Spacewire Interface Simulation

Gisbert Peter
German Aerospace Center
gisbert.peter@dlr.de

Ulf Rohbeck
CLIPhIT
urohbeck@cliphit.de

Rainer Berlin
German Aerospace Center
rainer.berlin@dlr.de

Natascha Russ
German Aerospace Center
natalie.russ@dlr.de

Bernd Ulmer
Ingenieurbüro Ulmer
bernd.ulmer@ib-ulmer.de

Abstract

Spacewire [1] is an interface standard for high rate peer-to-peer intercommunication between several nodes and routers for example between payloads and a spacecraft on-board computer. The functionalities of Spacewire continue to increase, mainly due to the increase of on-board software complexity. New facilities have to be provided to support the development and testing of Spacewire and on-board software applications.

This paper provides an overview of a generic "request - action list" concept for real-time interface simulation physically implemented as a Spacewire Interface Simulator (SWIS).

The SWIS concept provides a means of generic peer-to-peer interface simulations of the packet, protocol and application layer with free user- defined functionality for several simultaneous Spacewire links. A payload simulation can be established by "drag and drop" in a much shorter time than usually needed for project specific software development.

Using the SWIS test and system engineers as well as managers can prepare simulations without any need for real-time software development. The effort (i.e. cost and time) for development of the test facility is reduced and the quality may increase significantly.

1. Introduction

Development of embedded on-board software for space applications is becoming more and more important. The complexity of the applications is increasing rapidly and quality aspects are essential for success of the mission.

Due to budget and schedule constrains the processes of hardware and on-board software development are parallelized and the software has to be developed independently from the hardware. Hardware-in-the-loop simulations and test automation have been established as methods for supporting the development and testing of on-board software before its installation and validation on the target hardware. The quality of the test system, containing a high percentage of software, must have a comparable or higher level of quality, which is very often not the case. The test system may have insufficient quality, an unstable configuration and may not be available on time for on-board software development. Additionally, the development costs of on-ground test systems are high and of the same order of magnitude as the cost of the on-board software itself.

A way of reducing the cost and development time, as well as improving the quality, is to standardize the interfaces. For space missions this has already been, or soon will be, achieved from the hardware point of view via e.g. of interface standards like Spacewire [1], which is a new standard for high speed peer-to-peer inter-communication or for bus data exchange on Spacecraft platforms (MIL-Std-1553 [7]). There is a need to find concepts to supports this type of standardization, especially from the software point of view. Therefore, a generic concept for Spacewire interface simulation has been developed.

Spacewire [1] is of great importance for current and future space applications (e.g. for the European Space Mission BepiColombo), as a standard interface for high speed and reliable data communication but with low implementation costs. An overview of Spacewire applications is shown in [6].

The standardization of development processes and physical implementations leads to cost and development time reduction as well as to a product quality improvement. The Spacewire technology is an important step in this direction. Several hardware applications have already been or will be established using Spacewire for the development and testing of space applications (see [6]). In order to further promote standardization, a quasi "software standardization" is needed to support the application of specific implementations of the Spacewire technology. The SWIS contributes to the standardization process from the software or application point of view.

Major topics for the development of payloads as well as on-board computers are:

- High complexity of real-time interface simulations has to be managed,
- Electrical Ground Support Equipment (EGSE) has to be available before commencing the development of the space unit itself,
- Parallel and independent development of hard- and software is needed,
- The cost of development of test and simulation facilities should be minimized.

The SWIS has been designed with these topics in mind. Using the SWIS, developers, and system or test engineers of space units can easily establish a project for simulation of a slave (e.g. a complex payload) in a very short time without any software development. The hard- and software development of a master system (e.g. the on-board computer) can be parallelized because the simulated payload is already available physically by means of the SWIS. Development time and costs are reduced significantly.

The concept of the SWIS has been established on the basis of the experience from developments for the European Space Missions COROT [2], Rosetta [3] and Venus Express [4]. It was used successfully for developing and testing the embedded on-board software.

