SystemC-model for GigaSpaceWire Protocol Simulation

Dmitry Kuznetsov, Valentin Olenev
Saint-Petersburg State University of Aerospace Instrumentation
Saint Petersburg, Russia
{dmitry.kuznetsov, valentin.olenev}@guap.ru

***Abstract*—The paper describes implementation process of GigaSpaceWire protocol simulation System-C model. At current point there are no available or implemented simulation models of this protocol. Implementation of the GigaSpaceWire model will offer the possibility of protocol testing and validation. After the model description, the paper provides results of the GigaSpaceWire protocol specification validation, obtained during protocol simulation.**

I. INTRODUCTION

GigaSpaceWire [1] is an onboard communication protocol aimed to provide abilities to use gigabit link technologies and galvanic isolation for SpaceWire networks. For achieving this goal, GigaSpaceWire replace DS encoding schemes with 8b10b encoding. With 8b10b encoding using of Fibre and Serial RapidIO [2] channels become possible. Therefore, it becomes possible to implement galvanic isolation, increase data transfer rate and link cable length. However, adding of 8b10b encoding in SpaceWire channel causes significant changes in such basic elements of SpaceWire links as state machine, silence exchange procedure, flow control mechanism, encoding of SpaceWire symbols and complete replacement of physical layers.

The GigaSpaceWire PHY layer transmits and receives raw bit sequences over a physical link. Before starting to receive of data symbols, the PHY layer receiver establishes a bit synchronization. When the bit synchronization is achieved, performing a deserialization and a symbol alignment become possible. The symbol alignment allows to be sure, that 10b sequences after deserialization corresponds to GigaSpaceWire symbols. For performing the symbol alignment GigaSpaceWire use special symbols called Comma symbols. Comma symbols are sent with some delay, called a Comma-time and receiving of the Comma symbol trigger symbol alignment correction process. The GigaSpaceWire state machine has the same states as the SpaceWire state machine, but there are some changes in the rules for entering states and states operations. For the connection establishment GigaSpaceWire uses Comma symbols instead of IDLE symbols (which are SpaceWire NULL symbols) and static time delays replaced with calculated delays, basing on a value of the Comma time [3]. Comma symbols in GigaSpaceWire also used in the silence exchange process. This is due to the symbol alignment, that should be performed before the re-connection process. The GigaSpaceWire flow control protocol is the same as a flow control, which defined in the SpaceWire specification, but sets different values for parameters, expanding a maximum value of credits counter. But for support connections, where GigaSpaceWire links connects two SpaceWire nodes (means of GigaSpaceWire-SpaceWire bridges), the GigaSpaceWire specification allows to operate flow control with SpaceWire flow control values. Also, for supporting backward compatibility with SpaceWire networks, GigaSpaceWire does not affect the upper layers of the SpaceWire protocol (transport, network, and packet). In another words, GigaSpaceWire is replacement for only the lower layers in SpaceWire networks.

At this moment there are no implementations of GigaSpaceWire simulation models. Generally, only two GigaSpaceWire implementations were found in the public domain – a SDL formal implementation [4] and a VHDL implementation for FPGA and ASIC [5].

The SDL implementation is intended to describe the structure and functioning of the protocol. Specifications and descriptions on the SDL language allow to analyze and interpret a protocol specification uniquely. The advantage of the SDL language is in the structuring tools inherent in it, which facilitate description of a protocol. Significant disadvantages of the SDL model are that this model is implemented only on an example of one network topology and changing of the network topology is difficult process, with SDL models can work only well-trained specialists with knowledge of the SDL language.

Another implementation is the VHDL implementation for FPGA and ASIC. VHDL is actually a formal description of the final GigaSpaceWire implementation for FPGA and ASIC. The VHDL language allows to simulate developed components and results of simulation are the closest to a real device behavior. Disadvantages are in high cost of software developing in the VHDL language, dependence in deep knowledge of the hardware description languages and principles of FPGA and ASIC operations. The most significant disadvantages are in dependencies in specialized an IDE for modification and executing VHDL models and difficulties with files input\output operations. Compared with SDL models, changing of a network topology in the VHDL implementation is more accessible, however, constructing complex topologies that are characteristic of more real-life onboard networks is difficult.

