low resource usage requirement. If error detection is required, then the processing core itself has to take care of this.

4.3 RESOURCES

Since the SoCP hardware handler is implemented for every node in a SoCWire network, it is important that its resource consumption is minimal. Table 2 shows the amount of resources used by the current SoCP implementation on different space grade FPGA devices. For comparison the required resources for a SoCWire CODEC IP core are also listed in this table. Both cores were configured to 16bit data width and the SoCP IP core's optional user registers were disabled. These figures represent the device utilization without TMR (Triple Modular Redundancy) applied.

	SoCP IP Core		**SoCWire CODEC IP core wo/ RAM**	
Device	**Cells/Slices**	**Utilization[%]**	**Cells/Slices**	**Utilization[%]**
XQR4VSX55	115	0.47	272	1.11
XQR4VLX200	115	0.13	272	0.31
XQ5VFX130T	67	0.33	160	0.78
RTAX2000S/SL	269	0.83(*)	754	2.34(*)
RT3PE3000L	368	0.49	932	1.24
(*) All flip-flops employ TMR				

Table 2 SoCP FPGA resource utilization

5 CONCLUSION

With the DRPM architecture an effective and viable implementation of a reconfigurable hardware for future space mission has been presented. The SpaceWire based on-chip communication architecture SoCWire provides a fault-tolerant, high speed infrastructure for the exchange of data packets between processing nodes, interfaces and host system. With SoCP an efficient protocol implementation, tailored to the specific needs of a SoCWire network has been introduced and it has been shown, that the resource utilization of the SoCP IP core is very small.

6 REFERENCES

1. B. Osterloh, H.Michalik, B. Fiethe, "SoCWire: A SpaceWire inspired fault tolerant Network-on-Chip for Reconfigurable System-on-Chip Designs in Space Applications", ISC, Nara, Japan, 2008

2. F. Bubenhagen, Björn Fiethe, Harald Michalik, Björn Osterloh, "Enhanced Dynamic Reconfigurable Processing Module for Future Space Applications", ISC, St. Petersburg, Russia, 2010

3. ESA, "FPGA bases generic module and dynamic reconfigurator", TEC-EDP/2008.30/JI, Issue: 1 Rev.1, Noordwijk, Netherland, 2008

4. ESA-ESTEC, "Space Engineering: SpaceWire-Links, nodes, routers, and networks", ECSS-E-50-12A, Noordwijk, Netherlands, January 2003

5. H. Michel, F. Bubenhagen, B. Fiethe, H. Michalik, "AMBA to SoCWire Network on Chip Bridge as a Backbone for Dynamic Reconfigurable Processing Unit", AHS, San Diego, California, USA, 2011

6. ESA-ESTEC, "Space Engineering: SpaceWire - Remote memory access protocol", ECSS-E-ST50-52C, Noordwijk, Netherland, February 2010

Networks and Protocols 3

SPACEWIRE NETWORK SIMULATION OF SYSTEM TIME PRECISION

Session: Networks and Protocols

Long Paper

Brian Van Leeuwen, John Eldridge, Jacob Leemaster

*Sandia National Laboratories***

Albuquerque, USA

E-mail: bpvanle@sandia.gov, jmeldri@sandia.gov , jeleema@sandia.gov

ABSTRACT

Many applications sharing a SpaceWire network require synchronized system time and SpaceWire can be employed to distribute system time. However, in its current form general system time distribution capability is lacking. In this paper we present a system time distribution approach that employs a broadcast extension to the SpaceWire protocol. Broadcast messages distribute specific time values while SpaceWire Time-Codes clock-in or trigger the specific time contained within the broadcast. The broadcast approach is effective in minimizing network resource usage by distributing the broadcast-time message in a partial-parallel method. Additionally, for the objective of identifying the timing precision and jitter for specific network architectures and network states, high-fidelity models were developed to quantify the timing variations and to analyze overall SpaceWire networked system performance.

1. INTRODUCTION

The European Space Agency (ESA) in collaboration with other international space agencies supports a serial data link standard to enable the transfer of large amounts of data onboard satellites. The standard named SpaceWire and defined in [1], is a satellite communication network based in part on the IEEE 1355 standard of communications. A SpaceWire network is typically comprised of a number of links, nodes and routers. SpaceWire routers are necessary since a SpaceWire node will only support a few links and thus can only be directly connected to a limited number of nodes. Routers also reduce the number of point-to-point links and enable redundant paths in case of link failures. The current standard describes a mechanism that can enable modern satellite systems to transfer large amounts of data on board the satellite. However, the standard currently lacks a time distribution capability to enable time synchronization among the various applications on the SpaceWire network [1].

Additionally, a high-fidelity modeling and simulation capability to perform analysis of SpaceWire networked systems is lacking. This analysis capability should provide precise time

** Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

synchronization results under various proposed network architectures. This analysis capability should also provide results as the architecture under study dynamically changes when faults occur and redundant paths are utilized. To meet these analysis objectives we created high-fidelity model representations of SpaceWire nodes, links, and routers that can be configured to represent any proposed network architecture.

In this paper, we present a time distribution mechanism that can be implemented in a SpaceWire network that employs the standard SpaceWire Time-Code function along with a custom SpaceWire broadcast capability. Together, the Time-Code function and broadcast capability enable a means to distribute time to the various applications utilizing the SpaceWire network.

Our approach required that the general broadcast extension be a layer upon the existing SpaceWire standard. That is it would be compatible with the existing protocol and not necessitate the modification of existing intellectual property or the revision of the existing SpaceWire standard. Rather, the objective was to extend the standard to include the new capability. Our approach met this objective.

2. System Time Distribution with SpaceWire Broadcast Extension

The current SpaceWire standard lacks both a general time distribution function and broadcast function. Multiple solutions have been proposed each with their own benefits and deficiencies [2][4]. To this end, we designed a time distribution function that utilizes a SpaceWire broadcast capability. The time distribution function is designed to work in concert, be network efficient, and fully backward compatible with the current SpaceWire standard.

2.1 System Time Distribution

The time distribution function distributes what we consider to be *system* time. In our time distribution mechanism, *system* time refers to distributing actual time versus the SpaceWire standard Time-Code. The standard SpaceWire Time-Code comprises the SpaceWire ESC character followed by an eight-bit data character. The data character contains 6-bits of system time and two control flags. A time-master node asserts a periodic “tick” and immediately sends out a Time-Code with the 6-bit time field incremented prior to transmission [3]. This Time-Code mechanism is limited to a 6-bit resolution and increments each network device’s internal time counter from the current Time-Code value to the next. The counter, which is intended to prevent looping retransmission of the Time-Code and not necessarily to carry a time value, rolls from its maximum value of 63 to zero because of its 6-bit field size limit.

Our time distribution approach accomplishes synchronization of system time. This is done by the time-master node sending a system wide broadcast containing what the system time will be at the next Time-Code “tick.” The broadcast message is transmitted a predetermined time period prior to the transmission of the Time-Code. The predetermined time period is an estimated value that is equal to the worst-case time for the broadcast message to propagate throughout the network.

The time-master node transmits a Time-Code tick indicating to the network that the time described in the previous time message is now current. Thus, the various network applications have access to an unambiguous system time.

Unambiguous system time is a 32-bit integer representation and it is broadcast to all nodes in the network. When the endpoints receive a SpaceWire Time-Code the broadcasted system time message is accepted as the current time after having been validated by combinational logic. The time endpoint evaluates whether it is synchronized with the rest of the SpaceWire network with every received SpaceWire Time-Code "tick." If the endpoint believes itself to be synchronized with the rest of the endpoints in the SpaceWire network, it considers itself to be "locked" and asserts a corresponding signal.

The endpoint determines if it is "locked" in the following way: After every "tick," the expected value of the next system time message is calculated. The calculated value is considered to be the value of the current system time message plus one. If the next received system time message matches the expected value, the endpoint concludes that it is synchronized with the rest of the network.

If the next received time message does not match the expected value, or no system time message is received by the next "tick," then the endpoint assumes that a synchronization error has occurred, indicates that it's no longer "locked," and will simply increment its system time as a "best guess." If the time message arrives late, it will not interfere with operation so long as the subsequent time message arrives on time, as the new message will override the late message.

In our approach it takes two correct time message/tick pairs to achieve a synchronization "lock." It is a known issue that if a series of two or more time messages are consistently late by a tick period (or a consistent multiple of the tick period), then the timekeeper will erroneously indicate a lock and synchronize to the late packets as they appear identical to a correct time message/tick sequence. Expanding the number of previous packets considered when calculating the expected time value would reduce the likelihood of such a situation at the cost of increasing the number of correct packets it takes to achieve a "lock."

2.2 BROADCAST

Our time distribution function employs a hybrid broadcast approach derived from work described in [4]. The approach was modified with the goal of distributing system time, be compatible with existing SpaceWire hardware, and not suffer from loops or broadcast storms. The approach creates a "broadcast server" to be hosted by each router in the network. Our implementation of this approach has two main configurable aspects: which local ports will receive broadcasts and a list of the logical addresses of all *other* broadcast servers in the network.

A packet intended for broadcast is transmitted to the local "broadcast server," which then forwards the packet to all other broadcast servers in the network. Once this is completed every

broadcast server in the network will forward the packet to the appropriate local ports on its respective router. The broadcast servers use several techniques at the protocol level to guarantee that no loops, infinite broadcast storms, or spurious re-broadcasts occur. The broadcast mechanism used for our SpaceWire Broadcast Server (SpWBS) includes several stages:

Local-to-Server Stage - A SpWBS receives a Local-to-Server type packet containing the broadcast message

Server-to-Server Stage - The initiating SpWBS sends a Server-to-Server type packet containing the broadcast message to every other enabled SpWBS in the network.

Server-to-Local Stage - Once a SpWBS receives a Server-to-Server stage packet or the initiating SpWBS finishes the Server-to-Server stage transmission it sends Local-to-Server type packets with the broadcast message to every enabled and connected local port.

This broadcast approach has efficiencies in that it partially distributes bandwidth utilization across the network and obtains parallelization of the Server-to-Local stage of broadcast. The approach requires that every router with nodes receiving broadcast messages have an attached SpWBS and it requires an additional header byte to distinguish between Local-to-Server, Server-to-Local, and Server-to-Server type messages.

The SpWBS broadcast approach includes mechanisms to prevents broadcast storms. All local ports and broadcast server addresses are disabled by default and must be explicitly enabled by server configuration. A configuration error that results in a Server-to-Local packet to be received by another SpWBS will be detected by identification of an incorrect header byte and prevented from further broadcast.

3. MODEL DEVELOPMENT

Our SpaceWire model development is done in the OPNET Modeler network simulation environment [5]. OPNET Modeler includes an extensive model library of network devices; however, OPNET Modeler does not include SpaceWire models in its standard model library. Fortunately, OPNET Modeler includes the capability for users to develop nodes based on custom protocols. To analyze the performance of our time distribution mechanism detailed SpaceWire models are developed.

The models include many features of the SpaceWire standard including the functionality at the various communication stack levels in both the end nodes and wormhole router. The models faithfully implement the disassembly of application layer data and the reassembly of the resulting NChars at the destination node. Additionally, processes such as the startup sequence, flow control, Time-Code process, and realistic representation of various buffering and queuing functions.

A modeling objective was to have representative models of the various SpaceWire modules and protocols to support system design activities in all phases of a project. This modeling would range from custom protocol extension analysis to assessing SpaceWire architectures and their operation under stressful scenario conditions resulting from link and node failures. In pursuit of this objective, we developed models to be modular. The modular approach enables the combination of end nodes and routers in various architectures. Figure 1 illustrates an example SpaceWire network and one of the nodes in the example network. In this example, each node is comprised of three specific modules; an application node, a wormhole router, and a broadcast server. The SpaceWire router is the connection point that combines the various applications nodes and broadcast servers.

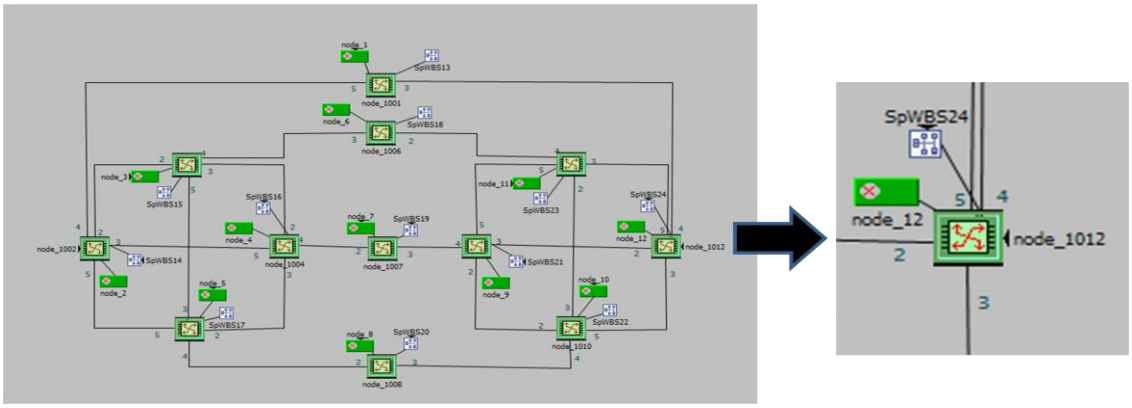


Figure 1: Example SpaceWire network topology.

Figure 2 illustrates a description of the custom SpaceWire node model and a single process model as developed in OPNET Modeler. On the left side of Figure 2 is the SpaceWire application node model that includes the protocols used in each layer of the SpaceWire communication stack. The node model includes various application types that access the network. Specifically, a state-of-health (SOH) application that periodically shares state of health (SOH) details. A standard application layer can be a data producer, such as a sensor, or a data consumer, such as a telemetry downlink, or a broadcast application that creates messages intended for broadcast.

Each of the square blocks in a node model represent a single or multiple process model state machines and implements the protocol of interest. Figure 2 (right side) illustrates an example process model. The process model illustrated in Figure 2 is a root process that can spawn child processes. Child processes are particularly applicable in modeling the wormhole router. The root process spawns a child process for each data flow through the wormhole router. In many cases multiple child process are in various states as they represent multiple simultaneous data flows through the router.

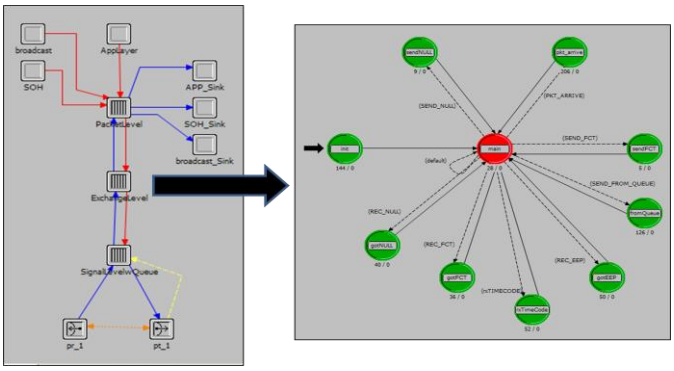


Figure 2: SpaceWire application node model (left) and an example process model (right).

OPNET Modeler includes rich mechanisms to create network traffic. In our time synchronization analysis, we are able to clearly identify when messages supporting system time distribution are created and when they arrive at their intended target. Additional application layer traffic can be generated to represent actual data files being transported through the network as NChars. Thus, Time-Code traffic is created and introduced into the network along with typical application layer traffic and its impact on delaying Time-Code messages.

4. System Time Distribution Precision Analysis with High-Fidelity SpaceWire Model

To demonstrate our time synchronization analysis capability we created a SpaceWire network comprised of 12 nodes, routers, and broadcast servers as shown in Figure 1. The architecture, constructed in OPNET Modeler, uses the various custom nodes and process modules. In this demonstration case Node 10 is considered the time master and thus originates both the Time-Codes and the system-time broadcast messages. Employing our time distribution mechanism, Node 10 will create a broadcast message immediately following a Time-Code transmission. The broadcast message will be transmitted to the broadcast server associated with the router shared by the broadcast server and Node 10 (i.e., Node 1010). This broadcast message contains the time that the next transmitted Time-Code will clock into the various network slave nodes. Since Time-Codes are not delayed by full application layer file transfers the broadcast will not arrive at a slave node prior to the previously sent Time-Code. However, there is no guarantee that the broadcast message will arrive at the slave nodes prior to the arrival of the following Time-Code transmission. In cases, where the following Time-Code arrives at the slave node prior to the broadcast time message the system is said to have lost synchronization "lock." We examine the network in Figure 1 for time synchronization precision.

5. Results and Discussion

The network in Figure 1 with Node 10 producing both the Time-Codes and the broadcast messages is assessed for time distribution delay variation. In this analysis, we record the receipt of a broadcast message and the time the broadcast message time value is clocked into the slave

node's clock. The node's time is then compared with a global absolute time. The difference of the absolute time and node clock time is recorded and plotted in Figure 3 as a probability density function (PDF).

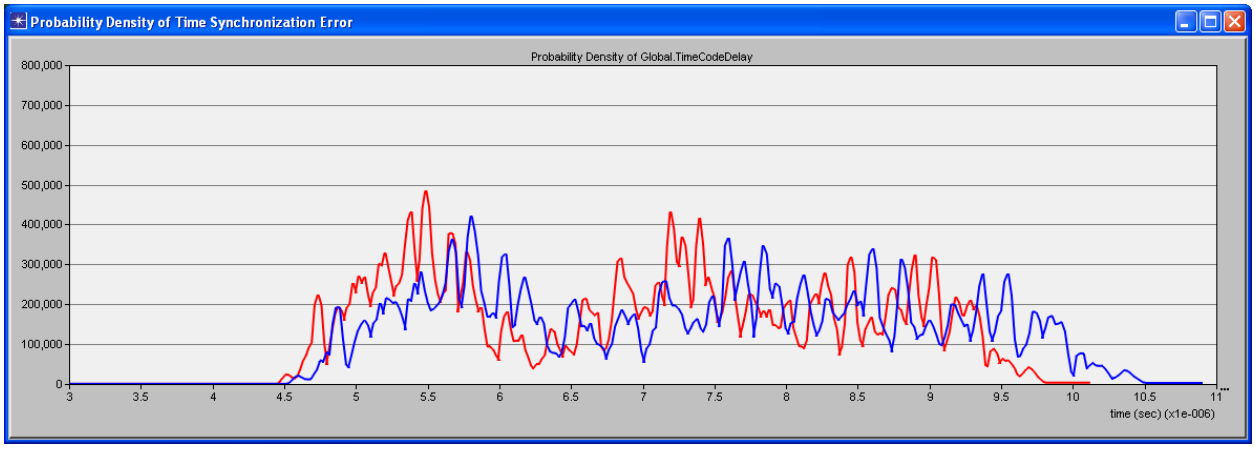


Figure 3: Resulting PDF of the time synchronization error when network is lightly loaded (red trace) and heavily loaded (blue trace). Note the Y-axis should normalized by dividing by 50E6.

Figure 3 describes a time synchronization error averaging approximately 7.0 µsec. The plot has three regions centered at approximately 5.5 µsec, 7.3 µsec, and 9.0 µsec. Each region describes the variation in time synchronization based on the number of hops to forward the Time-Code. Each additional hop adds more variation and thus leads to more spreading of the plot as you move from left to right on the time axis. The variation between the lightly loaded network (red trace) and the heavily loaded network (blue trace) results from additional NChars on the network that may delay the transmission of a Time-Code. The variation is not significant since a NULL can cause a delay of up to an 8-bit transmission time whereas an NChar can cause a delay of up to a ten-bit transmission time. In the demonstration network, the SpaceWire links operate at 10 Mbps. Also note that the Time-Code period was 6 msec. and maximum application-layer file size was less than 60 Kbits and thus were easily within range so the network would not loose time synchronization lock. We elaborate on synchronization lock in Section 6.

6. Future Work and Conclusions

Our approach's time synchronization resolution is limited by the frequency of Time-Code transmissions. The frequency of Time-Code transmissions is limited by the requirement of sufficient time for the broadcast system time message to propagate throughout the network. We believe it is possible to decouple the need for a one-to-one correlation of system time messages and Time-Code transmissions to obtain an improved synchronization error. However, the theoretical upper limit of system time synchronization precision is limited by the latency and jitter inherent in SpaceWire Time-Code function. Time-Code enhancement techniques [7] could be incorporated into our time distribution approach to improve time synchronization.

Additional features will be incorporated into the OPNET Models to expand the representation of the SpaceWire protocol and the nodes. Specifically a model of Remote Memory Access Protocol (RMAP) for SpaceWire will be developed. RMAP provides a standard method of reading and writing to registers and memory across a SpaceWire network. This will further our analysis capability of application performance.

Additionally, we have developed a Live/Virtual/Constructive capability at Sandia [6] that combines real devices, emulated devices, and simulated devices in a single hybrid experiment. We have identified use cases in our SpaceWire development activities that will benefit from merging our SpaceWire models into hybrid experiments to assess satellite network development ideas at various stages of the development. This approach is expected to support assessing the behavior of actual hardware prior to the availability of complete system hardware.

We have demonstrated a viable system distribution approach that can be employed without modification to the SpaceWire standard. The time distribution approach has been modeled in a high-fidelity simulator and our analysis has identified the range of time synchronization for various SpaceWire network architectures. Our broadcast solution has been fully developed in VHDL and tested in actual custom hardware. We continue with further integration and testing in actual hardware in our development activity.

7. REFERENCES

[1] European Space Agency, "SpaceWire - Links, nodes, routers, and networks." *ESA-ESTEC Requirements & Standards Division*. 24 January 2003.

[2] Klar, R., Dykes, S., Bertrand, A., Mangels, C., "Integration of Internet Protocols with SpaceWire using an Efficient Network Broadcast." *International SpaceWire Conference 2007.*

[3] Parkes, S., "The Operation and Uses of the SpaceWire Time-code." *ISWS International SpaceWire Seminar 2003.* November 2003.

[4] Roberts, A., Dykes, S. G., Klar, R., & Mangels, C. C. (2007, March). A Link-Layer Broadcast Service for SpaceWire Networks. *Aerospace Conference, 2007 IEEE* , 1-10.

[5] OPNET Technologies, Inc., www.opnet.com.

[6] Van Leeuwen, B., Urias, V., Eldridge, J., Villamarin, C., Olsberg, R., "Performing cyber security analysis using a live, virtual, and constructive (LVC) testbed," *IEEE MILCOM 2010,* October 2010.

[7] Cook, B., "Reducing SpaceWire Time-code Jitter." www.4links.co.uk/bibliography/Reducing-Time-Code-Jitter-on-SpaceWire.pdf , October 2003.

HARDWARE IMPLEMENTATION OF AN RMAP NETWORK SCHEDULER

Session: SpaceWire networks and protocols

Long Paper

Albert Ferrer, Steve Parkes

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

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

Alberto G. Villafranca, Martin Suess

On-Board Payload Data Processing Section, ESA/ESTEC, Noordwijk, The Netherlands

E-mail: gonzalez.alberto@gmail.com, martin.suess@esa.int

ABSTRACT

Payload control applications typically require that SpaceWire packets are delivered to the destination within certain time constraints, which is difficult to achieve with an event based wormhole switching network such as SpaceWire. One promising solution is to schedule the network to avoid contention and obtain a deterministic packet delivery time. This paper presents a proof of concept with a hardware implementation of an RMAP Network Scheduler compatible with current generation of network devices. The VHDL model developed configures and triggers the ESA RMAP IP Core, depending on the scheduling table stored in the memory allocated to the different channels, one for each pending user message. The highly configurable design supports segmentation and priorities, and is tolerant to network errors that could lead to temporally network congestion when using current generation of SpaceWire routers.

1 INTRODUCTION

SpaceWire [1] was designed to support payload data-handling applications using point-to-point links or networks. Data transfer is asynchronous and need not be deterministic. However, for spacecraft control applications, both payload and platform control, it is often required that data is delivered within certain time constraints. One promising solution is to schedule the network using time division multiplexing. With scheduling, there is no network contention and packet delivery time is deterministic. Is it then possible to obtain latency and throughput guarantees for the user data. The required periodic synchronization signal is easily provided using SpaceWire Time-Code (TC) characters. Time is divided into discrete time intervals or time-slots (TS) determined by the arrival of a Time-code.

SpaceWire uses wormhole switching, so packets are typically not buffered within the routers. Therefore, the scheduling is implemented at each transmitting node or network terminal using a local schedule table. Each local table must be configured following a global network scheduling, assuring that contention can not occur when no errors are present in the network.

One important question is which packet format or protocol should be used to encapsulate a user message. In this work the SpaceWire packets follow the Remote Memory Access Protocol (RMAP) specification [2]. RMAP is a transaction based protocol with one node, the Initiator, sending an RMAP command to read or write data to registers in a memory address located in another node, the Target. The use of RMAP has many advantages. It provides error detection using acknowledgments, it performs the most usual operations (read or write) with user messages of any size, and it is usually already implemented in typical SpaceWire systems. Besides, when the destination address is not a FIFO, it is safe to assume that the destination will be ready to handle the data of any write operation. In other words, it does not require end to end flow control to avoid stalling or rejecting a receiving packet.

This paper presents a proof of concept of an RMAP Network Scheduler using current generation of SpaceWire network devices. It tackles both design and implementation issues with a focus on simplicity, efficiency and compatibility with existing components.

The first sections will deal with the design challenges of a time division multiplexing technique, mainly:

- The duration of a Time-Slot has to be traded off to achieve a high data rate for payload data and a low latency for command and control operations.

- High priority event-based messages are difficult to schedule.

- Errors in the network can produce timing violations in the global schedule and induce unexpected contention.

The last sections deal with the actual implementation, based on the ESA RMAP IP Core [3] interfaced to a scheduler module developed in VHDL language.

2 RMAP SCHEDULER

As explained before, the basic idea of an RMAP scheduler implies that RMAP packets are sent at specific moments following a global synchronization that ensures that two different transactions do not use the same network resources at the same time. The simplest implementation may be the use of a local scheduler at each node that transmits an RMAP command just after the reception of a Time-Code. Figure 1 shows an example with two nodes that transmit read and write RMAP commands to a third node using a shared link.

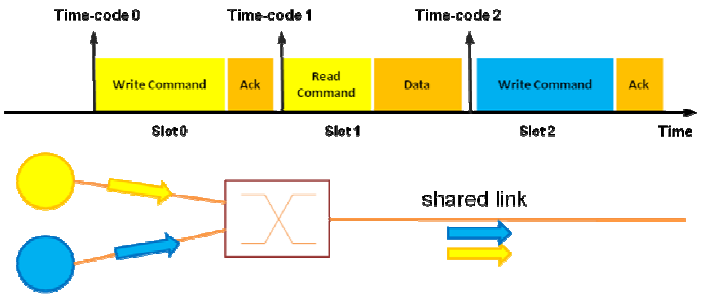


Figure 1: Two nodes transmit read and write RMAP commands to a third node using a shared link.

Multiple transactions could take place at the same time providing that they do not use the same network resources, i.e. they do not produce contention. RMAP transactions like SpaceWire links are bidirectional so each RMAP transaction should require a SpaceWire link. However, two RMAP transactions of the same type could use the same link without causing contention if they are coming from reverse directions, but this should be treated with caution.

The duration of a Time-Slot is a key parameter of this system and it must be the same across all the network, even if different link speeds are used. The data throughput increases with longer Time-Slots and the latency decreases with shorter Time-Slots. Besides, the maximum data length of an RMAP packet is restricted by its duration. This limitation is easily solved by implementing a segmentation layer.

For the trade off let's assume that only one transaction is allowed to be executed in a single Time-Slot and we want to optimise this period for the maximum link speed, 200 Mbps, which presents the best performance for space qualified hardware. If we consider that the protocol header, the network latency and the processing time cause an overhead of around 15-20 µs then the minimum Time-Slot could be around 62µs with RMAP data lengths of 512 or 768 bytes. Most command and control messages can fit within this RMAP data length, but most payload data messages will need to be segmented. Another option is to use a slot period of 125 µs with 2Kbyte segment size, but this increases the latency and the gains in throughput depends on the size of the payload message. For example, a 3Kbyte message will use two segments, one of them half size. Note that the segment size should be a multiple of a power of two, and the Time-Slot period should be a power of two division of one second (CCSDS CUC format compliance).

For longer slot periods then it is recommended to allow multiple transactions per Time-Slot. This increases the worst case latency and the complexity of the implementation. The biggest advantage is that it reduces timing constraints of software implementations and that multiple short control messages can be sent during the same slot leading to an increased link utilization. Our implementation does not yet support multiple transactions but provides another mechanism to increase the maximum throughput called multi-slotting. Multi-slots provide the capability to send a single RMAP command using multiple consecutive slots, reducing the overall overhead and increasing the throughput with short Time-Slots. The biggest advantage is that it allows a network to use different link speeds. For example, a node connected to a 50Mbit/s link would use four consecutive slots that will be equivalent to a single slot of a node working at 200Mbit/s.

3 RMAP SCHEDULER CHANNELS

Network scheduling is more efficient when the data traffic is known and periodic. It is difficult to schedule event-based messages that require low latency, specially if they use little bandwidth and are rarely generated. With a simple local schedule where each message must be allocated to a different slot, this rare, high priority message has to be allocated to at least one slot. This slot will be unused most of the time.

The solution is to implement a priority mechanism on top of the schedule table. This can be achieved with the introduction of the concept of priority channels within each node. A channel wraps a single RMAP message configuration (i.e. the header

including the destination) and its allocated Time-Slot numbers. It implements a transparent segmentation layer and provides sending status and error reporting. Multiple channels can be active at the same time. A channel is active when it has been configured with all RMAP parameters required and it has not send all its data. A long message may use multiple slots containing one segment of data. Each channel has a different priority level. Each Time-Slot the highest priority channel that is active is used. Once the highest priority channel finished sending its message, the lower priority channel is used.

To send critical sporadic messages efficiently we configure a high priority channel to use the same slots that have been already allocated to a long payload message using a lower priority channel. The long payload message is sent using multiple segments, one for each slot. When the control message must be sent it will be sent in the following allocated slot even if the long payload message is still active. The number of slots required must take into account the total bandwidth required. For example we could have one channel for payload data that requires six slots and two channels for two control messages that need half slot each, requiring a total of seven slots per epoch.

Our implementation requires that a channel must be reconfigured or retriggered each time an RMAP message is sent, except if the continuous mode is set. This allows to change the message rate or throughput of a particular channel and implement different slot allocations depending on the current epoch, increasing the efficiency of the system. The channel configuration can be performed by the node or by external network manager using RMAP.

4 ERROR HANDLING

Error detection is a critical feature for command and control operations so it is fully implemented in the proposed RMAP Network scheduler.

The network scheduling is highly sensitive to synchronization errors. The period between the arrival of consecutive Time-Code codes is measured with an internal clock and compared with the expected value. In case of discrepancy the system does not trigger the sending of any packet and report the error to the user application (early or late Time-Code arrival). The system also automatically resynchronizes without further user interaction.

The handling of transmission and reception errors also follows a fail safe approach, and the system goes silent to avoid error propagation. For example, if a packet is still being send at the end of a Time-Slot it implies that there is congestion in the network produced by this or another packet. SpaceWire routers based on the SpW10X model will remove stalled packets after some time. With our scheduling approach it can be guaranteed that only the packet that caused the network error will be removed first, allowing the others to continue towards the destination afterwards. Hence, the scheduler allows a packet being sent to continue being sent in the next slot. The slot following this one will be kept silent and not used, to avoid error propagation to all the network. This is analogue to the TCP congestion control mechanism.

Therefore, a channel is only stopped if a reply packet is not received after a programmable number of slots, considering that the command was not received or the

reply was lost. An error will be reported if there is congestion while transmitting, or if the RMAP reply did not arrive in the same slot, but the channel will not be automatically disabled. Note that if the channel is disabled, with the current generation of routers, any error in the network caused by a single channel could automatically disable multiple channels in other nodes of the network. It is preferred to add some extra latency and some loss of throughput than to lose multiple transactions across the network. This additional bandwidth can be taken into account at design time following an error probability model.

An optional feature is the activation of certain channels only when another channel presents an error. This can be used for remote error notification or for redundancy mechanisms. A retrial mechanism can also be implemented by the user application by retriggering the channel in error.

5 IMPLEMENTATION ARCHITECTURE

The RMAP scheduler has been prototyped in a Xilinx Virtex II and IV FPGAs with the ESA RMAP IP Core. A new IP Core has been created around the ESA RMAP IP Core. This solution was envisaged to minimize the development time.

The user or host application interacts with the scheduler by writing to a specific memory space containing the configuration for multiple RMAP channels and the global timing setup. The scheduler generates the RMAP information required by the ESA RMAP IP Core, mainly the RMAP header and data transaction pointers, for each segment of an RMAP message defined by the host. This segment is defined by the current status of the highest priority channel allocated to the next slot. When a Time-Code is received, it triggers the sending of the scheduled RMAP packet.

The RMAP Network Scheduler memory space (RNS configuration in Figure 2) can also be accessed by a remote network manager node using RMAP. So the system can work without a user application in the host and can be easily supervised. Figure 2 shows the architecture described.

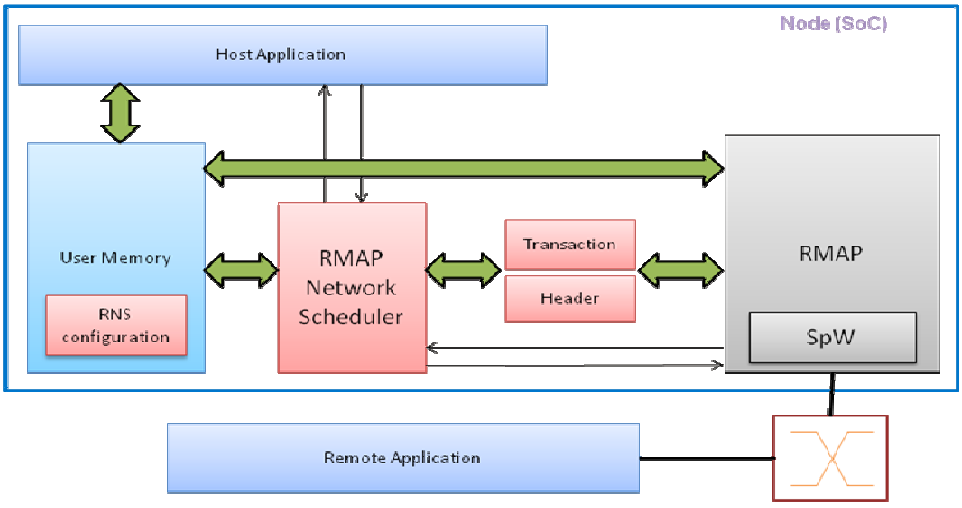


Figure 2. RMAP Network Scheduler implementation architecture.

In order to support error detection and the segmentation layer implemented, the lower byte of the transaction ID field of the RMAP packets sent is used. It contains the

channel identification, the sequence number and the flags start and end of message. It is filled automatically by the network scheduler. Because each channel implements a send and wait error detection scheme, only one bit for the sequence number is required.

Figure 3 shows a simplified block diagram. When a Time-Code is received, its timing it is check by the Time Code Handler. If it is a valid Time Code, the Error Detection module checks if the scheduled channel and the system is not in error. Then, the Transaction Trigger module triggers the ESA RMAP IP Core while the Channel Status Updater updates the state of the channel, including the segmentation status. Finally, the Transaction Generator selects the channel that will be active in the next slot, based on the schedule table, the priority level and the status of each channel. This module also generates the RMAP transaction associated to the selected channel.

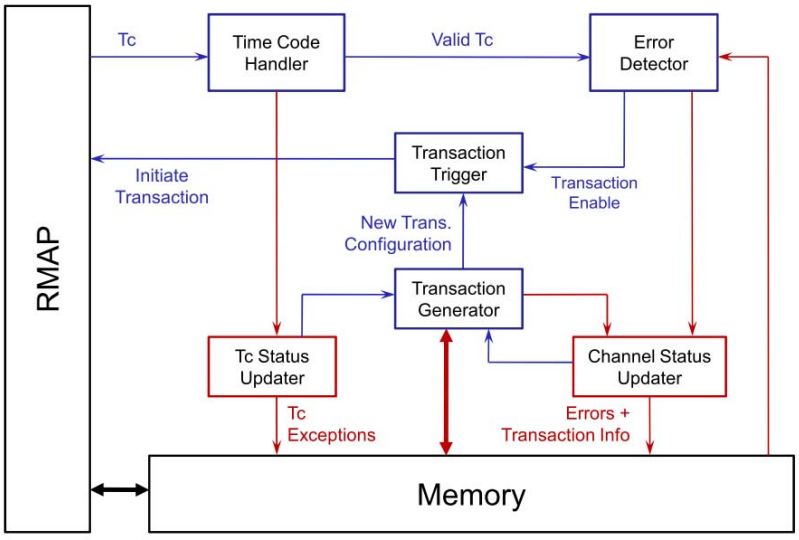


Figure 3. RMAP Network Scheduler block diagram.

The module can be configured at run-time with any Time-Code period and each channel can have a different message size and segmentation size value.

6 RESULTS

The implementation presents a very fast activation upon the arrival of a Time-Code. Specifically, the latency between a valid Time-Code and the subsequent activation of the RMAP module is just ~300 ns. The response time from the arrival of a Time-Code to the sending of the 1st segment (including RMAP IP core delay) is 3 µsec.

Table 1. Resource summary of the RMAP and the RMAP Network Scheduler IP Cores in a Virtex IV (LX100).

Logic Utilization	*RMAP IP Core*	*RMAP IP Core + Network Scheduler*	*Δ*
Slice Flip-Flops	3002	3868	+ 29 %
4-input LUTS	8722	10498	+ 20 %
Occupied Slices	5023	6207	+ 24 %

Regarding the area used, in Table 1 the values obtained for the whole prototype are compared against the original RMAP IP Core implementation. The results show how the part corresponding to the newly developed module roughly requires less than a quarter of the original RMAP IP core resources.

After having successfully verified the simulation stage, a set-up was designed to test RMAP Network Scheduler functionalities with real hardware. In Fig. 5 the test topology is presented. The idea behind the test is the simulation of a realistic scenario. A PC interacts through a SpaceWire brick with the different elements only for control purposes. All the system elements are connected through a SpaceWire router. There is an emulated mass memory element where the instrument and a payload processor (both featuring the Network Scheduler) can either read or write. An event logger simulates the registration of a high priority message.

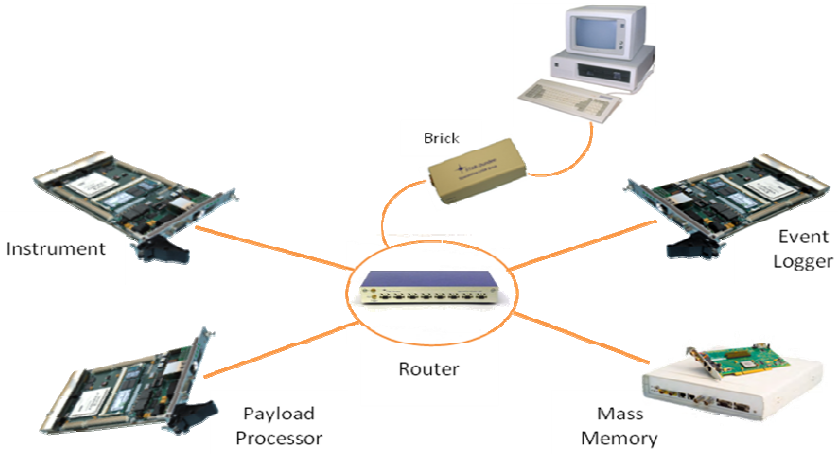


Figure 5. Topology of the test design with the Network Scheduler prototypes.

In the set-up, the instrument sends during even slots information packets to the memory. The payload processor alternatively reads and writes in the memory during odd slots simulating a compression process, for example. When manually asserting a signal, the Network Scheduler activates in the instrument a high priority channel to the event logger which can be sent in any slot. Note that in this case the payload processor can still R/W from the mass memory, as they use different paths. Finally, when the event logger link is disconnected – simulating a network error – an error channel is enabled at the instrument, then sending the error information to the payload processor on even channels. Note that the system has different features which are automatically activated to respond to different situations, providing a high degree of intelligence to the network. The entire memory space configuration required to implement this set-up was programmed through a series of Python specific applications which have been designed *ad-hoc*.

The test executed successfully, following the expected deterministic behaviour. The Time-Slot period was set to 50 µsecs. The link speed was set to only 100Mbit/s due to hardware constrains, and the segment size was set to 256 bytes. As stated, the response time was only 3 µsecs and there was not any NULL character inserted in the data stream. This means that the segment size could be slightly increased without requiring a longer time-slot period.

Figure 6 is a screenshot of a oscilloscope that illustrate the measurement of the latency and jitter of the high priority channel. The left side shows the assertion of the signal that enables the high priority channel. The right side shows a sequence of lines that indicate when the associated message has been received. We can see that the jitter is one slot period because the assertion of the signal is not synchronized with the Time-Slot period. The latency is higher than one slot period because the signal is sampled in the previous slot and the message is received some time after the next slot.

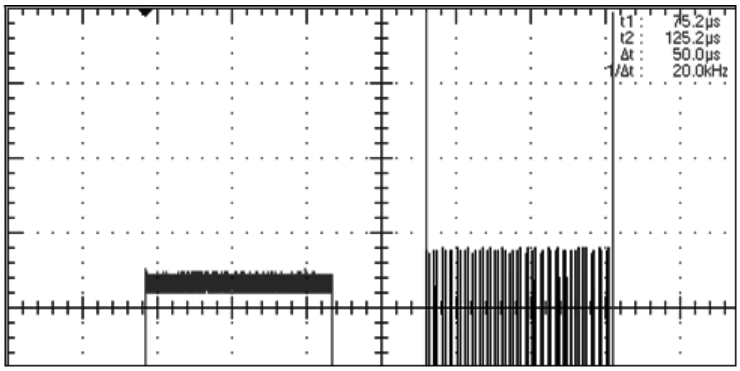


Figure 6. Measurement of latency and jitter for the high priority channel.

Another test set-up using a webcam and a computer mouse to simulate data and control information was presented in a SpaceWire Working Group [4].

7 CONCLUSIONS

A hardware prototype of the RMAP Network Scheduler has been developed in VHDL using the ESA RMAP IP core. It implements scheduling, segmentation and priority mechanisms, which provide latency and throughput guarantees with high network efficiency. The prototype also incorporates flexibility to support multiple configurations and user cases. Furthermore, the design incorporates advanced error detection and notification mechanisms. When combined, these solutions deliver high performance scheduling and segmentation capabilities to RMAP applications with a small additional cost. The RMAP Network Scheduler has been successfully validated through simulations and implemented in Virtex II and IV FPGAs. The FPGA version has proved its capabilities in a realistic scenario with several RMAP devices interacting in real time.

8 REFERENCES

[1] Parkes S.M. et al, 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.

[2] Parkes S.M. et al, European Cooperation for Space Standardization, Standard ECSS-E-50-11, “SpaceWire Remote Memory Access Protocol”, Draft-E, European Cooperation for Space Data Standardization, 21st December 2005.

[3] SpaceNet RMAP IP core User Manual Issue 1.5, 18th May 2009

[4] ESA, “SpaceWire Working Group Website”, European Space Agency, http://spacewire.esa.int/WG/SpaceWire/

AVOIDING SPACEWIRE NETWORK CONGESTION

Session: SpaceWire Networks and Protocols

Long Paper

Martin Suess

ESA, ESTEC, 2200 AG Noordwijk, The Netherlands

E-mail: martin.suess@esa.int

Albert Ferrer

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

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

ABSTRACT

SpaceWire links, interface components and routing switches support today data rates up to 200 Mbps between two units for the use on board of satellites. This is more than two magnitudes higher than what can be provided by Mil Std 1553 or CAN bus. Despite the very high bandwidth of the links, excessive network congestion can occur and this might be only discovered late in the project during system integration and validation. In order to avoid these problems the system designer needs to have a good understanding of the properties of the SpaceWire network. The right set of requirements needs to be put not only for the SpaceWire network itself but also for the design of the connected units and applications.

1 SPACEWIRE NETWORK

SpaceWire is a bidirectional point to point link and the data transmission is regulated by flow control. It uses data–strobe encoding and LVDS according to ANSI/TIA/EIA-644 as signalling level. The link starts up with a data rate of 10Mbps and is then switched to the operational data rate. This operational data rate can be set for each individual link and can be different for both directions. Whenever there is no data ready for transmission or if the receiving side is not ready to receive more data the link is kept busy by transmitting NULL characters.

In a SpaceWire network these links connect SpaceWire nodes as sources and destinations of SpaceWire packets. The packets start with the path and the logical address of the destination node followed by the Protocol-ID and end with an End-of-Packet marker (EOP). The next data character following the EOP is considered to be the first byte of the following packet. SpaceWire does not define any maximum length of the cargo transported between the packet header and the EOP.

One or more routing switches are needed if more than two nodes have to be interconnected. The SpaceWire network is operated asynchronously as the link data rate is tied to the local clock of the transmitting node. There is also no synchronisation

between the nodes and the time of the start of a packet transmission depends only on the transmitting node. The arbitration needed when two packets from different source nodes want to access the same link to reach their destination node takes place in the SpaceWire routing switch. For this paper round robin arbitration is assumed.

2 WORMHOLE SWITCHING

The SpaceWire standard specifies the implementation of wormhole routing within the routing switches [1]. As soon as the header containing the destination address information is received the routing switch is forwarding the packet on the next link leading to the destination while the tail of the packet is still being received. This way the packet can span like a worm through several links and routing switches in the network or even all the way between source and destination node. While being routed the time to reach the destination depends on the network congestion at the routers which are passed on the way. Once the packet has reached the destination the remaining transfer time depends on the packet length and effective transfer date rate.

The advantages of wormhole routing is that it only requires minimum buffer space in the routing switch keeping it simple, low power and offering minimum latency. In combination with the link level flow control it ensures that no packets are dropped by the network. Wormhole routing was previously used in Transputer systems and is today used in many Network-on-Chip.

Another common switching method is called store and forward. It is for example on of the methods applied in the widely used in Ethernet standard IEEE 802.3. The routing switches have to be able to buffer incoming packets before forwarding then to their destination. The size of this buffer automatically sets a limit on the allowed packet size or requires segmentation of larger packets. Compared to wormhole switching much more memory needs to be implemented inside the switches. In order to improve the latency of the store and forward approach the virtual cut through method [2] has been developed. It provides similar path latency as wormhole routing with the difference that the packet is buffered in the switch only if the route to the destination is blocked. By buffering the packet this method can reduce the propagation of congestion in the network. This is because the buffered packet will not block like in the wormhole routing case multiple links in the network because its destination is blocked or a path on the way to the destination is busy.

3 USE OF DIFFERENT LINK SPEEDS IN A NETWORK

As discussed before, SpaceWire can be set to run at a wide range of different link speeds. At first glance this appears to be an attractive feature but it has to be applied with great care. In the example shown in Figure 1 the nodes 1, 2 and 3 are instruments which have to transmit the generated payload data to the mass memory in node 4.

The setup in this simple example could work if the routing switch would implement store and forward. If wormhole switching is used as it is the case in SpaceWire it will certainly not provide the required bandwidth. With wormhole switching the slowest link in the path through the network determines the overall bandwidth of the end to end connection. For the period of the packet transmission the faster links are effectively running at the slower speed. In the example shown in Figure 1 the 100Mbps link between the routing switch and node 4 will be used with a maximum

throughput of 40Mbps during the transmission of packets from node 3 and even less during the transmission of packets from the two other nodes. The throughput of 40Mbps corresponds to the maximum data rate which can be transferred over a link running at 50Mbps. This 20% overhead is due to the parity and the data-control flag used in addition to the 8 data bits in the coding of a SpaceWire data character.

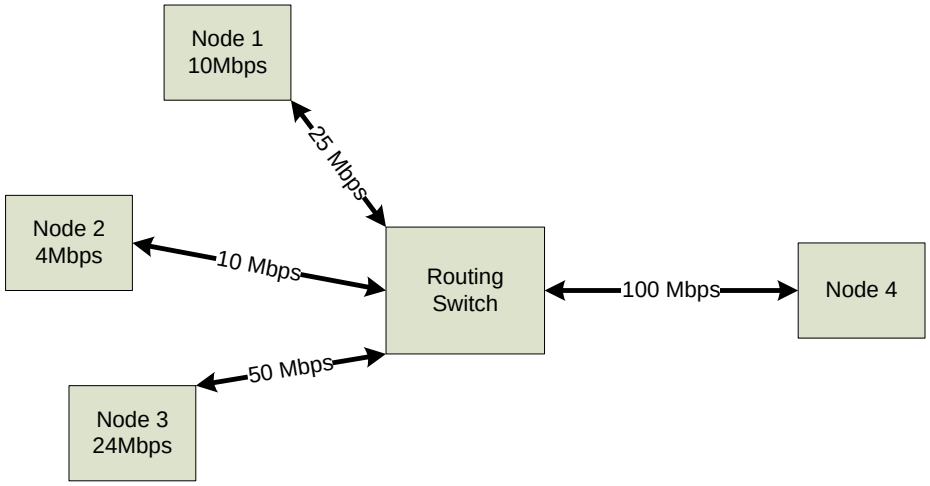


Figure 1: Simple SpaceWire network with links operated at different data rates

When assuming that the continuous payload data stream is segmented at the instrument nodes and that the individual segments are transmitted at the full link bandwidth a link occupation duty cycle can be defined as the average transmitted data rate divided by the actually achievable link bandwidth. According to this calculation the traffic from node 1 and 2 uses their links with a duty cycle of 50% each. The traffic from node 3 has a duty cycle of 60%. The combined traffic from node 1 and 2 is therefore already fully occupying the capacity of the link to node 4. With the additional traffic from node 3 the link to node 4 would have a theoretical occupation of 160% which is clearly exceeding the link capacity. When the instruments are generating data at a higher rate than the link capacity the local buffer space will fill up and overflow. This overflow will then lead to a loss of instrument data.

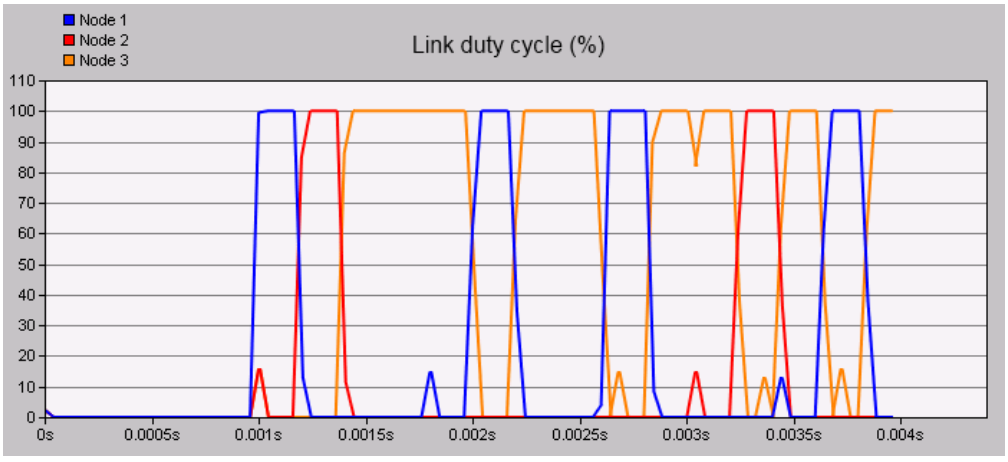


Figure 2: Link duty cycle in % for the links from nodes 1 to 3 if all links are operating at 50Mbps

In order to cure the situation in this example sum of the link occupation duty cycles of the links from nodes 1 to 3 which has to be transmitted through the single link to node

4 must to be kept below 100%. If all the links would be operated at 50Mbps (40Mbps net) the sum of the link duty cycles 1 to 3 would be 25%+10%+60%=95% which could be a workable solution as shown in Figure 2. This is of course not the only solution. When changing for example the speed of the links from node 2 and 3 to 100Mbps the sum of the link duty cycles would be 50%+5%+24%=79%.

Unfortunately avoiding the loss of data is not just a matter of selecting the correct SpaceWire link speeds. This is already indicated by the conditions given in the beginning of this simple example. Node specifications often only address the nominal SpaceWire link speed and not the actual achievable data rate of the node. This data rate can be significantly less due to internal limitations of the hardware or software in the node. This effective data rate has to be taken into account when analysing the network congestion and throughput.

4 DATA BUFFER SIZES

One common application of SpaceWire is the transfer of payload data to the mass memory on board. When the payload is operating it can often be modelled as a continuous source of data at a given rate. These data have first to be buffered and segmented inside the node. Only once the packet containing a data segment is complete it can be sent out though the link at a data rate that should be close to the net link data rate. The properties of the data buffer inside the node is therefore of crucial importance.

The size of this buffer which is required to avoid the loss of data depends on the product of average source data rate and the worse case latency for the end to end connection. It can be shown that an upper bound for the worse case latency of a connection can be calculated analytically for any given network [3], [4], [5]. This latency depends on the network topology, the interfering data connections, the effective link speeds and the packet sizes used. One additional factor which needs to be taken into account as well is the latency of the target node. The target node could for example be busy with some other task when the packet arrives on the SpaceWire link and it takes some time before the packet is ingested by this node. The overall maximum latency can be reduced by introducing segmentation and splitting large data packets into smaller segments which are transmitted independently.

5 REQUIREMENTS AT NODE LEVEL

All this can be calculated and must be analysed before the SpaceWire network and the nodes and units are specified. This can then be used to determine the requirements at node level concerning the SpaceWire network interface.

The following parameters need to be specified for source nodes:

- SpaceWire link speed,
- Minimum effective data rate with which a packet can be sent out on the network,
- Maximum average data rate the node is allowed to send out,
- Segment or packet size,
- Minimum source buffer size.

The following parameters need to be specified for target nodes:

- SpaceWire link speed,
- Segment or packet size,
- Minimum effective data rate with which a single segment or packet can be received from the network,
- Maximum delay time before packet is received at the effective data rate,
- Minimum average receive data rate that can be sustained which may sometimes be less than the effective receive data rate.

It can be considered a good practice to perform a network traffic simulation which included the properties of the nodes [6] in order to validate the specification.

Before system integration all these requirements should be tested and verified at subsystem level.

6 CONCLUSION

When resources have to be shared in a network there is always the possibility of contention. If not designed carefully this may lead to a loss of data due to source buffer overflow. The wormhole switching used in the case of SpaceWire network is well researched and understood. It does not use any buffering inside the network but it requires buffers located in the node instead. The required size of these buffers inside the nodes has to be carefully analysed. Sufficient buffer space is a key property to avoid data loss. There are a number of parameters which need to be specified for the nodes in order to control the level network congestion. It is important that these requirements are tested and verified already at node level. In this way it can be avoided that excessive network congestion is only discovered after integration during the overall system test.

7 REFERENCES

[1] ECSS Standard ECSS-E-ST 50-12C, “SpaceWire, Links, Nodes, Routers and Networks”, 31 July 2008

[2] P, Kermani and L, Kleinrock, “Virtual cut-through: A new computer communication switching technique,” Computer Networks, vol. 3, pp 267-286, 1979

[3] Albert Ferrer Florit, “SpaceWire and Determinism: concepts”, Proceedings SpaceWire WG meeting 14, 22 February 2010.

[4] T. Ferrandiz, F. Frances, Ch. Fraboul, “A Network Calculus Model for SpaceWire Networks”, Embedded and Real-Time Computing Systems and Applications (RTCSA), vol.1, no., pp.295-299, 28-31 Aug. 2011

[5] Dally, W.J.; "Performance analysis of k-ary n-cube interconnection networks," Computers, IEEE Transactions on , vol.39, no.6, pp.775-785, Jun 1990

[6] P. Fourtier et.al., “Simulation of a SpaceWire Network”, Proceedings International SpaceWire Conference 2010, 22-24 June, St. Petersburg, Russia

Networks and Protocols 4

LOW-LATENCY PACKET DELIVERY IN SPACEWIRE NETWORKS

Session: Networks and Protocols

Long Paper

Dr Barry M Cook, C Paul H Walker

4Links Limited

E-mail: barry@4Links.co.uk paul@4Links.co.uk

ABSTRACT

This paper quantifies typical latency requirements and describes a simple technique that uses virtualization and priorities with dynamic, on-demand segmentation, to provide deterministic, low latency delivery of packets whilst allowing high utilisation of the network. Segmentation is low-level and invisible to the user (and to the API). Packets, of any size, will be delivered to the destination node as a contiguous whole, without interleaving and with the contents in strict order. The technique offers the ability to carry data with real-time and low-latency requirements, such as command and control (including unscheduled events), at the same time as, and completely fire-walled from, high-bandwidth data such as that from experiments and instruments.

1 INTRODUCTION

This paper brings together with SpaceWire the themes of Virtualization and Time Triggering. Virtualization has proved to be a secure way to share resources on a computer, such as virtual servers. Time Triggering has become a popular means of providing deterministic (but often very high) latency over a bus.

SpaceWire, as defined in [1], offers very low latency for real-time control loops and for housekeeping accesses, provided that such accesses are not contending with large data transfers, or with blockage in the network.

We describe an enhancement to SpaceWire that provides Virtual SpaceWire Networks [2, 3] and thus brings the benefits of protecting users from each other and of isolating faults. Virtual SpaceWire Networks (VSNs) retain the exceptionally low latency of current SpaceWire for deterministic real-time traffic. Meanwhile, the low priority and bulk data transfer can use all the bandwidth that is not taken by the real-time traffic.

We use priority within the Virtual SpaceWire Networks to ensure that latencies and control loop times can be guaranteed. Each Virtual Network can have its own individual priority, or several Virtual Networks can share the same priority.

In this paper, we acknowledge that many missions have control loops and housekeeping accesses that repeat at 1 second, 100ms, 10ms or shorter periods. We simply group the accesses in each of these sets of periods into separate Virtual SpaceWire Networks.

2 AN EXAMPLE NETWORK AND CALCULATION OF AVAILABLE THROUGHPUT

Table 1 below shows an example set of traffic such as would be used on a satellite and Figure 1 shows a subset of the activity of that traffic on several Virtual SpaceWire Networks sharing a single physical SpaceWire link.

The table and figure include traffic for a couple of 5kHz control loops which need to access data and process it within 200µs. We give this highest priority (priority 1). Priorities 2 to 5 are used for slower control loops or for regular housekeeping updates, at frequencies from 1kHz down to 1Hz (priorities 3 to 5 are omitted from the figure). The lowest priority (6) is used for bulk data, such as image data from cameras.

3 ASSUMPTIONS

Any calculation of performance or guarantees needs to be based on assumptions. We'll make our assumptions explicit and then describe, in general terms, the way the numbers in the table have been calculated and what performance can be guaranteed.

1. We assume a worst-case that all the traffic is shared on a single Virtual SpaceWire Network link. In practice, performance scales with additional links.

2. We assume that the accesses are RMAP (Remote Memory Access Protocol) Read requests and responses. RMAP has higher overheads than some other protocols used on SpaceWire, so our use of RMAP here gives conservative results, and this analysis is in no way confined to use of the RMAP protocol.

3. We assume a very worst case that all the RMAP initiators, with all the different frequencies, start sending their requests at the same time and so will be queued.

Frequency	Number of requests in period	Response Payload, Bytes
5kHz	2	20
1kHz	10	50
100Hz	25	200
10Hz	50	200
1Hz	100	200

Table 1: Set of real-time traffic used as an example

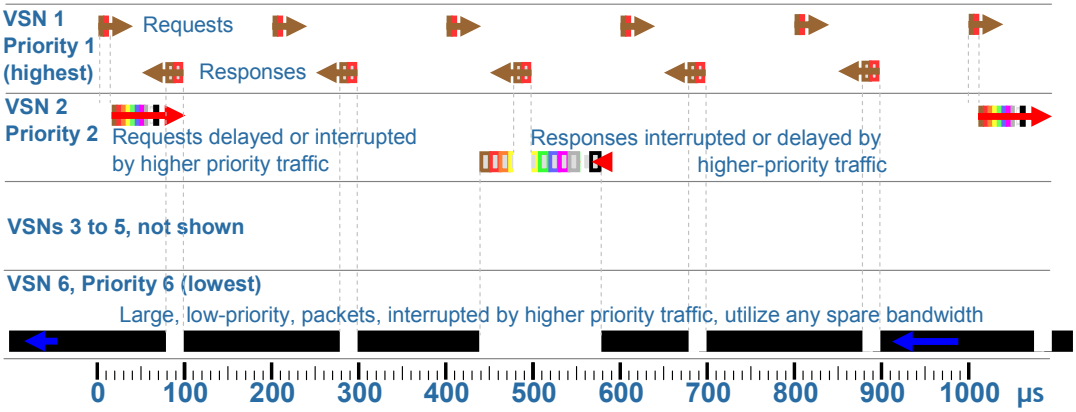


Figure 1: An example of activity on a Virtual SpaceWire Network link over time

4. We assume that a node transmitting an RMAP request has adequate buffer space to receive the full response that it has requested without stalling. And we assume that a node transmitting a response is able to send the complete response without stalling.

5. We assume that all the nodes are current SpaceWire standard nodes, so that a node can only send requests or responses to one Virtual Network. (There can be benefits in having nodes that support multiple Virtual Networks, which will be described later).

6. We assume a link speed of 50Mbits/s. While this amply meets the requirements, even with the worst case assumptions used here, faster link speeds could be used to carry more data or to give even faster responses.

With these assumptions, we can now look at an algorithm for calculating whether the latencies and processing times meet the requirements for completion within the relevant Period at a particular priority.

4 ALGORITHM FOR CALCULATING REAL-TIME AND THROUGHPUT PERFORMANCE

1. For each VSN, add up the **Network delay for requests on this VSN in this period** from the start of the first request being transmitted to the end of the last request reaching its target. These delays include:
    - Delay in the Initiator in transmitting the first packet
    - The transmission time for the total number of Bytes, in all the requests for this VSN, at the link speed of the SpaceWire link
    - An overhead on the transmission time, to allow for flow control characters, an occasional Time Code, and for the possible overhead of switching between Virtual Networks
    - The transmission time for any Nulls that the Initiator inserts into a request (some designs may be unable to send contiguous packets)
    - Total cable delay (although this is probably negligible)
    - Total Routing-Switch latency

2. For each VSN, determine the **Longest target latency** of the various targets on this Virtual SpaceWire Network. This is the time from the end of the request packet to the start of the response packet. The latency should be determined from the manufacturer's data sheet and confirmed by characterization with test equipment [4]

3. For each VSN, add up the **Network delay for responses on this VSN in this period** between start of the first response being transmitted to the end of the last response reaching its initiator. These delays are similar to the network delays for requests and include:
    - Delay in the Target transmitting the first packet
    - The transmission time for the total number of Bytes, in all the responses, at the link speed of the SpaceWire link; note that, while most of the requests are the same length as each other (or very similar) the responses may vary in length depending on the nodes being accessed.

- An overhead on the transmission time, to allow for flow control characters, an occasional Time Code, and for the possible overhead of switching between Virtual Networks
- The transmission time for any Nulls that the Target inserts into a transmitted request (there should not be any but some designs may not be able to send contiguous packets)
- Total cable delay (although this is probably negligible)
- Total Routing Switch latency

4. For each VSN, determine the **Longest processing time** of the various controllers/initiators on this Virtual SpaceWire Network. The processing time is the time from the end of the response packet arriving at the initiator to the end of any action it needs to take as a result of the response.

5. For each VSN, add up the following:
   - the **Network delays for requests on all (higher or equal)-priority VSNs in this Period**. Note that if there is a single VSN at the highest priority, this sum will be zero. Note also that the Network delays must account for *all* the delays at equal or higher priority during the Period of this VSN.
   - the **Network delay for requests on this VSN in this Period**
   - the **Longest target latency**
   - the **Network delays for responses on all (higher or equal)-priority VSNs in this Period**. Note that if there is a single VSN at the highest priority, this sum will be zero. Note also that the Network delays must account for *all* the delays at equal or higher priority during the Period of this VSN.
   - the **Network delay for responses on this VSN in this Period**
   - the **Longest processing time** [1]

   and if this total is less than the Period, then the set of accesses can be guaranteed to take place within the Period.

[1] It may be excessively conservative, on top of the worst case assumptions, to include the longest target latency and the longest processing time in the calculation. An alternative would be to sum the {target latency plus processing time} for each separate access, and then add the longest of these sums to the request and response network delays to ensure that the total is less than the Period.

5 A SPECIFIC EXAMPLE

We'll consider, as an example, a small subset of the activity shown in Figure 1, and check the behaviour for the highest priority Virtual SpaceWire Network.

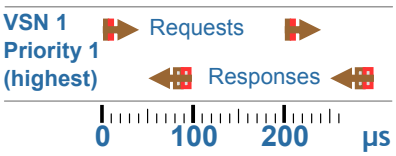


Figure 2: Detail of activity for an example 5KHz control loop

For this there are two initiators and two targets with the initiators and targets sharing a single link between two routing switches. The numbers we use are arbitrary, but are a reasonable estimate based on products that 4Links have characterized. Note that the period of 200µs implies 5kHz control loops, and that these have relaxed constraints on the end-nodes even with a link speed of 50Mbits.

The Delay in both the Initiators in transmitting the first packet is 1µs.

The transmission time for an RMAP Read Request is around 24 Bytes, making a total of 48 Bytes for the two Initiators. At 50Mbits/s, allowing for eight bits of data in ten bits transmitted, and for the Ends of Packet, that takes a total of 9.8µs. We'll allow a 10% overhead on the transmission time, to cover flow control characters and the possible overhead of switching between Virtual Networks. This brings the total transmission time to 10.8µs.

At 50Mbits/s, these Initiators do not insert Nulls into the data stream unless they are starved of flow-control credit, which should not occur when the Target is waiting for a Request.

Cable delay, at under 5ns per metre, and a total cable length of less than ten metres, is sub 50ns and so will be ignored.

Routing Switch latency, for several switches that 4Links has measured, is around 1µs. With two Routing Switches, the total delay in this direction is 2µs. This makes a total **Network delay for requests** of (1+10.8+2) = **13.8µs**.

The **Longest target latency** depends heavily on the devices used and whether the protocol and response are handled in hardware or software. In this case we take, as an example, an RMAP target that uses a processor and software dedicated to the one target and that it responds in **50µs**.

The **Network delay for responses** is a similar calculation to that for requests. In this case any equivalent of the Initiator delay is included in the target latency. The response payload is 20Bytes and the RMAP overhead is another 20Bytes. These 40Bytes for each of the two responses at 50Mbits/s, take 16.2µs. We again allow a 10% overhead on this to arrive at a transmission delay of 17.8µs. There should again be no Nulls inserted because the initiator should not request a response that it can not handle. Routing Switch Latency is as for Requests, at 2µs. This makes a total **Network delay for responses** of (17.8+2) = **19.8µs**

The **Longest processing time** needs to be measured by test equipment or calculated from simulation of the software or, perhaps preferably, by both. In this example, the sum of the two network delays, (13.8+19.8) = **34µs** plus the target latency of **50µs**, is 84µs. With the Period of 200µs, this leaves considerably more than **100µs** available for the processing time.

A more complete example of these calculations is given in [5] and is summarized in Table 2.

Virtual Space-Wire Network (VSN)	Prio-rity	Period, µs	Freq-uency	Number of requests in period (n)	Response Payload, Bytes	Worst case network delay for requests on this VSN in this period, µs (% of period)	Worst case network delay for responses on this VSN in this period, µs (% of period)	Shared link Request direction utilization	Shared link Response direction utilization
1	1	200	5kHz	2	20	13.8 (6.9%)	19.8 (9.9%)	5.4%	8.9%
2	2	1000	1kHz	10	50	57 (5.7%)	157 (15.7%)	5.4%	15.5%
3	3	10000	100Hz	25	200	138 (1.4%)	1214 (12.1%)	1.3%	12.1%
4	4	100000	10Hz	50	200	273 (0.3%)	2428 (2.4%)	0.3%	2.4%
5	5	1000000	1Hz	100	200	542 (0.1%)	4853 (0.5%)	0.1%	0.5%
						Real-Time utilization		12.5%	39.5%
6	6	**Available bandwidth for (lowest priority) bulk data**						>80%	>50%
		Total network utilization possible						>90%	>90%

Table 2: Calculations of latencies and bandwidth for the traffic shown in Table 1

6 More Complex SpaceWire Networks

The calculations above were done for a single SpaceWire link, and obviously SpaceWire is used for more complex topologies, such as the ring shown at right. A conservative measure (again worst-case) of both real-time and data throughput performance could be gained by simply treating the whole ring as a single SpaceWire link. If that gives adequate performance, no further work is necessary. If more performance is needed, each link between routing switches can be considered separately — which is still a much simpler calculation than would be needed for a conventional time-triggered network.

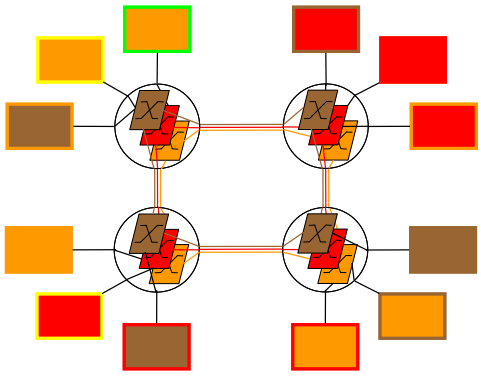


Figure 3: Ring network using Virtual SpaceWire Networks between routing switches

7 Reducing Power and Harness Mass

Most current recommendations for SpaceWire are that all the SpaceWire links should run at the same speed. Otherwise, for a current SpaceWire link multiplexing traffic between many nodes (as in the main example above), the throughput on the shared link drops to the throughput of the slowest link. Virtual SpaceWire Networks remove this dependency, and so permit the peripheral links to nodes or end-points to run at the appropriate speed for the node rather than for the whole network. Reducing link speed at the periphery of the network can result in substantial savings of power (and cost).

The major saving in harness mass is from using a single (Virtual) SpaceWire network instead of one (SpaceWire) network for data and another network/bus for control. Significant additional savings in harness mass can be gained compared with current SpaceWire by multiplexing several slow links over one faster link.

8 COMPATIBILITY WITH EXISTING NODES

All the examples shown have been with all the nodes being current SpaceWire. No change to hardware or software is necessary to any well designed node connected to a Virtual SpaceWire Network routing switch.

9 BENEFITS OF VSN NODES

Nodes that are accessed both for housekeeping and for large volumes of data could benefit from having access both to high priority traffic for the housekeeping and to low priority for the data. As described in [3, 4], such nodes could have a separate SpaceWire link for each priority, or use an extended CODEC that supports two or more priorities/VSNs. Nodes supporting multiple priorities must separate the two or more priority levels to prevent priority inversion.

10 FAULT ISOLATION AND RECOVERY

It is possible for a SpaceWire node to block another, by continuously transmitting (babbling idiot) or by failing to transmit for lack of flow-control credit or other error such as software stalling. Virtual SpaceWire Networks provide isolation for such faults:

1. **From faults at lower priority:** The highest priority Virtual SpaceWire Network sees an empty network, and is completely isolated from faults in lower-priority virtual networks. In general any VSN is isolated from faults in any VSN at lower priority than itself.

2. **From faults at the same or higher priority:** It might appear from (1.) that faults on higher priority VSNs are able to block everything at a lower priority. But, as in our example above, it will be normal for the real-time traffic to take a small proportion of the overall network bandwidth. VSN routing switches could police that proportion and discard data from a node that is exceeding the permitted percentage of bandwidth utilization for that priority level. This would leave up to 80% of the bandwidth available to lower priorities, even in the event of a fault at the highest priority — bandwidth which could be used to recover from the fault.

3. **From faults within a single Virtual SpaceWire Network:** Several existing routing switches perform a gatekeeper function by setting timeouts so that, if a node is blocked for longer than the timeout, the blocked packet is discarded. It can be difficult to calculate the appropriate timeout value if the minimum value to meet one criterion is longer than the maximum value to meet another criterion. This issue is much simpler to resolve when there is a separate VSN for each frequency of access, and the timeouts can be set appropriately for each VSN.

11 CONCLUSIONS

We have presented here a simple solution for supporting real-time requirements on a SpaceWire network. A link speed of 50Mbits/s is amply able to meet the requirements of 5kHz control loops.

The solution can, at the same time and with conservative (worst case) assumptions, offer well above 50% of the network bandwidth for volume data transfers, that may use very large packets, while still providing microsecond response times to real-time traffic.

One of the Virtual SpaceWire Networks could be used for a time triggered protocol.

By replacing only the routing switches in a SpaceWire network, Virtual SpaceWire Networks provide the following benefits for missions:

- the simplicity in both concept and use of Virtual SpaceWire Networks, with a corresponding reduction in mission complexity;
- use of a single physical network for both command/control and, separated by a firewall, for volume data;
- reduction in power consumption, cable/harness mass, and thence cost
- complete compatibility with existing SpaceWire nodes;
- complete compatibility with (and transparency to) higher-level protocols (including CCSDS, SOIS and PnP) running over SpaceWire;
- consistency with the layering of the SpaceWire standard so that no change is required to the ECSS SpaceWire standard;
- greatly improved fault-isolation and recovery.

12 REFERENCES

1. ECSS Secretariat, "ECSS-E-ST-50-12C 31 July 2008, SpaceWire - Links, nodes, routers and networks", ESA-ESTEC, Requirements & Standards Division, Noordwijk, The Netherlands
2. B M Cook (4Links) "Virtual Networks", SpaceWire Working Group Meeting 13, ESTEC, Noordwijk, The Netherlands, 2009-09-16
3. 4Links, "Virtual SpaceWire Networks", White Paper, 4Links Limited, UK, 2009-09-03
4. 4Links, "Characterizing SpaceWire Devices and Networks", White Paper, 4Links Limited, UK
5. 4Links, "Virtual SpaceWire Networks: Example latency and throughput calculations", White Paper, 4Links Limited, UK

VIRTUAL CHANNELS, BROADCAST CHANNELS AND SPACEFIBRE

Session: Networks and Protocols

Long Paper

Steve Parkes,

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

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

Martin Suess,

ESA, ESTEC, 2200 AG Noordwijk, The Netherlands

Email: martin.suess@esa.int

1 ABSTRACT

SpaceWire is a widely used spacecraft onboard data-handling network, which operates at up 200 Mbits/s in current radiation tolerant technologies [1]. While ideal in many respects for onboard data-handling applications it does not have sufficient data-rate for some applications, does not provide galvanic isolation which is important for fault detection, isolation and recovery (FDIR), and does not provide quality of service (QoS) which can make application integration much simpler. Furthermore SpaceWire networks can suffer from blocking of packet data if they and the data transfers are not designed carefully. SpaceFibre [2] [3] [4] plans to address these problems.

