DELTA

“Development of SpaceWire and CAN SystemC Transaction Level Models for ESA IP cores”

SpaceWire TLM User Manual

FINAL

PUBLIC

Date: 25 November, 2010

Prepared by:

Qualtek Sprl.
36 Avenue Gabriel Emile Lebon, B-1160, Brussels, Belgium

For:

ESA-ESTEC
Keplerlaan 1, Postbus 299, 2200 AG Noordwijk, The Netherlands

Page intentionally blank

TITLE	**SpaceWire TLM User Manual** **DELTA** **"Development of SpaceWire and CAN SystemC Transaction Level Models for ESA IP cores"**
COPYRIGHT	Copyright © 2010 Qualtek Sprl., Brussels, Belgium

CONFIDENTIALITY		**DISTRIBUTION**
Public	**X**	ESA-ESTEC Qualtek Sprl.
Government Confidential		
Military Confidential		

REVISION HISTORY

Version	**Date**	**Author**	**Status**
1.0	2 June, 2010	**Nikos Mouratidis**	PRE-FINAL
1.1	15 June, 2010	**Nikos Mouratidis**	PRE-FINAL
1.2	21 June, 2010	**Dimitris Tsaimos**	PRE-FINAL
1.3	30 June, 2010	**Nikos Mouratidis**	PRE-FINAL
1.4	28 September, 2010	**Dimitris Tsaimos**	FINAL
1.5	25 November, 2010	**Dimitris Tsaimos**	FINAL

List of Changes

Index	Change Description	Revision	Date
1	Minor editing/corrections	1.1	15-06-10
2	Addition of Section 5, minor additions to other sections	1.2	21-06-10
3	References to intermediate code version removed	1.3	30-06-10
4	Description adapted to the changes in the implementation	1.4	28-09-10
5	Modifications suggested at the Final Review	1.5	25-11-10

DOCUMENT APPROVED BY:

PROJECT MANAGER	QUALITY MANAGER	MANAGING DIRECTOR	CUSTOMER'S REPRESENTATIVE
Date:	Date:	Date:	Date:
Name: Dr. N. Mouratidis	Name:	Name:	Name:
Signature:	Signature:	Signature:	Signature:

Table Of Contents

List of Tables

List of Figures

1 SCOPE

1.1 IDENTIFICATION

This document presents the usage of the SpaceWire TLM 2.0 core delivered by DELTA. It aims at providing the details necessary for a core user to utilise the model within a system-wide model.

1.2 OVERVIEW

The DELTA project has delivered a TLM 2.0 model of the SpaceWire IP that was already in existence in the form of a RTL core written in VHDL. The objective of the current project has been to provide leverage for the attainment of simulation models that execute faster and natively (i.e. without the need for a separate simulator application) on a host platform. The present document describes how the establishment of a system-wide model making use of the SpaceWire core is possible, using the developed core.

Although the core was, as stated before, targeting the instantiation of a standalone simulation environment, where the simulation kernel constitutes part of the model itself, it was agreed as part of the project related validation, that the TLM core would be assessed against the existing RTL IP. As such, the necessary infrastructure was put in place to allow the mixed simulation of a TLM (SystemC) core and a VHDL testbench. Since this infrastructure was created, utilisation of the developed TLM within it is also discussed in this document.

1.3 DEFINITIONS

Shall

Identifies the mandatory requirements on the item or items. Statements which include the ***shall*** statement are to be considered as the only requirements to be tested or otherwise validated.

Should

Suggests an approach that is to be assumed as the approach to be taken and is a reflection of the current status at the time of document issue. Terms such as '***may***' or '***can***' also fall into this advisory but not mandatory category.

Will

Indicates factors that are imposed on the scope of this specification from outside and is to be regarded as a definition of factors that are mandatory by implication.

1.4 DOCUMENT OVERVIEW

This document is identified as follows:

Document type	-	User Manual
Document identifier	-	0310-01-004-01
Revision	-	1.5
Issue Date	-	25 November, 2010

2 APPLICABLE DOCUMENTS

The following documents of the exact issue shown form a part of this specification to the extent specified herein.

RFQ/3-12785/09/NL/JK/al	The Request For Quotation provides details on the expected form of the delivered model in terms of use cases, as well as compatibility with or dependency on simulation tools. **Revision. -, November 10, 2009**
OSCI TLM-2.0 LANGUAGE REFERENCE MANUAL	The LRM defines the classification, nomenclature, and details of the applicable coding style with reference to the model use cases. **Revision. JA32, July, 2009**
SpaceWire CODEC IP - User Manual	The User Manual of the RTL IP core provides grounds for the utilisation of the TLM together with the developed transactor in simulations, replacing the RTL IP **Revision. 2.4, March 27, 2009**

2.1 ORDER OF PRECEDENCE

In the event of a conflict between the text of this specification and the references cited herein, **except references to higher-level program-unique specifications for this program**, the text of this specification takes precedence. Nothing in this specification however, supersedes applicable laws and regulations, unless a specific exemption has been obtained.

3 DEFINITIONS

3.1 TERMS AND DEFINITIONS

Bit rate	Number of bits transmitted/received every second
byte	Eight bits of information
DDR	Double data rate. Two bits of data transmitted for each transmit clock period.
FCT	Flow control token. Transmitter sends one FCT when room in receive buffer for eight more N-chars.
N-char	Data character, EOP or EEP.
Null	Control code transmitted by SpaceWire link to keep connection with other end.
SDR	Single data rate. One bit of data transmitted for each transmit clock period.

3.2 C++ CODE TERMS AND DEFINITIONS

The term ".h file" is used in this document to refer to the public interface file because of the long-standing Unix/C practice of using a ".h" extension to the file name, although other operating systems and compilers may require a different extension. The term ".cpp file" is used in this document to refer to an implementation file for similar historical reasons.

The interface file provides a single point of declaration of the data entities and functions provided by a particular module. The interface is #included in all modules which refer to or make use of the data entities and functions provided. Sections of the interface file are not copied into other modules.

The contents of the interface file are surrounded by #ifdef/#endif pre-processor directives in order to avoid problems of multiple inclusion.

4 CONFIGURATION MANAGEMENT

4.1 DIRECTORY STRUCTURE

The directory structure of the SpaceWire CODEC TLM is shown in the following diagram:

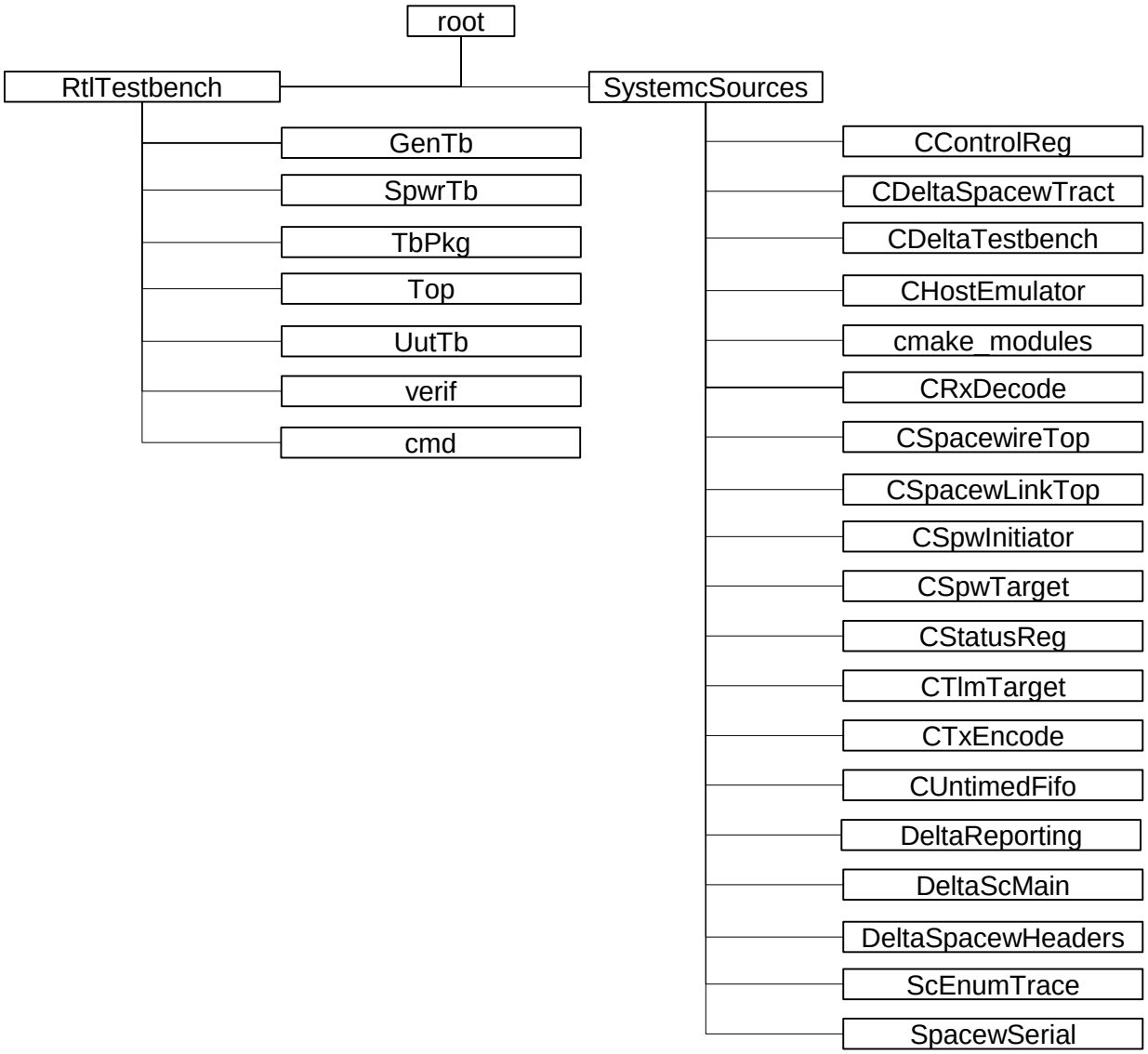

Figure 1: SpaceWire CODEC TLM directory structure

The SpaceWire CODEC sources, as well as the transactor sources are located under the SystemcSources directory, whereas the VHDL sources of the RTL test-bench along with the command files and scripts necessary to build the mixed SystemC-VHDL design and run the simulations are located under the RtlTestbench directory.

A description of the SpaceWire CODEC SystemC sources directory is presented in Table 1:

Directory	Description of contents
CControlReg	SpaceWire CODEC SystemC model control register
CDeltaSpacewTract	The SpaceWire CODEC transactor class
CDeltaTestbench	The class implementing the SpaceWire CODEC SystemC-only test-bench
CHostEmulator	Traffic generator for the SystemC-only test-bench
CRxDecode	Receiver and decoding logic

Directory	Description of contents
CSpacewireTop	Top level module of SpaceWire CODEC TLM
CSpacewLinkTop	Top level module of SpaceWire CODEC network i/f, encapsulating the CODEC transmitter and receiver
CSpwInitiator	SpaceWire TLM protocol initiator class, implementing the SpaceWire initiator i/f towards the SpaceWire link
CSpwTarget	SpaceWire TLM protocol target class, implementing the SpaceWire target i/f to the SpaceWire link
CStatusReg	The SpaceWire CODEC status register
CTlmTarget	TLM-2.0 Target class implementing the TLM-2.0 target i/f towards the host system
CTxEncode	Transmitter and encoder logic
CUntimedFifo	FIFO model for host interfacing option
DeltaReporting	Reporting functions for the creation of logs and monitors
DeltaScMain	The SystemC-only test-bench entry point, implementing the sc_main method.
DeltaSpacewHeaders	General purpose header files, including the definition of the CODEC configuration class and classes structures used in packet level simulations
ScEnumTrace	Utility header file used to enable the waveform tracing of SystemC enumeration values
SpacewSerial	The SpaceWire custom protocol and payload implementation

Table 1: Overall code structure of SpaceWire TLM

The description of the RtlTestbench sources directory is presented in Table 2:

Directory	Description of contents
GenTb	RTL test-bench commands file parser and generic log messages printing component
SpwrTb	VHDL sources of the SpaceWire interface reference implementation, sources of the components used to generate stimuli and traffic for the reference implementation, verify data received by the reference implementation and monitor its status
TbPkg	VHDL packages for the test-bench
Top	RTL test-bench top-level component, instantiates the VHDL sub-components along with the CODEC SystemC model
UutTb	VHDL sources of the components used to generate traffic and stimuli for the SpaceWire CODEC, verify data received by the CODEC and monitor its status
verif	scripts necessary to build the mixed VHDL-SystemC testbench and run the simulations
cmd	command files for the control and monitoring components of the RTL test-bench, used to form individual testing scenarios

Table 2: Overall structure of the RTL test-bench

4.2 MODEL CONFIGURATION

The SpaceWire TLM realises both the Loosely Timed and the Approximately Timed coding styles, as these are defined in the TLM 2.0 Language Reference Manual [1]. In addition, the model operates on two distinct levels of the SpW standard, namely at the exchange level, and at the packet level, as defined in [2]. The exchange level deals with transactions where the data quantity exchanged is the character, while in packet level, as the name suggests, a complete packet is transferred in every transaction, thus providing a higher level of data abstraction and leading to faster, albeit less detailed, simulations. Moreover, the user of the model is able to select any of the four possible combinations between coding styles and data abstraction.