SystemC [6] is a library for C++ programming language. This library provides necessary tools for modeling hardware systems at various levels of abstraction: a simulation time, events and event management mechanisms, special data types for describing devices and organizing communication between them. At the same time, it's possible to use all features of the C++ language object-oriented developing. Since C++ is a

common and widely applicable programming language, modifying a SystemC source code is a relatively simple task. Also, software written in C++ is quite easy to embed in more complex integrated software solutions, such as CADs. Developed SystemC models possible to compile as stand-alone programs, allowing to use SystemC models without any required IDE.

As already mentioned above, the GigaSpaceWire protocol is a replacement for the lower layers of the SpaceWire protocol. Based on this feature, it was decided to take as a basis for development already developed a SpaceWire protocol model in SystemC, replacing the lower layers in this model with GigaSpaceWire levels. One of these implementations is a software component for SpaceWire networks simulation [7] of a SANDS CAD system [8]. SANDS consist of 4 software components, which solving different tasks of the SpaceWire network design. Detailed information about SANDS is described in Chapter V. In current paper we interested in "Component #4" us the fourth of SANDS components, that is in order for the SpaceWire SystemC simulation.

II. Component for SpaceWire networks simulation

A key point for choosing this Component as basis for the GigaSpaceWire implementation is that the SpaceWire simulation core implementation is based on the SystemC language. Also, the architecture of this implementation has a modular structure.

Component #4 has two network simulation modes: Bit level — a simulation of the full stack of the SpaceWire network with transport protocol or application, and Packet level — a simulation of the upper layers only: network, transport, and applications. Fig. 1 shows difference between simulation modes. The Bit level simulation mode based on the SystemC language model time mechanism. This is done in order to achieve more accurate comparison between model behavior and hardware behavior. The Packet level simulation mode based on the SystemC language event mechanism. In this mode events, triggered by various functions, trigger other functions. Such modeling ignores some functions of SpaceWire, which greatly simplifies logic of the model.

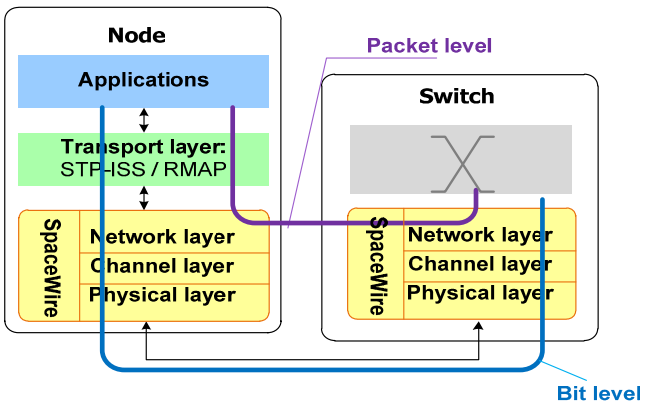

Fig. 1. Two simulation modes in SANDS

The architecture of Component #4 is implemented in such way that any object in the simulation network is an implementation of a pure virtual base class by inheriting from this class [9]. Objects essentially are network device models (nodes, channels, routers), messages, device components (ports, channel interfaces, application layers, etc.). It is possible to implement a new version of class with deferent functionality and behavior. Almost each class has its own configuration parameters. Each parameter is set up with default parameter during a topology construction process. However, it is possible to change all parameters by means of a special configuration file. The configuration file stores not only parameters value, but also the topology description. In fact, Component #4 is a bundle of classes and methods used to build the topology of particular network.

