# RMAP Protocol SystemC Model: Detailed Description and Modelling Features

Nikita Martynov
St-Petersburg University of Aerospace Instrumentation
190000, St-Petersburg, Bolshaya Morskaya 67, Russia
Nikita.Martynov@guap.ru

**Abstract**

Modelling becomes more and more important in the communication protocols development flow. It is a powerful tool in the hands of developers. By modelling the detailed validation and verification of the project could be done. It is applicable from the conceptual design stage to the physical implementation of the final product. Modelling helps to find the weak spots of standards and fix them. Also it becomes possible to experiment by creating combinations of several specified standards models that superpose in single executable system model.

This article introduce to modelling itself and to modelling by means of SystemC. The outcomes which were obtained by me during RMAP protocol modelling are represented in the article. These comprise detailed description and organization of RMAP transport layer protocol SystemC model, application operating over the protocol and various challenges which appear during design and testing of such kind of systems are considered also. One of paragraphs tells about different protocol models combining as possibility to check consistency and test communication of protocols, for example of SpaceWire protocol stack which covers first three layers of OSI model including network layer, and transport layer RMAP model. The model of SpaceWire protocol stack was designed by our team also.

SpaceWire is a perspective and fast-developing communication standard. Nowadays there are number of large companies take part in the development process. The standard is supported and implemented to modern spacecrafts by European Space Agency, NASA (USA) and JAXA (Japan). RMAP – transport layer protocol which was developed for joint operation with the SpaceWire standard.

**Index Terms:** Modelling, protocols, SystemC, RMAP, SpaceWire.

## I. Introduction

Modelling takes an important role in the development process as a solution to perform detailed check of the specification and verification of the project to the stage of physical implementation of the final product. This allows detecting and fixing errors on the project specification stage, so it could decrease future improvement costs and it requires less time for corrections make. Modern modelling methods are very flexible. And it allows spending less efforts, time and money. Combining of a number of specified protocols into a single model, joint performance checking, advantages and disadvantages identifying – all these can be allowed by modelling use [1, 2].

Depending on pursued aims and expected modelling results different models are designed by various development tools. So by means of C/C++ high level abstraction behavioral models are designed which define basic components of the system and theirs mutual interaction.

During functional modelling, languages of hardware description as VHDL and Verilog are used. By means of them RTL-models (Register Transfer Level models) are designed. RTL-model is close to hardware implementation including all essential characteristics for the system testing before hardware implementation.

Generally behavioral model is designed firstly and after RTL-model is developed with it helps. There aren't possible to reuse already developed behavioral model during designing RTL-model because of inconsistencies of C/C++ high-level language and VHDL, Verilog hardware description languages. This demands extra time for the model redevelopment but by means of the lower-level language. SystemC modelling language based on C/C++ language and provides essential possibilities for lower-level models implementation, therefore SystemC helps to avoid wastes of time that mentioned thereinbefore [1, 5, 6, 7].

SystemC was used for the modelling of RMAP transport layer protocol which is considered in this article.

## II. Main part

### *A. SystemC modelling*

SystemC model consist of some main components: ports, interfaces, processes (term process is defined as SC_THREAD in this article), channels and internal data [6, 7].

Fundamental conception of SystemC modelling consists of several basic paradigms [5]:

- concurrency — possibility of simple element parallel operation;
- connectivity — information exchange support between blocks by means of interfaces, ports, channels and hardware signals;
- reactivity — event mechanism realization, then models respond to appointed system events;
- real time performance — possibility of systems modelling with real time slices.

There are several SystemC model types: clock-oriented, event-oriented and models which combine conception of previous two types.

Event-oriented models – these are models in which the appropriate process receives control after specified *sc_event* occurred. If the event didn't occur, the process will stay in a waiting state. If the event reoccurs, the appropriate process will regain control and begin to execute. The real quantity of control receiving by concrete process depends on specified event frequency occurrence [8].

*Sc_event* principal feature is an absence of a value and duration of time which is depicted on the diagram at fig. 1. The diagram shows an example of *sc_event* starting in three different instants of time.