The SWIS runs on a PC under Windows XP with a Spacewire board provided by the UK Company 4links either as a stand-alone simulator or as a simulator integrated in to a test system for test automation. Using the SWIS any packet oriented communication protocol (e.g. CCSDS standards) can be simulated.

2. Typical applications of the SWIS

In the future the Spacewire interface will be used in many space applications, e.g. for communication between payloads and on-board computers. The SWIS is focused on two typical applications. 1. A payload (slave) simulator for developing and testing an on-board computer (master) as shown in Figure 2-1.

2. A payload (slave) simulator for test equipment (master) development (e.g. EGSE) as shown in Figure 2-2.

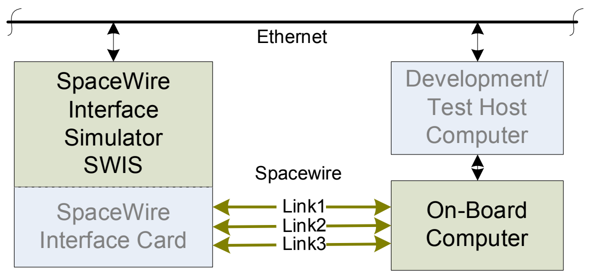


Figure 2-1: On-board computer development

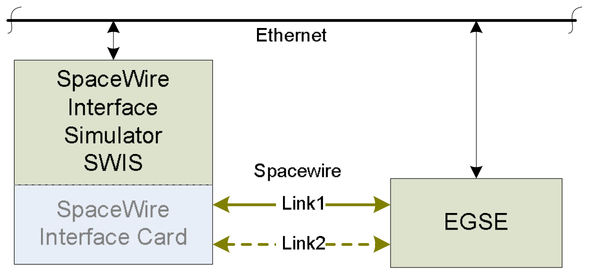


Figure 2-2: Test facility development

3. General Concept

The core of the SWIS is the "request - action list" concept. Requests are freely configurable command packets received over a Spacewire link. Each request can initiate a list of actions. An action has a small generic functionality for different levels of interface or application simulations (e.g. SendPacket, SetCondition, SetWord). An action list can contain one or several actions for simulating a particular behaviour of a SWIS link or for manipulating its received and sent simulation data or its simulation mode. Several action lists, triggered by different dedicated requests, run simultaneously with small latencies on the links. An action list can run once or several times up to "forever" in the background.

This concept is based on the following general requirements:

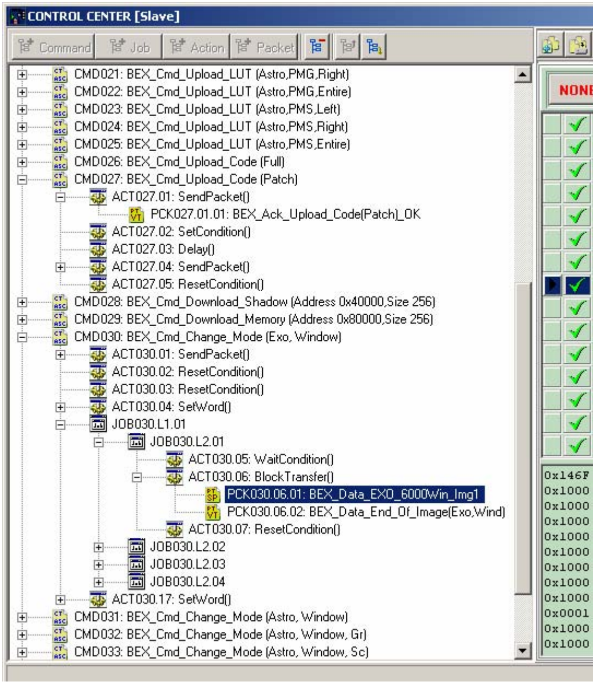
It shall be possible that

- Requesting the SWIS by a user defined command packet (at any time asynchronously) leads always to an immediate dedicated response with low latency (less than 10msec) or a predefined delay. A response can be a single packet or a sequence of packets with user predefined content.
- Requesting the SWIS by a command leads to a permanent change of its simulation mode. A simulation mode represents a specific interface behaviour depending on SWIS internal

conditions. A condition can be set or reset by a request command.