Component #4 perform only simulation and does not consist with a GUI. The network topology setting up with the special configuration file. Such configuration file can be created manually or in a specialized CAD system, such as SANDS. Results of simulation saving in a special encoded format. Each simulation event record created according to specific rules. Such way in creating events history allows to reduce computer resources usages and log-file size. Reduced load on computer in turn allows to save events records almost immediately, eliminating data loss. Cause encoded log events unreadable at all, it is required in special tools for decode log-files. On the other hand, log-files are compact and could be easily copied to another computer or sent to another engineer. In addition, it is possible to modify a view of simulator events notes during decoding process: change types of viewing events, language, etc.

III. GigaSpaceWire implementation process

The implemented protocol model should have a backward compatibility with the upper layers of the SpaceWire network, replacing only the lower layers. During the development, this particular feature of protocol was taken into account. Implementation work was divided on following subtasks:

- Implementation of GigaSpaceWire state machine;
- Implementation of 8b10b encoding and decoding;
- Implementation of messages class;
- Implementation of GigaSpaceWire protocol channel messages;
- Implementation of bit messages class;
- Implementation of simulated noisy channel;
- Adaptation of log system.

At the current point, developing of the GigaSpaceWire simulation model is performed only for more detailed the Bit level simulation mode. For this mode, the simulation model is implemented as close as possible to the specification, which will also allow to validate the protocol specification.

During researching of the Component #4 SpaceWire model implementation architecture (see Fig. 2), class with implementation of a basic channel interface was chosen as the starting point for the GigaSpaceWire implementation. This

class receives and transmits symbols from/to the channel, contains implementation of the state machine, the mechanism for silence exchanging, the exchanging SpaceWire service and information symbols at a higher level.

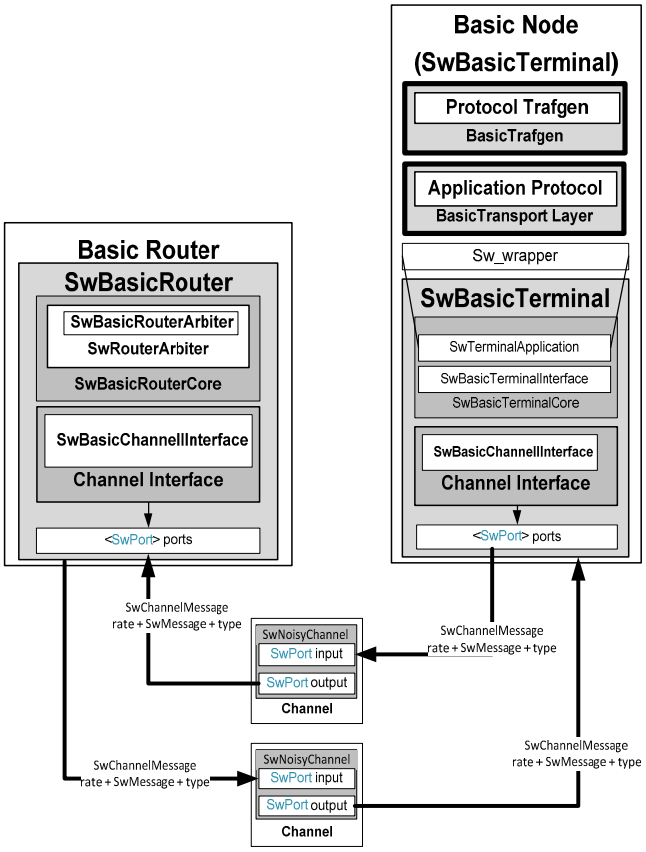

Fig. 2. Component #4 class architecture

A channel class is implementation of a half-duplex channel with two buffers for each direction of data transfer – input and output. Between input and output buffers a message corruption mechanism is implemented. The message corruption mechanism inverts a one random message bit in accordance with a specified Bit Error Rate (BER). Messages are transmitted as instances of a channel message class. In other words, messages in the channel are transmitted not in form of the bit-wide stream, copying values of bits from one level to another, but in the form of the message class instance pointers that were previously created during simulation. This solution is quite rational - unlike real networks, unpredictable events (such as effects of electromagnetic waves or radiation on the communication channel) cannot be happened during modeling. In addition, this solution allows reduce the workload of computer and speed up the simulation process. However, for inverting a random bit of symbol, a transmitted symbol must be represented in the form in which it is transmitted in a real device. This type, or rather the class corresponding to this type in Component #4, is called a bit message. After inverting a random bit, basing on the new sequence, the new instance of the message class should be created, it will be placed to output buffer.

