RMAP and STP protocols modelling over the SpaceWire SystemC model

Valentin Olenev, Ilya Korobkov, Nikita Martynov, Arkady Shadursky
St-Petersburg University of Aerospace Instrumentation
190000, St-Petersburg, Bolshaya Morskaya 67, Russia
Valentin.Olenev@guap.ru, Ilya.Korobkov@guap.ru,
Nikita.Martynov@guap.ru, Arkady.Shadursky@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 form 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.

SpaceWire is a standard that is de facto the main data streaming standard for onboard systems developed by ESA, NASA and JAXA. This paper gives an overview of RMAP and STP transport protocols modelling over the SpaceWire SystemC model activity.

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

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 [8].

In this paper we consider the multiple applications communication and work via the SpaceWire network. The SpaceWire model, transport protocols RMAP and STP models was implemented for this purpose. Also for this purpose we implemented the RMAP and STP SystemC models' operation on top of the SpaceWire protocol stack [7].

This article gives an overview of our activities and shares the important modelling results.

II. Project overview

SpaceWire standard covers three (physical, data-link and network) of the seven layers of the OSI model and does not specify the transport layer [2]. Therefore to implement the simultaneous operation of multiple applications, pursuing different goals over the SpaceWire it is necessary to connect the transport layer with the previously developed SpaceWire model (see fig.1).

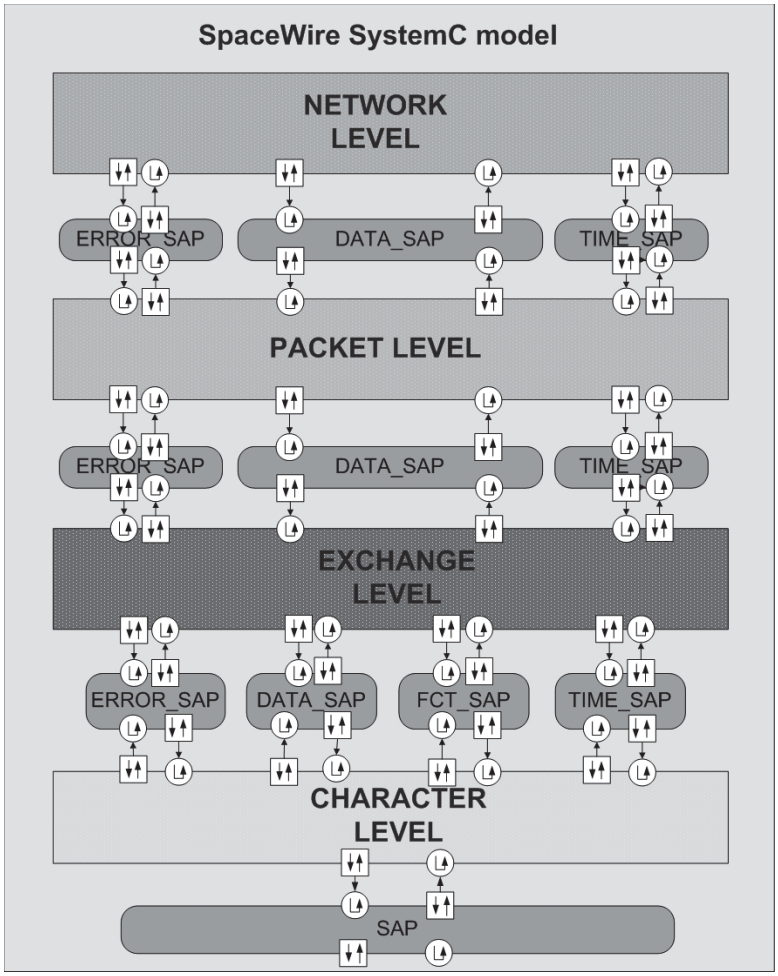

Figure 1. SpaceWire SystemC model [5]

Applications should have possibility to work with remote memory, registers, and support large data level transmission. There are a lot of different transport protocols. We used protocols RMAP and STP, because RMAP is designed to work with remote memory (registers, FIFO, etc.) [3], and STP (it is connection-oriented protocol) provides reliable transmission of large data level [1].