The selection of the coding style, as well as the specification of all other model parameters takes place through the SpaceWire CODEC configuration class CDeltaSpacewConfig. The defined parameters are passed to all classes affected by the respective values. Therefore, a single point of reference and configuration exists for the entire model, and parameters are propagated throughout the model without the need for further user action. The Spacewire CODEC TLM model is complemented with a set of controllable observation points, realized as logging messages and VCD trace file waveforms. The observation points are enabled/disabled via compilation directives. Additional examples of configuration parameters include CODEC SystemC module names, initiator and target socket names, as well as custom address assignment to memory-mapped CODEC registers, such as the transmitter FIFO full flag and the receiver buffer empty flag. For a complete reference on the configuration options of the SpaceWire CODEC, the reader is referred to the SpaceWire CODEC Development Manual [6].

Finally, regarding the configuration registers of the SpaceWire CODEC, the corresponding RTL implementation does not actually implement them; it merely places the relevant input signals on the interface of the IP core, so that either the simulation test-bench or the surrounding logic can drive them. The same holds true for the status information generated by the core, for which output signals are used in the interface. The SpaceWire TLM model realises the corresponding functionality in the form models of registers, integrated to the host interface. In this manner, the model shall be controllable at run-time, a feature that is not directly supported by its RTL counterpart. Thus, since no implemented reference exists, it was decided that the mapping of control and status registers shall follow that of the SpaceWire core in the Gaisler Research GRLIB [5], and, in particular, that of GRSPW (as opposed to GRSPW2).

5 SPACEWIRE CODEC TLM MODEL TIMING

The development of the SpaceWire CODEC TLM model required the identification of the CODEC operations relating to transactions and the measurement of the corresponding delays, in order to achieve a timing accuracy level as close as possible to the one offered by the RTL CODEC model. Towards this end, the RTL CODEC model delays of interest were identified, measured from the RTL test-benches and assessed against the VHDL code of the RTL IP core.

5.1 SPACEWIRE TLM TRANSACTION DELAYS

Delays extracted from the RTL model are relating to both the transmitter and receiver blocks of the SpaceWire CODEC RTL model. Specifically, for the transmitter block the following timing measurements were performed

- start-up sequence delay of the SpaceWire CODEC transmitter,
- delay between a time code being written to the CODEC transmitter block and the transmitter identifying the time code request from the host system,
- SpaceWire characters transmission delay,
- transmission FIFO write delay,
- recovery controller timing.

The semantics of each delay type are summarized in the table below.

Delay Type	Description
transmitter start-up sequence delay	Time interval between the transmitter being activated and the first transition on the transmitter data and strobe outputs.
time code request delay	Time interval defined by the timing point the time code controller writes a host time code to the transmitter block and the timing point the transmitter becomes aware of the time code request, thus selecting the host time code as the next character to transmit.
characters transmission delay	Time required for the transmitter to output all bits corresponding to a SpaceWire character to the link, i.e. the time between the first and last character bit transmission.
transmission FIFO write delay	Time required for the host system to write a SpaceWire character to the transmission FIFO. depends on the timing of the actual FIFO deployed along with the model; for the purposes of the SpaceWire CODEC TLM model, the FIFO that comes together with the RTL model was used to extract the measurements.
transmitter block recovery controller timing	The CODEC recovery controller is responsible for spilling the tail of the current packet when the link is disconnected as the result of an error. In the RTL model, the recovery controller operation is based on a FSM, therefore the actual delay between successive read operations performed by the controller during the recovery process.

Table 3: SpaceWire TLM Tx delay types

For the receiver block the timing measurements of interest are

- start-up sequence delay of the SpaceWire CODEC receiver,
- the time interval between a time code reception having been completed and the time code being available to the host system,
- the time interval between a data character reception having been completed and the data character storage to the receive buffer, from where it can be retrieved by the host system,
- the receive buffer error recovery delay.

The description of each delay type is included in the table below.

Delay Type	Description
receiver start-up sequence delay	Time interval between the first bit of the first character being received, and the first character reception being complete.
time code reception delay	The delay between the time code reception and the time code being available for the host system, dictated by the time code resynchronization delay.
data character reception delay	The delay between the data character reception and the data character being available for the host system, dictated by the data characters resynchronization delay and the data characters processing stages.
receiver buffer error recovery delay	Time interval that elapses between an error in SpaceWire characters reception being detected – which causes the transition of the link FSM from the RUN to the ERROR RESET state – and the receiver block writing an EEP into the receiver buffer.

Table 4: SpaceWire TLM Rx delay types

The resynchronization delays of the time code and data characters reception are introduced due to the use of different clock domains by the receiver block and the host system.

5.2 IMPACT OF RTL MODEL CONFIGURATION OPTIONS ON DELAYS

The RTL model contains a variety of configuration options for the SpaceWire CODEC. Amongst these, the ones that affect the transaction timing are the pipelining configuration option and the double data rate configuration option.

The pipelining option adds an extra transmitter bit clock period in the transmission of the first data character, and an additional receiver bit clock period in the reception of the first data character. The mechanism utilised in the RTL implementation introduces storage within long combinatorial paths, essentially shortening the critical path, and allowing the system to operate on a higher clock frequency. Thus, the mechanism introduces one additional clock cycle to the operation of the respective part of the system, which is compensated for by the higher overall speed the system is allowed to function at. As far as the TLM is concerned, the activation of the pipeline results in the addition of the extra clock cycle to the annotated delays. If the clock frequency is not increased, with respect to the system running with the pipeline disabled, the overall timing of the model shall indicate that the pipelined version runs slower than the non-pipelined one.

On the other hand, the double data rate configuration option, results in the transmitter block transmitting two bits every transmit bit clock period, effectively reducing the bit transmission time to half the time required when the transmitter is not using double data rate for the same transmitter bit clock.

6 FUNCTIONAL OVERVIEW

6.1 SYSTEM OVERVIEW

A block diagram of the SpaceWire CODEC and expected usage is given in the following figure.

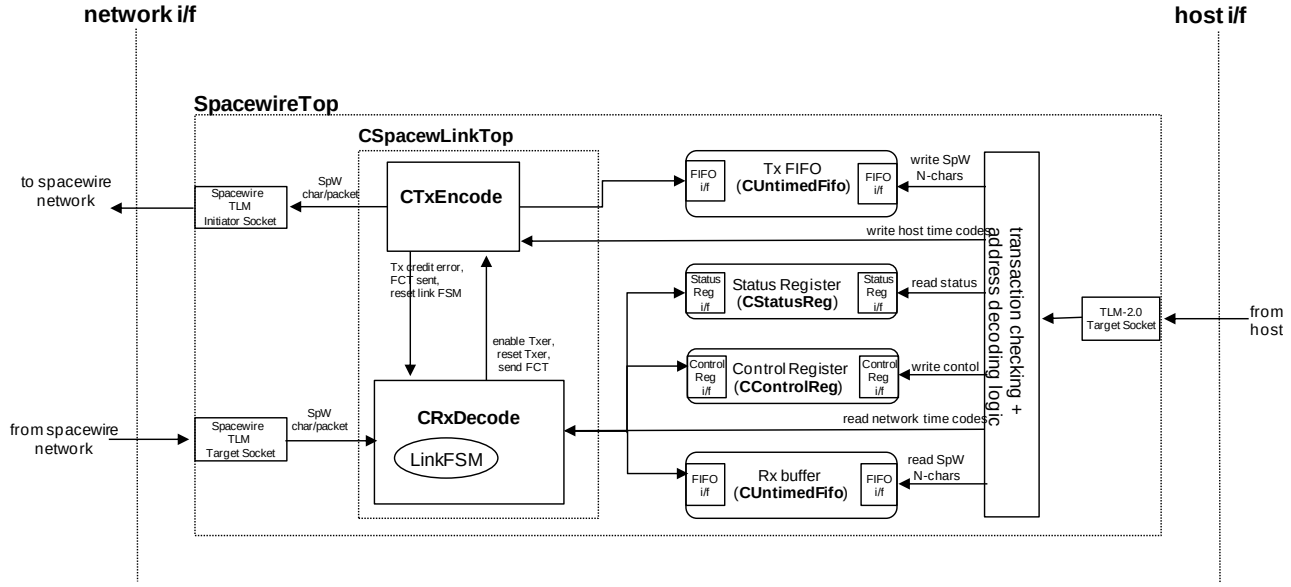

Figure 2: SpaceWire CODEC TLM block diagram

6.2 FUNCTIONS

The transmitter block is implemented in the CTxEncode class. Its duties include reading data from the transmit FIFO and initiating transactions utilizing the TLM-2.0 interface. In order to adhere to the SpaceWire protocol flow control, the transmitter block maintains an internal credit counter indicating the number of data characters it can send across the link. In case the credit counter exceeds the maximum value defined by the SpaceWire standard – 56 characters – a credit error is identified and indicated to the link FSM. Additionally, it performs transmitter FIFO error recovery whenever the transmitter is disabled by the host or a request to flush the transmitter FIFO is issued.

The receiver block is implemented in the CRxDecode class. The receiver block processes incoming transactions, extracts SpaceWire characters and stores them into the receiver buffer. Time codes are identified and stored in a separate variable. The time code reception is indicated to the host system via setting a particular status register byte. Once the host reads the status register value it can acknowledge the new time code reception and read-in the time code received. The CRxDecode class also implements the flow control requirements of the SpaceWire protocol, by keeping a record of the number of entries the receive buffer can accept and requesting FCTs transmission from the transmitter block. Furthermore, the receiver block implements the SpaceWire link FSM, writes the status register bytes when necessary, and reads in the control register value whenever its contents are modified. Both the transmitter and receiver blocks are encapsulated in the CSpacewLinkTop class, whereas the complete TLM of the SpaceWire CODEC, including the transmitter FIFO, receiver buffer, status and control registers is enclosed in the CSpacewireTop class

In order to enable the instantiation of the CODEC TLM model in the RTL test-bench of the RTL SpaceWire CODEC, a VHDL-to-SystemC and SystemC-to-VHDL translation component was realized, commonly reffered to as transactor.

The transactor functionality is implemented in the CDeltaSpacewTract class. It is responsible for translating VHDL signals to transactions for interfacing with the TLM model, and vice versa. The existence of the transactor allows the utilisation of both the RTL test-bench developed for the IP core, and provides the capability to the user to arbitrarily utilise the SystemC model within a HDL simulation, provided the deployed simulator is compliant to this mode of operation. The CSpacewireTop class, which embodies the complete TLM of the SpaceWire CODEC, along with the

transactor functionality are encapsulated in the DeltaTransactor class, which constitutes the top level module instantiated in the VHDL test-bench. As such, the model shall be usable in exactly the same fashion as its RTL counterpart; in the latter case, the SpaceWire CODEC IP User Manual [3] shall be applicable to a great extent to guide the utilisation of the model/transactor composite. A block diagram of the DeltaTransactor component is provided in the figure below.