For messages transmission it is necessary to implement a channel message class in addition to the message class. The channel message is a class, which fields are a transmitting message and an additional information for the message corruption mechanism operation in the channel — information about a delivery status, a channel speed, a type of error after detecting a message corruption. Since GigaSpaceWire sends 10b code sequences in the channel, which are result of the 8b10b decoder operation, it was decided to add these sequences to channel message fields. 10b code sequences are necessary not only for modeling of 8b10b decoder operations, but also for operations in the channel with simulated noise.

The implemented class of GigaSpaceWire message (in comparison with the same class from the SpaceWire implementation) has following innovations: functions for creating GigaSpaceWire characters basing on SpaceWire characters and vice versa, functions for creating GigaSpaceWire character. GigaSpaceWire characters is the form of ZXX.Y (for 8b sequences) or Zxx.y (for 10b sequences), where Z is the character type flag (where K means control character, D means data character), XX is subgroup of the 5 most significant bits of character, Y is subgroup of the 3 least significant bits, xx is subgroup of the 6 most significant bits of character, y is subgroup of the 5 least significant bits. These sequences were implemented as a separate class to simplify work with the 8b10b encoder and simplify modifications in the log system for the GigaSpaceWire model. For correct operation of the Component #4 log system, functions for internal SpaceWire and GigaSpaceWire messages identification numbers forced assignment was added. This ID is created automatically by the simulator and is used to track message transferring on the network. For supporting a backward compatibility with the upper layers, GigaSpaceWire characters are created based on previously created SpaceWire characters. After receiving of GigaSpaceWire symbols from the channel new SpaceWire characters are created for transfer. Thus, for each character, transmitted over the GigaSpaceWire network, the internal ID changes at least 2 times, it can cause some failures in log functions operation.

During the implementation of the GigaSpaceWire model, small changes also occurred in the bit message class. Bit messages are created on the base of 10b-code sequences; after a bit corruption the new 10b-code sequence is created on base of the new bit message.

It should be noted, that the simulation model of the GigaSpaceWire protocol does not implement signal and physical levels. This is due to principle of channels operation in Component #4, described previously. Implementation of serialization, deserialization and bit synchronization are not required.

One of main innovation in GigaSpaceWire is a modified state machine. The main changes of the state machine are in use of new Comma characters for establishing connections and detecting a disconnection. In addition, it uses timers with calculated values instead of static timers. These values are calculated based on a transmission time of two Comma characters.

Implementation of the 8b10b encoder was completed as functions of encoding and decoding. Component #4 by default creates a new class instance for each device in a network topology. However, encoding and decoding functions for each network elements are the same. Moreover, for the 8b10b encoder working it's also requires in special tables for encoding/decoding. Creating a new instance of the 8b10b encoder class will re-initialize these tables, allocating more memory for their storage. To exclude possible non-optimal usage of memory, as well as possible slowing down when performing the simulation, the class of 8b10b decoder was created with using a Singleton pattern [10]. In this case, the 8b10b decoder class initializing once and all next class instances initialization will have access to the first created 8b10b class instance. Similar instances of the 8b10b encoder class are creating in each GigaSpaceWire basic channel interfaces and they get access to encoding and decoding functions. Input arguments for encoding and decoding functions are an input sequence, a RD balance value and an encoding/decoding error status variable pointer. Thus, decoding tables are initialized once.

At the current stage of development, it was decided to make minimal changes in the log system for the simulation model of the GigaSpaceWire protocol. In the Component #4 log system was added only information about 8b10b encoder operations, names for GigaSpaceWire character types and state names of the GigaSpaceWire state machine. This information is minimal and more than sufficient for testing and debugging of the developed protocol model.