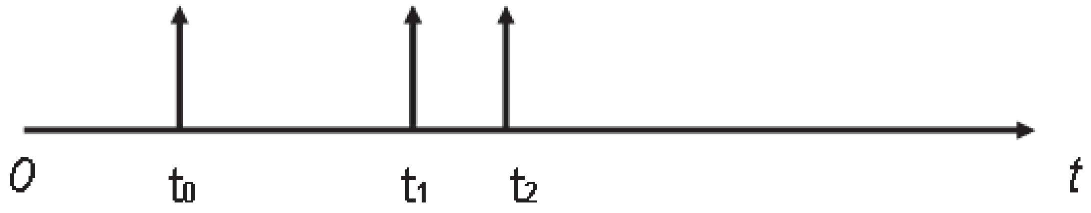

Fig.1. Time diagram of *sc_event* starting

Clock-oriented models – these are models in which the appropriate process receives control every time by getting a positive or negative edge of the *sc_clock* signal. The clock cycle (depict at fig. 2) and a modelling time value manipulates with the frequency of process control receiving and quantity of the process executions [8].

Thus very convenient to use clock signal for control transfer then some actions have to be performed throughout whole modelling time with constant time domains.

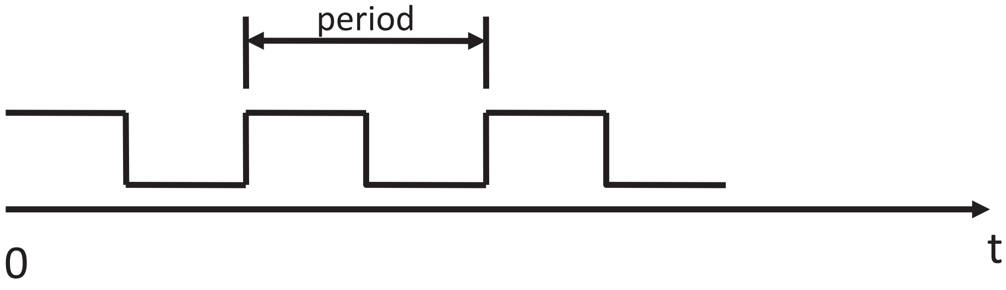

Fig.2. Time diagram of the *sc_clock* signal

Two models types comparison. These two models types obviously have essential differences. The basic difference is that in the event-oriented models the appropriate process receives control only then it is necessary for the developer. While in clock-oriented models the appropriate process receives control constantly by getting the specified edge of the clock signal even if there is no need, thus the process will be in an active waiting state and it naturally leads to additional resources costs, for example an increment of CPU load, a capacity of memory in use and model execution time. And a volume of additional costs depends on the model and active waiting state time.

Combined event-oriented and clock-oriented models usually use at complex systems, for example the model itself could be an event-oriented but testing data are generated throughout the whole modelling time with the constant time domain by a clock signal.

RMAP transport level protocol model is developed as an event-oriented model.

## *B. SpaceWire communication protocol*

SpaceWire is a spacecraft communication network based on an IEEE 1355 communication standard [1].

The purpose of SpaceWire is [9, 10]:

- to facilitate the construction of high performance on board data handling systems;
- to reduce system integration costs;
- to promote compatibility between data handling equipment and subsystems;
- to encourage reuse of data handling equipment across several different missions.

This Standard specifies the physical interconnection media and data communication protocols to enable the reliable sending of data at high-speed (between 2 Mb/s and 400 Mb/s) from one unit to another. SpaceWire links are full-duplex, point-to-point, serial data communication links.

This Standard covers the following protocol levels of OSI model: physical level, data link level and network level.

SpaceWire provides a means of sending packets of information from a source node to a specified destination node. SpaceWire does not specify the contents of the packets of information. SpaceWire provides a unified high speed data handling infrastructure for mutual connection of sensors, processing elements, mass memory units, downlink telemetry subsystems and electrical ground support equipment.

### *C. Transport layer protocol RMAP*

The remote memory access protocol (RMAP) has been designed to support a wide range of SpaceWire applications. Its primary purpose however is to configure a SpaceWire network, to control SpaceWire units and to gather data and status information from those units. RMAP may operate alongside other communications protocols running over SpaceWire [4].

RMAP may be used to configure SpaceWire routing switches, setting their operating parameters and routing table information. It may also be used to monitor the status of those routing switches. RMAP may be used to configure and read the status of nodes on the SpaceWire network. For example, the operating data rate of a node may be set to 100 Mbits/s and the interface may be set to auto-start mode [4].