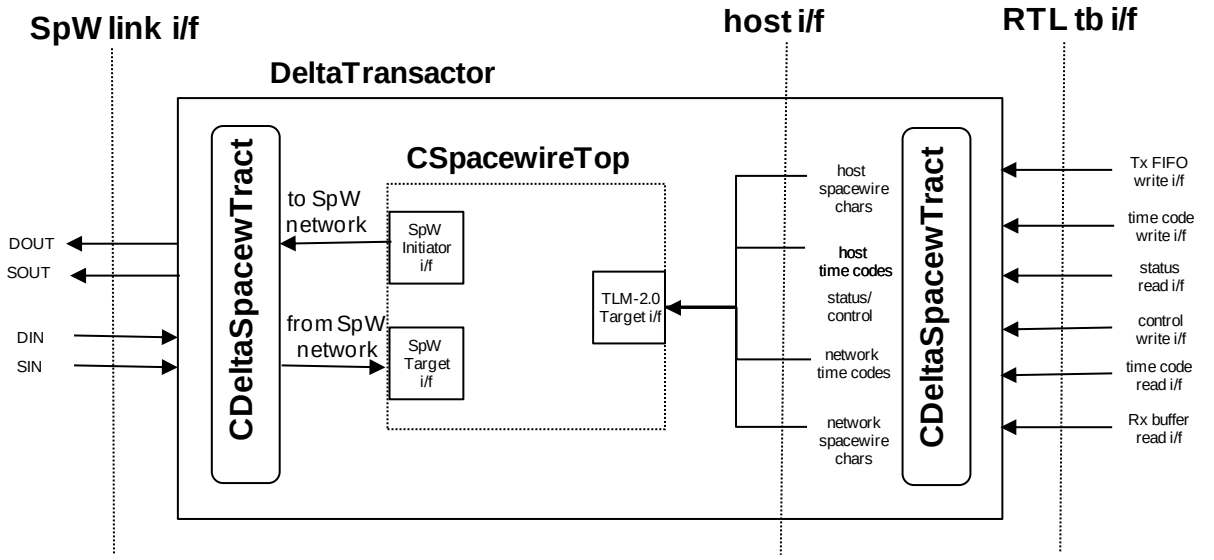

Figure 3: Delta transactor block diagram

6.3 CONFIGURATION OVERVIEW

In accordance to its RTL counterpart, the SpaceWire TLM can be configured to suit the applications of the users as follows:

- Pipelined or non-pipelined.
- DDR outputs or SDR outputs depending the on required data rate and the users selected technology.
- Transmission clock configuration options to allow an independent transmit clock and default reference clock therefore greatly increasing the achievable data rate and decoupling the transmit logic from the system clock logic.
- Capability to discard empty packets - i.e. EOP followed by EOP/EEP and EEP followed by EOP/EEP.
- Ability to reserve host time codes for transmission even when the link is not running.
- Configurable receive buffer/transmit FIFO size.

Furthermore, a number of user-configurable parameters specific to the TLM implementation have been introduced. These include the names and IDs of model components, storage width and depth, etc.

7 MODEL ABSTRACTION LEVELS

The SpaceWire CODEC model operates at two abstraction levels, the packet and the exchange level. The abstraction levels are differentiated by the data transfer unit utilized in network-side transactions. As its name suggests, at the packet level model entire SpaceWire packets are carried in each transaction between the SpaceWire link ends. Specifically, in the packet level mode of operation while the interface to the host system remains the same as in the exchange level, meanly each Tx FIFO/Rx buffer write/read transaction carries a single Spacewire character, a transaction on the network side is only performed when an entire packet is available in the Tx FIFO. To realize this approach, the Tx FIFO not empty flag is only asserted when the host system has stored an entire packet for transmission.

At the exchange level, single SpaceWire characters are carried in both host-side and network-side transactions. The SpaceWire characters are stored to the transmitter FIFO, mapped by the transmitter block logic to the custom SpaceWire payload fields and transferred to the other end of the link via TLM transactions. The other end of the link is expected to extract the SpaceWire character from the transaction and store it to the receiver buffer if necessary. An in-depth description of the custom SpaceWire payload, as well as of the data structures used at the packet level model can be found in [6], whereas example code for writing/reading SpaceWire packets or characters to/from the transmitter FIFO/receiver buffer, can be found in the class emulating the host system – CHostEmulator.cpp – instantiated in the example SystemC-only test-bench that comes along with the model.

8 BUILDING THE MODEL

8.1 NATIVE COMPILATION

The SpaceWire TLM is developed with the purpose of being deployed in system-level simulation models, fully comprised of SystemC TLM building blocks. As such, it is a SystemC implementation, relying on the existence of the SystemC simulation kernel and the TLM library, in order to be compiled with the use of a C++ compiler. In order to make the compilation of the model files easier, the build structure is based on the CMake system, and the GNU gcc compiler. The SpaceWire TLM has been developed using the OSCI SystemC-2.2.0 library, the OSCI TLM-2.0.1 library, and the Greensocket-4.1.0 library, whereas it has been successfully compiled using gcc-4.4.3.

The organisation of the model encompasses a set of *CMakeLists.txt* files which describe the structure of the code and its distribution in files and folders. During the build process, the CMakeLists.txt file included in every sub-folder instructs the creation of a static library from the *.cpp files located in the *src* sub-folder. If the creation of the static library requires header files from other sub-folders, cmake *find_package* directives are used to locate the header files necessary.

The SystemC, TLM and Greensockets packages required for the TLM model to be built, are searched for in the top level CmakeLists.txt so that they become available for all sub-modules. The SystemC package contains both header files and the SystemC static library while the other two packages contain only header files. Two alternatives are provided for locating the packages necessary. The first solution demands the user to manually define the SystemC, TLM, and Greensocs libraries installation directories during the cmake command-line invocation as follows:

- -DSYSTEMC_INSTALL:string={path to systemc static lib folder, include folder is assumed to be reachable by appending ../*include*},
- -DTLM_INSTALL:string={path to TLM installation, include folder is reached by appending *include/tlm*},
- -DGREENSOCKS_INSTALL:string={path to Greensockets installation, include folder is reached by appending *greensocket*}.

The second option assumes that *updatedb* has been executed after the installation of the above packages. The *updatedb* command may be executed with a local scope (i.e. updatedb --localpaths='<path to files of interest>'), only to insert to the filesystem database the files of interest. For each package XXX, a FindXXX.cmake script exists in the cmake-modules subfolder of the source code tree. The FindXXX.cmake script uses *locate* to find potential folders containing the appropriate header files and/or static libraries. In both cases the header/library files in the candidate folders are checked against the versions used during model development; if the verification succeeds, the header/library location is assigned to the XXX_INCLUDE_DIRS and/or XXX_LIBRARIES variables, where XXX is the library name. The SystemC library is considered valid if version 2.2.0 is reported, whereas the TLM library is considered valid if version “TLM 2.0.1 --- 2009-07-15” is reported. The libraries verification procedure requires compilation and execution of testing code included in the FindXXX.cmake scripts. The *dirline.sh* script in the root folder is used by the FindXXX.cmake scripts.

In order for the code to be fully compiled, the user needs to generate a build folder at the top-level of the file hierarchy. Using a command console, and once in the aforementioned “build” folder, the user issues the command:

cmake ../

or the command

cmake -DSYSTEMC_INSTALL:string={pathto} -DGREENSOCKS_INSTALL:string={pathto} -DTLM_INSTALL:string={pathto} ../

depending on the method used to locate the SystemC, TLM and Greensockets libraries. The cmake invocation generates the appropriate makefiles, therefore once its execution is completed, the command

make

should be issued, in order to build the complete model. As mentioned earlier, during the model build process, in the root folder of the project a sub-folder named *cmake_modules* is created. This folder is appended to the default cmake path that is used for locating FindXXX.cmake modules that aid the discovery of packages containing header files and/or static libraries necessary for building the CODEC TLM model. The final executable *DeltaTestbench* is created inside the DeltaScMain sub-folder of the Build directory.

Users of the model may study the source code of the testbench (CDeltaTestbench.cpp) to gain insight on the instantiation of the model within a virtual platform; they may also use the test-bench as a template for the creation of such a platform, by adding instantiation of other models or components.

8.2 MODELSIM COMPILATION

The TLM model produced in DELTA is distributed in two basic folder structures: the former contains the model files, together with the files utilised by the CMake system; the latter contains VHDL test-benches and associated command files, as well as Modelsim command files, used for the compilation of the SystemC and VHDL file, and the loading of the simulation model. The following text describes the process necessary to compile and instantiate the model within Modelsim.

The provided DeltaRunVerif.tcl file, located in the RtlTestbench/verif sub-folder, is the script that contains the commands required to build the mixed SystemC-VHDL simulation and load the simulation scenario the model user wishes to be executed. The DeltaRunVerif.tcl file is in essence a modification of the run_verif.tcl script used to simulate the RTL CODEC model, including commands and invocation of scripts to compile the CODEC TLM model. As such, its invocation is identical to the run_verif.tcl script of the RTL model. Specifically, the script file is run from the Modelsim shell by issuing the following command when in the RtlTestbench/verif sub-folder

#do DeltaRunVerif.tcl <args>

Script arguments <args> are

-help	Prints a help message on how to run the script.
-logwlf	Log all signals to wlf files using “add log -r /*” before run -all.
-config <id>	Run configuration <id> where <id> can be *1*, *2*, etc. or can be *all* to run the full suite of test configurations. Valid configuration IDs are in the range 1 – 83.
-config_range <low> <high>	Run a range of configurations from low to high inclusive. There are a total of 83 configurations.

In order for a simulation run to be performed, a specific model configuration or a configuration range must be specified with the -config <id> or the -config_range <low> <high> arguments.

The DeltaRunVerif.tcl script invokes the DeltaModelComp.tcl file to compile the SpaceWire CODEC TLM sources. In order for the SystemC sources to be compiled, the following paths must be set-up inside the DeltaModelComp.tcl script, prior to running the DeltaRunVerif.tcl.

1. SPACEW_SRC_DIR - points to the root of the SystemC sources,
2. SYSTEMC_SRC_DIR – points to the Modelsim SystemC libraries, required by the Modelsim SystemC compiler, sccom,
3. TLM_SRC_DIR – points to the OSCI TLM library sources,
4. GREEN_SOCKET_DIR – points to the root of the Greensocket library sources.

The DeltaModelComp.tcl script also creates a header file – DeltaMsim.h – inside the SPACEW_SRC_DIR/DeltaSpacewHeaders/include sub-folder. The particular header file contains the definitions of SpaceWire CODEC TLM and transactor parameters which are necessary for compiling the SystemC sources and can not be assigned at run time - specifically the CODEC transmission FIFO and receiver buffer depths - as well as the width in bits of configurable transactor input ports.

It should be noted that only the AT, exchange level model of the SpaceWire CODEC can be deployed in the Modelsim simulation, as the packet level model is too abstract for the RTL test-bench, whereas the LT, exchange level model does not provide the timing accuracy necessary to execute the RTL model test suite. Additionally, the definition of the kind of log messages the user wishes the TLM model to print, should be defined in the DeltaModelComp.tcl script by setting the relating compilation directives to be used when Modelsim invokes the C++ compiler. The CODEC configuration class instantiation in the CODEC top-level component is defined in the SPACEW_SRC_DIR/CDeltaSpacewTract/include/CDeltaSpacewTract.h file, which also contains the instatiation of the top-level SystemC module used in the mixed language simulation, which encapsulates the transactor and the Spacewire CODEC TLM model. One point to note here, is that there are a number of CODEC parameters in the Modelsim simulation which are set-up during the simulation run time and therefore the setting of the particular parameters by the model user has no effect. The following list summarizes the CODEC TLM configuration class parameters which are assigned during simulation run time. For each parameter, the corresponding CODEC configuration class member variable is enclosed in parentheses:

- pipelining configuration (cfgPipeline),
- DDR configuration (cfgDdrOut),
- whether the receiver buffer read clock will be synchronous to the system clock or not (cfgSyncRdClk),
- the system clock period duration (sysClkPeriod),
- the transmit bit clock period duration (txBitClkPeriod),
- the receive buffer read clock period duration (rdClkPeriod),
- whether time codes will be discarded when the link is not in the RUN state (cfgTickInKeep),
- whether empty packets will be discarded (cfgDiscardEmptyPkt),
- The transmission bit clock configuration (txClkCfg).

This is due to the fact that these parameters are specific to each test configuration and are decided during the selection of the configuration to be run. In the RTL model simulation, the aforementioned parameters are either defined via test-bench command files, or in the configuration-specific working_spwrlink_pkg.vhd file, created by the run_verif.tcl script. In the TLM model, they are realized as parameter values, assigned to the relating configuration class members during the transactor class construction. In particular, the DeltaRunVerif.tcl script writes the values of the aforementioned parameters into a text file – SystemcSpwCodecConfig.txt – and the transactor constructor reads in and assigns the values to the configuration class members.

The SpaceWire TLM model has been tested with Modelsim SE version 6.5d. It should be noted that Modelsim comes with a gcc compiler, which is the one utterly used by sccom; the gcc version included in Modelsim 6.5d is 4.1.2. In case the model user wishes to use another compiler, he/she may do so by specifying the path to the compiler executable via the -cpppath option of the Modelsim sccom compiler. The sccom command is invoked in the DeltaModelComp.tcl script.

8.3 MODELSIM TEST-BENCH OUTPUT

During the test runs, the test-bench generates console logs informing the model user about the simulation progress. The console log messages originate either from the RTL test-bench components, or from the SpaceWire TLM model components. RTL test-bench logs are preceded by the current simulation time and contain the command status, as well as the actual RTL test-bench component command in capitalized letters. For example, the log message

> *# 593500 ns: > Command Complete: UUT_DATA_CTRL TRANSMIT_PACKET T150us AAAA 100 01*

informs the model user that the UUT_DATA_CTRL RTL test bench component has completed the transmission of a SpaceWire packet, and, in particular, it has stored the last SpaceWire character of

the packet into the transmit FIFO. For a complete reference of the RTL testbench components commands, refer to [4].