IV. GIGASPACEWIRE SPECIFICATION ISSUES

During the implementation of the state machine, in the latest revision of the specification, following errors and inaccuracies were noticed. The specification does not imply a clear description for a DiscDetect parameter setting during the connection establishing process.

In the specification defined following parameters:

- CommaTime - time interval between transmission of two comma symbols;
- DiscDetect - time interval during which a connection error is determined;
- wait_conn_time - time interval for establishing a connection;
- etc.

Value of the DiscDetect parameter should be at least twice greater than the CommaTime value. Value of the wait_conn_time parameter must be no less than 8 times value of the CommaTime parameter. According to the specification, the wait_conn_time should be set each time, when we enter any state of the state machine. However, the text does not specify the DiscDetect value setting information. It is known that "… if the receiver of one of the sides did not accept the Comma symbol during the DiscDetect time interval, this event should be recognized as detecting of a disconnection error in the exchange level of this side …". Considering that "… A receiver error event is a connection error or a decoding error …", we can conclude, that the DiscDetect timer expiration – is a part of the receiver error in the state machine. Moreover, the receiver error and the wiat_conn_time are defined as events for entering the "Reset" state.

Unclear DiscDetect setting problem is especially arising in the following situation: according to the specification "... if the first Comma symbol is accepted in "Started" state, the Got_Comma condition should be considered as fulfilled. If the state machine is in the "Started" state and the Got_Comma condition is fulfilled, then the state machine must enter the "Connecting" state. …". If the DiscDetect parameter is setting with the channel enabling or even when entering the "Started" state together with the wait_conn_time timer, a number of "unsuccessful attempts" to fulfil the "Got_Comma" condition is limited by value of the DiscDetect parameter, and the wait_conn_time parameter becomes not working in principle, since the value of this parameter is many times greater than the value of the DiscDetect parameter. Such "unsuccessful attempts" could be a symbol alignment error or a decoding error due to the corruption of the first transmitted Comma symbol.

In addition, there is another unclear definition of the DiscDetect parameter. The specification defines: "... after a channel turning on or cold reset, duration of the DiscDetect time interval shall be 65 symbols..." As mentioned earlier in the article, the DiscDetect is twice as large as the CommaTime, when the CommaTime takes value equal to the transmission time from 8 to 128 symbols. The wording "… the DiscDetect duration shall be 65 symbols …" create ambiguity for the operation of calculating value of this parameter after channel switching on or a cold reset: a value must be calculated as the sending time of 65 symbols or as the CommaTime value if this was equal to transmission time of 65 characters, multiplied by two. In the first case, if the CommaTime value is longer than the transmission time of 66 symbols, the DiscDetect timer will always fire, triggering the connection error.

Not entirely clear the state machine's reaction on a detection of the decoding error. According to the specification "... if the received code sequence is not found in the 8b10b decoding table, then it should be considered incorrect and mean that a decoding error of the 8b10b decoder has occurred. NOTE - Detecting a decoding error does not necessarily mean that a decoded code sequence contains an error. A decoding error may be the result of an error that occurred earlier, but was not detected by the decoder and changed value of a RD balance ..." Also, "... Information about an error of the 8b10b decoder from the encoding level and information about a decoding error from the symbol level should be sent to the exchange level ...". As mentioned earlier, a receiver error and a decoding error are a part of a connection error. Also, according to the specification, "… A decoding error is the 8b10b decoder error or the symbol level decoder error. If the receiver is turned on and the Got_Comma condition is set, then a decoding error must be processed by the exchange level. If a decoding error is detected, the exchange level, the state machine should enter the "Restart" state ...". The wording "… A decoding error must be processed by the level of exchange" introduces ambiguity - it is not clear what does mean "processed". Regarding to the above mentioned, "processing" means the reaction of the state machine on a receiver error. A receiver error is an event for entering the "Restart" state of the state machine. The condition "Got_Comma" is charged when transition from the "Starting"

state to the “Connecting” state, which means transition from the “Ready” and the “Starting” state to the “Restart” state should not be carried out. Therefore, in these states, when a decoding error occurs (for example, as a result of distortion of one character in the channel), a value of the RD balance will be changed and lead to decoding errors of initially correct sequences until the wait_conn_time will not have triggered.