These models is implemented for testing ability of the transport level protocols work with SpaceWire protocol stack. In addition STP and RMAP models should operate over the SpaceWire simultaneously. Protocol Identifier is used for this purpose. We implemented the model of this protocol also.

Whereas several applications should have possibility to transfer data simultaneously by means of STP and RMAP we developed management units for each transport protocol models. They are *RMAP_manager* and *STP_manager*. It allows to route data stream between applications and proper transport protocol model. Also for testing joint models work we developed model of Application level. It consists of two modules: *App_STP* and *App_RMAP*. They perform an application role and they can simultaneously work with STP and RMAP transport protocols by means of *STP_manager* and *RMAP_manager*.

The total structure of our model is shown at the fig. 2.

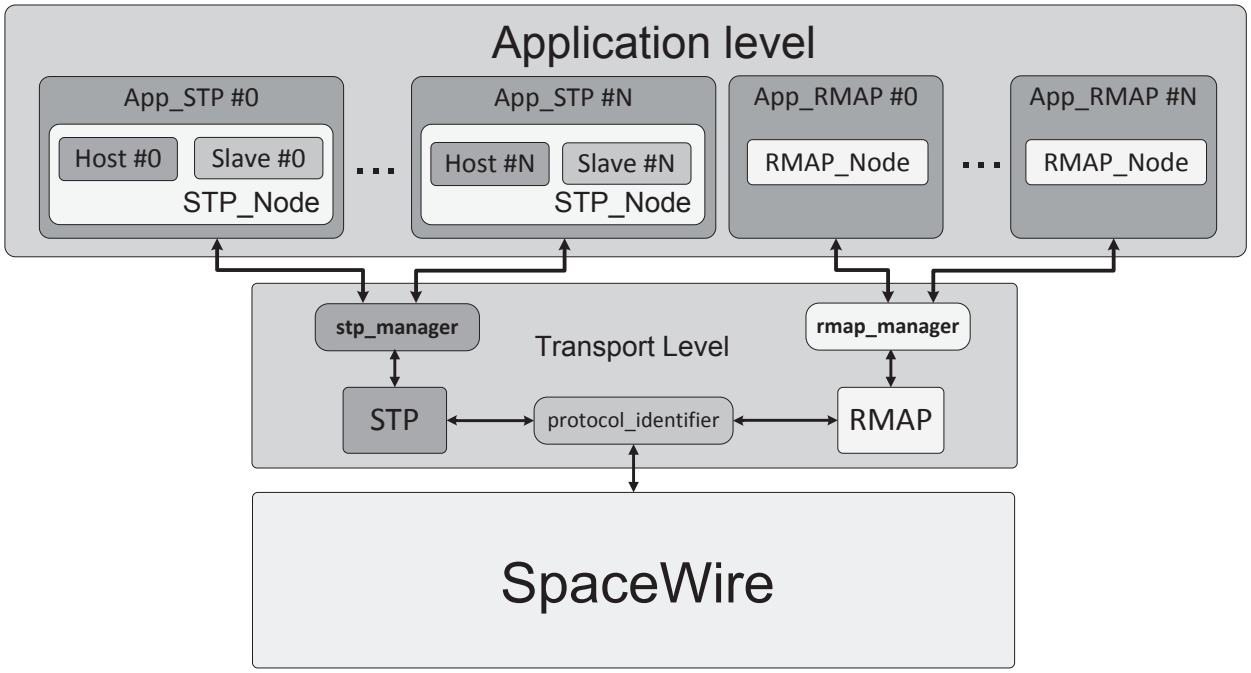

Figure 2. Total SystemC model architectural diagram

III. TESTING METHODS

A huge test bench needed to test protocol models. It is used for traffic generation and its analysis. The easier implementable and the most convenient method is a point-to-point connection of two models instances. Doing so each model can generate the traffic, but the analysis of the outgoing data mostly would be done by the receiving sides of the both models according to the mechanisms defined in the specification. Such kind of testing method is shown on fig. 3:

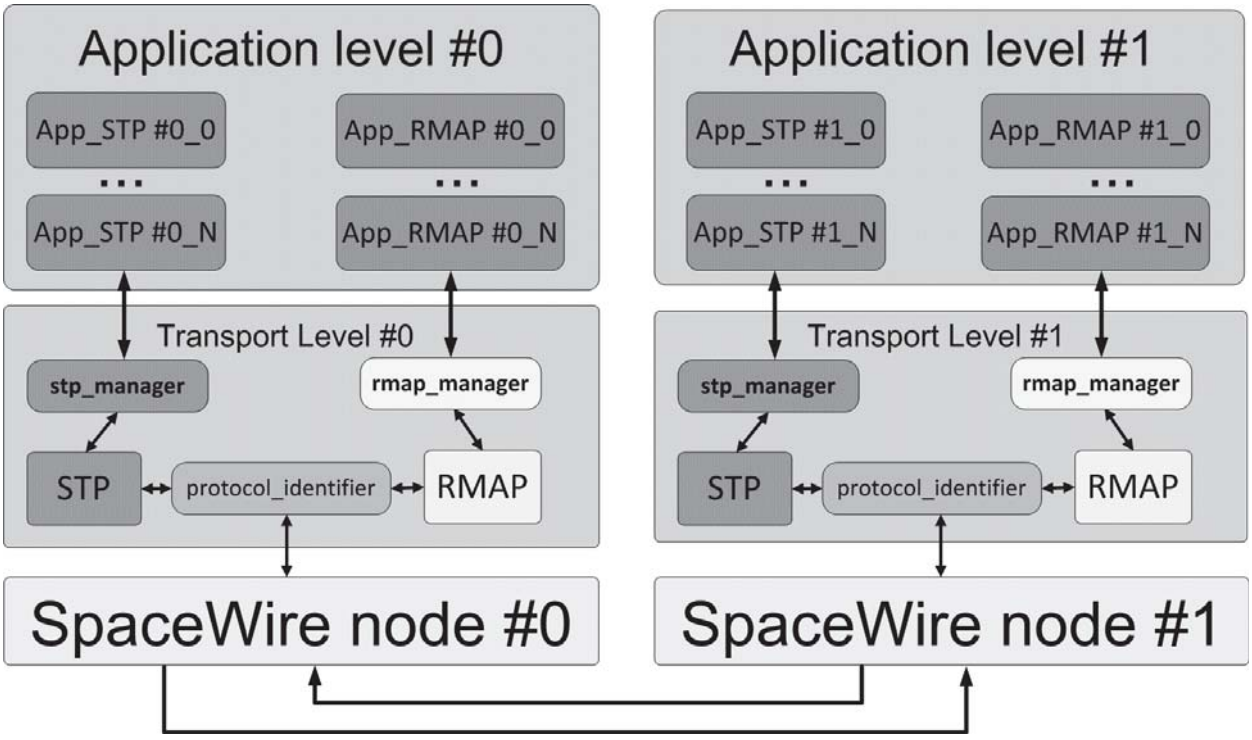

Figure 3. An example of two models point-to-point connection.

So, such modelling method gives a number of benefits:

- The model could test all the internal mechanisms;
- There is a possibility to trace the whole data exchange process in both directions;
- The traffic analysis is done by the model itself according to the specification;
- There is a possibility to check the joint work of protocols.

Traffic-generators are placed in every Application (*App_STPs* and *App_RMAPs*) for data generation. It is flexible module, used for generating various types of data and sending it to device for processing.

Generated data step by step passes all the layers of model. By comparing of data sent by *Node #0* and the data received by the *Node #1* the model's operation correctness could be estimated. If there is any corrupted data received then error is detected. If received data is correct it means that the test is passed and we can start the next one with another portion of data and other settings.

One of the most important advantages of this method is that point-to-point connection mode can simulate work of the model very close to the real devices communication. Applications create various situations of data exchange. Necessary corrections are applied for every unique case of modelling. Though such modelling case we could find errors in the model or in the specification itself and the developer could fix it.

