Software-to-Hardware Tester for the STP-ISS transport protocol verification

Valentin Olenev, Irina Lavrovskaya, Nadezhda Chumakova
Saint-Petersburg State University of Aerospace Instrumentation
Saint Petersburg, Russia
{valentin.olenev, irina.lavrovskaya, nadezhda.chumakova}@guap.ru

***Abstract*—Implementation of conformance testers for the communication protocols is an important task, which is being solved in the majority of industrial companies that develop the communication equipment. Current article gives a description of such kind of tester, which is developed to test the on-board devices that work in conformance to the STP-ISS transport protocol standard and SpaceWire networking standard. We give a brief description of the possible solutions for hardware testing; provide the description of STP-ISS protocol. Then we report on implementation of the Software-to-Hardware STP-ISS tester and fields of its application.**

I. INTRODUCTION

Compliance or conformance testing is basically a kind of an audit which is performed for the system to check whether all the specified standards are met or not. To ensure that the compliances are met, industrial companies develop special testing equipment or software. The widely used standards such as USB or Ethernet are developed by large organizations and could be tested on all stages of the implementation. We do not need a special equipment to test the hardware if we buy a USB stick or a networking card, we can just plug it into a computer and operation system will do it automatically. But if a company develops a new specialized protocol and a number of devices that should meet the requirements of a new standard, this company should carefully test the implemented equipment before integration and dissemination.

On-board equipment is such kind of equipment that needs very proper and detailed testing [1]. And if a new protocol for on-board communication is developed, then we need to be sure that the devices work as expected, before we can integrate it into an aircraft or a spacecraft.

We had a log-term project for the research, development and implementation of an STP-ISS transport protocol for the on-board communication via the SpaceWire networks. In this project we developed two revisions of STP-ISS protocol, simulated and investigated them. The first revision of STP-ISS is much simpler and compact, but the second one is more powerful. Nevertheless, the backward compatibility for these revisions is provided. After that we got the task to implement a tester for the STP-ISS rev.1 equipment, which could tell the manufacturer, that STP-ISS device operates correctly. Tester should examine the device with a set of different testing scenarios; each scenario should test a particular STP-ISS mechanism. So after the testing the manufacturer will know what STP-ISS mechanism failed and it can analyse the log-files for details.

For this reason we conducted an overview of different approaches for the implementation of hardware conformance testers, studied the main examples of conformance testers represented at the market.

II. HARDWARE AND SOFTWARE CONFORMANCE TESTING

On-board equipment always needs a proper testing before the integration into a spacecraft. Especially if we talk about equipment, that operates according to the newly developed communication protocol. The conformance testing should be provided to prove that this equipment meets the requirements.

Conformance testing is such kind of testing, which gives an ability to ensure that a hardware or software product, system or just a medium complies with the requirements of a specification or any other document. Various test procedures, testing software or hardware testers have been developed either by the standard's maintainers or external auditing organizations, specifically for testing conformance to standards. Also service providers, equipment manufacturers, and equipment suppliers rely on such testing to ensure Quality of Service through this conformance process.

Conformance testing may include some of these kinds of tests, it has one fundamental difference – the requirements or criteria for conformance must be specified in the standard or specification. This is usually in a conformance clause or conformance statement, but sometimes some of the criteria can be found in the body of the specification. Some standards have subsequent documentation for the test methodology and assertions to be tested. If the criteria or requirements for conformance are not specified, there can be no conformance testing [2].

Many companies that develop or just work with the new equipment have such kind of conformance testers and usually equipment testing is done by the testing organizations. But some standards have no official testing organizations. They rely on self-assessment by the implementer and acceptance testing by buyers.

