NEW APPROACH AND TECHNIQUES FOR TESTING AND DIAGNOSIS OF SPACEWIRE NETWORKS

Session: SpaceWire Test and Verification

Short Paper

Stéphane Davy, Jacky Rozmus, Matthieu Salanave,

SKYLAB Industries, 42 avenue du Général de Croutte, 31100 Toulouse, France

Frédéric Pinsard

DSM/IRFU, Bât 141, CEA Saclay, 91191 Gif sur Yvette Cedex, France

Mansour Talhaoui

Aerospace Services International Company, 3 rue du romarin, Taieb M'hiri, Tunis, Tunisia

E-mail: spacewire@skylab-corporate.com, pinsard@cea.fr, talhaouimansour@gmail.com

ABSTRACT

In this paper, we present a new testing approach and techniques for compliant SpaceWire[1] devices and networks. This approach is based on the emulation principle by using both physical and virtual devices. The conducted work has been done with the collaboration of **CEA** and **Aerospace Services International Company**.
Most of today's remaining issues and bugs in relation to test and integration are usually due to incompatible hardware, or simply, software misunderstandings between distant engineering teams, a common situation in space projects. The idea here is to introduce a new concept for emulating SpW networks based on physical equipments and/or virtual nodes and routers. Each of them could be located either in the same laboratory or at different locations around the globe. As a result, troubleshooting is expected to be more efficient and can be performed at early stages of the development, therefore ensuring successful flight modules.

After introducing the physical and virtual elements features which will act as SpW nodes for the network, the paper will detail the complete traffiController^{4SpW} software suite and how its architecture and characteristics can speed up network dimensioning, software testing and validation in a given heterogeneous SpaceWire topology. The software can actually be very flexible, allowing substantial freedom of use, not just for potential *4SpW* products users but also for SpW users without hardware equipment and theoretically for any type of SpW test equipment.

NETWORK PHYSICAL DEVICES

One main element of the test bench is the **PCI^{4SpW}** product: a PCI board with four SpW nodes, using CEA IP core. In addition to being able to transfer data with flexible configuration for each port, it provides standard and high-resolution time-codes capability. A SpW conformance testing feature is also present in the FPGA System on Chip, providing means to test robustness of any IP core. The PCI driver which is provided with the board can handle different data formats, depending on the project's performance and software constraints. Two main data transfer formats are available:

- A *short* format: one 9bit SpW characters is stored over a single 16bit word.
- ***SpaceWire Interpreted Protocol* format** (SIP):
  - *Data_8* format: limited to 256 values: one byte per SpW character.
  - *Data_32* format limited to FIFO size of hardware: 4 SpW characters per 32bit word.

Regarding memory capacity, the PCI board features a local and configurable 32K x 36bit SRAM memory dedicated to incoming data while a dedicated non-configurable 28KB of FPGA block-memory can be used for outgoing data. A PCI[4SpW] emulator has been implemented in order to anticipate the board delivery during SKYLAB early software prototyping, and can now act as a physical device emulator.
The other key element of the test bench is the **smartCable[4SpW]** product: a USB High-Speed single SpW node miniaturized device, overmolded with a male microD9 connector. It can act as node-to-node analyzer as well, using a dedicated plug (*spy mode*). It provides oscilloscope and logical analyzer capability, with possible buffering into the main 512Mbit DDR SDRAM. In addition, a custom LVDS eye diagram feature is available in the electronic device. The smartCable driver can handle enough bandwidth for most of the projects' constraints. During the smartCable prototyping validation, a throughput of 32MBytes per second uni-directionally was measured. SmartCable driver also provides implementation of hardware SIP format conversion for better processing efficiency: dedicated RTL modules actually ensures SIP to SpW decoding and SpW to SIP encoding in the smartCable System on Chip. Using dedicated routines of the API, the smartCable device also provides:

- **eye d iagrams** for both incoming and outgoing Data+/- and Strobe+/- differential lines. The Analog to Digital converter used for processing the analog signals is a dual 11-bit ADC with a 900MHz bandwidth, for potential LVDS measurements up to the highest standard transfer speeds. A low noise multiplexing front-end ensures very low disturbances regarding SpaceWire lines. The eye diagram is accessible in the API through direct picture format and dedicated routine.
- a **digital oscilloscope** function, providing useful serial data debug information.
- an **analyzer** function (in *spy* mode only). The additional *plug* of the smartCable (a female to female-female assembly which can be plugged to the male connector) reconfigures the FPGA of the USB bridge into receiving-only lines, providing a non-intrusive analyzer for two potential communicating nodes of, for example, a given set of space equipments.