This paper first introduces virtual channels and describes how they can be used to overcome the network blocking problem. SpaceFibre is then introduced, which provides around 10 times the data rate of SpaceWire. Its virtual channels are then described, which support several different QoS types, targeted at specific spacecraft onboard communication needs. A low latency message broadcast mechanism is then introduced.

2 VIRTUAL CHANNELS

SpaceWire employs a network made up of SpaceWire links and wormhole routing switches. When a SpaceWire packet arrives at a wormhole routing switch the first character of the packet determines which port a packet should be routed to. If that addressed port is available (not currently being used to send another packet) the incoming packet will be sent out of that port straightaway. This provides very low latency routing, unless the output port is blocked. When the output port is being used, the incoming packet has to wait until the output port has finished sending its packet. The incoming packet is then temporarily stuck, with its tail strung out across the network blocking other network resources.

The advantage of wormhole routing is that it is very simple and requires very little buffering in the routing switches. This was a significant consideration when

SpaceWire was being developed because of the limited memory space available on FPGAs and ASICs at that time.

To overcome the blocking of a wormhole routing switch there are several different approaches that can be considered, including: packet buffering and virtual channels.

Packet buffering, buffers a complete packet either at the input or the output of a routing switch. This prevents blocking in the routing switch itself but requires a large amount of memory for buffering. It also means that one of the advantages of SpaceWire, completely arbitrary packet length, is lost.

The virtual channel concept is a well know concept for multiplexing multiple sources of data over a single physical data link, which can be used to solve the packet blocking problem and to provide QoS support [5] [6]. Data to be transferred over a single data link from several different data sources is separated into chunks that are multiplexed over the single data link and then reassembled into the different data streams at the other end of the link.

One method of multiplexing the SpaceWire packets from several sources over the data link is to split each packet into small frames each of which contains a channel identifier. Information for checking for errors (CRC and frame sequence number) can be added to each frame to support fault detection. The addition of a simple go-back-N retry mechanism can then be used provide recovery at the link level.

A medium access controller determines which of the virtual channels is to next send data over the data link. This can be done using fair arbitration, priority arbitration, or another quality of service mechanism more closely aligned with the onboard data communication needs. Different quality of service mechanisms can be applied to each virtual channel.

A virtual channel operates over a single link. There are two ways in which this can be used at the network level: as an end-to-end virtual channel, or as a virtual network [5] [6]. The end-to-end virtual channel has one route through a network (with possibly one or more additional redundant routes) and connects one node to another node, for example an instrument to a mass-memory unit. A virtual network allows a source node to send to any node on the network, for example a control processor sending commands to several different instruments. A SpaceFibre routing switch, routing data according to its destination address, which uses the same virtual channel number that the packet arrived on for sending it out, will naturally implement a virtual network. The end-to-end virtual channel concept can be used to add further FDIR capability into the SpaceFibre routing switches.

3 SPACEFIBRE

An overview of the SpaceFibre CODEC architecture is provided in Figure 1.

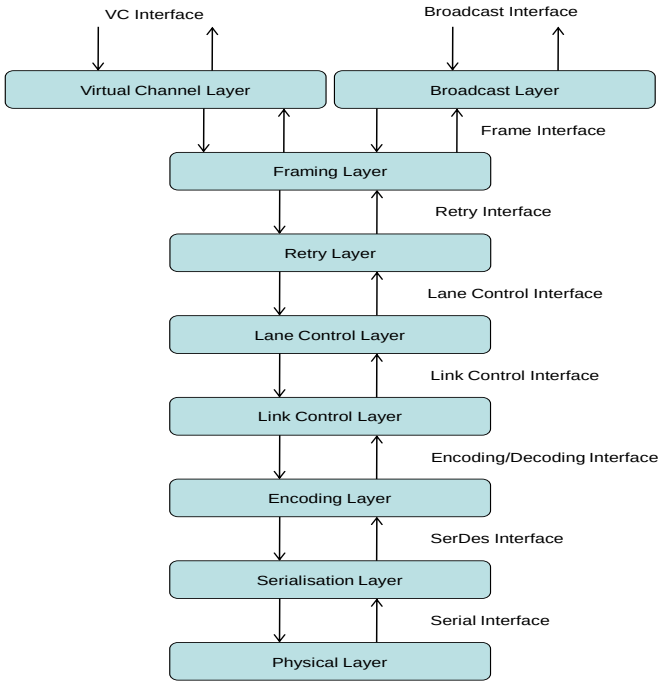


Figure 1 SpaceFibre CODEC architecture overview

The virtual channel layer and broadcast layer are considered in this paper. A more

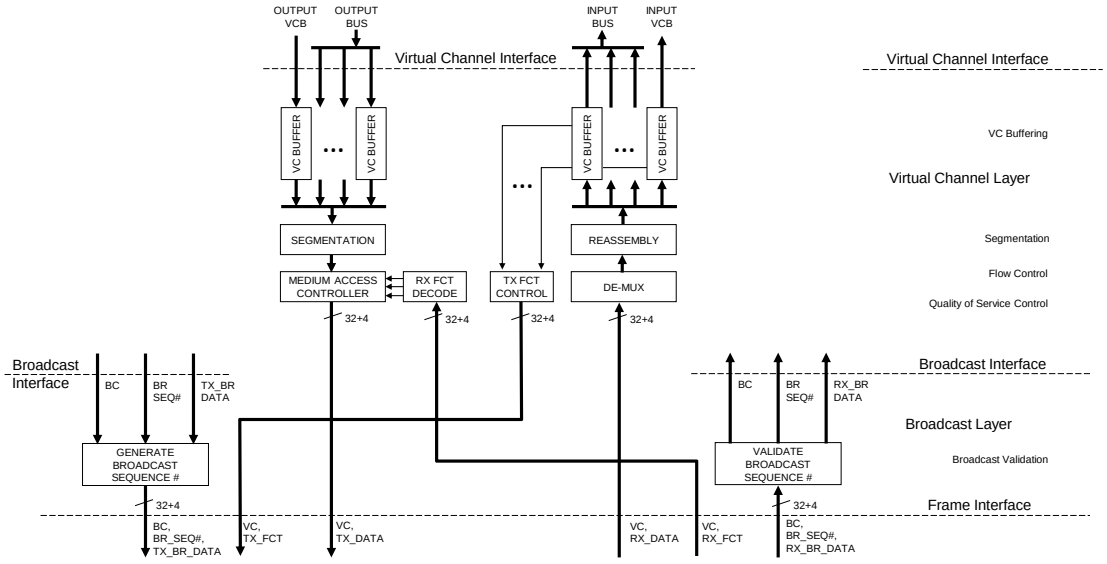


Figure 2 SpaceFibre Virtual channel and Broadcast Frame Layers

There are two different types of application interface to the SpaceFibre CODEC: the virtual channel interface used to send and receive SpaceWire packets, and the broadcast channel interface used to broadcast short messages across a SpaceFibre network and to receive those broadcast messages.

The virtual channel interface to the SpaceFibre CODEC comprises a number of virtual channel buffers (VCBs) for sending SpaceWire packets (output VCBs) and the same number for receiving SpaceWire packets (input VCBs). The output VCB

interface is used to send SpaceWire packets. Conceptually, each output VCB has a FIFO type interface which can accept SpaceWire data characters and EOP markers. To send a SpaceWire packet over a SpaceFibre virtual channel, the SpaceWire packet destination address and cargo are loaded sequentially into the appropriate output VCB, followed by an EOP. The specific interface to the VCB is application dependent. Interfaces to the input VCBs are used to read SpaceWire packets that have been received over the corresponding SpaceFibre virtual channel. Each input VCB has a FIFO type interface, from which SpaceWire data characters and EOP markers can be read.

The broadcast channel interface to the SpaceFibre CODEC is used for sending and receiving broadcast messages. These messages can be used for various functions including the distribution of time and synchronisation information, network management, and signalling events that occur in the nodes on the network. The broadcast interface comprises a set of registers for writing the parameters of a broadcast message (broadcast channel, broadcast sequence number, and the message) and a similar set of registers for reading received broadcast messages.

4 VIRTUAL CHANNEL LAYER

The virtual channel layer is responsible for quality of service and flow control over the SpaceFibre link. It controls the quality of service related to delivery of SpaceWire packets.

4.1 VIRTUAL CHANNEL BUFFERING

The output virtual channel buffers (VCBs) are used to buffer SpaceWire packet data before that data is sent over the SpaceFibre link. Data is sent in frames containing up to 256 SpaceWire N-Chars (data characters, EOPs or EEPs). The output VCBs permit this amount of data to be buffered before it is offered for transfer over the SpaceFibre link. Sending the data in frames and buffering data prior to framing, permits efficient interleaving of many SpaceWire packets travelling through different virtual channels over the SpaceFibre link.

The input VCBs provide a similar function for the reception of data arriving over the SpaceFibre interface. An input VCB provides storage for at least one maximum size data frame to ensure that when it arrives there is room for all the data it contains. The application using the SpaceFibre CODEC can then read data from the input VCB at its leisure, without causing loss of data on the SpaceFibre link.

4.2 SEGMENTATION

Data is sent over the SpaceFibre link in a series of data frames which each contain up to 256 N-Chars. Data in the output virtual channel buffer is segmented for sending into data frames. Data from received data frames are reassembled to form the original data stream which is placed in the input virtual channel buffer.

4.3 FLOW CONTROL

To manage the flow of data from all of the virtual channels across the SpaceFibre link it is necessary to know which output VCBs have data to send at one end of the link,

and which input VCBs have space for more data at the other end of the link. Exchange of this information is performed with credit based flow control: by exchanging flow control tokens (FCTs) for data frames. The input VCBs are monitored to determine when they have space for another maximum-sized data frame (up to 256 N-Chars). An FCT is sent to the other end of the link when a particular input VCB has space for another data frame. When the FCT is received at the other end of the link it permits another data frame to be sent over the corresponding virtual channel.

The potential loss of FCTs (along with data frames and broadcast frames) is handled by a retry layer, which ensures that FCTs cannot be lost unless the link suffers a permanent failure, in which case it is not possible to use that link any more.

4.4 QUALITY OF SERVICE CONTROL

A medium access controller determines which output VCB is allowed to send data over the SpaceFibre link. This depends on several things:

- Which output VCBs have data to send;
- Which input VCBs at the other end of the SpaceFibre link have space available to receive data, indicated by the reception of one or more FCTs for that virtual channel;
- The arbitration or quality of service (QoS) policy in force for each virtual channel.

For SpaceFibre several quality of service policies are provided:

- Fair arbitration, where each channel has an equal opportunity to use a link;
- Priority, where the virtual channel with the highest priority goes first;
- Bandwidth reserved, where the virtual channel with allocated bandwidth and recent low utilisation of the link will go first;
- Scheduled, where time-slots are defined by broadcast messages, and the virtual channel allocated to the current time-slot is permitted to send data. If this virtual channel has no data to send then another virtual channel may use this unused bandwidth opportunistically.

When a virtual channel has data to send in its output VCB and has room for more data in the input VCB at the other end of the SpaceFibre link, it competes with other virtual channels in a similar state. The virtual channel permitted to send a frame of data will be the one with the most urgent need to send data according to the QoS policies of all the competing virtual channels. The virtual channel layer then passes a frame of data containing up to 256 N-Chars from the selected output VCB to the framing layer for sending over the SpaceFibre link.

5 BROADCAST LAYER

The broadcast layer is responsible for broadcasting short messages across a SpaceFibre network and for receiving and checking those messages.

5.1 BROADCAST MESSAGES

A broadcast message is a short message that is sent by a node to all the other nodes on the SpaceFibre network. Broadcast messages propagate in a similar manner to SpaceWire time-codes. Each broadcast message contains a broadcast sequence number which is incremented each time a new broadcast message is sent. When a broadcast message arrives at a SpaceFibre receiver it is checked for errors and its broadcast sequence number is validated by comparing it to the broadcast sequence number of the last broadcast message received with the same broadcast channel number. The broadcast message is valid if its broadcast sequence number is one more than that of the previous broadcast message received. Only valid broadcast messages are passed out of the SpaceFibre CODEC. A SpaceFibre router will forward the broadcast message out of each of its other SpaceFibre links except the one that the broadcast message was received on.

5.2 BROADCAST CHANNELS

SpaceWire permits one set of time-codes to be broadcast, although by using the two flags in the time-code it is possible to have four independent sequences of time-codes operating concurrently. SpaceFibre broadcast messages permit up to 256 independent sequences of broadcast messages each of which is referred to as a broadcast channel. Each broadcast channel has a broadcast channel identifier and its own broadcast sequence number.

The broadcast channels are split into three types:

- 0-31: Network management broadcast channels.
- 32-253: Node broadcast channels, with each broadcast channel associated with a node that has a logical address of the same value as the broadcast channel number.
- 254, 255: Reserved broadcast channels.

The network management broadcast channels are split into two sub-types

- 0-3: Time synchronisation, which are used to provide regular and fault tolerant distribution of system time over the SpaceFibre network.
- 4-31: Network control, which are used to support configuration, control, and FDIR of a SpaceFibre network.

5.3 BROADCAST MESSAGE TYPES

Broadcast messages also carry 8 bytes of data, the first byte of which is a broadcast type field which determines the meaning of the remaining 7 bytes of data. For example, when type = TIME, the following seven bytes contain seven bits of time information. A broadcast message over one of the time synchronisation channels, would typically be of type TIME and the seven data bytes would contain a system time value (un-segmented time).

Typically a particular broadcast channel will be used by a specific node to broadcast information to all other nodes on the SpaceFibre network. This can be used to signal events that occur in that node to other nodes on the network. Each node can broadcast over a different broadcast channel.

A user application of a SpaceFibre CODEC can subscribe to receive broadcast messages from specific broadcast channels and of specific broadcast type. In this way the application will only be notified and receive those broadcast messages that it is interested in.

6 CONCLUSION

SpaceFibre can provide data rates of 2 to 5 Gbits/s over a single lane, depending on the SerDes device or technology used, and data rates in excess of 10 Gbits/s or more over multiple lanes. It can be operated over optical fibre covering distances of more than 100 m, or over electrical wires using current mode logic (CML) drivers. SpaceFibre provides galvanic isolation which is very important for fault isolation. It uses 8B/10B encoding and provides a complete range of quality of service (QoS). It supports FDIR with data frames, which contain a CRC checksums and frame sequence number, and a retry mechanism which automatically recovers from transitory errors without loss of information flowing over the link. SpaceFibre provides up to 256 virtual channels, and 256 broadcast channels. The broadcast channel provides low latency signalling capability over a SpaceFibre network, supporting time-distribution and event indication.

The proposed SpaceFibre standard has a number of benefits compared to SpaceWire:

- Virtual channels to overcome the SpaceWire router blocking problem;
- Broadcast messages to provide low latency messaging, based on an extension of the SpaceWire time-code mechanism;
- Coherent quality of service mechanisms to support deterministic data delivery for command and control applications and bandwidth sharing for payload data-handling applications;
- FDIR support including galvanic isolation, error detection, and error recovery to prevent fault propagation and to provide rapid recovery from transient faults;
- Lanes for higher throughput with graceful degradation and hot and cold redundancy support;
- QoS in the CODEC which provides inherent robustness against a range of system errors, like babbling idiots.

It achieves these benefits while remaining fully compatible with SpaceWire at the packet and network level, allowing ready migration of past applications to SpaceFibre.

7 ACKNOWLEDGEMENTS

The author would like to thank ESA for its support on several projects related to SpaceFibre: ESA contract number 17938/03/NL/LvH, call-off #2, “SpaceFibre”, ESA contract number 4000102641 “SpaceFibre Very High Speed Link Demonstrator”, ESA contract number AO/1-5975/08/NL/LvH “Next Generation Mass Memory

Architecture" led by Astrium GmbH, and ESA contract number 4000102660 "High Processing Power Digital Signal Processor", led by Astrium Ltd.

8 REFERENCES

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

[2] S.M. Parkes. C. McClements and M. Dunstan, "SpaceFibre Outline Specification", University of Dundee, 31st Oct 2007.

[3] S.M. Parkes. C. McClements, M. Dunstan and M. Suess, "SpaceFibre: Gbit/s Links For Use On board Spacecraft", International Astronautical Congress, Daejeon, Korea, 2009, paper IAC-09-B2.5.8.

[4] S.M. Parkes, " SpaceFibre", Draft B standard specification, to be published January 2012.

[5] W.J. Dally and B. Towles, "Principles and Practices of Interconnection Networks", ISBN 0-12-200751-4, Morgan Kaufmann Publishers, 2004.

[6] J. Duato, S. Yalmachili, & L. Ni, "Interconnection Networks An Engineering Approach", ISBN 1-55860-852-4, Morgan Kaufmann Publishers, 2003.

RAPIDIO OVER SPACEWIRE: BLENDING COMPLEMENTARY PROTOCOLS

Session: Networks and Protocols

Long Paper

Steve Belvin

Honeywell International, Clearwater, Florida, USA

E-mail: stephen.belvin@honeywell.com

ABSTRACT

SpaceWire defines a high-speed interconnect standard for on-board communications consisting of serial point-to-point links and switches. The protocols used are selected based on the application. Some protocols have been standardized and others are under study for standardization. Many of the services desired by users are available from a single protocol that is part of a full-featured commercial interconnect standard. RapidIO defines a common transport layer protocol that is independent of the physical implementation and provides a transport mechanism for several logical layer protocols. This paper describes the use of RapidIO common transport and logical layers over SpaceWire packets. After presenting details of the approach and adaptations, some benefits of using RapidIO over SpaceWire networks are discussed, along with some observations of fundamental differences between the two protocols. Work planned to further the definition is also provided.

1 INTRODUCTION

As Space systems increase in complexity and diversity, the communications systems must be capable of handling more and more capacity while being flexible enough to carry vastly different payloads. The integration of processing control functions, for instance, adds additional requirements on communications systems that can only be met with tighter integration with processing subsystems and flexible payload carrying capabilities with low processing overhead. Communications systems must provide a migration path from existing communications architectures to ones that can meet the demands of new subsystems. To do this, communications systems must offer a range of services that include, but are not limited to, support for the following.

- Efficient bandwidth utilization. The overhead of the packets should be low.
- Effective bandwidth sharing. Sharing of the network must be enforced through limited packet size and packet priority.
- Varying payload sizes. The payload requirements of channels may vary from very small to streaming.
- Low latency. The latency requirements of channels may be uncontrolled, deterministic or as low as possible.

- Reliable data delivery. Data delivery reliability may vary from best effort to assured delivery.
- Multiple communications mechanisms. The services of the network should be available from multiple communications mechanisms, including simple reads and writes, doorbells, messages and streaming.

SpaceWire [1] offers a versatile interconnect standard for on-board communications consisting of medium-speed (2 to 400 Mbps), duplex, point-to-point, serial data links between nodes. Routers are used to interconnect nodes in a network. Nodes are connected using a simple packet stream service and control characters are used to manage the flow of data. Restrictions on the packet length and mechanisms for reliable delivery are not defined. In order to introduce additional services to the network, users select upper level protocols with the desired services. A protocol identifier is used to distinguish between the various protocols. One standardized protocol offers simple reading and writing memory of a remote node, as well as network configuration and node control. Another protocol encapsulates CCSDS packets into SpaceWire packets.

An internationally certified interconnect standard called Serial RapidIO [2] offers high-speed (1.25 Gbps to 6.5 Gbps), duplex, point-to-point, serial data links between nodes. Similar to SpaceWire and other switch fabric interconnects, it uses switches to interconnect nodes. The standard is defined in three layers: physical, transport and logical. A component from each layer must be present to communicate. The physical layer defines the electrical signalling, 8B/10B coding, packet and flow control characters, packet transport mechanisms and link-level error management. The transport layer defines the node addressing and packet priority information. The logical layer defines the packets formats and how they will be used to transfer information. The common transport layer supports multiple physical and logical layer definitions.

With the help of an adaptation layer, the RapidIO common transport and logical layers may be used over SpaceWire (as a physical layer) to form a RapidIO over SpaceWire endpoint protocol stack. The RapidIO adaptation layer bridges the two protocols without requiring alterations to either one. The result is a method of reliably exchanging information between RapidIO endpoints using SpaceWire networks. This paper defines an approach to combining the two interconnect standards and identifies features that would improve the quality and reliability of the result.

2 COMMUNICATIONS LAYERS

In order to understand the services users need, we take a brief look at the common protocol stacks used by SpaceWire and RapidIO. The services provided by the SpaceWire protocols are compared with those available from RapidIO. This provides the foundation for discussing the RapidIO adaptation layer features needed to support these services. It also leads us into the presentation of a communications architecture based on the RapidIO over SpaceWire protocol stack. The discussion that follows is not intended to be a complete introduction to the protocols discussed and references are provided for more details.

2.1 SPACEWIRE PROTOCOL LAYERS

SpaceWire provides an interconnect standard in support of transferring payload data and control information. The ECSS-E-ST-50-12C [1] standard provides definition of physical, transport and logical layers in support of a stream service using First-In, First-Out (FIFO) buffers at the transmitter and receiver as the typical structure for data buffering. Link level flow control mechanisms are defined to avoid FIFO overflow errors at the receiver. No negotiation method is provided to reserve data storage or processing at the receiver. Path routing using router output ports and logical addressing using unique end point logical addresses are defined. Extended logical addressing is defined for packets that move between network regions. Error detection and reporting is defined but reliable delivery of data is not.

Various protocols may be used to form a SpaceWire network. A set of protocol standards, referred as the ECSS-E-ST-50-5x series, define some these protocols while others like the General Access Protocol (GAP) are defined and managed by other organizations. The ECSS-E-ST-50-51C [3] standard defines a protocol identifier header used to identify the protocol that constructed a packet. Two examples of these protocols are Remote Memory Access Protocol (RMAP) and Consultive Committee for Space Data Systems (CCSDS) Packet Transfer Protocol (PTP). A diagram showing these two protocol layers is provided in Figure 1.

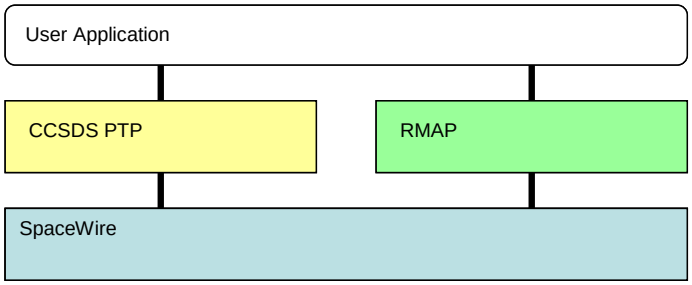


Figure 1. SpaceWire Protocol Stack

The RMAP defined in ECSS-E-ST-50-52C [4] is a protocol that works over SpaceWire to provide a mechanism for reading and writing memory of a remote node, as well as network configuration and node control. Memory addresses must be 32-bit aligned and the length must be a multiple of four bytes. Responses to memory read or write requests are optional and contain the operation status. Partial implementation of RMAP operations is permitted. Both SpaceWire and RMAP offer best effort delivery. Error detection is provided but there is no defined mechanism for recovering lost or erroneous data. Verification of network operations results in wasted bandwidth and user application processing.

CCSDS packets may be transferred over SpaceWire networks using the CCSDS Packet Transfer Protocol. As defined in ECSS-E-ST-50-53C [5], variable length CCSDS packets are encapsulated in a SpaceWire packet. As with RMAP, both SpaceWire and CCSDS PTP do not offer specific quality of service. The timing and reliability of delivery are not part of the services provided.

2.2 RAPIDIO PROTOCOL LAYERS

The RapidIO Common Transport Specification [6] provides logical addressing of the source and destination endpoints in the form of Source and Destination Identity fields. These fields may be either 8 or 16 bits in length. Path routing is not supported so

network switches maintain routing tables. The logical layer may consist of one or more protocols defined and managed as part of the overall RapidIO specification. Payloads range from 1 byte to 256 bytes. Response packets are very similar to request packets. Lost packets are detected through a packet timeout counter and retransmitted.

Three examples of RapidIO logical layer protocols as shown in Figure 2 are the Input/Output (I/O) Logical Layer, the Message Passing Logical Layer and the Data Streaming Logical Layer. In addition to the logical layer protocols, extensions such as error management and flow control are provided. Where additional information or guidance is needed, it is defined in the form of annexes such as system initialization and session management.

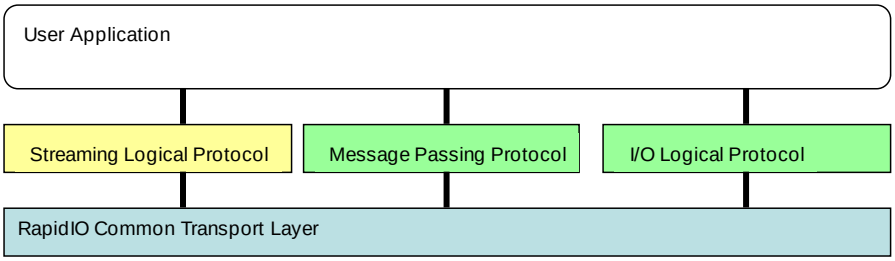


Figure 2. RapidIO Upper Layer Protocols

The RapidIO I/O Logical Layer Specification [7] defines packets used for performing read, write and read-modify-write operations that are independent of the bandwidth or latency of the physical layer. Varying data sizes from very small (byte granularity) to Direct Memory Access (DMA) style operations are supported, with a local address of 34, 50 or 66 bits. Efficiencies greater than 90% are possible using 256 byte payload writes [8]. Partial implementation of the I/O Logical Layer is permitted.

In multiple processing systems where access to address space is not desired, RapidIO defines a messaging service in hardware in the Message Passing Logical Layer Specification [9]. The messaging service defines messages and doorbells (which are equivalent to Message Signalled Interrupts). Messages are handled by hardware mailboxes at the destination. Messages up to 4-kByte are supported and, where payloads exceed 256 Bytes, segmentation and reassembly are supported in hardware. A message may consist of up to 16 packets. Messaging supports a reliable, efficient means of communicating between processing systems and is commonly found in RapidIO networks.

Data streaming supports data plane applications with multiple protocols that are sensitive to latency and less sensitive to loss. The Data Streaming Logical Specification [10] supports Protocol Data Units (PDUs) of lengths from 1 byte to 64-kByte with segmentation and reassembly in hardware. Hundreds of traffic classes and thousands of data streams are supported, allowing multiple PDUs to be transferred between a source and destination at one time. Data streaming is a relatively new logical layer definition and is not supported in legacy systems.

2.3 RapidIO Adaptation Layer

The RapidIO Adaptation Layer allows the use of RapidIO common transport and logical protocol layers to be used over the SpaceWire protocol layer defined in ECSS-E-ST-50-12C [1]. With this added layer, SpaceWire networks and RapidIO networks may be connected to form a larger, fault-tolerant network that provides a full set of features and services. In order to connect SpaceWire and RapidIO networks together, the RapidIO Adaptation Layer must perform the specific functions.

Support SpaceWire Protocol Identifier. The use of RapidIO common transport layer will be identified in the Protocol Identifier as defined in ECSS-E-ST-50-51C [3]. Protocol Identifier values are assigned by the SpaceWire Working Group.

Network Address Mapping. The network address must be mapped from the source protocol to the target protocol. RapidIO supports logical addresses and does not support path addressing. Path addressing requires every host to know the path to every other host. The use of path addressing is not desired in large, extensible networks. While network-unique logical addresses are preferred, path addressing is also supported. Where 16-bit logical addresses are used, the SpaceWire extended address will be required. Support for regional logical addressing has not been defined.

Packet Delivery Handshake. A packet handshake mechanism is used to insert packets into the fabric and extract them from the fabric.

Packet Integrity Check. A link level error checking feature is used to ensure packet integrity. This involves adding a CRC at the transmit side and deleting it at the receive side. Erroneous packets should be retransmitted.

Packet and Transaction Delivery Controller. Packet delivery ordering rules must be maintained based on priority, especially under exception processing such as retries.

End-to-end Acknowledgement Counter. An end-to-end level timeout counter is used to detect the lack of acknowledgement on the link and is treated as an acknowledgement error.

Control and Status Registers. Registers that control the functions and provide status are required. These registers should support the features of the RapidIO physical layer to be compatible with the RapidIO registers.

3 COMMUNICATIONS PROTOCOL STACK

In order to design a communications architecture, multiple views are required. One view is the protocol stack for a single endpoint including software drivers and the communications application programming interface. The RapidIO over SpaceWire endpoint communications architecture is shown in Figure 3. Since most onboard interconnects consist of multiple processors running both homogeneous and heterogeneous hardware and operating systems with synchronization mechanisms that manage control and data traffic, a bias toward a message passing interface is given. Initial support of streaming in legacy systems is expected to be using message passing. As time goes on, systems will begin to fully support streaming.

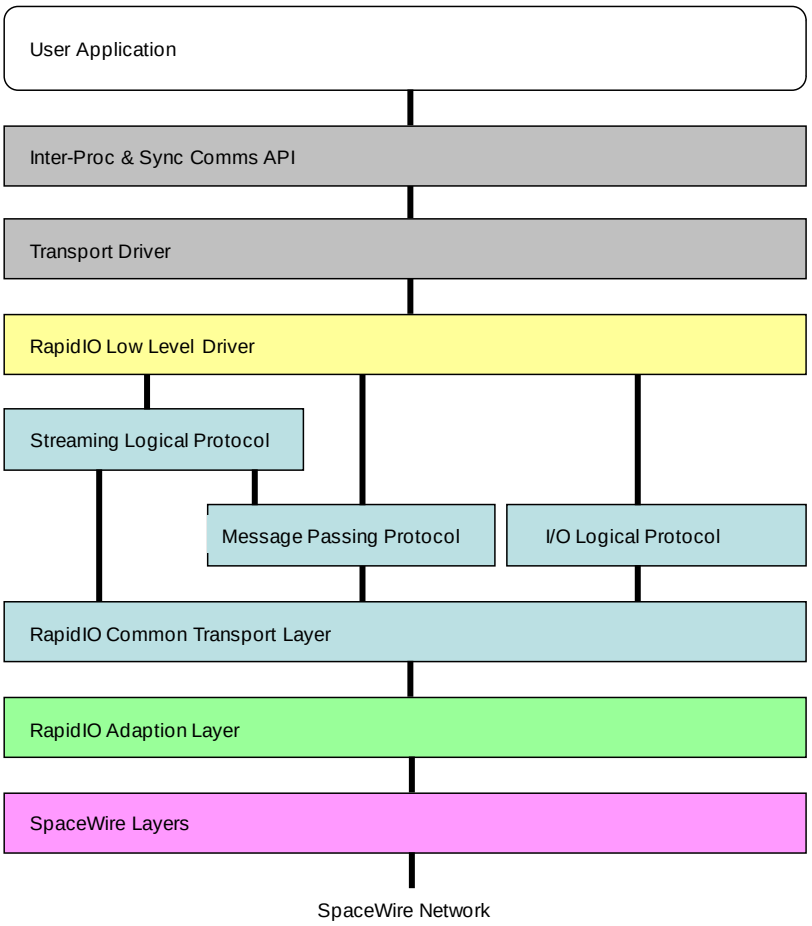


Figure 3. Complete RapidIO over SpaceWire Protocol Stack

3.1 RapidIO Over SpaceWire Benefits

Some of the benefits that RapidIO over SpaceWire offers embedded systems are listed below.

Complementary Protocols. The definition of RapidIO into three protocol layers, the independence of the common transport layer from the physical layer and the low protocol overhead make RapidIO a good choice for upper layer protocols. The multi-gigabit transmission rates, complex physical layer features and extreme environments of applications combine to make the effective cost of implementation of the RapidIO physical layer very high. SpaceWire with its lower transmission rates and lightweight physical layer allow for implementations with significantly lower area. The error control mechanisms of RapidIO help make up for the lack of low-level error control.

Low protocol overhead. The low protocol overhead of RapidIO keeps network utilization high.

Future Growth. RapidIO was developed by the RapidIO Trade Association consisting of over 30 members from industry leaders to small companies. RapidIO over SpaceWire uses the common transport layer, providing access to all logical layers defined by the standard.

Ease of Implementation. By keeping as many of the protocols compatible with their respective standards, intellectual property may be used without modification for nearly all of the RapidIO and SpaceWire functions.

3.2 SUGGESTIONS FOR IMPROVED INTEGRATION

When comparing the services offered by RapidIO and SpaceWire, the following differences were noted.

- Error recovery mechanisms. Error sources in SpaceWire networks include header or data errors, loss of packet and erroneous packet detection with EEP appended. SpaceWire reports errors but does not have reliable delivery at the link level. RapidIO supports reliable delivery at the link level by sending retry control symbols back to the sender on erroneous packets. Timeout counters detect loss of a packet and invoke retransmission. Note that the RapidIO Adaptation Layer adds the RapidIO error detection and retries for end-to-end reliable packet delivery. It also provides for reporting these errors.

- Packet Prioritization. Links carry packets of varying priority. RapidIO supports four priority levels for packet ordering and deadlock prevention. Switches use the priority filed to make output port arbitration decisions. SpaceWire does not support packet prioritization at the routers. Packet-based priority supports making response packets of higher priority than request packets and some data streams from the same sender higher than others. The RapidIO Adaptation Layer may support prioritization of packets delivered to the SpaceWire network for each logical flow.

- Flow control. RapidIO supports high level flow control using XON and XOFF at the logical layers to manage congestion. Also, receiver buffer status feedback allows transmitters to control data flow. These services are not available in SpaceWire.

- Segmentation and reassembly. Segmentation supports many data flows. Out-of-order reassembly reduces network traffic by allowing selective retries. RapidIO message passing supports hardware based out-of-order reassembly.

- Error management extension. RapidIO provides an error management extension to the basic protocol that provides added error condition and devices status reporting. This extension is useful to many onboard applications.

- Extensions for specific applications. RapidIO provides extensions for specific types of network users such as synthetic aperture radar applications.

4 FUTURE WORK

The RapidIO over SpaceWire protocol stack presented here focused on data and control communications. It did not cover some important aspects of defining a complete communications solution. Areas of further study include the following.

- Refinement of the RapidIO Adaptation Layer functions.
- Improved network addressing definition.

- Initialization and maintenance of the network.
- Distribution of time information.
- Communication channel types (asynchronous, connected) for API definition.
- Support for SpaceWire Plug-and-Play.

The current effort is focused on developing a model a SpaceWire network with an endpoint that includes a RapidIO protocol stack. After refining the SysML models, simulation in SystemC will be used to further define the functions and incrementally add features of the RapidIO Adaptation Layer.

5 REFERENCES

1. ECSS, "Space Engineering: SpaceWire - Links, nodes, routers and networks", EECS-E50-12C, 31 July 2008.

2. RapidIO Trade Organization, "RapidIO Interconnect Specification Part 6: LP-Serial Physical Layer Specification", Revision 2.2, June 2011.

3. ECSS, "Space Engineering: SpaceWire protocol identification", EECS-E50-51C, 5 Jan 2010.

4. ECSS, "Space Engineering: SpaceWire - Remote memory access protocol", EECS-E50-52C, 5 Jan 2010.

5. ECSS, "Space Engineering: SpaceWire - CCSDS packet transfer protocol", EECS-E50-53C, 5 Jan 2010.

6. RapidIO Trade Organization, "RapidIO Interconnect Specification Part 3: Common Transport Layer Specification", Revision 2.2, June 2011.

7. RapidIO Trade Organization, "RapidIO Interconnect Specification Part 1: Input/Output Logical Layer Specification", Revision 2.2, June 2011.

8. RapidIO Trade Association, "RapidIO, PCIExpress and Gigabit Ethernet Comparison: Pros and Cons of Using Interconnects in Embedded Systems", Revision 3, May, 2005.

9. RapidIO Trade Organization, "RapidIO Interconnect Specification Part 2: Message Passing Logical Layer Specification", Revision 2.2, June 2011.

10. RapidIO Trade Organization, "RapidIO Interconnect Specification Part 10: Data Streaming Logical Layer Specification", Revision 2.2, June 2011.

SpaceAGE Bus: Proposed Electro-Mechanical Bus for Avionics Interconnections

Session: Networks and Protocols

Long Paper

Glenn P. Rakow, Eric T. Gorman

NASA Goddard Space Flight Center, Greenbelt, Maryland, USA

Alexander B. Kisin

MEI Technologies / NASA Goddard Space Flight Center, Greenbelt, Maryland, USA

E-mail: Alexander.B.Kisin@nasa.gov Glenn.P.Rakow@nasa.gov Eric.T.Gorman@nasa.gov

ABSTRACT

This paper will describe a proposal for a standard that is being solicited to national space agencies, US government agencies, international industry and academia for feedback in order to develop consensus for an intra-box electrical and mechanical Printed Wiring Board (PWB) interface.

This PWB standard will be the building block element to develop avionics boxes and systems for a wide range of requirements and will show advantages over the current traditional approach. Firstly, the proposed standard requires no strict connector tolerances typical of backplane designs but rather the boards are cabled together externally using matched impedance, shielded, blind mating connectors. Secondly, the standard defines a serial communication physical interface that can support many popular protocols. Thirdly, the mechanical chassis design is not dependent of the number of PWBs required, as each PWB(s) integrates a portion of the overall mechanical box chassis design. Fourthly, the PWBs are inherently EMI and thermally compatible with each other as isolation exists in these realms allowing random integration of different cards within a box without the need for additional box level qualification. And lastly, the standard defines a common module (or HUB) that provides all the typical common functions for a suite of PWB modules in an avionics box to reduce the overhead if each PWB had to provide their own. These functions include the intra-box communications HUB, inter-box communications interfaces, primary power isolation and secondary power switching to the cards within the box; and the computational capabilities.

It is expected that suppliers of hardware built to this specification will allow avionics systems to be more rapidly architected and constructed to support centralized and distributed Integrated Modular Architecture (IMA) applications, allowing the leveraging of products across the entire aerospace community. This architecture will also support the distribution of software tasks across multiple processors if desired. The proposed standard is applicable to 90% of space missions and is targeted to habitable, as well to all classes of robotic spacecraft; the only class where this standard may not be useful are nano-satellite applications where high level of integration for optimization is required.

1 INTRODUCTION

The SpaceAGE Bus is intended to specify a standard method of integrating Printed Wiring Board (PWB) level building block modules into avionics boxes, i.e. focus on intra-box interfaces only. The box-to-box or inter-box interfaces for the system level defines the architecture and any architecture can be supported with the SpaceAGE Bus building blocks, similar to Lego's building different structures.

The SpaceAGE Bus was closely designed with spacecraft software architects to provide many unique benefits, including distribution of software, based upon the NASA/GSFC component based software design, core Flight Executive (cFE) using a Hypervisor on the HUB (common module) processing resources to time-space partition NODE related software functions. This approach has been developed to support the Distributed Integrated Modular Avionics (DIMA) architecture.

The SpaceAGE Bus defines intra-box (backplane) interfaces and not protocols, which allows the bridging of different protocols at one point (HUB) without significantly affecting the design. Interfaces are complete for a broad range of intra-box applications and minimized through serial communication and single voltage distribution.

Great attention was spent upon the mechanical aspects to reduce tolerances and provide flexibility in module height. Elimination of unique backplane designs and mechanical chassis that drive Non Recurring Engineering (NRE) through development and integration were met, as well as providing for isolation between modules both thermally and for EMI.

2 INTRA-BOX ELECTRICAL INTERCONNECTS

Currently a majority of space avionics data buses for CPU based Control and Data Handling systems (C&DH) are just commercial standards, which utilize parallel half duplex data buses (VME, cPCI, etc.), and low voltage power distribution. For space systems where Space, Weight and Power (SWaP) and efficient use of resources are important, better backplane implementations are possible.

Disadvantages of the current backplane implementations include: inability for concurrent bus operations; power cross talk among loads; high Common Mode Voltage (CMV) for single ended signals; incompatibility for reuse with other systems because of user defined signals; limited distances between boards and limitations of board height, which greatly effects connector real estate; difficulty in fault isolation between modules; and difficulty in EMI shielding.

A classical diagram of parallel bus is shown in Figure 1 below.

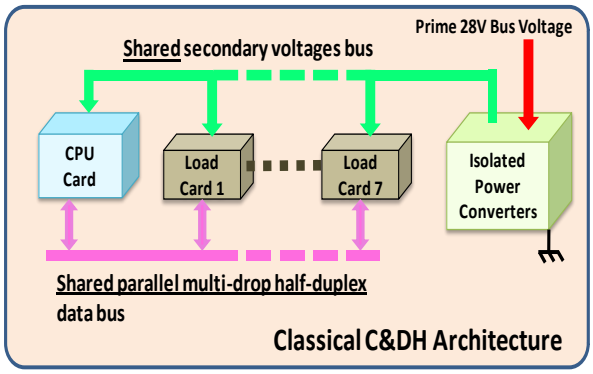


Figure 1 – Classical C&DH Bus Architecture

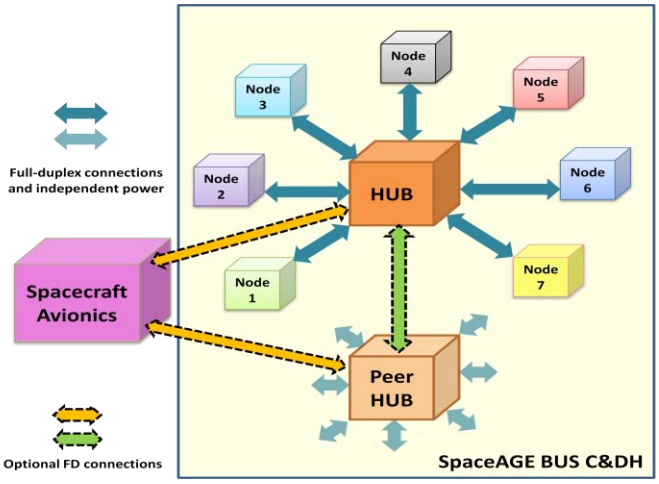


Figure 2 – SpaceAGE Bus C&DH Architecture

3 SPACEAGE BUS OVERVIEW

The SpaceAGE Bus will provide a low SWaP alternative to classical intra-box (backplane) buses used in today Spacecraft (S/C) avionics. It will eliminate shared parallel buses and low voltage distribution, and instead will provide each load with a dedicated high speed (Gigabit) serial full duplex differential

interconnect, and will distribute a single higher voltage (as compared to digital) to reduce load currents and system CMV. The ability to isolate power between modules will greatly reduce internal system crosstalk.

A suggested SpaceAGE Bus diagram is shown in above Figure 2.

3.1 MODULE TYPES

The whole system is designed using a star-like architecture: central part, called HUB, and peripheral part, called NODE. Each module will contain 1 or 2 of 6U 160mm size cards, dependent upon application, which through a trade study was found to be generically the optimal size for space electronics.

Theoretically, the SpaceAGE Bus does not restrict number of PWBs per module, but if more real estate is necessary, the use of a daughter card is the most straightforward implementation. Also the modules height (thickness) is not specified to accommodate connector real estate and tall components.

All PWBs are integrated into their module's card frame, which has direct thermal heat path to base plate. As an option, the card frame has the ability to be 100% EMI shielded. Only the intra-box interconnects, HUB-NODE and HUB-HUB are defined by the SpaceAGE Bus, which occupy one side of the modules card frame and each module has 2 sides for user defined interfaces.

3.2 HUB DESCRIPTION

The HUB contains the common functions for each avionics box, i.e., backplane and the common external box interfaces. Depending upon the redundancy requirement there may be 1 or 2 HUBs per avionics box. Specifically, the HUB provides the following functions: primary data processing capabilities (micro-controller or micro-processor), intra-box (backplane via hub) and external links (vehicle control bus) communications, power distribution for each NODE's internal functions, analog telemetry conversion for itself and all NODEs, and finally, clock synchronization and distribution. In general, it has 2 PWBs, one digital that resembles a Single Board Computer (SBC), and one analog for power distribution and analog conversion. It will be packaged in a dual PWB enclosure and consume between 5 - 20W depending performance.

3.3 NODE DESCRIPTION

The Node contains user specific functions required by the avionics box. The collection of nodes together with at least one HUB comprises an avionics box. Electrically, the SpaceAGE Bus defines for the NODE, the quantity and type of intra-box connectors. Mechanically, it defines the NODE's height and depth (board area), while width (height) dimension is flexible. All NODE intra-box interfaces are to the HUB. If NODE-to-NODE communications is required, the communications is switched via the HUB.

Power for a NODE other than power necessary for conversion to internal voltages is provided as an external user interface into the NODE, i.e., power switched by the NODE for pyro initiation, heater, or valves, etc., is provided from external source directly to NODEs user interface.

Although not specified by the SpaceAGE Bus, it is envisioned that the NODE modules will be partitioned along subsystem boundaries and contain multiple functions for a particular subsystem. For example if a distributed avionics system is architected, the SpaceAGE Bus may be used to implement a Remote Interface Unit (RIU). In a distributed avionics system, the RIUs would be placed close to the

effectors and sensors to which they interface in order to reduce harness mass. Instead of dedicating one (1) single subsystem functionality within a avionics box, a more efficient use of resources would be to allow different subsystem NODE modules to reside in the same avionics box thus reducing the overhead of HUB modules. This isolation of NODEs on subsystem boundaries is carried over to the software where a time-space partition (hypervisor) may run on the HUBs micro-controller (or processor) to protect the different subsystem functions from each other. This will especially be important for integration of the NODEs within a SpaceAGE Bus, especially if the number and type of NODE modules is not a priori known. The average power consumption of each NODE will vary greatly depending upon its function, e.g., memory versus heater driver, but the average power for the digital functions of the NODE is limited by the current carrying capacity of SpaceAGE power service, which is de-rated to 1.5 Ampere at between 16V to 40V bus voltages.

3.4 REDUNDANCY CONFIGURATIONS

Unlike commonly used commercial busses adapted for space applications, where redundancy is not inherently designed in, the SpaceAGE Bus is designed with redundancy architecture in mind. Two (2) types of hardware redundancy can be implemented at the box level: a) complete dual redundancy, where 2 independent strings of HUB/NODEs networks never intersect within each other, i.e., 2 HUBs per box and each NODE consists of 2 totally isolated identical entities; and b) cross redundant network also consisting of 2 HUBs and dual entity NODEs, with the addition of both HUBs talking to each other through crossed communication links and exchange clocks (Figures 2 and 3). A variation of cross redundant network is where the 2 HUBs are cross strapped to a single NODE where graceful degradation is acceptable. Both above intra-box redundancy schemes will require dual identical intra-box connectors per NODE.

Even for dual redundancy mode where HUBs are not cross-strapped to the other side's NODEs, cross-strapping between HUBs can be implemented because of the electrical, thermal and EMI isolation between HUBs: a failure of one side of a NODE will not jeopardize the mission because its mirror image entity will be able to communicate with its own HUB which will send all required information to a peer HUB which serves failed NODE. If redundancy is not required, then NODE may contain only 1 connector served by a dedicated HUB, thus reducing total cost, weight and power consumption.

3.5 INTRA-BOX NETWORK ARCHITECTURE

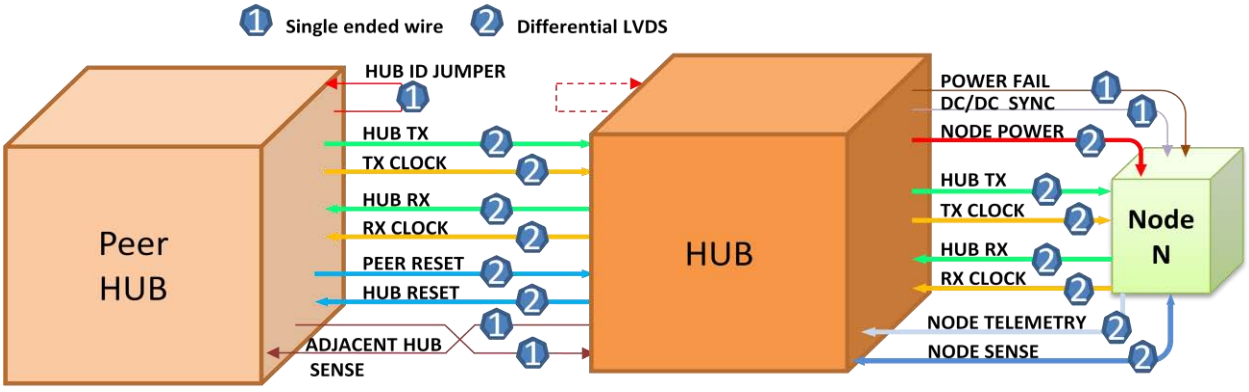


Figure 3 – SpaceAGE Bus HUB-to-HUB and HUB-to-NODE Connections

Complete network architecture will consist of the following signal domains (see Figure 3 and Appendix A for more details):

a) Power: The HUB will distribute 16-40Vdc power to the NODE for digital power regulation. The connector supports the ability to cross-strap power to redundant NODEs. The voltage was selected to be higher than digital voltages to reduce the current and provide greater tolerance to ripple.

The distributed voltage to each NODE can be isolated or non-isolated. If isolated, the DC/DC converter resides at each NODE. If non-isolated voltage is distributed to the NODEs (isolation at HUB) then switching Point-of-Load (PoL) and linear converters reside at the NODE. Consideration for current loops and CMV needs to be taken into account for the particular implementation. The initial prototype for the SpaceAGE Bus will use isolated DC/DC converters on each module.

Power distribution switches will be able to power up or down each NODE independently depending on operational requirements. The power capability supported is 1.5 A de-rated per NODE at 16-40V, but it is expected based upon the card frame thermal design that the power should not exceed 30W. If S/C bus uses higher voltages (e.g., 120V, etc.), there are several ways to handle this scenario. Either provide the isolating converter in the HUB, or provide a step-down converter external to the HUB.

b) Communication: Each HUB/NODE link consists of full duplex serial interfaces based on applicable user defined protocols. Only the physical layer is defined, which is 2 unidirectional differential signals in each direction. This scheme requires clock and data to be line encoded on the same signal, e.g., 8b/10b or Manchester, etc. However, synchronous clocked protocols are also allowed, using HUB clock.

The advantage of this approach is that it does not require all NODEs to conform to the highest conceivable signaling rate for the worst case application, so NODE application may be optimized. For example, if the NODE is just for driving valve solenoid coils, heaters, or gathering telemetry, etc. which is a low data rate application, then 1 Mbps Manchester encoding is sufficient. If another NODE is a memory module for data recorder that is connected to a Gigabit rate instrument then, an 8b/10b line encoding would probably be used with a protocol such as Gigabit Ethernet, Rapid IO or SpaceWire II as possible examples. The important point is the bridging between different communication protocols between NODEs (and to the external interface as well) is done at the HUB switch, which will most likely be implemented in a reprogrammable FPGA. It is expected that the communication interface will be capable of supporting Gigabit rates (up to 3.25 Gbps) with a defined number of services for lower rate interfaces, possible configurable as to the mix. It is expected that 1-2 Gbps LVDS interfaces will be supported (configurable via jumpers and re-programming).

c) Clock: The clock signals are sourced from the HUB to each NODE. Two (2) types of system clocks are defined by the SpaceAGE Bus. The first is a NODE non-dedicated clock with a frequency and function that is defined by the NODE end user. Examples for this clock may include communication clock between HUB and NODE, which can be used in conjunction with the communication interface (data signal); events synchronization clock, combination of both, etc. The clock frequency should not exceed 200 MHz. Because this clock function is user defined, it even can be used as an additional communication link from HUB to NODE.

The second, clock is a single frequency used to synchronize the NODE switching power converters. To reduce total EMI noise levels, converters for other NODEs can also be synchronized to this frequency. This synchronization frequency is common for all NODEs and will be defined by system requirements, depending on selected power converters. It is expected to vary from 200 to 800 KHz.

d) Analog Telemetry: All existing spaceflight adapted buses do not provide this function as standard, however it is required on the majority of S/C, thus forcing designers to create special

telemetry cards and uniquely customizing non-defined pins of standard commercial buses, while providing a low protection against environmental digital noise. The SpaceAGE Bus will have a solution for this by providing a standard differential analog interface. Each HUB will contain a filtered multichannel data acquisition converter system. It is expected to easily achieve an accuracy of 0.1% for most telemetry signals. Each NODE will get a single dedicated HUB telemetry channel. In its turn, each NODE shall contain either simple analog signal conditioner with built-in channel analog multiplexers, where each channel will be controlled by HUB-to-NODE command; or, if no NODE analog circuitry is desired, just a single thermistor (e.g. AD590): to report NODE's temperature. However, if NODE requires a full featured analog telemetry system with its own A/D and D/A converters, etc. – the SpaceAGE Bus architecture does not prohibit this either.

It is expected but not required that each HUB will provide a comprehensive range of internal telemetry, including measurements of each NODE's input bus voltage and consumption current.

e) Auxiliary: Three (3) types of auxiliary signals are provided by the SpaceAGE Bus, these include Reset, Sense, and Power Fail.

The **Reset** signal is sourced by the HUB to each NODE to perform a warm reset. The HUB may reset each individual NODE either by a certain operational condition (e.g., lack of time based NODE responses), or by an external command.

The **Sense** signal is sourced from each NODE to the HUB(s). The functionality of a Sense signal is as follows: the SpaceAGE Bus is designed to allow hot plugging and unplugging of any HUB or NODE without jeopardizing adjacent SpaceAGE Bus NODE or HUB module. This is achieved by implementing sense switches which will be mechanically engaged or disengaged by HUB/NODE special assembly torque bolt; this will "tell" the HUB's avionics to apply or cut power and other electrical links to a NODE module, which is already plugged and secured in its allocated slot, or about to be unplugged from its slot. This feature will be very desirable for human handling where avionics interchangeability may require continuous operation of other bus modules, as well as for integration and test (I&T) phase of avionics assembly process: to eliminate human errors by handling "hot" units.

The **Power Fail** signal is sourced from the HUB to each NODE and will notify the NODE that the input power is about to "die" in TBD μS and that NODE has to prepare itself for this situation. Because HUB may contain a non-volatile memory, each NODE may send HUB its important information for secure safekeeping.

3.6 HUBs CROSS CONNECTIONS

As previously mentioned, 2 HUBs may be used if a cross redundant architecture is required. Both HUBs will talk to each other using the HUB's manufacturer defined full duplex synchronous serial LVDS based protocol with data rates up to 200Mbps. This is considered to be a backbone communication link and is not end user re-programmable.

In addition, one of the HUBs, designated as Master, may supply its Sync clocks to a Slave HUB (defined by an externally placed harness jumper), so a whole C&DH assembly will be synchronized to one clock frequency and have single clock source for DC/DC converters. Both HUBs will also be able to reset each other either by functionally defined condition (e.g., lack of communication for a TBD period of time from an adjacent HUB), or by an external command.

A suggested routing of all interfaces is shown in Appendix B of this document.

3.7 CONNECTORS

The SpaceAGE Bus uses external harness for intra-box signal interfaces, instead of the more traditional PCW based backplane. This approach allows for quick design of avionics boxes from module building blocks because of the elimination of Non Recurring Engineering (NRE) cost associated with traditional design of backplanes, mechanical chassis and software to streamline integration. It eliminates additional volume and weight which is always associated with backplane PWB and take away distance between cards constraints. By using 100% EMI shielded metal shell connectors with protected pins inserts, thus creating a true Faraday chamber around harness, the need of an overall EMI box shield is eliminated too. As a base approach a ruggedized D-sub quadraxial connectors from Sabritec Inc. were selected.

The suggested connectors are quadraxial cable connectors containing two 100 Ohms impedance matched pairs per insert with the shield carried through the insert, which provides a continuous EMI shield for all signals.

In addition, by using AWG24 wires and having high isolation between pins, these inserts will also satisfy voltage and current distribution requirements for up to 50Vdc with 1.5A per pin after derating. Two (2) types of connectors will be used; 4 position quadraxial inserts (shown) for the NODE and 16 quadraxial inserts for the HUB.

3.8 HUB PORTS

a) Intra-box ports: Each HUB will contain 2 intra-box connectors with 16 quadraxial inserts each, for a total of 32 inserts: 28 of them to create 7 groups for a dedicated interface for up to 7 NODEs. The remaining 4 will be used for cross connections with a redundant peer HUB if redundancy is required. Cross HUB connections will be used for HUB high reliability operations. All unused NODE interfaces will be disabled by HUB.

b) External ports: Each HUB will contain the user defined connectors for the vehicle control bus used for spacecraft command and control. This interface is not defined by the SpaceAGE Bus. It will also contain the S/C power interface for module internal power (not pass through power).

c) Debug ports: It is recommended that each HUB contain a single debug connector (located on top of HUB unit), which will allow on-board FPGA testing and internal memory up/downloads.

	Back (SpaceAGE bus)	Front (S/C links)	Top (mostly for debug)
Number of ports	**8**: 7 NODEs + 1 Peer HUB	**6**: 4 + 2 SpaceWire	**2**
Physical Interface	Buffered LVDS or AC coupled CML SerDes		Buffered LVDS
Duplex	Full	Full	Full
Speed range	1Kbps to 3.125Gbps (up to 100Mbps for SpW)		10Kbps to 100Mbps
Additional Sync clock	Yes	No	No
Protocols	Any type sync or async	Any async + SpW	Async + 10M Ethernet
In-flight re-configuration	Yes (except Peer HUB)	Yes (except SpW)	Yes (if used for flight)
State when not used	Hi-Z	Hi-Z	Hi-Z
Multidrop network use	No	Possible (to 400Mbps)	No

Table 1. Proposed HUB Ports

3.9 NODE PORTS

Each NODE will contain at least 1 back side connector with 4 quadraxial inserts for single redundancy scheme; for double redundancy 2 connectors are required. Each connector will provide NODE with all required links with one of the HUBs. All user specified connectors (of any kind) may occupy empty area on top or front of a NODE module.

4 MECHANICAL OVERVIEW

4.1 GENERAL

The SpaceAGE mechanical design is aimed at reducing the Non-Recurring Engineering (NRE) costs of design through elimination of a unique PWB based backplane and mechanical chassis designs. It therefore allows modules to be assembled without prior knowledge of number and type. It also provides thermal isolation between modules and optional EMI isolation between modules.

The SpaceAGE chassis is comprised of individual Aluminum 6061 modules containing PWBs of Eurocard form factor 6U (160mm x 233mm) that are individually packaged and are fastened to a backplate assembly and to each other (Figure 4).

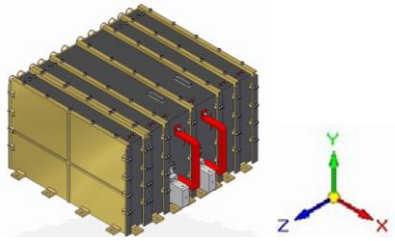


Figure 4 – SpaceAGE Assembly & Coordinate System

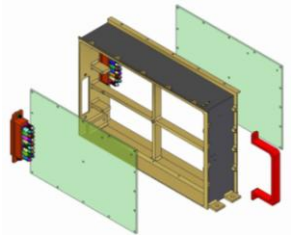


Figure 5 – Module w/2 PCBs Assembly

4.2 MODULE MECHANICAL DESIGN

Based on design and volume requirements the user can select a module that can house either one (1) or two (2) 6U 160mm PWBs. Each module contains an integrated stiffener design that provides structural rigidity as well as an additional heat path. The module contains additional space between the PWB mounting area and the outer wall, which allows additional room for either a flex PWB or floating lead connectors and the required wire harness. Modules that contain 2 PWBs have an integrated stiffener that is thicker. This allows PWBs to be bolted to both sides of the stiffener. All PWBs allow double sided component assembly. PWB assembly of the double module can be seen in above Figure 5.

The PWBs are mechanically fastened to the integrated stiffener using steel fasteners. The mechanical interface control drawing (MICD) for each PWB will be standardized and will contain the overall PWB dimensions, fasteners locations, hole sizes, component height restrictions, and p keep out areas. Each Module has the same form factor and is constrained in the X and Y coordinates. The Z coordinate is unconstrained. This allows the user to size each module in the Z direction based on PWB requirements and component height. It is recommended that modules be incremented in ¼ inch step in the Z direction. Figure 5 shows the integrated stiffener where the PCB is fastened to the module. The open space around the PWB allows extra room for connectors and when wiring floating I/O connections. In modules that house 2 PWBs, this extra space can be used to house cross-over cables or connectors that will allow communication between cards inside of the module so the user doesn't have to sacrifice external connector panel space.

The top and the front panel are reserved for user connectors. The user connector selection and their locations is user defined. The user is responsible for controlling the locations of these connectors and capturing this information in their own MICD. The intra-box connector locations are standardized and will be controlled in the module level MICD. Users may not place their connectors on the back side of the module. This area is reserved for the interface to the intra-box (backplane) assembly. Figure 6 below shows the location of the intra-box connectors.

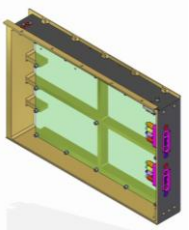


Figure 6 – Module Connector Location

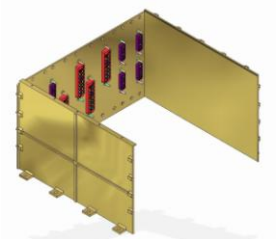


Figure 7 – Backplate Assembly

4.3 BACKPLANE MECHANICAL DESIGN

The backplate mechanical assembly is only necessary if the individual modules require easy removal, as it holds the mating connectors so the modules can be extracted and inserted without disturbing the intra-box wiring harness. The backplate Z coordinate length (module height) is not standardized and is to be designed based on user requirements. The X and Y coordinates are standardized and will be controlled on the backplate assembly MICD. The backplate connector locations are determined by the users' requirements. The height or Y coordinate of the connector is constrained. The Z coordinate location is user defined and is to be placed where needed. The back plate assembly consists of the back plate and the two end covers of the SpaceAGE assembly. Once the back plate and gussets are assembled the backplate assembly is populated with the interface connectors and they are wired with the wire harness. Figure 7 above shows the backplate assembly.

HUB and NODE module assemblies can be inserted into the backplate assembly during box level integration and testing as needed and or as they become available. NODE or HUB assemblies mate blindly with the backplate connectors. These connectors have outer shells that are keyed as well as guide pins to assist with mating. The connectors in the backplane assembly are mounted with floating bushings that allow the connectors to float and successfully blind mate with the module connectors without spark "scooping". Once the module is inserted into the backplate and properly mated, the module is mechanically fastened to the back plate assembly using captive fasteners that are installed on the backplane assembly. The modules also attach to each other as they are designed to interlock. This allows for a stiffer structure as well as an EMI barrier. Figure 8 shows the NODE and HUB modules inserted into the backplate assembly.

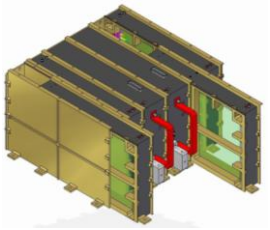


Figure 8 – SpaceAGE Assembly: Backplate and Modules

4.4 THERMAL AND STRESS ANALYSES

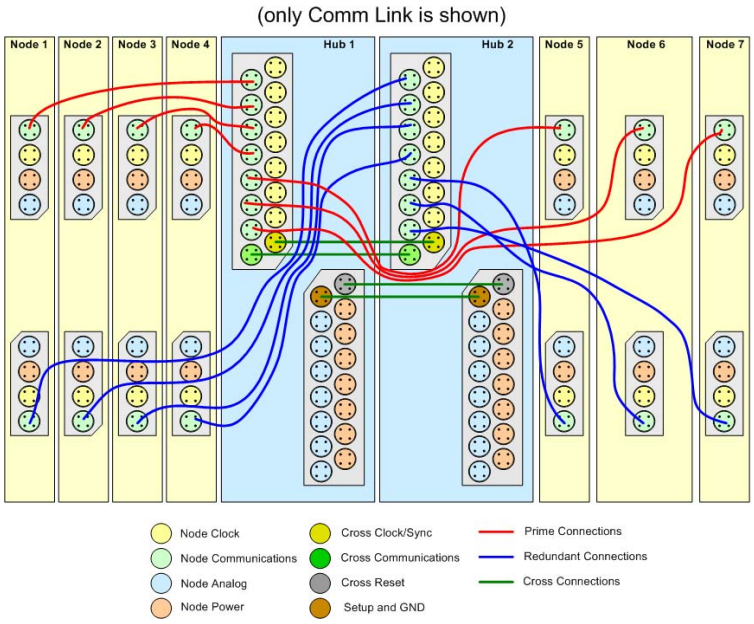
Each module is thermally independent and has a direct heat path to the spacecraft interface. This is an advantage over standard card locks and wedge locks as the direct heat path offers lower thermal impedance and where removal of the card breaks the thermal path (wedge locks) requiring requalification. Each module can be treated as its own thermal structure and can be analyzed and qualified independently of the other modules.

A preliminary stress analyses shows that SpaceAGE Bus assembly can successfully survive an ascend on any space launch vehicle.

Appendix A: Suggested SpaceAGE Bus Interconnections Between HUB and NODEs, and Between 2 HUBs.

Group	Sub Group	Function	Pin	Node Bus Connector	Flow Direction	Hub Bus Connector	Flow Direction	Redundant Hub	Notes
Hub to Node Bus (28 inserts out of 32 for 7 Nodes)	Digital	Serial Communication	1	RX+	?	TX+			Full Duplex link. Diagonal pins 1-3 and 2-4 provide 100Ω impedance
			2	TX+	?	RX+			
			3	RX?	?	TX?			
			4	TX?	?	RX?			
		Clock and Reset Distribution	1	Clock_in+	?	Clock_out+			Clock function is defined by Node end user Node can be reset individually by Hub
			2	Reset_in+	?	Reset_out+			
			3	Clock_in?	?	Clock_out?			
			4	Reset_in?	?	Reset_out?			
	Power and Analog	Power and Supply Sync	1	Node **Power**	?	Node Power			Up to 1.5A@28V of derated Node current; DC/DC Sync is 200-800KHZ free running 5V clock; Hub generated Power Fail
			2	**Power** Return	?	Power Return			
			3	DC/DC_Sync_in	?	DC/DC_Sync_out			
			4	Power Fail	?	Power Fail			
		Analog Telemetry and Node Sense	1	Analog_out+	?	Analog_in+			Each Node may have 4-16 analog telemetry slots or just 1 passive thermsitor; "Sense" tells Hub if Node is plugged in and secured
			2	Analog_out?	?	Analog_in?			
			3	Sense_out+	?	Sense_in+			
			4	Sense_out?	?	Sense_in?			
Hub to Hub Crossover Bus (4 inserts for an extra Hub)	Digital	Cross Communication	1			X_TX+		X_TX+	Full Duplex cross link. Diagonal pins 1-3 and 2-4 provide 100Ω impedance
			2			X_RX+		X_RX+	
			3			X_TX?		X_TX?	
			4			X_RX?		X_RX?	
		Cross Clock	1			X_Clock_out+		X_Clock_out+	Allows both Hubs to share common clock
			2			X_Clock_in+		X_Clock_in+	
			3			X_Clock_out-		X_Clock_out-	
			4			X_Clock_in-		X_Clock_in-	
	Reset and Config	Cross Reset	1			X_Reset_out+		X_Reset_out+	X_Reset allows each Hub to reset its peer Hub either by command, or by lack of communications for the TBD time period
			2			X_Reset_in+		X_Reset_in+	
			3			X_Reset_out?		X_Reset_out?	
			4			X_Reset_in?		X_Reset_in?	
		Mster-Slave Configuration and Peer Hub Plug-in	1			Peer_Hub out		Peer_Hub out	Tells each Hub that its Peer Hub is plugged-in
			2			Peer_Hub in		Peer_Hub in	
			3			Config_out		Config_out	Master HUB (A) - no jumper, Slave (B) - external jumper
			4			Digital GND		Digital GND	

Appendix B: NODE Ports Redundant Cross Connections Diagram.



Poster Presentations

TIME DISTRIBUTION OVER A SPACEWIRE NETWORK

FOR THE ARTEMIS SUBMILLIMETRIC INSTRUMENT

Session: SpaceWire missions and application

Short Paper

Cara Christophe, Eric Doumayrou, Pinsard Frederic

CEA Saclay DSM/IRFU/Service d'Astrophysique,

bât. 709 L'Orme des Merisiers, 91191 Gif-sur-Yvette, France.

E-mail: christophe.cara@cea.fr, eric.doumayrou@cea.fr, frederic.pinsard@cea.fr

ABSTRACT

The ArTeMiS submillimetric camera will observe simultaneously the sky at 450, 350 and 200 μm using 3 different focal planes. The 3 focal planes are made of thousands of pixels sampling completely the field of view by using the same technology processes than those used for the Herschel-PACS space-born imager. This camera will be mounted in the Cassegrain cabin of APEX, a 12 m antenna located on the Chajnantor plateau, Chile. The control and readout of the camera is achieved by the warm electronics acquisition system comprising 10 BOLERO (Bolometer electronic readout box) units and a COYOTTE camera control unit. The BOLERO electronics being derived from the Herschel-PACS readout electronics various parts of the existing design have been re-used. In particularly the data and command exchange between the BOLERO units and the quick-look and archiving workstation is relying on 10 SpaceWire links.

In this paper we present the implementation of IRIG-B standards in the ArTeMiS camera. We show this is achievable by adding support of the ESA 'Time Code Formats' as specified in CCSDS 301.0-b-3 Blue book' to our existing SpaceWire IP.

INTRODUCTION

Early tests performed on a prototype of the camera raised some unexpected synchronisation problems with the APEX facility software. This is mainly due to data buffers in the BOLERO electronics, which induce unpredictable delays to occur between the detector readout and effective frame acquisition by the acquisition computer. Then it has been decided to implement a GPS dating of the ArTeMiS images thanks to an IRIG-B signal available on APEX facility. Since the telescope uses this standard for absolute dating of all its instruments and in particularly the equipment, which is in charge of its motion during observation, it will be possible to correct the images of the sky from the drift induced by this movement.

THE TIME TAGGING IN ARTEMIS

Figure 1 depicts the data architecture of the ArTeMiS instrument. The 10 SpaceWire links are interfacing with the acquisition computer by means either of 3 SpaceWire-

PCIe acquisition boards either of only one SpaceWire-PCIe acquisition boards and 2x 8 to 1 router boards. A specific board also hosted by the acquisition computer receives the IRIG_B signal. This board decodes the incoming signal and distributes the time code format to the SpaceWire acquisition boards that are in turn forwarding the dating to the 10 camera units thanks to our extension.

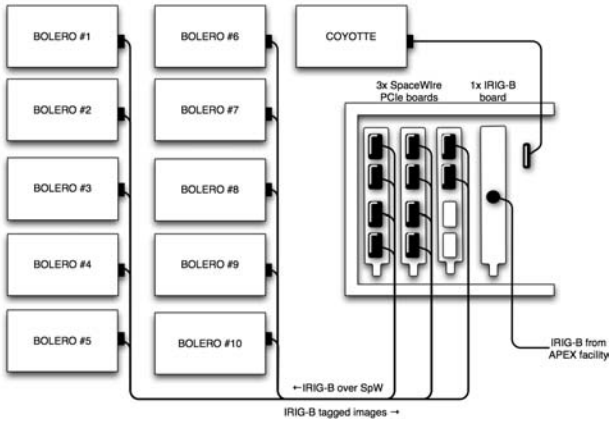


Figure 1 ArTeMIS data architecture

THE IRIG-B STANDARD

The inter-range instrumentation group time codes, commonly known as IRIG time codes consist in a family of rate-scaled serial time codes with formats containing up to four coded expressions or words. All time codes contain control functions that are reserved for encoding various controls, identification, and other special purpose functions. The latest version of the Standard is IRIG Standard 200-04. Depending on the resolution to be achieved various time codes are defined (alphabetic designation from A, B, D, E, G and H) with time frame ranging from 0.01 s to 1 mm. In turn the bit rate expressed in pulse per second (pps) is ranging between 1 pps and 100 pps. The time code is associated to various low-level encoding / transmission options such as the modulation frequency and mode (pulse width, amplitude modulated sine wave, Manchester). The ARTEMIS instrument implements only the most commonly used standard: the IRIG-B. The time frame for the IRIG-B standard is 1 second, meaning that one data frame of time information is transmitted every second. The 74-bit time code contains 30 bits of BCD (binary coded decimal) time-of-year information in days, hours, minutes and seconds, 17 bits of SB (straight binary) seconds-of-day, 9 bits for year information and 18 bits for control functions as shown in figure 2. The frame rate is 1.0 second with resolutions of 10 ms (dc level shift) and 1 ms (modulated 1 kHz carrier).

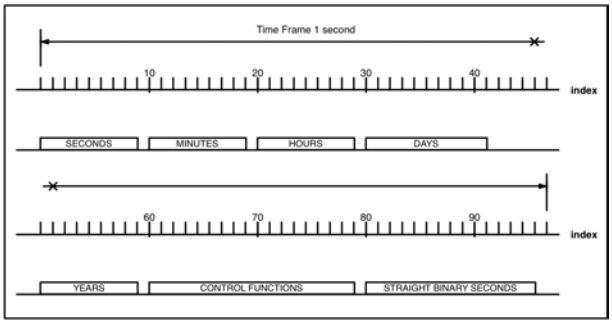


Figure 2 IRIG-B protocol

THE CCSDS TIME CODE FORMATS

Similarly to the IRIG time codes the CCSDS has defined various time codes in order to achieve time tagging on-board satellite. This time codes are specified in [1] where four formats are defined. All the formats are composed of two fields; the P-field, which specifies options for the time code, and T-field, which contains the time code. The following table sums-up three of the CCSDS time code formats: the CUC for CCSDS Unsegmented Time Code, the CDS for CCSDS Day Segmented Time Code and the CCS for CCSDS Calendar Segmented Time Code. Thanks to the available options these formats are extremely flexible and depending on the requirements in term of resolution and dynamic range it is possible to exactly trim the suitable

formats. In particularly the CCS format is identical to IRIG-B when setting the P-field to 0x3A and therefore the date is composed of the following fields: year - day of year – hours – minutes – seconds - tenth of milliseconds. For that reason and in order to extend the possible application of our development we decided to implement full support to CCSDS time code format rather than limiting our development to the IRIG-B time code only.