Depending on the available information we can elaborate two main approaches for the conformance testing that are widely used across the industry:

ISBN 978-952-68397-0-7 (paperback), ISBN 978-952-68397-1-4 (PDF)

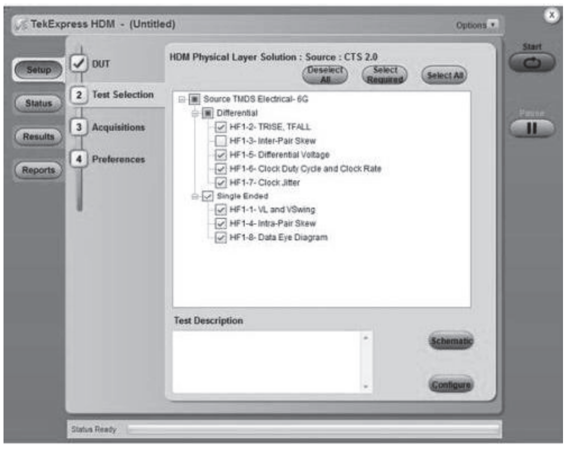
- Software testers;
- Hardware testers.

Software testers usually consist of a test entity (software) that includes a number of test cases. These tests are aimed to get the correct responses from the unit that is being tested. Testing software is running on a PC or any portable device and it is connected with the real hardware, that it tests. Conformance testing software usually includes a test tool (e.g., tool, suite, and/or reference implementation) and procedures for testing (test engine).

The software may be represented by a set of programs, a set of instructions for manual action, or another appropriate alternative. It is likely to be platform independent, and it should generate repeatable results. A reference implementation is an implementation of a standard that is by definition conformant to that standard. Such an implementation provides a proof of concept of the standard and also provides a tool for the developers of the conformance software. The reference implementation is of considerable importance on the early stages of conformance testing.

The conformance testing procedures should be agreed and implemented before testing begins. This would include the implementation of different types of tests.

There are many examples of the software testers for the communication protocols. One of them is the "HDMI compliance test software" that is implemented and distributed by Tektronix. It automates a comprehensive range of tests on conformance to HDMI 1.4a/b and HDMI 2.0 standards (see Fig 1) [3].



HDM Physical Layer Solution : Source : CTS 2.0

- Source TMDS Electrical- 6G
  - Differential
    - ✓ HF1-2- TRISE, TFALL
    - ☐ HF1-3- Inter-Pair Skew
    - ✓ HF1-5- Differential Voltage
    - ✓ HF1-6- Clock Duty Cycle and Clock Rate
    - ✓ HF1-7- Clock Jitter
  - ✓ Single Ended
    - ✓ HF1-1- VL and VSwing
    - ✓ HF1-4- Intra-Pair Skew
    - ✓ HF1-8- Data Eye Diagram

Fig. 1. HDMI compliance test software by Tektronix

The other good example of testing software is R&S®CMW – Conformance testing solution for eCall/ERA-Glonass implemented by Rohde&Schwartz Gmbh&Co (see Fig. 2). It is electronic safety systems for cars, developed by European Union and the Russian Federation to have intelligent telematics-based vehicle safety systems to speed up emergency response times in order to save human lives. This software tester is a solution for automated, reliable and reproducible end-to-end conformance tests on eCall/ERA-Glonass modules [4].

Fig. 2. Conformance testing solution for eCall/ERA-Glonass implemented by Rohde&Schwartz Gmbh&Co

Also there are a number of good conformance testing software implementations based on the formalized methodics and algorithms. These examples are described in [5] and [6].

The other way of conformance testing is using of the real hardware testers that produce the test sequences and test the remote device. Usually it is a device operating with full respect to the standard, which could have a different number of parameters and settings. Configuration of this device is performed via a special configuration software installed on PC. There are also many examples of hardware testers for the widely used communication standards as USB, LAN, RS232 and others (see Fig. 3).

Fig. 3. Implementations of hardware testers for different communication standards

There is another example that is related to the on-board equipment testing – the SpaceWire Conformance Tester implemented by Star-Dundee. It connects to a SpaceWire device and, through the host software, executes a variety of tests to check the device under test's (DUT) compliance to the SpaceWire Standard. Over 55 tests can be conducted. The user can easily select which tests they do and do not want to run. With each test, expected and achieved results are displayed, including a link to the appropriate clause of the SpaceWire Standard to dramatically reduce the time spent debugging the DUT. The SpaceWire Conformance Tester can also be used as a high speed packet generator, and one of the SpaceWire links can act as a data / time-code sink or loop-back [7].