CompatibleCable[4SpW] products have been used with 10-meter Ethernet cables in order to interconnect PCI devices at a transmit clock speed of up to 200MHz. A third possible test equipment for the test bench, the **PCI Express[4SpW]** board, could not be used for this set of testing because it was still under validation at the time of writing this paper. However, data transfers from PCI Express to SpaceWire have been successfully tested on the prototype model by CEA/IRFU. The test results for this board are promising and the board is set to provide high-performance transfer, especially because of the theoretical multi Gbps throughput of the PLX device associated with the performances of a Virtex 4 device, all two consistent with more demanding *ground and space* applications. A PCI Express[4SpW] emulator has been developed in order to anticipate the board availability and can act as a physical device emulator as well, as previously described for the PCI board.These three physical devices can be managed by the traffiController[4SpW] software suite, in order to be controlled and used as SpW nodes for transmitting and receiving data. RMAP, as described in ECSS-E-ST-11C, can be associated to each of these physical nodes.

NETWORK VIRTUAL DEVICE

In addition to this set of hardware, traffiController[4SpW] is also capable of managing virtual devices. These consist of the following elements, which use files for dump and load of data:

- **virtual node:** a SpW object-oriented element providing communication features, not related to any particular hardware, but dealing with data to and from a text file. Virtual nodes can communicate with real or emulated nodes from real or emulated hardware. They can also be virtually connected to one virtual router port (see hereafter).
- **emulated smartCable, emulated PCI, emulated PCI Express devices:** emulated object-oriented elements providing features equivalent to the real hardware, with degraded performances, also dealing with input and output files. These emulators provide emulated nodes for the SpW network, with more faithful characteristics and API methods regarding SKYLAB *4SpW* products.
- **virtual router**, allowing SpW routing, whose implementation is inspired from the SpW 10x device, as described in its User Manual[2] and which supports Group Adaptative Routing. It can be configured with up to 31 ports and can be controlled via RMAP port number 0.
- **virtual logical analyzer**, which can be inserted between nodes of any kind (physical, emulated, virtual) or router ports, allowing data to be monitored or stored for debugging purposes. Four modes are available: step-by-step, blocking buffer, non-blocking buffer and continuous modes.
- **IP tunnels**, which can be used to connect multiple API locally or through an intranet/internet network. These IP tunnel use a peer to peer architecture.

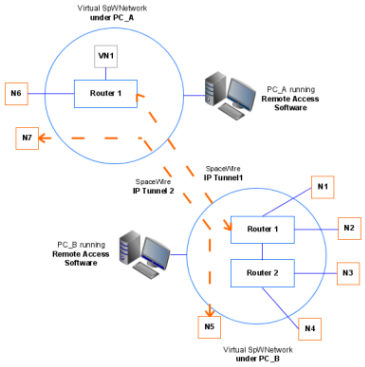


Figure 1: virtual SpW network and IP tunnelling

TRAFFICONTROLLER^{4SpW} ARCHITECTURE OVERVIEW

As previously described, the software suite enables space industry engineers to use network physical devices and network virtual devices. Emulated and real nodes are accessed through the Device Virtualization Service (DVS) layer, the lower layer of the traffiController^{4SpW} package. Virtual nodes are managed at API level, providing the rich set of communication elements for network routing and analysis, with the capability to transfer data through multiple tunnels. Additionally, Graphical User Interface (GUI) and console applications provide intuitive interaction with lower layer services. Each of these nodes (virtual, emulated and real) are accessed through DVS, whose architecture was inspired from the CCSDS SOIS PnP spatial related architecture[3]. More detailed information on DVS, API and GUI implementations is available online in the traffiController^{4SpW} product datasheet.