For simple SpaceWire units without an embedded processor, RMAP may be used to set application configuration registers, to read status information and to read or write data into memory in the unit. For intelligent SpaceWire units RMAP can provide the basis for a wide range of communications services. Configuration, status gathering, and data transfer to and from memory or mailboxes can be supported.

RMAP is used to write to and read from memory, registers, FIFO memory, mailboxes, etc, in a destination node on a SpaceWire network. Input/output registers, control/status registers and FIFOs are assumed to be memory mapped, so are accessed as memory. Mailboxes are indirect memory areas that are referenced using a memory address. All read and write operations defined in the RMAP protocol are posted operations i.e. the source does not wait for an acknowledgement or reply to be received. This means that many reads and writes can be outstanding at any time. It also means that there is no timeout mechanism implemented in RMAP for missing acknowledgements or replies. If an acknowledgement or reply timeout mechanism is required it must be implemented in the source user application.

Write commands can be acknowledged or not acknowledged by the destination node when they have been received correctly. If the write is to be acknowledged and there is an error with the write request, the destination will send an error code to the source that sent the command. The error can only be sent to the source if the write command header was received intact, so that the destination that detected the error knows where to send the error message. If no acknowledgement is requested then the fact that an error occurred may be stored in a status register in the destination node. Write commands can perform the write operation after verifying that the data has been transferred to the destination without error, or it can write the data without verification. To perform verification on the data requires buffering in the destination node to store the data while it is being verified, before it is written. The amount of buffering is likely to be limited so verified writes ought only to be performed for relatively short sets of data, that will fit in the available buffer at the destination. Longer writes can be performed but without verification prior to writing. Verification in this case is done after the data has been written. Verified writes should always be used when writing to configuration or control registers. The acknowledged/non-acknowledged and verified/non-verified options to the write command result in four different write operations:

- Write non-acknowledged, non-verified;
- Write non-acknowledged, verified;
- Write acknowledged, non-verified;
- Write acknowledged, verified.

The read command reads one or more bytes of data from a specified area of memory in a destination node. The data read is returned in a reply packet.

The read-modify-write command reads a register (or memory) returning its value and then writes a new value, specified in the command, to the register. A mask can be included, in the command, so that only certain bits of the register are written. This provides an atomic operation that can be used for semaphores and other handshaking operations.

### *D. RMAP protocol SystemC model*

Model of Remote Memory Access Protocol (RMAP) designed and implemented with SystemC. RMAP model is an event-oriented as was noticed thereinbefore. The whole model structure with main blocks and data flows is shown at fig. 3. All interblock relations are based on port-interface interaction (one of SystemC basic modelling conceptions), a port depicts as square with two arrows inside and an interface depicts as circle with one arrow inside.

The protocol model implemented with accurate respect to specification and it was one of the main criteria during the implementation.

The architectural diagram at fig. 3 contents RMAP protocol model and application carries out all required interaction with model and actually testing of this model.

Protocol model divides onto four main blocks:

- *Up_data_wrapper* – the block provides functions for an application interaction with the protocol model;
- *Command_handler* – the block performs assembling of a command and reply headers, leads handshaking with an application and further data transmission. This block performs the main management role and protocol commands processing in case of any specified error detecting a decision about further actions is considered exactly by the *Command_handler* block. For example, CRC data error was detected at the incoming packet by the *Error_check* block, this error initiates a notification about the error to the *Command_handler* block and a request for the command execution won’t be able to transfer to an application. The error will be statistically counted; if necessary an answer will be assembled and transferred with the occurred error code notification. Request rejection situations are similarly processed;
- *Error_check* – the block is responsible for calculating, setting and checking of the CRC header and data fields. The CRC header field is always checked in contrast to the CRC data field which is checked depending on a received command. The data field could have a big size.  The CRC check demands buffering of the whole data which demands a big receiving buffer size. Therefore the *Error_check* block is also reliable for verification buffer overflow checking;

- *Low_data_wrapper* – the block provides interaction functions for a lower level model with the RMAP protocol model. It also performs information conversion to a symbols sequence for the consistent further transmission.