Fig. 4. SpaceWire Conformance Tester

III. STP-ISS Protocol General Description

The main task for our research was to decide how to test the newly developed devices that should operate in conformance to the STP-ISS protocol specification. So firstly, we should describe what STP-ISS protocol is, what main mechanisms and distinctive features it has. In this section, we consider the second revision of the STP-ISS protocol which includes all required functionality.

STP-ISS is a transport layer protocol that describes informational and logic interaction between on-board devices, packets' formats and packet transmission rules for SpaceWire networks. STP-ISS protocol corresponds to the Transport layer and provides means for transmission of data between the nodes of the network with the required quality of service. This protocol gives ability for data resending in case of an error detection in the received data.

There are three interfaces for interaction between the STP-ISS and Applications: Data Interface, Configuration Interface and Control Codes Interface. At the bottom STP-ISS has two interfaces for interconnection with the SpaceWire layers: SpaceWire packets interface and Control Codes Interface. Through these interfaces STP-ISS provides transmission of control commands, data packets, SpaceWire time-codes, SpaceWire distributed interrupts and interrupt-acknowledges. The Configuration Interface provides means for the STP-ISS configuration parameters change, for transmission of status information, reset commands and connection establishment.

There are two types of application messages:

- urgent messages (higher priority);
- common messages (lower priority).

STP-ISS encapsulates applications' messages into SpaceWire packets. Length of each message data block should be not less than 1 byte and should not exceed 2048 bytes for the connectionless data transmission, and 64 Kbytes maximum for the connection-oriented data transmission.

STP-ISS provides the reliable data transmission by using CRC-16 for protection of payload and packet header and for errors detection [8]. CRC-16 covers the packet starting from the first byte of the STP-ISS packet header (excepting path address) till the last byte of data, excluding the end of packet symbol EOP.

The transmitter side of the protocol has separate buffers for each priority of the transmitted data:

- control commands buffer;
- urgent messages buffer;
- common messages buffer.

The size of these buffers should be set depending on the message or segment size, which the node uses for the data exchange.

If the buffer overflow occurs, the application should wait until the free space for the message is available. For each SpaceWire packet STP-ISS protocol has a special lifetime timer, which counts the time, when the packet is still relevant in the SpaceWire network. Each packet is stored in the buffer during its lifetime.

STP-ISS has two logical buffers at the receiver side. The first buffer is used for the connectionless data transmission, for all types of packets (control commands, common messages and urgent messages). The second buffer is used for the connection-oriented data transmission only. The receiving side should reserve required space in the buffer for each new connection. If one of the receiving buffers is full, then STP-ISS should indicate the Application layer about it and discard all the packets coming from the SpaceWire.

There are two additional signals that could be passed from the application layer to the STP-ISS through the configuration interface: Reset and Flush. Reset corresponds to the warm reset, and Flush is used for clearing of both transmit and receive buffers.

The important STP-ISS feature is its configuration flexibility. The protocol has a number of configuration parameters, which give ability to tune the protocol depending on the developer needs. There are some mechanisms that should be implemented as mandatory. For example, Priority QoS at least for one priority, Best effort QoS, transmit and receive buffers. The other mechanisms are extensions and could be optionally implemented in different combinations.

One of the STP-ISS benefits is the possibility to transmit data using the following quality of service types:

- priority quality of service;
- guaranteed delivery quality of service;
- best effort quality of service;
- scheduling quality of service.

A. Priority quality of service

Priority quality of service is the main quality of service type that should be supported by all the network end-node devices, which communicate by means of STP-ISS. According to this quality of service type, the data with the higher priority should be transmitted first. STP-ISS supports 9 levels of priorities.