On the other hand, SpaceWire TLM logs fall under three categories

1. Informational messages,
2. Warning messages,
3. Fatal messages.

All three message types result in log messages being output to the console, whereas fatal messages also result in the current simulation run being terminated. Each log message consists of two lines and has the following format: the first line contains the type of the message, the name of the source file which contains the process that issued the message, the simulation time at which the log message was issued and the name of the process issuing the message. For instance:

Info:/home/dimitris/Projects/Delta/spacewire/SystemcSources/Debug/CRxDecode/src/CRxDecode.cpp: 176700 ns - ExecuteLinkFsmM

CODEC_1_RxDecode : 140 - LINK FSM IN RUN STATE

where Info is the message type, the path following the semicolon is the path to the source file, 176700ns is the simulation time at which the message was issued and ExecuteLinkFsmM is the name of the process issuing the log message. The second line contains the actual log message, preceded by the SpaceWire CODEC module ID. In the particular case, the log message originates from the SpaceWire CODEC link FSM - which is implemented inside the receiver block source file CRxDecode.cpp – and informs the user that the link FSM is in the RUN state.

The exact message types generated depend on the observation points enabled by the model user. If for example the model user has activated the host target i/f transactions logging in the CODEC top-level module and the AT model was simulated, a TLM-2.0 transaction log messages sequence would be the following – the path to the source file has been deliberately trimmed for a better understanding of the log messages:

Info: /CTlmTargetTop.cpp: 115490 ns - nb_transport_fw

CODEC_1 : 100 TRANSACTION MOVED TO SEND-RESPONSE PEQ. RETURNING UPDATED (GP, END_REQ, 0 s) TO FORWARD NB CALL.

Info: /CTlmTargetTop.cpp: 115530 ns - BeginResponseM

CODEC_1 : 100 STARTING BEGIN RESPONSE METHOD. PERFORMING READ OPERATION.

Info: /CTlmTargetTop.cpp: 115530 ns - BeginResponseM

CODEC_1 : 100 CALLING nb_transport_bw(GP, BEGIN_RESP, 0 s).

Info: /CTlmTargetTop.cpp: 115530 ns - BeginResponseM

CODEC_1 : 100 nb_transport_bw RETURNED COMPLETED (GP, END_RESP, 0 s)

The log message sequence above informs the model user that the SpaceWire top-level module received a non-blocking forward call at 115290 ns, and placed the relating transaction object into the payload event queue of the begin response method. Forty nanoseconds later at time 115530ns, i.e. after one receiver buffer clock cycle has elapsed, the transaction is retrieved from the PEQ and the Begin Response method is executed. The method performs the command included in the transaction payload – a TLM_READ command in the particular case – and calls the backward non-blocking method with a phase equal to BEGIN_RESP and a delay annotation equal to 0ns. Thereafter, the transaction is completed, since the non-blocking call returned an updated phase equal to END_RESP.

Upon successful completion of a test scenario, the mixed-language test-bench prints the following message:

 # ** Failure: Testbench finished (all commands completed)

The AT, exchange level model CODEC TLM has successfully passed the entire suite of the RTL tests – all test scenarios from 1-83, whereas the LT, exchange level model fails on the execution of the tests, as it does not incorporate the timing accuracy necessary. Specifically, the LT model fails the majority of the RTL tests due to the timing inaccuracy introduced by the substitution of the SystemC process used to set the time code transmission request flag with a plain function call. This has the side effect of not scheduling the setting of the flag indicating the host request for the transmission of a time code after the delay interval extracted from the RTL model, but setting the relating flag immediately. During simulation, the CODEC TLM is first instructed to transmit a time code, and then the Spacewire link FSM is immediately disabled, in order for the time code to not be transmitted. In the AT model, the delay between the time code transmission request and the assertion of the time code transmission request flag, ensures that the link is disabled prior to the assertion of the request flag and therefore the time code is not transmitted. In the LT model however this delay is not accounted for, therefore the time code transmission request flag is asserted prior to the link being disabled. Due to the fact that the configuration parameter that permits a time code request to be reserved even if the link is not in the RUN state is set, the time code request flag remains asserted until the link enters the RUN state again, and the time code is transmitted though it should not. The test-bench then fails issuing the following message:

 # Testbench failed in TB_SPWR_STATUS TICKOUT VAL_ALWAYS_EQUALS T50us T1ns 0

In addition, the RTL test-bench tests scenarios require mechanisms provided by the exchange level model to be performed, such as flow control and parity checking. Hence, they are not applicable to the packet-level model implementation of the TLM. For example, the RTL test-bench examines the detection of various errors by the unit under test, transmitting SpaceWire character sequences causing such errors. For instance, a receiver credit error is induced to the unit under test via transmitting more characters than the unit under test has requested, a transmitter credit error is induced via the transmission of a series of FCTs in order to cause the transmitter of the unit under test to exceed its maximum credit, or an escape sequence error is generated by transmitting successive ESC characters. As the packet-level model, in addition to adhering to more relaxed timekeeping as compared to the exchange-level, cannot by definition support such mechanisms, it cannot be subjected to the RTL test-bench verification process either.

9 MODEL DEPLOYMENT

The present chapter provides details on the deployment of the SpaceWire TLM, targeting a SystemC simulation model. Details on the configuration of the model are provided, and its interface towards other components for the development of a system-wide model is discussed.

9.1 SPACEWIRE TLM MODEL CONFIGURATION

The SpaceWire TLM model components can be configured via the CODEC configuration class, which defines a number of public member variables; each variable holds a model configuration option. The configuration class, namely *CDeltaSpacewConfig* (CDeltaSpacewConfig.cpp), provides a centralized configuration point for the entire model, where component-specific parameters are defined, along with parameters that dictate the system-wide behaviour, extracted from the RTL model. The SpaceWire CODEC TLM accepts a pointer to the configuration structure as a constructor argument, thus granting access to the configuration options for the components it encapsulates. In general, the configuration options fall under the following categories

- simulation type,
- time codes transmission options,
- RTL model extracted options,
- clock period configuration,
- address configuration,
- naming and identification assignment,
- CODEC buffering.

Simulation type options are relating to the CODEC abstraction level used in a simulation, i.e. packet or exchange, as well as the coding style to be used, LT or AT. Additionally, for packet level simulations, the model can be configured to ignore errors injected in packets. Time codes transmission options enable the CODEC to transmit time codes, whereas they also provide a back-door to circumvent the Spacewire standard requirement that a single node must transmit time codes in a Spacewire network. However, the particular option allowing a Spacewire CODEC to both transmit and receive time codes is provided for functional test purposes only, and should not be set in a realistic simulation scenario.

Options extracted from the RTL model define behavioural and timing aspects of the TLM. Specifically, the model can be instructed to take into account the timing implications induced by the use of pipeline and/or double data rate outputs. Additionally, thresholds can be set for the model programmable empty flag – i.e. the number of SpaceWire N-char entries the receiver buffer should contain for the flag to be unset; this feature could be used for the incorporation of a DMA controller in the SoC, which would handle the transfer of N-chars/packets from the receiver buffer to host system memory. Thresholds can be also set for the maximum number of outstanding N-chars the CODEC receiver should expect, and consequently the number of FCTs that should be transmitted by the CODEC. Under normal operating conditions this should be set to the receiver buffer depth, however smaller values are permitted. It should be noted though that if the value configured exceeds the buffer storage capability, it is ignored and the buffer depth is used as the outstanding N-chars limit. Furthermore the maximum transmitter credit allowed – the number of SpaceWire N-chars the transmitter can send – can be set. The transmitter credit is incremented every time a FCT is received, and if the credit available exceeds the value configured, a transmitter credit error occurs. In order to be compliant with the SpaceWire protocol, the maximum transmitter credit should be set to 56. Finally, the CODEC TLM can be instructed to reserve time codes received from the host system when the link is not running until the link enters the RUN state, whereas the empty packets handling behaviour, i.e. whether empty packets are discarded or not, can be set as well. One point to note here regarding time codes handling, is that the RTL CODEC does not utilize a buffer to store all time code requests from the host. Therefore, in spite of the system being configured to reserve time codes, a time code shall be overwritten by the next one. Therefore only the most recently received time code will be transmitted once the link enters the RUN state.

The RTL CODEC model includes a plethora of clock configuration options, the majority of which are concerned with the way the clocks are physically generated. Since the clocking options have a direct impact on the deployment of resynchronization flip-flops which are added to the SpaceWire characters/time codes data path and in certain cases affect the data path timing, they are incorporated as a configuration parameter. Additionally, for the purposes of the TLM the clock period durations also affect the timing of data-related operations. Hence, the SystemC CODEC model system clock, transmission clock and receiver buffer read clock periods can be defined. The system clock period is used in calculating the delay relating to transmission FIFO operations – reading the fifo full flag, writing SpaceWire N-chars to the FIFO, reading from and writing to the status and control register respectively, writing the host time code to the transmitter block and reading the time code received from the receiver block. The transmission clock affects the characters transmission timing, as it defines the bit clock period and therefore the time interval required for a character transmission to be complete. Finally, the read receiver buffer clock affects the delay of the receiver buffer empty flag and SpaceWire N-chars read operations.

The CODEC address configuration options define the addresses of the CODEC model memory-mapped locations, in particular the transmission FIFO and receiver buffer along with the relating full/empty flags addresses, as well as the transmitter block host time code register and receiver block network time code addresses. Most of the aforementioned locations do not correspond to RTL model registers, with the obvious example being the FIFO/buffer full/empty flags, but have been modelled as such in the TLM. Last but not least, the memory map of the CODEC status and control register was based on the GRSPW SpaceWire core of the GRLIB, thus these addresses are hard-coded in the codec configuration class as static const variables.

The naming and identification assignment options enable the model user to assign meaningful IDs and names to the model building blocks, in order to gain observability of the events logged (e.g. to identify easily which components produced what events in the log). CODEC buffering options allow for the selection of the transmitter FIFO and receiver buffer depths and widths. However, the width values should not be set arbitrarily; the selection of the actual buffer width values depends on the simulation level used – packet or exchange. The overall configuration parameters are summarized in the following table, in the order they are declared in the CODEC configuration class.

Parameter	Type	Description
Simulation Type Options		
tlmCodingStyle	enum TTlmCodingStyle	TLM coding style to be used – LT or AT
spwAbstractLevel	enum TSpwAbstractLevel	Simulation abstraction level – packet or exchange
ignoreErrorInjection	bool	Ignore errors injected into SpW packets – valid only for packet level simulations
tcodesSendEnable	bool	Enables the CODEC TLM to send time-codes. If set to false, the CODEC will not transmit time codes received from the host system.
multipleTcodeMasters	bool	Enables the CODEC TLM to both send and receive time-codes when set to true, thus bypassing the Spacewire standard requirement that a single node in a Spacewire network must transmit time codes.
RTL model-Extracted Options		
spacewMaxCredit	unsigned char	Maximum transmitter credit allowed – should be set to 56 for SpW protocol compliance
cfgRxBufProgVal	unsigned char	Receiver buffer programmable empty flag threshold. The flag is set to true if the number of buffer entries <= threshold
cfgDdrOut	bool	use DDR outputs

Parameter	Type	Description
cfgPipeline	bool	Use pipeline in the transmitter and receiver blocks
cfgTickInKeep	bool	Keep host time codes when the link is not in the RUN state
cfgDiscardEmptyPkt	bool	Discard empty packets – EOP followed by EOP/EEP, EEP followed by EOP/EEP
cfgMaxCredit	unsigned char	Maximum N-chars expected – should be set equal to the receiver buffer depth
txClkCfg	enum TTxClockCfg	Transmission bit clock configuration options, as defined in [3]
cfgSyncRdClk	const bool	Indicates whether the receiver buffer read clock is synchronous to the system clock or not
Clock Period Configuration Options		
systemClkPeriod	sc_time	System clock period
txerBitClkPeriod	sc_time	Transmission bit clock period
rdClkPeriod	sc_time	Receiver buffer read clock period
Address Configuration Options		
netTcodeAddr	unsigned int	Time code received register address
hostTcodeAddr	unsigned int	Time code to transmit register address
rxBufferAddr	unsigned int	Receiver buffer address
rxBufEmptyFlagAddr	unsigned int	Receiver buffer empty flag address
rxBufProgFlagAddr	unsigned int	Receiver buffer programmable flag address
txFifoAddr	unsigned int	Transmission FIFO address
txFifoFullFlagAddr	unsigned int	Transmission FIFO full flag address
Component-specific Naming, Identification, Debugging Options		
Status Register Options		
statusRegId	unsigned int	Status register ID
Control Register Options		
controlRegId	unsigned int	Control register ID
SpaceWire Top Options		
spacewTopId	unsigned int	SpaceWire top module ID
spacewTopModuleName	char*	SpaceWire top module name
spwTopTlmTargetSockName	char*	SpaceWire top TLM-2.0 target socket name
spwTopSerialTargetSockName	char*	SpaceWire top SpaceWire target socket name
spwTopSerialInitSockName	char*	SpaceWire top SpaceWire initiator socket name
spacewLinkTopModuleName	char*	SpaceWire link top module name
Transmitter Options		
txId	unsigned int	Transmitter block ID
txEncodeName	char*	Transmitter block name
Receiver Options		
rxId	unsigned int	Receiver block ID
rxDecodeName	char*	Receiver block module name

Parameter	Type	Description
Receiver Buffer Options		
rxBufDepth	unsigned int	Receiver buffer depth in entries
rxBufWidth	unsigned int	Receiver buffer width in bytes – should be set to 2 for exchange level and to sizeof(void*) for packet level
Transmitter FIFO Options		
txFifoDepth	unsigned int	Transmitter FIFO depth in entries
txFifoWidth	unsigned int	Transmitter FIFO width in bytes – should be set to 2 for exchange level and to sizeof(void*) for packet level simulations

Table 5: SpaceWire CODEC TLM configuration parameters (CDeltaSpacewConfig)

As mentioned earlier, the model also caters for configurable debugging options, via a selection of configurable observation points of the CODEC model operations, as well as the creation of VCD trace files with waveforms for particular CODEC components. In order to increase simulation speed by compiling out code relating to the insertion of observation points when necessary, all debugging options are enabled/disabled via compilation directives instead of configuration class variables. The actual compilation directives required to enable the corresponding observation points are summarized in the table below.