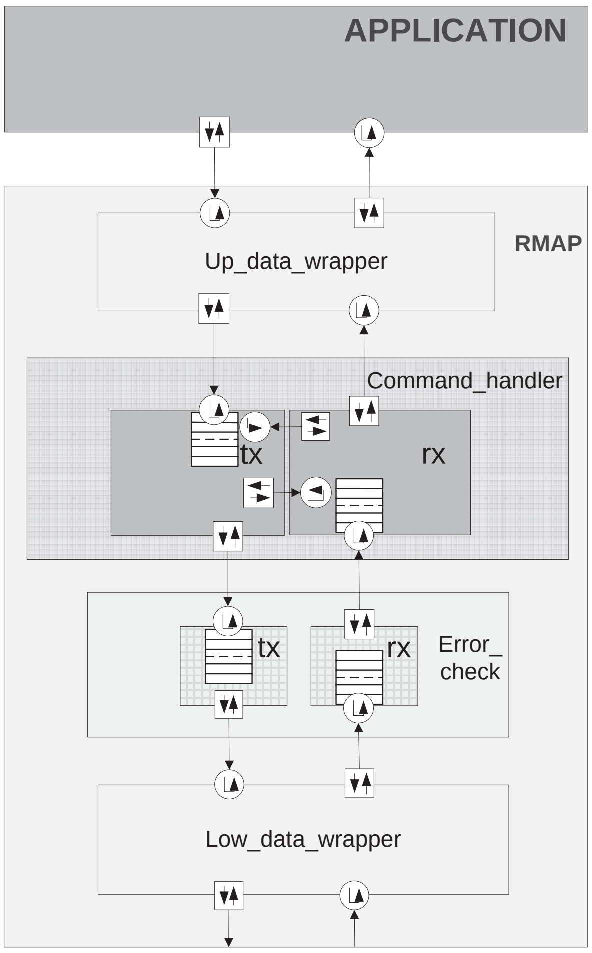

Fig. 3. RMAP model architectural diagram

Each of the *Command_handler* and *Error_check* blocks is consist of two sub blocks. First one is a receiver and second one is a transmitter. These sub blocks divide functionality of each block.

It should be noted that a term symbol is a nine bit word in current model, where eight lower bits are data and the ninth bit performs an end of packet definition. The ninth bit is always equal to null except of an end of packet symbol and an error end of packet.

The *Application* module provides necessary requirements for system performance: various test data generation, received data processing, dialoging with the RMAP model. Certainly the *Application* module obvious purposes are: performance and possibilities of the designing model check; initiating, tracking and processing of critical and error situations.

The RMAP model implementation works the following way. The request for a command transmission comes from an application to the RMAP. The request provides necessary parameters for RMAP command assembling. Because of the accurate parameters list is absented in the protocol specification a developer could prepare it on his own. In the present case the list of parameters is consist of symbols: the destination logical address; the packet type which includes command settings; destination node key; transaction identifier; write/read address; write/read data volume.

RMAP assembles a command header. The header is transmitted further to algorithm. The command header is transmitted as an array in single iteration within the RMAP model. After the CRC field is settled the command header is transmitted to the protocol model output. Then it is transmitted symbol-by-symbol to the channel.

Thereupon the RMAP model receives data for transmission from the application depending on an initialized command. A data transmission by small data cells is implemented. A data cell size may be various and it is settled by software. When data segment is buffered and CRC is set, data is transmitted symbol-by-symbol to the channel also.

The required addition is that the specification didn’t specify exact location for the end of packet symbol (EOP) addition. Therefore the end of packet symbol is created and transmitted with the data.

The RMAP command reaches a destination node after the channel. The command header is assembled by the node, the CRC field is checked. A few header fields are being transmitted to application as authentication request. If necessary the header is copied for the future response generation.

Data are received symbol by symbol also. Then they are assembled to data cells. They will buffer for the CRC field check, if verification is required. Thereupon data are transmitted further cell by cell to the buffer until the authentication completes.

In the successful authentication case data is transmitted to the destination node application and the transmission would be ended with the end-of-transmission signal. After destination node’s application received this signal, the reply is transmitted to the source node application depending on type of received command. The reply packet is transferred through the RMAP model, then enters the channel and comes to the source node application.

Therefore general algorithm of RMAP model work looks like. As mentioned thereinbefore the model is event-oriented and all interactions inside of it are based on events, each of them could initiate the next sequence of operations. The thread diagram of this model is shown at fig. 4.