B. Guaranteed delivery quality of service

Guaranteed delivery quality of service provides confirmation for the successful packet transmission by sending the acknowledgement packets. In addition, it resends the data from the transmitter end-node if the acknowledgement is lost (resending mechanism). Guaranteed delivery is provided by resend timers and acknowledges. If a packet is passed to the network layer with the guaranteed delivery quality of service, STP-ISS should start the resend timer for this packet. When the resend timer expires, the corresponding packet should be sent to the network again.

STP-ISS protocol provides the duplicate control commands detection in the receiver. The duplicate control command can occur in case of the loss of acknowledgement. The receiver should store the information on the last received control commands.

C. Best effort quality of service

Best effort quality of service provides data transmission without acknowledging. When an STP-ISS receiver gets a best effort packet it checks the CRC and data length only. In case of an error or if the packet ends with EEP, the data packet still should be sent to the Application, but with an error indication.

D. Scheduling Quality of Service

STP-ISS assumes to have a single data transmission schedule for the whole SpaceWire network. This schedule gives an opportunity for the node to send data only during particular time-slots. The schedule consists of a number of time-slots, in turn an epoch has a constant number of time-slots. The schedule table describes one epoch. The time-slot timer counts duration of the current time-slot for a particular node.

STP-ISS has the timer synchronisation mechanism. Synchronisation is performed once in an epoch. During the synchronisation the node should calculate a new value for the time-slot timer. The newly calculated value will be applied for the time-slot timer of a new epoch. The new epoch should start when the time-code is received.

There are K time-slots in each epoch, when the time-code is recognized as relevant. These time-slots are called Time-code relevancy window. If a time-code is received before the last $K/2$ time-slots of the epoch, or after the first $K/2$ time-slots of the epoch, then this time-code is considered as irrelevant and synchronisation should not be performed. If the time-slot timer for a last time-slot expires simultaneously with the time-code reception, then there is no need to correct the epoch timer value.

Reception of three irrelevant time-codes means that we work asynchronous to the time-master and determines the beginning of a new epoch. The node should terminate the time-slot timer and wait for reception of the next time-code. In this new epoch the node should not send data until reception of the next time-code. After reception of the time-code the node should update the time-slot duration value and then continue data transmission according to the schedule.

E. Connection-oriented data transmission

Connection-oriented data transmission gives an ability to transmit large sized data with minimum overheads. Only urgent or common messages could be transmitted over a transport connection. Maximum number of transport connections should not be more than 8 per one direction. Each transport connection is unidirectional: it connects the transmitter of the initiator node and receiver of the remote node.

An application, which needs to transmit or receive a large portion of data, should initiate the transport connection establishment. The maximum size of data, which could be transmitted over the transport connection in a packet, is 64 Kbytes. The transport connection establishment is performed by means of classical three-phase handshake [9], [10]. During the connection establishment the application can set different connection parameters.

For each transport connection receiver and transmitter has a standby timer. This timer counts the time of waiting for the next data or service packet transmitted over the connection. On standby timer expiration the transport connection should be closed.

During the data transmission, STP-ISS provides the flow control, which is performed by sending of the information about the available free space in the receiving buffer. This mechanism is applied only for the transport connections with the guaranteed quality of service.

STP-ISS rev.1 protocol is described in [11] while STP-ISS rev.2 protocol was previously described in details in [12].

IV. STP-ISS REFERENCE CODE

STP-ISS specification development was followed by a simulation phase [13]. During this simulation stage we precisely analysed, investigated and tested the specification. In order to check STP-ISS protocol mechanisms we used three different models:

- SDL model;
- SystemC network model;
- C++ reference code.

These modeling and investigation directions for STP-ISS were described in more details in [14].

The SDL model is needed for the clear formal description of the STP-ISS internal mechanisms and specification analysis [15]. The SDL specification is used as a separate document describing the specified mechanisms, and it is a useful part for the main protocol specification document.

The SystemC model shows the STP-ISS protocol operation over SpaceWire network, and it gives an ability to test the network configuration and test networking features [14].