- Each simulation action can be configured to be executed by each request command. That means, the same action type can be performed for different requests. Actions are SWIS generic functions for influencing the simulation.
- Data sent to the SWIS can be processed and used as data sent from the SWIS.
- Data sent by the SWIS (i.e. data to be simulated) can be manipulated on bit, word, packet and application data level by generic actions.
- Request commands and simulated data are free in format and content.

All these features require a multi-process and real-time simulation e.g. to simulate the typical performance of complex payloads. The user is able to establish a synchronous, asynchronous and condition controlled application, function, protocol, packet and timing simulation in a generic way. A simulation of a Spacewire interface in real-time is possible as a slave based on a freely configurable request command and data packet communication protocol.



```
CMD021: BEX_Cmd_Upload_LUT (Astro,PMG,Right)
CMD022: BEX_Cmd_Upload_LUT (Astro,PMG,Entire)
CMD023: BEX_Cmd_Upload_LUT (Astro,PMS,Left)
CMD024: BEX_Cmd_Upload_LUT (Astro,PMS,Right)
CMD025: BEX_Cmd_Upload_LUT (Astro,PMS,Entire)
CMD026: BEX_Cmd_Upload_Code (Full)
CMD027: BEX_Cmd_Upload_Code (Patch)
    ACT027.01: SendPacket()
        PCK027.01.01: BEX_Ack_Upload_Code(Patch)_OK
    ACT027.02: SetCondition()
    ACT027.03: Delay()
    ACT027.04: SendPacket()
    ACT027.05: ResetCondition()
CMD028: BEX_Cmd_Download_Shadow (Address 0x40000,Size 256)
CMD029: BEX_Cmd_Download_Memory (Address 0x80000,Size 256)
CMD030: BEX_Cmd_Change_Mode (Exo, Window)
    ACT030.01: SendPacket()
    ACT030.02: ResetCondition()
    ACT030.03: ResetCondition()
    ACT030.04: SetWord()
    JOB030.L1.01
        JOB030.L2.01
            ACT030.05: WaitCondition()
            ACT030.06: BlockTransfer()
                PCK030.06.01: BEX_Data_EXO_6000Win_Img1
                PCK030.06.02: BEX_Data_End_Of_Image(Exo,Wind)
            ACT030.07: ResetCondition()
        JOB030.L2.02
        JOB030.L2.03
        JOB030.L2.04
    ACT030.17: SetWord()
CMD031: BEX_Cmd_Change_Mode (Astro, Window)
CMD032: BEX_Cmd_Change_Mode (Astro, Window, Gr)
CMD033: BEX_Cmd_Change_Mode (Astro, Window, Sc)
```

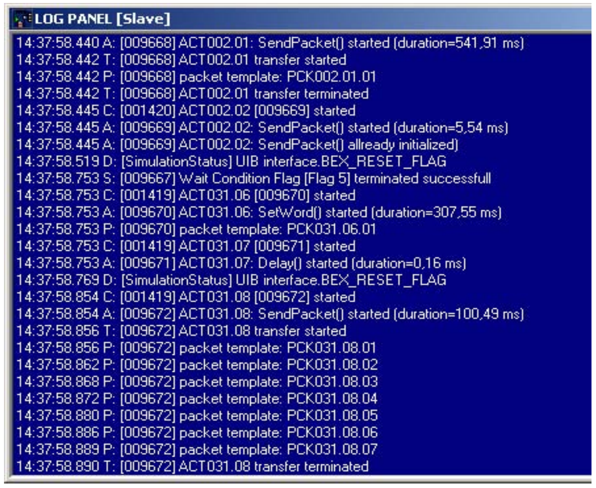
```
0x146F
0x1000
0x1000
0x1000
0x1000
0x1000
0x1000
0x1000
0x1000
0x0001
0x1000
0x1000
```

Figure 3-1: SWIS request - action list explorer

From the user point of view the SWIS has a configuration mode for establishing a simulation project and real-time simulation mode for simulating and monitoring up to 3 Spacewire links simultaneously.