Let us consider each created model in details.

IV. MODELS DESCRIPTION

A. RMAP SystemC Model

Remote Memory Access Protocol (RMAP) is a transport layer protocol. This protocol is connectionless. By using it you have possibility to read or write data to remote device's memory which could have an embedded processor or not [3]. This is a distinctive feature of current protocol.

The protocol's model implemented with accurate respect to specification and it was one of the main criteria during the implementation. RMAP's architectural diagram is shown at the fig. 4.

The architectural diagram contents two main blocks. They are Command handler and Checker. Each of these blocks is divided into two parts: transmitter (tx) and receiver (rx).

Command handler module carry out following functions:

- command header assembling;
- reply header assembling;
- handshake between RMAP and application.

Checker module is responsible for CRC calculation and checking. In case of CRC error the specified operations will be performed. For example, if error occurred in command's data segment (which needs acknowledgment), the transmission of invalid data to application will be prevented and will initialize assembling and reply packet transmission to the source node with appropriate status. Also destination node will be informed about data corruption.

The RMAP implementation works the following way. When command transmission request comes to RMAP it contains all necessary parameters for command assembling. RMAP assembles the command header and transfers it further. RMAP header is transferred between

modules as one massive. When the CRC is set, the header is transmitted symbol-by-symbol to the SpaceWire Network.

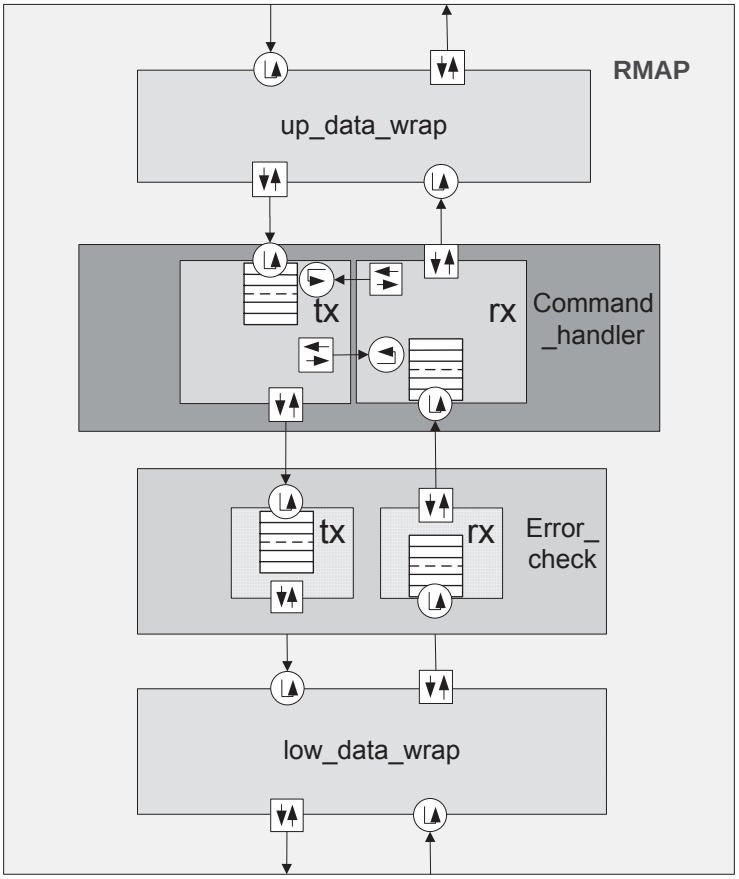

Figure 4. RMAP model architectural diagram

Then data for transmission comes to RMAP from an application depending on the type of initialized command. When data segment is buffered and CRC is set, then it is transmitted to the SpaceWire Network symbol-by-symbol also.

When the RMAP command passed through a SpaceWire Network reaches the destination node, the node assembles the command's header. Then it checks the CRC and makes the response. A few header fields are being transmitted to application as authentication request.

The data is transmitted symbol by symbol also. Then it is assembled to data cells and buffered for CRC check if verification is required. Then data goes further cell by cell to the buffer until the authentication completes.