SPACEWIRE NETWORKS TESTING AND DIAGNOSIS: TEST CASES

Using previously described software and hardware resources, we could setup a reconfigurable emulated SpW network using multiple nodes potentially dispatched between Saclay (CEA), Toulouse (Skylab) and Tunis (ASIC) geographical sites. Such a test bench gave us flexibility to be able to configure sub-networks using virtual routers within the overall network. In the next test cases, we consider such a testing configuration based on a local API 0 configuration and a distant API 1 configuration, whose characteristics are described in Figures 3 and 4 below, with the related GUI screen copies. Note: 'T[]' stands for Tunnels in the related boxes.

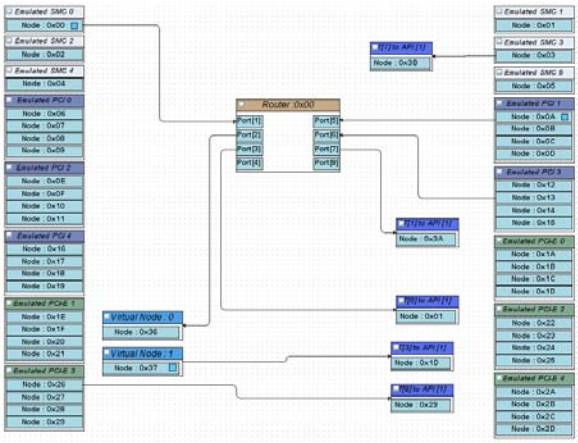


Figure 2 : API 0 configuration (local)

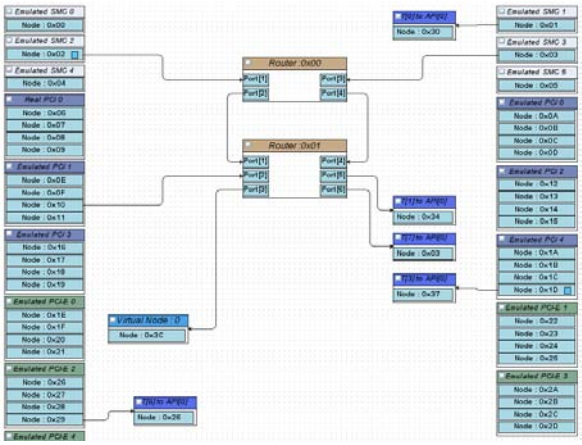


Figure 3 : API 1 configuration (distant)

These test bench configurations have been used to perform a few dozens of test scenarios which are summarized in the following three categories, according to their complexity:

1. Direct connection between nodes, with or without RMAP capability
2. Connection through just one router: many tests have been run using: (a) logical or physical addressing, (b) with or without RMAP, (c) router configuration packets
3. Connection through more than one router: same as above using intermediate routers

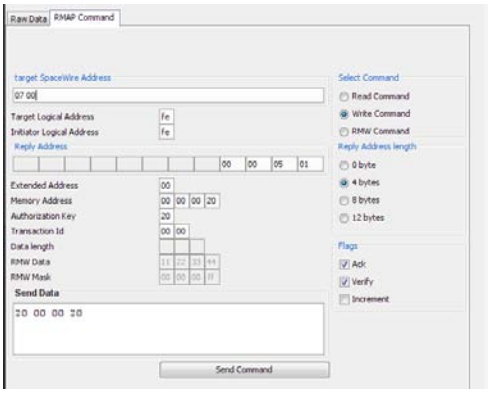
All these tests have been performed successfully using either a stand-alone configuration (API 0 and API running on the same PC) or a distributed configuration (API 0 and API running on different networks). For each of these tests, error codes processing have been treated to be able to handle bad commands, such as RMAP over SpW errors. The following are examples of test cases that were carried out:

- Test 1: Packet exchange between two remote nodes through a local and a remote router
- Test 2: Configuring a remote router
- Test 3: Using the analyzer function in *Blocking Buffer* mode

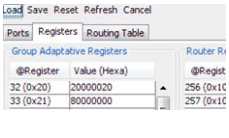
Test1: The objective was to enable the transmission and reception of data between two remote nodes through a local router and a remote router. The test case was to send data from the output file of Virtual Node 0 of API 0 while receiving them in the input file of the emulated smartCable of the API 1, located on anothercomputer.