The reference code is intended to be used as the reference for the programmers, who will implement STP-ISS in the on-board software. The reference code is a software implementation of the STP-ISS protocol in C++ language [16]. This implementation corresponds to the specification as accurate as it is possible. The C++ reference code describes the logical structure of the protocol, its interfaces and internal mechanisms. All methods, which describe protocol functionality, are provided with detailed comments for each line. In addition, in order to check and prove the accuracy of STP-ISS the model contains a number of test scenarios for studying and demonstration of protocol functioning. Each scenario launch produces detailed log files with event traces of nodes and of a channel.

This reference code is used for studying of the protocol functionality. Moreover, it could be translated into the other programming languages and used for the implementation of STP-ISS in the on-board software.

The other possible application of the reference code is an implementation of a tester, that could be useful for testing of the software models or hardware implementation of the protocol. In this case, reference code is used as a black box, which works with full conformance to the STP-ISS specification [17]. This reference implementation of the protocol could be placed on the one side of the connection, and the software model or hardware protocol implementation – on the other side. That software or hardware implementation is called Device Under Test (DUT). Reference code can generate different types of packets and the DUT should respond to them. Depending on the result of the data exchange we can make a decision, if the DUT works in conformance to the specification or not.

V. Software-to-Hardware Tester

We used the reference code to implement a Software-to-Hardware Tester (S2HT) for the STP-ISS protocol. This title means that we test the real on-board *hardware* with the *software* implementation of a protocol model (reference code). And this is a software conformance tester, if we refer to the overview from the chapter II.

Software part of the S2HT consists of the following parts:

- Test engine "stp_testengine", containing a set of testing scenarios;
- STP-ISS reference code "stp_reference";
- Error generation module "error_generator".

Test engine is a set of testing scenarios for checking of correctness of the testing equipment operation. This module is implemented in SystemC, which represents a simulation library of C++ programming language [18]. After the start of the tester operation the user is able to choose the number of a test scenario and a test starts to execute. In the course of the test the S2HT performs a fixed number of actions according to the particular scenario, for example, protocol configuration or transmission of different types of packets. When the test is completed the tester displays the results. During execution of the test, the tester gathers the information on test operation and different events to the log files.

STP-ISS reference code part of the tester is a reference implementation of STP-ISS protocol with some modifications to the network level. These modifications give an ability to work with Star-Dundee USB Brick drivers. In the tester implementation the reference code is a separate library that is used by the software.

Error generation module is implemented for testing of the non-nominal cases in the communication process. Similarly the Test Engine module it is implemented in SystemC. This module gets data from the STP-ISS and can inject errors into a valid packet depending on a testing scenario. Error generation module is able to:

- Distort the transmitting data;
- Delete the service packets;
- Delete the EOP/EEP symbols;
- etc.

The general architecture of the implemented Software-to-Hardware tester is shown in Fig. 6.

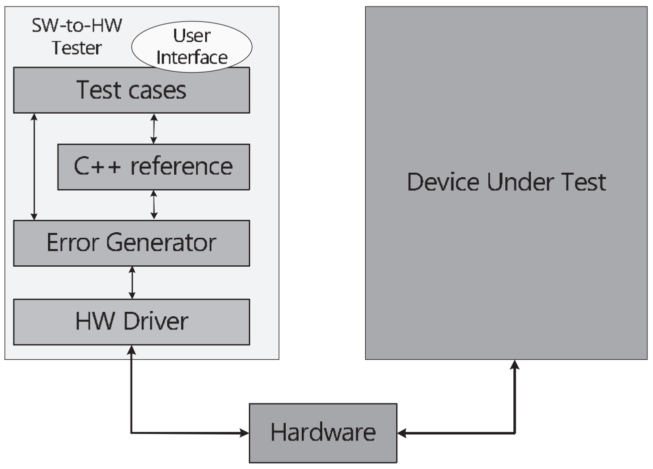


Fig. 5. Software-to-Hardware tester architecture

This software part of the tester should be installed on the PC. Current version of S2HT operates under the Ubuntu operation system.