Two RMAP protocol models with applications are depicted at fig.4 for the most convenient representation of thread sequence invokes. An application on the left is the initiator of a command transmission and an application on the right is receiver of commands, if necessary it assembles and transmits the reply on the command. The application initiator of a command transmission also receives respond from the command execution.

The description of illustrated threads follows (fig. 4):

1. The thread is responsible for command transmission to the channel (SpaceWire Network);
2. This thread is responsible for receiving, processing and analyzing of the command header and data from the channel (SpaceWire Network). After a header was received, it is assembled, the CRC is checked and the header is transmitted to the *Command_handler* block where it’s buffered until data won’t be received. Thereupon they are received.

If data have to be verified, they will be buffered symbol-by-symbol and the CRC will be checked. If there are no errors, thread 5 will be initialized, thereupon thread 3 and thread 5 launches. If there are occurred errors, thread 7 will launch.

If data were received without verification, thread 5 will be initialized and launched during the first data symbol is receiving. Thereupon data cells are completed by symbols and thread 4 is initialized.

3. The thread is responsible for data transmission from the *Error_check* block to the *Command_handler* block, when verification is necessary. Data are transmitted with the verification from the receiving buffer. They are buffered in the *Command_handler* block until the reply for an authentication request won't be received.
4. The thread is responsible for data transmission from the *Error_check* block to the *Command_handler* block when no verification is required. Data are transmitted cell-by-cell, when they reach the *Command_handler* block, they are buffered until an authentication completes.
5. There is a thread of header indispensible fields transmission to an application with the purpose of receiving an authentication reply. If necessary a reply header is assembled and an authentication reply is received (initialization of thread 6).
6. There is a thread of the data transmission to an application. If an authentication reply is successful, data will be transferred cell-by-cell into the application; thread 8 is initialized after transmission. If the authentication reply is negative, data will be discarded.
7. The thread is responsible for a reply packet assembling (if it's required). The reply packet is used for occurred errors notification. This thread initializes thread 9.
8. Into this thread reply data are generated and transferred to RMAP, it initializes thread 9 also.
9. This thread is used for reply header and data transferring into a channel (SpaceWire network).

Foregoing threads 2, 3, 4, 5, 6 descriptions are effective for RMAP commands processing. During replies process, simpler scheme of interaction works. It's depicted at fig.4 also.

## *E. Testing of the model*

Two duplicate instances connection by means of channel was chosen for the model testing. This is depicted at fig. 5. This approach includes launch, connection and joint operation of two models. Such approach is used for a model functionality checking and probable ways of improvement searching.

There are some characteristic features of this approach. The first is an absence of difficult testing signals created by a developer. Because testing signals with the different structure are generated by an application like requests for commands transmission. Results of command execution are reflected at the application of the destination node. There is a possibility to trace the whole data exchange process in both directions, if error is detected at the destination node [10, 11].