The possibly complex simulation, based on the "request - action list" concept, is transparent for the user due to the "request - action list explorer" structure as shown in Figure 3-1.

Advanced protocol reporting features allow the interface communication to be traced on-line during the simulation (see Figure 3-2) or by means of a log file. A simulation configuration can be stored as a project file. For test purposes the SWIS is able to work as a master in order to establish SWIS/Master to SWIS/Slave communication for setting up and testing the SWIS itself. A simulation project can be developed without having the master target hard- and software available.



```
14:37:58.440 A: [009668] ACT002.01: SendPacket() started (duration=541,91 ms)
14:37:58.442 T: [009668] ACT002.01 transfer started
14:37:58.442 P: [009668] packet template: PCK002.01.01
14:37:58.442 T: [009668] ACT002.01 transfer terminated
14:37:58.445 C: [001420] ACT002.02 [009669] started
14:37:58.445 A: [009669] ACT002.02: SendPacket() started (duration=5,54 ms)
14:37:58.445 A: [009669] ACT002.02: SendPacket() allready initialized)
14:37:58.519 D: [SimulationStatus] UIB interface.BEX_RESET_FLAG
14:37:58.753 S: [009667] Wait Condition Flag [Flag 5] terminated successfull
14:37:58.753 C: [001419] ACT031.06 [009670] started
14:37:58.753 A: [009670] ACT031.06: SetWord() started (duration=307,55 ms)
14:37:58.753 P: [009670] packet template: PCK031.06.01
14:37:58.753 C: [001419] ACT031.07 [009671] started
14:37:58.753 A: [009671] ACT031.07: Delay() started (duration=0,16 ms)
14:37:58.769 D: [SimulationStatus] UIB interface.BEX_RESET_FLAG
14:37:58.854 C: [001419] ACT031.08 [009672] started
14:37:58.854 A: [009672] ACT031.08: SendPacket() started (duration=100,49 ms)
14:37:58.856 T: [009672] ACT031.08 transfer started
14:37:58.856 P: [009672] packet template: PCK031.08.01
14:37:58.862 P: [009672] packet template: PCK031.08.02
14:37:58.868 P: [009672] packet template: PCK031.08.03
14:37:58.872 P: [009672] packet template: PCK031.08.04
14:37:58.880 P: [009672] packet template: PCK031.08.05
14:37:58.886 P: [009672] packet template: PCK031.08.06
14:37:58.889 P: [009672] packet template: PCK031.08.07
14:37:58.890 T: [009672] ACT031.08 transfer terminated
```

Figure 3-2: SWIS log panel

4. Interface simulation

4.1. Packet and application data simulation

The SWIS concept is based on packet request and packet response simulation. The user defines the content, structure and size of packets configured for command execution and data response simulation. Together with simulation actions and parameters the packet content can be modified (e.g. bits, words or areas) in real time in order to provide a dynamic data simulation.

4.2. Protocol simulation

It is possible to establish two simulation modes, a synchronous and/or asynchronous protocol simulation.

The synchronous mode is foreseen to simulate a command request and data response behaviour of single packets (e.g. command acknowledgement) as well as packet sequences with a predefined timing and packet content (e.g. sensor data simulation).

The asynchronous mode allows a ”background” simulation without a permanent request, e.g. a periodical or continuous transfer of data.

5. Application functional simulation

5.1. Synchronous functional simulation

Synchronous interface simulation allows an event triggered protocol communication with deterministic interface timing. Each packet request event has a predefined response (i.e. single packet or packet sequences) with a predefined timing. It allows a typical request/response simulation support by means of SWIS generic actions e.g. SendPacket() or BlockTransfer().

5.2. Asynchronous functional simulation

Asynchronous simulation is foreseen to simulate a “free running” system. It is interface event triggered or time scheduled. Event triggered means the asynchronous process is controlled (e.g. start/stop) by request packet commanding. Time scheduled means the simulation depends on periodic or one-shot timer controlled functions. Technically the SWIS supports that feature by means of an advanced “Job” concept, configure via “drag and drop”. An action “Job” covers an action list which can be counter, timer and condition controlled.