V. SANDS AND GIGASPACEWIRE INTEGRATION

SANDS is a computer-aided design system for SpaceWire networks. Essentially, it is a software tool that accompanies an engineer during onboard space networks development process. SANDS include four main components:

- Component #1: A component for onboard network topology design and evaluation of its structural characteristics [11, 12];
- Component #2: A component for tracking of the non-intersecting routes for the data transmission in a network [13];
- Component #3: A component for generation of the scheduling table for the STP-ISS transport protocol for the transmission of the data with Scheduled quality of service [14];
- Component #4: A component for simulation of the network operation with all the data that component got from other 3 components and graphical user interface.

This software is applicable at every stage of network development.

Graphical user interface (GUI) developed in the scope of VIPE project [15]. The graphical interface provides onboard network designer interaction with the system, including construction of network, configuration interface to set up each device parameters and abilities to control it and simulate. GUI allows to build network interactively from abstract devices - nodes, channels, routers. It is enough for user to set configurable parameters for each device - designed network will be exported to intermediate representation format to be used in other SANDS component (see Fig. 3).

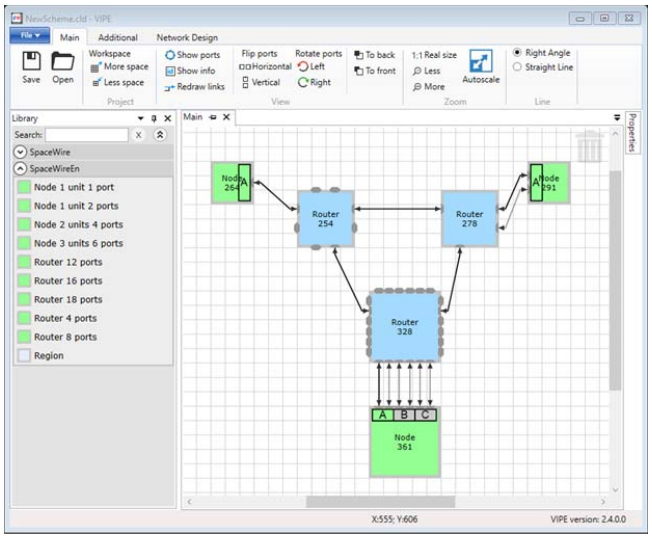

Fig. 3. SANDS graphical user interface

Based on the created topology and configuration parameters, the network can be modeled, and simulation results are displayed in a form of resulting tables. The process of simulation itself is hidden from a user, cause all useful information and results can be available only after simulation ending. Also, before the simulation it is possible to check a corresponding checkbox from a list of different available simulation logs: RMAP [16] log, STP-ISS log, SpaceWire network information in deferent layers and etc. All selected logs will be shown in a window with resulting tables (see Fig. 4).