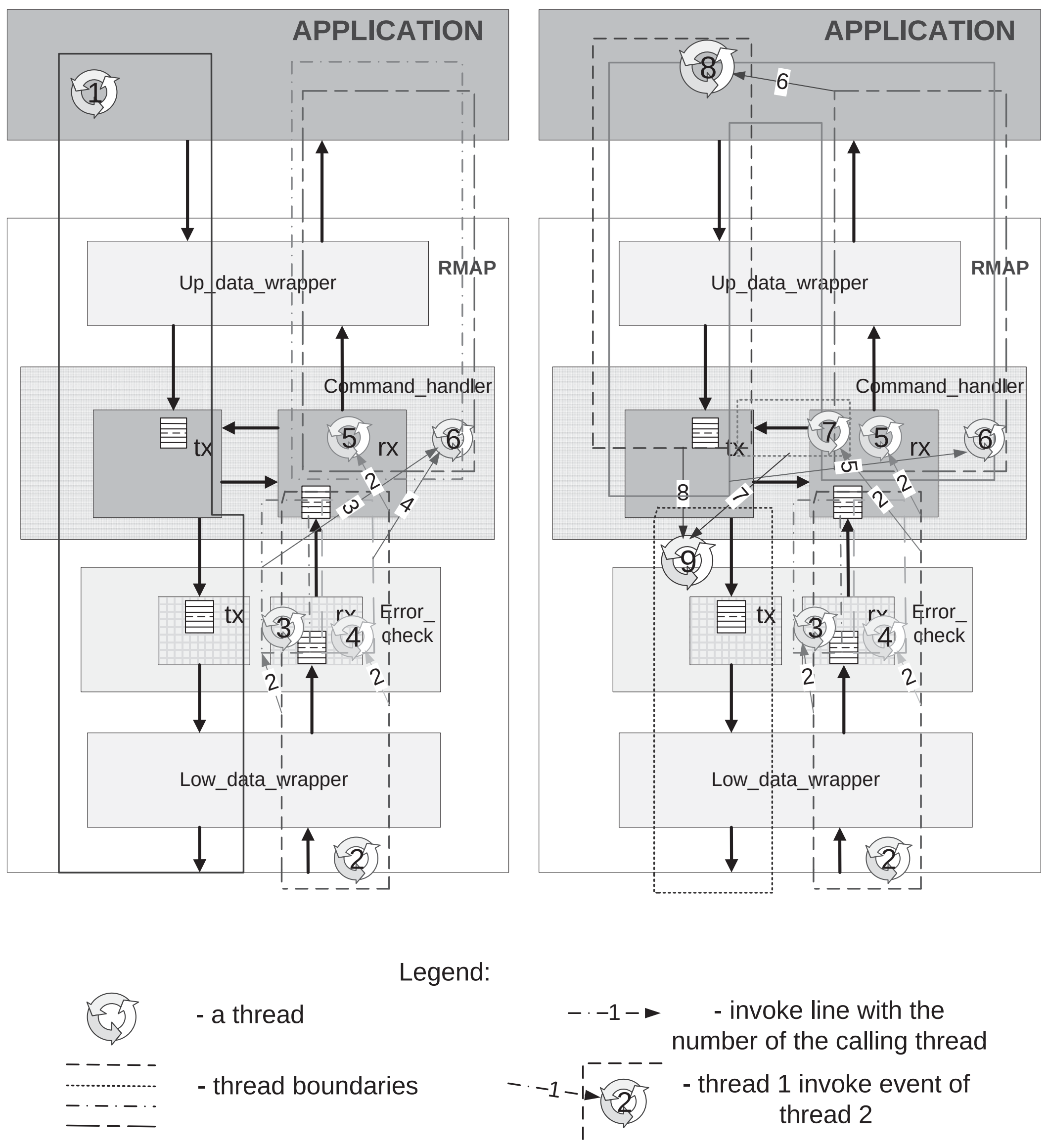

Fig. 4. The data thread diagram of the RMAP model

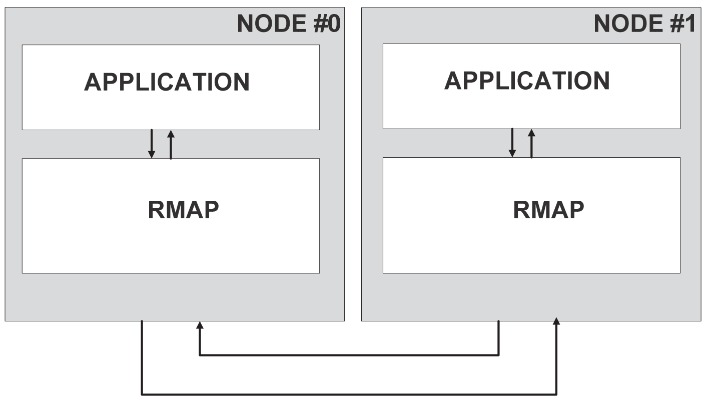

Fig. 5. The scheme of point-to-point model testing

Generated test signals at the destination node stimulate assembling, processing and symbol-by-symbol transmission into a channel. The destination node receives symbols from channel, assembles, process and executes command.

All required changes are made during the testing and debugging of the model [12]. As a result the model satisfies with all requirements of specification and all inconsistencies are improved [13].

### *F. Interaction and testing with the SpaceWire model*

Therefore one of the primary purposes of RMAP development was joint operation over SpaceWire. There are a SpaceWire network configuration, SpaceWire units control, data and status information gathering from those units and a wide range of SpaceWire applications support. One of purposes was the interaction testing of RMAP protocol and SpaceWire protocol stack by means of their models.