5.3. Condition controlled simulation

Complex systems have often several functional modes based on special parameter pre-configuration or internal conditions. That means it must be possible to influence an interface behaviour in real time initiated by command requests. A proper function or handling of failure cases have to be simulated, e.g. a sensor system sends different sensor data in two different modes depending on a special parameter configuration. The “condition controlled simulation” supports such behaviour. That means a command request packet can set SWIS internal conditions which influence the simulation protocol, response packet content and/or timing. In the same way a simulation of interface or system failures can be performed if needed. The SWIS supports simulation functions by generic actions like “SetCondition()”, “ResetCondition()” or “WaitCondition()”.

5.4. Timing simulation

There are several possibilities for supporting timing simulations, gaps between packets or packet sequences; time-outs during packet transfer, repetition periods for asynchronous protocol simulation, delays before and after packet transfer, etc. The time base is 1 msec. The real time accuracy is better than 5msec e.g. on a Windows XP, 3GHz Pentium system. The SWIS supports Timing Simulation by generic actions like “Delay()” or “WaitTime()”.

5.5. Time base synchronization

For large distributed systems a synchronized time stamping of data is needed. The SWIS supports a synchronization of its time base over a Spacewire link e.g. with the master time base. It allows common time stamping for the SWIS protocol data and the master, which requests the SWIS. The SWIS supports timing and time base synchronization by an generic action like “SetTime()”.

5.6. Power-on link connect simulation

In general the SWIS as slave sends data packets only after request by a master. But sometimes it is needed to send a “power-on” or “link successful connects” message packet after link connection from the slave to the master without packet request. A dedicated action “LinkConnect” is foreseen to simulate this special case and the action “LinkDisconnect” provides an opposite function in case of a link disconnect.

5.7. Failure simulation

For failure simulation the interface response must be deterministic for each command request even if a command is unknown for the simulation project. Therefore, a request command “UnknownCommand” is present in each simulation project by default. This allows actions to be simulated in the case an undefined or corrupted failure packet is received by the SWIS.

6. Reporting

The SWIS provides advanced reporting functions during a running simulation. Time tagged reporting information such as the link communication protocol, the packet type and content as well as event and error messages are written in a log file and/or displayed in real time. Several general reporting levels can be selected in order to adjust the volume of reporting information.

For dedicated reporting of e.g. an execution of one command request some reporting actions are provided, e.g. DisplayPacket() or Report ().

7. SWIS design

The SWIS is designed as a dynamic multi-threading application ensuring a latency of only few msec. It runs on a PC under Windows XP with a 4Link [5] Spacewire PCI board. Its architecture is shown in Figure 7-1. Figure 7-2 shows the adaptation of the "request - action list" concept on its physical design implementation. Each action list is executed in a separate thread dynamically established and executed after receiving a dedicated command request. The communication between the threads is optimized and a special real time kernel guarantees low latencies and low system load in the simulation mode.