THE SPACEWIRE TIME CODE

The SpaceWire standard specifies the TIME-CODE character to propagate the time across a network [2]. The TIME-CODE is an 8-bit character whose transmission is triggered by the assertion of the TICK-IN input of the SpaceWire transmitter. Two heading bits being reserved to define the type the TIME-CODE can therefore be used to propagate 6-bit time information across a network. Currently the TIME-CODE transmission request occurs asynchronously with respect to the transmitted character stream. However thanks to priority arbitration inside the transmitter the TIME-CODE is inserted within the data flow with a limited jitter of 13 clock periods [2][3]. When received the TIME-CODE is extracted from the data flow and made available in a specific FIFO-less output port of the receiver while a TICK-OUT signal is asserted. According to the standard specification routers broadcast the TIME-CODE to the next stages of the SpaceWire network.

TIME OVER SPACEWIRE

T7-T6	T5-T4	TimeCode	Definition
11	00	xC & jitter	delay for jitter correction
11	01	xD & dig(i)	subTop + time digit #i
11	10	xE0	subTop
11	11	xF0	top - latch time digits

Considering the SpaceWire capability related to TIME-CODE and in order to meet the requirement of the ArTeMIS instrument we have designed a small extension to our SpaceWire codec to support IRIG-B formatted time distribution across our instrument. By taking advantage of similarities between this time code format and the time code format specified by the CCSDS the implementation we propose is able to accommodate both formats. The next table defines the TIME-CODE we have defined. The two most significant bits are set to 11 to indicate the specific format, which will follow. This code is defined arbitrarily and could be modified according to SpaceWire standard constrains. Then bits 5 and 4 are used to define the significance of the TimeCode field. This identifier sets to 00 indicates the TimeCode field is carrying the jitter correction as proposed in [3]. When set to 01 it indicates the TimeCode is carrying one of the time 'digit' (one of 26) and a subTop. When set 11 it indicates the TimeCode corresponds to the time synchronisation used to latch the previously received TimeCode. This is similar to PPS signal as distributed on

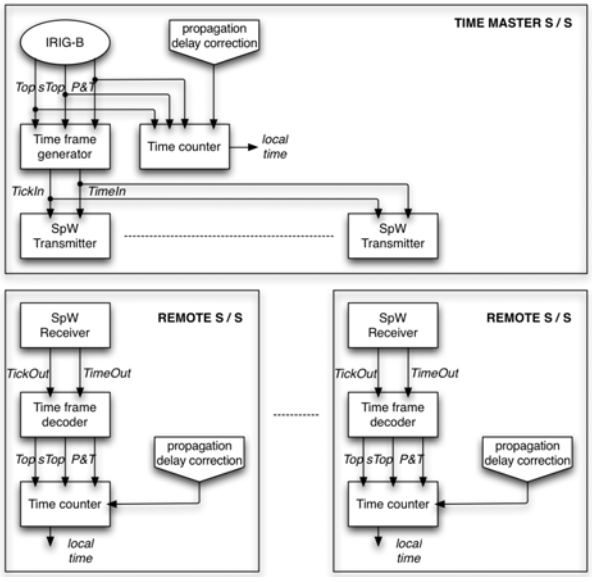


Figure 3 CCSDS implementation

board a satellite and used along with time messages to synchronise sub-system local times with the reference time of the platform. Additionally an identifier set to 01 indicates the TimeCode is a subTop only. This subTop is used to provide the remote sub-systems with a time base signal. Typically it can be transmitted once every ten millisecond. Since the transmission of a full time message requires only of few tenth of TimeCode characters every second the 'subTop only' code permits to provide the remote sub-system with a continuous time base signal. Next the four least significant bits may be used to carry the CCSDS time code. Since only four bits are available the incoming time code bytes are split into half byte characters. Then the transmission of the 1-byte long P-field and 13-byte long time code corresponds to the transfer of 28x TIME-CODE characters over the SpaceWire links (see table below). Figure 3 represents the functional architecture of the updated SpaceWire interface.

CCSDS time format					
		CUC	CDS		CCS
P-field	dig(0)dig(1)	0x2 or 0x4	0x8		0xA
	dig(2)dig(3)	2^{24} s	0x0		year
	dig(4)dig(5)	2^{16} s	2^{16} day		year
	dig(6)dig(7)	2^{8} s	2^{8} day	month	day of year
	dig(8)dig(9)	2^{0} s	2^{0} day	day of month	
	dig(10)dig(11)	0x0	2^{24} ms		hours
T-field	dig(12)dig(13)	2^{-8} s	2^{16} ms		minutes
	dig(14)dig(15)	2^{-16} s	2^{8} ms		seconds
	dig(16)dig(17)	2^{-24} s	2^{0} ms		10^{-2} s
	dig(18)dig(19)	0x0	2^{8} µs	2^{24} ps	10^{-4} s
	dig(20)dig(21)	0x0	2^{0} µs	2^{16} ps	10^{-6} s
	dig(22)dig(23)	0x0	0x0	2^{8} ps	10^{-8} s
	dig(24)dig(25)	0x0	0x0	2^{0} ps	10^{-10} s
	dig(26)dig(27)	0x0	0x0	0x0	10^{-12} s

CONCLUSION

This method defines a high-level CCSD time management and allows transmitting time independently from the user application data traffic. The supports of various CCSDS format allow fulfilling of mission specific needs. It takes full advantage of SpaceWire TIME-CODE broadcasting capability. To support this time management, two VHDL IP cores have been written: 'Time Frame Generator' and 'Time Frame Decoder'. This IPs can be adapted to any SpaceWire codec as they use only the standard interface. For very demanding application the time jitter correction will be added to all Time-Code transmission which solves the problem of latency, jitter and drift.

Reference documents:

[1] Time Code Format - 'CCSDS 301.0-b-3 Blue book' – January 2002

[2] Steve Parkes "The Operation and Uses of the SpaceWire Time-Code", International SpaceWire Seminar, ESTEC Noordwijk, The Netherlands, November 2003.

[3] F. Pinsard and C. Cara "High resolution time synchronization over SpaceWire links", Aerospace Conference 2008, IEEEAC paper#1158, 10.1109/AERO.2008.4526462

GR712RC – DUAL-CORE PROCESSOR WITH SIX SPACEWIRE LINKS – VERIFICATION RESULTS

Session: SpaceWire Components (Poster)

Sandi Habinc, Marko Isomäki, Jiri Gaisler

Aeroflex Gaisler, Kungsgatan 12, SE-411 19 Göteborg, Sweden

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

ABSTRACT

The GR712RC System-on-Chip (SoC) is a dual core LEON3FT system suitable for advanced high reliability space avionics. Fault tolerance features from Aeroflex Gaisler's GRLIB IP library and an implementation using Ramon Chips RadSafe cell library enables superior radiation hardness.

The GR712RC device has been designed to provide high processing power by including two LEON3FT 32-bit SPARC V8 processors, each with its own high-performance IEEE754 compliant floating-point-unit and SPARC reference memory management unit. This high processing power is combined with a large number of serial interfaces, ranging from high-speed links for data transfers to low-speed control buses for commanding and status acquisition

1. ARCHITECTURE

The GR712RC device comprises the following functions [2]:

- 2 x LEON3FT processor cores with MMU and GRFPU
  - Branch prediction and on-the-fly error correction resulting in 30% performance increase compared to regular LEON3FT
  - 4x4 kBytes instruction cache and 4x4 kBytes data cache
  - On-chip Debug Unit with instruction and AHB trace buffers
- PROM/SRAM/SDRAM fault tolerant memory controller (using BCH or Reed-Solomon)
- 256 kBytes on chip fault tolerant RAM
- 6 x SpaceWire links (2 with RMAP support)
- 6 x UARTs
- 6 x General Purpose Timers (2 with time latch capability)
- Multi processor Interrupt Controller with support for 31 interrupts
- 2 x 32 bits General Purpose I/O
- JTAG debug link
- 10/100 Ethernet MAC with RMII interface
- MIL-STD-1553B BC/RT/BM controller
- 2 x CAN 2.0 and one SatCAN controller
- CCSDS Telecommand decoder and Telemetry encoder
- SPI controller
- I2C controller
- SLINK controller
- ASCS16 controller

- Clock gating unit
- I/O switch matrix

The variation in interfaces allows different systems to be implemented using the same device type, which simplifies parts qualification and procurement. It also brings cost reductions to software development since the core functionality can be reused from application to application, only changing the drivers for the interfaces.

Due to the high amount of peripherals and a limited number of pins there is an I/O switch matrix that controls which peripheral is connected to each pin.

The clock-gating unit can turn off the clock for each major peripheral, thus lowering power consumption considerably. The processor clock is automatically turned off when a processor is in power down mode. The FPU is clock gated when floating point operation is disabled or when the corresponding processor is powered down.

2. DEVICE CHARACTERISTICS

The device will be manufactured by Tower Semiconductors Ltd. using standard 180 nm CMOS process and packaged in 240-pin 0.5 mm pitch CQFP and PQFP.

The following characteristics are expected [1]:

- Core voltage 1.8V +/- 10%, I/O voltage 3.3V +/- 10%
- 55ºC to +125ºC temperature range
- TID: 300 krad (Si) (RHA Class F according to PRF-38535 Sect. 3.4.3)
- SEL: LET > 106 MeV/cm2/mg
- SEU: Cross section < 20um^2
- Maximum clock frequency 125 MHz
- Optional 2x internal frequency multiplication by all-digital DLL

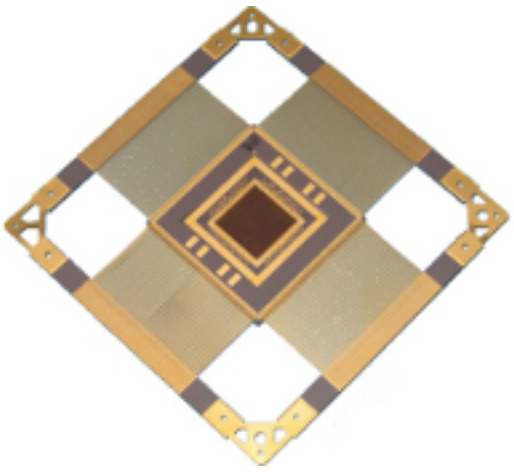


Figure 1: GR712RC die in a 240 pin ceramic quad flat package

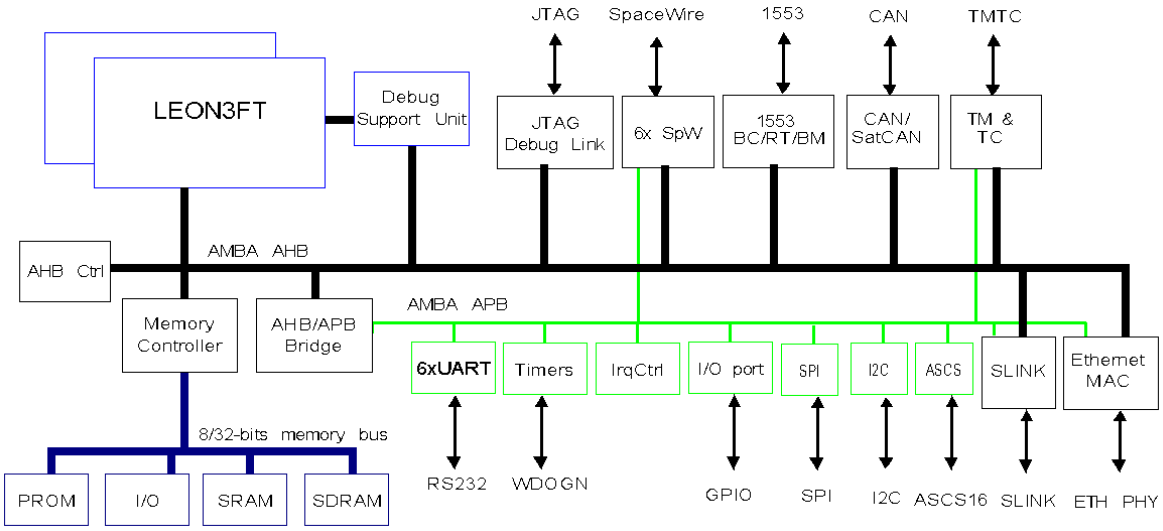


Figure 2. GR712RC block diagram

3. GR712RC DEVLOPMENT BOARD

In order to provide a platform for customers to begin developments using the GR712RC device, Aeroflex Gaisler provides a GR712RC development board. The board comprises a custom designed PCB in Compact PCI 6U format which can be used either stand-alone or inserted into a CPCI rack.

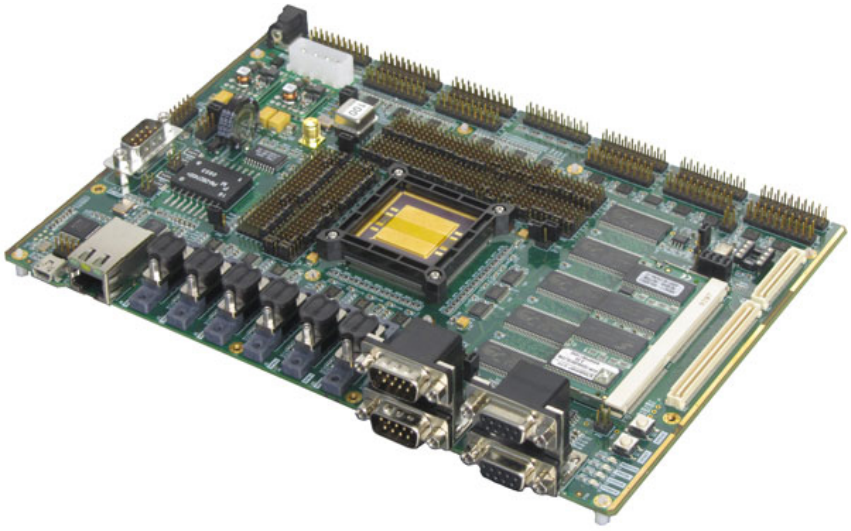


Figure 3. GR712RC development board

The board has interfaces for all peripherals and 8 MByte of SRAM (with checkbits), 8 MByte of FLASH, and a standard SDRAM SODIMM socket.

Each pin in the I/O switch matrix is configured with a jumper. The various configurations of interfaces are presented in the next section. All the standard interfaces are conveniently located on the front side of the board, allowing easy access to a CPCI front-panel.

4. VERIFICATION RESULTS

The measured performance of the GR712RC device at 125 MHz system clock frequency is 300 Dhrystone MIPS. The measured speed of the SpaceWire links is above 250 Mbps in room temperature. These values may be adjusted after the full qualification.

The GR712RC development board has been used during the verification of the SpaceWire performance of the device. The performance of all SpaceWire links operating at full speed has been assessed during the verification of the GR712RC devices, without any degradation in performance for example that could be due to a potential congestion on the on-chip AMBA bus.

The 192 kByte on-chip memory located on the AMBA bus has proven sufficient in size for implementing transmit- and receive-buffers handled by the SpaceWire software drives. This allows the GR712RC to implement in software a SpaceWire router with six SpaceWire ports. The performance of a single LEON3FT core is sufficient for this router implementation, not necessitating the use of the second core which allows it to be used for entirely different tasks, for example implementing the TCP/IP software stack for communication over the Ethernet 10/100 interface.

Although a SpaceWire / Ethernet software bridge has not been implemented as part of the current verification of GR712RC, a similar implementation has been done in the GRESB SpaceWire / Ethernet bridge using a single LEON3 core operating at 40 MHz which provides approximately 20 Mbit/s sustained throughput through the Ethernet side. A prediction is that it should be possible to support the maximum 100 Mbit/s Ethernet throughput using the spare LEON3FT core.

The GR712RC is implemented on the 180 nm Tower technology using the RadSafe radiation-hard-by-design library from Ramon Chip. The GR712RC has undergone radiation testing, it is latch-up free, and it is fully protected against single event upsets in registers and memory, and tolerates a high total ionizing dose.

5. CONCLUSIONS

The GR712RC device brings multi-processing to avionics and payload applications, increasing the processing performance compared to existing solutions, without consuming board real estate or demanding complex memory implementations.

The GR712RC development board has been designed to support initially stand alone operation, but also to fit into the future RASTA (Reference Avionics System Test-bench Activity) architecture where inter-board communication is realized through SpaceWire links.

6. REFERENCES

[1] Dual-CoreLEON3-FT SPARC V8 Processor, GR712RC, Preliminary Data Sheet, Aeroflex Gaisler, www.gaisler.com

[2] Dual-Core LEON3-FT SPARC V8 Processor, GR712RC, User's Manual, Aeroflex Gaisler, www.gaisler.com

CASCADING THE 10X SPACEWIRE ROUTER FPGA STANDARD PRODUCT IN A FLIGHT BOARD DESIGN

Session: Missions and Applications (Poster)

Short Paper

Marko Isomäki, Sandi Habinc

Aeroflex Gaisler AB, Kungsgatan 12, SE-411 19 Göteborg, Sweden

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

ABSTRACT

Aeroflex Gaisler has developed several rad-hard SpaceWire router standard products based on Actel RTAX and RT ProASIC FPGAs. The largest of these components has eight SpaceWire and two FIFO ports where the number of ports (10x) is restricted by area limitations in the FPGA. Several planned missions require more than eight SpaceWire ports which the current standard products do not fulfil. There are also no components available from other manufacturers with a higher number of SpaceWire ports. The solution described in this paper cascades two ten port routers using the FIFO ports resulting in a total of 16 SpaceWire ports. Where previous solutions of this type have required external glue-logic, this one only needs one configuration pin to be strapped at reset.

1 INTRODUCTION

While more and more customers require up to 16 ports in SpaceWire routers there are only up to eight ports available in current components on the world market. The largest router component available from Aeroflex Gaisler has eight SpaceWire ports and two FIFO ports [1]. It is based on the GRSPWROUTER IP Core [2] that supports up to 31 ports but, due to area limitations in the used Actel RTAX FPGA devices, is limited to ten ports in total.

This paper describes the solution of cascading two ten port routers compared to moving up to a larger FPGA to achieve a 16-port router.

2 SINGLE FPGA SOLUTION

The Aeroflex Gaisler 10-port router is implemented in an RTAX2000 device. One solution for achieving a 16-port router in a single FPGA approach would be to move up to a larger RTAX4000 device. This could easily be done since it would essentially only require reconfiguration of the GRSPWROUTER IP core which supports up to 31 ports.

The reason for this solution usually not being feasible is the lack of an inexpensive non-rad-hard prototyping device making system prototyping and validation difficult and costly. This in addition to the fact that the RTAX4000 is also a more expensive

device compared to the RTAX2000 makes many system designers reluctant to choose this solution. The AX2000 is an non-rad-hard version of the RTAX2000 that is comparatively low cost and has been used to validate the standard router configurations. Actel tools provide automated generation of FPGA programming files for the prototype from the original file targeting the rad-hard device. With the RTAX4000 this is not possible and the design would have to be validated using a qualified device or relying only on gate-level simulations. None of those two alternatives are feasible in practice which led to the search for other solutions.

3 CASCADING TEN PORT ROUTERS

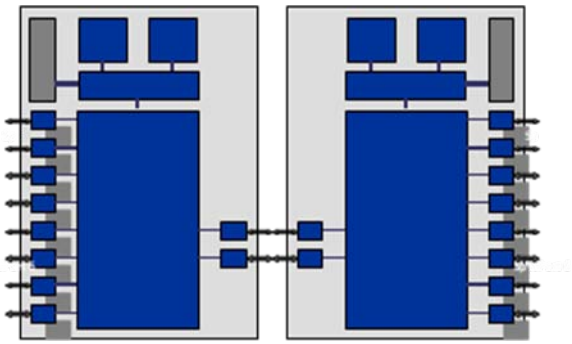


Figure: Two 10x GR-SPW-ROUTER-RTAX SpaceWire routers cascaded by dual FIFO interfaces in bridge mode, providing 16 SpaceWire links

An alternative solution to one large FPGA is to cascade two or more 10x devices to achieve a larger router using the FIFO ports. Many other devices have similar parallel data ports available and use solutions with extra FPGAs as glue logic to cascade multiple routers. This requires custom design of the extra FPGA and is costly in terms of power, area and development time. The risk is also higher since an extra custom step is required which needs to be verified and validated compared to using an already validated component that is already in use in existing systems.

Aeroflex Gaisler router FPGAs have a built-in bridge mode for the FIFO ports which allows the ports of two routers to be connected without any glue logic. Data and time-codes will be transferred in each direction automatically. The bridge mode can be enabled through the configuration port or via an external signal. The external signal sets the reset value of an internal bit controlling the bridge mode. This way, a board for a bridge application can be manufactured with a strap signal so that the two FPGAs enter bridge mode without any software intervention. To keep the solution flexible, it is possible to change this setting through the configuration port in systems with for example a processor. This is done through RMAP accesses to the router configuration port.

To avoid that the setting is accidentally changed by some malfunctioning device there are several ways to protect the configuration port. The whole configuration area can be write protected so that a write enable bit must be set before other configuration options can be altered. This significantly lowers the risk of an erroneous reconfiguration. Configuration accesses can also be disabled for each port individually making it completely impossible for a connected device to do any harm.

There are dedicated paths for both data and time-codes over the FIFO bridges. Data throughput over the FIFO bridges is at least the same as for the SpaceWire ports provided that the core frequency is at least 1/8 of the SpaceWire maximum bitrate. If the core frequency is increased, the bandwidth will be improved with the same factor. It will however never be possible (in any practical cases) to get a throughput over the bridges that matches the total throughput of all the SpaceWire links on one FPGA. This means that the bridges might become a bottleneck if a large part of the SpaceWire traffic is going between nodes connected to ports on different FPGAs. In many systems there are a few high bandwidth nodes that communicate with each other and can therefore be connected to ports on the same FPGA. In those cases the bandwidth problem would be eliminated at the expense of a, potentially, increased burden for the system designer.

4 CONCLUSION

To achieve a 16 port router device it is more cost effective to cascade the existing 10-port router FPGAs compared to moving to a 16-port device in a single FPGA. The cost difference is achieved by providing FIFO ports with a bridge mode that removes the need for external glue logic. The only downside is that the bandwidth between the cascaded devices is limited by the bridge ports. It is however anticipated that this problem can be avoided in practice by careful system design.

5 REFERENCES

1. RT-SPW-ROUTER Data Sheet and User's Manual, Aeroflex Gaisler, www.gaisler.com

2. GRSPWROUTER User Manual, Aeroflex Gaisler, www.gaisler.com

TACSAT-4: SPACEWIRE FOR RESPONSIVE INTEGRATION AND LAUNCH

Session: SpaceWire missions and applications

Short Paper

Paul Jaffe, Eric Rossland, Eric Bradley

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

Greg Clifford

Silver Engineering Inc., 255 East Drive, Melbourne, FL 32904, USA; GClifford@silvereng.com

Herb Axe

Sierra Nevada Corp., 1722 Boxelder St., Louisville, CO 80027, USA;Herb.Axe@sncorp.com

1 ABSTRACT

Rapid development, integration, and deployment of satellites in response to known and emerging needs have been ongoing areas of interest. Often collectively referred to as "Operationally Responsive Space" (ORS), one vision calls for positioning in a depot interchangeable satellite payloads and spacecraft buses with common interfaces. Upon direction to deploy a particular mission, the appropriate payload is selected and integrated with a bus, and the space vehicle is launched. This necessitates standardized hardware and software interfaces between the payload and bus. For the development of ORS Bus Standards, SpaceWire standard ECSS-E-50-12A was specified as part of the payload-bus interface for high rate data. With a 2011 launch, the TacSat-4 satellite demonstrates both a prototype Standardized Bus for small satellite national security missions and an example ORS payload, CommX. This implementation includes a SpaceWire interface as 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 the NASA Goddard Space Flight Center. The SpaceWire link runs between the Payload Data Handler (PDH) on the bus side and Universal Interface Electronics (UIE) on the payload side. Connector interfaces were adapted to be suitable for the launch depot environment. TacSat-4 and the ORS Standards Development effort led by the government, industry, and academia Integrated Systems Engineering Team (ISET) have demonstrated that use of existing standards blended with tailoring for rapid integration enables ORS.

2 INTRODUCTION

The motivation to reduce the cost and speed the fielding of space assets has been of interest since the dawn of the space age. To this end, different countries and organizations have implemented various approaches. In the 1970s, the Soviet Union kept reconnaissance satellites ready to launch within 24 hours, and they used them to collect intelligence during international crises such as the Arab-Israeli war in 1973[1].

More recently, the U.S. Department of Defense has supported a range of approaches to reducing the time and costs associated with taking advantage of spaceborne assets. At the instigation of Adm. Cebrowski in 2001, efforts were undertaken to find ways to streamline the deployment and exploitation of satellite resources[2]. This led to the development of TacSat-1 in 2003 as an Innovative Naval Prototype, and the christening of such efforts as pertaining to "Operationally Responsive Space", or ORS. The TacSat-1 development went from concept to launch-ready within about a year and for about $10M[3]. The need to identify those national security space

missions most subject to a rapid approach was recognized, and the Office of Force Transformation funded a Massachusetts Institute of Technology (MIT)/Lincoln Labs "Phase 1" study to investigate mission classes and their needs[4]. At about the same time, the Air Force Research Laboratory undertook an effort to develop standardized software and hardware component interfaces, dubbed "Space Plug & play Avionics", or SPA, to enable rapid custom mission design and spacecraft implementation through the assembly and self-organization of an essentially arbitrary number of components[5].

The results of the Phase 1 study by MIT/Lincoln Labs were used by a funded consortium of representatives from industry, academia, and government organizations to develop standards for an ORS system. The consortium, know as the Integrated Sytems Engineering Team (ISET), ultimately produced a set of documents prescribing standards for an ORS system that encompassed a range of small satellite national security missions. These standards outlined an approach that split the spacecraft into two major sections: (1) the bus, which provides services required by a typical satellite such as attitude control, power, propulsion, and command and telemetry; and (2) the payload, which performs the mission function, such as communications, imagery, intelligence, etc. One of the main points of this division was to allow the companies that would be contracted to build parts of the system to take advantage of their existing technical approaches, while constraining only the bus/payload interface to a standard.

As part of ORS Phase 3, the Naval Research Laboratory (NRL) and the Applied Physics Laboratory (APL) were selected to develop an ORS spacecraft bus that adhered to the ISET standards. Separately, a different team at the Naval Research Laboratory was selected to develop an example ORS payload. This payload performs a communications function and is designated COMMx. Together, the bus and payload form the TacSat-4 spacecraft.

Much attention was paid during the ISET standards development to the data interface to be used between the bus and payload. In the final standards, two data interfaces are specified: RS-422 for lower rate data (below 10 Mbps) and SpaceWire for higher rate data (10Mbps or above). The bus supports both, and the payload may use either or both. The selection of SpaceWire arose from a high rate data trade study that also considered IEEE-1394 and Ethernet[6].

SpaceWire was a natural choice for part of the data interface because of its simplicity, well-written standard, ability to be easily implemented in a variety of hardware, and significant existing user base. By taking advantage of a proven and accepted standard, the lessons learned and extant infrastructure could be utilized. Additionally, using SpaceWire fostered the possibility that system implementers might already have the relevant experience when developing ORS buses, payloads, and supporting equipment. The fact that NASA Goddard makes available free to U.S. entities VHDL cores for SpaceWire nodes and routers further enhanced the choice of SpaceWire.

3 TACSAT-4 SPACEWIRE IMPLEMENTATION

One aspect of the SpaceWire standard that was not ideally suited for ORS was the connector specification. The use of micro-D 9-pin connectors for cable interconnects requires tools, handling precautions, and attention to detail not necessarily conducive to a rapid-response launch depot environment staffed with relatively unskilled personnel. Because the bus and payload need to be mated quickly and reliably at the launch depot shortly before launch, alternative connectors were investigated. The connectors selected were series 38999-D 13-pin circular connectors that offer keying and fast, tool-less installation. Bulkhead varieties allow the SpaceWire link to be brought from the electronics to a convenient place on the bus or payload for mating during depot operations. Additional details of the cabling construction, characterization, and lessons learned are discussed extensively by Schierlmann[7,8].

On TacSat-4, the SpaceWire link runs between the Payload Data Handler (PDH) board in the Command and Data Electronics (CDE) on the bus side, across the standardized interface to the Universal Interface Electronics (UIE) on the payload side. The PDH was developed by one of us (Clifford) at Silver Engineering using the NASA Goddard SpaceWire VHDL core, and also incorporates routing and data storage functions. The UIE software development was performed by another of us (Axe) at Sierra Nevada Corporation (SNC) and can support a range of functions in addition to acting as a SpaceWire node. It uses a SpaceWire VxWorks driver developed by SNC. The SpaceWire cabling exists in three segments: from the PDH to the bus bulkhead, from the bus bulkhead to the payload bulkhead, and from the payload bulkhead to the UIE. At the PDH and UIE, the standard SpaceWire 9-pin micro-D connectors are used, and they are integrated with the boxes and attached to the inner side of the bus and payload bulkheads, respectively, during the manufacturing process. At the launch depot, the bus and payload are stored separately until mission call up. The mission then specifies which type of payload, from a variety of payloads, is to be mated to a bus. At this point, the bus and selected payload are removed from storage and mechanically mated. Then the electrical connections are made by mating circular connectors for data and power. For TacSat-4, this was tested during the manufacturing process in preparation for simulated depot operations.

4 RAPID SPACECRAFT INTEGRATION AT THE "LAUNCH DEPOT"

The ORS concept of a launch depot entails a storage and integration facility at a spacecraft launch range in which standard buses and different types of compatible payloads are stored to allow integration and launching in short duration in response to a national need. The TacSat-4 spacecraft is the first demonstration of the launch depot concept in which distinct and separate bus and payload sections are integrated. Since an actual ORS launch depot does not yet exist, the completed bus and payload were stored instead for a year at a storage facility at NRL in Washington, DC until national priorities called for the TacSat-4 launch. The two parts are shown in the leftmost picture in the figure.

After call-up, the ORS bus and COMMx, which were stored separately, were given one month to be removed from storage, tested independently, readied for shipment, packed along with all test equipment, and shipped to Kodiak Launch Complex (KLC) in Kodiak, Alaska for launch. Of particular note is that, in line with the ORS concept, the bus and payload were not electrically or mechanically mated after the storage period prior to shipment. (Of course the space vehicle (SV) had been mated earlier as part of the test campaign.) Upon arrival at KLC (the "launch depot"), the bus and COMMx payload were again tested independently and in parallel to verify functionality following the cross-country transport. Only after independent functional testing of the bus and payload were they integrated into the full space vehicle, after which all testing was repeated in preparation for launch.

One of the challenges of the depot concept is the requirement to expeditiously test a payload (or a bus) after removing it from storage before SV integration. To meet this end, it is important to have adequate simulation of all bus electrical interfaces. Among these interfaces on COMMx is the SpaceWire interface to the UIE box. During COMMx standalone testing, the UIE was tested with a breadboard of the ORS Bus electronics. Not only was this test configuration high fidelity, but it was also as flight-like as possible in keeping with the philosophy, "Test it like you fly it." This flight-like testing allowed for seamless integration with the flight ORS bus spacecraft at the launch depot with minimal risk.

Two of us, Bradley and Rossland, performed the actual mating of the SpaceWire and other interconnects between the payload and bus, respectively. The actual connection of the SpaceWire link between the bus and payload took less than 30 seconds and required no tools. A detail of the SpaceWire link between the bus and the payload, without blanketing, can be seen in the center picture below, and the fully integrated spacecraft can be seen on the right.

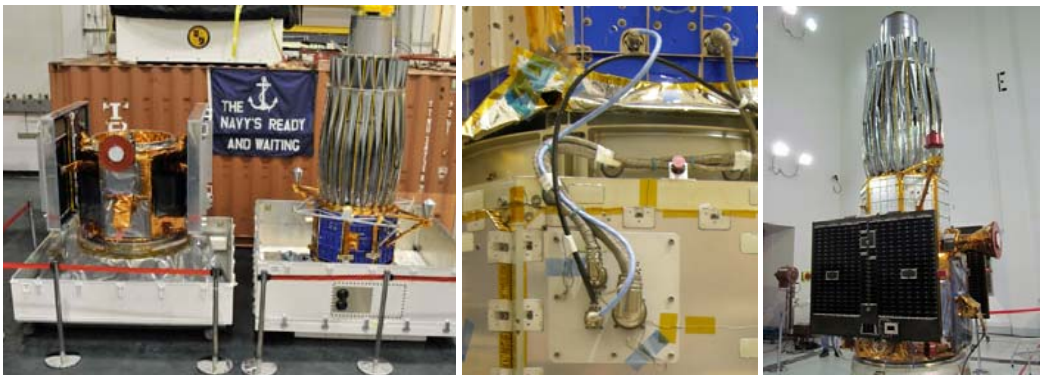


Figure 1: (left)The bus and payload; (center) blue SpaceWire cable between the bus and payload; (right) the integrated spacecraft at the launch site

Because of factors beyond the TacSat-4 program's control, the launch was delayed until September of 2011. Though the Spacecraft has remained in its integrated configuration, the shortened timeline demonstrated for mating of the bus and payload could easily have occurred just in advance of the planned launch. The ISET ORS standards and their instantiation in TacSat-4 have demonstrated that SpaceWire tailored for depot operations offers a compelling solution for high rate data links for ORS.

5 REFERENCES

1. W. E. Burroughs, *Deep Black*, page 258, Random House, New York, NY, (1987).

2. P. Wegner, R. Kiziah, "Pulling the Pieces Together at AFRL," Proceedings of the 4th Responsive Space Conference, RS4-2006-4002, (2006).

3. J. Raymond, et al, "TacSat-1 and a Path to Tactical Space," Proceedings of the 2nd Responsive Space Conference, RS2-2004-5003, (2004).

4. D. Brenizer, et al, "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, (2005).

5. T. Morphopoulos, et al, "Plug-and-Play – An Enabling Capability for Responsive Space Missions," in Proceedings of the 2nd Responsive Space Conference, Los Angeles, CA, Paper No. 5002, (2004).

6. P. Jaffe, et al, "SpaceWire for Operationally Responsive Space as Part of TacSat-4," page 2, in Proceedings of the 2007 International SpaceWire Conference, Dundee, Scotland, (2007).

7. D. Schierlmann, P. Jaffe, "SpaceWire Cabling in an Operationally Responsive Space Environment," in Proceedings of the 2007 International SpaceWire Conference, Dundee, Scotland, (2007).

8. D. Schierlmann, et al, "Lessons Learned from Implementing Non Standard SpaceWire Cabling for TacSat-4," in Proceedings of the 2008 International SpaceWire Conference, Dundee, Scotland, (2008).

SPACEWIRE EVOLUTIONS

Session: Networks and Protocols

Short Paper

David Jameux

European Space Agency / European Space Technology Centre,

Keplerlaan 1, 2201 AZ Noordwijk ZH (The Netherlands)

E-mail: david.jameux@esa.int

ABSTRACT

In this paper, we discuss the need for short term improvements of the current SpaceWire standard and its extension to new domains such as Gbps communications and reliability and real-time capabilities. Focussing on short term improvements, we recall the need for a revision of the current SpaceWire standard as well as the improvements foreseen to be developed, breadboarded and documented in ECSS standardisation format through the ESA/TRP R&D activity “SpaceWire Evolutions” started in September 2011.

1 BACKGROUND

Through several years of standardisation and technology development activities, the European Space Agency (ESA) has prepared the SpaceWire (SpW) technology that allows embarking high speed data networks on board spacecraft. This new technology has become widely adopted not only by ESA missions but also by other agencies and industries. However, some evolutions of the SpaceWire standard have been proposed by the SpaceWire Working Group ([5], [6], [7], [8]) over the last five years.

In particular, the SpW Working Group identified shortcomings of the current protocol for the support of Plug-And-Play (PnP) capabilities ([9], [10], [11], [12]) as defined jointly by ESA and the National Aeronautics and Space Administration (NASA) and drafted into [13]. The technical investigations on SpaceWire PnP also rose the awareness that the behaviour of “nodes” have to be clarified as well as their definition in the current standard [1], because this definition is not in line with international telecommunications core definitions of network items, and in fact ambiguous.

Among the discussed additional features to SpaceWire are the sideband signalling for interrupt distribution and the introduction of SpaceWire operating in half-duplex or simplex mode over wire-limited harness.

These new techniques, as well as the required clarification of SpaceWire node definition and behaviour, are highly promising but they need to be breadboarded prior to their eventual standardisation because they will be adopted by the SpaceWire community only if they are backwards compatible, i.e. if they can operate with existing SpaceWire devices.

This is currently being done in the frame of the “SpaceWire Evolutions” Research & Development (R&D) contract kicked off in September 2011. This contract is funded under the ESA Technology Research Programme (TRP).

2 OBJECTIVES

The objective of this activity is to breadboard a number of modifications and additional features to the SpaceWire standard, to validate these updates and check that they are backwards compatible with the current suite of SpaceWire standards ([1], [2], [3], [4]) by setting up some test bench and running verification procedures. The features to be modified or added are the following.

2.1 CLARIFICATION OF SPACEWIRE DEFINITIONS

For the modifications to the SpaceWire protocol aiming at clarifying the definition and behaviour of “SpaceWire nodes” and better supporting Plug-And-Play capabilities, the first stream of research is on how to align the SpaceWire standard to international telecommunications core definitions of network items. Namely, the network should be described as links and nodes, the links carrying digital information between pairs of nodes, and nodes being either terminal nodes (where information is either produced or consumed) or switching nodes (that can switch the digital information from an input link to one or more output links). However, the network item definitions must be adapted to the specific aspects of SpaceWire on-board networks, mainly redundancy at processing nodes and at communication path levels. Another important issue to consider is that redundancy schemes are influenced by the fact that routers are very likely to be fitted into on-board functional unit boxes (as opposed to having their own box, physically located in between the boxes of two functional units), i.e. physically very close to one or more terminal nodes to which they are linked.

The second stream of research is towards a consistent approach of configuration ports attached to nodes. Indeed, the Plug-And-Play capabilities – as defined jointly by ESA and NASA and drafted into [13] – that are also currently subject to standardisation effort, require registers to be read and written in each and every node, be it a terminal node or a switching node. These capabilities also require that these registered be accessed through the SpaceWire network itself (this is done in practice via the Remote Memory Access Protocol [2]). This implies, on one hand, that switching nodes contain a terminal node; and, on the other hand, that terminal nodes are able to switch some types of SpaceWire packets to a register handling process instead of to their functional host interface.

2.2 INTRODUCTION OF SIDEBAND SIGNALLING

For the broadcast of low-latency signals across a SpaceWire network (for the purpose of distributing on-board systems level interrupts), the baseline solution is to use the ESC + N-Char sequence of characters as described in [1]. Overcoming some ambiguities in the current version of the SpaceWire standard [1], the SpaceWire Working Group have recently agreed that the “Time-code” [17] shall have its Control Flags (bits T6 and T7) both set to 0. The SpaceWire Working Group have also agreed that Control Flag sequence 0b01 may be used for low-latency signalling broadcast, leaving the sequences 0b10 and 0b11 reserved.

A first solution ([14], [15], [16]), based on “Interrupt-codes” (extended Control Flag C5=0) and “Interrupt_Acknowledge-codes” (extended Control Flag C5=1) has been extensively discussed within the SpaceWire Working Group.

However, another research stream will start from the basics of information theory and investigate the possibility of unifying the already defined (and implemented) Time-codes with some general low-latency signal scheme (using the ESC + N-Char sequence of characters) that would allow broadcasting system-level interrupts but also other kinds of low-latency signals such as multiple time-domain clocks.

2.3 Introduction of Simplex and/or Half-Duplex SpaceWire

For the SpaceWire communications which are highly asymmetric in terms of data rate (e.g. sensor-storage), two research streams of SpaceWire Signal Level optimisation will be followed with the common target to reduce the SpaceWire logical signalling to one pair only of Data/Strobe signals (thereby reducing the number of wires required for the physical layer from 8 to 4).

Simplex operation [18] consists in having only one side of the link (the “sender”) sending NULL characters and Normal Characters (N-Chars). The other side of the link (the “receiver”) would only send Flow Control Tokens (FCTs). Sharing of the single physical channel between “sender” and “receiver” communications must be handled by some Medium Access Control (MAC) arbitration.

With Half-Duplex operation, the MAC arbitration allows both ends of the single physical channel to be alternatively “sender” and “receiver”. The robustness of this approach is still to be verified through proper testing. But the first implementations ([19], [20]) show that the MAC arbitration overhead can be kept within 20%.

Both schemes (Simplex and Half-Duplex) will be designed and formally verified. A traded-off between the two will lead to selection of one of them for breadboarding. This will allow for verification of robustness and integrity, but also of compatibility between this scheme and Full-Duplex SpaceWire as defined in [1]: a SpaceWire packet must be able to pass through a series of SpaceWire links that may be full-duplex, half-duplex or simplex, with no other modifications of its properties than would be introduced by passing trough a series of full-duplex links with differing link speeds (i.e. timing properties).

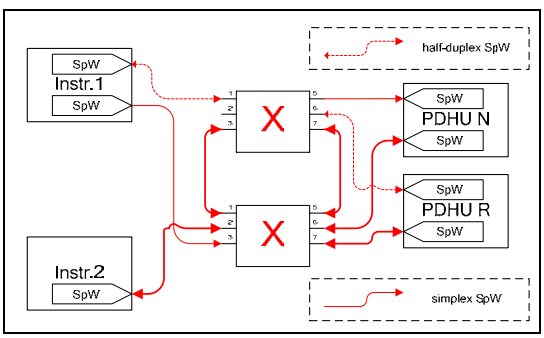


Figure 1 – Example of SpaceWire network mixing full-duplex, half-duplex and simplex links

3 Validation

For the validation of these new features at breadboard level, a test setup and verification procedures will allow demonstrating the functionality, the performance, and the backwards compatibility of the feature with the current SpaceWire standard [1]. The test setup will be mainly based on existing SpaceWire equipment modified and upgraded with the new features.

4 CONCLUSION

Once designed, formally verified, breadboarded, and validated, the three new features presented in this paper will be handed over to the SpaceWire Working Group for endorsement. They will then be integrated, together with a list of minor improvements to the current SpaceWire standard [1] endorsed by the SpaceWire Working Group, into a "SpaceWire 1.1" updated version of the SpaceWire standard. This version will then be subject to formal standardisation by the European Cooperation for Space Standardisation (ECSS).

Following requests from the SpaceWire community, ESA is also preparing for the medium term extension of SpaceWire to new domains such as Gigabit-per-second (Gbps) communications and reliability and real-time capabilities ("SpaceWire 2.0"), in parallel with the short term effort to have the currents SpaceWire standard revised.

5 REFERENCES

1. ECSS-E-ST-50-12C, "SpaceWire – Links, nodes, routers", 31 July 2008
2. ECSS-E-ST-50-51C, "SpaceWire protocol identification", 5 February 2010
3. ECSS-E-ST-50-52C, "SpaceWire - Remote memory access protocol", 5 February 2010
4. ECSS-E-ST-50-53C, "SpaceWire - CCSDS packet transfer protocol", 5 February 2010
5. Yuriy Sheynin, "Next release of the SpaceWire standard - some requests for change", 14th SpaceWire Working Group, ESTEC, February 2010
6. Martin Süß, "SpaceWire Standard Evolution", International SpaceWire Conference, Nara, November 2008
7. David Jameux, "SpaceWire for Command & Control", 10th SpaceWire Working Group, ESTEC, February 2008
8. David Jameux, Albert Florit Ferrer, "Towards the definition of Quality of Service classes for SpaceWire-based message passing", October 2007
9. Martin Süß, "SpaceWire Nodes", International SpaceWire Conference, Saint Petersburg, June 2010
10. Barry Cook, Paul Walker, "PnP aspects, 4Links contribution", 8th SpaceWire Working Group, ESTEC, January 2007
11. Albert Ferrer Florit, "PnP aspects, ESA contribution", 8th SpaceWire Working Group, ESTEC, January 2007
12. ESA & NASA, "ESA and NASA requirements on SpaceWire PnP", March 2007,
13. Peter Mendham, SpaceWire-PnP Protocol Definition, Draft A Issue 2.1, 16th September 2009
14. Prof. Yuriy Sheynin, "Distributed Interrupts in SpaceWire Interconnections", International SpaceWire Conference, Nara, November 2008
15. Prof. Yuriy Sheynin, "Distributed Interrupts in SpaceWire Networks", December 2006
16. Liudmila Onishchenko, Artur Eganyan, Irina Lavrovskaya, "Distributed Interrupts Mechanism Verification and Investigation by Modeling on SDL and SystemC", International SpaceWire Conference, Nara, November 2008
17. Steve Parkes, "The Operation and Uses of the SpaceWire Time-Code", International SpaceWire Seminar, ESTEC, 2003
18. Eugenue Yablokov, "Simplex Mode in SpW Technology", International SpaceWire Conference, Dundee, September 2007
19. Barry Cook, "Half Duplex SpW", 13th SpaceWire Working Group, ESTEC, September 2009
20. Barry Cook, Paul Walker, "Half-duplex SpaceWire: Reducing harness mass while retaining full compatibility with SpaceWire's modularity, configurability and adaptability", IAC-08, September 2008

DETERMINISTIC IMPLEMENTATION OF SPACEWIRE ON DATA RECORDER AND PAYLOAD INTERFACE UNITS

Session: SpaceWire Onboard Equipment and Software

Short Paper

Satoko Kawakami, Kazuyuki Yamada, and Hiroki Hihara

NEC TOSHIBA Space Systems, Ltd., 10, Nisshin-cho 1-chome, Fuchu, Tokyo, Japan

Kazuyo Mizushima, Takashi Kominato, and Kuniyuki Omagari

NEC Corporation, 10, Nisshin-cho 1-chome, Fuchu, Tokyo, Japan

Masaharu Nomachi

Laboratory of Nuclear Studies, Graduate School of Science, Osaka University,

1-1 Machikaneyama, Toyonaka, Osaka 560-0043

Takahiro Yamada, Motohide Kokubun, and Tadayuki Takahashi

Institute of Space and Astronautical Science (ISAS), Japan Aerospace Exploration Agency (JAXA), 3-1-1 Yoshinodai, Sagamihara, Chuo-ku, Kanagawa 229-8510, Japan

E-mail: s-kawakami@bk.jp.nec.com ,k-yamada@ea.jp.nec.com ,h-hihara@bc.jp.nec.com, nomachi@lns.sci.osaka-u.ac.jp, tyamada@pub.isas.jaxa.jp, kokubun@astro.isas.jaxa.jp, takahasi@astro.isas.jaxa.jp

ABSTRACT

Data recorders and payload interface units have been developed for ASTRO-H space X-ray observatory scheduled to be launched in 2014 using deterministic implementation of SpaceWire protocol interfaces. Data transmission with RMAP (Remote Memory Access Protocol) for the data recorder is realised in deterministic way, as such implementation of SpaceWire applied on the communication interface enables preventing congestion between prompt recording of scientific data and regular recording of house keeping data. Payload interface units developed for ASTRO-H in the same scheme consist of TCIM (Telemetry and Command Interface Module) and MSE (Mission Support Equipment). The purpose of these equipments is to translate legacy communication protocols of as-built design into SpaceWire.

1 DATA RECORDER

1.1 OUTLINE

Japanese X-ray astronomy satellite, ASTRO-H, has multiple scientific instruments to observe variety of X-ray sources in the sky [1]. Data produced from these sources varies with time and the data rate is often difficult to predict due to the nature of

sources. Therefore a data recorder is required to have flexibility to handle randomly produced data from these instruments. The requirement of the memory size is 2Gbytes which is sufficient to handle data taken every day. This data recorder has been developed on these requirements for ASTRO-H based on SpaceWire [2] and other future spacecrafts. As the feature of the data recorder, it enables recording of scientific data in RMAP initiator mode as well as recording of house keeping data in RMAP target mode. The data recorder accepts input data through the SpacePacket interface with the SpaceWire format. The input data is checked whether it organizes the right RMAP packet format. Then it is classified by the data mode, and the data recorder starts to operate the recording or the real time reproducing or the both. As the data recorder has RMAP, the data recorder has achieved to record the intermittent scientific data and the regular house keeping data at the same time.

Since the data recorder employs the deterministic implementation of SpaceWire protocol, it is not necessary to accommodate software. Therefore, this data recorder realised as the A6 size (150mm x 140mm) equipment which consists of all hardware including 16Gbits SDRAM memory modules.

Figure 1: Data Recorder structure image

1.2 THE PROTOCOL OF THE DATA RECORDER

The data recorder has an original protocol stack which is complied with the SpaceWire-D draft specification protocol stack [3]. Figure 2 shows the SpaceWire-D draft protocol stack and the data recorder protocol stack. The figure shows that the data recorder keeps the structure separated by the protocol layer clearly. That is, the data recorder is realised RMAP in deterministic way. The data recorder has no implementation for SpaceWire-D and SpaceWire-R in itself as shown in figure 2. These two protocol layers are accommodated in an attached on board computer.

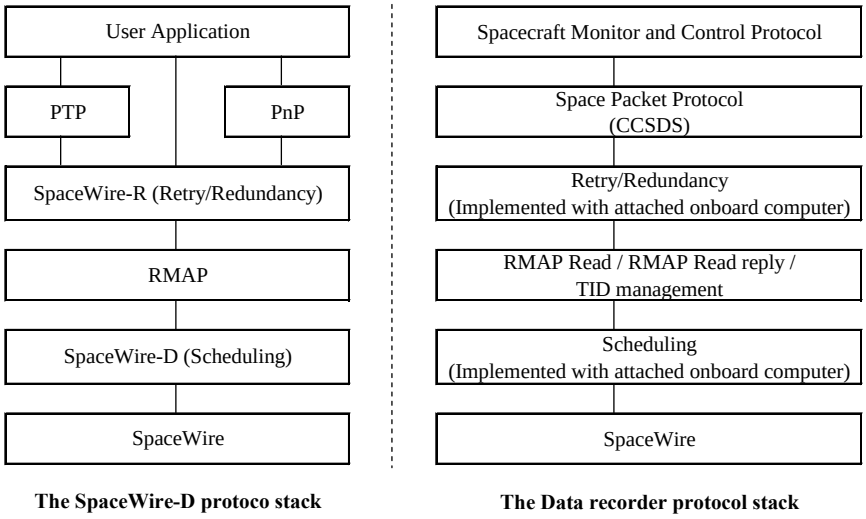


Figure2: The SpaceWire-D and the data recorder protocol stack

1.3 DATA RECORDER OPERATION

The data recorder for ASTRO-H operates recording the scientific data in the RMAP initiator mode, and the house keeping data in the RMAP target mode. The data recording operation is shown in figure 3. The input data goes through the SpaceWire interface at first, and the data is processed by the RMAP codec. Then the data recorder starts data transfer for recording into main memory and/or transfers the data for real time reproducing to TCIM through the RMAP codec.

The RMAP initiator mode recording is operated as follows; at first, the Data Recorder receives the RMAP write command to see a plan for collecting data. Then the data recorder creates and sends the RMAP read command to target nodes according to the received RMAP write command. In the process of creating the RMAP read command, the data recorder gives the original transaction ID and CRC. As the data recorder receives the RMAP read reply from the initiator nodes, the transaction ID and RMAP packet format including the CRC are checked. If they have no error, the data recorder starts recording that data to the main memory.

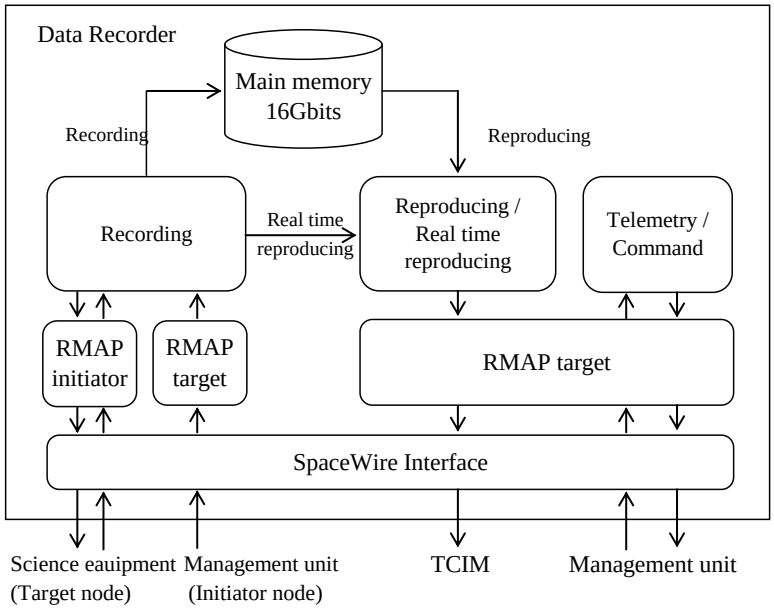


Figure3: The data recording operation

2 PAYLOAD INTERFACE UNITS

2.1 TCIM (TELEMETRY AND COMMAND INTERFACE MODULE)

TCIM has two main functions. Both of them are important for spacecraft bus system because this module is connected to both transponder and onboard computer and all telemetry and telecommand go through this module.

(1) Translate telecommand into SpaceWire format and telemetry into transponder format

The transponder receives telecommand from the ground station and transmits it through its serial communication interface. TCIM receives the signal and translate it into the SpaceWire format, then sends it out through the SpaceWire network. In the other direction, TCIM receives telemetry from the SpaceWire and translates it into the transponder format, then sends it out through serial communication interface.

(2) Translate legacy interface of RF (Radio frequency) communication equipment into SpaceWire

TCIM also has legacy interfaces for transponders or other RF communication equipments as pulse command interface to turn the equipments on or off and as bi-level telemetry interface to monitor the status of the equipments.

In ASTRO-H, three TCIMs are used. Two of them have interfaces with S-band transponder and are used for the spacecraft telemetry downlink and telecommand uplink; they consist of redundant system in each other and automatically change over on detection of error in one TCIM. The other has interface with X-band transponder and is used for mission telemetry downlink. In case of emergency rescue operation of the spacecraft, a part of the spacecraft system can be directly manipulated from the ground throughout TCIM.

2.2 MSE (MISSION SUPPORT EQUIPMENT)

MSE has developed to translate all other mission specific legacy communication protocol into SpaceWire. In ASTRO-H one MSE unit is installed. MSE gathers telemetry of legacy interface equipments and stores them in it. The spacecraft onboard computer; namely "SMU" in ASTRO-H, acquires the telemetry from MSE using SpaceWire network and its deterministic protocol. Also, when a command is to be transmitted to a legacy equipment, SMU transmits the command to MSE by SpaceWire, and MSE translates it into legacy communication protocol and send it.

3 CONCLUSION

Thanks to the deterministic implementation of SpaceWire protocol, the data recorder, which realises RMAP initiator mode data recording as well as RMAP target mode, has been achieved the small size equipment without accommodating software. Deterministic protocol implementation is also useful for employing as-built equipment such as TCIM and MSE, because those equipments often accommodate deterministic communication specification for the transmission of command and telemetry based on legacy protocol. In order to implement deterministic protocol on to SpaceWire, a protocol layer for time slot control is separated from re-transmission mechanism and redundancy control, because RMAP packet format, which includes CRC, can be fully exploited for diagnosis and re-transmission purpose leaving the time slot control capability within SpaceWire protocol layer. This scheme is formalised in SpaceWire–D draft specification and adopted for ASTRO-H.

4 REFERENCES

1. T. Takahashi, et al., "The ASTRO-H Mission", SPIE, 7732, 77320Z, 30 July 2010

2. ISAS/JAXA, ASTRO-H System Design (ASTH-100)

3. Takahiro Yamada, "Results of Analysis for SpW-D Draft Specification", Fifteenth SpaceWire Working Group Meeting, ESTEC, Netherlands, 18 October 2010

4. ESA-ESTEC Requirements & Standard Division, "Space engineering SpaceWire protocols", ECSS-E-ST-50-11C Draft 1.3, July 2008.

5. ESA-ESTEC Requirements & Standard Division, "Space engineering SpaceWire –Remote memory access protocol", ECSS-E-ST-50-52C, 5 February 2010.

THE SPACEWIRE LINK ANALYSER MK2

Session: SpaceWire test and verification

Short paper

Chris McClements, Stephen Mudie, Pete Scott, Stuart Mills, Steve Parkes

STAR-Dundee Ltd, Units 11&12, Dundee University Incubator, James Lindsay Place, Dundee Technopole, Dundee, DD1 5JJ, UK

E-mail: chris@star-dundee.com, stephen.mudie@star-dundee.com, pete@star-dundee.com, stuart@star-dundee.com, steve@star-dundee.com

ABSTRACT

The STAR-Dundee SpaceWire Link Analyser Mk2 is a key piece of equipment when performing test, validation and verification of a SpaceWire [1] system. The analyser sits between two SpaceWire devices and monitors traffic in both directions of the link providing the user with the functionality to monitor, record and analyse SpaceWire traffic. The new features of the SpaceWire Link Analyser Mk2 make it an invaluable tool when testing, debugging, validating or verifying any type of SpaceWire equipment.

INTRODUCTION

The SpaceWire Link Analyser Mk2 is the second generation of the STAR-Dundee link analysis solutions [2] and is designed to specifically support the testing and debugging of SpaceWire systems by providing a rich set of test functionality. The analyser benefits from increased traffic storage capacity which is up to 2000 times the capacity of the original analyser allowing millions of events to be stored in both directions of the link. RMAP and custom protocol analysis is supported, considerably reducing the effort required to capture and analyse RMAP traffic. Trigger in and trigger out ports can be configured to allow interaction with external equipment and provide a trigger source for an external scope or logic analyser. The analyser also has a Mictor breakout port which makes decoded SpaceWire traffic available to an external logic analyser.

The analyser is provided with a comprehensive set of software including an easy to use graphical user interface with context sensitive help and a new analysis API. This API exposes the full set of analysis features to automated user test suites where analysis can be coordinated with other test equipment.

OVERVIEW

The SpaceWire Link Analyser Mk2 hardware unit is depicted in Figure 1 and an example setup of the analyser is depicted in Figure 2.

Figure 1 SpaceWire link analyser hardware unit

On the front panel are two SpaceWire ports, input and output trigger connectors and status LEDs for the ports and triggers. To use the analyser a SpaceWire cable is connected from each device to be monitored and to the analyser. The link analyser buffers the LVDS signals internally and analysis is unobtrusive. A new feature of the analyser is the inclusion of an input trigger and an output trigger to allow cross triggering and synchronisation with other external EGSE equipment. Link status, error and data transfer information is provided by the SpaceWire status LEDs and trigger activity is provided by the trigger LEDs.

A Mictor connector is provided on the rear panel of the Link Analyser Mk2 to allow the analyser to be connected directly to a Logic Analyser. The SpaceWire traffic in each direction of the link is decoded into a set of characters which are provided on the logic analyser connector. The analyser connects to a host PC through the USB 2.0 interface and is powered by a provided 5V power brick.

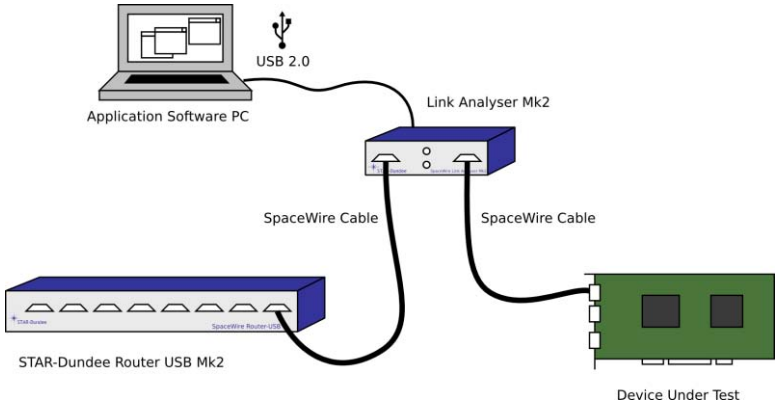


Figure 2 SpaceWire Link Analyser Mk2 example configuration

The analysis software which runs on the Application Software PC supports Windows (7, Vista, XP and 2000) and Linux (2.6 kernel) systems.

FUNCTIONALITY

The link analyser operates using a trigger to capture an event and a storage memory to capture the data which occurs before and after an event. The analyser software is used to setup the trigger condition, start and stop the analyser's trigger, monitor the trigger status and display the stored data when the trigger occurred. The link analyser also has a status monitoring function which provides an updating display, updated once per second, of the traffic on the SpaceWire link.

The analyser trigger condition can be set to capture one or more events on the link including: link errors, NULL, FCT and data characters or data packet comparators. A sequence of up to eight triggers can be set. Dependent on the debugging level the analyser can be configured to capture all link characters or can be set-up to filter out link control characters and only capture data. This greatly increases the amount of data storage available.

When the trigger condition has been met, and data has been stored in the analyser's deep internal memory, the data can be viewed using the analyser's extensive SpaceWire traffic displays including: a character level display which displays all link control, error and data information; a packet level display which can display raw data or protocol encoded traffic; and a bit level display which displays the raw data-strobe bits around the trigger condition at a resolution of 1.25 ns per sample. A new search feature has been added to the software providing the ability to quickly find information in the large storage memory. The character level and packet level displays are illustrated in Figure 3.

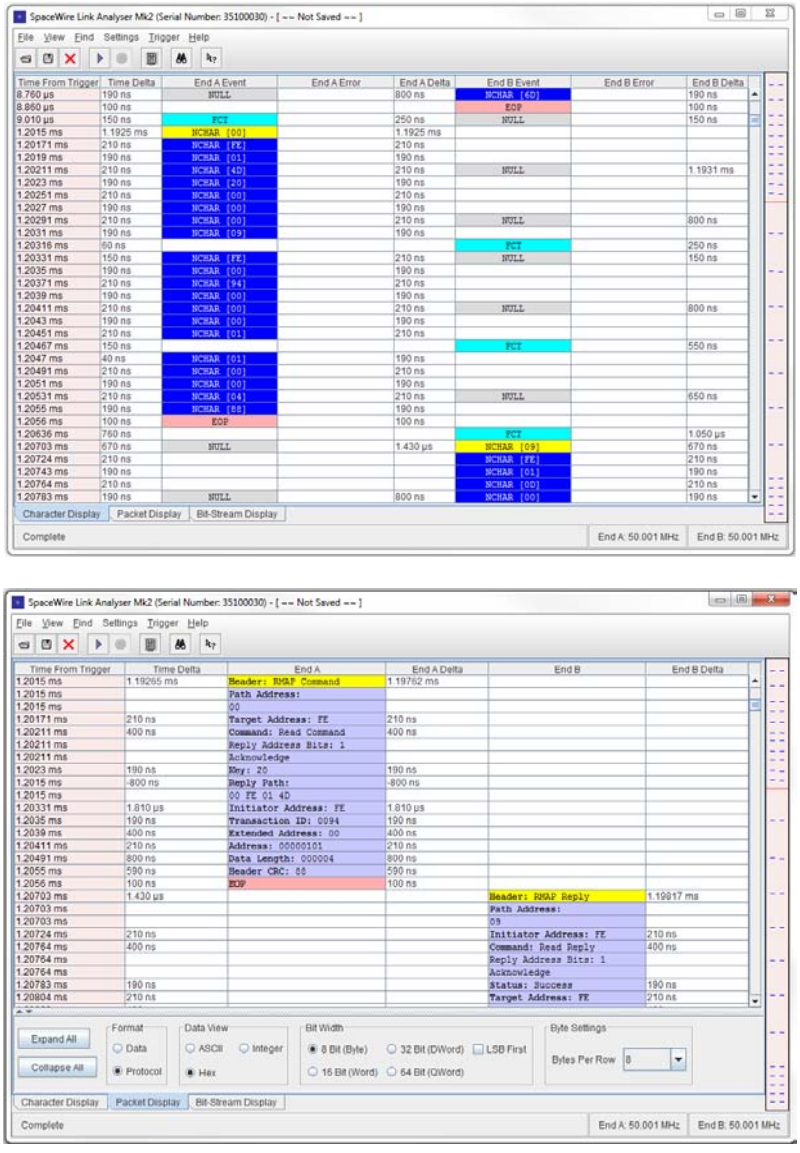


Figure 3 Character level and Packet level displays

Offline analysis is supported using the save and storage functions of the software. Recorded data can be saved in Link Analyser format for future analysis or saved to a text format file for display in other software tools. The software is also capable of saving raw N-Char data values (excluding EOPs and EEPs).

The functionality available in the Link Analyser software is replicated in an easy to use Application Programming Interface. For EGSE purposes the collection and analysis of the operation of the SpaceWire links often needs to be automated and coordinated with the operation of other test equipment. To support this, the Link Analyser Mk2 is provided with an API for C.

PERFORMANCE AND RESULTS

The SpaceWire link analyser Mk2 is capable of monitoring links running up to 400 Mbit/s and the bit-stream level display is capable of capturing data-strobe bit transitions at a rate of 1.25 ns (800 MHz).

The storage capacity of the link analyser has been greatly increased and up to 16 Million (16 Mebi) events can be captured using the C API and 1 million (1 Mebi) events in the software user application.

CONCLUSION

The new features of the SpaceWire Link Analyser Mk2 make it an invaluable tool when testing, debugging, validating or verifying any type of SpaceWire equipment.

REFERENCES

[1] European Cooperation for Space Standardization, Standard ECSS-E-ST-50-12C, “SpaceWire – Link, Nodes, Routers and Networks”, European Cooperation for Space Standardization, July 2008.

[2] S. M. Parkes, C. McClements, S. J. Mills and I. Martin, "SpaceWire: IP, components, development support and test equipment", DASIA Data Systems in Aerospace, SP-532, Prague, Czech Republic, June 2003

SPACEWIRE THERMAL INTERFACE NODE FOR SATELLITE THERMAL CONTROL

Session: Onboard Equipment and Software (Poster)

Short Paper

Minoru Nakamura, Tatsuya Ito, Yasutaka Takeda

Advanced Technology R&D Center, Mitsubishi Electric Corp., 8-1-1 Tsukaguchi-Honmachi, Amagasaki, Hyogo, 661-8661, JAPAN

Isao Odagi, Ichiro Takahashi, Toshihiro Obata

Kamakura Works, Mitsubishi Electric Corp., 325 Kamimachiya Kamakura, Kanagawa 247-8520 Japan

Ryoichiro Yasumitsu

Space Systems Div., Mitsubishi Electric, Corp., 2-7-3 Marunouchi Chiyoda-ku, Tokyo 100-8310 Japan

E-mail: Minoru.Nakamura@ea.MitsubishiElectric.co.jp,
Ito.Tatsuya@ak.MitsubishiElectric.co.jp,
Takeda.Yasutaka@aj.MitsubishiElectric.co.jp,
Odagi.Isao@cb.MitsubishiElectric.co.jp,
Takahashi.Ichiro@dx.MitsubishiElectric.co.jp,
Obata.Toshihiro@dp.MitsubishiElectric.co.jp,
Yasumitsu.Ryoichiro@aj.MitsubishiElectric.co.jp

ABSTRACT

The SpaceWire thermal interface node is a small piece of equipment that integrates thermal sensors and heaters into the SpaceWire network. In traditional satellite architecture, a lot of thermal sensors and heaters are directly connected to a thermal control unit. We propose a novel concept for a distributed satellite thermal control architecture in which thermal sensors and heaters are connected via distributed SpaceWire thermal interface nodes. The SpaceWire thermal interface node has a few A/D converter channels for a thermal sensor interface and a few solid-state switches for controlling thermal heater power. It provides a higher level thermal control interface that automatically converts a sensor value to a temperature value. It also provides intelligent thermal control functions in that the SpaceWire thermal interface node automatically controls heater switches to maintain a temperature specified by the satellite controller. This feature is represented by a small controller implemented in the SpaceWire thermal interface node. Therefore, the SpaceWire thermal interface node will improve the flexibility of satellite thermal control and reduce harness mass. In this paper, we present a distributed thermal control system concept demonstrated with a prototype of the SpaceWire thermal interface node. We also present an early implementation of the SpaceWire thermal interface node. Also, we discuss the topology of the distributed thermal control system and the control methods of the satellite thermal controls.

1 SATELLITE THERMAL CONTROL

One major task of satellite management is thermal control. This control measures the temperature of the measuring points on the satellite components and the satellite body panels and controls heaters to maintain a temperature within a specified thermal range. The number of the measuring points depends on the satellite size and/or the number of components. Scores of measuring points exist. In traditional satellite design, a lot of heaters and thermal sensors are connected to the thermal control unit and controlled by the satellite controller. This thermal control system is distributed through the entire satellite and requires many harnesses. In recent satellites, bus components are connected via a local area network like SpaceWire, and this represents the flexibility of modern satellite design. The satellite thermal control system uses many sensors and heaters that are distributed through the entire satellite and also requires analogue signals and power switches. This is because the thermal control system is difficult to integrate into the satellite bus network.

2 THERMAL CONTROL OVER THE NETWORK

We considered that network architecture that integrates the thermal control system into the satellite network is classified into two methods:

1. All heaters and thermal sensors are connected to and controlled by the thermal control unit in a traditional manner, and the thermal control unit is connected to a satellite network. A thermal control unit controlled via a satellite network is shown in Fig. 1.

2. One or few heaters and thermal sensors are connected to a satellite network via a small network adopter, as shown in Fig. 2.

Method 1 requires minimum modification to the traditional design, but its advantage is limited to the satellite controller being able to control the thermal system via a network. Method 2 has the disadvantage of requiring a lot of network adopters and the advantages of reducing the number of harnesses and integrating time in addition to method 1.

We assumed that a network based on a thermal control system like method 2 could be realized with a sufficient number of advantages if a small network adopter were used. We named the small network adopter “SpaceWire thermal interface node” and developed a functional prototype model.

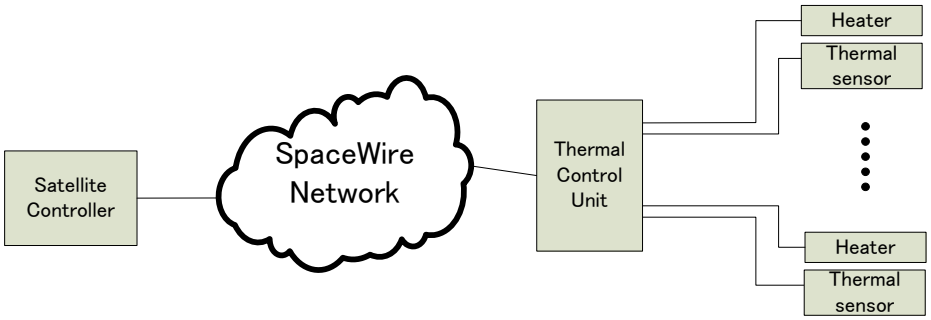


Fig. 1: Traditional thermal control system with SpaceWire interface

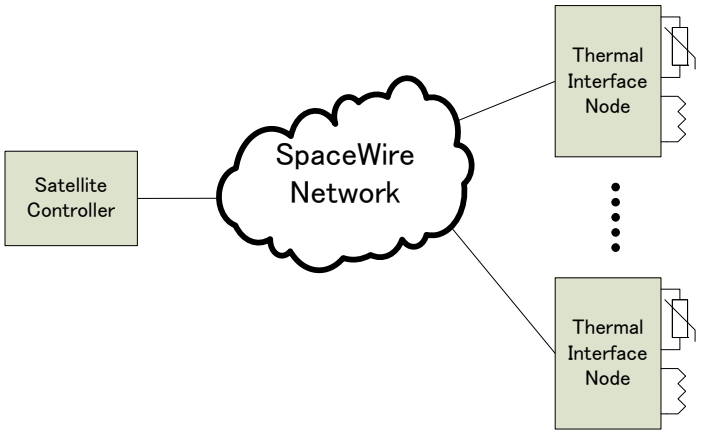


Fig. 2: Network based thermal control system

3 SPACEWIRE THERMAL INTERFACE NODE

The functional prototype model of the thermal interface node has one pair of heater switches, a thermal sensor interface, and two SpaceWire interfaces on two pieces of 5 cm by 6 cm PCBs. Figure 3 shows the architecture of the thermal control node, and Figure 4 shows the prototype model.

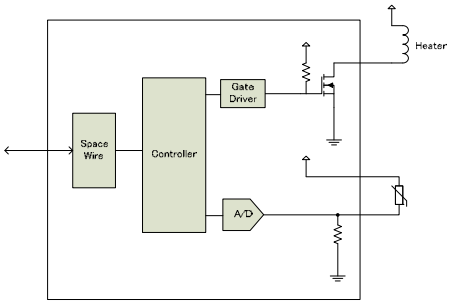


Fig. 3: Internal architecture of thermal interface node

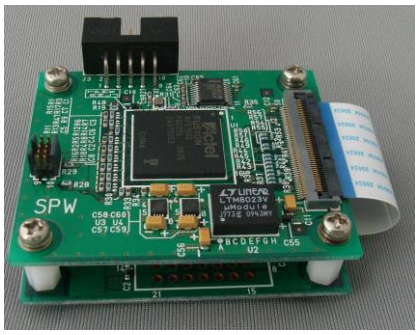


Fig. 4: Prototype module of SpaceWire thermal interface node

This thermal control node has a programmable control mechanism that controls the heater switch automatically within a specified temperature. It also has the ability to convert thermal sensor output into an actual temperature. This functionality is represented by an abstract thermal interface, so the satellite controller needs no S/W modification when implemented in a different satellite and/or if the sensor or heater is changed. Also, the satellite controller only needs to control the high level thermal control because the low level thermal control is implemented in the thermal control node.

Figure 5 shows the topology of the thermal control network. We will use ring topology to reduce cable mass with enough redundancy. To improve reliability, the thermal control network is divided into multiple rings, and two or more independent rings control each satellite panel or the thermal sensitive components. Figure 6 shows

an example of the thermal network configuration. In Figure 6, the ring topology is simplified as a single line, but actual wiring requires a return path.