The scheme of models connection is depicted at fig. 6. There are two RMAP models with applications and a SpaceWire model below each of them. SpaceWire models are connected by the channel. The SpaceWire model was earlier designed and implemented by SystemC also.

The result of models interconnection is protocols interaction proof. Synchronization was implemented by SpaceWire mechanisms. It could arise some collisions. The network level of SpaceWire model is responsible for command header “Destination Logical Address” field checking which is filled at a transport level outside of SpaceWire. Error situations were tested by theirs artificial generation.

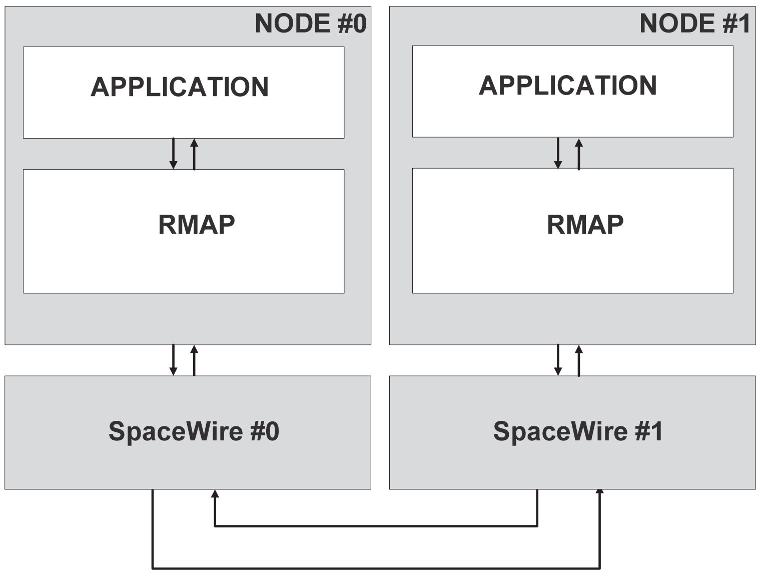

Fig. 6. The RMAP and SpaceWire models interaction diagram

### *H. The RMAP protocol and such kind of systems modelling and testing features.*

The protocol modelling and model testing was based on an iterative approach. The model functionality was increased step-by-step during designing [14]. Firstly the process of one single command generation, transferring and receiving was debugged and tested.

The other commands were implemented after. Thereupon the process of a reply packet generation, transferring and receiving was debugged and tested. Then transferring and processing of several different commands were done. Various commands combinations were tested. The last but not least was the stage of implementation of an error situation handling. This stage was the most difficult. It caused a lot of algorithm changes. The algorithm complexity was highly increased.

Some over difficulties were occurred during the RMAP protocol model elaboration. The symbol-by-symbol transmission into a channel (SpaceWire network) complicated data receiving. A header assembling demands concrete fields analyses. Small segment data transmission complicated model also.  Therefore a command receiving or transmitting algorithm consists of huge number of iterations. Different iterations depending on type packet includes different algorithm branches. This point makes debugging process much more difficult.

The feature of SpaceWire protocol stack is that all header fields are filled by transport layer. Incidentally the RMAP protocol is responsible for header fields filling. It means that RMAP protocol includes necessary information about contents of header fields. Various error situations processing is defined by RMAP protocol either. Therefore a set of model tests becomes more and more complicated.

The important aspect is that RMAP directly interacts with an application. This aspect highly expends a set of functions provided with the RMAP interface for interaction with an upper level (application). In comparison with any middle layers which only provide ability to transfer and receive data. But mainly it depends on the protocol complexity.

Ambiguous descriptions of some protocol specification points provide deadlock situations. A deadlock situation solution and amendments could be accepted by workgroup. A developer should choose the most appropriate way of implementation, if there is an absence of any situations descriptions. It requires some additional time costs.

## III. Conclusion

As an important result of the RMAP protocol and SpaceWire protocol stack co-modelling the RMAP protocol model which satisfies the specification requirements was designed. This model could be used as the separate module, as well as a part of a stack which might consist of some different protocols models. Then the interaction and compliance of protocols can be tested. It supports the pros and cons list compilation. This list could help to make correct decision about benefits of combination.