PC should be connected to the DUT via the SpaceWire cable. The Star-Dundee SpaceWire Brick Mk2 is used to connect the PC to the SpaceWire device [19]. SpaceWire Brick Mk2 provides a special driver (HW Driver in Fig. 5) with an API for sending and receiving SpaceWire packets and time-codes.

The SpaceWire Brick Mk2 is an interface device that provides the essential capabilities now demanded of SpaceWire test and development equipment. The SpaceWire Brick Mk2 allows the user to easily connect their PC to a SpaceWire device or network through a USB port. It provides two SpaceWire interfaces, the ability to act as a time-code master, support for high speed data transfer, the capability to inject various types of errors on demand, and comes complete with highly optimised host software support for low latency transmission of SpaceWire packets directly to and from the host PC.

The other side of the connection is the DUT. This device should have the SpaceWire port and should satisfy the following general requirements:

- implementation of STP-ISS at least rev. 1;
- SpaceWire packets sending and receiving functionality;
- indication of data packet or command reception;
- implementation of a SpaceWire link interface.

The DUT can be represented by the following devices:

- real on-board equipment which is a “black-box” in the sense that we do not have a model of it, thus, can rely only on its observable input/output behavior;
- PC with the SpaceWire interface (including a special SpaceWire networking board).

If we use an other PC as the DUT, we have some additional abilities for testing and verification. We can set up the reference code of STP-ISS to DUT and observe, how both sides of the connection communicate with each other via the SpaceWire link. The other beneficial option is to test the real VHDL implementation of STP-ISS IP Core [20], but for this purpose we should implement a special test environment.

Fig. 7 shows the Software-to-Hardware tester that is implemented in our laboratory. This tester consists of a laptop with a pre-installed Ubuntu OS and Test Software. This laptop is connected to a SpaceWire Brick Mk2 via USB cable. The DUT is connected to the Brick via SpaceWire cable.

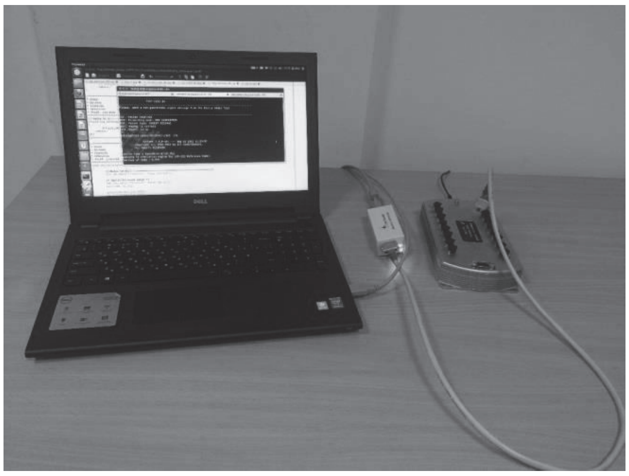


Fig. 6. SUAI Software-to-Hardware tester for STP-ISS

VI. Application of Software-to-Hardware Tester

Current version of the Software-to-Hardware tester is able to test the following mechanisms of STP-ISS:

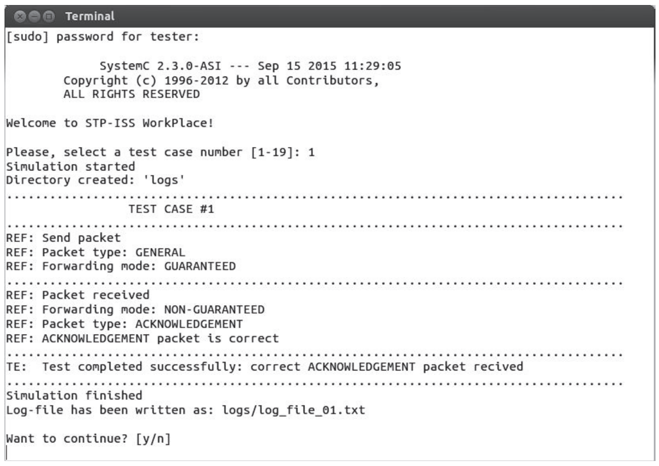
- assembling and disassembling of STP-ISS user data packets and service packets;
- data transmission mechanisms;
- best-effort quality of service;
- guaranteed quality of service;
- SpaceWire time-codes transmission and reception.