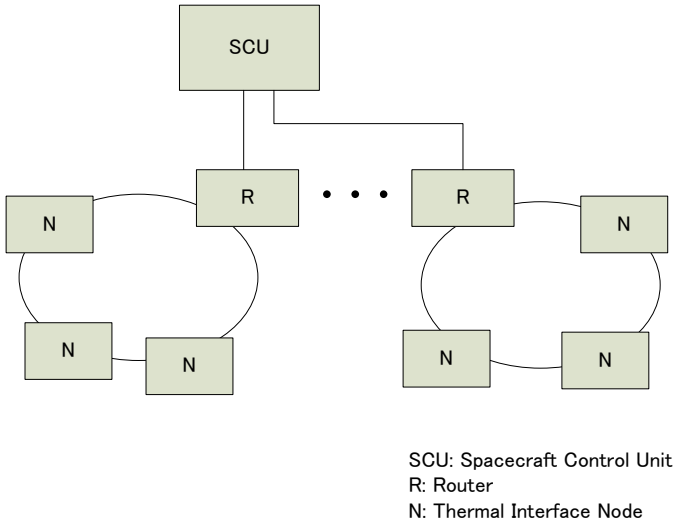


Fig. 5: Topology of thermal control network

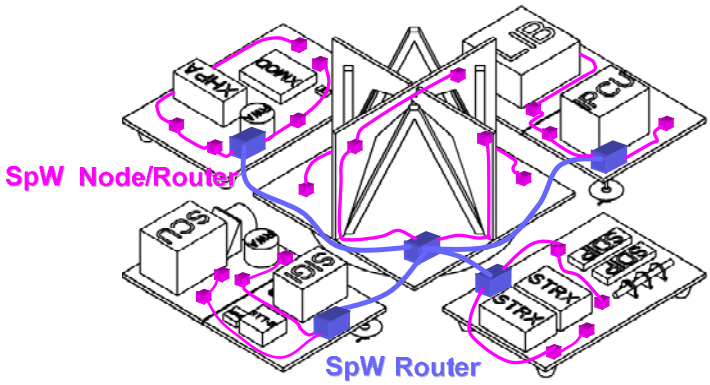


Fig. 6: Example of network configuration

4 SUMMARY

We presented a SpaceWire thermal interface node and a satellite thermal control system that uses these thermal interface nodes. This thermal control system reduces harness weight and improves reliability. Also, the thermal interface node enables the abstraction of the thermal control interface because the thermal interface node can execute a part of the thermal control process. However, to reduce harness mass, a half duplex and/or low-mass cable specifications are required. We will evaluate a thermal control system that uses the thermal interface node.

CAMERA SIMULATOR FOR PLATO MISSION

Session: Space Wire Missions and Applications

Short Paper

Vanderlei Cunha Parro, Sergio Ribeiro Augusto, Rafael Corsi Ferrão e Tiago Sanches da Silva

Mauá Institute of Technology – Praça Mauá, 01 – CEP 09580-900 – São Caetano do Sul – SP Brazil

Philippe Plasson and Loic Gueguen

LESIA / Paris-Meudon Observatory – 5, Place Jules Janssen – 92195 – Meudon - France

E-mail: vparro@maua.br, sergioribeiro@maua.br, corsiferrao@gmail.com, tiago.eem@gmail.com, philippe.plasson@bbspm.fr, loic.gueguen@obspm.fr

ABSTRACT

This work describes the development of an Electrical Ground Support Equipment (EGSE) to be used by the flight software development team and during the integration and test activities for the PLATO (PLAnetary Transits and Oscillations of stars) Mission. The EGSE will be used to feed the Data Processing Units (DPUs) with dynamic scientific data, representative of expected sky scenarios, using SpaceWire links. Its functionality is fully compliant with the real camera specification. The main system is implemented in a Altera Stratix IV FPGA (VHDL language). This work presents the electrical and software architecture used to implement the EGSE. The main point is the conversion of USB packets to the RMAP SpaceWire packets, under the PLATO electrical constraints, putting in evidence hardware and firmware solutions.

KeyWords: SpaceWire, RMAP, VHDL, FPGA, Embedded System

1 OVERVIEW PLATO SYSTEM

The scientific goal of PLATO [1] is the discovery and study of extrasolar Planetary System by means of planetary transits detection.

The instrumental concept proposed by the PLATO Payload Consortium is based on a multicamera approach. There are 32 normal cameras arranged in four sub-groups of 8 cameras, and 2 fast cameras working independently. Each camera is equipped with its own CCD (Charge Coupled Devices) focal plane array, comprised of 4 CCDs. The CCDs work in full frame mode for the normal cameras, and in frame transfer mode for the fast cameras (attitude control). The proposed system simulates only normal cameras and normal front end devices (N-FEE).

There are 16 normal data processing units (N-DPU). Each N-DPU is responsible for processing the data of 2 normal cameras belonging to 2 separate optical groups. The processing cadence for N-DPUs is 25 seconds, and each camera has one N-FEE associated.

2 EGSE DESCRIPTION

The EGSE tests a half N-DPU, sending one complete camera image through two links SpaceWire [2] with RMAP [3]. The Figure 1 gives an overview of the EGSE. The frame "Plato Image Subsystem" illustrates the system implemented in the satellite and that is emulated in this work.

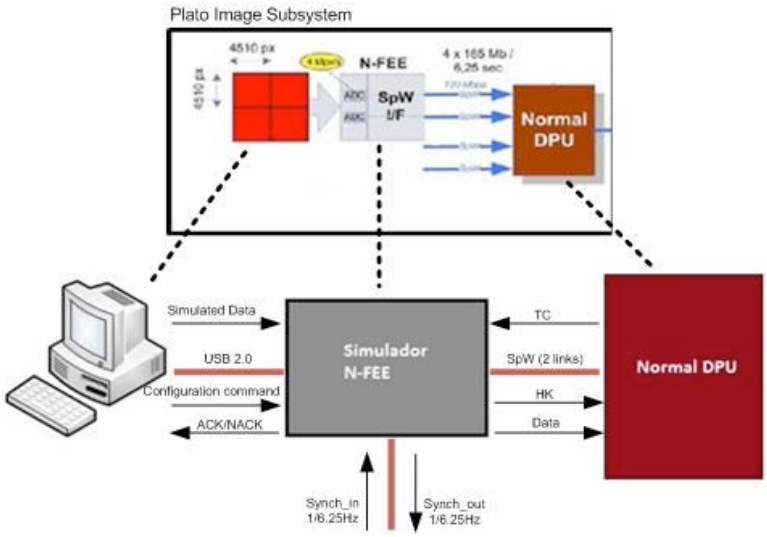


Figure 1: EGSE overview

The EGSE is composed by two subsystems: 1- a workstation running an user interface and responsible for sending images through USB (Universal Serial Bus) to the N-FEE simulator, like a normal camera; 2- the N-FEE Simulator, that sends the image through SpaceWire using RMAP protocol.

2.1 N-FEE SIMULATOR DESCRIPTION

Each N-FEE in the PLATO is responsible for digitize the video signal, send the digitized image to the N-DPU over a SpaceWire link using the RMAP protocol, receive and execute commands from N-DPU, receive and propagate synchronization signal, as well as manage housekeeping (HK). A full CCD image transfer (around 39 MBytes) starts at the time the synchronization (*Synch_in*) signal is received, needing to be performed in less than 3.3 sec. The remainder time is used to transfer the image to a memory zone. The figure 2 illustrates the time constraints involved in the application and which is emulated in the EGSE. The *Synch_out* is the synchronization signal propagated through the internal logic.

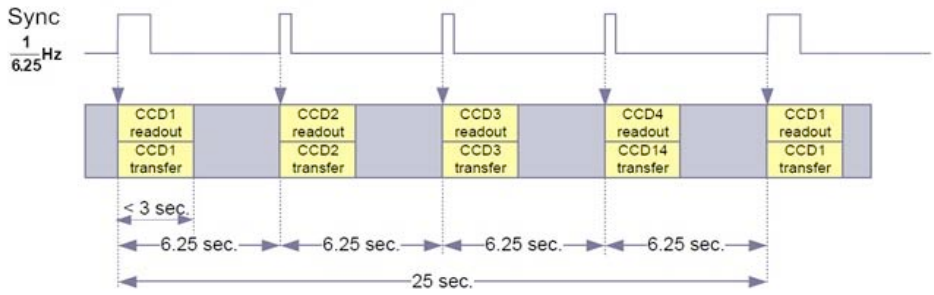


Figure 2: Time constraints, Sync is synchronization signal

The RMAP protocol write command is used to transfer data from N-FEE to the N-DPU, one write command per half line, and, in the opposite direction, from N-DPU to N-FEE, (ie: memory address, operation mode). A RMAP read command is used from N-DPU to N-FEE to access the housekeeping information.

The two SpaceWire links run at 100 Mbits resulting in a instantaneous data rate (with 25% SpaceWire overhead and RMAP header), for full CCD transfer, of 80 Mbits and an averaged transfer at 70.5 Mbits. An SpaceWire time code is sent by the N-FEE simulator to the 2 SpaceWire links at the time a synchronization signal (*Synch_in*) is received. This time code serves to synchronize the N-DPU with the external *Synch_in* signal (6.25s) and to indicate which CCD is being transferred (0, 1, 2, 3).

The N-FEE Simulator has four types of operation modes which can be set by the N-DPU: 1- **Operational mode**: the CCDs are read with the synchronization signals. Data packet including image and housekeeping are sent; 2- **Stand-By mode**: Only housekeeping data are sent on request from the N-DPU; 3- **Integration mode**: The N-FEE may function without synchronization signals from the N-AEU; 4- **Test Mode**: the N-FEE sends a data pattern to the N-DPU.

2.2 N-FEE SIMULATOR ARCHITECTURE

The architecture proposed to the N-FEE Simulator is given in the Figure 3. All architecture is embedded in a Stratix IV [4] FPGA. The USB module implements USB 2.0 interface acting as a device.

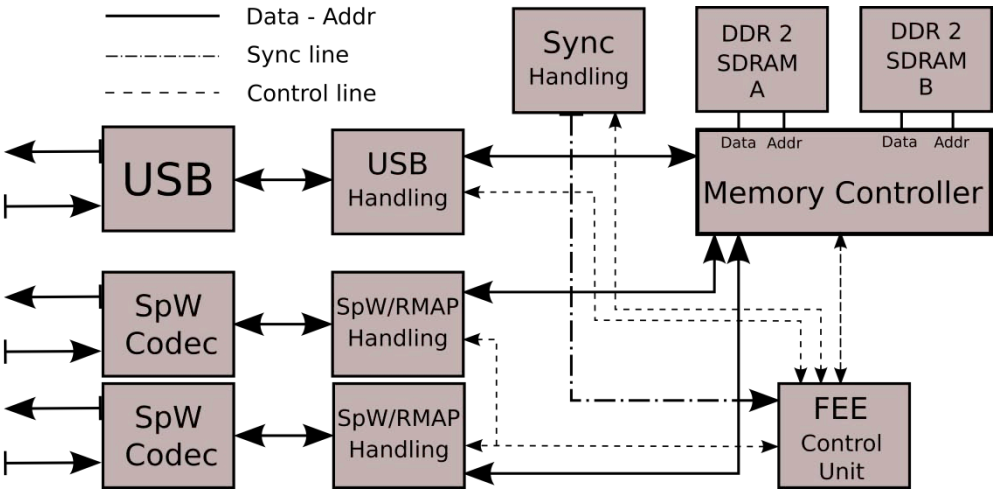


Figure 3: N-FEE Simulator hardware architecture

The image sent from the computer is stored in a one of the two DDR2 SDRAM memories in less than 6.25 seconds. When the synchronization signal (*synch_in*) is detected the N-FEE Unit Control (FEE UC) swaps the memory, connecting the fresh loaded memory to the SpaceWire/Rmap handler, and linking the other memory to the USB handling. So, while the workstation sends the next CCD data to the memory allocated for the USB handling, the SpaceWire/Rmap handler sends the image previously loaded in the other memory to the N-DPU.

Sync handling is responsible to detect the S*ynch_in* signal (6.25s and 25s signal that came in the same line). It has also the ability to generate the synchronization signal in order to the system work in the integration mode.

The data from the work station is sent to the USB2.0 as a 32 bytes burst transfer, and then the USB handler interprets and executes a write burst to the memory controller. The memory controller is implemented using an Altera IP core [5], and deals with the two DDR2 SDRAM of 1GB each one. There is an internal controller that only gives write or read access to one memory per time, this prevents reading unread data.

A SpaceWire/Rmap handler which interfaces with the SpaceWire codec is also implemented. This block is able to create an RMAP write command using the data from the SDRAM memories, respond to write commands (reply), record the data received on auxiliary internal memory, as well as respond to read commands using data from auxiliary internal memory.

The FEE UC control all the previously blocks, swapping the read/write access to the memories when the synchronization signal is received, configuring the blocks with the operation mode, as well as loading default values at start-up.

3 PRELIMINARY RESULTS AND CONCLUSION

The system is in the implementation phase using the Altera Quartus II V11 development environment. Gate level simulations have been made and the system is being able to reach the time constraints proposed. The architecture with two memories allows future improvement of the EGSE to support more than one N-FEE, simulating a more complex system.

4 REFERENCES

1 ESA/SRE(2011)13, PLATO Next-generation planet finder, Definition Study Report , July 2011, 121.

2 ESA-ESTEC, SpaceWire – Links, nodes, routers and network, ECSS-E-ST-50-12C, 31 July 2008, 129.

3 ESA-ESTEC, ESA Requirements and Standards Division ECSS-E-ST-50-11C - SpaceWire proto-cols ,18 November 2008, 124.

4 ALTERA corporation,A Design Guide for Stratix II, Stratix III, and Stratix IV Devices, June 2009, 122

5 ALTERA corporation, External Memory Interface Handbook Volume 5, June 2011, 70

SPACEWIRE REMOTE TERMINAL CONTROLLER DEVELOPMENT SYSTEM

Session: Onboard Equipment and Software

Short Paper

David Paterson, Alan Spark, Bruce Guoxia Yu

STAR-Dundee, c/o School of Computing, University of Dundee, Dundee, Scotland, UK

Steve Parkes

University of Dundee, School of Computing, Dundee, Scotland, UK

E-mail: david.paterson@star-dundee.com, alan.spark@star-dundee.com,

bruce@star-dundee.com, sparkes@computing.dundee.ac.uk

ABSTRACT

The STAR-Dundee SpaceWire Remote Terminal Controller (RTC) Development System is intended to facilitate the development of both hardware and software for spacecraft systems based on the Atmel AT7913 SpaceWire RTC device.

The development system hardware consists of an AT7913 device, plus an FPGA and additional on-board resources. The FPGA provides a number of different connection options between the RTC and a host computer, allowing the downloading and debugging of software, and the simulation of the RTC device interfaces.

The RTC can be connected to a number of internal, on-board, hardware resources, or to external equipment via case-mounted connectors. The FPGA allows for flexible control over these connections, switching among different configurations or working modes, and providing full control for debugging.

The software for the development system is based on the widely-used Eclipse IDE, and will be immediately familiar to existing Eclipse users. It also offers a shallow learning curve for new users, allowing them to quickly start developing or porting software for the RTC. A wide range of debugging features is available, including breakpoints, single-stepping (source and assembler), and inspection of memory and registers.

An additional system component provides facilities for the simulation of spacecraft instruments or other devices, with the generated data being passed to the RTC as if it were produced by the real instruments or devices.

This paper describes the main features of the RTC Development System, and examines its possible uses in spacecraft hardware and software development.

1 STAR-DUNDEE RTC DEVELOPMENT SYSTEM

The STAR-Dundee RTC Development System is designed to provide an all-in-one environment for developing, testing and debugging on-board hardware and software, and consists of a development board plus associated host computer software. The development board provides a number of interfaces which can be connected to the host computer to support program download, debugging and testing, as well as a range of other connections for external instruments or other equipment.

Figure 1 - SpaceWire RTC Development System

2 HARDWARE

The STAR-Dundee RTC development board is built around the SpW-RTC SpaceWire Remote Terminal Controller device, AT7913, from Atmel [1]. This single chip embedded system centres around a LEON2-FT (SPARC V8) processing unit, plus a double precision floating point unit. Connected to these via on-chip busses are several other peripheral modules, including CAN, ADC/DAC, GPIO, FIFO, timers and two SpaceWire interfaces supporting RMAP.

The RTC device also includes 64KBytes of EDAC protected on-chip memory, into which software can be copied via a SpaceWire link, allowing it operate as a single-chip system, forming a very compact solution for remotely controlled applications. Alternatively it can operate in a fully-featured system, with software being loaded from a local PROM and executed from a local SRAM. The development board includes 160Mbit of Flash PROM and 160Mbit of SRAM which can be used while developing this type of system.

The hardware unit is shown below in Figures 2 and 3. Connectors are provided for an interface to the host computer using high-speed USB, SpaceWire or RS232, as well as CAN, SpaceWire, GPIO, FIFO and ADC/DAC connectors for external devices.

Figure 2 - SpaceWire RTC Development Unit front view

Figure 3 - SpaceWire RTC Development Unit rear view

The hardware unit has been specifically designed to be compact and portable and is only 220 mm wide, 30 mm high and 115mm deep.

3 SOFTWARE

Eclipse is an open-source Integrated Development Environment (IDE) that has been widely adopted in software development, including development of embedded systems [2]. These factors made it an ideal choice for the front end environment for the RTC development system. The Eclipse environment has been extended to operate with a version of GCC that has been optimised for use on the RTC. Programs written in C or C++ can be compiled and then run and debugged on the development board.

Both the STAR-Dundee RTC hardware and software have been designed with simplicity in mind. The steps required to start debugging are minimal. This is essentially a case of connecting the device to the host PC and starting a debug session in Eclipse (see Figure 4).

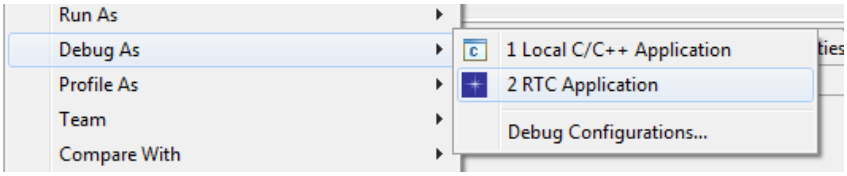


Figure 4 - Starting a debug session in Eclipse

When using a fast USB or SpaceWire connection between the host computer and the development board, Eclipse gives a smooth single stepping experience. While debugging, Eclipse provides familiar views to allow inspection of variable and register values (Figure 5).

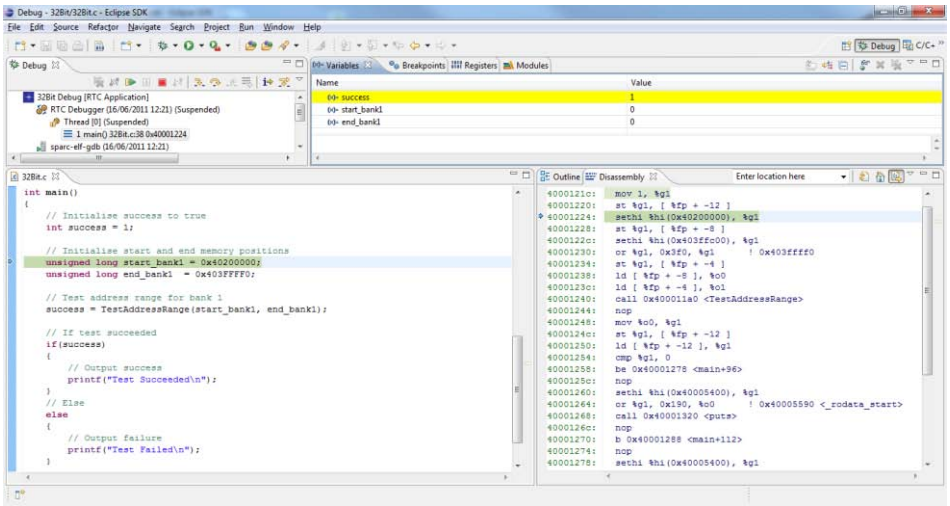


Figure 5 - Single-stepping through code running on the RTC

Custom views added to the Eclipse environment also provide access to other aspects of the hardware, such as the device register view shown in Figure 6.

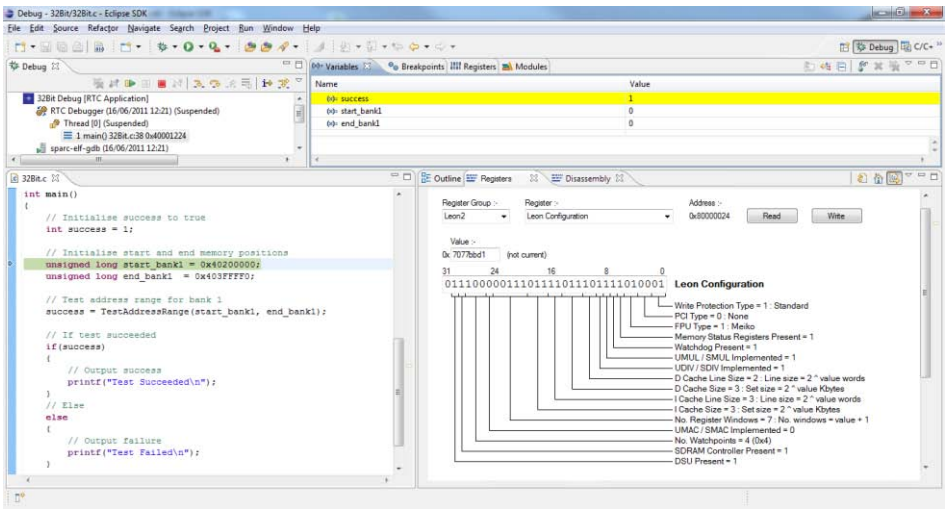


Figure 6 – Device Register View

Whilst making it easy to get up and running, the development environment also allows configuration of more advanced options such as the interface to use (e.g. USB or SpaceWire), the size of available memory, clock frequency, etc. [3]

4 CODE ROCKET

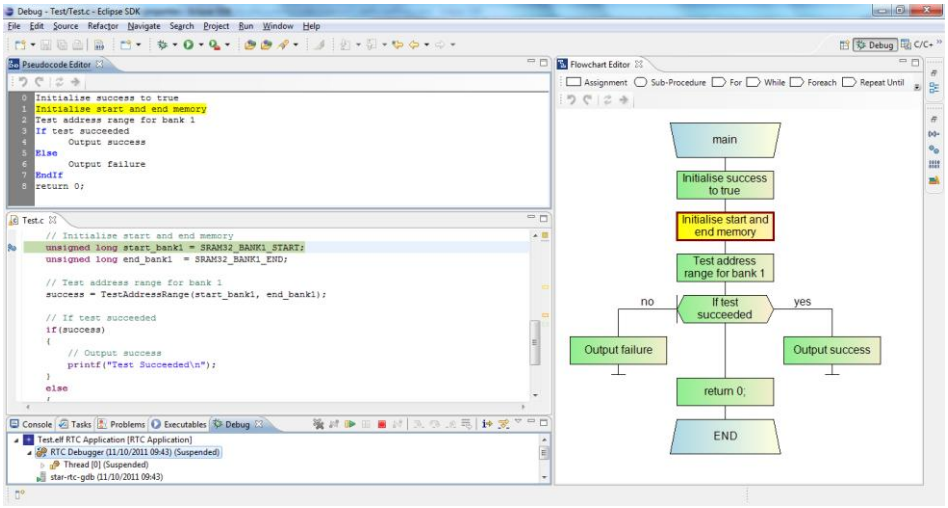


Figure 7 - Code Rocket design views integrated inside Eclipse

Code Rocket from Rapid Quality Systems [4] is a detailed software design tool that provides abstract pseudocode and flowchart visualisations of algorithms. Plugins are available for the Visual Studio and Eclipse IDEs that allow the developer to design and visualise methods on demand (see Figure 7). When changes are made in the code, the design views are automatically synchronised. Similarly, when changes are made in the pseudocode or flowchart editors, the method can be re-populated with the associated forward-engineered code. The synchronisation between code and design ensures that neither gets out of date and the ad-hoc nature of the tool means that it unobtrusively fits into the developer's working process.

A new feature in Code Rocket is the ability to see the current statement highlighted in the design views when stepping through code in the debugger. Whilst Code Rocket is already an extremely valuable resource for designing and understanding algorithms, this new addition makes it a powerful debugging tool too. Given that the RTC Development System has been designed specifically to speed up the development of on-board spacecraft software using Eclipse, Code Rocket fits naturally into this paradigm. When combined, both products provide a unique way of designing and executing software on the RTC hardware as well as a dynamic flowchart that is highlighted as a debug session is running on the RTC hardware unit. The combined system will be used by developers to ensure that their software meets the high quality demanded by space applications.

When stepping through code the current statement is highlighted in yellow, with a red outline if a breakpoint is encountered or with a green outline when stepping over other statements that don't have breakpoints (see Figure 8). The flowchart is naturally easy to follow and the addition of the debugging information makes it very easy to pinpoint exactly which part of the software is being debugged and the surrounding context.

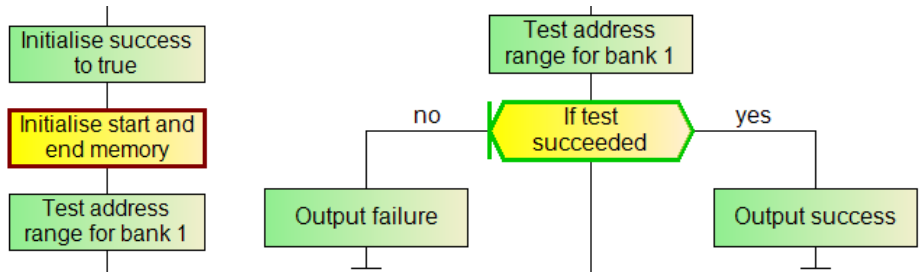


Figure 8 - Breakpoint highlighted with red outline and non-breaking statement highlighted with green outline

5 POTENTIAL APPLICATIONS

In the early development stages of a spacecraft instrument or payload, the software development team will not normally have a full hardware platform to work with. As a result, unexpected issues may be discovered much later during system integration. The STAR-Dundee RTC Development System aims to address this issue by providing virtual devices that are interchangeable with the actual devices that may be connected to the physical ports. This allows software development to progress before the physical hardware becomes available, and when the hardware does become available the actual devices can easily be substituted for the virtual ones, and then tested.

For example, the data stream from an onboard instrument which will be connected to the FIFO can be simulated, with data being provided by the host computer, to allow initial testing of its data management software before the hardware development is complete. When the instrument hardware is available it can be connected to the RTC development system via the external FIFO connector, and further testing carried out – the aim being to ensure that software and hardware can be thoroughly tested before integration with other flight systems.

The development board is also equipped with ADC and DAC chips which can be used for hardware prototyping. These chips are commercial counterparts of Space Qualified ASICs, so a prototype system based on this on-board ADC or DAC could be easily migrated to a flight qualified design.

Some of these possible applications are illustrated in Figure 9, including platform OBC, controller or data handler to instruments, bridge to low-speed CAN bus, separate data processor, and mass memory controller.

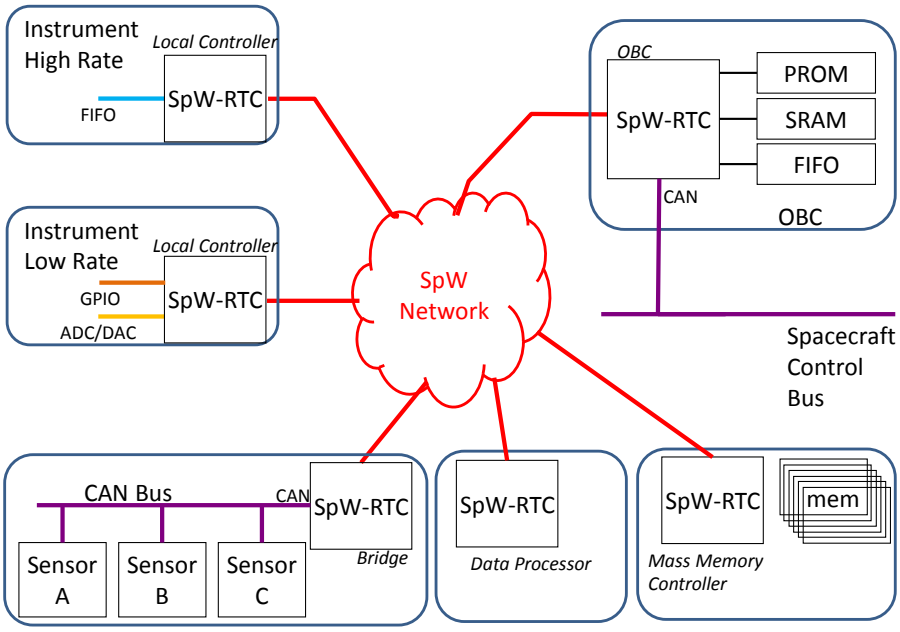


Figure 9 - Possible Applications of the SpW-RTC Device

5 CONCLUSIONS

The STAR-Dundee SpaceWire RTC Development System is a combination of test and development hardware and software tools for developing, testing and debugging onboard spacecraft systems. This paper has described and discussed both the hardware and the software tools used with it.

Alongside the compact but capable hardware unit, the software development environment has been designed with complete simplicity in mind. The ability to connect to the device and quickly start debugging is an improvement over existing solutions that may be over-complex for the end user.

When combined with Code Rocket, the RTC Software Development System becomes an all-in-one spacecraft software design, development, test and debugging toolset that paves the way for improved developer productivity and software quality.

5 REFERENCES

1. Atmel, “SpaceWire - Remote Terminal Controller DataSheet”, 7833F–AERO–01/10
2. Aurangzeb, “Eclipse and Embedded Software Development”, Proceedings of ICOSST 2007, 17-18 December 2007, Lahore, Pakistan.
3. STAR-Dundee, http://www.star-dundee.com, STAR-Dundee website.
4. Rapid Quality Systems, http://www.rapidqualitysystems.com, Rapid Quality Systems Website.

INCORPORATION OF SPACEWIRE WITHIN THE BEPICOLOMBO RIUS

Session: Poster Session

Short Paper

P. Worsfold, A.Senior.
SEA, Building 660, Bristol Business Park, Coldharbour Lane, Bristol, BS16 1EJ, United Kingdom

E-mail: peter.worsfold@sea.co.uk, alan.senior@sea.co.uk

1 ABSTRACT

Two Remote Interface Units (RIUs) have been developed by SEA for the upcoming ESA BepiColombo mission to Mercury. Each RIU will be used as a stand-alone unit within the BepiColombo Data Management System (DMS) and provide the connection between the onboard computer and a large range of sensors and actuators. One RIU provides sub-system interfaces within the Mercury Planetary Orbiter (MPO), whilst the second provides sub-system interfaces within the Mercury Transfer Module (MTM). The implementation details of SpaceWire interfaces within the two RIUs are described and the performances achieved discussed.

2 MPO RIU

The MPO RIU provides two SpaceWire links running at 10Mbps. These interfaces utilise an EIA-644 LVDS electrical layer compliant with the ECSS-E-ST-50-12C (SpaceWire – Links, nodes, routers and networks) standard. The MPO RIU incorporates the Aeroflex UT54LVDS031-LV LVDS quad transmitter and the UT54LVDS032-LV LVDS quad receiver for the SpaceWire interfaces. Figure 1 below shows the SpaceWire interface components for the MPO LVDS interface. Figure 2 below shows the SpaceWire outputs when the link is exchanging "null" packets.

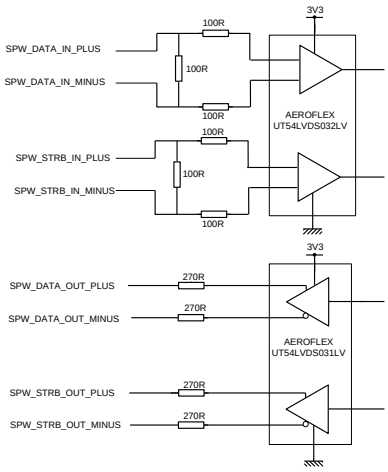


Figure 1: MPO RIU SpaceWire Interface

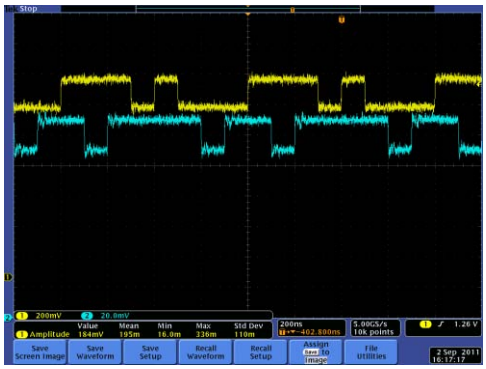


Figure 2: Waveform of MPO SpaceWire LVDS link – yellow trace (SPW_DATA_OUT_PLUS), green trace (SPW_STRB_OUT_PLUS).

3 MTM RIU

The MTM RIU provides two SpaceWire links running at 4Mbps (initialising at 10Mbps). This unit however utilises an SBDL (Standard Balanced Digital Link) electrical layer. This requires the use of EIA-422 devices. The MTM RIU incorporates the Intersil HS-26C31RH RS422 transmitter and the Intersil HS-26C32RH RS422 receiver for the SpaceWire interface. Figure 3 below shows the SpaceWire interface components for the MTM SBDL interface. Figure 4 below shows the SpaceWire outputs when the link is exchanging "null" packets.

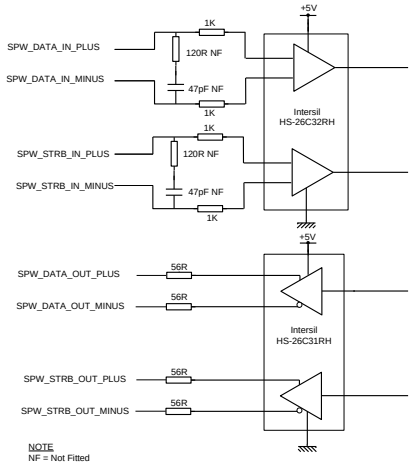


Figure 3: MTM RIU SpaceWire Interface

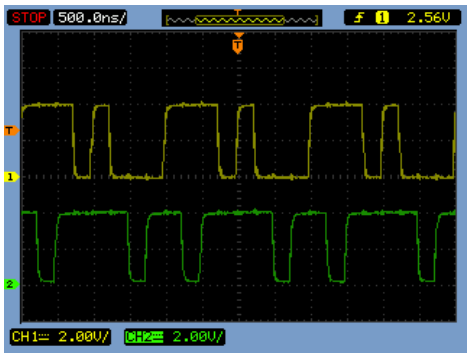


Figure 4: Waveform of MTM SpaceWire SBDL link – yellow trace (SPW_DATA_OUT_PLUS), green trace (SPW_STRB_OUT_PLUS).

4 SIGNALLING PERFORMANCE

The LVDS buffers can operate at a switching rate of 400Mbps (200MHz). Signal tracks on the PCBs have been routed in accordance with [RD1] to minimise the skew introduced by signal length mismatch. According to signal timing analysis the total cumulative skew is approximately 9ns (including external SpaceWire router). The link could therefore readily run at speeds in excess of 100Mbps, however in practice it is limited by performance limitations of the FPGA (see section 5).

The SBDL link utilised on the MTM is however far slower. The defined maximum data rate of an RS-422 link is 10Mbps over a length of 1200m with up to 10 receivers on the signal line. This electrical layer has been chosen to improve the common mode voltage / signal levels across the spacecraft separation interface. This improvement is gained by a much larger difference between the output voltage transition levels VOH and VOL, however a result of the increased signal swing is much longer signal transition times. Due to the performance of RS422 transmit/receive components, the link cannot meet the rise/fall/skew times specified within ECSS-E-ST-50-12C, however for the low data rate requirement of the MTM this is not an issue.

No channel to channel skew figures are available for the Intersil HS26C32RH and HS26C31RH RS422 devices, therefore max propagation delay – min propagation delay is taken as worst case skew. According to signal timing analysis the total cumulative skew is approximately 65ns (including external SpaceWire router). The link could theoretically therefore run at speeds in excess of 15Mbps.

5 FPGA PERFORMANCE

The BepiColombo RIU has two SpaceWire Controller modules. Each module contains an RTAX2000S FPGA containing two ESA RMAP IP cores with embedded SpaceWire CODECs [RD2]. Both RMAP IP cores are configured as Target only (to ensure that no messages are initiated by the RIU).

The RTAX FPGAs have limited global clock resources which causes a constraint on the implementation of the SpaceWire CODEC core when more than one is being instantiated within the same FPGA. The RTAX FPGAs have four hardwired clocks HCLKA/B/C/D and four routed clocks CLKE/F/G/H. Hardwired clocks are only routed to register clock inputs, whereas the routed clocks can be used for any global signals requiring low skew. It is standard practice for one of the routed clock nets to be utilised for the internal synchronous reset signal. A routed clock net is also required for each of the SpaceWire receive clocks (recovered by Exclusive OR of the incoming data and strobe signals. This is necessary to ensure low skew of the receive clock and follows the guidance within a Microsemi application note [RD3]. This only leaves one spare routed clock network left between the two SpaceWire CODEC transmit clocks.

The SpaceWire CODEC requires to have a transmit clock which runs at 10Mbps (link initialisation) and then to link running rate (variable by configuration). This can be achieved by utilising a configuration (known as SYS_DIV or TXCLK_DIV), which creates a divided version of the clocks however this creates asynchronous clock signals which require global nets to reduce skew. It can also be achieved by utilising a configuration (SYS_EN) which allows the default 10Mbits/s transmit rate and the variable transmission rate to be generated by an internal clock enable generator. This has the advantage of the transmit clock being the same as the main system clock so a dedicated global net for the transmit clock is not required.

In order to minimise the number of global clock nets the SpaceWire CODECs have therefore been configured to SYS_EN within the BepiColombo RIU. The design meets the required timings at MIL spec operating conditions and with 50kRAD total dose. Figure 5 shows a screenshot of the Microsemi Libero Designer Smart-Time software showing the performance achieved on the internal clock nets.

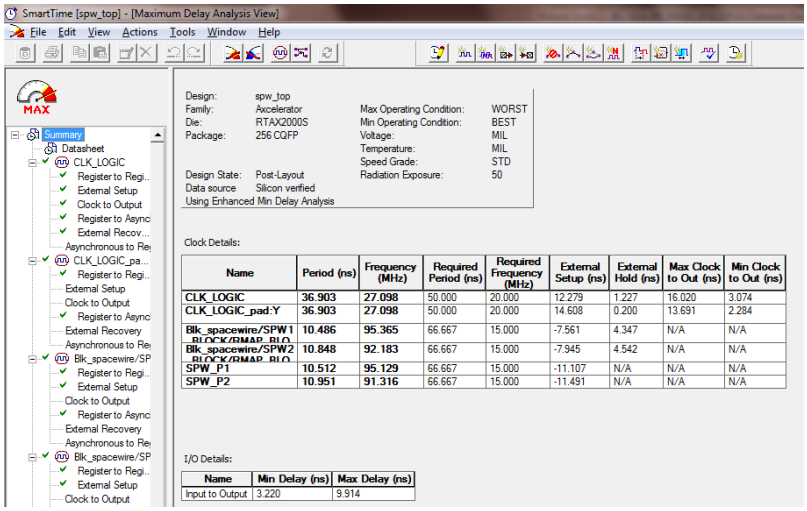


Figure 5: Libero Designer SmartTime screenshot.

The speed of the transmit clock within this configuration is limited by the inherent speed of the FPGA routed clock network. A speed of 27MHz could be achieved with the current FPGA design. The maximum SpaceWire rate required for the BepiColombo RIU is 10Mbps, therefore this performance is acceptable and no further optimisation has been carried out.

Performance could be improved in a number of ways if required:

- Investigate the worst case delays limiting the clock speed (which may/may not be related to the SpaceWire CODEC) and then optimise the code.

- Procure a higher speed grade FPGA. The BepiColombo RIU utilises –STD speed grade devices, however -1 and -2 speed grade devices are available which have guaranteed higher internal operating performance.

- Move the SpaceWire CODEC into a device with a radiation hard embedded phase locked loop (PLL). This would almost certainly be an ASIC or a SpaceWire Router / SpaceWire receiver IC. This has the advantage of allowing a much higher transmit speed unconstrained by maximum clock speed of the RTAX routed clock network.

6 SCHEDULE

The FPGA SpaceWire interface has been fully tested using Microsemi Pro-ASIC3E devices. The next stage of the development will be to migrate the design into Commercial Axcelerator AX2000 devices following the Microsemi prototype development approach outlined in [RD4]. The design will then be programmed into RTAX B-grade devices for EQM qualification testing in Q4 2011.

7 REFERENCES

1. ECSS-E-ST-50-12C, (SpaceWire – Links, nodes, routers and networks, Issue 2, 31st July 2008.

2. SpaceWire Codec VHDL User Manual, University of Dundee, 2005.

3. Microsemi AC305 Application Note Implementation of the SpaceWire Clock Recovery Logic in Actel RTAX-S Devices.

4. Microsemi AC170 Application Note Prototyping RTAX-S Using Axcelerator Devices.

DEVELOPMENT OF SPACEWIRE HIGHER LAYER PROTOCOLS BASED ON THE CCSDS SOIS ARCHITECTURE

Session: Standardisation (Poster)

Short Paper

Takahiro Yamada

JAXA/ISAS, 3-1-1 Yoshinodai, Sagamihara, 229-8510, JAPAN

E-mail: tyamada@pub.isas.jaxa.jp

ABSTRACT

The Consultative Committee for Space Data Systems (CCSDS) published a document called the Spacecraft Onboard Interface Services (SOIS). This document specifies a set of standard services and traffic classes to be provided by onboard networks for onboard applications, but not all of the services and traffic classes can be realized with the existing standards on SpaceWire. This paper proposes developing some protocols so that SpaceWire can provide all the subnetwork services and traffic classes defined by SOIS. Specifically, this paper proposes developing two protocols to support the traffic classes. One of them provides the functionality of managing the traffic on the network, and the other provides the functionality of performing retransmissions to ensure reliable delivery.

1 INTRODUCTION

The Consultative Committee for Space Data Systems (CCSDS) published a document called the Spacecraft Onboard Interface Services (SOIS) in 2007 [1]. This document proposes a layered architecture of onboard communications services that should be provided by onboard networks of spacecraft for onboard applications. According to this architecture, SpaceWire is a type of Data Link. The Data Link should be accessed by applications or other upper-layer standard services with a set of standard services, which are called the SOIS subnetwork services. The SOIS document also defines four traffic classes, each of which corresponds to a set of quality of service (QoS) levels.

There are already some standards related to SpaceWire (for example, [2] and [3]), but not all of the subnetwork services and traffic classes defined by SOIS can be realized with the existing standards. Some more protocols that run on top of the existing SpaceWire protocols are required. This paper proposes developing some more protocols so that SpaceWire can provide all the subnetwork services and traffic classes defined by SOIS. Specifically, this paper proposes developing two protocols to support the traffic classes. One of them provides the functionality of managing the traffic on the network to ensure guaranteed bandwidth and timely delivery. The other provides the functionality of performing retransmissions to ensure reliable delivery without loss, without duplication, and in-sequence. Two of the SOIS subnetwork services with all the traffic classes can be provided by combining the SpaceWire specification [2], the Remote Memory Access Protocol (RMAP) [3], and the two

protocols mentioned above. The three other services can be provided by adding simple protocols to these protocols.

2 SOIS SUBNETWORK SERVICES AND TRAFFIC CLASSES

This section briefly introduces the subnetwork services and traffic classes defined by SOIS.

2.1 SERVICES

SOIS [1] defines the following five subnetwork services.

1) Packet Service – supports the transfer of packets over a subnetwork (that is, a SpaceWire network).

2) Memory Access Service – provides the capability to read or write data from or to a memory location in a device.

3) Time Distribution Service – provides the capability to distribute a centrally maintained reference time to multiple users throughout the spacecraft.

4) Device Discovery Service – provides the capability to detect devices becoming active following a change in the hardware configuration of the spacecraft.

5) Test Service – used for checking data system functionality and connectivity.

2.2 TRAFFIC CLASSES

SOIS [1] defines the following four traffic classes. For each of the subnetwork services shown above, these traffic classes can be used.

1) Best Effort – provides for non-reserved (that is, no network bandwidth are reserved), try once communication.

2) Assured – provides for non-reserved communication with retries.

3) Reserved – provides for best-effort communication over a resource reserved logical link.

4) Guaranteed – provides for resource reserved communications with retries.

3 PROPOSED PROTOCOLS

In this section, two protocols that augment the capabilities of SpaceWire networks are presented.

3.1 SPACEWIRE-SCHEDULING

This protocol provides the functionality of managing the traffic on the network to reserve bandwidths for data flows that require them and guarantee timely delivery of data to the receiving users. To reserve bandwidths, this protocol uses time slots, which are defined using the Time-Codes defined in SpaceWire [2]. Time Slots are allocated to instances of subnetwork services with Reserved and Guaranteed traffic classes.

This protocol does not have any more functionality than defining, allocating, and ensuring the use of time slots. This protocol is used as a building block combined with other protocols to support traffic classes that require reserved bandwidth and timely delivery.

3.2 SPACEWIRE-RELIABLE

This protocol provides the functionality of performing retransmissions to ensure reliable delivery without loss, without duplication, and in-sequence. It also provides the capabilities for (1) segmenting data units provided by the users if they are longer than the size allowed by the underlining network, (2) adjusting the rate of data transmitted from the sender based on the reception capability at the receiver (flow control), and (3) managing redundant routes (that is, if a route has been found not functioning, the protocol automatically switches to an alternative route). This protocol is used as a building block combined with other protocols to support traffic classes that require reliability.

4 HOW SOIS SUBNETWORK SERVICES AND TRAFFIC CLASSES ARE PROVIDED

This section shows how the SOIS subnetwork services and traffic classes are provided using the two protocols presented in the previous section.

Figure 4-1 shows how the SOIS traffic classes are provided using SpaceWire-Scheduling and SpaceWire-Reliable. If traffic classes Best Effort and Assured are to be used together with traffic classes Reserved and Guaranteed on the same SpaceWire network, SpaceWire Scheduling will be used to support all these traffic classes, allocating sufficient bandwidths to the Reserved and Guaranteed traffic classes.

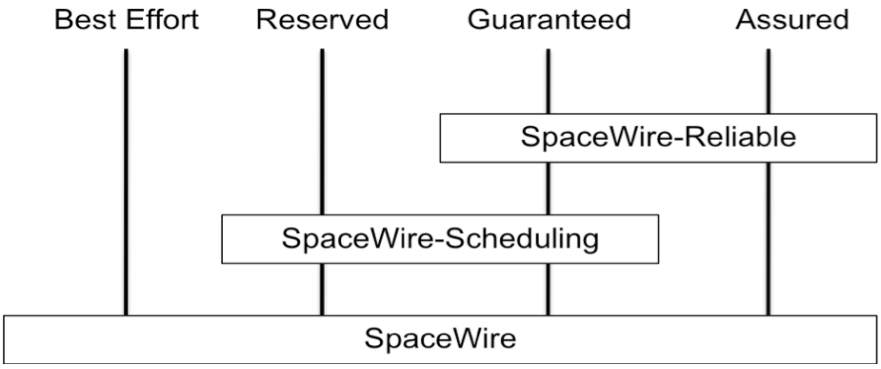


Figure 4-1: How the SOIS Traffic Classes are Provided

Figure 4-2 shows how the SOIS subnetwork services are provided using SpaceWire-Scheduling and SpaceWire-Reliable. The Packet Service is provided by using the combination of protocols shown in Figure 4-1. The Memory Access Service is provided by using RMAP together with the combination of protocols shown in Figure 4-1. To provide each of the Time Distribution, Device Discovery and Test Services, a simple protocol for defining messages and message sequences is needed together with the combination of protocols shown in Figure 4-1 (possibly with RMAP for some Services).

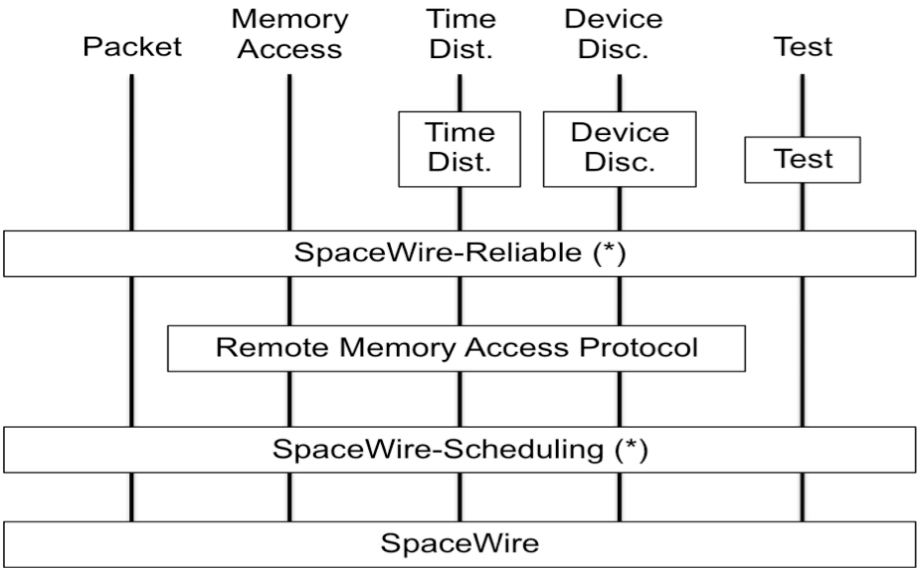


Figure 4-2: How the SOIS Subnetwork Services are Provided
(*) Whether to use SpaceWire-Scheduling and SpaceWire-Reliable depends on the traffic class selected.

5 CONCLUSION

This paper presented some protocols that run on top of SpaceWire and are used to provide the SOIS subnetwork services and traffic classes. This paper only presented rough concepts of these protocols, and further investigation is needed to determine the full specifications of these protocols. Furthermore, a roadmap showing how SpaceWire should evolve to accommodate the requirements of future spacecraft should be created and shared by both developers and users of SpaceWire. The diagrams presented in this paper are examples of such a roadmap.

6 REFERENCES

1. CCSDS, “Spacecraft Onboard Interface Services”, CCSDS-850.0-G-1, June 2007.

2. ECSS, “SpaceWire – Links, nodes, routers and networks”, ECSS-E-ST-50-12C, July 2008.

3. ECSS, “SpaceWire – Remote memory access protocol”, ECSS-E-ST-50-52C, February 2010.

Test and Verification 1

USING TVS TO VERIFY SPACEWIRE DESIGNS

Session: Test and Verification

Long Paper

Damaris L. Guevara, Omar A. Haddad

NASA/Goddard Space Flight Center, 8800 Greenbelt Road, Greenbelt MD, 20771

E-mail: Damaris.L.Guevara@nasa.gov , Omar.A.Haddad@nasa.gov

ABSTRACT

Spaceflight designs often contain a mix of standard and custom interfaces. Although test bench equipment for standard interfaces such as SpaceWire is often available for testing flight designs in the lab, testing custom interfaces often presents a challenge. The Total Verification System, TVS, addresses this issue and more.

The TVS provides the user with the ability to modify its functionality at a low level to allow for verification of non-standard features of the device under test. Furthermore, interactions between SpaceWire interfaces and other custom interfaces can be properly verified in the TVS because of the user-programmability.

1 COTS SPACEWIRE GSE

Commercial off-the-shelf (COTS) SpaceWire ground support equipment (GSE) is readily available. So, the question is: why do they not suffice? Most COTS GSE is missing some desirable qualities for design verification. Two such qualities are a lack in simulation models and the ease of customization. Let's take a closer look at both of these and why they are important for creating a more efficient and cost effective test and verification process.

1.1 LACKING IN SIMULATION MODELS

Why do we even need GSE simulation models? Typically, GSE has been designed for lab use with little or no thought of its use in a simulation environment. Doing this has kept us blind from the possibility of having a more efficient and cost effective test and verification process.

FPGA capacities are continuing to grow. As they grow, board designs are becoming more FPGA-centric. With FPGA-centric designs come FPGA-centric systems. If FPGAs are tested and verified in a simulation environment then it also makes sense to test and verify FPGA-centric board designs and FPGA-centric system designs in a simulation environment as well.

If we want to advance our test and verification process to one with higher efficiency and cost effectiveness, then we need to shift our simulation applicability from FPGA level to board level. However, running board level simulations require driving

SpaceWire interfaces with *something*. That *something* is a simulation model of the GSE.

Once we have a simulation model of the GSE, we are then able to maximize portability to the lab. This is because the same set of tests used in simulation can now be the same set of tests used in the lab. So, where there used to be two totally different set of tests written by two separate verification engineers, there is now only one set of tests that works in two environments written by one verification engineer. This allows for more design issues to be caught in the simulation environment where it's less costly to fix than in the lab environment where changes are more costly to fix.

It makes for smarter practice to be able to see how the system as a whole behaves before hardware is even built. This is done through simulations and saves you schedule, money, and even manpower. However, in order to simulate the entire system one needs...you guessed it, GSE simulation models.

1.2 NOT EASILY CUSTOMIZABLE

Why do we need to be able to customize our GSE? GSE can be quite costly and with that comes the need for their reuse in order to make the purchase worthwhile. When it comes to space flight designs, it seems that no two designs are ever alike. They may have very close similarities but they are almost always slight if not major changes to design requirements from project to project. This brings us the need to be able to customize our GSE in order to meet the ever-changing needs of our ever-changing designs. This brings on the ability to reuse our GSE as we move from one project to the next.

Most GSEs are also interface specific and if an interface changes, this means the purchasing of a different type of GSE. If the GSE is customizable, this would save not only money, but space (be it rack or bench space).

In some cases, a design requirement which requires a modification to the SpaceWire standard can't be tested at board level if the GSE is not customizable. This means that the requirement can't be tested until a later phase of integration where fixing possible issues becomes more expensive. An example of customizing GSE is the GSFC implementation of SpaceWire time codes and exercising a built-in Bit Error Rate (BER) feature.

2 TVS SPACEWIRE GSE

The TVS is a custom-designed, fully FPGA-reprogrammable piece of GSE that is optimized for verifying digital designs at the board level. It can implement up to 6 SpaceWire ports as well as other custom interfaces that use RS422, TTL, LVTTL, and I2C. The PLL provides 3 clock sources up to 200MHz each. The TVS can be used with software applications ranging from directed tests to Labview-like GUI programs. The functionality of TVS is modeled with high-fidelity simulation models so that it can be used to fully verify the spaceflight design before hardware is even built. The simulation test bench is easily ported over to the lab environment allowing reduced schedule times and cost savings in manpower.

The TVS GSE is also scalable which allows the use of multiple units as needed to cover all DUT interfaces. Its scalability is due to the fact that it connects to a host PC over USB, allowing more than one TVS to be connected to the host PC and controlled by the host PC.

3 TVS FPGA

The TVS FPGA is a USB-reprogrammable Xilinx Spartan 3. The gate count, 1500k or 4000k, depends on which Opal Kelly board the TVS is configured with. The XEM3010 houses the Spartan 3-1500 while the XEM3050 houses the Spartan 3 - 4000. There is also the option of upgrading to a Xilinx Spartan-6 (XC6SLX45) by configuring the TVS with the XEM6010 board. If you wanted to upgrade even more, there are two other XEM6010 boards LX45 and LX150 that are available for the TVS that also use the USB bus. The XEM6110 uses PCIe which is faster, but requires the host PC to have a PCI board installed that supports PCIe.

No matter which board the TVS is configured with, the FPGA can be loaded with GSFC-developed verification IP Cores such as SpaceWire Nodes and can be used to create complex data patterns in real time. The programmability of the TVS FPGA allows users to replace costly GSE equipment and provide the user with full control over its behavior allowing unique mission requirements to be tested and verified, thus reducing costs.

4 TVS USAGE

The TVS has many uses in test and verification from the simplest of designs to the more complex. The TVS is applicable to any industry where electronic designs are developed such as space, medical, automotive, or consumer. In any application, its function is to test and verify designs and it can do so in both the simulation environment and lab environment.

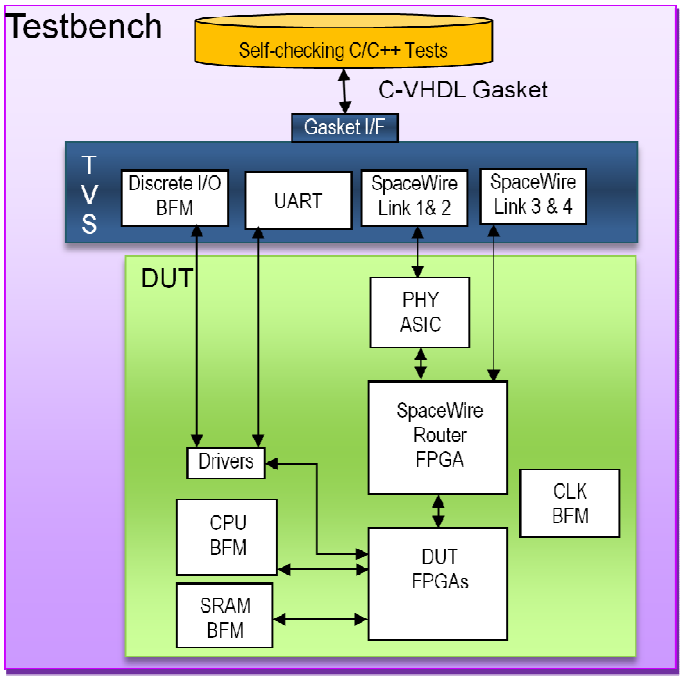
Although the TVS is mainly designed for use with digital designs, it also has the capability to support analog designs with the addition of some external hardware. For example, an ADC or DAC that supports I2C can be connected to the TVS's I2C signals, allowing the TVS to drive and sample analog signals.

4.1 TVS IN SIMULATION

The TVS can be used in the simulation phase, prior to building hardware. Typically, the simulation phase consists of a suite of self-checking and automated tests written in C/C++, the TVS simulation model, the device under test (DUT) with its component simulation models and FPGA RTL, and the C-VHDL gasket interface which bridges the tests to the simulator.

The tests communicate with a simulation model of the TVS and the verification IP cores in the TVS FPGA via the C-VHDL gasket interface to exercise specific functions of the DUT. The TVS thus promotes a board level test bench which allows netlist problems to be caught in simulation where it is easier and cheaper to address.

The diagram below shows the TVS in a typical simulation environment. The orange block represents the test suite. The TVS simulation model is shown in blue. It uses a gasket interface which is provided as part of the C-VHDL gasket. The DUT contains the FPGA RTL code and component simulation models.

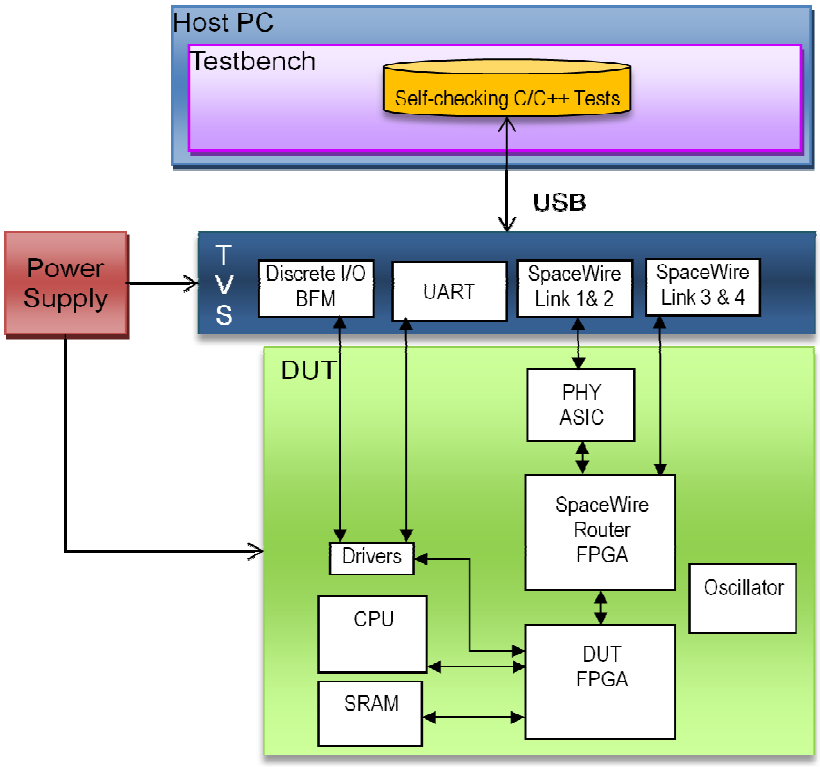


4.2 TVS IN THE LAB

Once we have the design fully simulated with all FPGA RTL code coverage having reached the desired goal, we can confidently move on to DUT fabrication. When the DUT is fabricated, the same suite of simulation tests is recompiled for use with a real TVS to exercise the real DUT. The advantage of using the same suite of test for the lab as we used in simulation is that if a bug arises during lab testing, it can usually be recreated in simulation and fixed more easily with full visibility into the entire design. Please note that although the advantages of running board level simulations are great, some bugs require real time operation to present themselves.

The diagram below shows the TVS in a typical lab environment. Notice the high degree of similarity to the simulation environment. The main difference is that the GSE and DUT models are replaced with real hardware. The tests are the same as in the simulation environment, and only talk to the GSE. The abstraction layer is also recompiled to talk to APIs instead of the C-VHDL gasket and the TVS verification IP cores are targeted to the TVS's Xilinx Spartan 3 FPGA.

The fun doesn't have to end at board level testing. The TVS can even be used in the box level GSE rack to exercise a box's front panel signals during environmental testing. This allows for more of the SpaceWire network to be exercised.



5 TVS COMMERCIALIZATION

As a tax-funded agency, NASA gives back to the US economy through its technology transfer program. This section discusses the possibility of commercializing the TVS concept.

5.1 POTENTIAL MARKET

The TVS has gone from an idea to concept to realization and has been used successfully on multiple NASA missions. As mentioned earlier, the fact that it is so customizable allows for the TVS to be applicable for not only Space applications but for medical, automotive, or consumer industries where electronics designs is developed. The TVS truly is versatile.

5.1.1 Product Offerings

The TVS hardware can be sold as a stand-alone GSE that comes with a high-fidelity simulation and a test bench environment that provides the portability from simulation to lab testing. TVS FPGA verification IP cores can be developed and sold as add-on modules to the TVS unit. The verification IP cores cater to various industries and come with software drivers (DLLs) and User's Manual documentation. With certain TVS verification IP cores, the TVS could potentially replace far costlier equipment. It is usually possible to replace multiple COTS GSE units with a single TVS unit.

5.1.2 Training Classes

The TVS concept is suggested for advanced designers due to the high level of sophistication and customization, so training will be necessary. However, there are quite a few training possibilities for learning that will cater to anyone. Not everyone learns by the same method as others and it is important to offer diverse training opportunities such as tutorials, webinars, seminars, on-site training, or a one--on--one consultation.

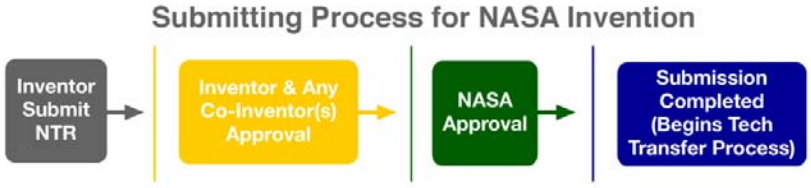
5.1.3 Usage For Design Verification Services

Companies that offer design verification services can benefit from making the TVS a part of their process. Efficient users of the TVS can offer their clients reduced costs, shorter schedules, and more thorough design verification, an advantage that benefits the company and makes it more competitive.

5.2 NTR SUBMITTED

A new technology report (NTR) has been submitted for the TVS. *New Technologies are defined as any invention, discovery, improvement, or innovation whether or not patentable, either conceived or first actually reduced to practice in performance of NASA work. This includes, but is not limited to, new processes, machines, manufactures, and compositions of matter, and improvements to, or new applications of, existing processes, machines, manufactures, and compositions of matter. New Technologies also include new computer programs, and improvements to, or new applications of, existing computer programs.* (1)

The submitting process for NASA inventions is shown in the diagram below.



5.3 HW/SW AVAILABILITY

The TVS GSE hardware (HW) is not yet an off-the-shelf product, however NASA can license out the design files necessary for fabrication. The manufacturing expenses are estimated at $1k (using the XEM3010). A reasonable retail price of $5k would yield an 80% profit.

The TVS software (SW) can be made available. The test bench software should come along with the hardware and the FPGA IP cores (VHDL and DLLs) can be individually licensed for use.

For more information on the TVS HW, SW, or both please contact Omar Haddad at Omar.A.Haddad@nasa.gov.

1. Orans R. (2009, April 1.) *Technology Reporting*. Retrieved August 19, 2011, from NASA Technology Transfer System website: https://ntr.ndc.nasa.gov/

SPACEWIRE EGSE

Session: SpaceWire Test and Verification

Long Paper

Stephen Mudie, Paul E. McKechnie

STAR-Dundee, c/o University of Dundee, School of Computing, Dundee, DD1 4HN

Steve Parkes, Martin Dunstan

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

E-mail: stephen.mudie@star-dundee.com, paul.mckechnie@star-dundee.com, sparkes@computing.dundee.ac.uk, mdunstan@computing.dundee.ac.uk

ABSTRACT

The SpaceWire Electronic Ground Support Equipment (EGSE) is a STAR-Dundee product [1] designed to simulate and stimulate SpaceWire devices. It provides a means of generating user defined packets in pre-defined sequences at specific times and data rates. The SpaceWire EGSE is configured using a script that is compiled and loaded onto the SpaceWire EGSE unit. Once configured, the EGSE can generate complex SpaceWire packet sequences without further interaction from host PC software.

Real-time SpaceWire Electronic Ground Support Equipment can be implemented easily with the SpaceWire-EGSE unit, avoiding the need for complex and expensive, real-time software development.

1 INTRODUCTION

SpaceWire Electronic Ground Support Equipment (EGSE) is needed to support the integration and testing of spacecraft that use SpaceWire. The EGSE has to simulate instruments and other equipment during integration and test, and has to do this with similar if not identical timing. Furthermore the SpaceWire EGSE has to integrate with other test equipment, either responding to events or triggering other pieces of equipment. Typically SpaceWire EGSE is implemented using a SpaceWire interface board in a rack with a host computer which controls the SpaceWire interface, often at the same time as controlling other EGSE interfaces. To provide representative timing of SpaceWire packets, the software has to operate in real time, which is both costly and difficult to develop. Last minute changes are very difficult to implement, especially when the software is controlling multiple interfaces.

What is needed is a unit that will allow arbitrary SpaceWire packets to be transmitted, in a predefined sequence, at a specified user data rate. It should initiate the sending of a packet sequence on command from the host software, when a particular SpaceWire packet is received, or when an external trigger is asserted. It should do this without the need for any real time software development.

The new STAR-Dundee SpaceWire-EGSE unit is just such a unit. It is provided with a special scripting language which allows SpaceWire packets to be defined using easy to understand terms. This language also specifies the time sequencing of packets and the event or series of events that cause various packet sequences to be sent. The information thus provided is compiled and loaded on to the SpaceWire-EGSE hardware. Thereafter the only interaction with user real-time software controlling the SpaceWire-EGSE and other equipment is through software events that can be asserted by the user application or indicated by the SpaceWire-EGSE.

2 HARDWARE

The SpaceWire EGSE is configured via a USB connection to the host PC. It has two SpaceWire ports from which packets can be generated and received. It has four external triggers, three input and one output. It also has a large memory for storing packet definitions.

3 SPACEWIRE EGSE SCRIPTING LANGUAGE

The SpaceWire EGSE is configured using a simple yet powerful scripting language. The language can be used to define variables, events, packets, packet generation schedules and state machines.

3.1 PACKET DEFINITION

Packet definitions can consist of data defined in hexadecimal or decimal bytes, variable references, CRC and checksum calculations and EEP and EOP control characters.

Example	Description

`packet myPkt`
`  hex(0A 0B 0C 0D)`
`  eop`
`end packet`
	Defines a packet named “myPkt” consisting of data specified in hexadecimal bytes (0A 0B 0C 0D) followed by an end of packet marker.

Above is a very basic packet definition. As with much of the SpaceWire EGSE language, packets are defined using a header, body and footer. The header consists of the “packet” keyword followed by the packet name and indicates the start of a packet definition. The packet body defines the packet contents. The packet footer consists of the keywords “end packet” and indicates the packet definition end.

Example	Description
packet myPkt start(crc8) hex(0A 0B 0C 0D) dec(01 02 03 04) * 2 stop(crc8) crc8 eop end packet	Defines a similar packet to the previous example but contains additional data specified in decimal. It also contains a CRC calculation and reference.

The start and stop statements in the example above can be used to calculate CRC and checksum values for the data between them. The CRC or checksum value can then be referenced in the packet definition.

3.2 VARIABLES

The SpaceWire EGSE provides variables that are used to define packets with dynamic data. Variables have names by which they can be referenced in packet definitions along with a type and (optionally) an initial value. Each variable performs a function upon its value when read, based upon its type. The variable types available are increment (increments variable value by one when read), decrement (decrements variable value by one when read), rotate left (performs rotate left bit shift to variable value when read), rotate right (performs rotate right bit shift to variable when read) and random (assigns a random value to the variable). The example below demonstrates the use of a variable to dynamically set the ID of each packet sent.

Example	Description
variables transactionID inc8 = 0 end variables packet myPkt hex(0A 0B 0C 0D) transactionID eop end packet	Defines an increment variable named “transactionID” with an initial value of 0. A packet definition containing a reference to the incrementing variable “transactionID”.

3.3 PACKET GENERATION SCHEDULES

A schedule is used to define a timed sequence of packets for packet generation. The schedule references packets defined earlier in the script. Packets can be sent relative to the start of the schedule or relative to the previous packet. We can also specify the number of times the packet is sent.

Example	Description
schedule mySchedule1 5ms send myPkt1 * 2 10ms send myPkt2 end schedule schedule mySchedule2 5ms send myPkt1 +10ms send myPkt2 end schedule	Schedule named “mySchedule1” sends packet “myPkt1” twice, 5ms after the schedule starts, and “myPkt2” once 10ms after the schedule starts. Schedule “mySchedule2” sends “myPkt1” 5ms after the schedule starts then “myPkt2” 10ms after “myPkt1” is sent.

3.4 STATE MACHINE

The state machine definition is responsible for control of the EGSE state. The state machine consists of state definitions. Each state has an associated schedule which is run when the state is entered at the data rate specified. Along with a schedule each state contains statements that determine when to change state and when to transition from one state to another.

```
statemachine 1
      initial state state1
            do mySchedule1 @ 20Mbps
            transition at end of schedule
            on myTrigIn1 goto state2
      end state
      state state2
            do mySchedule2 @ 50Mbps repeatedly
            transition at end of packet
            on myTimer goto state1
      end state
end statemachine
```

The above example is a state machine definition for SpaceWire link 1 of the SpaceWire EGSE. Two states are defined named “state1” and “state2”. On entering “state1” the schedule named “mySchedule1” is run once at a data rate of 20Mbps. If the event named “myTrigIn1” is received then at the end of the current schedule the state will change to “state2”. On entering “state2” the schedule named “mySchedule2” is run repeatedly at a data rate of 50Mbps. If the event named “myTimer” is received then once the current packet is sent the state will change to “state1”.

3.5 EVENTS

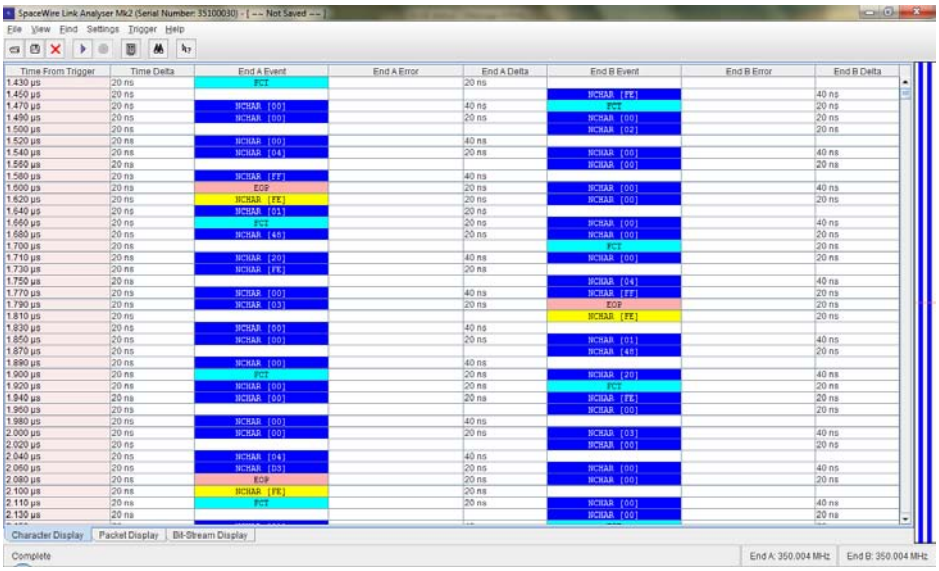
As seen in the state machine definition above, events are used to control the current state and therefore the current packet generation schedule. There are pre-defined events and user defined events. Predefined events include link started, link errors (parity, escape, credit, and disconnect), time-code received, and packet generation events. User defined events are timers, counters, software and external triggers.

Example	Description
timers myTimer 10ms start on mySWEvent1 end timers counters myCounter 10 on myTrigIn1 end counters software mySWEvent1 1 end software triggers myTrigIn1 input 1 rising output high on myTimer end triggers	A 10ms timer that begins when the event “mySWEvent1” is received. A counter that generates an event when the event “myTrigIn1” is received 10 times. Declaration of a software event. Generates “myTrigIn1” event when rising signal received on external input trigger pin 1. Generates a high external output trigger signal when “myTimer” event is received.

Timers generate an event when a specified time is reached. The timer starts when the associated event is received. A counter has an initial value that is decremented each time an associated event is received. When the counter reaches zero it generates an event.

External trigger-in events specify the event to generate when a trigger-in signal is received on the associated input pin. The external output trigger generates a signal when it receives a specific event.

Software events permit host PC software to trigger a change in the state machine in the SpaceWire-EGSE. They provide a means of interaction with the host PC software. An API makes available functions the user can call upon to generate a software event. The host software can also be signalled when a specific event occurs or when a particular state is entered in the EGSE state machine.



The screenshot above is taken from a SpaceWire Link Analyser and shows the EGSE generating a sequence of small packets on both SpaceWire links, at the maximum rate possible on the link. Note that the link is running at 350MHz and no NULL characters are seen in the trace.