Compilation Directive	Impact
Status Register Directives	
STATUS_REG_LOG	Enable the printing of the status register value whenever it is written
STATUS_REG_LOG_WAVE	Enable the creation of a VCD trace file with status register values
Control Register Directives	
CONTROL_REG_LOG_WAVE	Enable the creation of a VCD trace file with control register values
SpacewireTop Directives	
TLM_TARGET_TRANS_LOG	Enable log messages coming from the implemetation of the TLM-2.0 target i/f methods, which relate to TLM-2.0 transactions
SPW_INITIATOR_TRANS_LOG	Enable log messages coming from the implementation of the SpaceWire initiator i/f methods
SPW_TARGET_TRANS_LOG	Enable log messages coming from the implemetation of the SpaceWire target i/f methods
Transmitter Directives	
TX_LOG_TRACE	Enable the printing of all log messages coming from the transmitter block
TX_LOG_TCODES	Enable the printing of log messages relating to time codes
TX_LOG_DATA	Enable the printing of log messages relating to N-chars
TX_LOG_FCTS	Enable the printing of log messages relating to FCTs – FCTs that are part of the NULL sequence are not logged

TX_LOG_NULLS	Enable the printing of log messages relating to the transmission of NULLs
TX_LOG_RECOVERY	Enable the printing of log messages relating to the Recovery Controller operation
TX_LOG_WAVE	Enable the creation of a VCD trace file for the transmitter block
Receiver Directives	
RX_LOG_TRACE	Enable the printing of all log messages coming from the receiver block
RX_LOG_TCODES	Enable the printing of log messages relating to time codes reception
RX_LOG_DATA	Enable the printing of log messages relating to N-chars reception
RX_LOG_FCTS	Enable the printing of log messages relating to FCTs reception – FCTs that are part of the NULL sequence are not logged
RX_LOG_NULLS	Enable the printing of log messages relating to the reception of NULLs
RX_LOG_CONTROL_REG	Enable the printing of the value of the control register whenever it is read
RX_LOG_FSM	Enable the printing of log messages from the link FSM
RX_LOG_WAVE	Enable VCD trace of the receive block, including the link state variables of the link FSM
Rx Buffer directives	
RX_BUFFER_LOG_WAVE	Create VCD trace file with receiver buffer values. Currently up to 256 bytes may be traced.
Tx FIFO directives	
TX_FIFO_LOG_WAVE	Create VCD trace file with transmitter FIFO values. Currently up to 256 bytes may be traced.

Table 6: CODEC TLM observation points compiler directives

9.2 SPACEWIRE TLM MODEL INSTANTIATION

The SpaceWire TLM possesses three distinct interfaces:

a. One SpaceWire initiator type interface for the network side
b. One SpaceWire target type interface for the network side,
c. One TLM-2.0 target type interface for the host side,

All three interface types are connected to the SpaceWire top-level module, which acts as the communication wrapper for the CODEC TLM. For network side transactions, the Tx logic of the model prepares transactions, i.e. fills in the custom SpaceWire payload fields and utilizes the top-level module callback to perform the network-side blocking/non-blocking call. Similarly, the Rx logic provides a callback used by the top-level module to pass down network-side transactions for further processing. On the host side, the SpaceWire top-level component performs address decoding on incoming transactions, and uses the relating component callback to perform the actual transaction, whether that is targeting the CODEC FIFOs, configuration or status register.

Instantiation of the model in a larger scale simulation or a virtual platform shall end up being as simple as declaring a constructor of the following form:

SpacewireTop (sc_module_name moduleName, CDeltaSpacewConfig configStruct)*

provided the configuration class has been properly instantiated and parameterized in accordance to the user requirements.

The shows a code snippet that carries out the configuration of the model, by assigning values to the variable members of the configuration class. Having configured the model, its instantiation, as well as the instantiation of any component that shall be connected to it, needs to be realised. Finally, the interconnection between the components shall be defined. The code snippet in Figure 5 presents an example where an instance of the SpaceWire TLM and an instance of a host emulator are instantiated and interconnected.

```
CDeltaSpacewConfig c_uutCodecConfig1 (
   //========== system-wide configuration ================
   TlmCodStyleApproxTimed, //codec TLM coding style
   SpwAbstrXchangeLevel,   //model Abstraction level
   false,                  //ignore error injection
   true,                   //enable the TLM to send time codes
   true,                   //if set to true, the TLM may both send
                           //and rx time codes without issuing an
                           //error
   56,                     //Spacewire Protocol max credit allowed
   //========== RTL-relating configuration ================
   16,                     //CODEC programmable empty flag threshold
   false,                  //DDR outputs
   true,                   //pipelined Tx + Rx blocks
   false,                  //Keep time codes when link not running?
   false,                  //Discard empty packets?
   32,                     //max number of outstanding Nchars expected
   TxClockCfgSysSlowclkDiv,//Tx bit clock configuration
   false,                  //Is the read rx buf clock synchronous
                           //to the system clock?
   //========== clock periods set-up =====================
   sc_core::sc_time(5, SC_NS),    //system clock period
   sc_core::sc_time(5, SC_NS),    //tx clock period
   sc_core::sc_time(20, SC_NS),   //read clock for the rx buffer
   //========== CODEC address space configuration ==============
   HostNetTcodeAddress,    //network time code register address
   TxTimeCodeRegAddress,   //host time code Register Address the host
                           //system writes time codes to
   RxBufferAddr,           //rx buffer address
   RxBufEmptyFlagAddr,     //rx buffer empty flag address
   RxBufProgFlagAddr,      //rx buffer programmable flag address
   TxFifoAddr,             //tx FIFO address
   TxFifoFullFlagAddr,     //tx FIFO full flag address
   //============== status register configuration options ===================
   210,                    //status register ID
   false,                  //print log message whenever status register is written
   //============== control register configuration options ==================
   220,                    //control register ID
   //============== CSpacewireTop module configuration options ==================
   200,                          //CSpacewireTop module Id
```

```
        "UUT_CODEC_1",                  //CSpacewireTop module name
        "UUT_CODEC_1_TlmTargSock",      //CSpacewireTop TLM-2.0 target socket name
        "UUT_CODEC_1_SpwTargSock",      //CSpacewireTop serial target socket name
        "UUT_CODEC_1_SpwInitSock",      //CSpacewireTop serial initiator socket name
        "UUT_CODEC_1_LinkTop",          //SpacewireLinkTop module name
        //========= CSpacewireTop module transaction debugging messages options =========
        false,                  //enable log msgs from TLM target i/f
                                //implementations to debug TLM-2.0
                                //transactions
        false,                  //enable log msgs from network initiator
                                //i/f to debug SpW Initiator transactions
        false,                  //enable log msgs from network target
                                //i/f to debug SpW Target transactions
        //============== Tx Block configuration options ============================
        230,                        //TxEncode ID, used for logging
        "UUT_CODEC_1_TxEncode",     //TxEncode module name used by SystemC
        //============== Rx Block configuration options ============================
        240,                        //RxDecode ID, used for logging
        "UUT_CODEC_1_RxDecode",     //RxDecode module name used by SystemC
                                    //simulation kernel
        //============== Rx Buffer configuration options ============================
        32,
        2,
        //============== Tx FIFO configuration options ============================
        32,
        2
    );
```

Figure 4: Example of configuration class instantiation

```
CHostEmulator c_hostEmulator1 (c_hostEmCfg1->moduleName, c_hostEmCfg1, c_spwCfg1);
CHostEmulator c_hostEmulator2 (c_hostEmCfg2->moduleName, c_hostEmCfg2, c_spwCfg2);
CSpacewireTop c_spacewireTop1 (c_spwCfg1->spacewTopModuleName, c_spwCfg1);
CSpacewireTop c_spacewireTop2 (c_spwCfg2->spacewTopModuleName, c_spwCfg2);

 //bind TLM-2.0 sockets
    c_hostEmulator1.m_initSocket(c_spacewireTop1.m_targetSocket);
    c_hostEmulator2.m_initSocket(c_spacewireTop2.m_targetSocket);

    //bind SpW sockets
    c_spacewireTop1.m_spwInitSock(c_spacewireTop2.m_spwTargetSock);
    c_spacewireTop2.m_spwInitSock(c_spacewireTop1.m_spwTargetSock);
```

Figure 5: Example of model instantiation

10 TLM MODEL HOST-SIDE TRANSACTIONS

The SpaceWire CODEC TLM model implements a TLM-2.0 target interface towards the host system, through which access to the model control and status registers, the receiver buffer and transmitter FIFO is granted. Additionally, the host system may retrieve time codes received from the SpaceWire link – stored in the receiver block – and write time codes to be transmitted – in the transmitter block. In order for the LT and AT coding styles to be supported by the CODEC TLM, both the blocking and the non-blocking interface are implemented.

10.1 TLM-2.0 TRANSACTION EXECUTION FLOW

Once the SpaceWire CODEC TLM receives an incoming transaction – either in the LT or AT modelling style – an initial check on the transaction attributes is performed. The CODEC TLM does not support transactions with the byte enable pointer set, or transactions having a streaming width less than the transaction length, since streaming transfers to the CODEC are not required. Thereafter, the transaction address is decoded. If the transaction is a TLM_WRITE to a read-only target or a TLM_READ to a write-only target, an ADDRESS_ERROR_RESPONSE is sent back to the initiator and the transaction is not performed. If the transaction streaming width is less than the transaction length specified for a particular address, the transaction is not performed and a TLM_BURST_ERROR_RESPONSE is sent back to the initiator. In case all of the above conditions are satisfied, the command is executed. A summary of the transaction verification process is provided in the following figure.