In the successful authentication case data is transmitted to the application and it responses 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 goes through RMAP model, then enter the SpaceWire Network and comes to the source node application.

So its a general algorithm of RMAP model's work. The model is event-oriented and all interactions inside it are based on events, each of them could initiate the next sequence of operations. In terms of SystemC language event could initiate thread execution. And the thread diagram of this model is shown at the fig. 5.

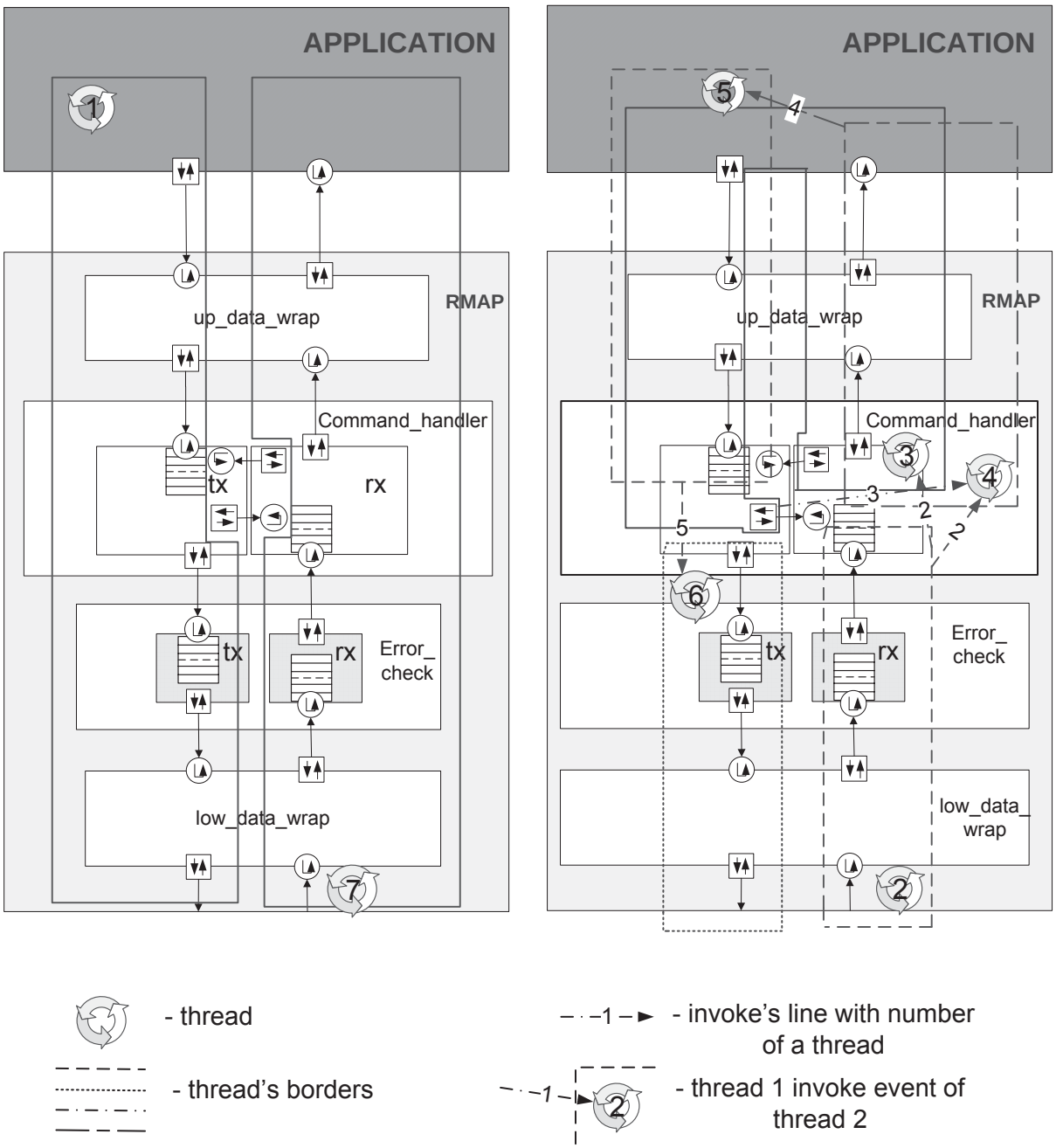

Figure 5. RMAP model's thread diagram

The description of illustrated threads follows:

- Thread 1 is used for command transmission to the SpaceWire Network;
- Thread 2 is used for data receive (including header) from the SpaceWire Network.
- Thread 3 is used for header transfer to application (authentication process), for reply header assembling and for authentication indication;
- Thread 4 is used for data transfer (after authentication). After transmission is over thread 5 is initialized;
- Thread 5 is used for reply data generation and transmission;
- Thread 7 is used for receiving of data (including the header) from SpaceWire Network and transmission of it to the application.

B. STP SystemC Model

Streaming Transport Protocol (STP) is a new connection-oriented transport layer protocol. STP is developed for streaming data transmission from a local memory of the host device over SpaceWire link [1]. But this protocol is still in the process of development, so the model's task is to help making important design decisions and to verify already specified mechanisms.

STP supports coherent data sources that can synchronous transmit data flows. It is oriented for asymmetric transport connection use: there is a host (master), and the slave device on the other hand. The host device is an initiator of a transaction session. It performs set-connection control, initialization of connection parameters and data flow control [1].

We developed the STP model in SystemC modelling language according to the STP specification. The structure of STP model is shown at the fig. 6.

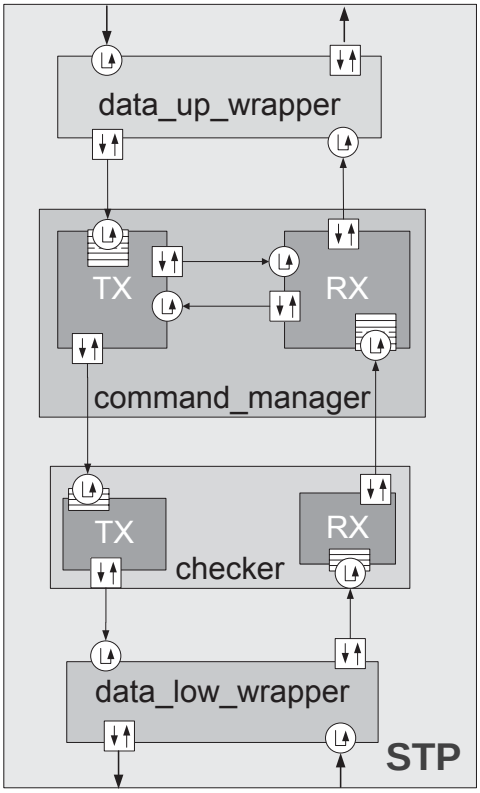

Figure 6. STP model architectural diagram

STP model consists of four main blocks: *data_up_wrapper*, *command_manager*, *checker*, *data_low_wrapper*. *Data_up_wrapper* performs communication functions between STP model and application level. *Command_manager* is responsible for generating header and body of STP command. This block is divided into two parts: transmit (TX) and receive (RX). *Checker* is used to calculate and check CRC field in STP commands. In the CRC error case *Checker* should send corresponding notification to application level. This module is divided into transmit and receive parts also. This model is event-oriented too and all blocks interact with each other via ports and interfaces [7]. STP model can work simultaneous at host and slave mode. Let's consider how it works.

STP model gets the open connection request from an application (host). Open and close connection mechanism between host and slave is shown at the fig. 7.