4 CONCLUSION

This paper has briefly described the SpaceWire EGSE and the scripting language used to configure it. The scripting language can quickly be used to configure the SpaceWire EGSE unit to mimic the behaviour of the SpaceWire device of interest. The SpaceWire EGSEs ability to generate packets independent of host software means it can produce very similar if not identical packet generation behaviour to the simulated instrument. The external input and output triggers on the SpaceWire EGSE provide a means by which to integrate with other test equipment. Such capabilities make the SpaceWire EGSE a quick and efficient way of simulating SpaceWire devices. Using the SpaceWire EGSE it is possible to develop a complete SpaceWire instrument or other device simulation with real-time behaviour, in little more than one day.

5 REFERENCES

1. STAR-Dundee, http://star-dundee.com/products.php, STAR-Dundee SpaceWire Products, STAR-Dundee Website.

TESTING SPACEWIRE SYSTEMS ACROSS THE FULL RANGE OF PROTOCOL LEVELS WITH THE SPACEWIRE PHYSICAL LAYER TESTER.

Session: SpaceWire test and Verification

Long Paper

Pete Scott

STAR-Dundee c/o University of Dundee, School of Computing, Dundee DD1 4HN Scotland, UK

Paul Crawford, Steve Parkes

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

Jorgen Ilstad

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

E-mail: pete@star-dundee.com, psc@sat.dundee.ac.uk, sparkes@computing.dundee.ac.uk, jorgen.ilstad@esa.int

ABSTRACT

STAR-Dundee have previously reported on test equipment which is capable of testing the Network, Packet, Exchange, Character and parts of the Signal level of the SpaceWire standard. This paper introduces the SpaceWire Physical Layer Tester (SPLT), which is a new tool designed to test, validate and verify a SpaceWire system across all levels covered by the SpaceWire standard.

Two SpaceWire ports on the SPLT employ a special LVDS interface which allows the transmitted signals to be deliberately and measurably manipulated to test the capability of a unit under test (UUT) to receive signals of varying quality. The SPLT SpaceWire drivers offer full, independent control of voltage offset and amplitude for data and strobe pairs. Skew can be introduced both in-pair and between data-strobe pairs. Slew rates can be individually configured on each half of the differential pairs. To facilitate acquisition of an eye-diagram, the signal received on the termination resistors on these ports is buffered on external connectors to allow easy interfacing to an oscilloscope. This allows a comprehensive suite of tests to be performed through the UUT SpaceWire port without the need to open the unit.

In addition to the LVDS interface, the SPLT also implements many of the capabilities of existing STAR-Dundee devices: Link Analyser Mk2, Conformance Tester and USB Brick in addition to the packet generator and checker capabilities of the newly announced SpaceWire EGSE. A pair of Gigabit Ethernet ports and a USB 2.0 port in addition to an API allow for great flexibility in interfacing the SPLT to existing test environments. The device is rack mountable in a 1U, half width format.

1 EXISTING TECHNIQUES FOR TESTING THE PHYSICAL AND SIGNAL LAYER

1.1 INTRODUCTION

The SpaceWire Standard is defined across six levels ranging from the Physical level up to the Network level. A successful SpaceWire system must be implemented in a way which conforms to the specifications laid out across all of these levels. A range of existing techniques can be implemented to analyse the performance of such a system at the physical and signal layer. A more detailed overview of the devices and test techniques discussed in this section can be found in [1].

1.2 TIME DOMAIN REFLECTOMETRY

Time Domain Reflectometry is used to evaluate the performance of connectors, differential pair traces on Printed Circuit Boards (PCB), backplanes, cables and any other media which is used to transmit SpaceWire LVDS electrical signals. A pulse is transmitted down a SpaceWire differential pair and the analysis of the reflections can show up impedance discontinuities or incorrect termination of the line. An appropriate interface to the SpaceWire port is required to inject the test signals and measure the responses.

1.3 EYE PATTERN MEASUREMENT & SPECTRUM ANALYSIS

Measuring the eye-pattern of a SpaceWire signal using a high speed oscilloscope gives insight to the transmitter characteristics of the unit under test. The signal's rise and fall times and jitter can be observed. A mask can be applied to the opening of the eye for a sustained test period to verify that the signal remains in a valid region. Eye pattern measurement can also be employed to measure Data-Strobe skew characteristics by measuring the Data eye pattern when triggering on the Strobe signal and vice-versa. A Spectrum Analyser can be used to display the power spectrum of a SpaceWire signal being transmitted from a UUT and analyse parameters such as drift and jitter.

One of the biggest difficulties in performing Spectrum Analysis or Eye Pattern measurements on a SpaceWire system is that it requires probes to be fixed to the LVDS termination resistors in the receiver. It is often undesirable or not permitted to remove the case of flight equipment and attach probes onto the relevant components. A further difficulty is that these tests work best when pseudo-random bit streams (PRBS) are being analysed. It may be difficult to configure a unit under test to generate such a condition.

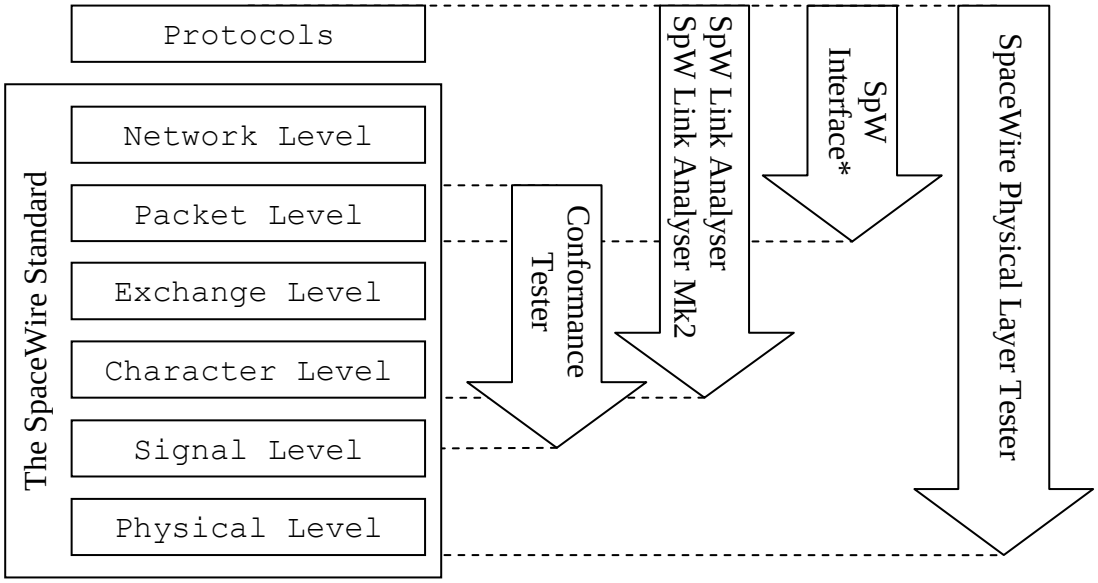
1.4 BIT ERROR RATE TESTING (BERT)

BERT systems will typically use a transmitter to generate a PRBS or pre-programmed bit sequence for transmission to the UUT. A receiver in the BERT system will compare the received signal to an expected signal and flag any bit-errors that are detected. One difficulty with these systems is that an interface from the transmitter/receiver to a SpaceWire port would have to be implemented. It also assumes that the unit under test can conveniently output PRBS test data through a loopback mode.

2 THE STAR-DUNDEE SPACEWIRE PHYSICAL LAYER TESTER

2.1 TESTING ACROSS THE SPACEWIRE STANDARD

STAR-Dundee currently supply a range of test equipment which tests across most, but not all levels of the standard. This is summarised in Figure 1.



*Includes EGSE, SpW Brick, SpW PCI Mk2, SpW cPCI Mk2, SpW PMC Mk2, SpW PCI Express and SpW Router Mk2.

Figure 1: Testing, Monitoring and Verifying SpaceWire systems across the SpaceWire standard with STAR-Dundee test equipment.

2.2 OVERVIEW OF THE SPACEWIRE PHYSICAL LAYER TESTER

The unique feature of the STAR-Dundee SPLT is the capability of manipulating the analogue characteristics of the output LVDS signals. This facilitates the analysis of what the UUT is capable of successfully receiving. The severity of skew, slew, swing, common-mode and jitter can be controlled to mimic a range of physical media, environments and device output characteristics. The operation and capabilities of these special LVDS drivers is explained in Section 4. The SPLT also features the capability to connect a high speed oscilloscope via SMA connectors on the front panel in order to observe the received SpaceWire signals. High speed analogue buffers are utilised to buffer the signal close to the termination resistors allowing the SPLT to receive and decode the SpaceWire signals whilst they may be simultaneously monitored by an oscilloscope.

The SPLT is a 1U rack mountable ½ width device which interfaces to a host PC through either USB 2.0 or one of its 2 Gigabit Ethernet ports. A Mictor connector allows a logic analyser to interface to the device's inbuilt STAR-Dundee Link Analyser Mk2 [2]. An enhanced version of the STAR-Dundee Conformance Tester [3] allows advanced conformance testing of the physical layer to be performed.

3 PERFORMING TESTS WITH THE SPACEWIRE PHYSICAL LAYER TESTER

3.1 INTRODUCTION

A range of tests can be performed with the SPLT operating in different modes.

3.2 IN-LINE MARGIN ANALYSIS

The SPLT is connected between two SpaceWire UUTs in the same way a STAR-Dundee Link Analyser would be connected. This configuration is shown in Figure 2.

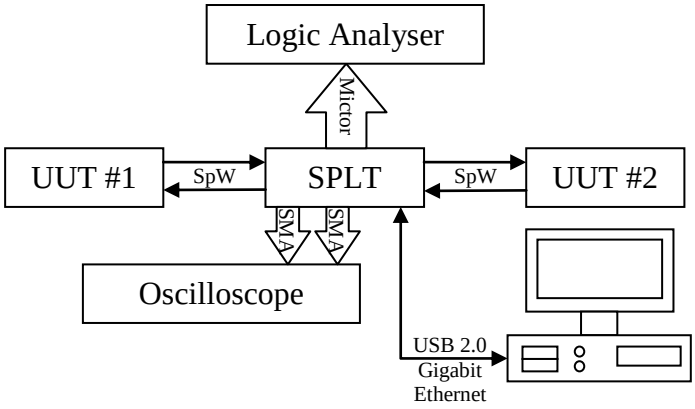


Figure 2: Using the SPLT to perform in-line margin analysis between two devices under test.

When operating in in-line analysis mode, the two UUTs will communicate as normal, sending and receiving data to each other. The SPLT buffers the incoming signals on one SpaceWire port and then drives them out through the analogue LVDS drivers on the other SpaceWire port. The SPLT can then manipulate the SpaceWire signals in one, or both, directions to explore the receive margins of either, or both, UUT devices.

3.3 LOOP-BACK MARGIN ANALYSIS

In loop-back configuration, the SPLT is connected to a single UUT as shown in Figure 3.

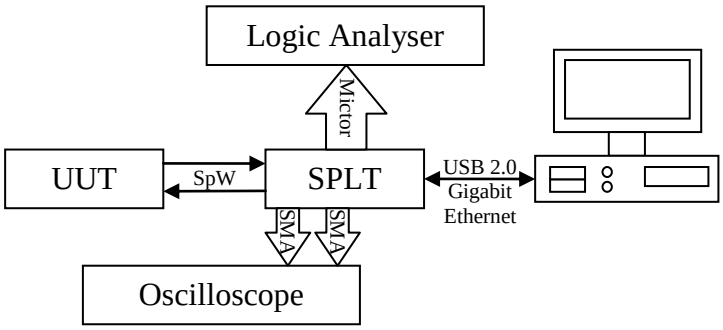


Figure 3: Using the SPLT to perform loop-back margin analysis, conformance testing, pattern generation & checking on a single UUT. This setup can also be implemented to use the SPLT as a SpaceWire interface to a Host PC.

In loop-back analysis, the SPLT receives data from the UUT and loops the data back through the same SpaceWire port. The LVDS transmitters can manipulate the data to test the receive margins of the UUT. The signals received from the UUT are buffered by the SPLT and made available for analysis on an oscilloscope. Loop back SpaceWire data is decoded onto the Mictor connector for easy interface to a Logic Analyser. Loop-back analysis requires the UUT to be able to start a SpaceWire link and to send and receive SpaceWire commands.

3.4 CONFORMANCE TESTING

The SPLT implements an advanced version of the STAR-Dundee Conformance Tester with additional tests which take advantage of the analogue LVDS driver capabilities of the SPLT. The test environment is set up in the same configuration as Figure 3.

3.5 PATTERN GENERATION AND CHECKING

A series of Packets can be pre-programmed into the SPLT for transmission as well as a series of expected packets that should be returned by the UUT. The SPLT can be set up to mimic a SpaceWire device that will be interfaced to the UUT. The test packets are transmitted from the SPLT at high speed and the UUT response checked against the pre-defined expected response. Any errors in the received bit-stream will then be flagged up by the SPLT.

3.6 SPACEWIRE INTERFACE WITH RMAP CAPABILITY

In SpaceWire interface mode, the SPLT is used to send and receive SpaceWire packets to a UUT from a PC through either USB 2.0 or Gigabit Ethernet connections. In this way, the SPLT works in a similar fashion to the STAR-Dundee SpaceWire Brick, but with added capabilities of LVDS margin testing. An RMAP target provides memory space which can be written to or read from by the UUT.

3.7 DETECTING ERRORS AND DEBUGGING CAPABILITIES OF THE SPLT

As the output LVDS signals are progressively degraded, bit errors on one, or both of the interfaced UUTs become increasingly likely. SpaceWire bit errors will manifest themselves as link disconnects due to detection of parity errors. Errors which are not detected by parity may still be picked up by implemented protocol features such as the RMAP cyclic redundancy check (CRC). The SPLT inbuilt Link Analyser Mk2 could then be triggered on detection of an RMAP header which reports a CRC failure.

The analogue signals received at the inputs of each SpaceWire port are buffered by the SPLT. An oscilloscope can be used to monitor the received data and strobe signals from both SpaceWire ports simultaneously. The SpaceWire data flowing through the SPLT SpaceWire ports is decoded by an inbuilt Link Analyser Mk2 and the characters are output on a Mictor connector suitable for interfacing to a Logic Analyser. A Host PC can be used to control the inbuilt SPLT Link Analyser Mk2 to trigger, store and read out captured data.

4 OPERATION OF THE LVDS DRIVERS

Figure 4 shows the components of the LVDS driver which are employed to manipulate the output of the SpaceWire signals.

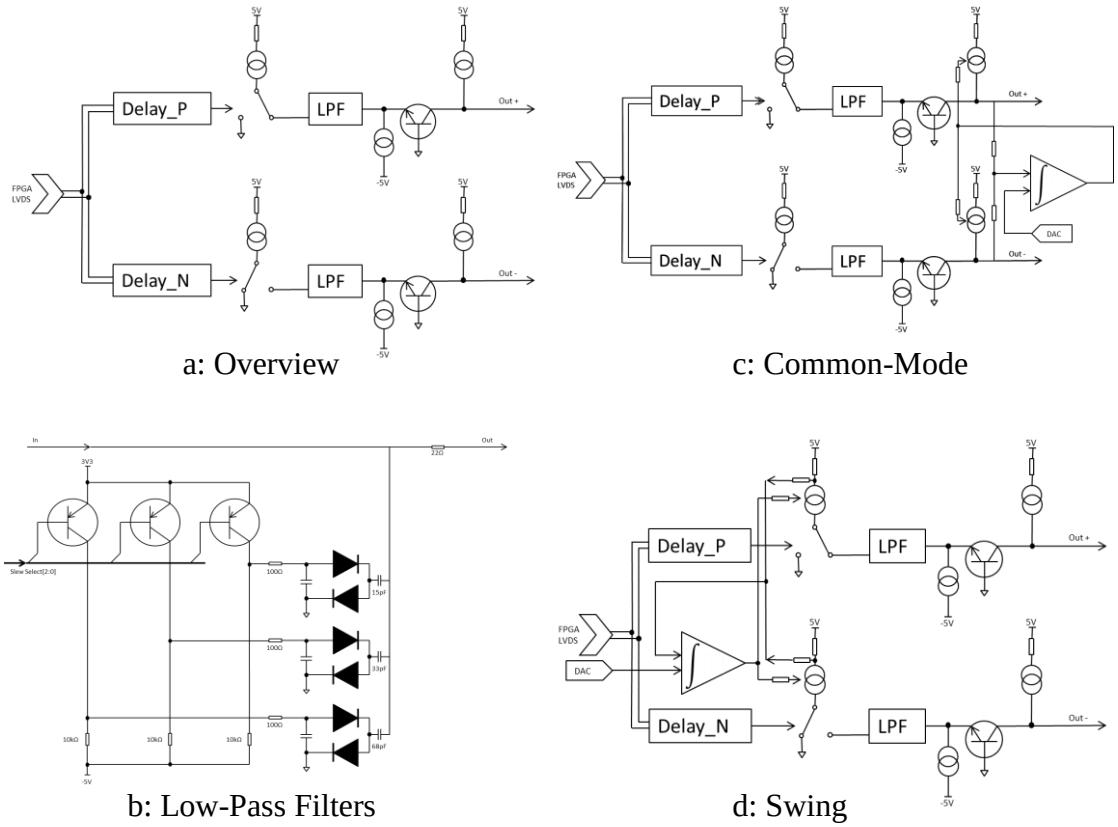


Figure 4: Operation of the LVDS driver circuitry. Figure 4a gives a simplified overview of the LVDS driver chain. b shows the operation of the low-pass filter (LPF) component. c and d respectively show the circuitry responsible for controlling the common-mode and swing of the LVDS signalling.

4.1 MANIPULATION OF SKEW AND JITTER USING THE DELAY LINES

Delay lines on the positive and negative transmission lines allow the measurable introduction of in-pair and Data-Strobe skew to a resolution of 10ps. A SpaceWire device should state what its maximum operating frequency is for a given Data-Strobe skew. The SPLT allows this test to be performed directly. Cables will typically state how much skew there is per unit length of cable. This information can be used to simulate different cables of different lengths.

Jitter can also be introduced into any of the delay lines. This can be used to simulate a device's stated jitter characteristics as well as deterioration of signal caused by electromagnetic interference (EMI).

4.2 MANIPULATION OF SLEW USING THE LOW PASS FILTERS

The low pass filters work by switching a combination of 3 capacitors of differing capacitance into or out of of the chain. This allows for eight different levels of slew to be introduced to a signal. Signal slew is typically caused by the capacitive effect of

the cable down which the signal travels. This allows the SPLT to simulate different cable characteristics such as length and mutual capacitance.

4.3 CONTROLLING THE SWING AND COMMON-MODE OF THE LVDS SIGNALLING

Digital to Analogue Converters (DACs) are used to set both the swing and the common-mode voltage by controlling reference voltage levels into operational amplifiers. Additional calibration voltages are used to ensure that both positive and negative components of the LVDS pair swing by the same magnitude about an equal common-mode.

Modification of the swing of the output signal simulates attenuation of the SpaceWire signal as it propagates through pcb traces and cables from transmitter to receiver. Attenuation can be simulated for media of differing lengths.

SpaceWire is not DC balanced and requires adequate common grounding between the communicating systems. Imbalances in ground plane, or power supply voltages between units could cause the DC level of the LVDS signalling to drift. The LVDS can test the permitted margins of this drift by manipulating the common-mode voltage of the output signal.

4.4 NO SINGLE POINT OF FAILURE

The design of the SPLT electronics guarantees that there will be no single point of failure on the SPLT that could damage SpaceWire equipment to which it is connected.

5 MEASURMENTS FROM THE LVDS DRIVERS

In order to demonstrate the capabilities of the SPLT, the Data and Strobe signals of a SpaceWire port were driven with a 25MHz square wave before manipulation by the analogue-electronics. The SpaceWire port was looped back into the analogue SpaceWire buffers on the SPLT so that an oscilloscope could be used to analyse the outputs. Screenshots from the oscilloscope measurements are presented in Figure 5 to show the discrete sources of signal disruption that the SPLT can introduce. Figure 5h then shows these sources combined to give a typical margin-testing waveform.

6 CONCLUSION

The SpaceWire Physical Layer Tester incorporates and builds upon established STAR-Dundee test products: the Link Analyser Mk2, the Conformance Tester and SpaceWire interface devices. Whether used with just a host PC, or in conjunction with a high speed oscilloscope and logic analyser, the SPLT can be connected in a variety of configurations to test SpaceWire systems. The ability to measurably deteriorate the SPLT output LVDS signals and to easily measure the input signals on an interfaced oscilloscope allows the SPLT to test, validate and verify SpaceWire systems from the physical and signal layer of the SpaceWire standard, right up to any protocols which are running on top of the SpaceWire standard.

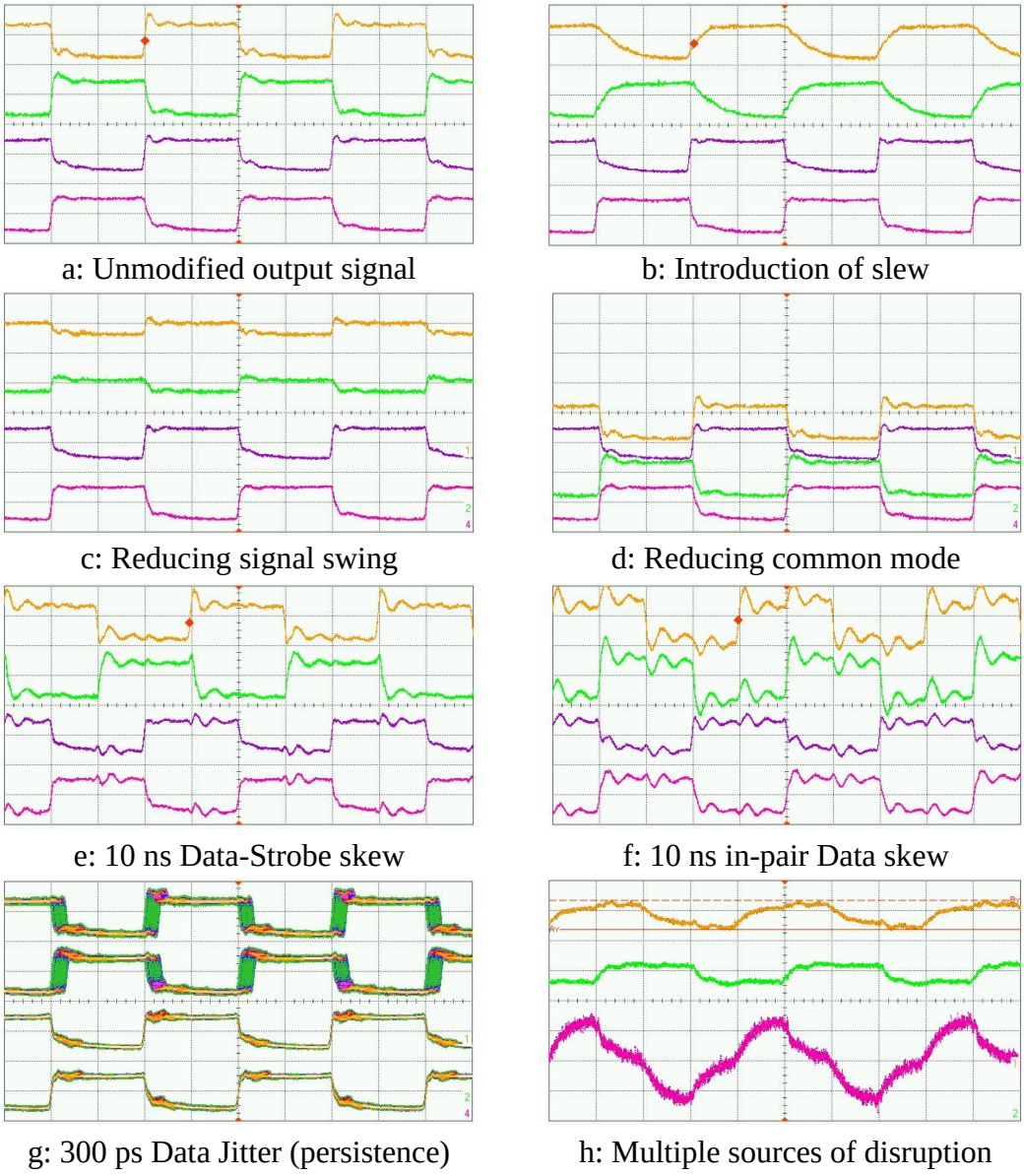


Figure 5: Measuring the different methods of deteriorating the LVDS signals from the SPLT. In these measurements, the Data (top pair) is manipulated with the strobe (bottom pair) untouched. Figure 5h omits the Strobe signal and replaces it with the subtraction function Data(+) – Data(-). Figure 5g uses infinite persistence to show applied jitter. Horizontal axis is 10 ns per division and vertical axis is 300 mV per division for all measured signals in all figures.

7 REFERENCES

1. Agilent Technologies, "Signal Integrity Solutions. Find Problems Now, Prevent Problems Next Time", 30th April 2010, Agilent Reference: 5988-5405EN.

2. Pete Scott, Steve Parkes, "SpaceWire Link Analyser Mk2: A New Analysis Device for SpaceWire Systems", International SpaceWire Conference 2010, St Petersburg, 22nd – 24th June 2010.

3. Steve Parkes, Martin Dunstan, "Debugging SpaceWire Devices using the Conformance Tester", International SpaceWire Conference 2007, Dundee, 17th – 19th June 2010.

SYSTEMATIC AND COMPLETE VERIFICATION OF SPACEWIRE BUS WITH MODEL CHECKING

Session: SpaceWire test and verification

Long Paper

Zhiping SHI[1], Zhiquan DAI[1],Yong GUAN[1], Minhua WU[1], Shengzhen JIN[1], Jie ZHANG[2], Xiaojuan LI[1]

[1] *Beijing Engineering Research Center of High Reliable Embedded System, Captial Normal University, Beijing, China*

[2]*College of Information Science & Technology, Beijing University of Chemical Technology, Beijing, China*

E-mail: shizhiping@gmail.com, woyun_23@163.com, guanyxxxy@263.net

ABSTRACT

The SpaceWire bus is usually used in safety-critical areas like aerospaces and other harsh environments. Therefore, it is vital to verify the correctness of SpaceWire designs and implementations. In this paper, the model checking method is employed to verify the SpaceWire bus system designed by the Beijing Engineering Research Center of High Reliable Embedded System of China. The eight modules of the SpaceWire Bus are verified and three errors are found. After corrected the errors according to the counter examples returned from the result of model checking verification, all the properties extracted from the protocol specification are proved valid. The results show that model checking is a simple and effective method with high level automation for SpaceWire protocol verification.

1 INTRODUCTION

The SpaceWire protocol [1] which was developed by the European Space Agency (ESA) has been applied to multiple in-orbit space equipments. ESA puts forward the protocal in natural language description; and there is not standard SpaceWire communications equipment so far. The designs and implementations of different developers of the SpaceWire protocol will be different. In the hardware circuit design, it is vital important to verify the circuit design satisfies the target specification or not.

In the hardware circuit design, finite-state machine is often used to show the circuit's function and can be well associated with Kripke structure of formal verification model. Secondly, in functional verification, finite-state machine only needs to verify its timing sequence correctness, don’t care about the specific concept of time, in this way,

high-order temporal logic in model checking method can well represent the functional properties. Last, there have been relatively mature automated verification tool that can help people efficiently finish the completeness verification of design and implementation [2-3]. It is very necessary that we use model checking to achieve the systematic and complete verification of SpaceWire systems.

2 VERIFY FLOW

In this paper, the verification object is the SpaceWire system circuit design implemented by the Reliable Embedded Systems Laboratory of the Capital Normal University China. The system is composed of the following eight modules: the baud rate selection module, the recovery module, the credit module, the time module, the faulting module, the control module, the sending module and the receiving module. Because the eight modules are functionally independent, the combined model checking is adopted to verify the eight modules respectively. The divide and rule method avoids the state space explosion problem to which the model checking is prone.

The verification process is as follows. At first, the system design is modelled using Kripke structure. Then, we use the branching-temporal logic to describe the properties of the protocol specification into the CTL formulas. Finally, a model checking tool, SMV, is employed to verify whether the logic formulas of the properties are valid in the formal models of the system design. The verification results show whether the implementation of the SpaceWire system accords with the protocol specification. The verification process is shown in Figure 1.

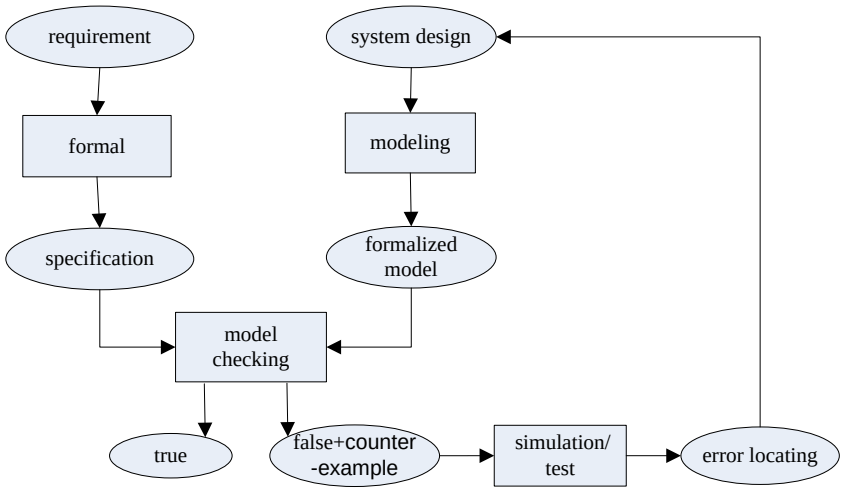


Figure 1. Formal Verification Process Graph

The SpaceWire circuit design is described in hardware description language VHDL, whereas the SMV system can accept Verilog description language [5] or SMV

language. Therefore, before verification, we use the third-party software X-HDL to transform the VHDL code into Verilog code. The state transition relations are abstracted from the Verilog code, and expressed by Kripke structure. In addition, through analyzing the protocol specification, we extract the properties that the system should satisfy and formalize the properties in higher-order temporal logic formulas. Finally, in order to reduce unnecessary verification cost, some signal variables in the design code, such as system reset signal, clock signal, are limited in certain range in accordance with the protocol specification. After the property formulas are merged into the design code, we use automated model checking tool SMV to carry out the verification.

3 SYSTEM VERIFICATION

According to protocol specification, we extract a total of fifty-five properties and describe them in higher-order temporal logic formulas. The verification shows that the seven formulas out of them are invalid. According to the counterexamples given by SMV, we check the code and dig out three design errors. Based on the protocol specification, we revise the design code, and then all the properties pass the new verification.

As the length is limited, we take a time code register module as an example to introduce the model checking process.

3.1 MODELLING

According to the SpaceWire protocol specification, the module of the time code registers will achieve the function as follows: If the module is reseted, the output data are invalid all-zero data, or if the Tick_In signal, which requests sending the time code, is valid and the HoldRegister signal, which represents the register pending, is invalid, this module will output a valid time code. Design code is shown below.

```
if (Reset == 1'b1)
        begin
            Time_Out <= {6{1'b0}} ;
            TimeControlFlag_Out <= {2{1'b0}} ;
        end
else
        begin
            if (Tick_In == 1'b1 & HoldRegister == 1'b0)
            begin
                Time_Out <= Time_In ;
                TimeControlFlag_Out <= TimeControlFlag_In ;
            end
        end
```

Figure. 2 Time code register module design code

According to the design code and combining the method of formal modelling, this module is modelled as the state transfer diagram as Figure 3.

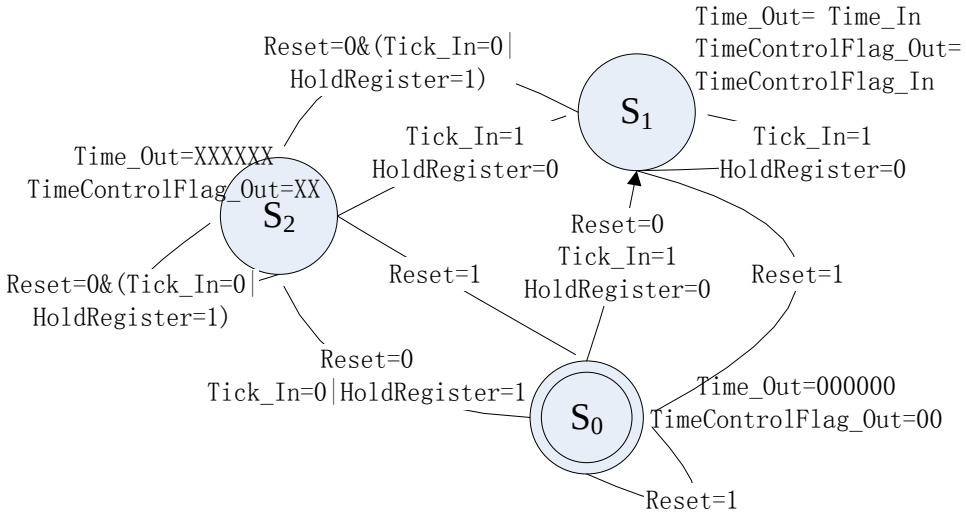


Figure 3. The state transfer diagram of the time code register module

If the system is reset, the module steps into the reset state. In this state, Time_Out, which requests the time code been sent out, and TimeControlFlag_Out, which requests control flag of the time code, are all assigned to zero. In the state S_0, if reset signal is invalid, Tick_In is valid, and HoldRegister is invalid, then the system will in the state S_1. In this state, Time_Out and TimeControlFlag_Out will be valid. If reset signal and Tick_In are invalid and HoldRegister is valid, the system will in the state S_2. In this state, Time_Out and TimeControlFlag_Out will random. Between the state S_1 and the state S_2 can also be interchangeable.

3.2 ANALYSIS OF PROPERTY VERIFIED

There are three properties to be verified in this module. Property1: If the system reset, the output time code and the control flag of the time code will be set zero. Property2: If the system is not in reset state, when Tick_In is valid and HoldRegister is invalid, the value of the time code and the control flag of the time code are equal to a moment of the corresponding input values. Property3: If the system is not in reset state, when Tick_In is invalid and HoldRegister is valid, the output time code and the control flag of the time code will be set zero.

Property1: SPEC AG(Reset -> AF (Time_Out =0 & TimeControlFlag_Out=0));

where, Reset is reset signal in the module; Time_Out is the time code data value that is need to be send; TimeControlFlag_Out is the time control code to send.

Property2: SPEC AG(Tick_In & ! HoldRegister & !Reset-> AF ((AX Time_Out[0] <-> Time_In[0]) & (AX Time_Out[1] <-> Time_In[1]) & (AX Time_Out[2] <-> Time_In[2]) & (AX Time_Out[3] <-> Time_In[3]) & (AX Time_Out[4] <-> Time_In[4]) & (AX Time_Out[5] <-> Time_In[5]) & (AX TimeControlFlag_Out[0] <-> TimeControlFlag_In[0]) & (AX TimeControlFlag_Out[1] <-> TimeControlFlag_In[1])));

Property3: SPEC AG ((!Tick_In | HoldRegister) & !Reset-> AF (Time_Out =0 & TimeControlFlag_Out=0));

Where Tick_In is the request signal that request to send the time code, and HoldRegister is the register pending signal, Time_Out is the value of time code to send out, and TimeControlFlag_Out is the control flag of the time code to send.

3.3 VERIFICATION

The SMV checker gives the result: Property1 and Property2 are true, but Property3 is false. The SMV gives the counter-example of Property3 (see Figure 4).

Through analyzing the counter-example, we find that the value of TimeControlFlag_Out is always one and TimeControlFlag_In is always zero from the third moment. By analyzing the design code, we find the error occurs because the condition sentence is incomplete. It does not designate the value of the output signal when the condition is false. So, we add the situation state assignment which is equivalent to the output signal of the reset, the revamped code is shown in Figure 5.

After modifying the design code, we verified the time code register module again, and the results show the three properties are true as in Figure6.

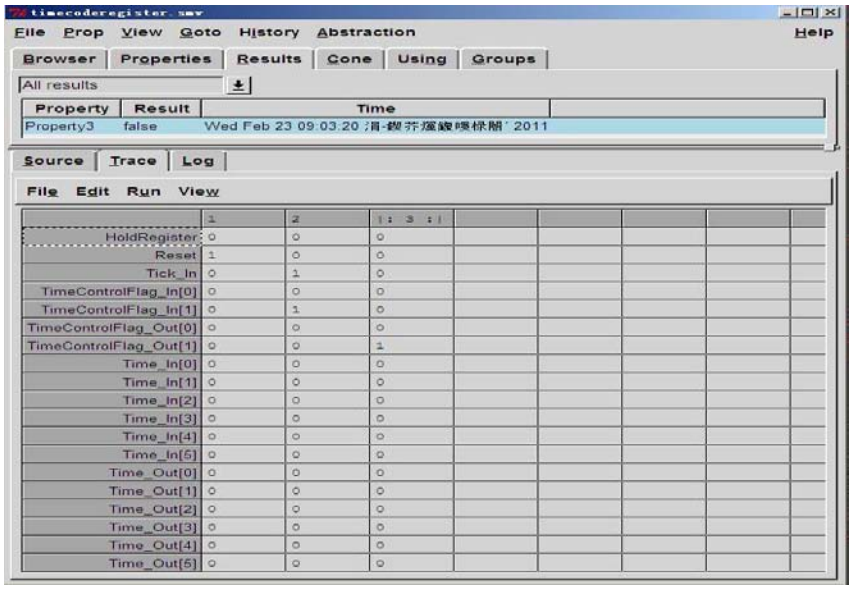


Figure 4. The counter-example of Property3

```
if (Reset == 1'b1)
        begin
            Time_Out <= {6{1'b0}} ;
            TimeControlFlag_Out <= {2{1'b0}} ;
        end
else
        begin
            if (Tick_In == 1'b1 & HoldRegister == 1'b0)
            begin
                Time_Out <= Time_In ;
                TimeControlFlag_Out <= TimeControlFlag_In ;
            end
        else
        begin
                Time_Out <= {6{1'b0}} ;
                TimeControlFlag_Out <= {2{1'b0}} ;
            end
    end
```

Figure 5. Design code modification of the time code register module

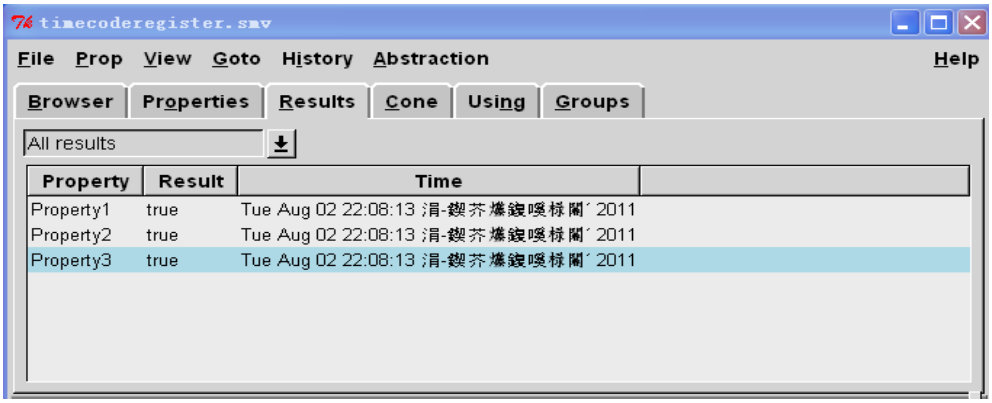


Figure 6. Verification results after modifying the design code

3.4 Errors Found and Solution

Errors have been found in three modules: the time code register sub-module,the send register sub-module in the sending module, and the faulting module.

In the time code register sub-module of the sending module, we find a property shown false. Through analying the design code and the counter-example, the error owes the

incomplete condition sentence. In the above example, we have illustrated the wrong reason and solution.

In the send register sub-module of the sending module, when it sends time code packet, we verify whether the data-control flag is consistent the protocol specification. The verification results of the two property formulas don't pass. By analyzing the counterexamples and the design code, we find the data-control flag of time-code packet is set to zero, inconsistent with the requirements in the protocol specification. In the original design, we modified the data-control flag setting based on the protocol specification, then the two property formulas pass verification..

In the faulting module, the generated errors have responding priority.the parity error, the escape error, the disconnect error, the credit error and the character sequence error, these five kinds of errors have increasing priorities in order. The 4 out of 7 properties of the faulting module failed to pass verification, because when both the high priority and the low priority error processing requests occur, the low priority error processing requests will be shielded by the high priority ones. Thus, it is possible that the high priority requests continuously come before the low priority errors got response, then low-priority requests would be starved to death. In order to prevent the low priority error processing requests from starving to death because the errors always fail to get response. For instance, we can count response times of diffident priority errors, if the response times of high priority errors have reached the configured maximum number of response times, the error priority will lower a level. So, the low-priority errors will not starve to death.

3.5 VERIFICATION RESULTS

We extract fifty-five properties totally from the eight modules, and most of them pass verification. There are seven properties failed to verify in the faulting module and two sub-modules of the sending module. As shown in table 1.

Table1 Verification Results of SpaceWire

Module name	**CTL formulas**	**True**	**False**
Baud rate selection module	3	3	0
Recovery module	3	3	0
Credit module	3	3	0
Time module	4	4	0
Faulting module	7	3	4
Control module	22	22	0
Sending module	5	3	3
Receiving module	8	8	0

For invalid properties, SMV put forwards corresponding counterexamples that cause properties invalid. Based on the counterexamples for the checked properties, we found the design errors in the design code after analyze the protocol specification and design code. Finally, we modify and improve the SpaceWire system design, then the property which prior failed to verify pass the verification.

4 CONCLUSIONS

We use the model checking method to verify the design of SpaceWire bus system implemented by Capital Normal University China. In order to avoid states explosion problems, this paper adopts divide and rule methodology to independently verify the eight modules. We abstract and verify fifty-five properties, out of which seven properties failed to verification . The design flaw are found and revised according to the counterexamples that SMV system given. After revised, the seven properties pass verification. The SpaceWire system has higher reliability after verified and revised.

Model checking is a simple and effective method of formal verification with high level automation, and is a feasible protocol verification method.

5 REFERENCES

1 ECSS-E-50-12A. “Space engineering. SpaceWire - links, nodes, routers, and networks”[S]. European Cooperation for Space Standardization. http://www.ecss.nl/forums/ecss/dispatch.cgi/standards/showFile/100302/d20090722143301/ No/ECSS-E-50-12A(24January2003).pdf. 2003.

2 E.M.Clarke, E.A.Emerson, and A.P.Sistla. “Automatic verification of finite state concurrent systems using temporal logic specifications”[J].ACM transactions on Programming Languages and Systems, 1986,pp. 244-263.

3 M.C.Browne, E.M.Clarke, D.L.Dill. “Automatic verification of sequential circuits using temporal logic”. IEEE Transactions on Computers, C-35(12),1986,pp.1035-1044.

4 J.R.Burch, E.M.Clarke, D.Long, K.L.McMillan, D.L.Dill. “Symbolic model checking for sequential circuit verification”. IEEE Transactions on CAD,13(4),1994,pp. 401-424.

5 Tomáš Kratochvíla, Vojtěch Řehák and Pavel Šimeček. “Verification of COMBO6 VHDL Design”[J] CESNET. 2003.

Thursday 10 November

Test and Verification 2

THE DEVELOPMENT OF THE SPACEWIRE COMMUNICATION TESTER (SPACEWIRE TEST MODULE)

Session: SpaceWire Test and Verification

Short Paper

Shoji Komatsu, Naohisa Anabuki, Hiroshi Tsunemi

Earth and Space Science, Graduate School of Science, Osaka University, 1-1 Machikaneyama, Toyonaka, Osaka 560-0043, Japan

Masaharu Nomachi

Laboratory of Nuclear Physics, Graduate School of Science, Osaka University, 1-1 Machikaneyama, Toyonaka, Osaka 560-0043, Japan

E-mail: s-komatsu@ess.sci.osaka-u.ac.jp, anabuki@ess.sci.osaka-u.ac.jp, nomachi@lns.sci.osaka-u.ac.jp

ABSTRACT

SpaceWire is going to be adopted to some Japanese satellite missions, and the development of SpaceWire devices of satellites is increased. And, the support devices of the development such as a protocol converter to communicate between a general computer and SpaceWire devices and a debug tool for SpaceWire links also become important in laboratory experiment. Thus, we are developing the SpaceWire communication tester named SpaceWire Test Module (STM) as a part of the support devices. STM is a SpaceWire communication analysis and debug tool which has four function modules, Statistics Counters, User-defined Function (Analyser), Signalling Rate Counters, and Self-checking Function. We would like to emphasize that STM can be introduced to one's laboratory at low cost and customizable for any one's purpose because the FPGA logic and the application software will be opened.

1 INTRODUCTION

In Japan, SpaceWire is adopted to future scientific satellite missions, for example, BepiColombo/MMO, SPRINT (Small space science Platform for Rapid Investigation and Test) series, and ASTRO-H. The demonstration of SpaceWire technologies has already been performed by SDS-I/SWIM launched in 2009, and the opportunity of developing instruments with SpaceWire interfaces will be increased in small/large satellite missions and also in balloon-borne experiments. In addition, SpaceWire IP and RMAP Target IP distributed at Shimafuji site and SpaceWire/RMAP Library released via SpaceWire Users Group Japan encourage one to use SpaceWire in laboratory experiments. Therefore, we have developed a low-cost and user-customizable SpaceWire communication tester named SpaceWire Test Module (STM) for laboratory use. We have fabricated the board and designed a FPGA logic using SpaceWire IP described above, and developed the application software. These products are also open. This paper presents a hardware, FPGA logic, and application software of STM.

2 HARDWARE AND THE SETUP

STM has FPGA, three SpaceWire ports, RS-232C port, and two for input and two for output LEMO ports (LVTTL level) as shown in Figure 1. FPGA is Altera Cyclone III. SpaceWire0 is RJ45 connector and SpaceWire1 and 2 are D-sub 9 pin connectors. The clock of SpaceWire reciever is 175MHz. RS-232C port is to communicate a computer. The baud rate of RS-232C is 115.2 kbps.

Figure 2 shows the hardware set-up of STM. STM is placed between two pieces of SpaceWire devices to SpaceWire1 and SpaceWire2 in Figure 1. STM is controlled by a Linux computer through RS-232C. STM User may use a Serial-USB convertor because a laptop computer usually has no serial port.

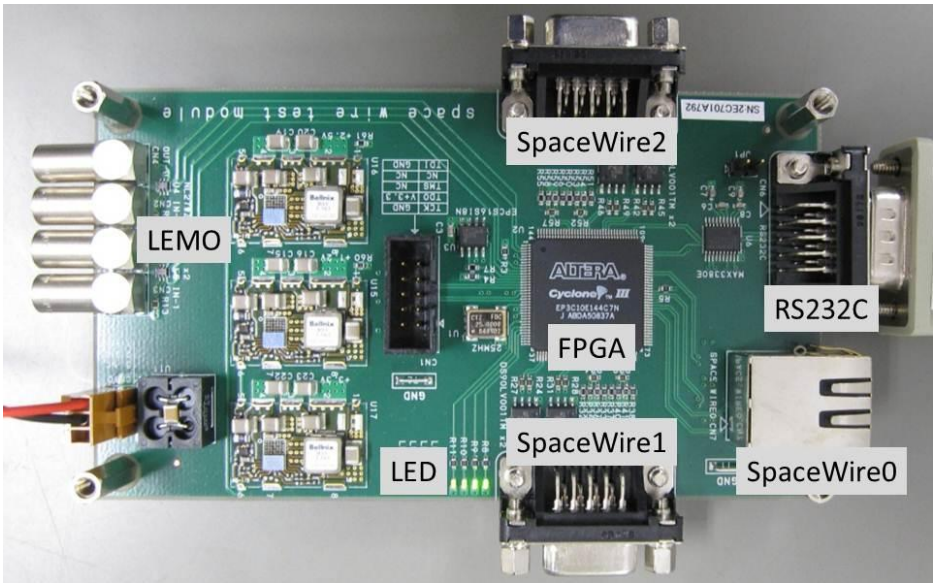


Figure 1: Outside of STM

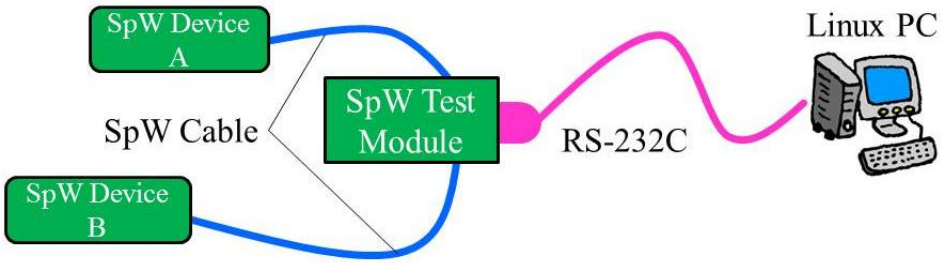


Figure 2: Hardware Set-up

3 WHAT STM MONITORS

Figure 3 shows a functional block diagram of the STM FPGA. There are four main function modules, “Signalling Rate Counters”, “Statistics Counters”, “User-defined Functions”, and “Self-checking Function”. SpaceWire link interfaces in STM are made by utilizing open SpaceWire IP distributed by SpaceWire User Group Japan.

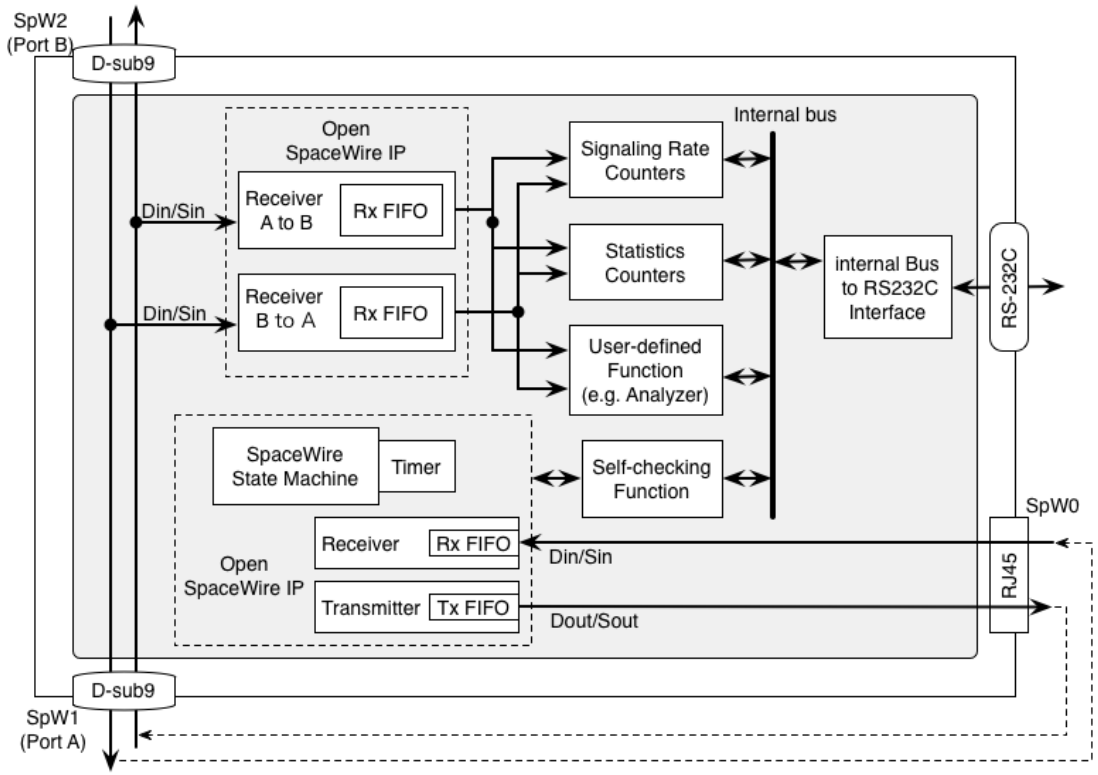


Figure 3: Functional Block Diagram of the FPGA

3.1 SIGNALLING RATE COUNTERS

Signaling Rate Counters measures transmission speeds of bidirectional SpaceWire links. This module counts the rising-edge of receiver clocks recovered by receivers connected to each SpaceWire port. The measurements are recorded on FPGA registers and updated per 0.1 second.

3.2 STATISTICS COUNTERS

As described in ECSS-E-ST-5012C, receivers of the SpaceWire IP recognize SpaceWire control characters and control codes, and the detection signals such as gotNull, gotFCT, gotN-Char, GotTime-Code are set. RxErr (disconnect, parity, and escape error), credit error, and character sequencer error are also detected in the receiver modules. Statistics Counters counts these flags and records the statistics (cumulative total value or the rate per a second) on the registers.

3.3 USER-DEFINED FUNCTION (ANALYSER)

User-defined Function is provided to STM users as an extra room to implement advanced features for any purpose. Our sample logic will offer advanced features to capture a series of SpaceWire packets triggered by specified SpaceWire characters, codes, or errors.

3.4 SELF-CHECKING FUNCTION

Self-checking Function works as a dummy SpaceWire device to debug Statistics Counters and Analyser. As shown in figure 3, the arbitrary patterns including invalid

SpaceWire packets set by a computer through the RS-232C are sent to both SpaceWire1 and 2, and then the other modules captures those patterns.

4 STM SOFTWARE

STM software is a multi-platform and user-friendly graphical interface software designed with C++ and cross-platform application and UI frame work Qt. STM is controlled by the software. The software reads FPGA registers that store the data from Signaling Rate Counters, Statistics Counters, and Analyser modules per a second, and then it shows all data numerically and plots the data from Statistics Counters and Signalling Rate Counters on the time series graph. The log files for each functional module and each link are generated automatically. It is also possible to plot the past data.

Figure 4: GUI window of STM software

5 STATUS OF DEVELOPMENT

The status of the development of STM is under the verification of FPGA logic and the software together. The verification of Signalling Rate Counters for 24 hours was conducted using two SpaceWire devices. This verification was repeated enough to say that Signalling Rate Counters is able to continue working right and long enough. The verification of Statistics Counters and Analyser that Self-checking Function sends a large variety of the arbitrary SpaceWire packets as I mentioned in section 3.4 will be conducted in near future.

6 CONCLUSION

We are developing SpaceWire Test Module (STM) as a low-cost and customizable SpaceWire communication tester for the development of SpaceWire devices. STM has four function modules, “Statistics Counters; User-defined Function (Analyser); Signalling Rate Counters; and Self-checking Function”. These function modules provide us the information of SpaceWire links. We are in the stage of the verification, and the verification for Signalling Rate Counters has done. After the verification for the rest function modules has done, we are going to deliver STM and open the source codes through SpaceWire User Group Japan. Then, STM users are able to modify the source codes as the users like.

ADVANTAGES OF A SPACEWIRE BACKPLANE DURING SPACECRAFT UNIT INTEGRATION AND TEST

Session: SpaceWire Test and Verification

Short Paper

A. Senior, P. Worsfold.
SEA, Building 660, Bristol Business Park, Coldharbour Lane, Bristol, BS16 1EJ, United Kingdom

E-mail: alan.senior@sea.co.uk, peter.worsfold@sea.co.uk

1 ABSTRACT

Spacecraft units are typically composed of a set of Printed Circuit Boards (PCBs) which are connected together within the unit via a backplane PCB. Each of the PCBs incorporates a subset of the unit's functionality. As the semiconductor technologies have achieved higher and higher levels of integration the functional complexity of the PCBs has increased and this in turn has lead to increasing the performance requirements on the test environment, including the test software.

From the Assembly Integration and Test (AIT) view point this modular unit construction is particularly attractive since the functions can be verified individually and then integrated, so that a complex unit can be built from known working sub-functions. However traditionally the number and types of backplane interfaces is many and varied and this introduces a significant level of additional complexity to the integration and test activities.

A SpaceWire[1] Active Backplane[2] is one solution, providing the possibility of module emulation and packet monitoring to permit functional verification with high visibility of the data traffic between PCBs.

2 BACKPLANE BASED SPACECRAFT UNITS

Figure 1 shows a typical spacecraft unit, which consists of a set of PCBs (Modules) that slide into a card frame and mate with a backplane PCB (highlighted in green).

The advantage of this modular unit design is that an individual Module can be removed or replaced without the need to dismantle the complete unit. From the Assembly Integration and Test (AIT) view point this type of modular construction is also attractive since the Modules can be tested individually and then integrated so that a unit with a high functional complexity can be built from simpler sub-functions that are easier to debug individually.

The backplane is a key PCB since it carries the communication signal and power lines to each Module. Figure 2 shows an example unit that has been designed for "Spacecraft A" and uses a set of Modules that are powered from the backplane and interconnect with a mix of discrete control and monitoring signals.

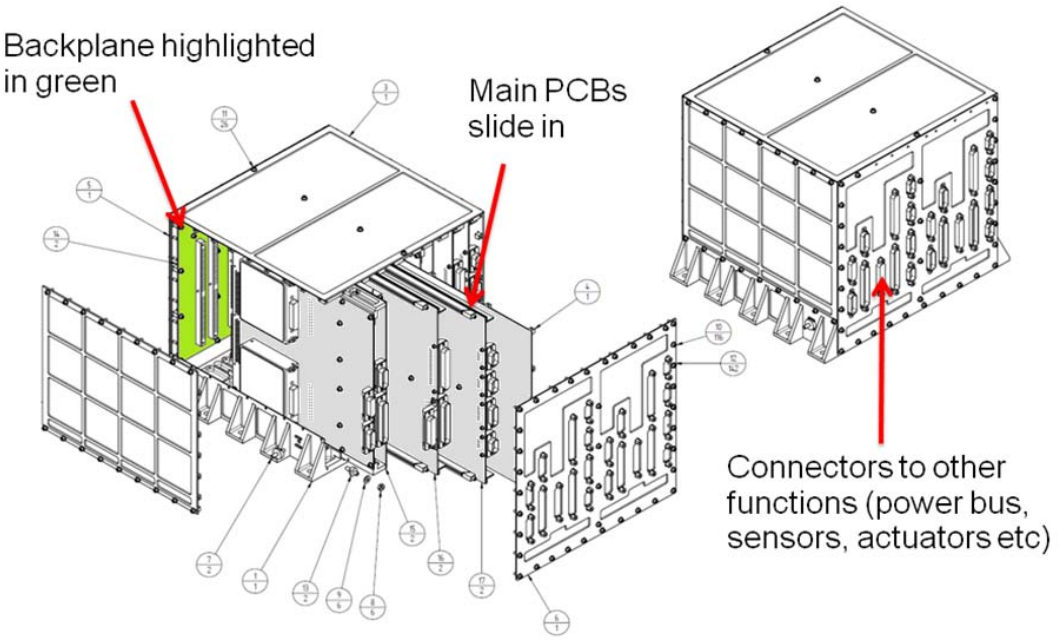


Figure 1: A typical Spacecraft unit consists of a box containing PCBs that mate with a backplane

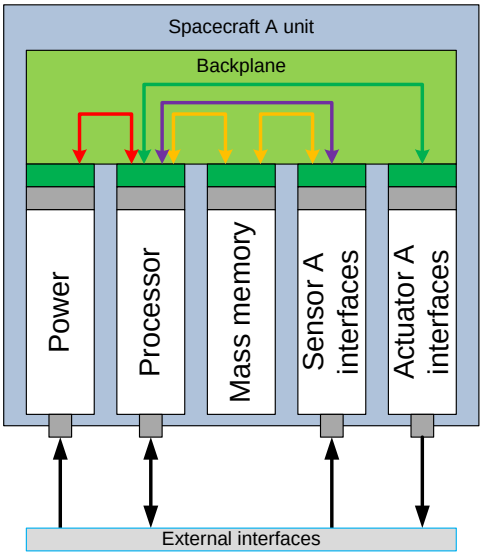


Figure 2: In a typical unit the backplane routes a mix of signal types that use different protocols

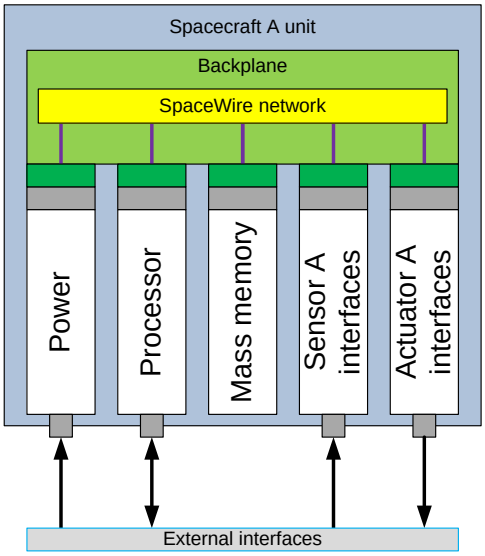


Figure 3: An alternative SpW based backplane approach that simplifies AIT

3 IMPACT ON ASSEMBLY, INTEGRATION AND TEST

The backplane interfaces in current unit designs are typically chosen to interface directly between the varieties of different semiconductor device types used on each Module, for example these interfaces may be simple serial interface busses, multiplexer address lines, pulses or clocks as well as discrete bi-level status and control signals. These interfaces are likely to use different electrical levels and information exchange protocols. Within Figure 2 the different backplane interface types are represented by different coloured links between the set of modules. During the AIT activity, the variety of electrical interfaces makes testing the modules individually much more difficult since the test equipment must reproduce all the

required interface types so that the boards are stimulated in an electrical environment that is representative of flight. Furthermore these interfaces are likely to change for the next mission dependant on the mix of device technologies used and the varying performance requirements for the different application, thus new AIT Electrical Ground Support Equipment (EGSE) must be designed.

A solution is to change all the backplane communications interfaces to SpW, as shown in Figure 3. This change in design approach adds complexity at the early design stages and at the PCB component level, however there are major advantages during the later AIT activities since now the module test environment needs only to support one backplane communication standard and in most cases the required test equipment can be bought off the shelf from suppliers[3,4,5] complete with configuration, test and monitoring software tools. When a unit is required for the next generation Spacecraft, Figure 4, then the existing unit can be expanded if required by extending the SpW network to support new Modules without impacting on the inherited module hardware designs, and a high proportion of the EGSE can be re-used.

Though changing the backplane interfaces to just SpW links and power connections will cater for the majority of the PCB module needs, it is anticipated that the backplane connector interface may also need to route a limited number of discrete signals (Figure 5). These extra signal paths could be, for example, for timing signals, FDIR status and control signals, backplane slot address codes, power status and reset signals etc. that cannot be easily carried via the SpW communication channel without a disproportionate increase in system complexity. Minimisation of the number, variety and complexity of these extra interface types is clearly an important design aim[6].

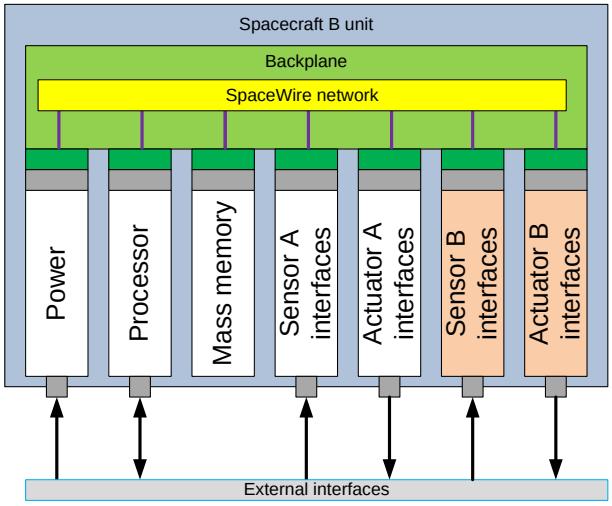


Figure 4: Spacecraft B can re-use Spacecraft A modules as well as their associated AIT equipment, the SpW network can be expanded to support the additional interface functions

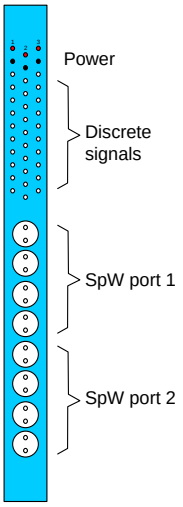


Figure 5: A SpW backplane offers a common backplane interface to modules

4 MODULE EMULATION AND UNIT TEST

A key advantage of the SpW backplane during AIT activities is that any missing PCB module (or modules) can be emulated by EGSE that is connected to a spare backplane or specifically provided SpW EGSE port (Figure 6). Even if a module is present in

the unit, the network addressing can usually be reconfigured to permit emulation of that module to debug the unit hardware and software functions via a SpW port dedicated to AIT EGSE. This EGSE port can also permit network packets to be observed by passing the packets to the EGSE and back into the system. For time critical cases where packet latency is an issue "packet sniffing" can be achieved by using an extender card, with the SpW signals routed to the backplane from the module as well as to the SpW test equipment.

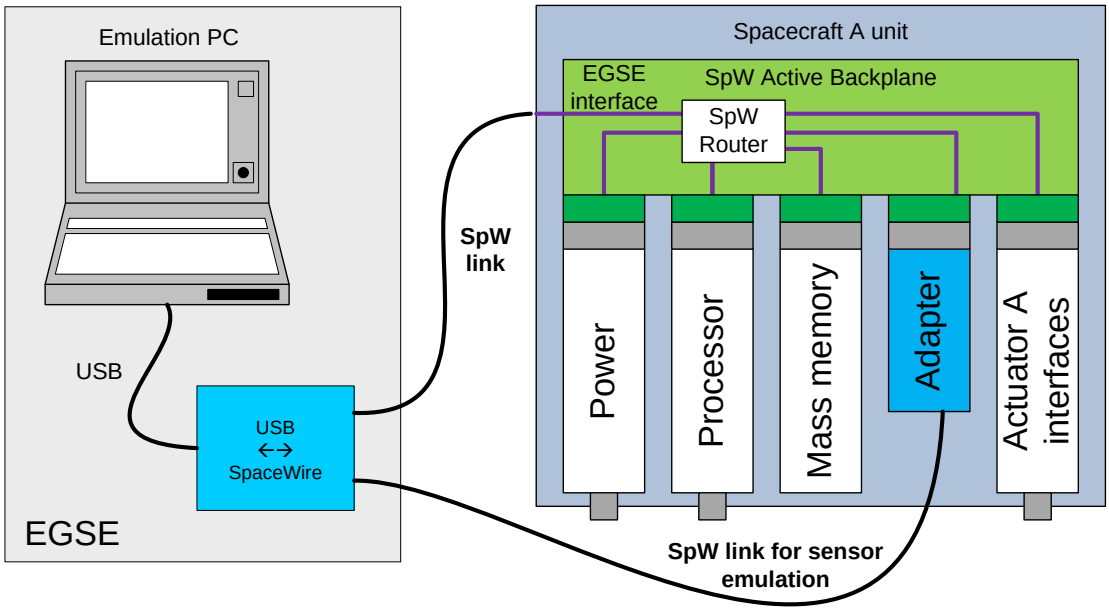


Figure 6: Potential unit test setup during Assembly Integration and Test (AIT)

5 REFERENCES

1. ECSS-E-ST-50-12C, (SpaceWire – Links, nodes, routers and networks, Issue 2, 31st July 2008.

2. A. Senior, P. Worsfold, "A SpaceWire Active Backplane Specification for Space Systems", International SpaceWire Conference 2010.

3. 4-Links: www.4links.co.uk

4. Star Dundee: www.star-dundee.com

5. Skylab Industries: www.skylab-corporate.com

6. A. Senior, W. Gasti, O. Emam, T. Jorden, R. Knowelden, S. Fowell, "Modular Architecture for Robust Computation", International SpaceWire Conference 2008.

ETHERNET TO SPACEWIRE BRIDGE - AN EVOLUTION OF SERVICES

Session: Test and Verification

Short Paper

Kristoffer Glembo, Marko Isomäki, Sandi Habinc

Aeroflex Gaisler AB, Kungsgatan 12, SE-411 19 Göteborg, Sweden

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

ABSTRACT

The Aeroflex Gaisler Ethernet to SpaceWire bridge [1] facilitates rapid development and testing of SpaceWire equipment by providing bridging between three SpaceWire links and one 10/100 Mbit/s Ethernet link. In addition to the three physical SpaceWire links six virtual links are interfaced through TCP/IP sockets over the Ethernet link.

This product has been available since 2006 with 100 Mbit/s SpaceWire links and 25 Mbit/s aggregate effective throughput on the Ethernet link. To support future applications requiring higher bandwidth and other interfaces new versions of the bridge are under development which will provide up to 500 Mbit/s over the Ethernet link. This paper presents the technical details of the current device and the road-plan for future versions.

1 INTRODUCTION

The Ethernet to SpaceWire bridge provides bridging between three 100 Mbit/s SpaceWire links and one 10/100 Mbit/s Ethernet link. The Ethernet communication is handled by six virtual links interfaced through TCP/IP. This allows a developer to generate test data on a workstation, send it over TCP/IP and forwarded by the GRESB to the appropriate SpaceWire or Ethernet destination.

The link speeds on both the SpaceWire and Ethernet links do not support the highest speeds available on the respective network and the bridge might thus not be suitable for all applications.

Several other interfaces such as CAN 2.0B, CCSDS/ECSS TM/TC MIL-STD-1553B and SPI are often used in space applications. The intention is therefore to add one or more of these interfaces to the bridge making it more versatile.

This paper begins with a detailed presentation of the current features of the core and their technical details followed by how speed and functionality will be improved in future versions. Lastly the introduction of additional interfaces will be discussed.

2 CURRENT FEATURES

The three SpaceWire links provided by the bridge support up to 100 Mbit/s and are addressed by a SpW address. In addition to this there are six virtual links interfaced through TCP/IP sockets over the Ethernet link which are also addressed with a SpW address.

All nine links can be configured in a routing table which allows a developer to generate test data on a workstation, send it over TCP/IP and forwarded by the GRESB to the appropriate SpaceWire or Ethernet destination. The routing table and TCP/IP sockets are implemented in software using uClinux (linux-2.0.x). This older kernel was chosen because it introduces less load on the processor thus allowing for higher throughput on the TCP/IP links.

The aggregate throughput on the Ethernet link is up to 50 Mbit/s (full-duplex) being one order of magnitude lower than what is (ideally) available full-duplex on the three SpaceWire links (456 Mbit/s). This is still very useful in a lot of applications but not in maximum throughput testing.

The hardware is implemented on a Xilinx Spartan 3 FPGA which limits area and frequency requiring suboptimal configurations of both CPU and Ethernet core but makes it a cost effective solution.

The bridge supports Internet tunneling without the need for a workstation or PC to be connected to the unit (unlike other solutions on the market). Tunneling allows SpaceWire based equipment and satellites to be integrated at multiple remote sites and be interconnected through SpaceWire networks.

Configuration of the routing table, SpaceWire links and the Ethernet connection can be done through a web interface provided by a webserver running on the bridge's Linux kernel. It also shows status such as packet/data counters and SpaceWire link status. Configuration can additionally be done through the TCP/IP sockets using a custom protocol.

The bridge also fully supports the GRMON software debugger tool which allows remote upload and debugging of software on SPARC based processors such as LEON2/LEON3/LEON4 through the use of RMAP. Any link on the bridge and any SpaceWire destination address can be accessed by specifying command line parameters. Once a session started it will run transparently with no further configuration needed.

Figure: First generation of GRESB

3 IMPROVING PERFORMANCE

The current version of the GRESB has a competitive set of functions with the main limitation being throughput. This will be addressed by new versions of the device planned for the near future. To achieve higher bandwidth several steps are performed.

For the first new version a new FPGA is chosen enabling higher frequency and larger caches for the LEON3 processor, faster SpaceWire links and a faster Ethernet device. The benefit of doing only these changes is that existing IP cores can be used and software does not need to be modified. Thus a significant performance improvement will be achieved with little effort. The intention is to use a Xilinx Spartan 6 FPGA which in addition to a higher frequency will fit a LEON3 with larger caches, a Gigabit Ethernet device and 200 Mbit/s GRSPW2 SpaceWire links.

The total full-duplex SpaceWire throughput will now be 912 Mbit/s but the main bottleneck will still be the TCP/IP connections. Although the device now contains a Gigabit Ethernet device the TCP/IP connections will still be in software and thus processor limited. The new Ethernet core contains performance improving features such as scatter-gather DMA and checksum offloading but the full-duplex throughput will probably not be more than 80 Mbit/s. Trial runs show that the frequency improvement between Spartan3 and 6 for Leon3 systems is 10-20% and the total processor performance increase (taking into account caches etc.) can be up to 30%.

The second step results in a much bigger leap forward in terms of both functionality and performance. The software will mostly be kept as is with the webserver and TCP/IP links. But in addition to this there will be one channel available for Ethernet communication using an UDP based protocol handled completely in hardware. This will allow for a throughput up to 500 Mbit/s which is now in the same order of magnitude as the upgraded SpaceWire links.

The SpaceWire links will also be replaced with a router core handling all SpaceWire routing in hardware further offloading software. Forwarding of TCP/IP packets will still be handled by software which needs to hold a routing table for the virtual channels. There will still be a large gain in performance since any SpaceWire to SpaceWire communication will not be seen by the processor and high bandwidth data from the host can now be transferred using the UDP based hardware protocol.

4 ADDITION OF NEW INTERFACES

Due to customer requests several custom made bridges have already been deployed to customers with a CAN 2.0B interface. There has been a demand for other interfaces as well such as CCSDS/ECSS TM/TC, MIL-STD-1553B and SPI.

Due to area limitations the versions previously shipped needed removal of other cores in the configuration to fit the CAN 2.0B core. With the change to the larger Spartan6 device it will now be possible to fit all the interfaces without the removal of any present ones. The new interfaces will be accessed from the TCP/IP links in the same way as for Ethernet to SpaceWire communication. Each interface will have its dedicated TCP/IP connection thus leaving the routing table unmodified.

A possibility is also to increase the number of SpaceWire ports. There has not been a specific demand but probably even up to 8 ports could be useful. This combined with the other new interfaces requires a new box to be used as well.