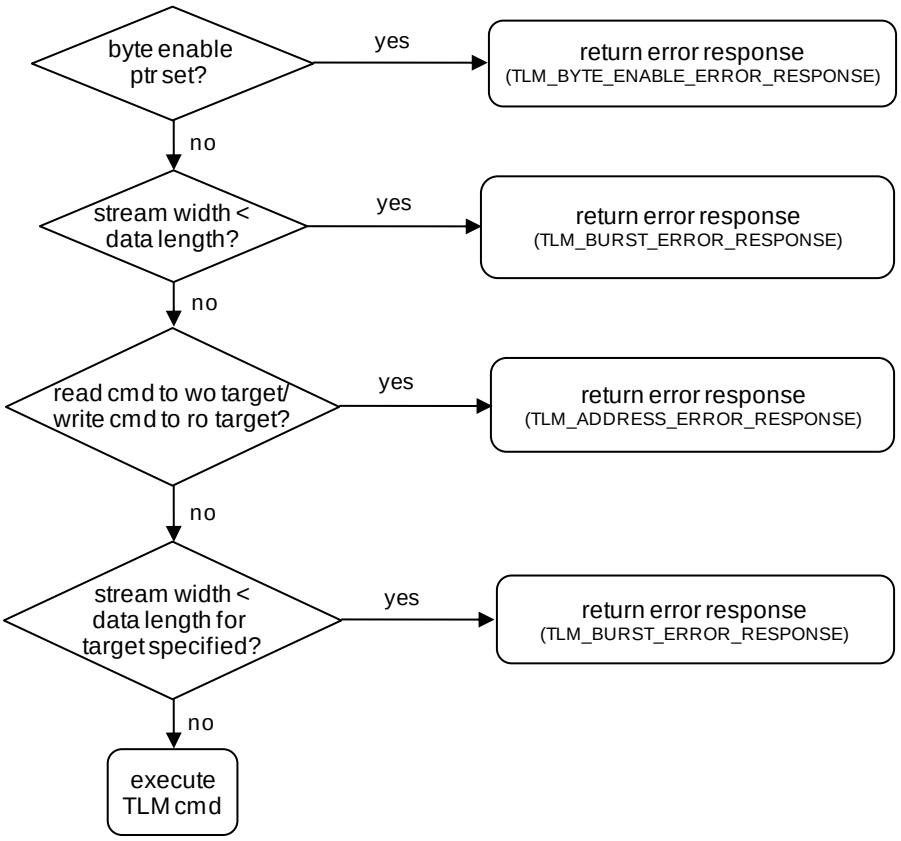

Figure 6: CODEC TLM-2.0 transaction execution flow

10.2 TLM-2.0 CODING STYLES SUPPORT

The SpaceWire CODEC TLM supports both the AT and LT coding styles, i.e. it implements both the blocking and non-blocking interface of the TLM-2.0 standard. The LT model performs the command requested – after the previously mentioned transaction validity checks have been passed – and increments the transaction delay argument by the internal delay of the operation requested. Therefore, the use of temporal decoupling is allowed on the host-side interface, as the implementation of the b_transport does not call the wait() method. An exception to this approach is the write operation of the control register: whenever a write is performed on the control register, the wait() method is called with a delay argument equal to the one included in the transaction; once the wait interval has expired, the control register is written. This approach is due to the fact that the writing of the control register may trigger an entire sequence of events – i.e. writing the bit that enables the transition of the link FSM state from READY to STARTED - that will lead to increased timing inaccuracy, unacceptable even for the LT model.

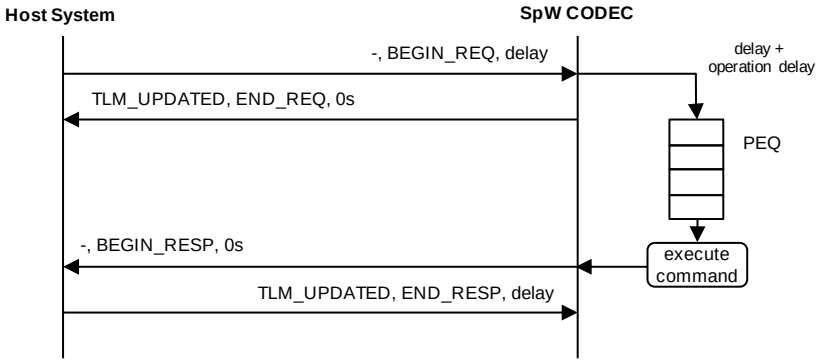

Figure 7: CODEC TLM-2.0 AT transaction timing diagram

As far as the AT model is concerned, the nb_transport_fw implementation places the transaction in a payload event queue, from which it is retrieved when the timing point at which it should be served has been reached. The exact timing point is dictated by the delay annotated onto the transaction plus the delay required to perform the operation specified- e.g. the transmission FIFO write delay. Thereafter, the command is executed and the nb_transport_bw method is used to inform the initiator of the command execution. AT transactions are marked by 4 timing points, using the relating methods return path. The exact message sequence is summarized in the diagram above.

The delays associated with each read/write command, were decided based on the clock period of the clock used to read/write the actual TLM read/write target. The delay for each operation as well as the type of operation supported for each target is provided in the following table.

TLM Target	Operation Type	Delay
time code received	read-only	1 system clock cycle
time code to tx	write-only	1 system clock cycle
rx buffer	read-only	1 read buffer clock cycle
rx buffer empty flag	read-only	1 read buffer clock cycle
rx buffer programmable empty flag	read-only	1 read buffer clock cycle
tx FIFO	write-only	1 system clock cycle
tx FIFO full flag	read-only	1 system clock cycle
status register	read-only	1 system clock cycle
control register	write-only	1 system clock cycle

Table 7: SpaceWire CODEC TLM-2.0 targets

10.3 TLM-2.0 DEBUG INTERFACE

The SpaceWire CODEC TLM model also implements the transport debug interface of the TLM-2.0 standard. The particular interface is a non-intrusive interface, therefore the corresponding implementation performs the read/write operation requested, without triggering any relating events and/or advancing simulation time. Due to the fact that the CODEC TLM control register implementation notifies an even whenever it is written so that a relating method can read its value and trigger the link FSM if necessary, it is evident that performing a transport debug write operation in the control register will have no side-effect on the system operation, as it will only be intercepted the next time a “normal” write operation will be performed – and provided that the new operation will not overwrite the control register bytes set in the debug transaction. The debug targets are also categorized into read-only and write-only, according to the operations they support in the forward blocking/non-blocking interface methods – see previous table.

As the TLM-2.0 standard dictates, if the entire number of bytes requested in the debug transaction could not be read/written, the interface implementation returns the actual number of bytes read/written.

10.4 STATUS REGISTER

The CODEC status register is an array of 19 bytes, where each byte is associated with a particular event occurrence. The status register bytes along with the information they hold are summarized in the following table.

Status Register Byte No	Description
0	Time code received indication; set whenever a new time code has been received from the SpaceWire network
1	Receiver credit error indication
2	Transmitter credit error indication
3	Escape error indication
4	Disconnection error indication, set when a disconnection error occurs, i.e. whenever the other side of the link injects a disconnection error.
5	Parity error indication
6	Link FSM state. The state is encoded using 3 bits as per the RTL CODEC : ERROR RESET 0x00 ERROR WAIT 0x01 READY 0x02 STARTED 0x03 CONNECTING 0x04 RUN 0x05
7	Set when the link FSM is in the RUN state
8	Receiver got NULL indication; remains asserted after the first NULL is received
9	Receiver got FCT indication; remains asserted after the first FCT is received
10	Receiver got N-chars indication; remains asserted after the first N-char is received. For packet level simulations, this byte is set once the first packet received.
11	Receiver got time code indication; remains asserted after the first time code is received
12	Set if the transmitter has credit to send one more N-char
13	N-char sequence error indication, set if an N-char is received before the link FSM enters the RUN state

Status Register Byte No	Description
14	Time code sequence error, set if a time code is received before the link FSM enters the RUN state
15	Credit error in RUN state indication; set when a receiver or transmitter credit error occurs while the link FSM is in the RUN state
16	Escape error in RUN state indication; set when an escape error occurs while the link FSM is in the RUN state
17	Parity error in RUN state; set when a parity error occurs while the link FSM is in the RUN state
18	Disconnection error in RUN state; set when a disconnection error occurs while the link FSM is in the RUN state

Table 8: CODEC TLM status register fields

The status register byte 0, the time code received flag, does not correspond to a RTL CODEC status register bit but has been added to the status register to enable the time code reception based on status register polling by the host system.

The SpaceWire CODEC status register is accessible via the TLM-2.0 target interface implemented by the top-level module as a read-only target. Hence for a status register read operation to be realized, the host system must prepare a TLM-2.0 based transaction and set the generic payload fields accordingly. The generic payload data pointer should be set to a 19-byte wide array, which will contain the status register fields upon returning from the forward blocking/non-blocking call. The transaction data length, streaming width must be set to the byte array size, whereas the generic payload address field must be equal to the one defined in the GRSPW core of the GRLIB (0x04). It should be noted that the status register time code received byte is automatically cleared after every read operation, to ensure that the same time code is read by the host system exactly once. Additionally, the status register is cleared whenever the CODEC TLM is reset.

Figure 8: Status register read example

The CODEC implementation includes a constant defining the status register length in the DeltaSpacewCodec.h header file – *StatusRegFlags,* and an enumeration type for the status register fields - *TStatusRegisterFields*, where each enumeration member corresponds to a register byte, in the DeltaSpacewTypes.h file. Therefore, instead of using hard-coded values for the transaction length and streaming width, the programmer may choose to use the *StatusRegFlags* constant. Additionally, he may choose to use the enumeration type to retrieve individual register bytes values instead of using hard-coded array indexes. The status register fields enumeration type is presented in the SpaceWire CODEC TLM Development Manual. The accompanying code snippet is taken from the CHostEmulator class implementation, used in the CODEC SystemC-only test-bench and summarizes the procedure necessary to read the status register, as well as the use of the aforementioned constant and enumeration type. The m_statusRegRead variable is the 19-byte wide array where the register contents are copied to. The m_statusRegReadEv used in the non-blocking call is the event notified via the nb_transport_bw() method implemented by the CHostEmulator class. The event is notified whenever the BEGIN_RESP phase is received by the CHostEmulator class.

10.5 CONTROL REGISTER

The control register stores control information originating from the host system which affects the CODEC operation. It is implemented as an 11-byte wide array, accessed by the host system via the TLM-2.0 target interface of the SpaceWire top-level module for write operations – i.e. it is a write-only target. Each byte corresponds to a specific control option. The control register address is based on the GRLIB GRSPW SpaceWire core, therefore it is equal to 0x00.

The control register bytes along with their meaning is summarized in the following table.

Control Register Byte No	Description
0	Link Disable. Causes the link to be disabled when the link FSM is in the RUN state, i.e. causes a transition from the RUN to the ERROR RESET state
1	Link Start. When set, causes the link FSM to transit from the READY to the STARTED state.
2	Auto Start. When set, the link FSM may transit from the READY to the STARTED state once it receives a NULL – while in the READY state.
3	Flush Tx. When set causes the transmitter FIFO to be flushed. Coupled with the Recovery Controller operation.
4	Tx Rate. Emulates the variable transmission data rate generation of the RTL CODEC model, where the txRate is used for loading the counters present in the clock divider, or the clock enable generator, depending on the transmit bit clock configuration. In the TLM, the tx rate value is used to calculate the multiplier of the transmission bit clock period to generate a variable data rate.
5	System Reset. Emulates the hardware reset of the RTL CODEC model. Likewise the RTL CODEC, it is active low.
6	Inject Disconnection Error. Used by the host system to emulate the disconnection mechanism of the SpaceWire protocol. When set and if the TLM is configured to not ignore errors injected, the TLM sends a disconnection error at the other side of the link, causing link re-initialization. The byte is auto-cleared once the register is read by the CODEC model, therefore it need not be cleared by the host system.Valid at both the exchange and packet level models.
7	Inject Parity Error. Used by the host system to emulate the parity error mechanism of the Spacewire protocol in packet level simulations. When set and if the TLM is configured to not ignore errors injected, the TLM sends a parity error at the other side of the link, causing link re-initialization. The byte is auto-cleared once the register is read by the CODEC model, therefore it need not be cleared by the host system. Valid at the packet level simulation only.
8	Inject Escape Sequence Error. Used by the host system to emulate the escape sequence error mechanism of the Spacewire protocol in packet level simulations. When set and if the TLM is configured to not ignore errors injected, the TLM injects an escape sequence error in the transaction to the other side of the link, causing link re-initialization. The byte is auto-cleared once the register is read by the CODEC model, therefore it need not be cleared by the host system. Valid at the packet level simulation only.
9	Inject Character Sequence Error. Used by the host system to emulate the character sequence error mechanism of the Spacewire protocol in packet level simulations. When set and if the TLM is configured to not ignore errors injected, the TLM injects a character sequence error in the transaction to the other side of the link, causing link re-initialization. The byte is auto-cleared once the register is read by the CODEC model, therefore it need not be cleared by the host system. Valid at the packet level simulation only.
10	Inject Credit Error. Used by the host system to emulate the credit error mechanism of the Spacewire protocol in packet level simulations. When set and if the TLM is configured to not ignore errors injected, the TLM injects a credit error in the transaction to the other side of the link, causing link re-initialization. The byte is auto-cleared once the register is read by the CODEC model, therefore it need not be cleared by the host system. Valid at the packet level simulation only.

Table 9: CODEC TLM control register fields

The SpaceWire CODEC control register is accessible via the TLM-2.0 target interface implemented by the top-level module. The steps necessary to perform a control register write operation are analogous to the ones required for the status register read operation: the generic payload data pointer should be set to a 7-byte wide array, where the control register contents will be copied from. The transaction data length, streaming width must be set to the byte array size, whereas the generic payload address field must be equal to the one defined in the GRSPW core of the GRLIB.