Virtual Node 0 with ID 0x36 was selected to send various lengths and types of SpW data with the following specific physical addresses header: 0x07 0x01 0x03. When routed by Router 0x00, data were sent through the API tunnel T[1] with ID 0x3A, arriving Port 5 of Router 0x01 API 1, routed to port[1], routed to port[3] of Router 0x00 API 1 finally reaching emulated smartCable ID 3 of the API 1.

Test 2: The goal was to setup a router register using a Write Single Address RMAP command. The test case intended to configure Router 0x0 of API 0 from emulated smartCable 0.



As described on the left, a Write Single Address RMAP command could be sent from the *Test Node* window and checked that it was performed correctly by viewing the response in the Raw Data tab. The content of the register located at address 0x20 of the router could be checked as described in the following screen-copy.



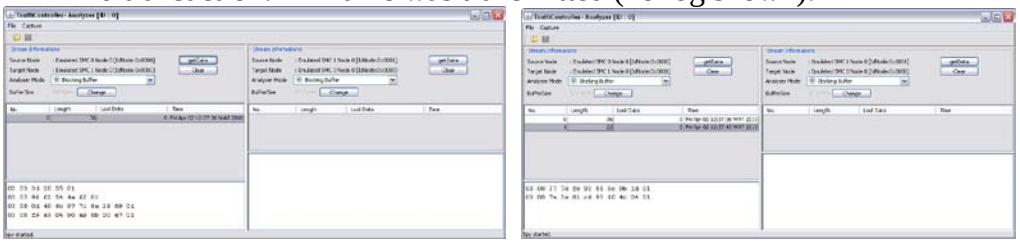
Test 3: The objective was to use the logical analyzer in *Blocking Buffer* mode, with a fixed buffer size of 50 bytes. The two following possibilities were considered: *sent data length* less than buffer size, then *sent data length* greater than buffer size: in this case, only the first frames had been recovered during the first reading. The normal behavior was that during this time, the next frame was on a pending state for the buffer release. All other sending attempts, after the buffer was full, had been ignored.

To illustrate the functioning of the *Blocking Buffer* mode, we considered theses hex frames:

Frame 1: 03 03 3d 20 55 01, 03 05 9f f2 54 4a f2 01
Frame 2: 03 08 0d 40 6b 97 7c 8a 29 89 01, 03 08 f9 a3 04 90 a9 8b 00 e7 01
Frame 3: 03 08 37 7d fc 93 55 5c 9b 1d 01, 03 08 7a 3a 81 c6 93 10 4c 04 01
Frame 4: 03 08 14 db 64 c6 d1 db 81 24 01, 03 06 a8 5c f7 53 a5 3d 01

The test was divided into three sequences:

- First transaction: 1st and 2nd frames were sent, as logged on the left window below.
- Second transaction: 3rd frame was then sent, as logged on the right window.
- Third transaction: 4th frame was transmitted (no log shown).



The first two frames were normally analyzed (sent data length is 36 less than 50 bytes). The second transaction was also analyzed because 3rd frame data was in pending mode. After completion of the first transaction, the available space in the buffer was only (50 – 36 =) 14 bytes and since the 3rd frame contained 22 bytes, it was immediately placed in a pending state to prevent data loss. The third transaction failed to be analyzed because the buffer was already full and there was another pending frame. This behavior complies with *Blocking Buffer* mode.

CONCLUSION

The paper introduced in details basic elements in test bench, typical API configurations and unitary/robustness tests and integration results of the test bench. The configuration aimed at representing most of the realistic scenarios met by node-to-node SpW users, router users and software developers. Most of these tests were conducted using representative emulators when drivers for physical devices were not yet available. Unfortunately, due to late driver validation for the smartCable and the unavailability of certain features like eye diagram or physical logical analyzer, unitary tests for these features could not be done and performance tests for this device with the traffiController could not be performed. However, in addition to the final PCI and smartCable performance tests, additional measurements will be done in the coming months with the new PCI Express equipment, giving complementary results to this approach.

REFERENCES

1 ECSS-E-ST-50-12C (Spacewire – Links, nodes, routers and networks)
2 UoD_SpW_10X_UserManual (SpW-10X SpaceWire Router - User Manual)
3 CCSDS-SOIS PnP on MILBUS (SOIS PnP Device and Service Discovery an example of an adaptation model for MILBUS (On ECSS-E-50-13 Compliant System))