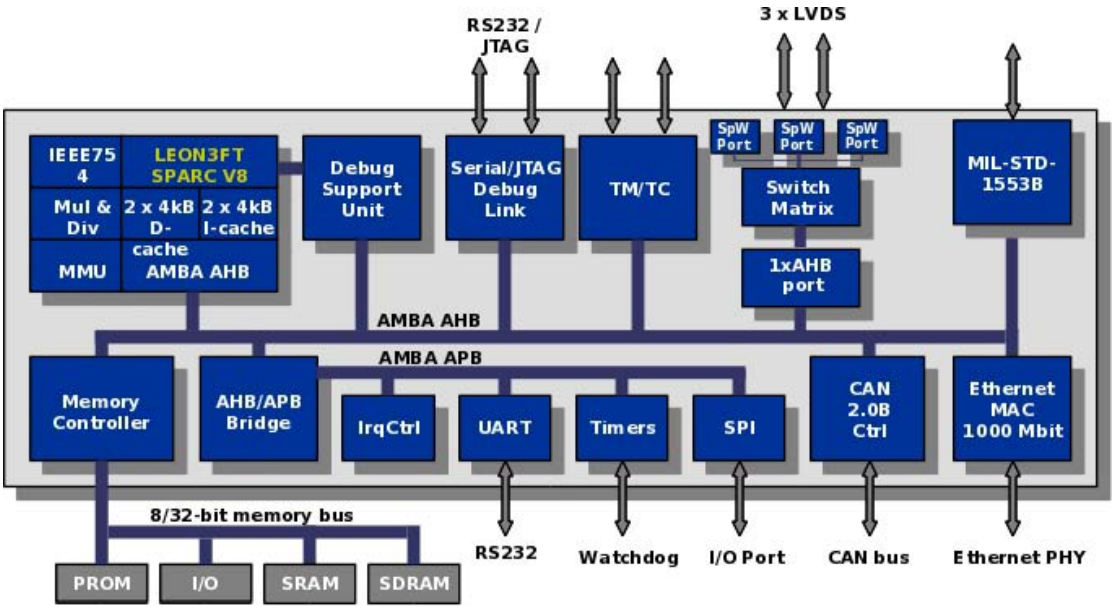


Figure: Second generation GRESB block diagram

5 CONCLUSION

The Ethernet to SpaceWire Bridge provides a versatile and easy to use unit for SpaceWire testdata generation on typical host computers. With the upgrade to next generation it will also be a high performance device with the possibility to achieve maximum SpaceWire throughput as well as extending the concept with additional interface types.

6 REFERENCES

1. GRESB User's manual, Aeroflex Gaisler, www.gaisler.com

OFF THE SHELF WIRELESS BRIDGES INTERFACING TO SPACEWIRE: POSSIBILITIES, PRACTICALITIES AND OPPORTUNITIES

Session: SpaceWire Test and Verification

Short Paper

Eric Pritchard, Dick Durrant and Alan Fromberg

Systems Engineering & Assessment Ltd (SEA),
SEA House, Building 660, Bristol Business Park,
Coldharbour Lane, Bristol BS16 1EJ. United Kingdom

Jean Francois Dufour

D/TEC-EDD Computer and Data Handling Engineer
European Space Agency, ESTEC,
Keplerlaan 1, 2201AZ, Noordwijk, The Netherlands.

E-mail: ewp@sea.co.uk, rjd@sea.co.uk, aff@sea.co.uk,

Jean-Francois.Dufour@esa.int

ABSTRACT

SEA has performed two activities to assess the practicality of providing a wireless bridge to interface to SpaceWire. The first conducted experiments interfaced IEEE.802.11 based protocols with SpaceWire via Ethernet, then a practical test of Ultra Wideband USB. The second assessed the feasibility of combining wireless data and power transmission to minimise physical interaction during system Assembly Integration and Verification (AIV), for example when Planetary Protection (PP) requires aseptic assembly.

A number of cases can be considered for making use of Commercial Off The Shelf wireless communications to a SpaceWire network, e.g. to reduce harnessing and connector make/break during AIV, for communication with the passenger satellite during launch and immediately post-separation and even to provide communications links for localised formation flying. The potential merits of these use cases and the maturity of the technology to address these markets is considered.

Implications for how a SpaceWire architecture can best interface to a wireless bridge and the measured performance data rates are reported. In particular the data rates achieved are much lower than those claimed by manufacturers and mitigating steps must be taken if acceptable data rates are to be achieved.

The feasibility study for the development of contactless telemetry exchange & power supply between a spacecraft under test and its Electrical Ground Support Equipment is also reported. The particular case for which PP and cleanliness requirements will require aseptic assembly is considered.

STOCHASTIC PETRI NETS MODELING AND ANALYSIS OF FAULT TOLERANCE FOR SPACEWIRE BUS

Session: Test and Verification
Short Paper

Xie Weihua, Jing Xiaochuan, Lin Xiaofeng, Chen Xianglong
China Aerospace Engineering Consultation Center
E-mail: myhello@126.com , jingxch@gmail.com

ABSTRACT

As the factual standard for the Intelligent Space Bus of new generation, SpaceWire exhibits enormous advantage in transmission speed, structural expansibility, systematic fault-tolerance, etc, which make SpaceWire outperform traditional bus techniques, eg. the CAN Bus. Therefore, SpaceWire has been widely applied to onboard spacecrafts. The fault-tolerance of SpaceWire is the key factor to implement the high-reliability design for the control and load system of spacecrafts. However, there are many limitations to verify the effectiveness of fault-tolerance using traditional test methods. In this paper, the fault-tolerance of SpaceWire is firstly studied, and then the stochastic Petri nets is used to model SpaceWire in formal way, finally a generalized stochastic Petri nets model is established. According to the computational analysis on the tool kits, meanwhile based on the verification on the utility of the fault-tolerance of SpaceWire under the failure modes of spacecrafts, this paper can provide technical support for the design of the control and load system of typical spacecrafts.

Key Words: SpaceWire bus; Fault tolerance; Stochastic Petri net;

1. Introduction

As the satellite and deep space exploration technology is developing gradually, the requirements of the data bus is becoming more and more stringent. A general-purpose space data bus which is high-speed, scalable, low-power, low-cost is needed urgently to meet the data processing requirements. SpaceWire is proposed by ESA in order to solve the on-board data processing issues. SpaceWire which can build modularization and reconfigurable adaptive systems has been applied in the Mars Express, Smart-1 and other space missions successfully.

In-depth analysis of the protocol is an important application of SpaceWire bus. The protocol analysis by the formal technologies has become an important technical means, which attaches great attentions to research in this area in many countries, such as the UK's National Physical Council (NPL), the French National Communications Research Centre, German National Communications (GMD) and U.S. National Research Council standardization Bureau.

Finite state machine is the most commonly used protocol formal description techniques, usually using State Transition Graphs to represent. Petri Nets (PN) has been widely used in the communications field. Petri Nets can clearly express the communication between two processes. By adding some special models, Petri Nets have a variety of extensions, including Stochastic Petri Net (SPN), Colored Petri Net (CPN) and Time Petri Net (TPN), etc.

2. The Stochastic Petri net Model of SpaceWire

The working process of SpaceWire is as follows: when the system electrification is reset, the system enters the ErrorReset state, simultaneously the sender and receiver enter the reset state. And then, the receiver is enabled, and is started to check data flow. After waiting for a sufficient preparing time, the system enters the Started state. And then, the sender sends a NULL byte, and the receiver continuously checks the NULL byte. When the receiver receives the NULL byte, it enters the Connecting state, where the SpaceWire controller starts to send the FCT and NULL byte. When the receiver receives the FCT, it transfers to the Run state. If the receiver fails to receive the FCT within a given period, it enters the ErrorReset state immediately. In the Run state, the receiver starts and the sender sends the data such as Time-Code, FCT, N-Chars, and NULL, etc. During the working process, if any mistake occurs in the connection, the system enters the ErrorReset state immediately.

SpaceWire uses a peer-to-peer full-duplex protocol, therefore both the sender and receiver have the sending and receiving functionality. The Petri net model for the working flow is shown as fig.1. The initiation of the Petri net is marked as M0 that is placed at P1 (the sender is in ErrorReset state) and P6 (the receiver is in ErrorReset state).The definition of the symbols in Fig.1 is shown in Tab.1 and Tab.2.

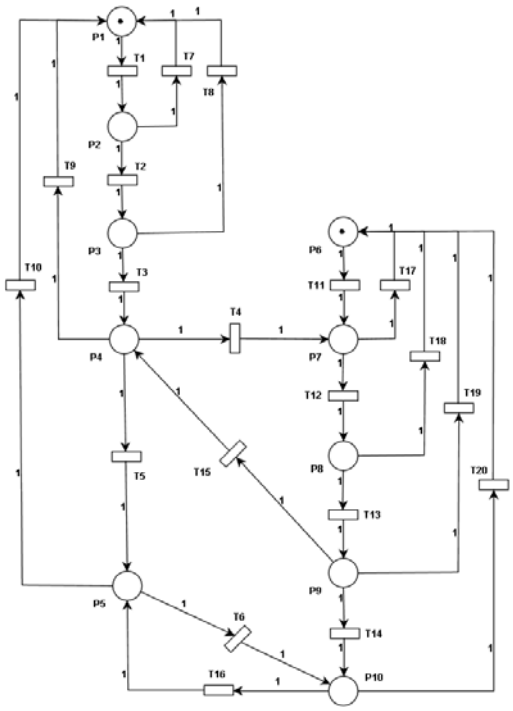


Fig.1 The Petri net model for SpaceWire

Table.1 The definition of Place

Place	Definition
P1	sender ErrorReset
P2	sender ErrorWait
P3	sender Ready
P4	sender Started
P5	sender Connecting

P6	receiver ErrorReset
P7	receiver ErrorWait
P8	receiver Ready
P9	receiver Started
P10	receiver Connecting

Table.2 The definition of transition

Transition	Definition
T1	sender 6.4us delay
T2	sender 12.8us delay
T3	sender LinkEnabled signal valid
T4	sender sends NULL signal
T5	sender creates GotNull signal
T6	sender sends data
T7	link problem in sender Waiting state
T8	link problem in sender Preparing state
T9	link problem in sender Starting state
T10	sender overtime or link problem
T11	receiver 6.4us delay
T12	receiver 12.8us delay
T13	receiver LinkEnabled signal valid
T14	signal receiver creates GotNull signal
T15	receiver sends NULL
T16	receiver sends data
T17	link problem in receiver Waiting state
T18	link problem in receiver Preparing state
T19	link problem in receiver Starting state
T20	receiver overtime or link problem

3. Verification of the Stochastic Petri net Model of SpaceWire

The verification on the SpaceWire protocol is based on the analysis on the Petri net model. From the perspective of Network theory, a network mainly contains the key properties such as boundedness, activity, reachability, and invariability.

Reachability means all the states in the Petri net model are reachable. The reachability analysis is used to check if all the states and their expected behavior meet the requirement of the protocol. Usually, the behavior includes deadlock, unexpected sending/receiving, transition ability and the boundedness of the number of token. The SpaceWire fault mechanism can verify through the Petri network state equation which can analyze state under specific transition.

3.1 The Analysis Based on Reachability Graph

The reachability graph is an important Petri net based analysis technology. The reachability analysis starts from an initial global marker, and creates the branches based on every transition.

By this analysis, we obtain the reachability graph and state space analysis on Petri net shown in Fig.2 and Fig.3.

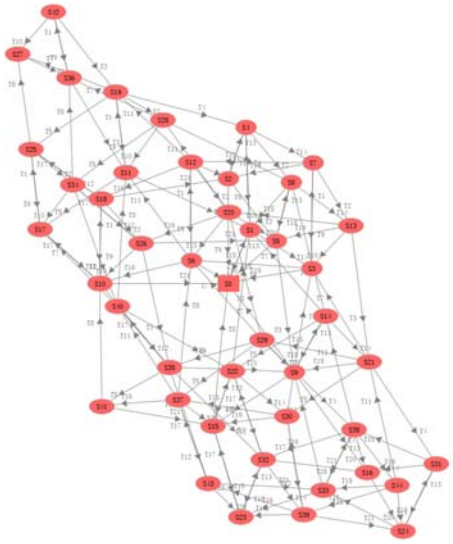


Fig.2 The reachability graph of Petri net model.

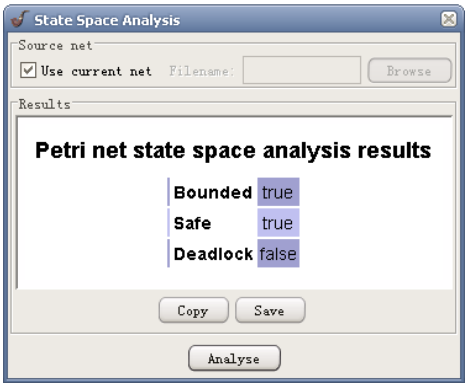


Fig.3 The state space analysis on Petri net model.

(1) Boundedness. The number of token in reachability tree is limited within two, therefore the protocol is bounded.

(2)Activity. Every transition is activated at least once, and therefore there is not the transition that is not active. In the reachability tree, every marker owns its predecessor that can be activated. For a reachable set $R(M_0)$, every marker M' has a transition rout from root M_0 to M', namely $\exists\beta : M_0[\beta > M'$. Based on the definition of activity, we know the network is active, and the deadlock never occurs.

(3) Reachability and Integrity: There is not redundant cycle in the reachable tree, and therefore the overall initialized protocol of communication is reachable.

3.2 The Specified State Analysis on SpaceWire

Based on Fig.1 and the Petri net theory, we can obtain the correlation matrix C of the Petri net. The matrix element is

$$C(P_i, T_j) = W(T_j, P_i) - W(P_i, T_j)$$

$$C=\begin{bmatrix}
-1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & -1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & -1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & -1 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & -1
\end{bmatrix}$$

Based on the system state formula $M_k = M_0 + CU$ of Petri net (M0is initial state, and U is corresponding transition series), we obtain the conclusion:

(1) All marked state reached by the dynamic execution of the model has activated transition and their predecessors. This means starting from the initial state M0（1，0，0，0，0，1，0，0，0，0）, role in the activation sequence δ, the model can return its initial state. Therefore, the protocol is reachable.

(2)If there occurs problem at any working state, such as link problem and overtime problem, the model can always return initial state. Therefore, the protocol has fault tolerance ability. For instance, in state M4（0，1，0，0，0，0，1，0，0，0）, there is link problem, namely the transition T9 occurs, based on state formula we can get the predecessor of M4 is M0（1，0，0，0，0，1，0，0，0，0）, which shows the system returns to initial state.

4. Conclusion

In this paper, the key properties of SpaceWire Bus protocol and stochastic Petri net are primarily studied. Aiming at the drawback of the traditional analysis on SpaceWire Bus protocol, we build a formal model for the running process of this protocol using stochastic Petri net. The model is then simulated with the Petri net analysis tool ‘Pipe’, and a reachability tree is obtained. Finally, by analyzing the reachability tree, the properties of the model are verified so that the fault tolerance mechanism of SpaceWire Bus protocol is confirmed correctly.

Stochastic Petri net shows a big advantage on analyzing SpaceWire Bus protocol, but it can’t describe the protocol sufficiently and consequently is lack of stable simulation accuracy. Therefore, we will focus on these two problems in the future study.

5. References

[1] Zhao Jianli, Shang Ruiqiang, Zhao Linliang. Petri Net Model of Multiplex Network Management Protocol and Its Verification [J]. Journal of System Simulation. 2005,

[2] Lars Miehael Kristensen. Application of Coloured Petri Net System Development ACPN 2003.LNCS30.pp.626 — 685，

[3] Kurt Jensen. An Introduction to the Practical Use of Coloured Petri Nets[M] .Department of Computer Science，University of Aarhus，2000

[4] Peter Shanmes. CCSDS and NASA Standards for Satellite Control Network Interoperability

[5] ECSS, SpaceWire - Links, nodes, routers and networks (ECSS-E-50-12A)

[6] ESA. ECSS-E-ST-50-52C, 2010

Standardisation

DC-BALANCED CHARACTER ENCODING FOR SPACEWIRE

Session: Standardisation

Long Paper

Clifford E. Kimmery

Honeywell International, Clearwater, FL

E-mail: clifford.kimmery@honeywell.com

ABSTRACT

Standard SpaceWire has limited support for applications requiring galvanic isolation between link endpoints. The limitation is derived from the combination of the low common-mode tolerance of ANSI/TIA/EIA-644 LVDS devices and the unbalanced character-level encoding method established by ECSS-E-ST-50-12C, Clause 7 [1].

This paper describes the search for a practical alternative character-level encoding method capable of supporting galvanic isolation using ANSI/TIA/EIA-644 LVDS devices and conventional Alternating Current (AC)-coupling circuits. Other goals of the research were to maintain the clock recovery benefits of Data-Strobe encoding, provide error detection comparable to the standard SpaceWire parity check and minimize the impact to link bandwidth efficiency.

The result is a class of codes that simultaneously Direct Current (DC)-balance both the Data and Strobe bit streams while maintaining the clock recovery behavior of Data-Strobe encoding. Class members are differentiated by the code size and the effort needed to minimize low-frequency content. Members of the class with a larger code size have a direct impact on encoding overhead (decrease in link bandwidth efficiency) while members with a smaller code size have an inverse impact on algorithm complexity (running disparity tracking, etc.). Several examples of the class are described: some have large code size (12-bits to 16-bits) that reduces bandwidth efficiency significantly relative to standard SpaceWire, some have simple encoding methods (12-bits and 16-bits) and others have smaller code size (10-bits and 11-bits) or complex encoding methods (10-bits to 15-bits, excluding 12-bits) and bandwidth efficiency closer to that of standard SpaceWire (the 10-bit example is within 5%).

In this paper, the term character is used as defined by the SpaceWire standard and includes data characters and control characters. The term code is defined as a binary value used to represent a character when transmitted on the SpaceWire link. In standard SpaceWire, a character and the corresponding code are identical.

1 SPACEWIRE CHARACTER ENCODING BACKGROUND

SpaceWire character-level encoding starts with Non-Return-to-Zero (NRZ) encoded ten-bit characters serialized as the Data signal. The Strobe signal is generated from the Data signal by Exclusive OR (XOR) with an alternating binary one-zero pattern of identical length (see Figure 1). The alternating one-zero pattern represents a one-half-rate clock with transitions corresponding to the bit intervals of the Data signal (commonly known as a Double-Data Rate (DDR) clock).

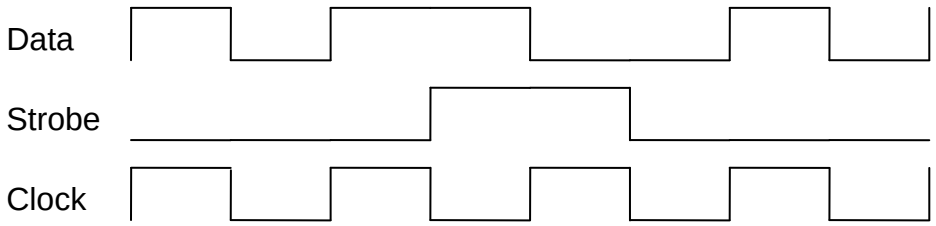


Figure 1 – Data-Strobe Encoding Waveforms

Because standard SpaceWire encoding uses raw binary values to form NRZ-encoded characters for the Data signal, the degree of DC balancing achieved is determined by the character sequence transmitted. As can be observed from Figure 1, introducing a balanced bit stream for the Data signal doesn't automatically create a balanced bit stream for the corresponding Strobe signal.

2 DESIRED PROPERTIES OF DC-BALANCED CHARACTER ENCODING

The search for DC-Balanced character encoding originated from concerns about the limited common-mode tolerance of standard LVDS signaling technology. Many applications migrating to SpaceWire from alternate communications protocols (e.g. MIL-STD-1553) provide much greater tolerance for long-term and transient differences between ground references.

Several goals were established (see Table 1). The primary goals were considered mandatory while the secondary goals were a factor in ranking alternatives.

Category	Goal	Note
Primary	Allow galvanic isolation of a SpaceWire link using typical AC-coupling methods	Requires intensive validation effort
	Maintain the standard SpaceWire electrical interface and clock recovery mechanism	
	Contain changes within the character encoding level of the SpaceWire standard	
Secondary	Provide comparable error-detection capability	
	Maximize the bandwidth efficiency	
	Minimize the encoding/decoding complexity	Lowest priority

Table 1 – Goals

2.1 FACTORS CONSIDERED IN ESTABLISHING GOALS

The use of typical capacitive or inductive AC-coupling methods for galvanic isolation depends on a transmitted bit stream with an average value of 0V (assume binary 1

corresponds to +1V and binary 0 corresponds to -1V electrical levels). The average must be maintained over a sliding time window with a length dependent on the characteristics of the communications channel and the sensitivity of the receiver.

The DC-balanced encoding must be applied to both the Data and Strobe signals so that each can be AC-coupled in the same manner. The encoding method attempts to maintain the same average DC-balance on both signals simultaneously to maximize the clock recovery opportunity of the standard SpaceWire Data-Strobe receiver.

By limiting changes to the SpaceWire character encoding level, all other aspects of the standard are unaffected. The well-proven SpaceWire physical and signal levels, the link protocol (exchange level), and the packet and network levels continue to function as with standard SpaceWire.

2.2 SpaceWire Factors Impacting Goal Achievement

Because SpaceWire link traffic primarily consists of data characters and flow-control-token (FCT) characters, the size of encoded characters dominates when evaluating link bandwidth efficiency. Any change to the encoded size of a character has a corresponding impact on link overhead. Because the FCT traffic associated with one link direction is overhead to the other link direction, any increase in FCT character code size has a greater impact than increases in the other control character code sizes.

The standard SpaceWire error detection mechanism using a parity bit for each encoded character consumes approximately 10% of available link bandwidth. Because the SpaceWire standard establishes a lagging-parity mechanism (the parity bit associated with one character is transmitted as the first bit of the following character), the encoded value of each standard SpaceWire character is dependent on the parity of the preceding character, essentially defining two representations of each character. These factors make replacing the parity-based error detection mechanism highly desirable. Because transmission error detection is required by the SpaceWire exchange level, any replacement error detection mechanism must provide an comparable capability.

The standard SpaceWire NRZ-based encoding is very simple to implement. Any alternate encoding method is likely to be more complex.

3 DC-Balanced Character Encoding Background

The transmitted bit stream created by DC-balancing must have an average value of 0V over a limited time interval to minimize undesirable biases in the galvanic isolation circuits. The average value over small time intervals is a function of the transition density and run-length of the encoded values composing the bit stream.

The transition density capability of 8b10b encoding was used to establish the initial transition density and run-length benchmarks. The 8b10b encoding scheme guarantees a transition-rich data stream so that the receiving device can perform clock recovery on the incoming serial data. Transition rich means that for every 20 successive bits transferred, the difference in the number of ones and the number of zeros cannot be more than two, and there cannot be more than five ones or five zeros in a row [2]. Note that the difference in the number of ones and zeroes is termed the disparity.

Note that the 8b10b encoding scheme cannot be used directly because of the need to DC-balance both the Data and Strobe signals simultaneously. The Strobe bit stream generated from an 8b10b encoded Data bit stream does not have adequate DC-balancing characteristics.

The DC-Balanced Data character encoding must be chosen so that the resulting Strobe bit stream has characteristics comparable to the Data bit stream. By selecting Data character encodings based on the transition density and run-length of both resulting bit streams, the desired average signaling value can be achieved for both.

4 DEVELOPMENT OF DC-BALANCED CHARACTER ENCODING

Evaluating the feasibility of a DC-balanced character encoding mechanism begins by determining the number of binary values with the appropriate properties for each potential code size. The first step is to create a list of the binary value pairs for additional evaluation by exclusive-ORing each binary value with an alternating one-zero pattern (clock) to produce the other binary value of the pair.

Because high quality DC-balancing is a function of the transition density and run-length of the encoded values, a number of characteristics must be determined for each candidate binary value. The benchmark criteria established from 8b10b encoding require that any twenty-bit sequence have a disparity of two or less and that the number of consecutive same-value bits be five or less.

The run-length benchmark criterion can be translated in a straightforward manner by establishing that the number of consecutive same-value bits within any two successive encoded characters should be five or less (including the boundary between the two characters). The character boundary run-length can be constrained using one of three methods:

1. By establishing that the leading and trailing run-length must be two or less (corresponding to a boundary-crossing run-length limit of four or less),
2. By establishing that the leading run-length must be three or less and the trailing run-length must be two or less,
3. By establishing that the leading run-length must be two or less and the trailing run-length must be three or less.

Since method 1 is stricter than the 8b10b benchmark, allowing either method 2 or method 3 is preferred. Evaluating each set of candidate binary values using both method 2 and method 3 and choosing based on the best result is desirable.

Translating the disparity benchmark criterion is nontrivial, but a somewhat weaker approximation can be achieved by establishing that two successive codes should have a combined disparity of two or less. Since any character can occur in combination with any other character (including itself), a number of distinct disparity combinations must be considered as shown in Table 2. Note that a binary value with an even number of bits has even disparity 0, 2, 4, etc. while a binary value with an odd number of bits has odd disparity 1, 3, 5, etc.

Sequence	Description	Combination
Even length values		
Zero-Zero	Each code has zero disparity	Zero disparity
Zero-Two	The first code has zero disparity and the second code has two disparity	Two disparity
Two-Zero	The first code has two disparity and the second code has zero disparity	Two disparity
Two-Two	Each code has two disparity	Zero disparity
Odd length values		
One-One	Each code has one disparity	Zero disparity
One-Three	The first code has one disparity and the second code has three disparity	Two disparity
Three-One	The first code has three disparity and the second code has one disparity	Two disparity
Three-Three	Each code has three disparity	Zero disparity
Note that to achieve the combined result for nonzero disparity sequences, each code that has nonzero disparity must have at least one alternate code with the opposite (negative) disparity.		

Table 2 – Disparity Combinations

4.1 EVALUATION OF CODE VALUE CANDIDATES

The evaluation was performed for the code lengths identified in Table 4 to determine the feasibility of each length. For each code length, a table was generated containing 2^{length} values. For each value, the table included the corresponding code formed by exclusive-OR with the alternating one-zero pattern and a variety of metrics for both codes of the pair.

The metrics computed for each code were:

1. The total number of one bits
2. The maximum number of consecutive one bits
3. The number of consecutive leading one bits
4. The number of consecutive trailing one bits
5. The total number of zero bits
6. The maximum number of consecutive zero bits
7. The number of consecutive leading zero bits
8. The number of consecutive trailing zero bits
9. The disparity (the total number of one bits minus the total number of zero bits)

The metrics for each code pair were used to determine whether that pair should be selected for membership in a candidate DC-Balanced code set. The parameters used to determine the members of the code set were:

1. Maximum disparity (the disparity of each code in the code set must be no greater than the maximum disparity parameter)
2. Maximum run-length (the number of consecutive one bits or zero bits must be no greater than the maximum run-length parameter)

3. Maximum leading run-length (the number of consecutive leading one bits or zero bits must be no greater than the maximum leading run-length parameter)
4. Maximum trailing run-length (the number of consecutive trailing one bits or zero bits must be no greater than the maximum trailing run-length parameter)

Note that the maximum leading and maximum trailing run-length parameters provide fine-grained selection within the set of codes that are selected by the maximum run-length parameter. This was done to address the code boundary run-length issue discussed previously.

Based on the 8b10b benchmark criteria, the initial evaluation of each code length was performed using the parameter values in Table 3. Note that the 8b10b criterion that any twenty-bit sequence must have a disparity of two or less has been simplified to require that any two successive codes must have a disparity of two or less.

Maximum Disparity	Maximum Run-Length	Maximum Leading Run-Length	Maximum Trailing Run-Length
Even length values			
2	5	2	3
Odd length values			
1	5	2	3

Table 3 – Initial Evaluation Parameter Values

4.2 CODE SET SIZE

The SpaceWire character set consists of 256 data characters and 4 control characters, so at least 260 distinct code pairs are needed to encode the complete character set. A set of code pairs with a nonzero maximum disparity characteristic must include sufficient pairs to allow multiple encodings per character.

If a set of code pairs has a non-zero maximum disparity, each pair in the set must be matched with another pair in the set with the opposite disparity to allow representation of the same SpaceWire character with either pair. Note that any code pairs in the set that have zero maximum disparity can represent a SpaceWire character uniquely. If none of the code pairs in the set have zero maximum disparity, a minimum of 520 distinct code pairs is needed to encode the complete SpaceWire character set.

Since odd-length codes inherently have non-zero disparity, the minimum of 520 distinct code pairs always applies. As an additional complication, the nature of Data-Strobe encoding causes two identical odd-length Data codes in succession to produce different Strobe codes. This makes selection of an appropriate set of odd-length codes more difficult, so it is convenient to require that successive identical SpaceWire characters be encoded to different odd-length codes. This increases the minimum number of distinct odd-length code pairs needed to encode the complete SpaceWire character set to 1,040.

To summarize the set size criteria:

1. If each code pair in a set has zero disparity, a minimum of 260 pairs are needed to encode the complete SpaceWire character set.

2. If each code pair in a set of even-length code pairs has a nonzero maximum disparity, a minimum of 520 pairs are needed to encode the complete SpaceWire character set.
3. Each code pair in set of odd-length code pairs must contain a minimum of 1,040 pairs to encode the complete SpaceWire character set.

4.3 CODE SET SELECTION RESULTS

Based on the initial evaluation, the code lengths of 14-bits, 15-bits and 16-bits easily produced code sets of sufficient size that met the benchmark criteria. Of the remaining code lengths evaluated, the code sets of sufficient size produced for 12-bits and 13-bits met the disparity criteria, but failed the maximum run-length criteria. The 10-bit and 11-bit code sets of sufficient size failed both the disparity criteria and the maximum run-length criteria. Table 4 shows the smallest set of sufficient size to encode the complete SpaceWire character set and the corresponding set metrics for each code length evaluated.

Bits	Maximum Disparity	Maximum Run-Length	Set Size
Even length codes require a set size of 260 (disparity 0) or 520			
10	4	7	552
12	0	6	284
14	2	4	1144
16	0	4	260
Odd length codes require a set size of 1,040			
11	3	8	1048
13	1	7	1040
15	1	4	1188
Note: the Maximum Run-Length is the greater of the Run-Length and the sum of the Leading Run-Length and the Trailing Run-Length Note: the Set Size is the number of codes that met the corresponding evaluation criteria. The codes to be used are chosen from the full set as desired.			

Table 4 – Code Result by Length

4.4 EFFECTS OF CODE LENGTH ON BANDWIDTH EFFICIENCY

Because standard SpaceWire characters are encoded as values with differing lengths depending upon function, any code length greater than the standard length can have a significant impact on link bandwidth efficiency. Table 5 shows the efficiency of each candidate code length for the various SpaceWire character types. Note that each character type is assumed to use codes of the full candidate length rather than the variable lengths defined for standard SpaceWire.

SpaceWire Character Type		Standard SpaceWire	Candidate Length						
			10-bit	11-bit	12-bit	13-bit	14-bit	15-bit	16-bit
Data	10 Bits	100.0%	100.0%	90.9%	83.3%	76.9%	71.4%	66.7%	62.5%
FCT	4 Bits	100.0%	40.0%	36.4%	33.3%	30.8%	28.6%	26.7%	25.0%
EOP/ EEP	4 Bits	100.0%	40.0%	36.4%	33.3%	30.8%	28.6%	26.7%	25.0%
Time Code	14 Bits	100.0%	70.0%	63.6%	58.3%	53.8%	50.0%	46.7%	43.8%
Null	8 Bits	100.0%	80.0%	72.7%	66.7%	61.5%	57.1%	53.3%	50.0%
Note that the SpaceWire Time Code consists of a control character followed by a data character.									

Table 5 – Code Length Efficiency by Character Type

Overall SpaceWire link efficiency is dynamically determined by the mix of SpaceWire characters transmitted. Fortunately, data characters and FCT characters dominate link traffic; the end-of-packet characters are relatively rare, time code characters are rarer still and null characters are only used when necessary to keep the link active. Table 6 shows the link efficiency of each candidate length relative to standard SpaceWire for three representative cases of SpaceWire traffic where the link is fully utilized in both directions with packets of the same size. The table clearly shows that the overall link efficiency is significantly impacted by code length when the majority of link traffic is small packets. Since standard SpaceWire data characters have 10-bit length, the 10-bit code length case shows the impact of the increased FCT code size on link efficiency.

SpaceWire Character Mix Data/Flow/EOP	Candidate Length						
	10-bit	11-bit	12-bit	13-bit	14-bit	15-bit	16-bit
10 byte packets 89.3%/7.1%/3.6%	86%	78%	72%	66%	62%	57%	54%
100 byte packets 94.7%/4.9%/0.4%	93%	84%	77%	71%	66%	62%	58%
1000 byte packets 95.2%/4.8%/0.0%	93%	85%	78%	72%	67%	62%	58%
The left column contains the percentages of transmitted bits for data characters, flow-control-token characters and end-of-packet characters respectively.							

Table 6 – Code Length Effects on Bandwidth Efficiency

4.5 MIXING CODE LENGTHS TO IMPROVE BANDWIDTH EFFICIENCY

Standard SpaceWire minimizes link overhead very effectively by using different code lengths. For small packets, the 4-bit code length used for FCT characters improves bandwidth utilization by 14% over the utilization when using a 10-bit FCT code. DC-Balanced character encoding can reduce the impact of increased code length by selectively using a shorter code length for control characters.

Standard SpaceWire character encoding distinguishes between the various code lengths by making all codes unique in the first bits transmitted/received. The four standard SpaceWire control characters are each defined as a 4-bit code that does not match the first four bits of any other SpaceWire code. The same technique may be used to choose a small number of members of a DC-Balanced code set as candidates for use as control characters.

4.6 Error-Detection Capability

The DC-Balanced codes described have intrinsic characteristics that make transmission error detection straightforward. The error response defined by standard SpaceWire is unchanged.

Standard SpaceWire adds a parity bit to each encoded character to detect transmission bit errors. The error response is to disconnect the link, report the error and attempt to reconnect the link (the same approach is used to recover from all link errors).

The decoding mechanism that translates DC-Balanced codes to the equivalent SpaceWire character has inherent error detection capability since an unrecognized code is considered an error. A transmission error occurring in a DC-Balanced code must convert that code to a different valid code for the error to be undetectable. The members of a DC-Balanced code set can be selected to have sufficient Hamming distance to prevent many transmission errors from being undetectable.

In addition, either the Data code or the Strobe code (or both) can be decoded to the equivalent SpaceWire character. In cases where an adequate Hamming distance is not achievable, the Data code and the Strobe code can be independently decoded and then compared to detect most transmission errors. The two mechanisms can clearly be combined to provide very robust transmission error detection.

5 Results and Conclusions

Of the DC-Balanced code sets that fully met the 8b10b benchmark criteria, the 14-bit length code set is the most bandwidth efficient. Unfortunately, the bandwidth efficiency is at best 66% that of standard SpaceWire. All of the other fully qualified code sets are less efficient than the 14-bit code set.

5.1 Relaxing the Benchmark Criteria

Relaxing the benchmark criteria allows use of DC-Balanced code sets with greater bandwidth efficiency. The effects of relaxing the benchmark criteria on link performance must be determined by signal integrity analysis and experimentation.

Although the 12-bit length DC-Balanced code set misses the run-length benchmark criterion by 20%, it has the advantage of zero-disparity implementation simplicity. The approximately 75% bandwidth efficiency relative to standard SpaceWire can be improved to approximately 80% by choosing an FCT code with at most 6-bit length.

5.2 Most Efficient, Complex Implementation

The much better bandwidth efficiency of the 10-bit DC-Balanced code set makes further relaxation of the benchmark criteria worth consideration. Taking advantage of

a 6-bit FCT code length allows the 10-bit DC-Balanced code set to achieve bandwidth efficiency within 5% of standard SpaceWire.

The 10-bit DC-Balanced code set misses the disparity criterion significantly (the running disparity achievable by the 10-bit code set varies based on the tracking method used). As with 8b10b encoding, DC-Balanced encoding must manage the running disparity to limit the difference in the number of ones and zeroes in the transmitted bit stream. Unlike 8b10b encoding, DC-Balanced encoding must track the running disparity for both SpaceWire signals (Data and Strobe) simultaneously. The goal is to minimize the running disparity of each signal without minimizing one at the expense of the other. Modeling has shown that the 10-bit code set running disparity can be limited to eight or less for a bit stream composed of a random code sequence.

The 10-bit DC-Balanced code set misses the run-length criterion by 40% with no mitigation method available. Clearly, the ability to take advantage of the bandwidth efficiency of the 10-bit code set depends on further analysis regarding the link performance.

5.3 SUMMARY

This paper has shown that an alternative character-level encoding method that is capable of supporting galvanic isolation of SpaceWire links using conventional AC-coupling circuits is possible. The alternative method limits changes to the character-level and provides transmission error detection comparable to the standard SpaceWire parity check. The major tradeoffs to be considered are the impacts to SpaceWire link bandwidth efficiency, encoding/decoding implementation complexity and frequency performance over AC-coupled circuits.

6 REFERENCES

1. ECSS, “Space engineering - SpaceWire - Links, nodes, routers and networks”, ECSS-E-ST-50-12C, 31 July 2008, pages 52-56, http://spacewire.esa.int/content/Standard/ECSS-E50-12A.php.

2. Alex Goldhammer and John Ayer Jr., “Understanding Performance of PCI Express Systems”, September 4, 2008, page 2, http://www.xilinx.com/support/documentation/white_papers/wp350.pdf.

STANDARDISATION OF SPACEWIRE SOFTWARE APIS

Session: SpaceWire Standardisation

Short Paper

Stuart Mills, Alex Mason

STAR-Dundee, Dundee University Incubator, James Lindsay Place, Dundee Technopole, Dundee, Scotland, UK

Steve Parkes

University of Dundee, School of Computing, Park Wynd, Dundee, Scotland, UK

Takayuki Yuasa

JAXA/ISAS 3-1-1, Yoshinodai, Sagamihara, Kanagawa, Japan

E-mail: stuart@star-dundee.com, alex@star-dundee.com, sparkes@computing.dundee.ac.uk, yuasa@astro.isas.jaxa.jp

ABSTRACT

As SpaceWire has gained a greater market share in recent years, the number of software products available for SpaceWire-related activities has also grown. Software APIs are provided by test and development equipment manufacturers, flight board manufacturers, chip manufacturers, etc. to control and configure their devices. Each company provides their own API, often with different APIs required for each device from the same company.

The purposes of the SpaceWire standard include reducing system integration costs, promoting compatibility between data-handling equipment and subsystems, and encouraging reuse of data-handling equipment across several different missions. This paper argues that standardisation of software APIs would further these aims, greatly improving compatibility between equipment and encouraging software reuse across missions, thereby reducing development and integration costs.

1 INTRODUCTION

An API, or Application Programming Interface, is the interface provided by a module so that software can interact with that module. In SpaceWire terms, it may be the programming interface used to transmit and receive packets on a SpaceWire device.

STAR-Dundee has considerable experience developing APIs for SpaceWire equipment, beginning with the API for a SpaceWire PCI device before the SpaceWire standard was released, through to our new API system, STAR-System which supports multiple device types and operating systems in a consistent manner [1]. We have also worked with NEC TOSHIBA Space Systems to port our SpaceWire USB API to the Shimafuji Space Cube [2], in order to provide a consistent platform to run our SpaceWire CUBA Software.

Despite the great international collaboration taking place in SpaceWire-related activities, currently there are no standard APIs for interacting with SpaceWire devices. As a result, hardware and software manufactures can provide completely different APIs for each of their products. This is clearly not beneficial to anyone; flight and test software developers must learn a new API for each device or module they work with, while manufacturers may need to develop a new API for each device they release.

It can be argued that there is no alternative to this situation. API implementations may differ depending on their target uses. For example, an API which is to be used on a flight system is unlikely to require the same functionality as an API used to test devices on a network.

Regardless of this, there are a great number of implementations being created to do very similar tasks. This paper looks at the various APIs which may be used in a SpaceWire system, and considers whether standardisation of these APIs would be of benefit to the SpaceWire community.

2 USE OF EXISTING APIS

A number of users who are new to SpaceWire expect to use existing, known APIs to access SpaceWire devices. Many assume that they can use the POSIX Sockets API, based on the Berkeley Sockets API and part of the POSIX standard [3], with STAR-Dundee devices. While we do provide a network interface for our USB devices which enables the use of the Sockets API, we strongly discourage anyone from using this to write their own code.

The reason for this is that the Sockets API cannot directly take advantage of the full benefits of SpaceWire, or provide access to the many test, development and debug features provided in STAR-Dundee devices and APIs. For example, it is not possible to transmit or receive time-codes using the Sockets API directly, or to terminate a packet with an EEP. Even if the Sockets API is used, additional APIs must also be used to configure devices. A further limitation is that the Sockets API is normally used to carry streams of data over TCP, with no regard for end of packet markers. This is normally not what is required by SpaceWire users, who wish to carry raw data over a SpaceWire network, with packet start and end points clearly marked. Note also that there is currently no standard for carrying TCP/IP over SpaceWire, so different implementations may be unable to communicate.

Despite this, there may be an argument for using the POSIX functions in some situations, simply because many developers are familiar with the interface, and this may therefore shorten development and test times. On a flight system, where the software is concerned primarily with transmitting and receiving packets over a device which does not need any configuration (e.g. starting the link, setting the link speed, etc.) the Sockets API may be a suitable solution. The Sockets API could be modified to transmit a single packet in response to a `send()` function call, and to pass up a single packet in response to a `recv()` call.

This would not provide a particularly high performance interface, however. An API designed specifically for transmitting and receiving packets to/from a SpaceWire device is likely to result in better performance and better quality code. Note that this

API may be built on top of the Sockets API, for example when communicating with a device over Ethernet.

3 TYPICAL SPACEWIRE APIS

The APIs required to access a SpaceWire device are not simply limited to transmitting and receiving packets. Support for protocols such as RMAP [4] and the CCSDS Packet Transfer Protocol [5] requires additional APIs, while SpaceWire devices typically provide a number of configuration options which are made available through software. These may include setting the device and/or link speed, configuring routing tables, and starting and stopping links.

Some of the APIs which may be used both in test, development and debug environments and in flight systems are described below.

3.1 PACKET TRANSFER API

The Packet Transfer API is the most important API, and the one that STAR-Dundee users generally have the most exposure to. This is the API that is used to transmit and receive packets, and is also used to open and close connections to the device.

Although it might be assumed that this is quite a simple API, STAR-Dundee's Packet Transfer API, STAR-API, includes a great deal of functionality. For example, a function to transmit a single packet is unlikely to provide very high performance. Instead, the transmit function must allow multiple packets to be submitted, these may be interleaved with time-codes, and some may be terminated with an EEP. Similar functionality is required for receiving traffic items in order.

Although not all of this functionality will be required in a flight system, performance may be of even greater importance. The number of interrupts which are generated when packets are transmitted and/or received will have a huge influence on the overall performance of a system. An API which can cope with multiple packet transmit or receive operations generating a single interrupt will provide better performance and use less resources than one that interrupts on each and every packet.

3.2 REMOTE MEMORY ACCESS PROTOCOL APIS

STAR-Dundee has developed three different RMAP APIs, in order to separate out the diverse functionality that may be required by users developing RMAP applications. At the lowest level is the RMAP Packet API. This provides functions for building each of the RMAP packet types (read commands and replies, write commands and replies, etc.), for interpreting and validating the contents of RMAP packets and extracting the values of fields in the packets.

The functionality provided by the RMAP Packet API was then used to implement the RMAP Initiator and Target modules, each with their own API. These modules provide software implementations of RMAP initiators and targets. Both modules make use of STAR-Dundee's Packet Transfer API to transmit and receive commands and replies.

An API providing functions to configure RMAP targets or initiators implemented in hardware may require a different API, although there are likely to be some functions which are required for both the physical and software implementations.

3.3 Other Protocol APIs

As with the RMAP APIs, other higher layer protocol APIs, such as the CCSDS Packet Transfer Protocol, GOES-R RDDP [6] and SpaceWire-PnP APIs [7], may be split up in to a packet building/validating API, an initiator API, and a target API. The initiator and target APIs may be combined, depending on the nature of the protocol.

Unlike the other protocols discussed above, the CCSDS Packet Transfer Protocol can be used over other networks and buses, and not just over SpaceWire. This means that it may be possible to use existing APIs, modified to support SpaceWire addressing. Similarly, if TCP/IP packets are to be carried over SpaceWire, the Sockets API can be used, as discussed earlier.

3.4 Device Configuration API

One API that may initially appear to be impossible to standardise is the Device Configuration API. This provides the functions that allow the features of a device to be configured, and are likely to be very specific to that device. However, there are a number of features which are common between devices, and the SpaceWire-PnP draft protocol definition has identified some of these features.

STAR-Dundee's Device Configuration API contains a number of functions which are common to all STAR-Dundee devices, such as setting the speed of a link, starting a link, etc. There are then additional functions to provide functionality specific to individual devices. A standardised Device Configuration API could be produced in a similar manner, using the features identified in the PnP definition as a basis.

4 Summary and Conclusions

One of the many advantages of SpaceWire is that it has allowed organisations to reuse equipment and software. But without standardisation of APIs, the benefits of reuse cannot be fully realised. With standard APIs, developers can create software using development equipment like the STAR-Dundee SpaceWire-USB Brick [8] running on consumer operating systems such as Windows or Linux, then migrate their code to flight hardware running real-time operating systems such as RTEMS or VxWorks. Developers can also write code that will work on multiple devices, without providing a "shim" layer which handles the differences between devices. On long term projects, supporting new devices and replacing devices with alternatives, may require no additional code to be written at all.

The disadvantages of standardised APIs are few. One concern may be that a system may not require all the functions provided by the standardised API, and these additional functions might take up precious resources. If the standardised API allows some or all functions to be optional, then this problem is eliminated. This would also allow test and development systems to include additional functions not required in a flight system.

This paper has identified a number of APIs which are typically used in SpaceWire systems. It is clear that standardisation of these APIs would be of great benefit to the SpaceWire community, and we urge the community to work towards this. The effort required to reach a consensus would be minor when compared to the potential savings that could be achieved.

5 REFERENCES

1. STAR-Dundee, http://star-dundee.com/products.php, STAR-Dundee SpaceWire Products, STAR-Dundee Website.

2. Shimafuji, http://www.shimafuji.co.jp/product/spacecube01.html, Space Cube, Shimafuji Website.

3. IEEE, "Single UNIX Specification", IEEE Std 1003.1-2008, Version 4, Institute of Electrical and Electronics Engineers, September 2008.

4. ECSS, "SpaceWire – Remote Memory Access Protocol", Standard ECSS-E-ST-50-52C, Issue 1, European Cooperation for Space Standardization, February 2010.

5. CCSDS, "Space Packet Protocol", CCSDS 133.0-B-1, Blue Book, Issue 1, Consultative Committee for Space Data Systems, September 2003.

6. GSFC, "Geostationary Operational Environmental Satellite (GOES), GOES-R Series, GOES-R Reliable Data Delivery Protocol (GRDDP)", 417-R-RPT-0050, Baseline Version 2.1, NASA Goddard Space Flight Center, July 2005.

7. P. Mendham, A. Ferrer Florit, S.M. Parkes, "SpaceWire-PnP Protocol Definition", Issue 2.1, University of Dundee, September 2009.

8. STAR-Dundee, http://star-dundee.com/products/SpaceWire-USB%20Brick.php, SpaceWire-USB Brick, STAR-Dundee Website.

Components 2

NGMP – QUAD-CORE NEXT GENERATION MICROPROCESSOR WITH ON-CHIP SPACEWIRE ROUTER

Session: SpaceWire Components

Long Paper

Jan Andersson, Marko Isomäki, Sandi Habinc, Jiri Gaisler

Aeroflex Gaisler AB, Kungsgatan 12, SE-411 19 Göteborg, Sweden

Luca Fossati, Roland Weigand

European Space Agency, Keplerlaan 1 - PO Box 299, 2220AG Noordwjik ZH, The Netherlands

E-mail: jan@gaisler.com, marko@gaisler.com, sandi@gaisler.com, jiri@gaisler.com, luca.fossati@esa.int, roland.weigand@esa.int

ABSTRACT

The Next Generation Microprocessor (NGMP) is a quad-processor system-on-chip currently being developed by Aeroflex Gaisler in a n activity commissioned and funded by the European Space Agency. Compared to earlier generations of European space processors, the NGMP design provides higher performance and places greater emphasis on support for both symmetric and asymmetric multiprocessing. Another significant difference is the introduction of a SpaceWire router instead of multiple node cores which have typically been used in other System-on-Chip devices. In addition to this the system contains a dedicated RMAP core used for debug access.

1 BACKGROUND

The LEON project was started by the European Space Agency in late 1997 to study and develop a high-performance processor to be used in European space projects.

The LEON family includes the first LEON1 VHSIC Hardware Description Language (VHDL) design that was used in the LEONExpress test chip developed in 0.25 µm technology to prove the fault tolerance concept. The second LEON2 VHDL design was used in the processor device AT697 from Atmel (F) and various system-on-chip devices. These two LEON implementations were developed by ESA. Gaisler Research, now Aeroflex Gaisler, developed the third LEON3 design that is used in a number of avionics systems and also in the commercial sector. Following the LEON3 processor Aeroflex Gaisler developed the LEON4 processor that has improved performance thanks to wider internal buses and a modified pipeline.

Following the development of the TSC695 (ERC32) and AT697 components in 0.5 and 0.18 µm technologies respectively, ESA has initiated the NGMP activity targeting a European Deep Sub-Micron (DSM) technology in order to meet increasing requirements on performance and to ensure the supply of European space processors. Aeroflex Gaisler was selected to develop the NGMP system that will be centered around the LEON4FT processor.

2 SYSTEM OVERVIEW

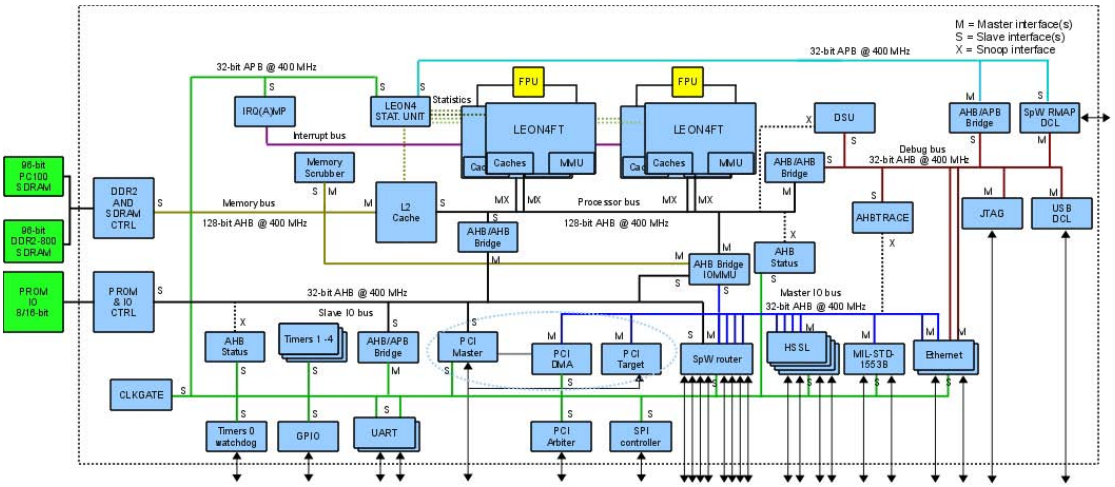


Figure 1 shows an overview of the architecture.

The system consists of five Advanced High-performance Buses (AHB); one 128-bit Processor bus, one 128-bit Memory bus, two 32-bit I/O buses and one 32-bit Debug bus. The Processor bus houses the four LEON4FT processor cores connected to a shared Level-2 (L2) cache. The Memory bus is located between the L2 cache and the main external memory interfaces, DDR2 SDRAM and PC100 SDRAM, and it is also connected to a hardware memory scrubber. Only one of the main memory interfaces (DDR2-4800 or PC100 SDRAM) can be used at a time and can provide up to 2 GiB of external memory. As an alternative to a large on-chip memory, part of the L2 cache can be turned into on-chip memory by cache-way disabling.

The two separate I/O buses house all the peripheral cores. All slave interfaces have been placed on one bus (Slave I/O bus) and all master/DMA interfaces have been placed on the other bus (Master I/O bus). The Master I/O bus connects to the Processor bus, or alternatively to the memory bus, thus bypassing the L2 cache, via an AHB bridge that provides access restriction and address translation (IOMMU) functionality. The two I/O buses include all peripheral units such as timers, interrupt controllers, UARTs, general purpose I/O port, PROM/IO controller, PCI master/target, High-speed Serial Links, Ethernet MACs, 1553, SPI and SpaceWire interfaces. All I/O master units in the system contain dedicated DMA engines and are controlled by descriptors located in main memory that are set up by the processors. Reception of, for instance, Ethernet and SpaceWire packets will not increase CPU load. The cores will buffer incoming packets and write them to main memory without processor intervention.

The fifth bus, a dedicated 32-bit Debug bus, connects a LEON4FT Debug Support Unit (DSU), PCI and AHB trace buffers, and several debug communication links. The Debug bus with the debug communication links allows for non-intrusive debugging through the DSU and, as the Debug bus is not placed behind an AHB bridge with access restriction functionality, has direct access to the complete system.

The target frequency of the NGMP is 400 MHz, but depends ultimately on the ASIC technology.

2.1 LEON4FT Microprocessor and L2 Cache

The LEON4FT processor is the latest processor in the LEON series. LEON4FT is a 32-bit processor core conforming to the IEEE-1754 (SPARC V8) architecture. It is designed for embedded applications, combining high performance with low complexity and low power consumption. LEON4 improvements over the LEON3 processor include: Branch prediction, 64-bit pipeline with single cycle load/store and 128-bit wide L1 cache.

The LEON4FT processor connects to an AMBA AHB bus with a 128-bit data width. This leads to a 4x performance increase, compared to LEON3, when performing cache line fills. Single cycle load and store instructions increase performance and also take advantage of the wider AHB bus.

Static ("always taken") branch prediction has shown to give an overall performance increase of 10%. The LEON4FT also has support for the SPARC V9 compare and swap (CAS) instruction that improves lock handling and performance.

The L2 cache acts as a high-speed buffer between external memory and the AHB bus. An important factor to high processor performance and good SMP scaling is high memory bandwidth coupled with low latency. A 128-bit wide bus is therefore used to connect the L2 cache with the external memory controller. This will allow 32 bytes to be read in two clocks, not counting initial memory latency. The L2 cache features a configurable replacement algorithm with least-recently-used (LRU) replacement as the default. It is a 256 KiB (baseline size, actual size limited by target technology) copy-back cache with BCH Error Correcting Code (ECC). One or more cache ways can be locked to be used as fault-tolerant on-chip ("scratchpad") memory.

2.2 Main Memory Interface

The baseline decision for the main memory interface is to support 96-bit (64 data bits and up to 32 check bits) DDR2-800 and PC100 SDRAM on shared pins. However the selection between DDR2 and DDR(1) SDRAM should be regarded as open. The flight models of the NGMP are scheduled several years into the future. At that time there may be additional information available regarding memory device availability. Availability of I/O standards on the target technology may also impact the final decision.

The data width of the main memory interface is dynamically configurable between 32 and 64 data bits (plus check bits), allowing for NGMP systems with a reduced width of the memory interface to support packages with low pin count, and also use in systems with fewer components. The PC100 SDRAM interface will be able to run at the same or one fourth of the system frequency. The DDR2 interface will be run at the same or twice the system frequency. The clock scaling factor between the memory interfaces and the rest of the system, and also the data width of the memory interface, is selectable via external signals.

External dynamic memory is in the normal case protected with a Reed-Solomon code that uses 32 check bits to protect the 64 data bits (or 16 check bits to protect 32 data bits). To further improve resilience against permanent memory errors, the system supports an on-line ECC code switch where the number of check bits is halved and a

faulty memory is removed from system use. The scheme allows any byte, in the check bit or data vectors, to be switched away while the system keeps operating.

2.3 I/O INTERFACES

A set of standard peripherals required for operating system support is included on-chip. These include support for simple memory mapped I/O devices, two basic console UARTs and one 16-bit I/O port for external interrupts and simple control.

The high-speed interfaces that are intended to be used in flight are a twelve port SpaceWire router, two 10/100/1000 Mbit Ethernet links, four 6.25 Gbit/s High-Speed Serial Links, 1553 and SPI interfaces and one 32-bit PCI 2.3 master/target interface.

2.4 PCI INTERFACE

The currently used AT697 processor and several LEON3FT designs have a 32-bit PCI interface. This makes a 32-bit PCI bus a suitable candidate for the local backplane, since it will make the NGMP backward compatible with existing backplanes. The downside with the PCI interface is that it requires many I/O pins and is relatively slow. However, selecting a more modern interface, such as PCI express would increase demands on companion chips. This could prevent the use of many types of currently available programmable logic devices as companion devices.

2.5 SPACEWIRE ROUTER

The SpaceWire router is based on the GRSPWROUTER IP core which is the common building block for all Aeroflex Gaisler router designs. In NGMP it is configured with eight external SpaceWire ports and four internal AMBA ports connecting to the internal Master I/O bus.

The SpaceWire router allows the NGMP to act both passively and actively in a SpaceWire network. The router in the NGMP can act separately from the rest of the system-on-chip-design or the NGMP system can connect to the SpaceWire network through the four AMBA ports available on the router. The interface of the AMBA ports is identical to the interface of Aeroflex Gaisler's GRSPW2 SpaceWire core, allowing re-use of driver software. This also allows the router to be used by the host system as a normal node the only difference being that a leading physical address is required specifying the output port (at the same time adding flexibility). An AHB slave interface is also available connected to the slave I/O busing providing direct access to the configuration port.

Preliminary results for the 400 MHz target frequency show that, using only internal routing (data not going out on external SpaceWire ports), the architecture is able to sustain a data throughput of 1.5 Gbit/s per SpaceWire AMBA port. In a scenario where the two full-duplex Ethernet links and all SpaceWire AMBA ports are run flat out, the sustainable throughput is roughly 1.5 Gbit/s per Ethernet link and 1 Gbit/s per SpaceWire AMBA port. In addition to this, the SpaceWire router will also be able to simultaneously route packets at maximum speed.

2.6 10/100/1000 MBIT ETHERNET

The Ethernet controllers support 10/100/1000 Mbit/s operation and have internal RAM that allows buffering a complete packet. Support for multicast will be included to allow reception of multicast packets without setting the interface in promiscuous mode.

2.7 HIGH-SPEED SERIAL LINKS

The availability and specification of the High-Speed Serial Link (HSSL) IP cores to be integrated within the European DSM ASIC platform is at the time of writing very limited. Aeroflex Gaisler is working with ESA to be able to provide, at the minimum, a descriptor based DMA controller to control the SerDes macros that are expected to provide 6.25 Gbit/s of bandwidth per link. The support of SpaceFibre is a goal, it is however subject to maturity of the standard amd availability of a SpaceFibre IP core.

2.8 DEBUG COMMUNICATION LINKS

The NGMP has a wide range of debug links; JTAG, SpaceWire RMAP, USB and Ethernet. The controllers for the first three links are located on the Debug bus and will be clock gated off in flight. The controllers for the two Ethernet debug links are embedded in the system's Ethernet cores.

The two Ethernet debug links use Aeroflex Gaisler's Ethernet Debug Communication Link (EDCL) protocol, which is completely supported in hardware and does not require processor intervention. The Ethernet controllers allow users to connect each debug link either to the Debug bus or the Master I/O bus. The Ethernet cores' normal function is preserved even if the debug links are active. The selected buffer size for the debug traffic in the NGMP gives an Ethernet debug link bandwidth of 100 Mbit/s.

A USB debug communication link controller provides a debug connection with relatively high bandwidth (20 Mbit/s). The wide adoption of USB will allow the NGMP system to be debugged from nearly any modern workstation without the need for configuration that is typically required when using an Ethernet Debug Communication Link.

The JTAG debug communication link provides a link with modest bandwidth of around 500 kbit/s, typically limited by the JTAG adapter. With modern USB JTAG adapters it is possible to run the JTAG link at 6 Mbit/s.

A dedicated SpaceWire RMAP target is included on the Debug bus in order to use the NGMP in SpaceWire networks. With a dedicated SpaceWire debug link it becomes easy to use existing infrastructure to control the NGMP system. The SpaceWire RMAP target will typically provide a debug link bandwidth of 20 Mbit/s. This is the rate seen in practice with the GRMON debug software through bridge devices. The hardware core itself runs at 200 Mbit/s nominally and is able to provide the ideal 152 Mbit/s in throughput.

3 FAULT-TOLERANCE

The fault-tolerance in the NGMP system is aimed at detecting and correcting SEU errors in on-chip and off-chip RAM. The L1 cache in the LEON4FT cores are

protected using byte parity and the register file in each processor is protected using TMR. As previously mentioned, the L2 cache is protected using BCH ECC and the external SDRAM memory is protected with Reed-Solomon. The boot PROM will use BCH. All RAM blocks in on-chip IP cores are protected with parity, TMR or parity DMR. Flip-flops will be protected with SEU-hardened library cells, if available and adequate, or TMR otherwise.

4 Improved Support for Time-Space Partitioning and Multi-Processor Operation

Beyond support for symmetric multiprocessing (SMP) configurations, e.g. with a central multiprocessor interrupt controller, NGMP also features extended support for asymmetric multiprocessing (ASMP) configurations: duplicated interrupt controller functionality and several timer units allow running separate operating systems on separate processor cores.

Each processor core has a dedicated memory management unit (MMU) that provides separation between processes and operating systems. The system also includes an IOMMU that provides access restriction and address translation for accesses made by the DMA units located on the Master I/O bus. The MMU and IOMMU provide access restriction and address translation to blocks of memory divided into 4 KiB pages. In order to grant selective access to the registers of one and only one peripheral core, all peripheral register base addresses are aligned on 4 KiB address boundaries.

In addition to the MMUs in each of the processor cores and the IOMMU, memory read/write access protection (fence registers) are implemented in the L2 cache. This functionality is primarily intended to protect backup software but can also be used to add another layer of protection with regard to space partitioning.

5 Improved Support for Debugging and Profiling

The NGMP includes new and improved debug and profiling capabilities compared to existing LEON2FT and LEON3FT devices. The selection of available debug links has previously been described. Additional debug support features of the NGMP include: AHB trace buffer with filtering and statistics, processor instruction trace buffers with filtering, performance counters for taking measurements in each processor core; hardware break- and watchpoints, interrupt time stamping in order to measure interrupt latency and a PCI trace buffer with filtering.

All performance counters and trace buffers can be accessed via the Debug AHB bus without causing traffic on the system buses. The processors can also access the performance counters via the Slave I/O bus.

6 Expected Performance

Several FPGA downsized configurations of the NGMP in the form of Field Programmable Gate Array (FPGA) prototypes have been developed during the architectural design phase. To compare the performance of the NGMP to previous LEON2 and LEON3 systems, a small collection of benchmarks have been assembled. While not providing an exhaustive performance profile, these benchmarks still

provide interesting compare points in the development of the LEON processor. The benchmarks have been run on the following systems:

- AT697: LEON2FT, 32 + 16 KiB cache, 5-clock multiplier, Meiko FPU
- UT699: LEON3FT V1, 8 + 8 KiB cache, 5-clock multiplier, GRFPU
- GR712RC: Dual core LEON3FT V2, 16 + 16 KiB cache, 5-clock multiplier, GRFPU, branch prediction
- NGMP: Quad core LEON4FT, 16 + 16 KiB cache, 2-clock multiplier, GRFPU, 256 KiB L2 cache

The benchmark collection consisted of the following benchmarks: 164.gzip, 176.gcc, 256.bzip2, AOCS benchmark, Basicmath_large, Coremark-1.0, Dhrystone-2.0, Linpack-DP, Whetstone. The three first benchmarks are from the SPEC CPU2000 suite. All benchmarks were compiled with GCC-4.3.2 tuned for SPARC V8. All systems were clocked at 50 MHz during the tests, using 32-bit SDRAM (LEON2/3) or 64-bit DDR2 (NGMP). Table 1 shows the performance figures relative to AT697.

Benchmark	AT697	UT699	GR712RC	NGMP
164.gzip	1	0.94	1.1	1.31
176.gcc	1	0.79	0.97	1.3
256.bzip2	1	0.93	1.06	1.33
AOCS	1	1.2	1.52	1.79
Basicmath	1	1.3	1.46	1.62
Coremark, 1 thread	1	0.89	1.09	1.21
Coremark, 4 threads	1	0.89	2.05	4.59
Dhrystone	1	0.94	1.05	1.39
Dhrystone, 4 instances	1	0.94	1.05	1.39
Linpack	1	1.2	1.26	1.71
Whetstone	1	1.94	2	2.22
Whetstone, 4 instances	1	1.94	3.7	8.68

Table 1: Relative benchmark scores

Table 1 shows that the LEON4/NGMP has approximately 30% better CPI than AT697 on integer benchmarks, and up to 100% better CPI on floating-point benchmarks. The Coremark benchmark can also be run multi-threaded, which shows on the high 4-thread results for GR712RC and NGMP. The benchmark will fit in the L1 cache, and therefore scales almost linearly with the number of cores. It should also be noted that these figures are for systems running on the same system frequency and that the target frequency for NGMP is significantly higher than the maximum frequency of the other devices.

All benchmarks were run using the BCC runtime. Using the Linux SMP OS, multiple instances of Dhrystone and Whetstone was run. These tests show that performance scales better on NGMP than GR712RC, mostly due to wider buses and the L2 cache.

7 TARGET TECHNOLOGY

The baseline target technology is the European ST Microelectronics 65 nm space technology. Possible backup options for target technology include UMC 90 nm with the DARE library and Tower (130 nm) with a library from Ramon Chips.

Power consumption of the NGMP ASIC core (without IOs) under worst case operating conditions and maximum software load is required to not exceed 6W. Maximum power consumption in idle mode (no software activity, but conservation of status and SEE protection) is required to not exceed 100 mW.

8 SOFTWARE SUPPORT

The GRMON debug monitor from Aeroflex Gaisler has been extended to support all new functionality included in the NGMP. The hardware platform provides full instruction set compatibility with existing LEON3FT software and all standard compilers that can produce correct SPARC V8 code can be used. Aeroflex Gaisler's bootloader creation tool MKPROM2 has been extended with support for booting ASMP configurations.

Board support packages for the NGMP will be delivered for RTEMS 4.10, eCos, VxWorks 6.7, Linux 2.6. Other operating systems that are already ported to LEON3/4 include: LynxOs, ThreadX and Nucleus.

9 PROTOTYPE

A functional prototype (FP) to be manufactured on ASIC Nextreme2 structured ASIC technology is currently being developed at Aeroflex Gaisler. Validation boards with FP devices are scheduled to be available in Q2 2012.

Aeroflex Gaisler can currently provide downsized FPGA prototypes of the NGMP system. The prototypes are described in the NGMP preliminary data sheet available at the NGMP website (http://microelectronics.esa.int/ngmp/ngmp.htm).

10 CONCLUSION

The NGMP is a SPARC V8(E) based multi-processor architecture that provides a significant performance increase compared to earlier generations of European space processors, with high speed-interfaces such as SpaceWire and gigabit Ethernet on-chip. The versatile on-chip SpaceWire router broadens the possible applications of a space processor chip. The platform will have improved support for profiling and debugging and will have a rich set of software immediately available. The NGMP also includes extended support for ASMP configurations and time-space partitioning.

The NGMP is part of the ESA roadmap for standard microprocessor components and it will be commercialised under fair and equal conditions to all users in the ESA member states. The NGMP is fully developed with manpower located in Europe, and it only relies on European IP sources. It will therefore not be affected by US export regulations.

The NGMP preliminary data sheet and other related documents are posted at the NGMP website following link: http://microelectronics.esa.int/ngmp/ngmp.htm

DEVELOPMENT OF A NOVEL 18X SPACEWIRE ROUTER

Session: SpaceWire components

Long Paper

Marko Isomäki, Sandi Habinc

Aeroflex Gaisler AB, Kungsgatan 12, SE-411 19 Göteborg, Sweden

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

ABSTRACT

The 18x SpaceWire router is a new 18 port stand-alone router component currently being specified by Aeroflex Gaisler. Today there is no component available on the world market exhibiting more than eight SpaceWire ports. The goal with this new development is to provide this missing key component to the ever increasing number of customers requiring manifold ports.

The 18x router is based on the GRSPWROUTER configurable SpaceWire IP core developed by Aeroflex Gaisler. Two configurations are foreseen as technically and commercially viable. One with 16 SpaceWire LVDS ports and either two SpaceWire LVTTL ports or two FIFO ports and the other with 16 SpaceWire ports and two internal AMBA ports bridging to external pins via a PCI interface. Which of these solutions will be selected is still open.

It is also an open item whether the device will include support for SpaceWire revision D (ECSS-E-ST-50-12D) and the new SpaceWire-D protocol.

1 INTRODUCTION

Currently there is no SpaceWire router component on the market with more than 8 SpaceWire ports. Both ESA and several companies in the space industry have indicated 16 ports as the most viable for routers in the near future. Aeroflex Gaisler intends to provide this key component with a new 18 port SpaceWire router ASIC. The design will be based on the GRSPWROUTER configurable SpaceWire router IP core [1]. This core supports three different port types: SpaceWire ports, AMBA ports and FIFO ports. These will be further explained later in the IP core section.

Two configurations of the IP core have been identified as potential candidates for the final ASIC: One with 16 SpaceWire LVDS ports and two LVTTL SpaceWire ports or two FIFO ports (Configuration 1) and the other with 16 LVDS SpaceWire ports and two AMBA ports (internal) connected to a PCI interface (Configuration 2). Both will be evaluated to determine which one will eventually be used for manufacturing.

Other considerations made for the ASIC is whether to include support for the upcoming revision D of the SpaceWire standard (ECSS-E-ST-50-12D) and the new

SpaceWire-D protocol. The problem is the lack of a schedule for finalization of these two standards which might then not be mature enough to fit the schedule of the ASIC.

This paper begins with briefly presenting key properties of the GRSPWROUTER IP core which is the major core in the designs. Then the two configurations are presented and compared. The new protocols will then be briefly introduced followed by a motivation for the desired inclusion in the router ASIC. The next section shows some platforms already available for the router IP which can be used for prototyping and evaluating the ASIC configurations. Lastly the preliminary information about the ASIC technology is given.

2 ROUTER IP CORE PROPERTIES

The GRSPWROUTER IP core [1] is the central component in both of the suggested configurations. It supports from 2 to 31 ports of three different types: SpaceWire, AMBA and FIFO. The SpaceWire ports are normal SpaceWire links and will support at least 200 Mbit/s. FIFO ports provide 9-bit parallel interfaces with control signals in each direction (read/write) which can be used to interface external units or to cascade two or more 18x routers without any glue logic. The AMBA ports interface to an AMBA AHB bus using DMA on the bus. All three port types connect to the core routers witch matrix using identical FIFO based interfaces. There is no way to distinguish the three ports on the SpaceWire packet level and upwards.

The configurability provided by the IP core makes it usable in many different applications. It has already been used in several standard rad-hard components on Actel RTAX2000SL and RTP roASIC3 FPGAs [2] and is also used in the Next Generation MicroProcessor (NGMP) [3] system-on-chip activity funded by the European Space Agency.

All mandatory features currently in the ECSS SpaceWire standard are supported by the core as well as some additional key functions not being available in other implementations e.g. packet distribution.

3 FEATURES COMMMON TO BOTH CONFIGURATIONS

This section lists the key features common to both configurations of the router. The list consists of features available in the router IP core as well as external auxiliary interfaces.

The base of both routers consists of the 16 SpaceWire LVDS ports. Each router port, regardless of type, is equipped with a timer which can be enabled/disabled. It is used to prevent deadlocks resulting from stalling source or destination nodes which could lock a port indefinitely. This feature might be introduced in the upcoming revision-D of the SpaceWire standard but is already available in this design.

All addressing modes mentioned in the standard are fully supported. Physical and logical addresses can be individually enabled to use group adaptive routing or packet distribution to any number of physical ports available in the router. The addressing is setup using a routing and port setup table.

The addressing tables and port FIFOs in the router consist of a considerable amount of RAM blocks which can experience SEUs and the contents can thus be corrupted. All RAMs are protected by parity DMR which means single errors are detected and corrected automatically. FIFO memories do not need any additional mitigation as they only contain data for a very short period until it is read. The routing and port setup tables however are much larger than the FIFOs and can contain static data for very long periods and are therefore much more susceptible to error buildup. To prevent the data from being corrupted with multiple bit-errors the core uses an automatic scrubber device which periodically refreshes the routing-table contents. Scrubbing does not have any impact on routing performance since the refresh reads are issued on idle cycles. In the improbable event of a multiple error occurring when performing a lookup in the routing table the packet being routed will be discarded and status bits and an external signal will be asserted. It is then up to the system designer to have some kind of system monitor handling this situation.

All configuration and status access are handled through configuration port 0 which is accessed using the RMAP protocol [4] from any of the other ports. The allowed ports for configuration accesses can be restricted if needed using several configuration options.

For diagnostic and test purposes SPI, I2C, UART and JTAG interfaces will be provided. These low pin count interfaces are suitable in the small package that will be used (see below) but at the same time have sufficient bandwidth for the amount of status and configuration in the router internals. As this method is available most of the router configuration options have been set to known good values after the reset which can then be changed using these interfaces. Very few are available from configuration pins at reset.

4 CONFIGURATION 1

The first configuration considered for the ASIC consist of the base mentioned in the previous section with 16 SpaceWire LVDS ports and in addition either two SpaceWire LVTTL or two FIFO ports. The only difference between the two different SpaceWire port types is the I/O type of the pads. The major design choice for this configuration is whether to include two FIFO ports or two SpaceWire LVTTL ports. The target package for the router is a simple to handle low-pin QFP which is quite limited and does require reducing the amount of configuration pins even more than previously mentioned to fit two FIFO ports. Choosing two LVTTL SpaceWire ports instead would save 36 pins but could reduce flexibility of the chip.

One of the applications of the FIFO ports is to cascade one or more routers without any glue logic. For this purpose the SpaceWire ports will work equally well and would in fact simplify matters. In most cases cascading would be done on a PCB and it is well understood how to route SpaceWire signals on a PCB. The FIFO interfaces are most useful when connecting directly to external processors and memories. To use a SpaceWire link instead would require the insertion of glue-logic providing a complete SpaceWire codec which would typically be done using a FPGA which increases design complexity considerably. It is however anticipated that the need to interface to external processors using parallel interfaces will be less required in the future since most processors will be equipped with SpaceWire interfaces which means

it is most likely that the two SpaceWire LVTTL ports will be chosen. This reason is also the motivation for considering configuration 2.