Testing should focus not only on normal protocol operation checking, but also on operation in exceptional and critical situations.

There is a number of STP-ISS mechanisms the Software-to-Hardware tester is not able to test:

- Receiving and Transmitting of SpaceWire distributed interrupts and interrupt acknowledges, which are not supported by a SpaceWire Brick Mk2;
- Settings of configuration parameters for DUT;
- Any problems in SpaceWire link-level functionality and other SpaceWire equipment errors (e.g. SpaceWire cable and Brick Mk2), because it is out of S2HT scope.

Fig. 5 shows a terminal window of launched S2HT software with a selected test scenario #1.



```
Terminal
[sudo] password for tester:

        SystemC 2.3.0-ASI --- Sep 15 2015 11:29:05
        Copyright (c) 1996-2012 by all Contributors,
        ALL RIGHTS RESERVED

Welcome to STP-ISS WorkPlace!

Please, select a test case number [1-19]: 1
Simulation started
Directory created: 'logs'
..................................................................................
                  TEST CASE #1
..................................................................................
REF: Send packet
REF: Packet type: GENERAL
REF: Forwarding mode: GUARANTEED
..................................................................................
REF: Packet received
REF: Forwarding mode: NON-GUARANTEED
REF: Packet type: ACKNOWLEDGEMENT
REF: ACKNOWLEDGEMENT packet is correct
..................................................................................
TE:  Test completed successfully: correct ACKNOWLEDGEMENT packet recived
..................................................................................
Simulation finished
Log-file has been written as: logs/log_file_01.txt

Want to continue? [y/n]
```

Fig. 7. Execution of the test scenario #1

Test scenario #1 is intended to check assembling and disassembling of STP-ISS user data packets and service packets mechanisms of the DUT. The tester sends a guaranteed packet to the DUT and waits for an acknowledgement. If a correct acknowledgement is received, then this mechanism is correctly implemented inside the DUT.

During the S2HT exploitation the user should be always sure that all the equipment is correctly connected and set. Moreover, if DUT is not able to send data, thus some of the test scenarios could not be executed successfully, because the tester needs a response form the remote side of the connection.

VII. Conclusion

The implemented Software-to-Hardware tester is a promising tool that could help the developers to ensure that STP-ISS equipment operates correctly. The implemented list of testing scenarios should give the full test coverage for the testing devices, so the result of the tester exploitation should be simple – true or false. That means, did the DUT successfully passed all the tests or not. If there are any faults in particular test scenarios, then the developer could analyse log-files and find out, which mechanism is implemented incorrectly.

Current implementation of a tester is able to test the equipment that operates in conformance with STP-ISS protocol specification rev.1. So the work that is still need to be done in this field is updating the tester to the 2nd STP-ISS revision conformance.

VIII. Acknowledgement

The research leading to these results has received funding from the Ministry of Education and Science of the Russian Federation under the contract RFMEFI57814X0022.

References

[1] P. Marwedel, *"Embedded System Design"*, Springer, 2006. 241 p.

[2] Martha Gray, Alan Goldfine, Lynne Rosenthal, Lisa Carnahan. "Conformance Testing", *National Institute of Standards and Technology*, Gaithersburg, USA, 2010.

[3] Tektronix official website, HDMI compliance test software, Web: http://www.tek.com/datasheet/product-software/options-hdm-hdm-ds-hdm-dsm-ht3-and-ht3-ds-datasheet-1.