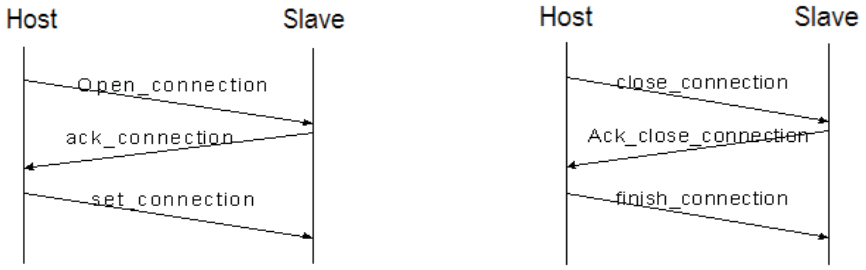

Figure 7. Message sequence chart for open and close STP connection

Host generates an *open_connection* command depending on the received request. Then it calculates the CRC and sets it to command header and body. Then host sends this command to the SpaceWire Network symbol-by-symbol. When the command reaches the slave node, slave node assembles the command and then checks the CRC fields. If CRC field is invalid, the command would be ignored. Otherwise, command will be analyzed in details. If it is necessary, slave node will save all useful information about connection parameters from this command. After that it generates corresponding response command – *ack_connection*. And this command goes to host node via the SpaceWire Network. Host receives the response, analyzes it and generates another response command *set_connection*. If slave node receives correct response, connection is successfully set. Such mechanism allows decreasing of the reliability of connection establishment between applications.

After that slave node notifies the application about successful connection establishment. Application sends request with data for transmission to remote application (host). Slave generates data commands and then sends them with specified frequency. In addition slave will send data cargo according to the number of credits, which have been specified during the set connection transaction. If host buffers are full, host can notify slave to stop the data transfer. And later host can inform slave to start transfer data again. So STP protocol allows controlling data flow.

C. Protocols' models interaction

The next step of our work was to implement interaction between RMAP, STP and SpaceWire models. And RMAP and STP models should operate over the SpaceWire simultaneously. This task was implemented by using Protocol Identifier. It performs such tasks as arbitration and addressing of data between different protocols. So in addition we implemented Protocol Identifier SystemC model.

One of the modelling tasks was the implementation of data transfer between a set of applications by means of STP and RMAP protocols. We developed two management modules for the STP and RMAP communication with applications. They are RMAP manager and STP manager.

D. Protocol Identifier

The structure of Protocol Identifier model is shown at the fig. 8.

The Protocol Identifier model has been implemented according to the specification [4]. All specified functionality is implemented and the model operates on top of SpaceWire model.

The main purpose of the Protocol Identifier model is the routing of STP and RMAP commands from the SpaceWire network to one of the protocol's models. Routing is carried out by a special key – PID.

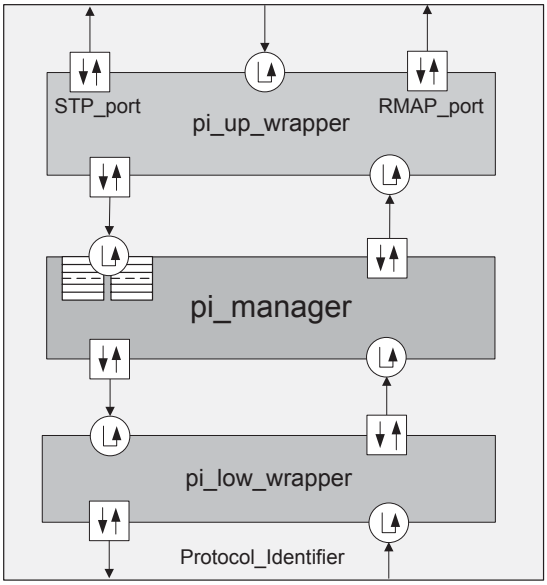

Figure 8. Protocol Identifier SystemC model architectural diagram

PID key is uniquely defined in the specification of Protocol Identifier. RMAP have its own PID – 1. For the STP protocol PID is not specified, so the PID equal to 240 have been chosen. According to the Protocol Identifier specification 240-254 identifiers shall be assigned by the project and can be used. [4]

There is one requirement for packets passed through Protocol Identifier: the second byte of each packet should be PID key. Only this way Protocol Identifier can work correctly.

Also model provides arbitration of commands coming from RMAP and STP models and sends them to the SpaceWire network. Two buffers (*buffer_rmap* and *buffer_stp*) was created to store two first bytes of the received packet. The structure of STP and RMAP manager is shown at the fig. 9.