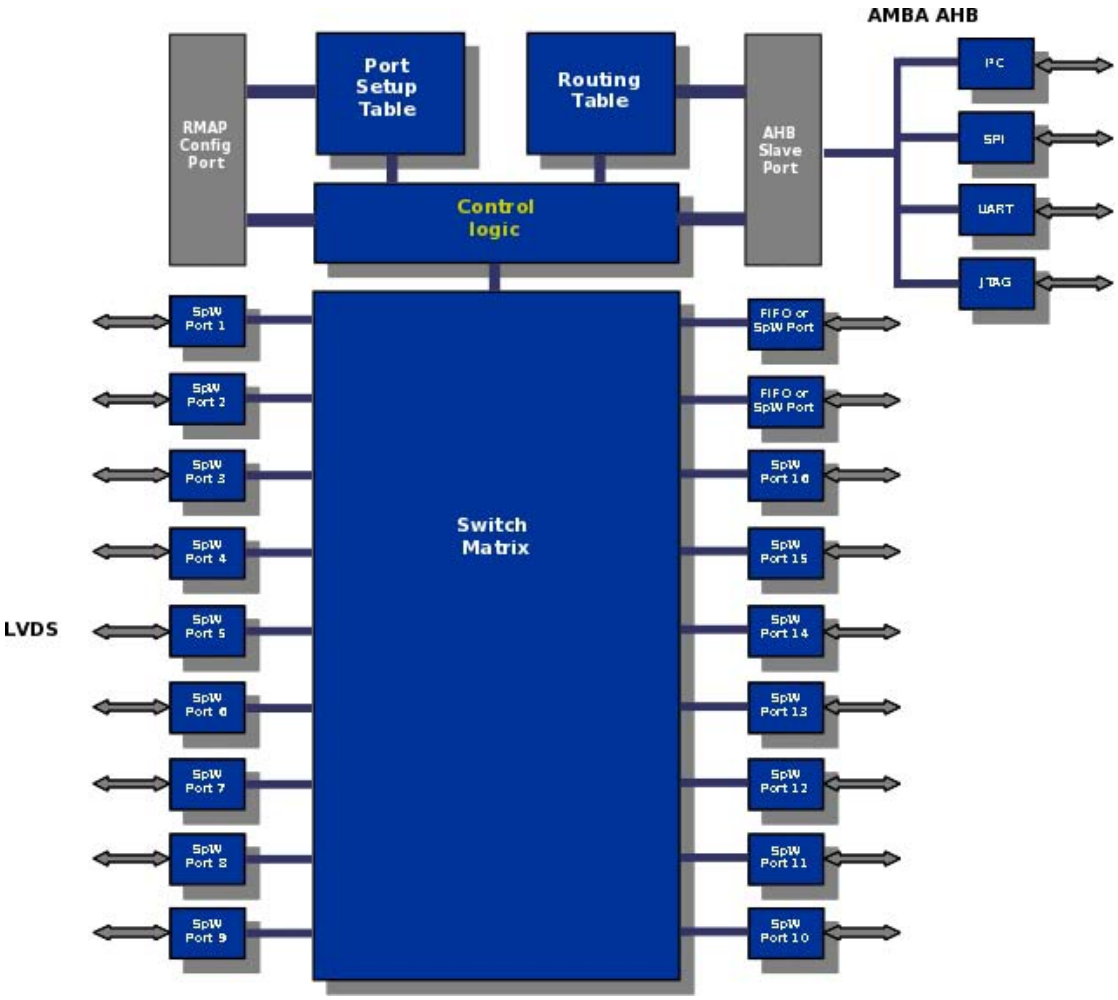


Figure: 18x SpaceWire router configuration 1 block diagram

5 CONFIGURATION 2

Similar to the first configuration the second one has 16 SpaceWire LVDS links but instead of FIFO ports or SpaceWire LVTTL ports a PCI interface is used instead. The router is configured with two internal AMBA ports which provide bridging using DMA to an on-chip AMBA-AHB bus where a PCI initiator/targets core resides.

Through the PCI interface any PCI master can get access to the whole AMBA AHB bus and send/receive SpaceWire packets through the two AMBA ports to any of the 16 SpaceWire links. There is also an AHB slave interface allowing direct access to the router configuration port. This speeds up configuration and status accesses considerably since the alternative would be to transfer RMAP packets over the AMBA ports addressed to the configuration port.

As mentioned in the section for configuration 1 it is believed that the need for external parallel interfaces will be less useful in the future and a bus interface like PCI will be more appropriate. An almost identical router with a PCI interface using the GRSPWROUTER IP core is already in use in an evaluation system at ESA as part of

the RASTA [5] framework. It is implemented on a Xilinx Virtex4 FPGA on a CPCI board. This is thought to be a more suitable type of external interfacing for future systems but the on-going evaluation process will determine which configuration is eventually selected.

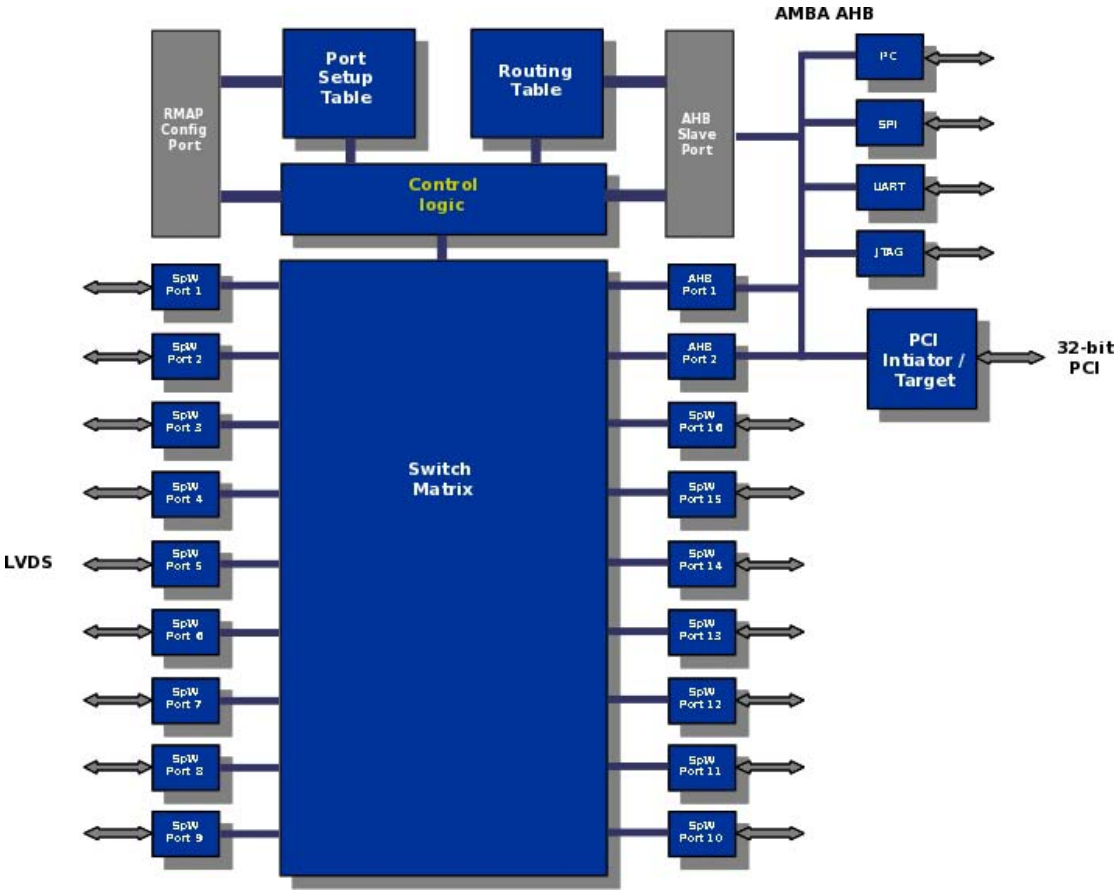


Figure: 18x SpaceWire router configuration 2 block diagram

6 SPACEWIRE REVISION D SUPPORT

An upcoming revision D of the SpaceWire standard is planned for the near future which contains some changes affecting the router ASIC development. Some additions result in old devices potentially not being forward compatible. It has to be carefully considered if and how these new features are implemented. The final details of the updates have not been decided yet and there is no date set for when this will be ready so there is a considerable risk in implementing these new features before the standard is finalized.

Three changes have been identified as having technical impact. The first one is the addition of timers in routers. This will probably be optional in the standard and not restricting the implementation details to any larger extent. The GRSPWROUTER IP core already contains a timer feature as previously mentioned which makes it probable that no changes will be needed to the core.

The second change is a modification of the link interface FSM. Two requirements have been identified [6] that potentially can cause the codec to make unwanted transitions. These are unlikely corner cases and very few if any problems have been

seen in practice. This will probably not affect backward compatibility with old codecs and so the risk is estimated to be very low to include these fixes in the router. Tests will be made during validation on FPGA that no disturbances occur with older devices.

The final and most complicated change is the addition of an interrupt code [6]. It uses one of the reserved control bit combinations of time-codes and it must therefore be made sure that it cannot interfere with the normal time-code facilities. Existing devices might not be forward compatible with revision D compliant devices due to the interrupt code. Some issues with these new codes are still under discussion and it is not known exactly how it will be implemented in the standard. This is therefore indentified as the part of revision D causing the highest implementation risk if included in the router ASIC. The desired way to go is that the router is flexible enough to allow ports' handling of the new code to be configured individually. In this way the router can be used as a device enabling old and new equipment to be used in the same SpaceWire network.

7 SPACEWIRE-D SUPPORT

There is a new protocol emerging called SpaceWire-D [7] where D stands for deterministic. This is anticipated to be widely used in the future to provide deterministic and low-latency transfer of control and command information while still preserving the high bandwidth of SpaceWire. It basically consists of a time-slotting table replicated in each unit (node or router) in the SpaceWire network. Therefore a router needs to have support for SpaceWire-D if it is used in a network utilizing that protocol.

The main complication here also lies in the fact the protocol has not been finalized and it lacks a time-table when this will be the case. It is not possible to make a sensible implementation with the information available at the moment. Since this requires RTL changes in the device a specification has to be available very soon for a possible inclusion in the router.

8 PROTOTYPING

Prototypes for evaluation of the router configurations are already available and are based on Xilinx Virtex 4/5 FPGAs with an accompanying evaluation board compatible with RASTA. The board provides the possibility to interface both through FIFO ports and the PCI interface depending on the configuration. All features planned for the ASIC are included and run at full-speed.

PCI drivers are also under development and will be available before the end of 2011. This is a plug and play driver which automatically detects the router design on the PCI bus. An API is provided for configuring, reading status, sending and receiving packets.

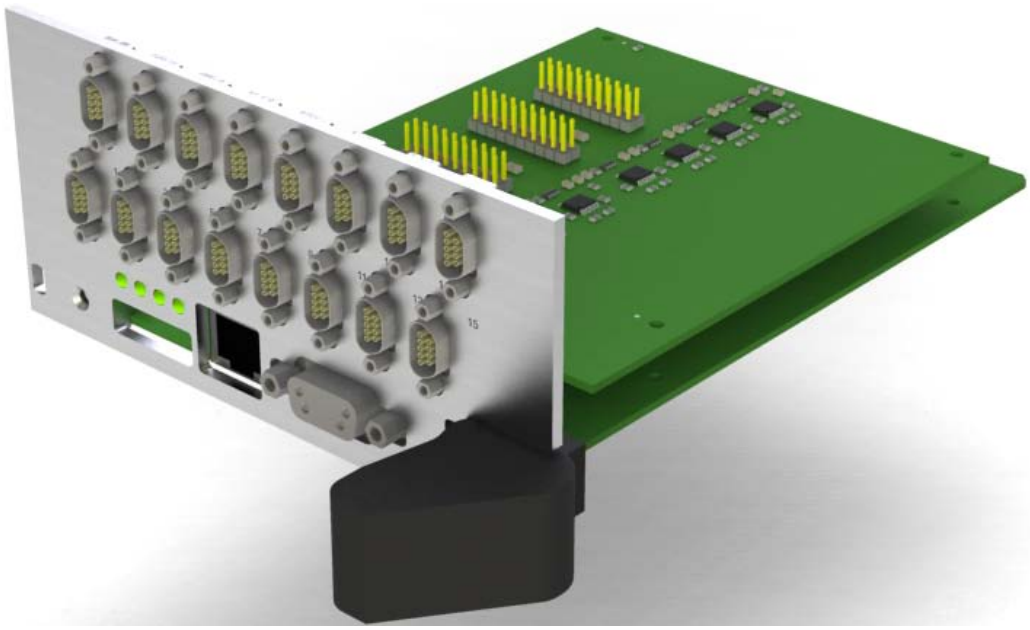


Figure: 18x SpaceWire router evaluation board with sixteen SpaceWire links

9 ASIC TECHNOLOGY

The ASIC will be targeted for a 0.18 µm or smaller technology. It is required to be SEE free and tolerate a TID of at least 100 kRad. Another important factor is low-power consumption. The actual process and library is yet to be determined.

The package is targeted for a simple to handle QFP type.

10 REFERENCES

1. M. Isomäki, S. Habinc, J. Gaisler, "A Configurable SpaceWire Router VHDL IP Core", Proceedings of the 3rd International SpaceWire Conference, 2010, 229-232

2. J. Andersson, M. Själander, J. Gaisler, R. Weigand, "Next Generation Multi-Purpose MicroProcessor", Data Systems in Aerospace Conference, DASIA2010, June 2010

3. RT-SPW-ROUTER Data Sheet and User's Manual, Aeroflex Gaisler, www.gaisler.com

4. Remote Memory Access Protocol, ECSS-E-ST-50-52C

5. RASTA Interface Control Document (ICD) - Hardware, TEC-EDD/2007.31/GF

6. ECSS Change Request/Document Improvement Proposal, 16th SpaceWire Working Group Meeting, www.spacewire.esa.int

7. ESA-PhA-SpW-DR equirements and baseline V 0-7, 16th SpaceWire Working Group Meeting, www.spacewire.esa.int

LEVERAGING SPACEWIRE NETWORK PROTOTYPING TO CREATE FLEXIBLE SPACEWIRE COMPONENTS AND SUPPORT SOFTWARE

Session: SpaceWire Components

Long Paper

Joseph Marshall, Steve Santee, Mary Hanley, Jeff Robertson, Dan Stanley

BAE Systems, Manassas, Virginia USA

E-mail: joe.marshall@baesystems.com, steve.santee@baesystems.com, mary.hanley@baesystems.com, jeffrey.robertson@baesystems.com, dan.stanley@baesystems.com

ABSTRACT

SpaceWire continues to find new usage in satellite systems worldwide. BAE Systems has created a demonstration and software development laboratory focused on rapid prototyping of network management, fault diagnosis and recovery algorithms for SpaceWire networks in a variety of topologies. This paper will describe BAE Systems' demonstration laboratory and results to date in topology and application modelling. It will also describe experience adding, implementing and prototyping the recently released SpaceWire Endpoint ASIC and describe its potential usage along with its support software in spacecraft systems utilizing SpaceWire especially those utilizing RMAP and plug and play.

1 INTRODUCTION

In 2004, BAE Systems released its SpaceWire ASIC, a combination router and system on chip (SOC). This joint BAE Systems and NASA Goddard based design [1] has been used standalone, controlled by its internal embedded microcontroller, and as a PCI-connected device to attach to other processor, memory or peripheral functions. Based on the success of missions such as the Lunar Reconnaissance Orbiter (LRO) [2][3] and future networking requirements of our customers, BAE Systems has expanded its planned offering of SpaceWire products to address a wider variety of applications including more advanced bridges, remote endpoints and large routers.

Understanding the ramifications and needs of these expanded networks led BAE Systems to setup a networking laboratory targeted for both LVDS and SERDES types of fabrics. The initial network implementations focused on SpaceWire. Several video sources and sinks are tied into the network to provide data for transport. A RAD750 processor provides control and network management. Various network topologies have been realized, network management approaches explored and middleware software has been developed. FPGA-based network nodes enabled prototyping two new SpaceWire ASICs: an enhanced system on a chip with a 4 port router and an endpoint with a single link. A large 16 port router and data funnel is also under development.

2 DEMONSTRATION SETUP AND TOPOLOGIES

The laboratory is set up with multiple FPGA prototyping boards[4][5] outfitted with 9 pin micro-d connectors. FPGAs utilized are Virtex 4 and Virtex 5 of varying sizes. These are connected by standard SpaceWire cables to each other to form the various network configurations. **Figure 1** shows the configuration used in most of the work and **Figure 2** shows a photo of the setup. One Virtex 5 FPGA supports a 12 port router/switch, four other smaller Virtex support 4 port endpoints, each tied to a laptop to generate video and a LCD display to display video. A third Virtex 5 FPGA type of board, containing the largest Virtex 5, an LX330T, provides a four port connection and maximum logic for prototyping. There is also a Virtex 4 board with sufficient LVDS I/O to support a 16 port router/switch.

A general purpose processor is provided in a small CompactPCI chassis. A 3U RAD750™ board[6], 3U Ethernet and a 6U SpaceWire ASIC evaluation board provide an embedded spaceborne type processor with four SpaceWire port connections to the network. One of these is used to interface to a 4Links SpaceWire 1U Test Board used for diagnostics, package insertion and performance monitoring.

Most of our connectors, though 9 pin micro-d in size, were not wired to the SpaceWire standard. Thus, the network can only reliably achieve around 100-110 Mbps by running up to 133 MHz. This was sufficient to test all behaviors and topologies and could provide a ready source of higher frequency errors for the network management software to handle.

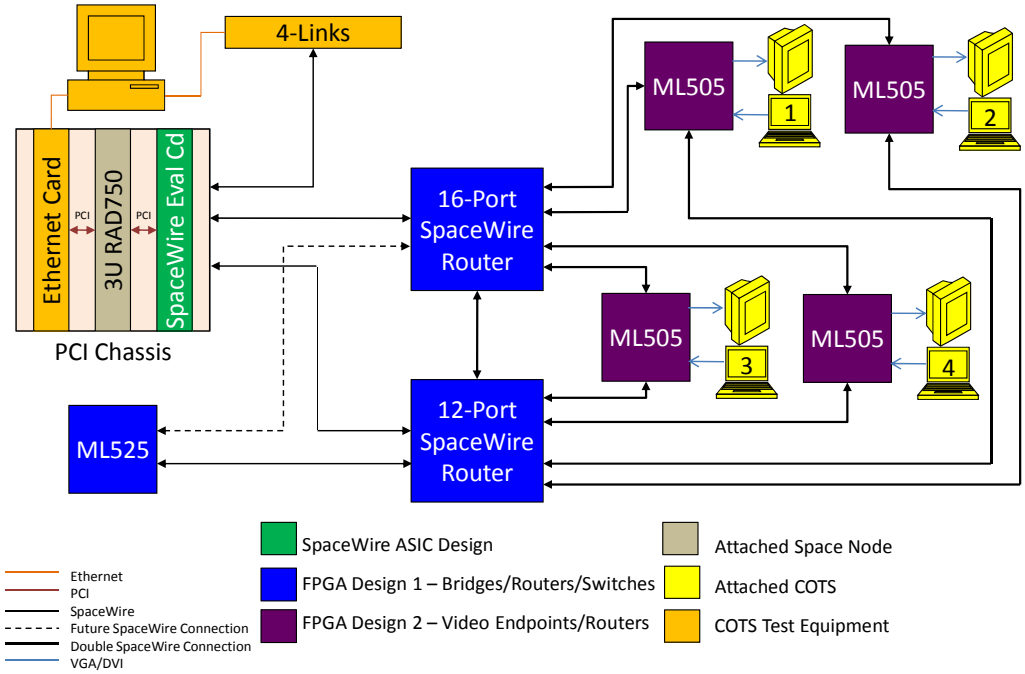


Figure 1 - Demonstration Network Diagram

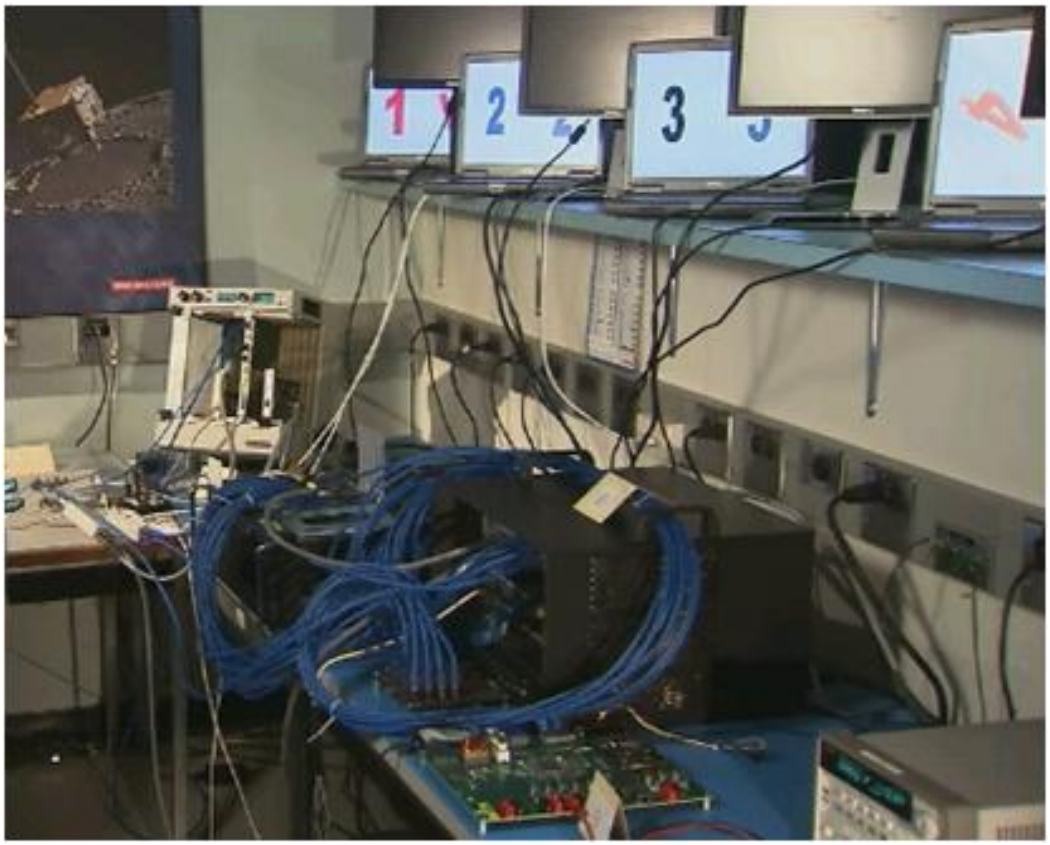


Figure 2 - Photo of Demonstration Laboratory

SpaceWire[7] supports a variety of topologies as shown in **Figure 3**. Rings are used to attach strings of processors in an efficient manner, in that only two links are required at any point[8]. If a second ring is included this provides a single fault tolerant solution making use of 4 port devices like the SpaceWire ASIC. However, the latency between any two points on the ring may grow beyond an application's performance requirements. Trees are used to fan out connections to larger numbers of nodes when a central switching approach is not possible. This is used in IEEE 1394 networks. Large routers/switches are used when latency is important and thus a minimum number of hops from a central resource is possible. Meshes are used between all equal nodes where all have requirements to communicate with other nodes. Hybrids of these four topologies are of course possible and often represent the actual implementation in a system.

Figure 4 captures the effects of each topology and compares between node sizes. It shows a mapping of different numbers of nodes to each of the topologies identified and which devices were optimal for each network type. This confirmed that endpoints and routers could be more useful and if available, more likely selected for some uses than 4-port bridge implementations.

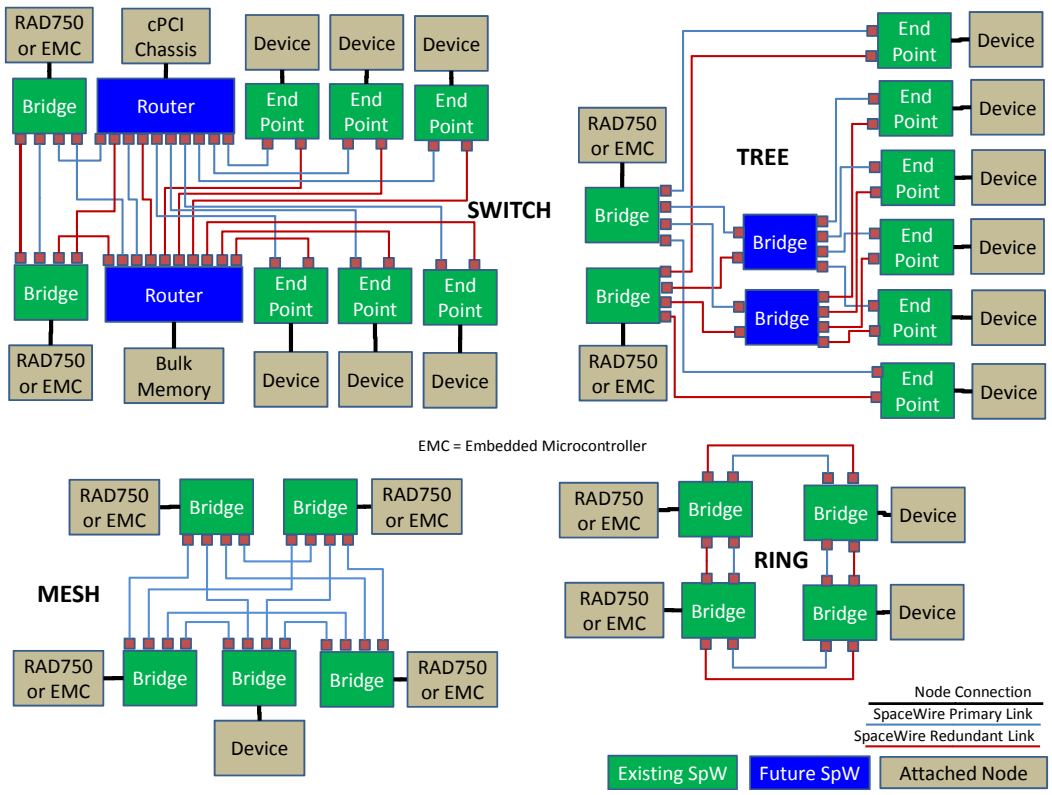


Figure 3 - SpaceWire Topologies

	Ring Network	Tree Network	Switch Network	Mesh Network
Topology	General purpose; dual path to each node	Large number of devices under control of small number of nodes - 1553 replacement; longer	backplane or localized network; common assets may be shared	many peers all sharing or sending and receiving data; localized
Nodes using only 1-2 port endpoints	Expandable Ring - must have 2 port router	can only be leaves of tree	can only be nodes connected to switch	two hosts with one port; up to three hosts with two ports
Nodes using only 4 port routers	Two expandable Redundant Rings: 2 ports each	Each Node supports up to three branches	up to four port switches; then must daisy chain (2x4=6; 3x4=9, etc.)	up to five hosts
Nodes using only 6 port routers	Three expandable Redundant Rings: 2 ports each	Each Node supports up to five branches or dual sets of two branches	up to six port switches; then must daisy chain (2x6=8 or 10; 3x6=12or15)	up to seven hosts
Nodes using only 8 port routers	Four expandable redundant rings: 2 ports each	Each node supports up to seven branches or dual sets of three branches	up to eight port switches; then must daisy chain (2x8=12 to 16; 3x8 = 15to24)	up to nine hosts
Nodes using only 12 port routers	Six expandable redundant rings: 2 ports each	Each node supports up to eleven branches or dual sets of five branches	up to twelve port switches or dual six port switches; then must daisy chain (2x12=18to22)	up to thirteen hosts
Nodes using only 16 port routers	Eight expandable redundant rings: 2 ports each	Each node supports up to fifteen branches or dual sets of seven branches	up to sixteen port switches or dual eight port switches; then must daisy chain (2x16=20to30)	up to seventeen hosts
Optimal Network Implementation	Two port routers for single ring; Four port routers for redundant rings	Use different sized routers to match localized groups of nodes	Smallest number of largest switch covering number of nodes to get most crossbar effect; if more than one; multiple cross links	Switch sized or used to match number of hosts

Figure 4 - Topology Affects on Network Attributes

3 VIDEO APPLICATION

To demonstrate a SpaceWire or other medium to high speed network, a variable high speed data source and sink is required. The FPGA boards we used for some of the smaller nodes included both a VGA in and a VGA out connection. Thus, with the addition of a laptop for data generation (connected to VGA in), an LCD display (connected to VGA out) and some FPGA personalization, we created variable high speed network sources and sinks. Due to the age of the VGA interface, it was not easy to find any good descriptions of what an interface device needed to produce or accept. Descriptions were hard to find and then turned out not to match what was actually being created or used. Experimentation and probing filled in the gaps that led to a successful implementation. A block diagram of the FPGA design for the video application is shown in Figure 5. Each block is an embedded core. RIFs are bi-directional FIFO based DMA interfaces between the SpaceWire router and the rest of the ASIC. JTAG and I2C blocks interface to industry standard interfaces. Most of the other blocks are self explanatory. All the demonstration designs included RMAP functionality so that all internal registers and connected memory and devices could be remotely loaded without device intelligence.

SpaceWire is a full duplex interface and thus can support video simultaneously in both directions. Full lowest resolution standard color VGA required just over 400 Mbps. By changing a color signal to one color, this was reduced to around 150 Mbps. This was still more than our simple lab setup could reliably transmit. Thus, two links were used for each video signal with separation at the source and recombining the data at the sink. This technique could easily be expanded to handle larger data sources and syncs with additional SpaceWire ports. Whether using one link or four, this FPGA demonstrates the endpoint function of sourcing data onto or sinking data from the network.

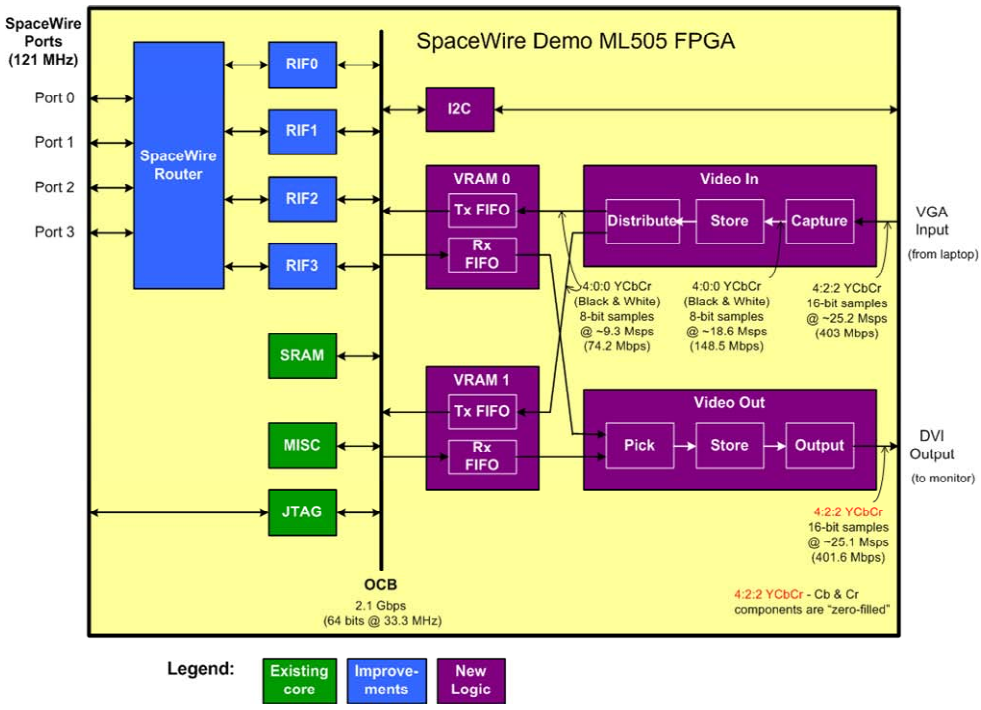


Figure 5 - Video FPGA Block Diagram

4 BRIDGE / ROUTER FPGA

The other major demonstration FPGA utilized the larger FPGA boards. Thus the design held all of the typical interface elements in a bridge or router with the exception of the PCI Bus, which was not available on these boards. **Figure 6** shows a block diagram of this design. The number of SpaceWire links depended on the number of LVDS signals brought out to the board connectors. Three implementations were wired, one with four ports, one with twelve ports and one with sixteen ports. The EMC is an embedded microcontroller that is used in all BAE Systems bridge and interface ASICs and along with 4x the embedded memory represent the biggest addition to the chip. Using the EMC, the ASIC can be used in a standalone mode or in a remote assist mode. The UART provides a low speed standard debug interface for the EMC-based code.

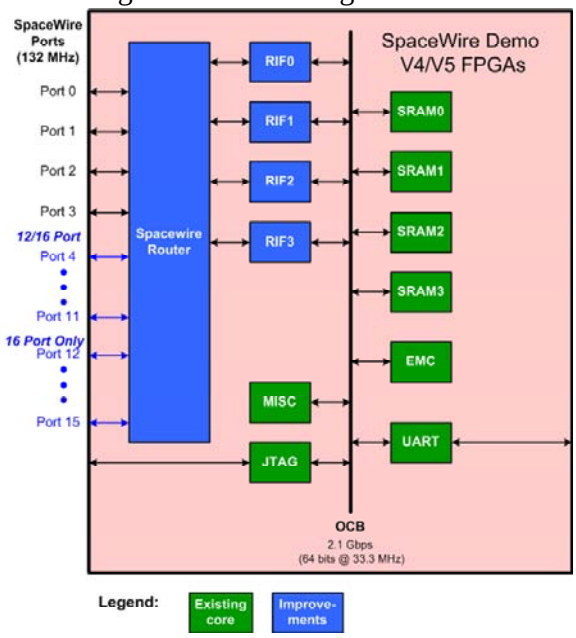


Figure 6 - Bridge / Router FPGA Block Diagram

5 SPACEWIRE ENDPOINT ASIC

As a result of the SpaceWire demonstration efforts, three new ASICs are in varying stages of development. The Golden Gate Bridge ASIC[8][9] provides an updated RAD750 processor bridge function for the RAD750 and parts are working in the lab. A 16 port router is in initial design and will provide a high performance crossbar between a 64-bit PCI Bus, 16 SpaceWire ports, internal and external memory and an EMC. Last year, the third design, a SpaceWire Endpoint completed design and this year was fabricated on BAE Systems' radiation hardened RH15 150nm CMOS line. A block diagram of this ASIC is shown in **Figure 7**.

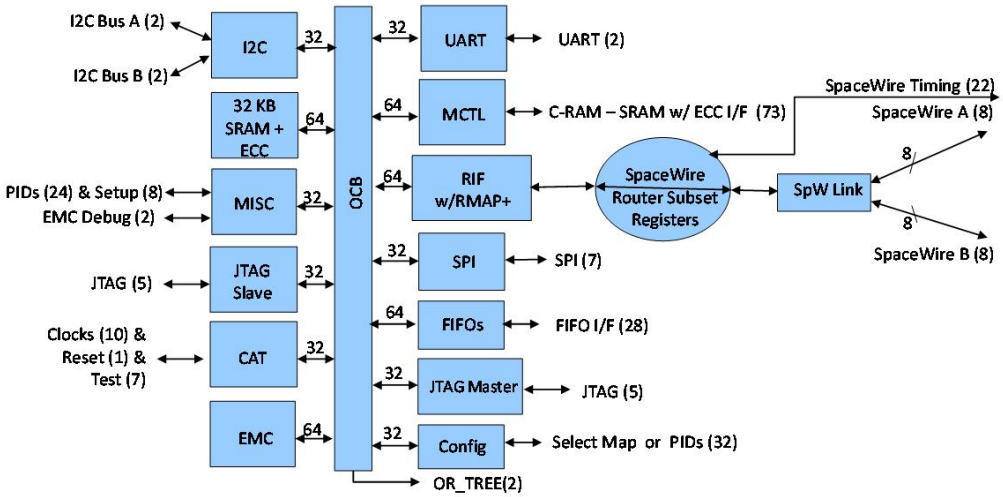


Figure 7 - SpaceWire Endpoint ASIC Block Diagram

The SpaceWire Endpoint ASIC was designed to function where SpaceWire was the only system interface, likely at the extremities or remote locations on a spacecraft. It has a single SpaceWire port with a redundant physical layer for fault tolerance. The SpaceWire port is rated to 320 MHz or about 250 Mbps of true data movement. It contains an EMC[9][10] with 32 KB of ECC-protected SRAM so that it can be used either standalone or directed through its RMAP registers remotely. It is a good match as a controller for a nanosat or CubeSat class satellite and at maximum speed can process at a 16 Dhrystone MIPS rate. It contains a set of matched interfaces to connect to a variety of remote devices such as memory, flash, logic, FPGAs, subsystems or instruments. Among these are two I2C, a 32 bit memory with ECC, SPI, UART, an 8-bit bi-directional FIFO, JTAG Master and SelectMAP. It also has 32 discrete signals, various timers and counters and a watch-dog timer. All resources are available to the EMC or to the remote SpaceWire master. The JTAG Master and SelectMAP allow it to configure and mitigate errors from RAM-based FPGAs. The SpaceWire Endpoint ASIC layout is shown in **Figure 8**.

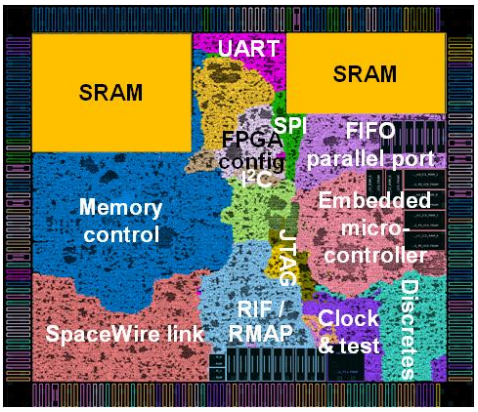


Figure 8 - SpaceWire Endpoint ASIC Layout

6 DEMONSTRATION SOFTWARE

The software created for the demonstration is what brought everything together and controlled it. Originally written to run on a PC, this software has been ported to the RAD750 running VxWorks and works as a middleware layer between an application and the typical board support package for the hardware. A diagram of the software is shown in **Figure 9**. The demonstration software includes a discovery algorithm that looks for different known variations of SpaceWire hardware, "discovers" the network hardware and then sets up the network based on the discoveries. If necessary, this

software cooperates with processes in the application. The software is able to manage the network by polling diagnostic registers in the various devices to keep track of status.

The discovery and management software communicates back to a demonstration program running on a PC that graphically shows the network, the connections, and applications running over those connections. A screen shot of a demonstration network is shown in **Figure 10**. Outside nodes represent endpoints while inside nodes represent bridges and other routers. Red arrows indicate assignments made for application data transfers.

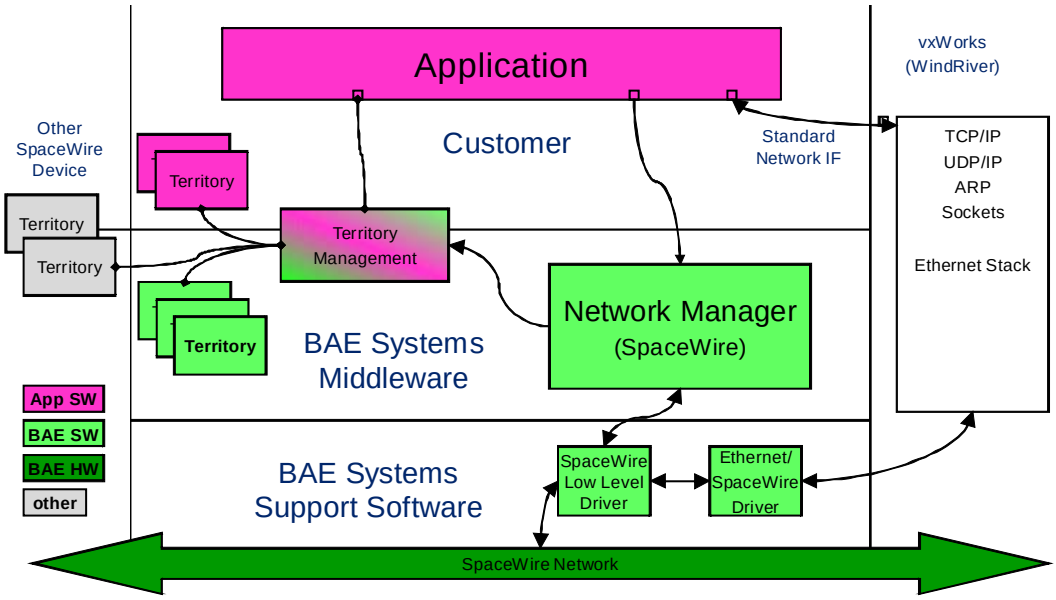


Figure 9 - SpaceWire Network Management Block Diagram

Each of the new ASIC designs as well as all of the demonstration FPGAs contain additional diagnostic registers that let the network management software monitor the traffic and health of all of the links. This has also been captured by the demonstration software on the PC. A picture of some software measurements of different links with and without traffic during a demonstration run is shown in **Figure 11**. An additional capability was developed and proven to run IP packets over a SpaceWire link. With this capability, a spare SpaceWire port may be used as a test interface for communication with test equipment. We successfully used this to load software and run VxWorks and its debuggers on the RAD750 using only a SpaceWire link. A diagram of the test hardware and software is shown in **Figure 12**.

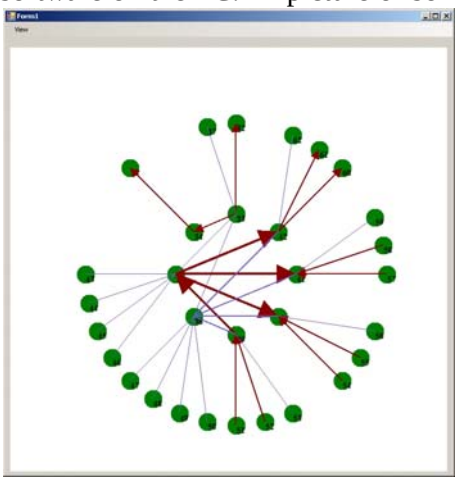


Figure 10 - SpaceWire Network Software Discovery Map

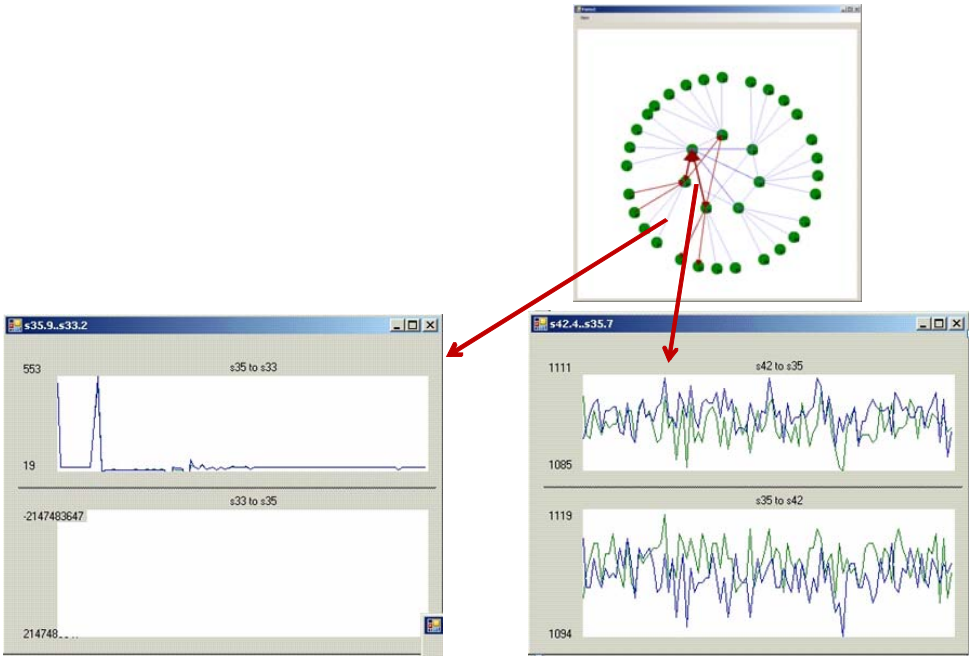


Figure 11 - Demonstration Software Measurements of Different Links

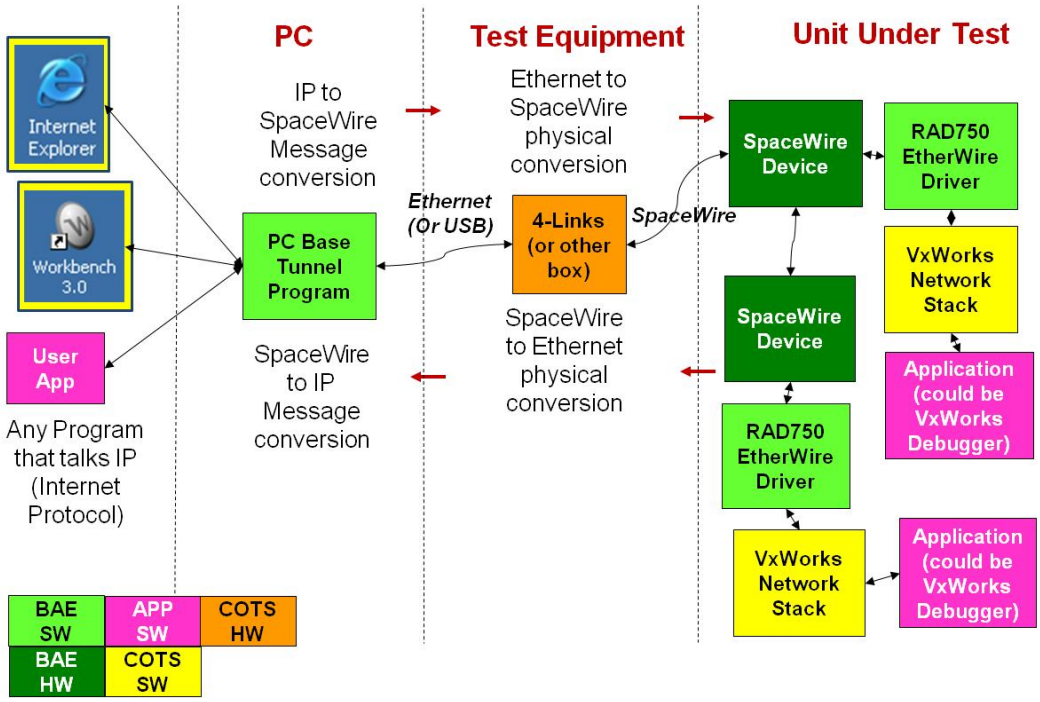


Figure 12 - SpaceWire Support Software

7 SUMMARY

In this paper we have discussed the SpaceWire demonstration laboratory, its hardware and software elements and how it has enabled a new set of SpaceWire ASICs and software based on elements prototyped and demonstrated in the lab. These new products address potential SpaceWire applications in big and small systems. The

laboratory provides a place to benchmark applications as well as a stepping stone to SERDES based future products.

8 REFERENCES

1. Marshall, J. R., Berger, R. W. and Rakow, G. P., "A One-Chip Hardened Solution for High Speed SpaceWire System Implementations", 1st International SpaceWire Conference, Dundee, Scotland, 2007.

2. Berger, R. W., et. al., "RAD750 SpaceWire-Enabled Flight Computer for Lunar Reconnaissance Orbiter", 1st 1st International SpaceWire Conference, Dundee, Scotland, 2007.

3. Marshall, J. R., "Evolution and Application of System On a Chip SpaceWire Components for Spaceborne Missions", 2nd International SpaceWire Conference, Nara, Japan, 2008.

4. "ML52x User Guide – Virtex-5 FPGA RocketIO Characterization Platform", Xilinx Corporation, April 2008.

5. "ML505/ML506/ML507 Evaluation Platform User Guide", Xilinx Corporation, July 2008.

6. Marshall, J. R. and Berger, R. W. "A Processor Solution for the Second Century of Powered Space Flight", 19th Digital Avionics Systems Conference Proceedings, Indianapolis, IN, 2000.

7. ECSS Secretariat, SpaceWire – Links, nodes, routers and networks, ECSS-E-ST-50-12C, July 31, 2008, Noordwijk, The Netherlands.

8. Marshall J. R., Wood N., Milliser, M., Ferguson R. and Maher E., "Higher Performance BAE Systems Processors and Interconnects Enabling Spacecraft Applications", IEEE Aerospace 2009 Conference, Big Sky, MT, 2009.

9. Marshall, J.R, Stanley, D. L. And Robertson, J. E., "Matching Processor Performance to Mission Application Needs", Infotech@Aerospace 2011 Conference, St. Louis, MO, 2011.

10. Marshall, J. R. and Robertson, J. E. "An Embedded Microcontroller for Spacecraft Applications", IEEE Aerospace Conference 2006, Big Sky, MT, 2006.

SPACEFIBRE CODEC: USE OF THE TLK2711-SP

Session: SpaceWire Components

Long Paper

Steve Parkes, Chris McClements

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

E-mail: cmcclements at computing.dundee.ac.uk, sparkes at computing.dundee.ac.uk

Martin Suess,

ESA, ESTEC, 2200 AG Noordwijk, The Netherlands

Email: martin.suess@esa.int

1 ABSTRACT

SpaceFibre is a very high speed serial communications link which is being designed for use on spacecraft. A SpaceFibre link connects high data rate payloads into the on-board data handling system and also interoperates seamlessly with a SpaceWire network. The link is able to operate over a copper or fibre optic communications medium and can support real data rates of more than 2 Gbit/s improving the data rate of SpaceWire by at least a factor of 10.

University of Dundee is currently developing a SpaceFibre VHDL IP core for ESA which is able to operate with an external SerDes device. It is also able to operate with the Texas Instruments TLK2711-SP Wizard Link device which includes an 8B/10B encoder and other logic as well as the SerDes.

The SpaceFibre IP core is being used in several ESA studies and will also be implemented on a demonstration board. The demonstrator system will use currently available radiation tolerant devices including the TLK2711-SP and the Actel RTAX FPGA device.

2 INTRODUCTION

SpaceWire [1] provides point-to-point and networked payload communication services for use on board spacecraft. It connects instruments to mass memory units and processing systems and provides the connection from the mass memory to the downlink telemetry system. SpaceWire uses bi-directional data links that operate up to 200 Mbits/s. Higher speed operation is possible when matched impedance connectors are used. SpaceWire is being used on many space missions across the world. This success is due to many factors including standardisation, simplicity of implementation, performance and flexibility.

The SpaceFibre standard is designed to work with existing high speed serialiser/deserialiser devices. This paper examines the issues raised when using the SpaceFibre protocol with the radiation tolerant Texas Instruments TLK2711 device

and addresses modifications required in the SpaceFibre specification to enable the use of this device in a SpaceFibre implementation.

3 BACKGROUND

The University of Dundee has been working on a Gbit/s data link technology for several years [3]. Trade-offs of ground data link technologies that could possibly be used as the basis for a new spacecraft Gbit/s data link have been carried out. An initial outline specification for SpaceFibre was written and various prototypes were implemented and tested.

Several instruments, including synthetic aperture radar and multi-spectral imagers, require higher data rates to the mass memory unit. Downlink telemetry systems are being designed that can support Gbit/s data transfer leading to the need for similar data rates to transfer the data from the mass memory unit. There is a growing requirement for a data communication link with an order of magnitude higher performance than SpaceWire. Standardisation, simplicity of implementation and flexibility are also import characteristics that need to be provided for a new data link technology to be successful. Furthermore, it must be possible to implement the high-speed serial interface in radiation tolerant, space-qualified technologies.

4 SPACEFIBRE CODEC

An overview of the SpaceFibre CODEC architecture is provided in Figure 1.

There are nine conceptual layers to the SpaceFibre CODEC:

Virtual Channel and Flow Control: responsible for quality of service and flow control over the SpaceFibre link.

Broadcast: responsible for broadcasting short messages across a SpaceFibre network and for receiving and checking those messages.

Framing: responsible for framing SpaceWire packets data, broadcast messages and FCTs to be sent over the SpaceFibre link. It is also responsible for scrambling SpaceWire packet data for EMC mitigation purposes.

Retry: responsible for recovering from transient and persistent errors on the SpaceFibre link, and for reporting errors and link failure. Detects missing and out of sequence frames.

Lane Control: responsible for operating several SpaceFibre links in parallel to provide a higher data throughput and to provide redundancy with graceful degradation.

Link Control: responsible for initialising the link, detecting link errors and re-initialising the link after an error has been detected.

Encoding/Decoding: responsible for encoding data into symbols for transmission and decoding symbols into data for reception.

Serialisation: responsible for serialising and de-serialising SpaceFibre symbols so that they may be transferred over the physical medium.

Physical: responsible for transferring the electrical signals across a fibre optic or copper medium.

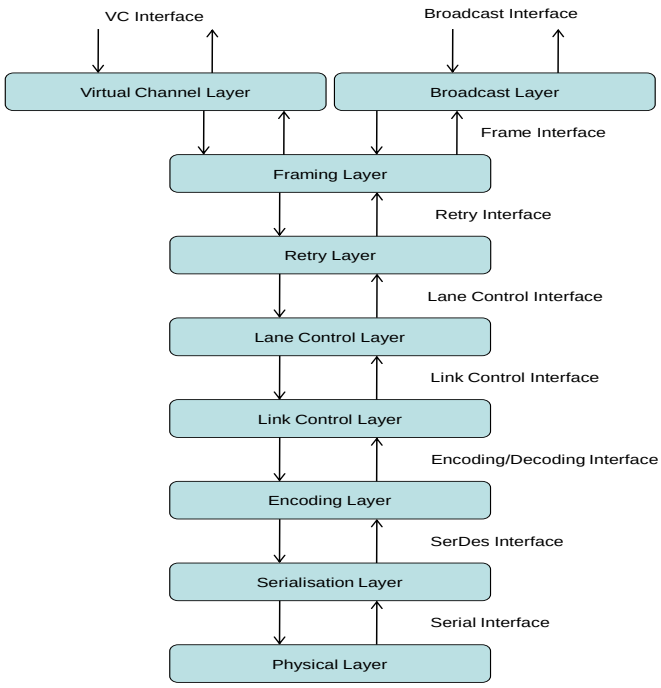


Figure 1 SpaceFibre CODEC architecture overview

5 TLK2711 WIZARD LINK

Wizard link [4] is family of high-speed serial communications devices. One of these is available in radiation tolerant form: the TLK2711-SP. This device contains both a transmitter and receiver and offers data rates from 1.28 to 2.0 Gbits/s (1.6 to 2.5 Gbits/s data signalling rates). The transmitter takes in 16-bit wide serial data, encodes it using 8B/10B encoding and serialises it for transmission over a VML differential signal pair. The receiver takes the serial data, de-serialises it, and performs 8B/10B decoding to provide the 16-bit parallel data. The TLK2711-SP is currently the device of choice when the data rate requirements exceed 1Gbit/s and it is widely used in a number of missions.

The TLK2711-SP device is attractive for use within a SpaceFibre CODEC as it provides the essential high-speed serialisation and de-serialisation technology, which is difficult to implement in a FPGA or ASIC unless radiation tolerant phase-locked loops are available in those devices. A complete SpaceFibre interface could be implemented using a radiation tolerant FPGA for the higher layers of the SpaceFibre protocol and a TLK2711-SP device for the Serialisation layer and part of the Encoding layer. The problem is that there are some characteristics of the TLK211-SP which prevent it being used to implement the initial SpaceFibre specification. The SpaceFibre specification needs to be modified to be able to use the TLK2711-SP and serialiser/de-serialiser devices with similar characteristics.

The following sub-sections describe the TLK2711-SP operation in some detail. The transmitter operation is described first, followed by that of the receiver.

5.1 TLK2711-SP TRANSMITTER

A block diagram of the TLK2711-SP transmitter is shown in Figure 2. Note that both the transmitter and receiver are provided in a single device.

The parallel data input to the TLK2711-SP transmitter comprises two bytes of data (TXD0-7 and TXD8-15) along with two control/data flags (TKLSB and TKMSB respectively). The control/data flags are high when the corresponding data byte contains a control code (K-code) and low when it contains data. The two data bytes and the control/data flags are latched into an 18-bit register on the rising edge of the TXCLK signal. Each data byte and its corresponding control/data flag is passed to an 8B/10B encoder, which converts them into a 10-bit code. The two 10-bit codes are passed to a 2:1 selector which selects the least significant 10-bit code first (generated from TXD0-7) followed by the most significant 10-bit code (generated from TXD8-15). Each 10-bit code is serialised in turn by a parallel to serial converter with the least significant bit being sent first. The serial data stream is passed to a differential, voltage mode logic (VML) driver for sending over a 50 ohm medium.

The TXCLK signal must be a continuous clock with a frequency in the range 80 to 125 MHz. This is used to register the data bytes and control/data flag into the 18-bit register, to drive the 10-bit code selector, and as the input to the clock synthesiser which multiplies up TXCLK by 20 to provide the clock to drive the parallel to serial converter. This means that the data signalling rate on the serial outputs is 20 x TXCLK, whereas the data rate is 16 x TXCLK (due to the 8B/10B encoding). The clock synthesiser also provides a reference clock for the clock recovery circuitry in

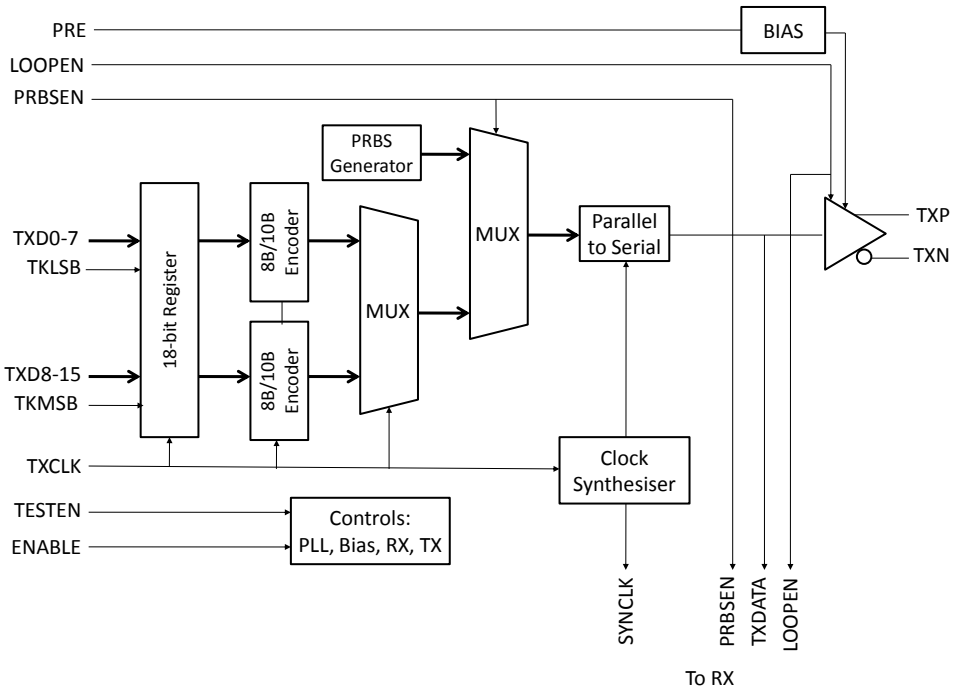


Figure 2 TLK2711-SP Transmitter

the receiver.

Copper transmission media have higher losses at higher frequency. This is seen as a slow rising and falling edges in the eye diagram at the receiver. To mitigate this problem it is possible to apply pre-emphasis to the transmitted signal: increasing the amplitude during the first part of the signal which compensates for the loss of this part of the signal through the transmission medium. Two levels of pre-emphasis may be selected using the PRE input. When low the pre-emphasis is 5%, when high it is 20%.

5.2 TLK2711-SP RECEIVER

A block diagram of the TLK2711-SP receiver is shown in Figure 3. Note that both the transmitter and receiver are provided in a single device.

The received serial data is received on the RXP and RXN pins and converted to a single ended signal inside the device. The TLK2711-SP device includes line termination at the input to the receiver. The received signal is fed via a pair of multiplexers to a serial to parallel convertor and to an interpolator and clock recovery block. The interpolator and clock recovery block recovers the received clock, to provide bit and word synchronisation.

Bit synchronisation is achieved using a phase locked-loop (PLL) that takes the transmit bit clock from the transmitter (SYNCLK) as a reference and provides an output frequency locked to the transitions on the received serial bit stream. To be able to do this the frequency of the transmit bit clock and the receiver bit stream must be almost the same i.e. within +/- 100 ppm. There must also be a sufficient number of bit transitions in the received serial bit stream for the receiver PLL to lock on to. This is guaranteed by the use of the 8B/10B encoding, one of the characteristics of which is plenty of bit transitions in each 10-bit code.

Using the bit and word synchronisation signals from the interpolator and clock recovery block, the serial data is converted to a correctly aligned pair of 10-bit codes. The two 10-bit codes are decoded by a pair of 8B/10B decoders, each providing an 8-bit data byte and a control/data flag RKMSB and RKLSB). These signals are registered in an 18-bit register.

6 TLK2711 COMPATIBILITY WITH SPACEFIBRE

The TLK2711 Wizard Link has some anomalies and features which made it functionally incompatible with the initial SpaceFibre specification. The inclusion of support for the TLK2711 device requires some adjustment to the layering and functionality of the SpaceFibre standard.

The functions of the device which can be used with the current SpaceFibre standard include: 8B/10B encoding and decoding, serialiser and deserialiser, line driver and receiver, clock recovery, and symbol synchronisation.

6.1 PROBLEMS WITH THE TLK2711-SP

The functions of the TLK2711 device which are not compatible with the initial SpaceFibre specification are now discussed.

Bit-Stream Inversion: The TLK2711 device does not support bit-stream inversion which is very useful to aid high-speed board layout. Bit-stream inversion should be mandatory for new devices, but optional for legacy devices. While legacy devices, like the TLK2711-SP, are being used bit inversion on the printed circuit board is not permitted.

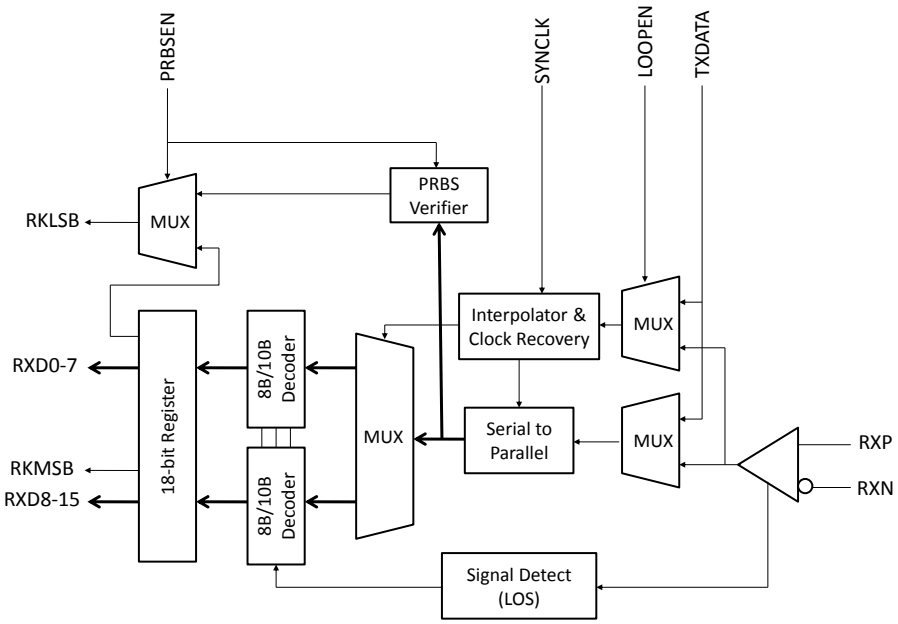


Figure 3 TLK2711-SP Receiver

Bit-Synchronisation: Bit synchronisation and symbol synchronisation are performed internally in the device but status information is not provided to indicate that bit

synchronisation has taken place. This impacts the receive synchronisation state machine, which used the bit-synchronisation signal.

Parallel Loopback: The device does not have the capability to support parallel loopback operation. This is not a serious limitation as the TLK2711-SP does provide the more important serial loopback capability. Parallel loopback needs to be made optional in the SpaceFibre specification.

Symbol Synchronisation: The TLK2711-SP does not support symbol synchronisation on negative disparity commas. This leads to the possibility of the link never being synchronised depending on the data being sent over the link. In SpaceFibre data and control words each contain four symbols with a data word decoding to a 32-bit data value. A synchronising control word starts with a comma in the least significant symbol position which is sent first. When this is detected in the receiver both symbol and word synchronisation can be performed. To support symbol and word synchronisation in the TLK2711-SP it is necessary to send two synchronisation control words, one after the other, and to ensure that the symbols following the comma in the word have even disparity. If the initial running disparity is negative, the first synchronisation control word will contain a positive disparity comma, and synchronisation will be performed successfully. If the initial running disparity is positive, the first synchronisation control word will contain a negative disparity comma, and synchronisation will not occur on that comma, but the running disparity will now be negative. The following three symbols all contain even disparity so the running disparity will be negative when the subsequent comma has to be sent. This comma with therefore have positive disparity and synchronisation will occur on this comma. The solution is to ensure that the control words being used for link initialisation start with a comma and are followed by three symbols with even disparity, then symbol and word synchronisation will be ensured during link initialisation. This solves the problem with synchronisation, but there is another problem when bit inversion is implemented. It is possible that the symbols forming the link initialisation are inverted when they are received. It is therefore necessary for the symbols forming the initialisation control word to have bit-wise inverse symbols that are both valid and also have even disparity. The initialisation control words for SpaceFibre have now been carefully selected to exhibit these properties.

Interface: The TLK2711-SP has a 16-bit interface (16 data bits + 2 D/K bits). The 32-bit data and control words from SpaceFibre have to be multiplexed over this interface and recovered with correct alignment in the receiver. The SpaceFibre specification needs to define the interface to the 8B/10B encoders in such a way as to permit different interfaces: 8+1, 16+2, or 32+4.

Word Synchronisation: The TLK2711-SP synchronises commas in the least significant byte position of a 16-bit word. The SpaceFibre specification needs to specify that commas are in the least significant symbol position of the control words. A means of aligning two 16-bit words into a 32-bit data or control word is required in the receiver.

Error Indication: The TLK2711-SP uses an invalid symbol to indicate the occurrence of errors in the receiver: K0.0 indicates the reception of an invalid symbol or the detection of a disparity error. This error indication must be decoded for use in the SpaceFibre receive synchronisation state machine.

Line Drivers and Receivers: The TLK2711-SP uses Voltage Mode Logic (VML) rather than Current Mode Logic (CML). The fibre optic components for SpaceFibre have been designed with CML in mind. CML provides better conducted emissions than VML. It is possible to translate from CML to VML using resistors so this is not a significant issue.

6.2 REVISED SPACEFIBRE ARCHITECTURE

It is important that SpaceFibre defines an interface to the lower layers which is compatible with different serialiser/de-serialiser devices. It may then be necessary to adapt a particular device to this common interface. The resulting architecture is illustrated in Figure 4.

There are two interfaces identified in this revised architecture which are relevant to serialiser/de-serialiser devices: the Encoding/Decoding interface and the SerDes Interface.

The Encoding/Decoding interface provides an interface which transfers control and data words. The Encoding layer is then responsible for the 8B/10B encoding of these words into groups of four symbols, which are then passed to the SerDes interface for serialisation and transmission. The Encoding layer receives unsynchronised parallel data over the SerDes interface and performs symbol synchronisation, 8B/10B decoding, and word synchronisation. The resulting stream of data and control words are passed out of the Encoding/Decoding interface. The Encoding layer also includes the Receive Synchronisation State Machine and an Error Decoder which translates error indications from the 8B/10B decoder into a form suitable for the Receive Synchronisation State Machine to use.

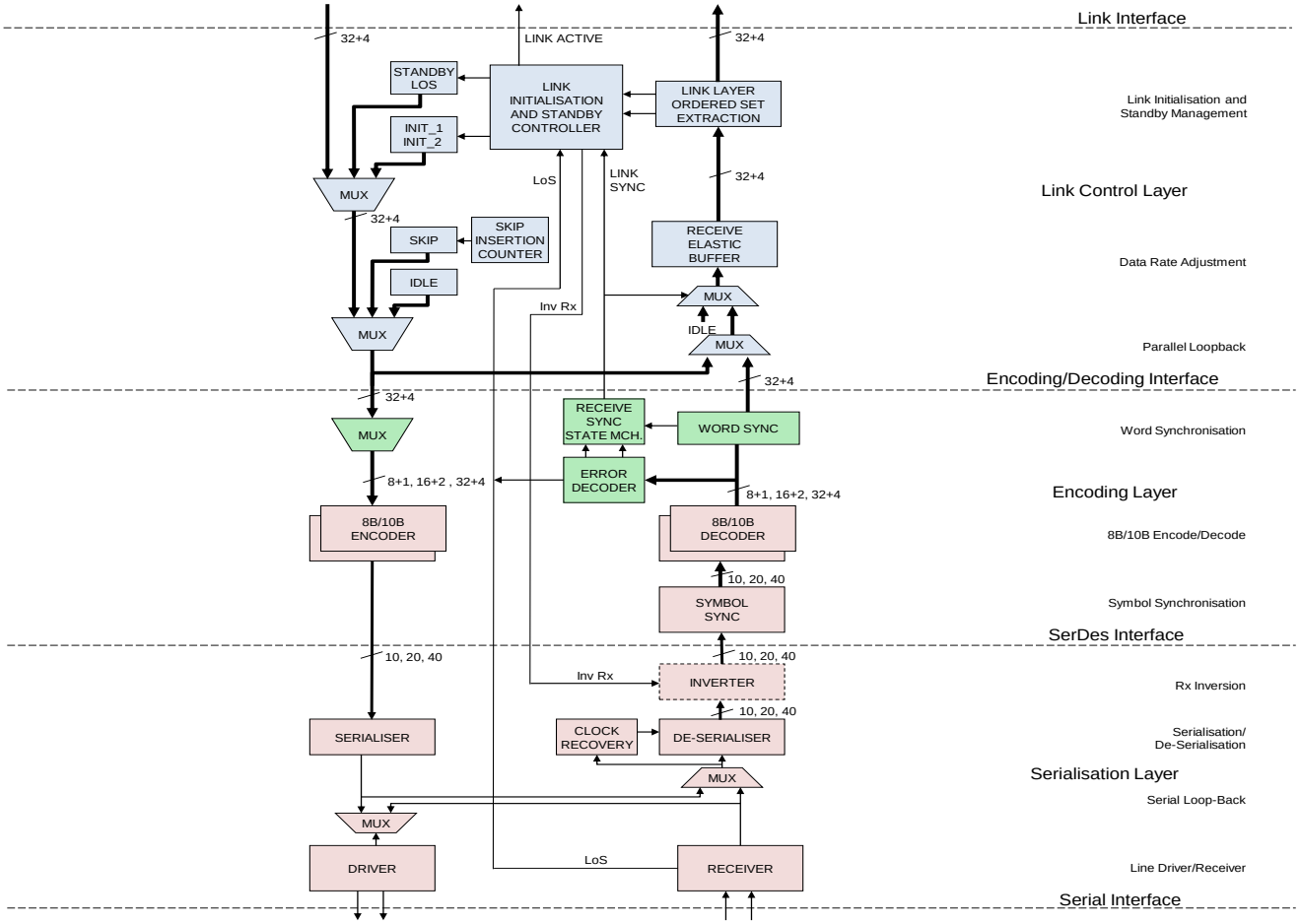


Figure 4 SpaceFibre CODEC transceiver supporting TLK2711

The SerDes layer contains the serialiser, line driver, line receiver, de-serialiser, bit clock recovery, and an optional bit-stream inverter.

The functions contained in the TLK2711-SP are shown in pink in Figure 4 and those that are required to adapt this device to the common Encoding/Decoding interface are shown in green.

7 CONCLUSION

SpaceFibre is designed to meet the high data-rate, onboard communication needs of future spacecraft. The requirement for radiation tolerant and space-qualified approach, obliges a pragmatic approach to the standard specification which permits the use of existing space-qualified components, without constraining the functionality and performance of SpaceFibre. A key component for SpaceFibre in the short term is the TLK2711-SP device which provides 8B/10B encoding/decoding and serialisation/de-serialisation functions in a radiation tolerant, space-qualified device. The SpaceFibre specification has been revised to permit the use of this and other SerDes devices with similar limitations. The SpaceFibre specification has been layered to permit ready adoption of this and future SerDes devices.

8 REFERENCES

[1] S.M. Parkes. C. McClements and M. Dunstan, "SpaceFibre Outline Specification", University of Dundee, 31st Oct 2007.

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

[3] S.M. Parkes. C. McClements, M. Dunstan and M. Suess, "SpaceFibre: Gbit/s Links For Use On board Spacecraft", International Astronautical Congress, Daejeon, Korea, 2009, paper IAC-09-B2.5.8.

[4] Texas Instruments, *TLK2711-SP Data Sheet: 1.6-Gbps to 2.5-Gbps Class V Transceiver*, Reference Number SGLS307D, July 2006, Revised July 2009.

Missions and Applications 2

BACKPLANE DESIGN CONSIDERATIONS FOR HIGH SPEED SPACEWIRE NETWORKS

Session: Missions and Applications

Long Paper

Shahana Aziz Pagen

MEI Tech Inc., NASA Goddard Space Flight Center, Greenbelt, Maryland, USA

E-mail: Shahana.A.Pagen@nasa.gov

ABSTRACT

SpaceWire is becoming a preferred protocol for board to board communication over a backplane in addition to its existing use over cabled interfaces, replacing other protocols due to its simplicity and readily available flight quality physical layer devices, IP cores and test equipment. However, without specific guidelines for implementing SpaceWire over a backplane, designers are left to make trade decisions regarding connector selection, layout design rules and test accessibility issues. This paper will discuss NASA's Goddard Space Flight Center's implementation of high speed SpaceWire over backplane on James Webb Space Telescope and other missions.

1 INTRODUCTION

SpaceWire has been used for several years for communication between spacecraft sub-systems over a shielded twisted pair cable interface. The SpaceWire interface is well suited for long length cables, while maintaining the signal quality required for high speed propagation. The SpaceWire standard has well defined specifications for the necessary design considerations for communicating over cabled interfaces.