As is the case for the status register, the control register length is also defined in the DeltaSpacewCodec.h header file – Control*RegFlags,* whereas an enumeration type for the control register fields – *TControlRegisterFields*, where each enumeration member corresponds to a register byte, is defined in the DeltaSpacewTypes.h file. Therefore, the *ControlRegFlags* constant may be used to set the transaction data length and streaming width fields. Additionally, the enumeration type may be used to individually set register bytes values instead of using hard-coded array indexes. The control register fields enumeration type is presented in the SpaceWire CODEC TLM Development Manual. The code snippet below is extracted from the CHostEmulator class implementation, used in the CODEC SystemC-only test-bench and acts as an example of the procedure necessary to write the CODEC TLM control register, as well as a demonstration of the usage of the aforementioned control register constant. Them_controlRegWrite variable is the 7-byte wide array where the CODEC TLM register contents are copied from, whereas the m_controlRegWriteEv is a SystemC event variable used in the AT modelThe m_controlRegWriteEv variable is notified via the nb_transport_bw() method implemented by the CHostEmulator class, whenever the control register write transaction receives the BEGIN_RESP phase from the Spacewire CODEC TLM model.

Figure 9: Control register write example

10.6 TRANSMITTER FIFO

The CODEC TLM transmitter FIFO is implemented as a byte array, the depth and width of which are user-configurable, via the CODEC configuration class. If the TLM packet level model is used, the FIFO depth must be set equal to the maximum Spacewire packet length size expected by the host system. The FIFO width should be set equal to 2, as each Spacewire character is modelled using a two-byte wide array, where byte[0] holds the character data/control flag and byte[1] holds the actual byte value.

For the purposes of the TLM model, the host system utilizes polling to read the transmitter FIFO full flag. Hence, the FIFO full flag is an addressable location accessible via TLM-2.0. Since both the transmitter FIFO and the relating full flag do not correspond to addressable locations in the RTL CODEC model, their addresses are user-configurable parameters included in the CODEC configuration class. The transmitter FIFO is a write-only target, whereas the full flag is a read-only target.

If the model user wishes to write data to the transmitter FIFO, he/she should primarily read the FIFO full flag, and if the FIFO is not full, write the SpaceWire character/packet structure pointer, depending on the model abstraction level. If the user attempts to write a FIFO entry when the FIFO is full, the write is not performed and the CODEC model returns a *TLM_COMMAND_ERROR_RESPONSE.* The following code snippet summarizes the procedure necessary to write a SpaceWire character to the transmitter FIFO. The m_spwChar variable is a 2-byte wide array holding the Spacewire character to be written to the transmission FIFO, whereas the m_dataWrittenEv SystemC event variable is only used in the AT model. The m_dataWrittenEv event is notified by the nb_transport_bw() method, whenever the BEGIN_RESP phase to the FIFO write transaction is received.

Figure 10: Transmitter FIFO write example

10.7 RECEIVER BUFFER

The TLM receiver buffer implementation follows the transmitter FIFO paradigm, i.e. it is implemented as a byte array of configurable depth and width, with the actual depth value selection to be dependent on the model abstraction level – exchange or packet . The host system utilizes polling to read the receiver buffer empty flag. Likewise the transmitter FIFO, the buffer empty flag is an addressable location accessible via TLM-2.0; both the receiver buffer empty flag and the receiver buffer addresses are user-configurable parameters, set via the CODEC configuration class. Furthermore, both are read-only targets.

The normal read receiver buffer procedure would be for the host system to read the empty flag and if the receiver buffer is not empty, read the receiver buffer contents. If the host system attempts to read a buffer entry when the buffer is entry, the read is not performed and the CODEC model returns a *TLM_COMMAND_ERROR_RESPONSE.*

10.8 READING/WRITING TIMECODES FROM/TO THE SPACEWIRE CODEC TLM

The CODEC TLM stores time codes received from the SpaceWire link to a member variable of the receiver block class CRxDecode. In the transmitter block, another member variable of the CTxEncode class holds the value of the time code that the host system has requested to be transmitted. Within the context of the CODEC TLM, both of these variables are accessible via TLM-2.0 by the host system. The time code received variable is read-only, whereas the time code to be transmitted is write-only. The length of each variable is equal to *sizeof(unsigned char)*, as the relati ng time code in/out ports of the RTL CODEC are 8 bits wide. Due to the fact that these variables do not correspond to actual RTL CODEC registers, their addresses can be configured via the CODEC configuration class.

As a result of the above, in order to read/write a time code, the model user must set the transaction length and streaming width attributes to *sizeof(unsigned char)*, the address to the time code “register” address and the command type to read/write, depending on whether the model user wishes to read a time code received, or write a time code to be transmitted. An example of writing a time code to be transmitted is supplied below. The TcodeDataLength constant is set equal to sizeof(unsigned char), whereas the m_tickWriteEv SystemC event varaible is used in the AT model. The m_tickWriteEv event is notified via the nb_transport_bw() method implementation of the CHostEmulator class, whenever the BEGIN_RESP phase to the time code write transaction is received.

Figure 11: Time code write example

11 TLM MODEL NETWORK-SIDE TRANSACTIONS

For the CODEC TLM on the SpaceWire link, a custom payload and protocol have been defined, utilizing the Greensockets library serial payload and protocol as the basis. The SpaceWire custom protocol defines a single phase for a link transaction for AT models, due to the fact that SpaceWire characters transmission cannot be disassembled into multiple phases. This approach is justified by the following arguments

- there is no acknowledgement that has to be received by the transmitter for a SpaceWire character sent,
- SpaceWire data characters transmission is interleaved with NULLs, FCTs and time codes transmission.

11.1 SPACEWIRE TRANSACTION EXECUTION FLOW

Likewise TLM-2.0 transactions on the host-side of the CODEC, once a transaction originating from the other end of the link is received, an initial check on its attributes is performed. Specifically, the parity and data control flag pointers of the SpaceWire payload are checked to examine if they are equal to NULL; thereafter the transmitter bit clock period field – used to emulate the recovery clock mechanism of the SpaceWire signal layer - is checked against zero. Transactions having this field set to zero are considered invalid. Additionally, the transaction length field is checked against zero, a zero length indicates an invalid transaction. If any of the above validity checks fails, the simulation is terminated issuing an error message informing the model user of the condition that lead to termination. It should be noted that there is a differentiation between the packet and exchange level models of the CODEC. At the exchange level, the disconnection error injection flag is first checked; if a disconnection error is injected in the transaction a SPW_ERROR_RESPONSE is sent back to the other end of the link and no further processing takes place. This is due to the fact that disconnection errors at the exchange level are injected using transactions that have just the disconnection error extension set, and carry no actual payload.

Assuming that the transaction is valid, it is further processed. If the transaction causes an error at the receiver – i.e. if an error injection flag is set at the packet level or if an error occurs at the exchange level - an SPW_ERROR_RESPONSE is sent back to the initiator and the link re-initialization procedure is initiated. It should be noted that in the packet level model, the packet that contains the error is still stored in the receiver buffer, but its EOP character is replaced by an EEP character to indicate the error to the host system. The aforementioned procedure is summarized in the following diagram – the dashed green line refers to the processing step applicable only at the exchange level.

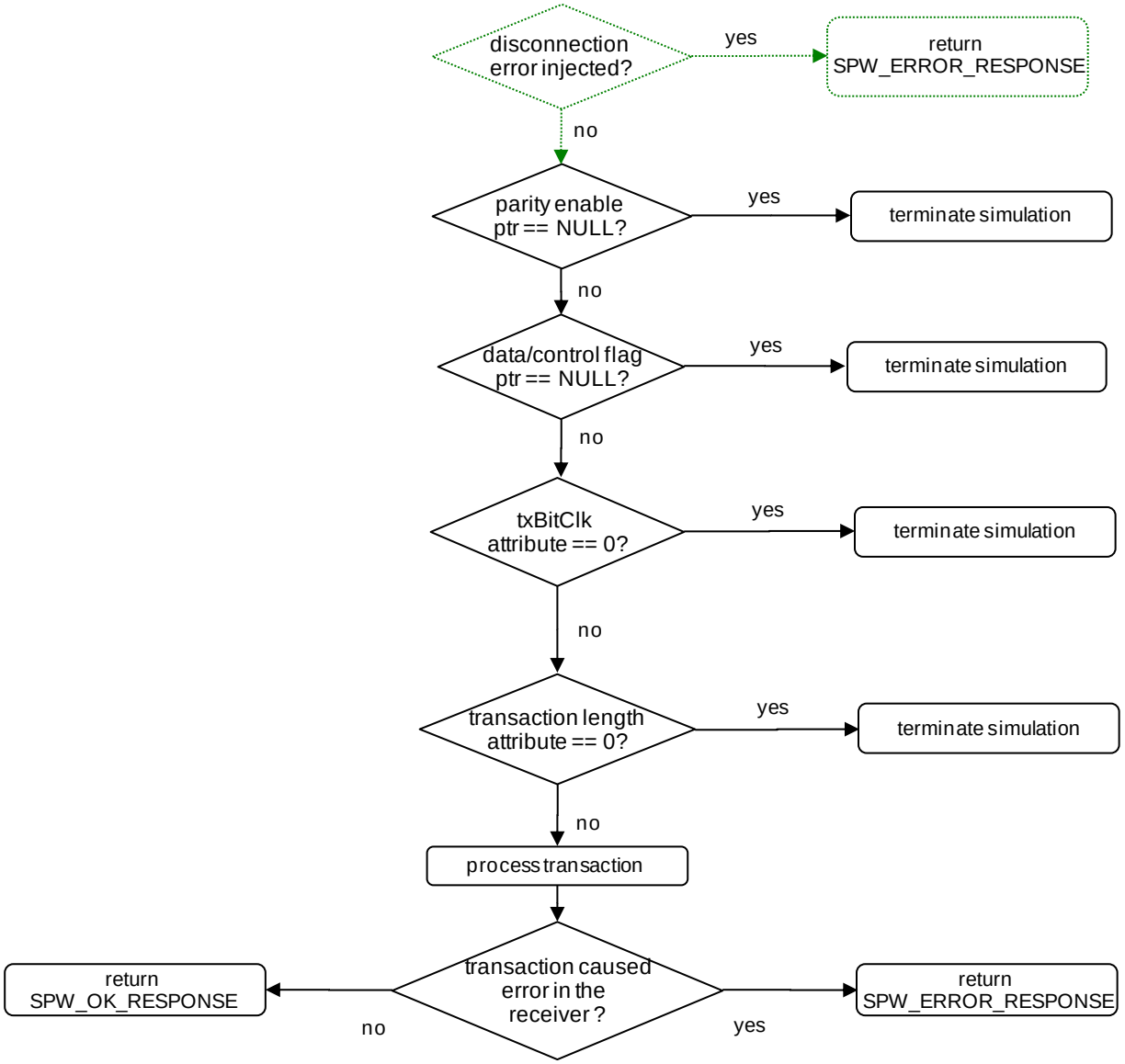

Figure 12: SpaceWire network-side transaction execution

11.2 NETWORK-SIDE TRANSACTIONS CODING STYLES SUPPORT

The SpaceWire custom payload can be used for the communication between the transmitter and the receiver through the blocking as well as the non-blocking interface. However, due to the fact that the non-blocking call realizes what is essentially a single-phase transaction, the differentiation between the LT and the AT coding style for network-side transactions is limited to the call of the *b_transport()* or *nb_transport_fw()* method. The timing diagrams for both the blocking and non-blocking calls on the forward path for the SpaceWire TLM CODEC are illustrated below.