The designed model can be used for the RMAP protocol hardware testing for specification compliance. Hardware could be incorrectly implemented also.

During the model designing weak spots, inconsistencies were found in the protocol specification. The revision list with corrections, additions and recommendations to the specification was compiled.  This makes easier and more effective the process of protocol development on specification stage. It provides important information for significant decisions which should be accepted.

According to the received modelling experience the affirmation about iteration approach benefits is possible. The whole functionality could be hardly predicted, but it’s easier to fill blocks with the functionality during elaboration.

There is an imperfection. Some unforeseen subtleties in the beginning of the design could provide essential algorithm correction. When block functionality is huge enough, the alteration of the algorithm demands big efforts and time wastes. During the RMAP model designing this kind of situation was occurred upon specified errors processing. It confirms the importance of the model structure development before an implementation stage.

## REFERENCES

[1] V. Olenev, Y. Sheynin, E. Suvorova, S. Balandin, M. Gillet, “SystemC Modelling of the Embedded Networks”, *Proceedings of 6$^{th}$ Seminar of Finnish-Russian University Cooperation in Telecommunications (FRUCT) Program (pp. 85-95).* Saint-Petersburg University of Aerospace Instrumentation (SUAI), Saint-Petersburg, 2009

[2] V. Olenev, I. Korobkov, N. Martynov, A. Shadursky “RMAP and STP protocols modelling over the SpaceWire SystemC model”, Proceedings of 8th Seminar of Finnish-Russian University Cooperation in Telecommunications (FRUCT) Program (pp. 111 – 121), Lappeenranta University of Technology (LUT), Lappeenranta, 2010

[3] Y. Sheynin, T. Solokhina, Y. Petrichkovitch “SpaceWire technology for the parallel systems and onboard distributed systems”, ELVEES, 2006,

[4] ESA, specification RMAP, “SpaceWire Remote Memory Access Protocol”, University of Dundee, Applied Computing, Dundee, DD1 4HN, Scotland, UK, 2006.

[5] I. Shugarin, V. Kanishev “Application of the SystemC language and its development tools for SoC development”// Chip news. 2006. № 9. C. 51-56.

[6] “SystemC User's Guide. Open SystemC Initiative (OSCI). Version2.0”, 1996-2002.

[7] D. C. Black, J. Donovan, B. Bunton, A. Keist, “SystemC From the Ground Up”, 2004

[8] N. Martynov “The comparative analysis of the event-oriented and clock-oriented SystemC models, *Proceedings of 62rd Saint-Petersburg University of Aerospace Instrumentation student scientific and technical conference*, Saint-Petersburg University of Aerospace Instrumentation (SUAI), Saint-Petersburg, 2009.

[9] ESA (European Space Agency), standard ECSS-E-50-12A, “Space engineering. SpaceWire – Links, nodes, routers and networks. European cooperation for space standardization”, ESA Publications Division ESTEC, Noordwijk, The Netherlands, 2003.

[10] V. Olenev, I. Korobkov, N. Martynov, A. Shadursky “Modelling of the SpaceWire communication Protocol”, *Proceedings of 7th Seminar of Finnish-Russian University Cooperation in Telecommunications (FRUCT) Program (p. 96),* Saint-Petersburg University of Aerospace Instrumentation (SUAI), Saint-Petersburg, 2010

[11] V. Olenev, I. Korobkov, N. Martynov, A. Shadursky “Modelling of the RMAP and STP transport layer protocols as a part of the SpaceWire protocol stack model”, *Proceedings of 63rd Saint-Petersburg University of Aerospace Instrumentation student scientific and technical conference*, Saint-Petersburg University of Aerospace Instrumentation (SUAI), Saint-Petersburg, 2010.

[12] A. Jantsch, “B. SpaceWire communication protocol Embedded Systems and SoCs”, *Morgan Kaufmann Publishers, Stockholm,* 2004

[13] V. Olenev, “Different approaches for the stacks of protocols SystemC modelling analysis”, *Proceedings of the Saint-Petersburg University of Aerospace Instrumentation scientific conference (pp. 112-113),* Saint-Petersburg University of Aerospace Instrumentation (SUAI), Saint-Petersburg, 2009.

[14] Eric J. Braude, “Software Engineering. An Object-Oriented Perspective", 2004