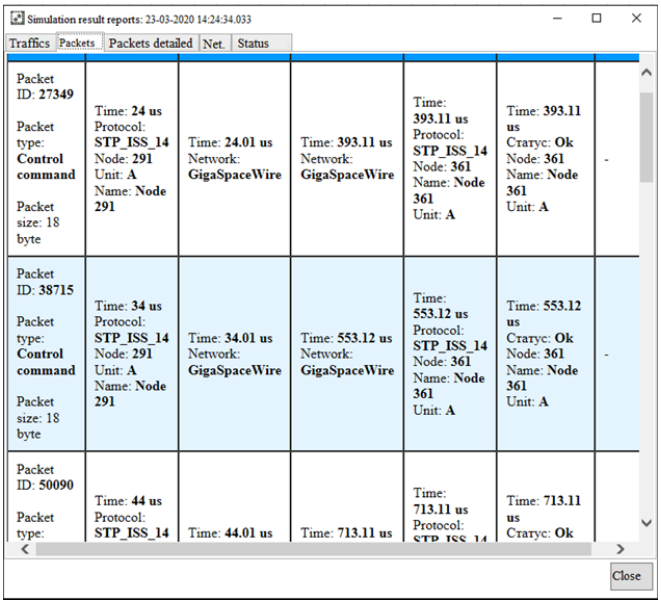

Fig. 4. Results of network simulation

As experiment, the developed simulation model was integrated in SANDS CAD. Cause developing of the GigaSpaceWire protocol model was done based on Component #4, the only one way to integrate the protocol model in CAD is only replacing the original Component #4 with modified. For integrating of developed model into CAD system, in VIPE was added necessary fields for switching a type of required protocol and fields in resulting tables. Also, in the procedure of the topology constructing in Component #4 was made some correction. Type of a protocol used in device (SpaceWire or GigaSpaceWire) in the configuration file is stored in one of fields in a "Port" section and the topology construction process starts from the upper levels. Because of this, it is not clear what type of the basic channel interface to use during device creation. Additional difficulties cause the fact that the basic channel interface was initialized from a constructor of a Node/Router core - from these constructors there is no access to the configuration file. To solve this problem, a port name was added to a kit name of a configurable device for determining which type of the basic channel interface that kit corresponds. It is also important to note that the basic channel interface for both nodes and routers is the same.

During the GigaSpaceWire protocol testing, the implemented model was checked for possible problems described in Chapter IV. If in the “Start” state the first comma

symbol is corrupted and the state machine does not enter the "Restart" state – the RD value, after detecting an error, can become, for example, from the expected value "1" to the value "-3". Next received symbols decoding can be complete with the RD- decoding table value, when the encoding of these symbols was performed with the RD+ value. However, in the model, transmitted IDLE symbols were coded and decoded with a constant value of the RD balance (the IDLE symbol n-1 is encoded and decoded with RD+, the symbol n is encoded and decoded with RD +, the IDLE symbol n + 1 - with RD +). This feature allows to "restore" the value of the dynamic balance and continue a network operation in normal state.

The network model was also tested for a situation, in which one of the routers operate as the SpaceWire – GigaSpaceWire bridge. Test was performed using STP-ISS protocol [17] messages with acknowledgment. This topology option also turned out to be working, although initially this goal was not set. Test results confirms a correct operation of the implemented GigaSpaceWire protocol model, a correct transmission of characters from GigaSpaceWire protocol levels to SpaceWire protocol levels, and vice versa.

VI. Conclusion

During this work, was implemented the simulation model of the GigaSpaceWire protocol on the SystemC language. Comparing with previously created GigaSpaceWire protocol models, developed simulation model can be added to other software, such as CAD. Integration is possible only as modification for Component #4 of SANDS CAD. At the moment, the developed model is preparing for integration in SANDS CAD. Integration of developed model will not only expand functionality of the SANDS CAD systems but will also allow researchers to conduct experiments with a relatively fresh protocol that can replace existing solutions. Attracting the attention of space vehicle network designers to the GigaSpaceWire protocol, can accelerate process of developing protocol specification and its preparation for implementation in onboard computer networks. In comparison with already implemented models, the model described in this article has two significant advantages. Implemented SystemC model quite flexible and allows making experiments with different and difficult network topologies. Comparing with the SDL implementation, the topology construction process can be proceeds in an intuitive understandable graphical user interface, which is much simpler. In addition, comparing with the VHDL implementation, the SystemC model allows to use files for input and output. Thus, SystemC model configuration parameters are set by means of input file and results are stored in output files. Accessibility to files I/O streams makes interaction with the developed model much simpler and reliable. In general, the described simulation model quite portable – a topology description, results of simulation and the simulator its own are standalone files, that can be shared between specialists. Speaking about disadvantages, the developed simulation model doesn't take in a count the physical layer of GigaSpaceWire networks. Thus, the GigaSpaceWire protocol is replacement of the SpaceWire lower layers, the developed model ignores significant part of the protocol specification. The GigaSpaceWire model provides the possibility to study and examine exchange and decoding layers only.