E. RMAP_manager and STP_manager

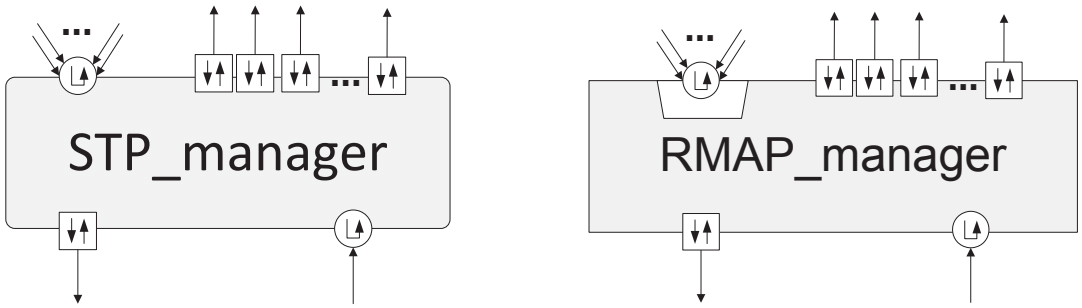

Figure 9. STP and RMAP manager SystemC model architectural diagram

Since our task was the realization of several applications for each created model of the transport protocol, we developed the *RMAP_manager* and *STP_manager*. These units route the data stream from each transport level model to the set of applications. These applications are connected to managers via a number of ports.

Any received primitive should be analyzed. The manager should find the destination address of the particular application and store it in memory. Then a list of primitives should be sent to the application via appropriate port. Since each of the transport protocol has its own unique sequence of primitives, the concrete manager has to control flow of primitives to detect the end of current packet. Several simultaneously operating applications can send the data to the appropriate manager, that in turn sends them to the RMAP or STP model.

V. Application level simulaton

During the complex models step-by-step development it is necessary to have a testing system. It should help to find bugs in implementation [6]. Besides it should do all necessary requirements for model maintenance – incoming data processing, logging and so on. And also the most important purpose of such a system is a conformance testing of the developed model

Thereby we developed some testing units. They represent application layer model and have the following functionality:

- generation of different kinds of traffic;
- received data processing;
- response traffic generation;
- error and critical situations handling.

The Application level has two modules *App_RMAP* and *App_STP*. They perform an application role and they can simultaneously work with STP and RMAP transport protocols by means of *STP_manager* and *RMAP_manager*. In one's turn these transport protocols have possibility at the same time interact with SpaceWire model via Protocol Identifier.

VI. Conclusion

The result of our work is the complex system model. It satisfies to all specification requirements. It consists of all above mentioned models and units: SpaceWire, Protocol Identifier, RMAP, *RMAP_manager*, STP, *STP_manager*, Application modules (*App_RMAP*, *App_STP*).

There were problems with some specification points during the implementation. We found some bugs in specifications. For example, there is a field for node address in STP and RMAP packages, but there is no field for the concrete application address. STP protocol is still in the process of development, so it has not yet described methods of error processing and description of difficult situations that may occur during his work. However at the model testing stage we have faced with such situations and it was necessary to handle them. Now we have analysed our results and have brought the contribution to STP protocol development.

So we have done the executable system model, which allows simultaneously data transfer between a set of applications via SpaceWire network. It was done by means of developed models of STP and RMAP protocols and interaction modules. The given model allowed us to combine several protocols together and make their joint work research.

REFERENCES

[1] SUAI, STP specification, "Streaming Transport Protocol", *presented at International SpaceWire Conference 2010*, 2010

[2] 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

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

[4] ESA, standard ECSS-E-ST-50-51C, "Space engineering. SpaceWire protocol identification", Publications Division ESTEC, Noordwijk, The Netherlands, 2010

[5] 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

[6] 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

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

[8] Y. Sheynin, T. Solokhina, Y. Petrichkovitch "SpaceWire technology for the parallel systems and onboard distributed systems", ELVEES, 2006, *http://multicore.ru/fileadmin/user_upload/mc/publish/SpW-part1.pdf.*