However, SpaceWire can also be used within a sub-system for communicating between cards connected by a printed circuit board (PCB) interface (such as a backplane). SpaceWire has several advantages over other backplane based communication protocols like CompactPCI; with its relatively simple software interface, fault tolerance support, high data throughput and ease of expansion using nodes and routers. However, unlike CompactPCI, which has a well defined backplane standard; there are no rules or recommendations established in the SpaceWire standard that addresses the unique challenges of designing this interface for a backplane. While several cable based design considerations still apply, there are other design considerations that are unique to this application but not addressed in the SpaceWire standard. This can leave designers unsure of how to implement the protocol to achieve desired performance as well as meet adequate design margins.

Test and verification access is another area where currently available test equipment and test methodologies may not be adequate when the interface operates across a backplane. While most available test equipment has built in interfaces to the SpaceWire defined connector; it is up to the design engineer to consider accessibility

issues in the backplane environment and plan accordingly. If this is not considered early enough in the design phase, it may not be possible to accommodate later in the project's development.

2 OVERVIEW

This paper takes a step by step look at the various design trades that need to be made when designing SpaceWire interface over a backplane. The topics covered by this paper include the following:

- Connector selection: issues to consider include choosing a connector that is suited for high reliability applications and has the appropriate characteristics for high speed signal propagation
- Impedance control: specifying a stackup and routing constraints to meet differential impedance requirements
- Signal integrity and crosstalk: impacts to the design, methods of mitigating problems, analysis tool options
- Power integrity: methods of mitigating power distribution problems, analyzing return current flow, analysis tool options
- Test and accessibility: ways of providing probing access, verifying margins, interfacing to available validation and test equipment

3 DESIGN CONSIDERATIONS

In a backplane environment, multiple cards plug into the common backplane, high speed signaling passes between cards through PCB connectors across the peripheral cards and backplane PCBs. To ensure functionality and margins, several things need to be considered as part of both the peripheral cards and backplane PCB designs.

3.1 CONNECTOR SELECTION

Connector selection is an integral part of doing design for any high speed interface, and SpaceWire is no exception. The SpaceWire standard specifies 9-pin Micro-D (MDM) connectors, cabling and shielding, however, none of these apply well to a backplane interface. Peripheral cards and backplanes typically use PCB mounted connectors, which, if not selected correctly, can result in problems ranging from unreliable operation to complete failure at the required speeds.

Rugged connectors traditionally used for backplane interface design in space flight often have high inductance/capacitance contacts which do not adequately pass high frequency signals. Additionally, the connector contacts may not be properly matched to the trace impedance, causing an impedance discontinuity which may also degrade performance. Not all vendors provide high speed propagation data for their connectors. However more and more vendors are providing this service, most often vendors whose products are commonly used for high speed applications and not for lower speed space flight applications. For the JWST and ICESAT-2 missions the backplane connectors chosen for their high speed SpaceWire applications have excellent high speed performance characteristics up to 1GHz [1]. This data was obtained from the vendor (Hypertronics Corporation) who designed these connectors for CompactPCI – another high speed application. Hypertronics makes TDR and eye pattern data readily available along with connector models for customers to use to

validate their designs by simulation. Based on their modelling, they are also able to recommend an optimal pinout for arranging the differential pairs that minimizes interfering noise. Figure 1 shows the recommended pinout and routing pattern for alternating the "+" and "-" of each differential pair within a column, separated by ground and staggered from the location of the "+" and "-" pair in the adjacent column of the connector.

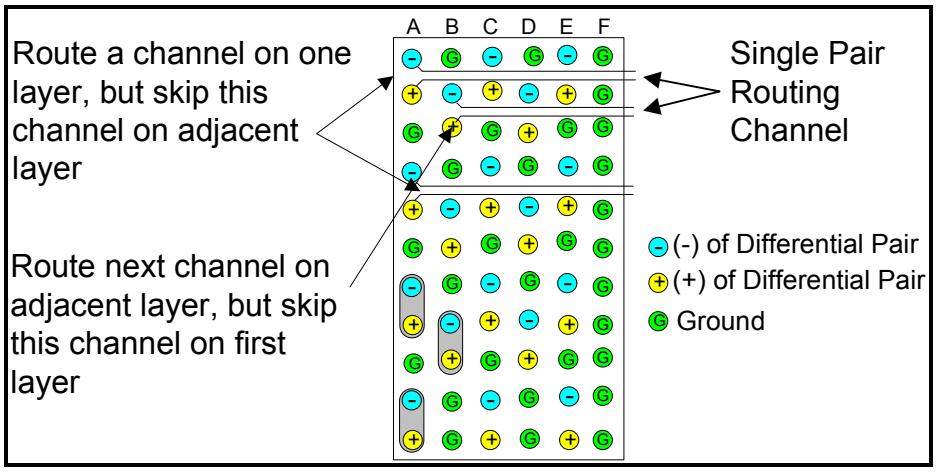


Figure 1: Connector Arrangement of a typical high density BP connector

This figure also demonstrates the difficulty with routing differential signals through the connector's pin grid. With densely spaced pins within a single connector and often multiple connectors lining up along the backplane, only a single routing channel may be routed between the pins for a single differential pair.

Connector vendors may also provide guidance on the size of the pad and antipad of the connector to reduce noise, EMI, jitter, improve manufacturability and reduce reflections that can in turn reduce data rates [2].

Designers can use various modeling tools to verify vendor data and ensure performance meets their custom requirements before locking down a design. This type of Multi-Board simulation can provide both single ended and differential simulation waveforms, along with eye pattern data [3].

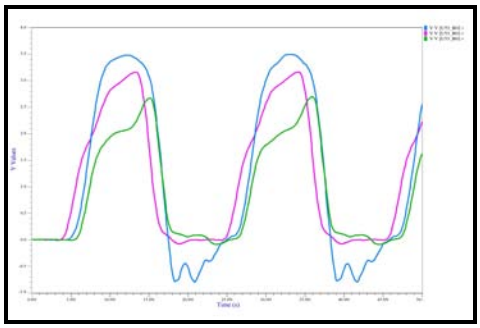


Figure 2: Differences in Signal Quality Depending on Connector Type

While all connectors make electrical connections, not all electrical connections are well suited for high speed propagation. A connector that might be qualified for flight and perfectly suitable for low edge rate signaling, may not function at the required

speeds for SpaceWire. Figure 2 shows simulated waveforms of a signal propagating between peripheral cards through a backplane using connectors with different R, L, C parasitic values. The contact R, L, C affects the path impedance and delay of the signal and can greatly change signal behaviour.

3.2 IMPEDANCE CONTROL

The electrical signaling requirements for SpaceWire over a backplane are the same as over a cabled interface, thus the 100-ohm differential impedance rule still applies. Engineers must take care to specify a set of routing rules and a PCB stackup that will meet these criteria over the entire length of the trace pair.

Figure 3 shows a typical impedance controlled stackup [3]. However, it is not enough to specify rules that meet the theoretical impedance numbers. The stackup and routing rules must also comply with a PCB vendor's manufacturing constraints. Vendors have material and process variations that mean that a set of rules that work for one vendor may not work for another and meet the same tolerances. Even with the same vendor not all materials achieve the same results. Surface finishes and the coatings used on the surface layers can change the impedance of traces routed on the outer layers. All of this must be considered upfront when choosing a vendor.

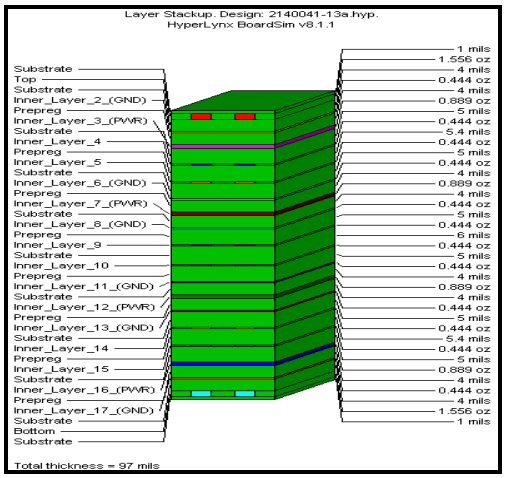


Figure 3: Example Impedance Controlled Stackup

Another trade is the differential trace routing topology. Two structures are commonly used for differential routing - edge coupled and broadside. With edge coupled, the differential pair is routed on the same layer side by side. With broadside the pair is routed on adjacent layers over-under. Figure 4 shows the difference between these two topologies. Edge coupled often presents a better solution for tighter impedance control. On the other hand, for broadside differential process and materials variations might have a larger impact on impedance variations. Vendors may not guarantee the tolerance for each broadside routing layer-pair [4].

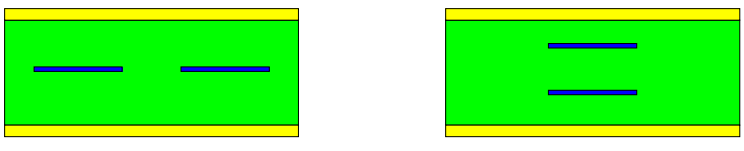


Figure 4: Edge-Coupled vs. Broadside Differential Routing

While edge-coupled may be superior for impedance control, it can be difficult to have enough space between high density connector land patterns to route a differential pair with the desired width and spacing for edge coupled impedance control as shown previously in Figure 1. This creates the need for tightly coupled differential routing, which comes with its own difficulties. Broadside routing can provide additional routing density, however depending on the di-electric thickness, may or may not create tightly coupled differential traces as well. Trades need to be made to select the appropriate structure that does not impose impossible constraints on either the design or the manufacturing process. If these things are not determined upfront, a design may not be manufacturable or may not be able to meet the 100 ohm differential impedance requirements.

3.3 Signal Integrity and Crosstalk Concerns

Signal Integrity and crosstalk concerns are not unique to SpaceWire. Any high speed PCB design has to pay special attention to ensuring proper signal integrity and minimizing crosstalk. When SpaceWire signals are not isolated by cable shielding and are routed on a backplane, they are far more susceptible to noise. This problem is exacerbated by the fact that LVDS SpaceWire signals may run on the same layer or adjacent to densely routed noisier single ended traces, such as LVTTL.

Differential traces need to be routed in a way to minimize the chance of coupling from an adjacent differential pair or an adjacent single ended trace, while at the same time maintaining the required coupling to meet differential impedance. Coupling can occur on the backplane or on the peripheral cards which source the signals or the destinations where they end. Traces run on adjacent layers, because of thin dielectric materials the separation between two signal layers might be less than a typical trace separation, causing more crosstalk than from signals routed on the same layer. Additionally, unlike in a twisted pair cable, aggressor nets can, and usually, couple asymmetrically, as opposed to common mode coupling, to each trace in the pair causing timing and jitter problems. It is important to ensure possible aggressor nets are sufficiently distant from the pair that coupling effects are insignificant.

Signal integrity can also be affected by the connector selection as mentioned earlier, the difference in trace length, and the driver or receiver devices used for the link. A practical approach to trace matching should be taken by considering the skew budget instead of trying to obtain an exact match in trace length. Adding serpentine delay lines in order to match a pair can cause more degradation of the circuit than having a practical length difference that still meets the skew budget of the fastest rise and fall times at the receiver [4].

Signal integrity analysis tools provide the best ways to trade these issues and quantify the noise risk. Eye pattern analysis can give a designer early indication of problems that might occur due to impedance mismatches or the particular type of connector and driver-receiver devices. Crosstalk can also be verified using simulation tools in a multi-board simulation environment that provides worst case numbers for coupling accumulated over the entire route. This eliminates the risk of bit failures that may only happen intermittently under certain switching situations. Corner case simulations can be used to verify margins. Figure 5 and Figure 6 show examples of simulation tool

results that designers can use to verify their designs before fabrication, avoiding costly respins and compromising mission success [3].

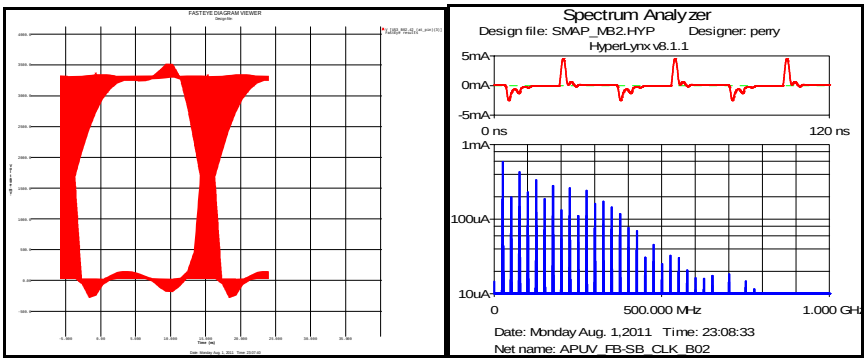


Figure 5: Example Signal Integrity/EMC Simulation Results

Type	E-Net	Receiver	Meas.	Source	Contrib
crosstalk	bp/1/net/AD21	bic/1/U1-100	354.1	AD20/bic/1/U1-104	186.9
crosstalk	bp/1/net/AD21	hk/1/U50-100	257.4	AD20/hk/1/U50-104	165.9
crosstalk	bp/1/net/AD23	bic/1/U1-94	240.3	AD21/bic/1/U1-100	161.3
crosstalk	bp/1/net/AD23	sbc/1/U1-G5	221.9	AD21/bic/1/U1-100	44.3
crosstalk	bp/1/net/AD21	sbc/1/U1-J7	217.6	AD20/bic/1/U1-104	167.5
crosstalk	bp/1/net/AD20	bic/1/U1-104	192.3	AD21/bic/1/U1-100	192.3
crosstalk	bp/1/net/AD23	fpap1/1/U1-94	182.8	AD21/hk/1/U50-100	35.7
crosstalk	bp/1/net/C_BE3_N	sbc/1/U1-J8	180.3	AD23/sbc/1/U1-G5	180.3
crosstalk	bp/1/net/AD20	hk/1/U50-104	169.5	AD21/hk/1/U50-100	169.5
crosstalk	bp/1/net/AD20	sbc/1/U1-F1	160.4	AD21/sbc/1/U1-J7	160.4
crosstalk	bp/1/net/C_BE3_N	fpap1/1/U1-86	150.3	AD23/fpap1/1/U1-94	150.3
crosstalk	bp/1/net/AD21	fpap1/1/U1-100	150.2	AD20/fpap1/1/U1-104	112.3
crosstalk	bp/1/net/AD21	fpap3/1/U1-100	148.5	AD20/bic/1/U1-104	99.2
crosstalk	bp/1/net/AD23	hk/1/U50-94	145.9	AD21/hk/1/U50-100	72.5

Figure 6: Example Crosstalk Simulation Results

3.4 Power and Ground Noise

When routing SpaceWire on a PCB, care must be taken to ensure proper routing of the ground plane as well as minimizing noise on the power delivery network (PDN). In a backplane environment there is no shielded cable that runs the differential pairs across large distances, so the shielding must be handled via ground routing on the PCB itself. Care needs to be taken to design the power distribution network where noise transients are adequately minimized. This includes having adequate decoupling capacitors but more so inter-plane capacitance that is effective at higher frequencies where decoupling capacitors are not effective. Simulations can again be used to verify PDN noise and margins.

Another important element is the location of power and return planes and the impact of return currents on inducing noise on other signals or planes. Differential traces are best routed adjacent to a ground reference plane and not crossing planes through vias, which can have unintended results with return currents and induced reverse crosstalk. This is true for the single ended signals that may share the same PCB. If care is not taken on providing for a clear return path, then unaccounted for reverse crosstalk may induce noise onto the differential signals reducing noise margins.

4 Test and Accessibility

When designing backplane distribution for SpaceWire, test and accessibility considerations must be made during the design phase as access cannot be built into

the system once the PCB's are fabricated. Again, off the shelf SpaceWire test equipment is designed to interface to the standard 9 pin MDM connectors, thus without necessary access points, – test and verification when peripheral cards are installed into the backplane may prove to be impossible.

4.1 On-board Probe Access

Eye pattern measurements are a common way of verifying performance and margins. These measurements are made by attaching a differential probe near the receiver and apply to both cable and backplane based systems. If access is not designed into the PCB, optimal measurements cannot be made and the results will be inaccurate. Designers should consider placing test terminals close to the receiver in a 3 pin arrangement that complies with the dimensions of the particular model of differential probe with ground pin that will be used during testing. This makes it possible to properly connect a measurement probe without degrading the measurement. However, care must be taken that the type and placement of the test terminal will not degrade the signal itself. Modeling can again be done to ensure that the location of the terminal or the via used does not adversely affect the signal.

Another potential problem is being able to access the test terminal itself. If the card is installed into a backplane next to other cards, that that test terminal may not be accessible. During testing it may not be feasible to demate the card and test it on a bench top environment where probe access is possible or recreation of the problem may require the existence of the other cards in the system. Extender cards are an excellent way to provide access to a single card when installed in a system. However, extender card designs have to take signal and power integrity issues into consideration and may need to be custom designed for this purpose. Because adding an extender changes the trace length, any differences in propagation delay and skew must be accounted for post measurement. Multi-board simulations can again be used to validate the extender card design, and identify differences between the extender and non-extender signaling by correlating the simulated vs. actual measurement results.

4.2 Interfacing to Test Equipment and Analyzers

Test requirements often dictate the need to use link analyzers or other test equipment for functional and margin testing of the SpaceWire interface. Such equipment is likely to be available only with the standard 9 pin MDM interface. Duplicating test features with custom ground support equipment can cause an impact to schedule or be cost prohibitive. Thus ensuring that existing ground support equipment (GSE) can be used without modification is a goal designers must achieve.

One way to accomplish this is to include the footprint of a PCB mounted MDM on the peripheral card itself. However this requires additional space and may degrade the SpaceWire signals due to the location of additional stubs and vias. In this case an extender card and/or a test backplane with breakout connectors are likely to provide the best solution. In either of these conditions the unit under test is installed into the extender or test backplane. The extender or test backplane includes a breakout connector to a PCB mounted MDM connector to which test equipment can be readily connected. This offers a way to test the board in a similar arrangement to the standard cable interface without incurring any additional development cost. Figure 7 shows an

arrangement where the peripheral card backplane connector is installed on one side of a test backplane with breakout connectors on the back.

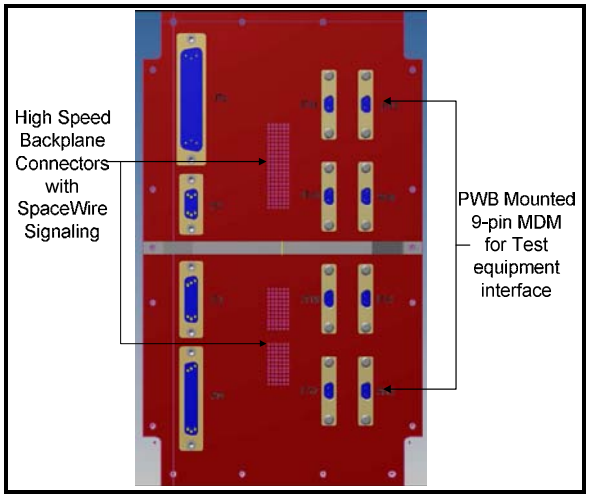


Figure 7: Peripheral Card Test Access

Designers need to accommodate the proper mechanical mounting of the PCB mounted MDM. A ground connection to the metal shell of the connector should be maintained such that the SpaceWire cable used for interfacing to the test equipment has the same grounding path as a panel mounted MDM. Without taking this into consideration it is possible to damage of degrade the flight and/or test hardware. Many PCB mounted MDM connectors do not include a metal body, so care must be taken when selecting a connector to provide proper grounding.

4.3 CONCLUSION

This paper has taken a brief look at some of the various complexities regarding a backplane distribution system for SpaceWire. While SpaceWire provides an excellent solution for board to board interfaces within a backplane distribution system, failure to consider the issues unique to this environment risk degradation of system performance, and even mission failure.

5 REFERENCES

1. Frank Morana, Hypertronics Corporation, “Single Ended and Differential TDR Characterization Data”, August 2010.

2. Tyco Electronics, “AMP Z-Pack HS3 Connector Routing”, Report #20GC004-1, November 15, 2000.

3. Hyperlynx SI, Hyperlynx PI and Interconnectix Synthesis, Signal and Power Integrity Tools, Mentor Graphics Inc.

4. Lee W. Ritchey, “A Treatment of Differential Signaling and its Design Requirements”, Sept 9, 2008.

NEXTAR: SMALL SATELLITE BUS BASED ON SPACEWIRE DETERMINISTIC IMPLEMENTATION

Session: SpaceWire Missions and Applications

Short Paper

Hiroki Hihara

NEC TOSHIBA Space Systems, Ltd., 10, Nisshin-cho 1-chome, Fuchu, Tokyo, Japan

Toshiaki Ogawa and Kenji Kitade

NEC Corporation, 10, Nisshin-cho 1-chome, Fuchu, Tokyo, 183-8551, Japan

E-mail: h-hihara@bc.jp.nec.com, t-ogawa@dt.jp.nec.com, k-kitade@cq.jp.nec.com

ABSTRACT

The NEXTAR (NEC Next-generation Star) standard platform provides a payload application development framework based on deterministic communication protocol through SpaceWire and Remote Memory Access Protocol (RMAP), and system integration can be completed in a short time without reducing reliability. The protocol layer for time slot control is separated from re-transmission and redundancy control protocol layer in order to implement determinism in the communication protocol for the NEXTAR standard platform, because RMAP packet format, which has inherent transaction capability, can be fully exploited for diagnosis and assured transmission leaving the time slot control capability within SpaceWire protocol layer. This scheme is formalised in SpaceWire-D draft specification.

1 DETERMINISM REQUIRED FOR SATELLITE BUS SYSTEM

Small satellites are expected to be used widely for remote sensing purposes. Since the earth observing satellite are required to be put on orbit promptly for commercial use as well as scientific purposes, assembly and integration duration are desired to be as short as possible. NEXTAR standard platform responds to this need by providing determinism without any modification on SpaceWire and RMAP protocol.

SpaceWire is often used for payload subsystem because of its high-speed transmission capability. We also use SpaceWire for the bus system in order to unify testing environment for satellite bus and payload, and additional characteristics are required on SpaceWire. The major requirement is determinism, and it is going to be incorporated in the additional specification of SpaceWire discussed as SpaceWire-D in the SpaceWire working group. 'D' stands for determinism.

One reason of the usefulness of deterministic implementation of SpaceWire is effectiveness for reducing test cases during the validation of communication among onboard equipments. Deterministic communication protocol is also useful for employing as-built equipment, because those equipments often accommodate

deterministic communication characteristics for the transmission of command and telemetry based on legacy protocol like MIL-STD-1553B, UART, or CAN.

2 DETERMINISM IMPLEMENTATION EXPLOITING RMAP PROTOCOL

Existing SpaceWire and RMAP protocol specification is to be used without any modification, in order to keep compatibility. The specifications we use are the original SpaceWire protocol, and two upper layer protocols, which are Protocol Identification and RMAP. The implementation scheme has been established through scientific satellite projects as ASTRO-H [1].

2.1 UTILIZATION OF RMAP PACKET FORMAT

One interesting thing about RMAP is that it has transaction control capability within itself. RMAP read and write reply packets have several characteristics for determinism. They have Cyclic Redundancy Check (CRC) code and Status Field in RMAP layer. Then also have End of Packet (EOP) and Error End of Packet (EEP) code in SpaceWire layer. Fault detection function can be implemented without modifying SpaceWire and RMAP protocol.

2.2 UTILIZATION OF RMAP TRANSACTION SEQUENCE

Write action and Read action are specified in RMAP, as well as transaction identifiers in RMAP read reply and write reply packets. Therefore assured transmission can be achieved without any modification on RMAP specification.

3 PROTOCOL LAYER FOR SCHEDULING AND ASSURED TRANSMISSION

The conventional protocol used in Japanese scientific satellites data handling system has the same capability as RMAP. Physical layer and lower portion of data-link layer are dedicated, whereas we could replace the layers with SpaceWire and RMAP. It is possible to maintain the scalability as wide variation on configuration and size using SpaceWire and RMAP, which is required for those satellites, and NEXTAR standard platform inherits the capability.

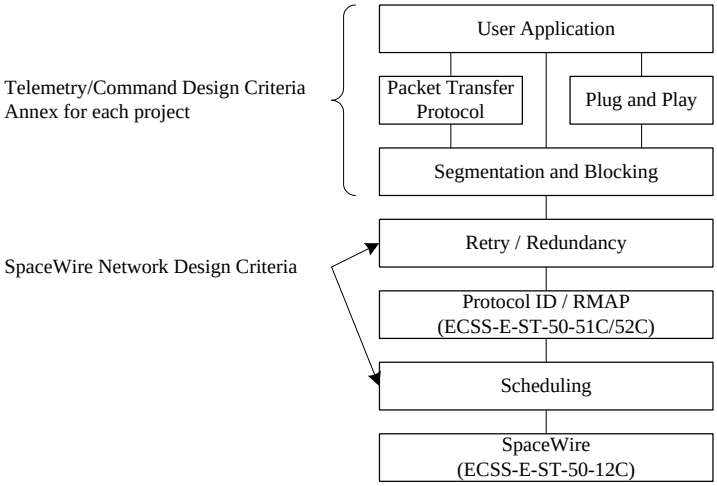


Figure 1 Protocol Stack Layer for Determinism

In order to add deterministic characteristics, scheduling protocol layer is added between SpaceWire and RMAP layer [2], [3]. SpaceWire Time-Code is utilised for the definition of time slots in this protocol layer. Latency condition is specified in the layer in order to carry on successive transactions in one time slot. Assured transmission is realized by adding retry and redundancy capability on RMAP protocol layer. The inherent RMAP characteristics described in section two is exploited in order to implement the assured transmission capability without any modification on RMAP. The additional protocol stacks are shown in figure 1. The specification for the protocol is provided as NEXTAR's SpaceWire Network Design Criteria.

Segmentation and blocking capability for large amount of data transmission are provided between User Application layer and Retry / Redundancy protocol layer. The specification for the protocol is established in JAXA as Space Monitor & Control Protocol (SMCP) [4].

4 COMMUNICATION SERVICES

Telemetry and command handling functions are realised through communication services, which utilize transaction capability inherent in RMAP packet formats.

Distribution service comprises three communication services. Command distribution service performs retransmission of an RMAP write command packet in case of detecting error status in an RMAP write reply packet. Data distribution service doesn't perform retransmission even if an error status was found in an RMAP write reply packet. Time distribution service doesn't use an RMAP write reply packet, and no retransmission occurs. The time value sent through the time distribution service is different from SpaceWire Time-Code, and the usage of the value is dependent on the system requirement.

Collection service comprises five communication services. A user request code can be transmitted through user request service. An RMAP initiator reads a user request code through the user request service using an RMAP read command. The initiator sends request acknowledge for the user request with an RMAP write command packet, and can perform retransmission in case of detecting an error in an RMAP write reply packet. An initiator collects a variable length Space Packet or a fixed length raw data packet through master triggered collection service using an RMAP read command. No retransmission occurs through the master triggered collection service. An initiator collects essential house keeping (HK) telemetries through essential HK collection service. A fixed length telemetry packet is collected through the service, and the telemetry packet is to be collected even in the system safe hold mode. On-demand data collection is carried on through guaranteed user triggered collection service or non-guaranteed user triggered collection service. An initiator performs re-transmission through the guaranteed user triggered collection services, and sends acknowledge when it receives telemetry data successfully. The target must keep the same telemetry data on its buffer memory until it receives acknowledge from the initiator. No retransmission nor acknowledge transmission occurs through non-guaranteed user triggered collection.

Each communication service is associated with pre-determined interface buffer memory address, so as to distinguish each communication service with the associated memory address. The memory address map is called as standard RMAP memory

address, and the memory address map is maintained for the plug and play capability. The memory map is shown in figure 2.

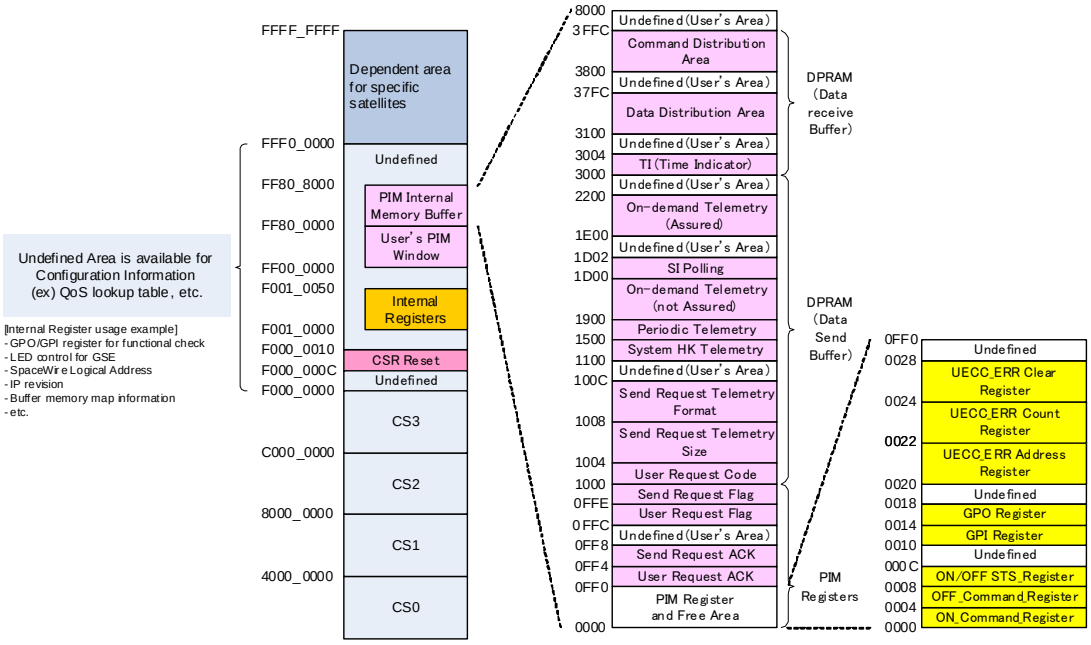


Figure 2 The standard RMAP memory address on NEXTAR bus.

5 CONCLUSION

Scheduling and assured transmission capability for determinism are realised without any modification on SpaceWire and RMAP specification. The NEXTAR satellite bus employs the implementation scheme and to be planned next year [5].

6 REFERENCES

1. Tadayuki Takahashi, et al., ”The ASTRO-H Mission”, SPIE, 7732, 77320Z, 30 July 2010.

2. Takahiro Yamada, and Tadayuki Takahashi, “Standard Onboard Data Handling Architecture Based on SpaceWire”, International SpaceWire Conference 2008, 4-6 November 2008, p.253-256.

3. Takahiro Yamada, “Proposal for Defining Standard Services Over SpaceWire - Revision A -”, The sixteenth SpaceWire working group meeting ESTEC, Netherlands, 22 March 2011.

4. Takahiro Yamada, “Spacecraft Monitor & Control Protocol (SMCP)”, GSTOS 200, 15 September 2009.

5. Toshiaki Ogawa, Yusuke Kobayashi, Shoichiro Mihara, Koichi Ijichi, and Hideyuki Hamada “Outline and Progress of ASNARO (Advanced Satellite with New System Architecture for Observation) Satellite System”, 8th IAA Symposium on Small Satellites for Earth Observation, Berlin, Germany, 04 – 08 April 2011.

A DETERMINISTIC SPACEWIRE NETWORK ONBOARD THE ASTRO-H SPACE X-RAY OBSERVATORY

Session: Missions and Applications

Short Paper

Takayuki Yuasa, Tadayuki Takahashi, Masanobu Ozaki, Motohide Kokubun

Institute of Space and Astronautical Science/JAXA, 3-1-1 Yoshinodai, Chuou-ku, Sagamihara, Kanagawa 252-5210, Japan

Masaharu Nomachi

Osaka University, 1-1 Machikaneyama, Toyonaka, Osaka 560-0043, Japan

Hiroki Hihara

NEC TOSHIBA Space Systems, Ltd., 1-10 Nisshin, Fuchu, Tokyo 183-8551, Japan

Kazuyo Mizushima, Takashi Kominato, Kuniyuki Omagari

NEC Corporation, 1-10 Nisshin, Fuchu, Tokyo, 183-8551, Japan

Kazunori Masukawa

Mitsubishi Heavy Industries, Ltd., Nagoya Guidance & Propulsion systems works, 1200 Higashi-Tanaka, Komaki, Aichi 485-8561, Japan

E-mail: yuasa@astro.isas.jaxa.jp, takahasi@astro.isas.jaxa.jp, ozaki@astro.isas.jaxa.jp, kokubun@astro.isas.jaxa.jp, nomachi@lns.sci.osaka-u.ac.jp, h-hihara@bc.jp.nec.com, kazunori_masukawa@mhi.co.jp

ABSTRACT

ASTRO-H is the space X-ray astrophysical observatory which is scheduled to be launched in 2014, and has been constructed by an international collaboration lead by JAXA. SpaceWire and RMAP compose the fundamental infrastructure of the highly redundant data-handling network of the satellite. For constructing a dependable and deterministic network, a set of constraints are designed and applied to communications over the network. In the paper, the concept of the constraints is described followed by a short report on a SpaceWire integration test joined by components developed under the constraints by different companies.

1 INTRODUCTION

The ASTRO-H satellite [1] is one of the very first missions in Japan that fully utilizes SpaceWire as an onboard data-handling infrastructure. As of 2011, engineering models of its subsystems have been fabricated and tested, and flight model productions will start aiming integration in 2012-2013 followed by a launch in 2014.

Almost all of onboard subsystems of ASTRO-H such as the command/data handling system, the attitude control system, and four types of X-ray/gamma-ray telescope instruments are connected to the SpaceWire network using a highly redundant topology [2]. Figure 1 shows a topology of the onboard network with representative components being only illustrated. The number of physical SpaceWire links between

components exceeds 140 connecting ~40 separated components (i.e. separated boxes), and there are more links in intra-component (intra-board) networks.

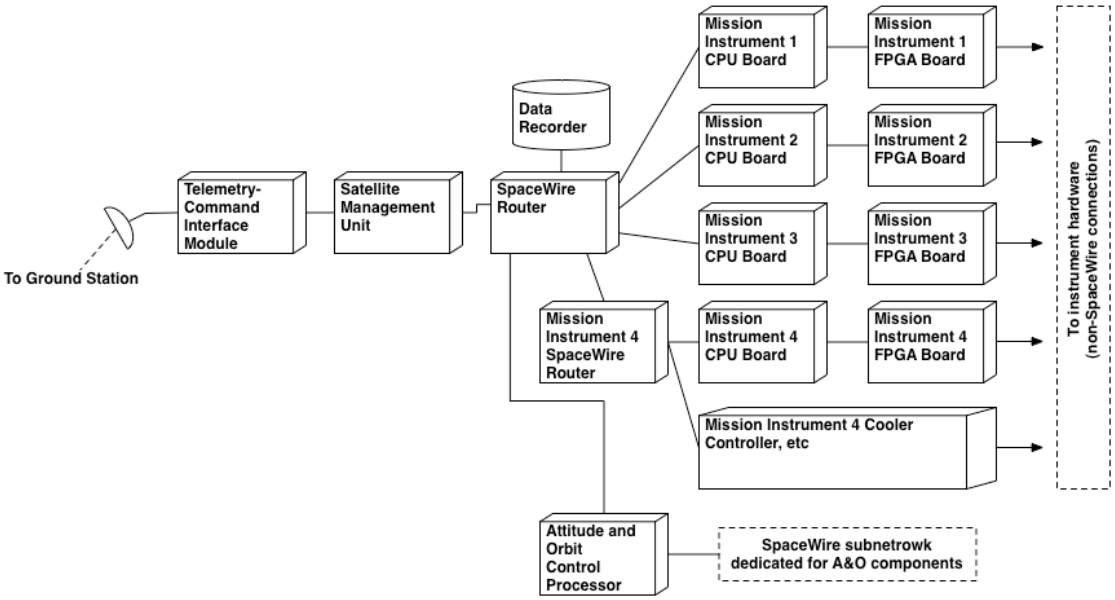


Figure 1 A schematic diagram of the onboard SpaceWire network of ASTRO-H.

2 RULES APPLIED FOR DETERMINISM

The onboard network of ASTRO-H is very large as explained above, and therefore, strict network management policy is necessary for the whole system to operate in a deterministic manner without suffering from congestions and unexpected delays of packet deliveries. For achieving this, several rules described below are applied when designing the network and its operation scheme.

2.1 SINGLE MASTER: THE SATELLITE MANAGEMENT UNIT

The all transactions in the network are controlled by the central master called Satellite Management Unit (SMU). The SMU is responsible for distributing commands sent from the ground stations to onboard components, and collect house-keeping data from them using RMAP Write and Read, respectively. In addition, the SMU can be configured to detect an abnormal state of a certain component from house-keeping data and autonomously send a series of commands to control the component. Data output from the scientific mission instruments which are relatively large compared to the house-keeping data are collected by RMAP Read transactions initiated by the Data Recorder (DR) following data collection schedule planned by the SMU. Since the transactions from the DR are fully managed by the SMU, the DR acts like a delegate of the SMU thus leaving the SMU as the single master. This single master configuration well simplifies allocation of time slices described below, and helps to qualitatively estimate achievable bandpass.

2.2 TIME SLICING BASED ON TIME CODES

The real time is divided into 64 time slots using 64-Hz time codes emitted by SMU. All types of transactions from the SMU and the DR, such as the command distribution, the house keeping data collection, the scientific mission data collection, and the

auxiliary polling of request flags, are performed in any of specifically allocated time slots. Figure 2 summarizes transaction types allocated for individual time slots. The spacecraft bus system (e.g. the SMU and the DR) and the mission instruments shares this allocation table, and the latter updates for example house-keeping data stored in registers or memory after particular time slots where the SMU RMAP-reads the data.

In order to gain the bandpass under the moderate time code frequency, to perform multiple transactions in one time slot is allowed. The present design allows a packet to be transferred over the network crossing the boundary of two time slots, i.e. there is no explicit time-of-silence which is a technique sometimes used for clearing the network.

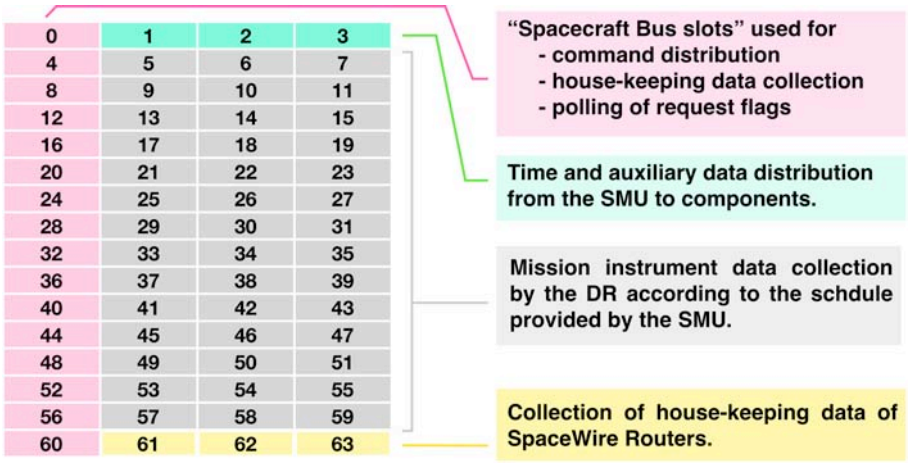


Figure 2 Time slot allocation used in ASTRO-H.

2.3 LIMIT ON THE PACKET LENGTH AND THE RESPONSE LATENCY

The maximum packet length is limited to 1024 bytes including the RMAP header and an additional header part defined in the mission. This well limits the maximum blocking time in the wormhole routing path in SpaceWire routers.

An RMAP target nodes should reply to received RMAP commands as soon as possible to maximize an achievable bandpass. Although hardware-implemented RMAP target nodes can relatively quickly respond to commands, there are several software RMAP targets especially in mission instrument electronics. Considering these conditions, response latencies are defined depending on the transaction types (HK collection, mission data collection, etc) starting from ~500us to a few ms. If the SMU does not receive an RMAP reply from a certain component within defined timeout duration, it cancels the transaction.

SpaceWire routers used in ASTRO-H are equipped with watch-dog timers to prevent a dead lock of the routers. When a packet occupies a certain wormhole for specified time duration, the packet is killed, and the router tries to recover from the anomaly.

2.4 BANDPASS

In ASTRO-H, the highest throughput is required for transferring mission instrument data to the DR. Since the DR can perform 70 transactions in three consecutive time slots (e.g. time slots 5,6, and 7, or 41, 42, and 43) and there are 14 sets of these slots

in the present time-slot allocation, 980 packets can be transferred from 9 mission instrument electronics (CPU boards) used in the 4 instruments. Referring to the packet length limitation, this translates into ~980 kB/s. Note that link rates of individual nodes are rather heterogeneous, ranging from 10 to 50 MHz.

3 TESTS OF THE CONCEPT

To integrate components developed by several manufacturers smoothly, the ASTRO-H project arranged three-step preliminary tests that should be done in manufacturer sites; SpaceWire-layer test done by STAR-Dundee's SpaceWire Conformance Tester (step 1), RMAP-layer test examined by STAR-Dundee's RMAP Conformance Tester (step 2), Telemetry/Command-layer test done with an SMU simulator provided by the spacecraft bus team (step 3).

The mission instrument electronics developed by Mitsubishi Heavy Industries and the spacecraft bus components from NEC/NEC-Toshiba Space Systems have experienced the preliminary tests, and then joined the first integration test held in JAXA in June-July 2011. Thanks to the pre-tests, the components are successfully communicated using RMAP without having a big problem, and it is revealed that basic telemetry/command functionalities work as designed. After completing implementation of full functionalities of the SMU software and the DR hardware logic, the second integration test will be held in October 2011 to examine the mission instrument data collection where cooperation of the two is essentially important.

Figure 3 An overview of the first SpaceWire integration test.

4 REFERENCES

1. Tadayuki Takahashi, "The NeXT mission", SPIE meeting 7011 "Space Telescopes and Instrumentation II: Ultraviolet to Gamma Ray 2008", Marseille (2008).

2. Masanobu Ozaki et al., "SpaceWire-driven architecture for the ASTRO-H satellite", International SpaceWire Conference 2010, St. Petersburg, Russian Federation, June 22-24, 2010.

APPLICATION OF SPACEWIRE TECHNOLOGY IN HYDROACOUSTICS

Session: SpaceWire missions and applications

Short Paper

Petr Eremeev, Sergey Kozyrev, Viacheslav Grishin

SUBMICRON, 2, 4, 4806 Street, Zelenograd, Moscow, Russia

E-mail: epm@se.zgrad.ru, seyoza@mail.ru, grishin@se.zgrad.ru

ABSTRACT

This article covers the theme of hydroacoustic complexes construction using the SpaceWire technology. The task of large volume data arrays collection and commutation between receiver and handler is solved by means of this technology. High bandwidth, low power consumption, channel reliability and communication features permit to develop integrated system of getting high-quality image. The structure of the system intercommunication network is based on CompactPCI PICMG 2.16[(1)] backplane. Such backplane has differential links that agrees with SpaceWire transmission standard. The connection is realized in double-star topology that allows organize the route reservation, thereby the hardware reliability increases. This project presents the signal preprocessing equipment that performs collecting and compacting data received from hydroacoustic antennas.

1 INTRODUCTION

"Submicron" Company takes the leading position in Russian aerospace equipment development, but that is not an only area of Company's work. One of the actively upcoming courses is presented with hydroacoustics. The new concept of the hydroacoustic complexes construction based on the SpaceWire technology was developed to perfect already engineered systems and to solve the main tasks of the hydroacoustics. Utilization of this technology permits to decrease power consumption, to solve synchronization problem and to create full-connected network at the protocol level.

The principle of the high-precision hydroacoustic complexes development is based on the multipoint signal collection by means of the multielement antenna array. The quality of the result primarily is affected by the preprocessing equipment of hydroacoustic signals. The purpose of the work is to create the system capable to link a number of input analog channels with information processing system.

2 PROJECT DESCRIPTION

The project of the hydroacoustic complex is realized as a set of Input and Compacting crates (Figure 1). The Input crate executes digitization and compaction (the first level) of data received from antennas and further transmission of compacted arrays to the handler. The Compacting crate is set behind the Input crates. It executes the second

level compaction and organizes intercommunication between elements of the complex. The hierarchical system construction permits to increase quantity of the input analog channels adding more Input crates.

Each Input crate is meant for 224 input analog channels. It consists of 14 Modules of Input Hydroacoustic Signals (MIHS) and two Modules of Compacting Hydroacoustic Signals (MCHS). The modules are founded on the integrated chips developed by "Elvees"[2][3]. All MIHS and only one MCHS can work in each point of time and at the same time the second MCHS is reserved. The modules redundancy permits to increase reliability of hydroacoustic complex in case of one of the networked modules failing. The intercommunication between modules in the single crate realizes with SpaceWire interface using standard backplane CompactPCI PICMG 2.16[1].

The MIHS module (Figure 2) is assigned for input analog signals digitization and data transmission to SpaceWire channel. Eight ADC microcircuits are set on the MIHS. Each ADC simultaneously operates with two analog channels. Therefore the MIHS board has 16 input analog channels in whole. The ADC microcircuits are sequentially joined through SpaceWire interface that is used for control information exchange and digitized data issue.

The number of the issuing digitized data modules is 14, and the rate of the packet transmission does not exceed 25 Mbps from each MIHS in whole. Hence there is the necessity of the MCHS modules (Figure 3) utilization, where it performs data array collection and compaction. The MCHS is designed on the Concentrator microcircuit that commutes and accumulates the data from 14 MIHSs and issues the tightened flow through SpaceWire or RapidIO interface. At these conditions the average speed of the data issue after the first level compaction does not exceed 250 Mbps. During operation the data filtering and compression does not occur, while the compactness and speed of the exchange through SpaceWire channels increase. I.e. all received digitized data transmits to the primary processing system that finally increases the accuracy of the data processing result.

The modules are made in the universal design of the 6U standard. The intercommunication between models within one input crate executes through CompactPCI PICMG 2.16 backplane. The MIHS modules are Node Boards, and the MCHS are Fabric Boards. The modules are linked using differential pairs topology of the backplane. This network operates in the full-duplex mode with 100 Ohm wave impedance. The bandwidth of these lines can be up to 4 Gbps. The connection is realized in a double-star topology as the main route of information packages transfer from MIHS to MCHS and transfer of control information from MCHS to MIHS. Thus, the boards are linked using SpaceWire technology implemented with the communication lines of the backboard.

3 CONCLUSION

The main idea of the SpaceWire technology application in this project consists in the integration of all system components in the single communication area of the data packets and control information transmission using standard backplane. Thus, the task of high data volume collection in one processing machine is solved as one of the main tasks of the hydroacoustics. The selection of SpaceWire interface as the primary method of the information exchange is also caused by the high bandwidth and

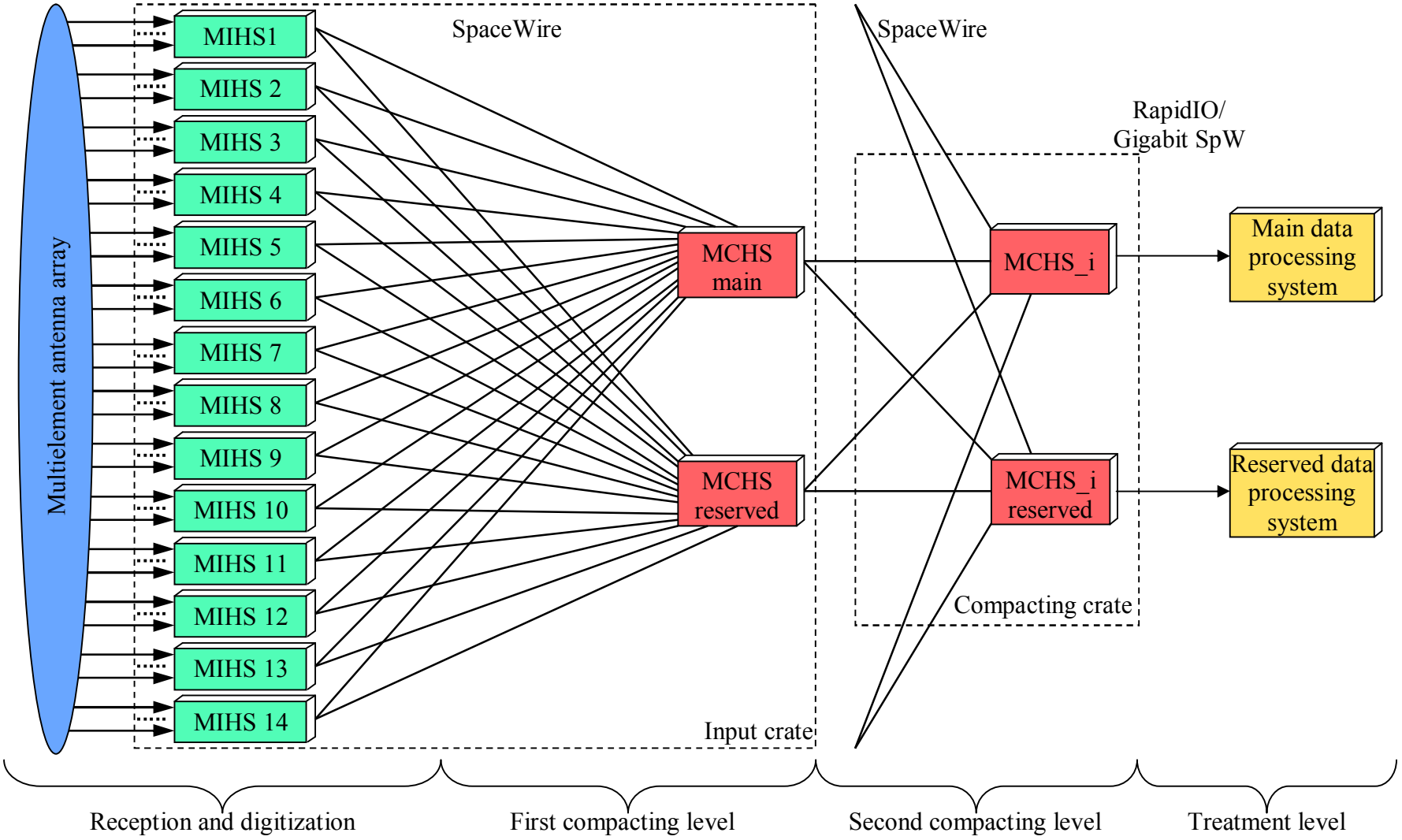


Figure 1 Structure of Hydroacoustic Complex

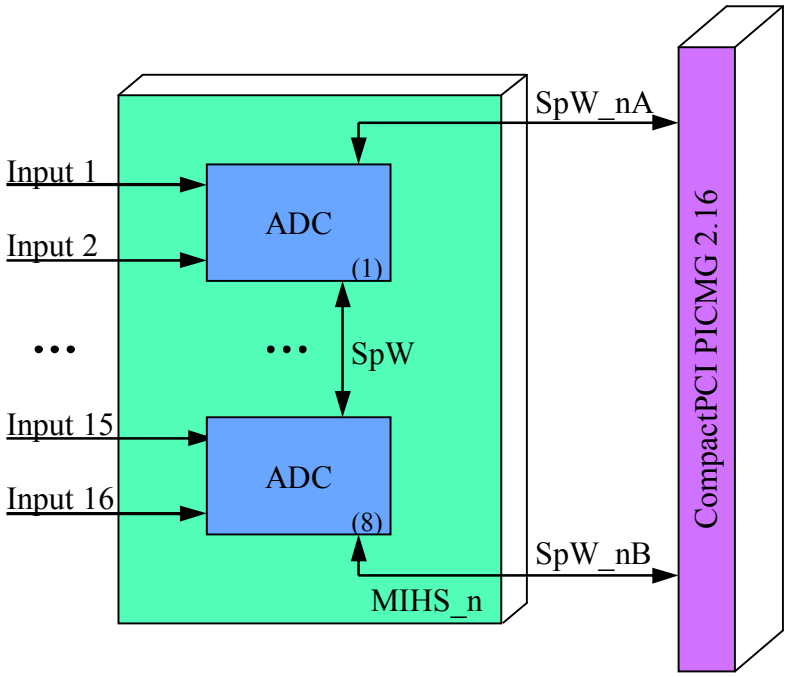


Figure 2 Module of Input Hydroacoustic Signals

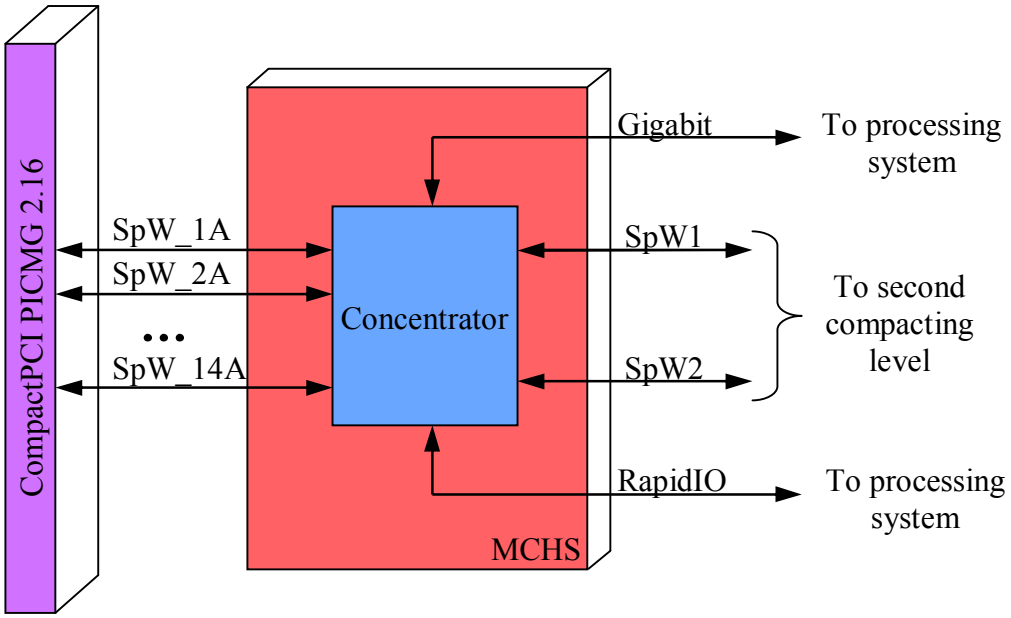


Figure 3 Module of Compacting Hydroacoustic Signals

reliability of the channel work, as well as by its compatibility with ready to use developments.

4 REFERENCES

1. PCI Industrial Computer Manufacturers Group, “CompactPCI”, Packet Switching Backplane, PICMG 2.16 Draft 0.9.1, July 3, 2001.

2. www.elvees.com

3. www.multicore.ru

SPACEWIRE, A BACKBONE FOR HUMANOID ROBOTIC SYSTEMS

Session: Missions and Applications

Short Paper

Nickl Mathias and Jörg Stefan, Bahls Thomas, Nothhelfer Alexander, Strasser Stefan

Institute of Robotics Mechatronics, German Aerospace Center

E-mail: mathias.nickl@dlr.de

ABSTRACT

The DLR Hand Arm System is an anthropomorphic system with 52 actuators and 430 sensors of different types. In order to maintain good performance the application must have the most direct access to all actuators and sensors. Therefore, a SpaceWire network connects FPGAs and CPUs and acts as real-time communication backbone. This publication focuses on the SpaceWire protocol implementation and the dedicated extensions that are defined for that system.

1 INTRODUCTION

The DLR Hand Arm System (see Fig. 1) is an anthropomorphic system that is aimed to reach its human archetype regarding size, weight and performance. It features intrinsic compliance implemented as variable stiffness actuation [1].

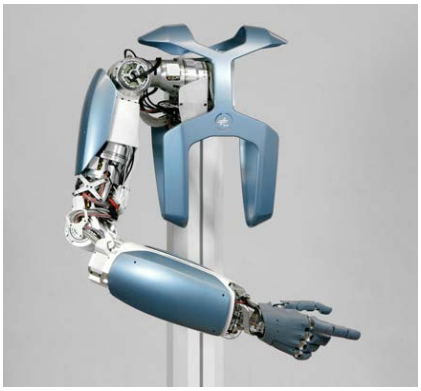


Fig. 1. The DLR Hand Arm System

The hand arm system has in total 26 DOF, thereof 19 DOF in the hand, 2 DOF in the wrist, and 5 DOF in the arm. To implement all those DOF, the hand arm system comprises 52 actuators and 430 sensors of different types. To operate that many actuators and sensors precisely for a certain control application the complexity of the system needs to be hidden from application designers. On the other hand, in order to maintain good performance the application must have the most direct access to all actuators and sensors.

In other words, a valuable means of abstraction with only minimal execution overhead is required. This is the task of the Computing and Communication Architecture. It incorporates the operating software and the computing and communication infrastructure of the DLR Hand Arm System. The aim is to provide a convenient high-level hardware abstraction that still allows high-performance feedback control with cycles beyond 1 kHz.

To balance the opposing requirements of flexibility and high integration, the DLR Hand Arm System's computing and communication platform is laid out hierarchical: At the top are general purpose, commercial-of-the-shelf (COTS) components. The footprint decreases towards the bottom end which is defined by the dedicated physical interfaces of sensors and motors. The available computing power and communication bandwidth decreases along with the decreasing footprint. A modular layout on each level together with the aggregation of components on successive levels by the means of suitable communication creates the desired platform flexibility (see Fig. 2). This hierarchy is not driven by a functional separation but only by the requirement of small footprint sizes at the physical interfaces. The functionality of an application can be flexibly mapped onto this hierarchy as required.

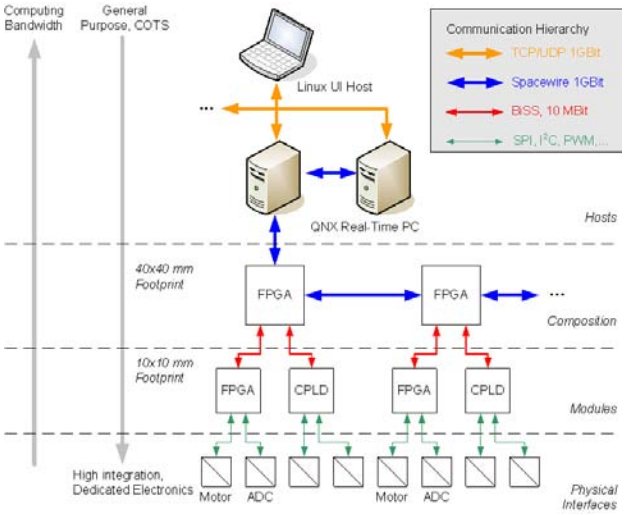


Fig. 2: The DLR Hand Arm System's hierarchical computing and communication platform

A SpaceWire network provides the necessary flexibility within the architecture and acts as a real-time communication backbone that connects FPGAs and CPUs. This publication has the focus on the SpaceWire protocol implementation and the dedicated extensions that are defined for the DLR hand arm system. A more detailed description of the entire Communication and Computation architecture is given in [2].

2 THE PROTOCOL STACK

2.1 PHYSICAL LAYER, CHARACTER LAYER, AND LINK LAYER

Inspired by the IEEE 1355 specification for fibre optical links as well as the Gigabit Ethernet and the FiberChannel specifications, the character-layer is realized with 8b10b [4] encoding. Therefore, a commercial GigE physical-layer interface circuit from Texas Instruments (TLK1221) is used, which has a dedicated ten-bit interface suitable for 8b10b encoding. The 8b10b encoding is implemented on FPGAs. This design allows SpaceWire links with data-rates of 1Gbit/sec and heterogeneous networks with fiber and copper.

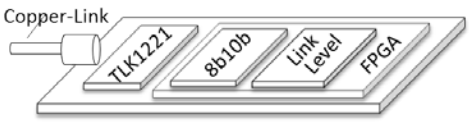


Fig. 3: SpaceWire Link with 8b10b encoding

The link-layer implementation meets the SpaceWire specification. It is adapted to 8b10b encoding by mapping the SpacewWire escape-characters to the 8b10b-K.Chars (see Table 1]). This implementation is flexible, since the link-layer implementation can be used for different character-layers. But the broken-link propagation with timeout is not efficient. A dedicated SpaceWire link-layer specification for 8b10b would be useful.

ESC	KChar
IDLE	K28.5
TC	K28.1
FCT	K28.2
EEP	K28.3
EOP	K28.4
NULL	K28.6

Table 1: ESC to KChar mapping

2.2 NETWORK LAYER

Table 2 shows the links and switches that are developed for the backbone of the DLRs Hand Arm System:

Name	Platform	Comment
SW-Switch	QNX	SpaceWire Crossbar Switch for QNX with optional LUT for logical address resolution
HW-Switch	FPGA	SpaceWire Crossbar Switch for QNX with optional LUT for logical address resolution
HW/SW-Switch	FPGA and QNX	Runs on in-house PCIe interface card. Routes packets in dedicated DMA-buffers or HW-links. Allows high performance packet routing with minimum latency.
Copper Link	FPGA to FPGA	See 2.1
HW-Link	FPGA	Connects HW-Switches and/or HW-Nodes within an FPGA. Optional FIFO allows to buffer characters or packets. If packet-buffering is switched on, EEP-packets can be deleted.
IPC-Link	QNX	connects SW-Switches and/or SW-Nodes
HW/SW-Link	FPGA and QNX	connects HW-Switches or HW-Nodes to SW-Switches or SW-Nodes
Copper2Fiber		Transceiver for seamless connection of fiber and copper networks

Table 2: Building blocks for SpaceWire backbone

2.3 TRANSPORT LAYER

The Datagram Protocol defines a simple non-reliable connection between Sink and Source. A Datagram is a single Spacewire Packet. The payload of the datagram is validated by crc (see Fig. 4).

Fig 4: Datagram Protocol

The RequestResponse Protocol is a transmission control protocol optimized for the implementation on FPGAs. The payload is validated by a crc. The process flow is validated by a configurable timeout (see Fig 5).

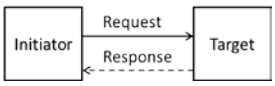


Fig 5: RequestResponse Protocol

The timeout control is located at the Initiator. Hence, the footprint on the target side is reduced. A detected timeout triggers an error-cycle, which is repeated until the Target acknowledges the error (see Fig. 6).

Datagram Sink and Source as well as Initiator and Target are SpaceWire Nodes. Source, Initiator, and Target store the address of their peer Node in a lookup table. A Node Configuration Protocol allows the configuration of this peer-address-LUT during runtime (see 2.4).

Fig. 7 shows the SpaceWire-packets of Datagram Protocol and RequestResponse Protocol.

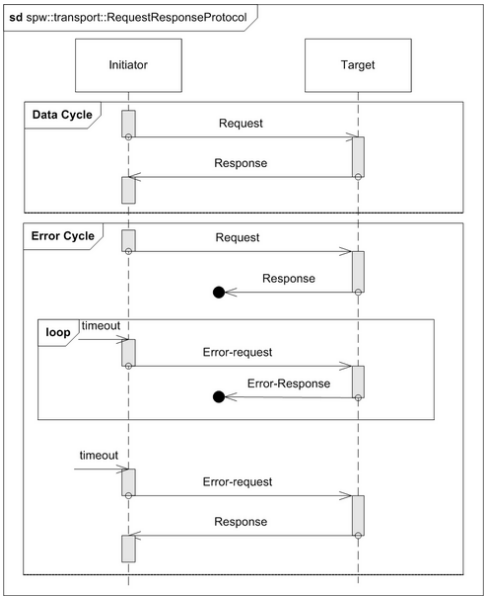


Fig 6: RequestResponse Protocol

2.4 APPLICATION LAYER

Switches with dynamic logical address mapping are configured by the Switch Configuration Protocol. The Configurator is an independent Node that configures the SpaceWire network. If the configuration has failed, the response packet is determined by EEP. Thus, configuration errors yield a Configurator timeout. Analogous, the Configurator configures the lookup tables of Nodes (see 2.3) by the Node Configuration Protocol (see Fig. 7).

Furthermore, a configurable Test Suite is available. A dedicated Test Node, which can act as Sink, Source, Initiator, or Target, generates periodical or random network traffic.

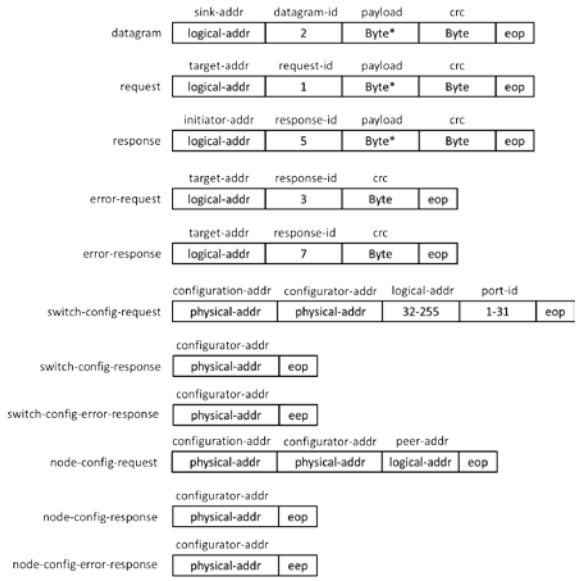


Fig 7: SpaceWire Packets for Datagram, RequestResponse, Switch Configuration, and Node Configuration Protocol

3 CONCLUSIONS

The concepts of FIFO channels and wormhole packet routing of the SpaceWire specification [3] combined with GigE physical-layer circuits results in a valuable communication platform for complex applications that require hard real-time. In [2] is shown that the Hand Arm System operates with control sample rates of 3 kHz and latencies below 333 us.

Especially the extended communication bandwidth of 1 Gbit/sec and the determinism (for known network topologies) make SpaceWire to be a good choice for high performance signal processing, since there is still no common alternative for deterministic communication beyond 1 GBit.

Beyond that, dynamic network configuration and the configuration of connections (i.e. peer-address) by an independent Configurator is a scalable solution with small footprint and a high degree of flexibility.

4 REFERENCES

1) M. Grebenstein et al., “The DLR hand arm system”, Proc. IEEE International Conf. on Robotics and Automation, April 2011.
2) Stefan Jörg, Mathias Nickl, Alexander Nothhelfer, Thomas Bahls, Gerd Hirzinger, “The Computing and Communication Architecture of the DLR Hand Arm System”, IEEE International Conf. on Intelligent Robots and Systems, 2011, tbp.
3) ECSS E-50-12A SpaceWire - Links, nodes, routers and networks, European Cooperation for Space Standardization (ECSS), http://spacewire.esa.int, 2003.
4) A. X. Widmer, P. A. Franaszek, “A DC-Balanced, Partitioned-Block, 8B/10B Transmission Code”, *IBM Journal of Research and Development, (1983) Volume 27, Number 5, Page 440*

Exhibitors

4LINKS

- 4Links test equipment is the de-facto SpaceWire reference, with unparalleled maturity in our design and an unparalleled record of finding errors, and providing the information to correct them;
- The family includes bridges, diagnostic interfaces, routing switches, and monitors, a time interface (IRIG-B) plus an RMAP responder to give hardware response times - all controlled from a single (possibly remote) PC;
- Products interface to Ethernet and Internet, able to be interfaced with virtually any computer, any OS, any where;
- All products are available with connectors for synchronization and triggers, so that multiple test units can be synchronized and recordings time tagged consistently between different computers and discs;
- 4Links test equipment helps to reduce cost and delay by enabling users to detect bugs that other methods miss, and by providing information to fix those bugs where other methods fail. See 4Links news about a Tutorial on SpaceWire Test on the Monday preceding the Conference.

AEROFLEX

Aeroflex Colorado Springs is a supplier of integrated circuits and custom circuit card assemblies. We supply a broad range of standard products for HiRel applications including a LEON 3FT microprocessor, logic, FPGAs, memories, serial communication interfaces for MIL-STD-1553, 1773, Clocks, an LVDS family of products and our SpaceWire products - Transceivers, Protocol IP, Routers. Our RadHard-by-Design Digital and Mixed-Signal ASICs handle design complexities up to 3,000,000 usable gates.

Aeroflex Gaisler is a provider of SoC solutions and IP-cores for exceptionally competitive markets such as Aerospace, Military and Commercial applications. The Aeroflex Gaisler's IPcores
consist of user-customizable 32-bit SPARC V8 processor and floating-point-unit cores, SpaceWire cores, peripheral IP-cores and associated software and development tools. Aeroflex Gaisler solutions help companies develop application-specific SoCs that are highly competitive for customer specific applications. Gaisler Research's personnel have extended design experience, and have been involved in establishing standards for ASIC and FPGA development.

AXON' CABLE

Axon' Cable manufactures wires, cables, cable assemblies and connectors for high tech applications including space, aeronautics, medical, automotive and electronics. The consolidated turnover in 2011 amounts 100 million €, 60% of which is achieved through export. The headquarters of the company is situated in France (100 km east of Paris) and employs 1500 staff world-wide in 11 subsidiaries across Europe, America and Asia.
Axon' Cable has been involved in many space projects such as the ISS, leo and geo satellites, rocket launchers including Ariane 5.

The group offers various types of products for space applications:

- ESCC wires and cables, aluminum round cables for power distribution in satellites.
- Bus bar for power distribution in satellites.
- MIL-STD-1553 databus cables, couplers and connectors for digital transmission systems.
- High data rate links for Voice-Data-Image transmission used in on-board electronics (SpaceWire, IEEE1394, Ethernet, Fibre Channel, etc).
- Custom designed products.

BAE SYSTEMS

BAE Systems is a global defense, security, and aerospace company, delivering a full range of products and services for air, land, and naval forces, as well as advanced electronics, information technology solutions, and customer support services. BAE Systems' Space Products and Systems (SPS) division specializes in radiation-hardened electronics and space applications, developing and producing a wide variety of space products from radiation hardened components (processors, ASICs, memories, FPGAs, Spacewire routers and interfaces) and single-board computers solutions, to complete system payloads. Our facilities are accredited as DoD Category 1A Trusted, covering design, wafer foundry, packaging/assembly, and test services, and our space product portfolio is QML qualified to MIL-STD specifications and test methods. With more than 600 computers in space, including the 16-bit GVSC1750, 32-bit RAD6000®, and the RAD750® family of products, BAE Systems space computers and electronics have logged over 5,000 years in orbit. For more information, please visit www.RAD750.com.

GLENAIR – MINIATURIZED CONNECTORS AND CABLES

Glenair manufactures ultra-miniature interconnect solutions for high-performance applications such missile systems, satellites, and fighter-jets. Our innovative contacts, connectors and cables are used in air and space platforms that require reliable performance as well as miniaturized packaging. Glenair is the world's largest manufacturer and supplier of both mil-qualified and commercial Micro-D and Nanominiature connectors in wired and unwired space-grade formats. We also offer turnkey flex circuitry assemblies as well as space-grade wire harnesses terminated to our high-availability connector products.

Glenair Inc

1211 Airway
Glendale
California
USA 91201-2497
www.glenair.com

Contact details re: Micro D space wire connectors and cables.
Fred Van Wyk, Product Manager :
Phone: +1 818 247 6000
fvanwyk@glenair.com

Ross Thomson, Business Development Manager (Europe):
Phone: + 44 1623 638114
Cell: +44 7711 029 715
rthomson@glenair.com

JAPAN SPACEWIRE USER GROUP

Web: https://galaxy.astro.isas.jaxa.jp/SpaceWire/
Japan SpaceWire User Group is a consortium formed by Japanese space agencies, JAXA and USEF, and multiple companies that develop and use SpaceWire technology. The consortium aims to promote SpaceWire and satellite design based on SpaceWire to wider users, and has been developing and releasing SpaceWire development environment such as the SpaceCube computer and the SpaceWire-to-GigabitEther converter (Shimafuji/JAXA) for ground testing, and the SpaceCube2 (NEC) and SpaceCard (MHI) onboard computers with SpaceWire capabilities. An open-source SpaceWire codec IP core is also available from the consortium for free.

NEC CORPORATION

www.nec.com

NEC Corporation is one of the world's leading providers of Internet, broadband network and enterprise business solutions dedicated to meeting the specialized needs of a diversified global base of customers. NEC delivers tailored solutions in the key fields of computer, networking and electron devices, by integrating its technical strengths in IT and Networks, and by providing advanced semiconductor solutions through NEC Electronics Corporation.

The NEC Group employs more than 140,000 people worldwide. For additional information, please visit the NEC Web site at: www.nec.com.

SOUTHWEST RESEARCH INSTITUTE

Southwest Research Institute® (SwRI®) was founded in 1947 as a public service scientific corporation to provide contract R&D to both industrial and government clients. The Institute provides extraordinarily technical capabilities through 10 technical operating divisions, with approximately 3300 staff members and gross annual revenue of $540 million.

SwRI's Department of Space systems has a long and distinguished track record of producing high quality, high reliability spacecraft avionics for NASA, DoD, ESA, and commercial space missions. Since the first SC-1 spaceflight computer was developed in 1979, SwRI has developed hardware for over 53 space flight missions without a single on-orbit failure. The track record of the last 32 years is a product of a strong commitment to support the current and future needs of the space community. SwRI is recognized as one of the leaders in space instrument design and development, command and data handling (C&DH) systems and mission management.

STAR-Dundee Ltd is dedicated to the development and advancement of SpaceWire, providing expert support to users and developers of SpaceWire technology.

Our products cover everything needed to design, develop, integrate and test SpaceWire sub-systems:

- Chips and industry leading IP cores: enabling our customers to develop their own flight subsystems and providing custom IP cores to fulfil specific customer needs
- Interface devices, Debug and Analysis Tools: enabling the development, simulation and testing of SpaceWire networks and devices
- Bespoke Design Services: Equipment and design of electronic circuit boards for custom requirements.
- SpaceWire Training: Onsite expert tuition direct from our experienced engineers, tailored to suit the customer

STAR-Dundee has the largest product line of SpaceWire test and development equipment of any manufacturer. We pride ourselves on the quality of our products and are continually enhancing their capabilities to meet the needs of our customers.

The STAR-Dundee team has leading expertise in all areas of SpaceWire technology. Our commitment is to help our customers to quickly and efficiently get up to speed with SpaceWire technology and to provide continued support through the full development life cycle.

Papers Indexed by Author

Author Surname A - J

Author Surname K - R

Author Surname S – Z