Acknowledgment

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

References

[1] *Interfaces and protocols of high-speed inter-instrument information exchange and integration of spacecraft onboard systems. SpaceWire-RUS*. GOST R. Moscow, Standardinform, 2018

[2] *RapidIO Interconnect Specification Part 6: LP-Serial Physical Layer Specification Rev. 2.2*., RapidIO Trade Association, June 2011.

[3] Yablokov, E., "GigaSpaceWire - Gigabit Links for SpaceWire Netwroks" / E. Yablokov, Yu. Sheynin, E. Suvorova et al // *SpaceWire-2013. Proceedings of the 5th International SpaceWire Conference*, Gothenburg 2013. Space Technology Centre, University of Dundee, Dundee, 2013, pp. 28-34

[4] Stepanov, V. E., "Verification of the draft Russian standard SpaceWire-RUS using the SDL language", graduation project (work) / V. E. Stepanov, I. Ya. Lavrovskaya; Saint Petersburg State University of Aerospace Instrumentation. - St.Petersburg, 2016 Accession No. 1739-16B S 79

[5] Yablokov E.N., "Methods of research and development of network controllers of the channel level for high-speed onboard computer networks of spacecraft". The Dissertation of the candidate of technical sciences. Sciences: 05.13.15: protected 30.05.2019: approved. 10/31/2019. St. Petersburg.

[6] *IEEE Standard for Standard SystemC® Language Reference Manual*. IEEE, January 2012

[7] V. Olenev, I. Lavrovskaya, I. Korobkov, N. Sinyov and Yu. Sheynin, "Hierarchical simulation of onboard networks", Intelligent Distributed Computing XIII.191-196 pp.

[8] Olenev V.L., Lavrovskaya I.I. "Computer-aided design system for on-board SpaceWire networks simulation and design", *Proceedings of SUAI Scientific session*, Part 1, Technical sciences/ Saint Petersburg, SUAI, 2017.– C. 160-173.

[9] A. Eganyan, L. Koblyakova, E. Suvorova. "SpaceWire network simulator", *SpaceWire-2010. Proceedings of the 3rdInternational SpaceWire conference*, St.Petersburg, 2010, pp. 403-406.

[10] The "Gang of Four": Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. AddisonWesley Professional, 1997.

[11] Lavrovskaya I., Olenev V., Korobkov I. "Fault-Tolerance Analysis Algorithm for SpaceWire Onboard Networks", *21st Conference of Open Innovations Association FRUCT*, University of Helsinki, Helsinki, Finland, 2017, pp. 217-223.

[12] Lavrovskaya, I., Olenev, V., "Network Topology Transformation for Fault Tolerance in SpaceWire Onboard Networks*", 22nd Conference of Open Innovations Association FRUCT*, Jyvaskyla, Finland, IEEE, 2018, pp. 131-137.

[13] Kurbanov, L., Rozhdestvenskaya, K., Suvorova, E. "Deadlock-Free Routing in SpaceWire Onboard Network". *22nd Conference of Open Innovations Association FRUCT*, Jyvaskyla, Finland, IEEE, 2018, pp. 107-114.

[14] I. L. Korobkov, N. Y. Chumakova, "Algorithm of scheduling-table's design for STP-ISS transport protocol", *Proceedings of SUAI Scientific session*, Part 1, Technical sciences, Saint Petersburg, SUAI, 2019, pp 198-204.

[15] Syschikov, A., Sheynin, Y., Sedov, B., Ivanova, V. "Domain-specific programming environment for heterogeneous multicore embedded systems", *International Journal of Embedded and Real-Time Communication Systems, Volume 5, Issue 4*. 2014, pp. 1-23.

[16] ESA. Standard ECSS-E-ST-50-52C, SpaceWire — Remote memory access protocol. Noordwijk : *Publications Division ESTEC*, February 5, 2010.

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