[4] Rohde&Schwartz official website, R&S®CMW - Conformance testing solution for eCall/ERA-Glonass, Web: https://www.rohde-schwarz.com/en/applications/r-s-cmw-conformance-testing-solution-for-ecall-era-glonass-application-card_56279-106883.html.

[5] A. Krupp, W. Muller "A Systematic Approach to the Test of Combined HW/SW Systems", in *Proc. Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Paderborn University / C-LAB, Paderborn, Germany, 2010, pp. 323 – 326.

[6] H. Kahlouche, C. Viho, M. Zendri "Hardware Testing Using a Communication Protocol Conformance Testing Tool", *TACAS/ETAPS'99, LNCS 1579, Springer-Verlag*, Berlin Heidelberg, 1999, pp. 315-329.

[7] Star-Dundee website, SpaceWire Conformance Tester, Web: https://www.star-dundee.com/products/spacewire-conformance-tester.

[8] Koopman, P., Chakravarty, T. "Cyclic *redundancy code (CRC) polynomial selection for embedded networks". DSN '04 Proceedings of the 2004 International Conference on Dependable Systems and Networks*, IEEE Computer Society, 2004. pp. 145-154.

[9] Tanenbaum, A. S., *Computer Networks*, Fifth Edition; Prentice Hall, 2011. 962.

[10] Tomlinson, R.S., "Selecting Sequence Numbers", *Proceedings of the ACM SIGCOMM/SIGOPS Interprocess Communication Workshop, and ACM Operating Systems Review*, Vol. 9, No. 3, July 1975, Association for Computing Machinery, New York, 1975.

[11] Y. Sheynin, V. Olenev, I. Lavrovskaya, I. Korobkov, D. Dymov "STP-ISS Transport Protocol for Spacecraft On-board Networks", *Proceedings of 6th International Conference SpaceWire 2014 Program*, Athens, Greece, 2014, pp. 26-31.

[12] Y. Sheynin, V. Olenev, I. Lavrovskaya, I. Korobkov, S. Kochura, S. Openko, D. Dymov "Second Revision of the STP-ISS Transport Protocol for On-Board SpaceWire Networks", *Proceedings of 17th Conference of Open Innovations Association FRUCT*. Yaroslavl: Russia, 2015. pp.192-200.

[13] D. Dietterle, *"Efficient Protocol Design Flow for Embedded Systems"*, PhD thesis in Computer Science. Cottbus, 2009.

[14] Y. Sheynin, V. Olenev, I. Lavrovskaya, I. Korobkov, S. Kochura, S. Openko, D. Dymov "STP-ISS Transport Protocol Overview and Modeling", *Proceedings of 16th Conference of Open Innovations Association Finnish-Russian University Cooperation in Telecommunications (FRUCT) Program*. Oulu: University of Oulu, 2014. pp.185-191.

[15] P. Morozkin, I. Lavrovskaya, V. Olenev, K. Nedovodeev, "Integration of SDL Models into a SystemC Project for Network Simulation", in F. Khendek et al. (Eds*.), SDL 2013: Model-Driven Dependability Engineering, Lecture Notes in Computer Science, Volume 7916* (pp. 275-290). Berlin: Springer Berlin Heidelberg, 2013.

[16] B. Stroustrup, *The C++ Programming Language*, 4th Edition. USA: Addison-Wesley, 2013.

[17] M. Krichen, S. Tripakis, "Black-Box Conformance Testing for Real-Time Systems", *11th international SPIN workshop on model checking of software (SPIN'04)*. LNCS, vol. 2989. Springer, Berlin.

[18] D. Black, J. Donovan, B. Bunton, A. Keist, *"SystemC: From the Ground Up"*, NY: Springer, 2010.

[19] Star-Dundee website, SpaceWire-USB Brick Mk2, Web: https://www.star-dundee.com/products/spacewire-usb-brick-mk2.

[20] A. Ben Abdallah, *"Multicore Systems On-Chip: Practical Software/Hardware Design"*, Second Edition. Atlantic Press, 2013.