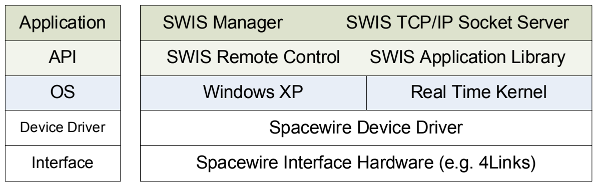


Figure 7-1: SWIS Architecture Design

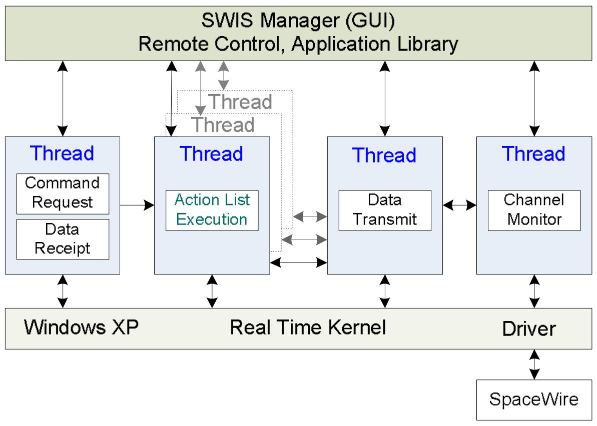


Figure 7-2: SWIS Action List/Threading Architecture

For testing of a simulation project two SWIS applications can run on one PC, one configured as master, the other one as slave connecting at least two links of one 4links board to each other. This allows a SWIS test without any other hardware. In such a configuration a simulation project can easily be developed and tested before connecting a real Spacewire link to the master hardware.

A TCP/IP socket communication is possible for using the SWIS in a LAN environment where the SWIS application runs on one PC and other Spacewire hardware is installed on a second separate PC.

Optionally, it is possible to integrate the SWIS within a system for automatic testing. A remote application control feature allows configuration, start and stop of the simulation without using its GUI.

8. Outlook

The SWIS concept and its real time design allow for an easy extension of the simulation features. Additional generic actions can be added to allow more possibilities e.g. for data manipulation. It is planned to add a dynamic data reload for the simulation of large sets of specific application data.

An adaptation of the SWIS on other Spacewire hardware platforms is foreseen in order to support new hardware developments and to achieve higher data rates than about 100Mbit/s with the 4links PCI card.

9. Conclusion

A generic concept of a peer-to-peer real-time interface simulation has been developed, and physically implemented. SWIS can be used as a Spacewire slave simulator for the development and testing of embedded software either as a stand-alone simulator or inside a system for test automation. The user is free to define an interface protocol, timings, packet contents and its simulation functionality. A "request – action list explorer" allows him to establish any slave application by "drag and drop".

Development of specific simulation software is no longer needed the for testing or development of Spacewire applications. The SWIS user does not need any knowledge of real-time software design or programming.

Changing the interface protocol specifications during a project life cycle is no longer a problem. In this case the SWIS simulation project can be modified with little effort in a very short time and the general quality of simulations is not affected.

The SWIS has been successfully used for complex Spacewire interface simulations requiring few msec latencies, e.g. for the development and testing of on-board (master) flight software without having the real slave available as hardware.

It is possible to adapt the SWIS concept to each interface where peer-to-peer communication at the packet level has to be simulated in real time. In this case the hardware driver library must be exchanged or adapted in order to provide a real time simulator with a different interface to that of Spacewire.

10. Acknowledgement

The authors would like to thank the COROT team for their support of the on-board software development during which the SWIS concept finally became established. We are particularly grateful to Philippe Plasson from LESIA/France-Meudon and Bernard Pontent from CNES/France-Toulouse.

11. References

[1] ESA-ESTEC, ECSS Secretariat Noordwijk, Netherlands, “Spacewire Standard,” ECSS-E-50-12 Draft 4, September 24, 2002

[2] CNES, “COROT, a French Space Mission for Searching of Exosolar Planets and Astroseismology,” http://smsc.cnes.fr/COROT

[3] ESA, “Rosetta, a Space Mission for Observing the Comet Churyumov-Gerasimenko,” 1996-2014; http://sci.esa.int/science-e/www/area/index.cfm?fareaid=13
“VIRTIS, an imaging IR spectrometer on Rosetta,” http://servirtis.obspm.fr/virtis.html

[4] ESA, “Venus Express, a Space Misson for observing the Venus,” 2003 – 2009, http://sci.esa.int/science-e/www/area/index.cfm?fareaid=64
VIRTIS, an imaging IR spectrometer on Venus Express, http://www.rm.iasf.cnr.it/ias-home/Venus-Express/Venus-Express.htm

[5] 4links Spacewire PCI board, http://www.4links.co.uk/spacewire_pci.htm

[6] ESA, “Spacewire Overview,” http://www.spacewire.esa.int/

[7] US Department of Defence, “MIL-Std-1553B, Interface Standard for Digital Time Division Command/Response Multiplex Data Bus,” 1973-1993