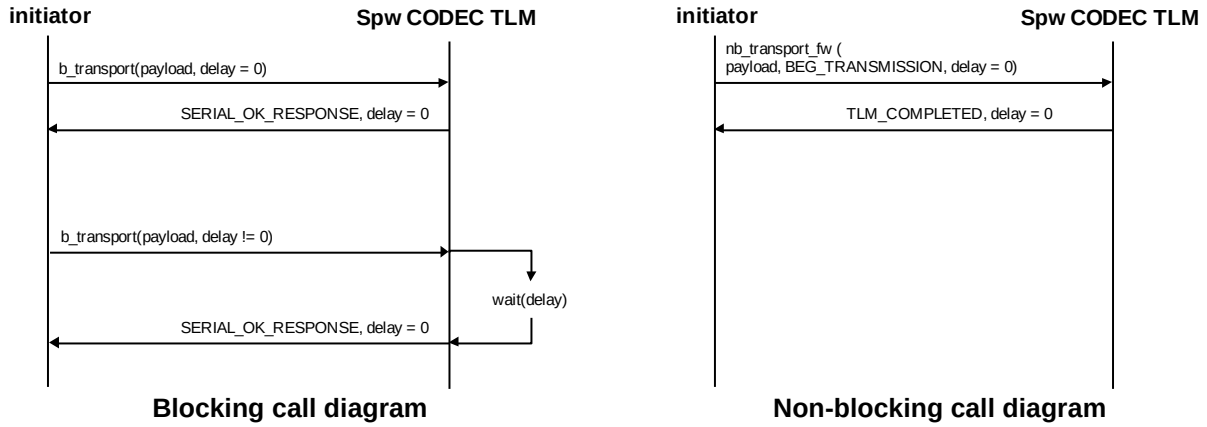

Figure 13: SpaceWire network-side transactions timing diagram

As can be seen in the diagram, non-blocking calls with a delay argument not equal to zero are supported, but require the initiator to call the wait() method after the blocking/non-blocking call to emulate the characters/packets transmission time. On the other hand, the implementation of the blocking interface in the CODEC TLM calls the wait() method if a transaction with a non-zero delay argument is received, thus disallowing temporal decoupling on the network-side. This is due to the fact that allowing temporal decoupling on the network side could lead to significant timing inaccuracy, well beyond the level accepted by a LT model. The issue with temporal decoupling is threefold. On the one hand, Spacewire characters reception also affects the execution of the link FSM. Consider the following example: if temporal decoupling is allowed and while the link is in the CONNECTING state, the CODEC receives an FCT with a delay annotation t, and in the next transaction annotated with delay t + 10*txBitClkPeriod the CODEC receives a transaction that generates a character sequence error - i.e. a transaction containing a time code or a data character. When the link FSM is executed, then while it will be in the CONNECTING state, it will find both the got FCT and the character sequence error flags set. Since the character sequence error flag has precedence over the got FCT, the link will go to the ERROR RESET state. However, this is not correct as by the time the character sequence error is generated, the link would have transited to the RUN state and the error would be ignored. On the other hand , the network-side temporal decoupling is effectively coupled with the rx buffer empty space where Spacewire characters are actually stored. Therefore, even if temporal decoupling was allowed, it would be limited by the receiver buffer space. Last but not least, the receiver buffer space affects the CODEC credit management, therefore allowing temporal decoupling would constitute the implementation of the Spacewire credit management mechanisms inapplicable.

In order for the model user to be able to perform a transaction with the CODEC TLM over the SpaceWire link, he/she has to declare a CTlmSpwPayload object, fill in the fields necessary, calculate the number of bits to be transmitted in order to be able to calculate the transmission time interval, call wait() with a delay argument equal to the transmission time interval - to emulate the transmission process - and finally perform the forward blocking/ non-blocking call. This process is depicted in the following figure.

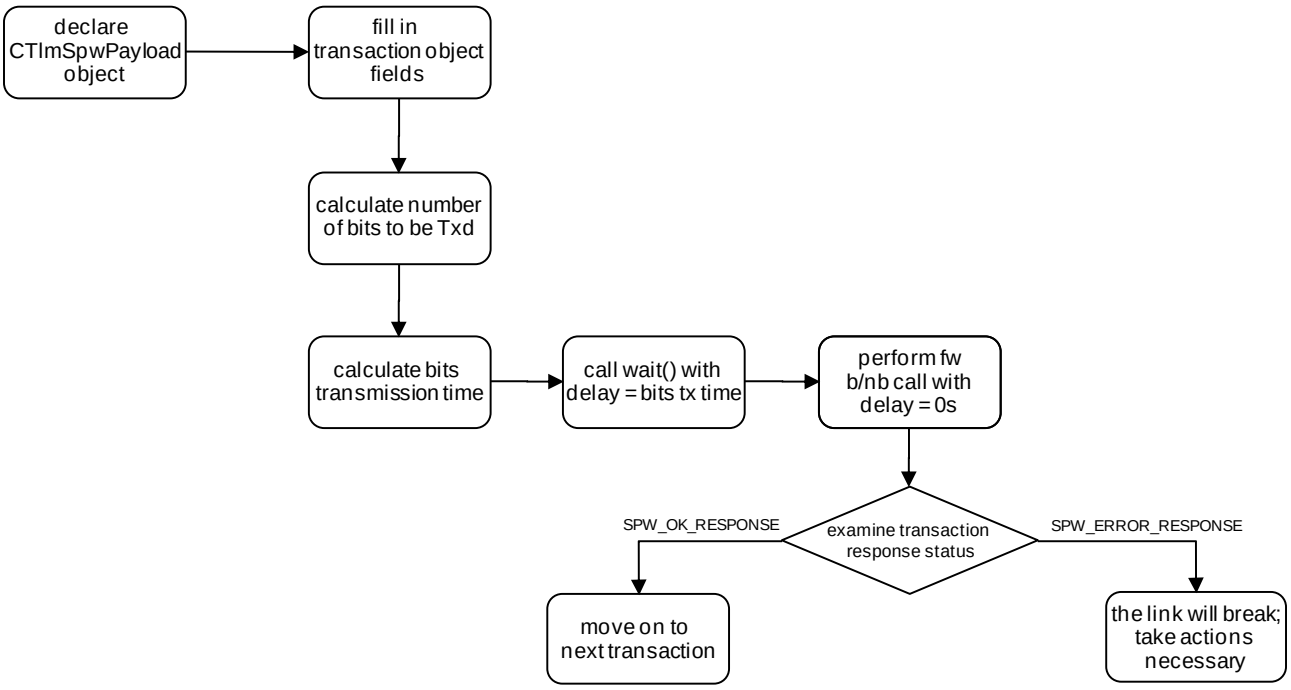

Figure 14: SpaceWire transaction execution flow

Once returning from the forward call, the initiator must examine the transaction response status to check if the transaction was successful. Upon successful completion, the response status should be set to SPW_OK_RESPONSE. In case there was an error encountered at the other end of the link, the response status is set to SPW_ERROR_RESPONSE. An SPW_ERROR_RESPONSE indicates that the other side of the link will reset its link FSM, thus leading to link re-initialization and indicating to the initiator that it should reset its link FSM as well.

For an analytical description of the custom SpaceWire payload and protocol, including the API provided to set/get individual transaction object fields, please refer to [6]. Example code of performing a transaction over the SpaceWire link – essentially implementing the steps described in the figure above - can be found in the CODEC transmitter block, i.e. CTxEncode.cpp.

12 CONCLUSIONS

The document constitutes the user manual of the SpaceWire TLM developed within DELTA. It presents an overview of the core structure, its configuration characteristics, and its interfaces , accompanied by code snippets to serve as coding guidelines. Furthermore, the methodology for compiling the model files, and building simulation binaries is described, both with respect to a native execution (i.e. executable for the host platform) and to a Modelsim based simulation (i.e. instantiation of the design within a dedicated simulator).

Appendix A.- DELTA SYSTEMC-ONLY TEST-BENCH DESCRIPTION

A.1. TEST-BENCH ARXCHITECTURE

The Delta Testbench comprises a total of four modules: a class representing the reference Host System, - CHostEmulatorTb, a class representing the reference Spacewire CODEC – CSpacewireTopTb which is a peripheral device of the reference host system, a class representing the Spacewire CODEC TLM, which is the system under test, and finally a class emulating the host system which uses the CODEC TLM as a peripheral device. Each host system is connected via a TLM-2.0 interface to the respective Spacewire CODEC, whereas the two Spacewire CODECs are connected back-to-back, using the custom Spacewire Serial interface developed for the purposes of the DELTA project.

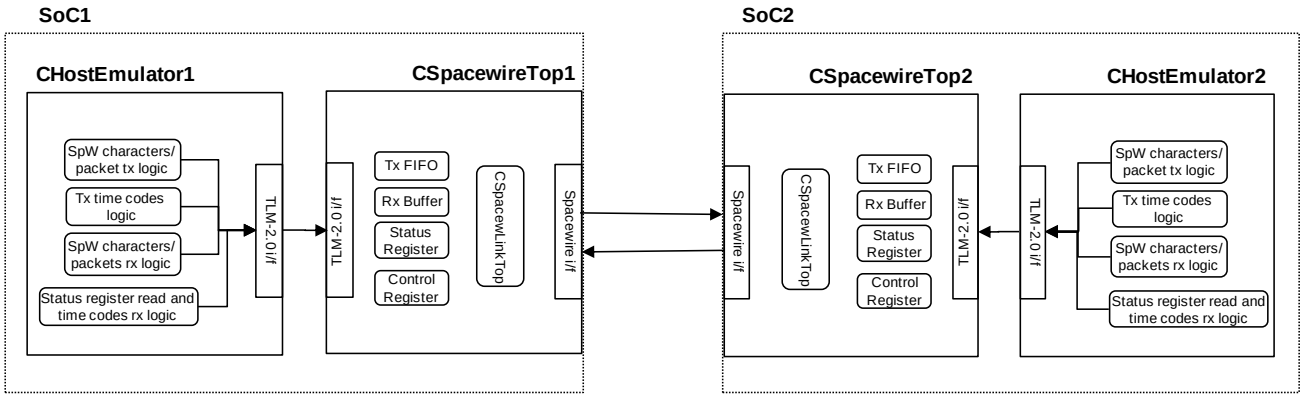

Figure 15: SystemC-only testbench topology

Each host system utilizes 4 SystemC processes to generate data for and receive data from the Spacewire CODEC model:

1. TxCharsTh(),
2. RxCharsTh(),
3. ReadStatusRegTh(),
4. TxTcodesTh()

The TxCharsTh() and TxTcodesTh() are the the host system traffic generators. On the one hand, the TxCharsTh() generates data characters or spacewire packets – depending on whether the test-bench is configured to operate at the exchange or the packet level and feeds them to the CODEC transmit FIFO, and on the other hand the TxTcodesTh() generates time codes and supplies them to the CODEC. On the reception part, the RxCharsTh() polls the CODEC receive buffer to check for data characters/spacewire packet pointers, and the ReadStatusRegTh() polls the CODEC status register. If it finds that there is a new time code received, which is indicated by the relating status register bit, it fetches the time code from the CODEC.

A.2. DATA GENERATION AND VERIFICATION

The host system traffic generator facility creates Spacewire packets of random length, based on a maximum packet length value configured by the test-bench user. The packet contents are created using a counter, where each Spacewire character is equal to the previous plus one. Due to the fact that Spacewire characters are modeled using two bytes, the first byte contains the data flag and is always equal to zero whereas the second byte is a number between 0 and 255 – unsigned chars are used to store the Spacewire character. The EOP character representation is identical to the one used at the RTL model, i.e. byte 0 equals to one and byte 1 equals to 0.

On the receive side, the data received is verified by using the data generation counter property. Specifically, the first character of a new packet is set as the base value of the packet characters received, and all characters that follow until the EOP character are expected to be equal to the previous character plus one. In case a Spacewire character received does not match the one expected, an informational message is issued and the test-bench is terminated.

A.3. HOST SYSTEM CONFIGURATION

The test-bench is configurable by means of the CODEC configuration structure CDeltaSpacewConfig, which defines operational parameters of each CODEC instance, and the host system configuration structure CHostEmulatorConfig, which define data generation and reception parameters for each host system emulator instance. The host system configuration parameters are summarized in the following table.

Parameter Name	Parameter Type	Parameter Purpose
ID	unsigned int	host system ID, used when printing log messages
module name	const char*	host module name
initiator socket name	const char*	name of the host system TLM initiator socket
status register polling interval	sc_time	defines the time interval between two successive status register read operations
tx FIFO polling interval	sc_time	defines the time interval between two successive CODEC Tx FIFO full flag read operations
rx buffer polling interval	sc_time	defines the time interval between two successive CODEC Rx buffer fifo empty flag read operations
tx packets number	unsigned int	the number of Spacewire packets the host system should transmit to the CODEC FIFO
tx timecodes number	unsigned int	the number of time codes the host system should transmit to the CODEC
tx timecode interval	sc_time	time interval between the transmission of two successive time codes to the CODEC
packet error injection interval	unsigned int	instructs the host to insert one error for every packet error injection interval packets transmitted. i.e. if this is set to k, then every k^{th} packet an error shall be injected to the packet. This parameter is valid for the packet level only.
maximum packet size	unsigned int	the maximum Spacewire packet length allowed when generating Spacewire packets
rx packets number	unsigned int	number of packets the host system should receive. Used as a test-bench termination condition and should be set equal to the number of packets the 2^{nd} host system will transmit.
rx timecodes number	unsigned int	number of time codes the host system should receive. Used as a test-bench termination condition and should be set equal to the number of packets the 2^{nd} host system will transmit.

Table 10: Host emulator configuration class parameters

Additionally, configurable observation points can be defined for the host emulator classes, relating to transactions and/or data processing functionality. The observation points may be defined via compilation directives, summarized in the table below.

Compilation Directive	Impact
Reference Host Emulator	
TB_HOST_EMULATOR_DATA_DEBUG	enables the print out of host system log messages relating to the transmit and receive operations
TB_HOST_EMULATOR_TRANS_DEBUG	enables the print out of host system log messages relating to the TLM transactions – messages originating from the b_transport, nb_transport_fw, and nb_transport_bw calls.
Host Emulator Under Test	
HOST_EMULATOR_DATA_DEBUG	enables the print out of host system log messages relating to the transmit and receive operations
HOST_EMULATOR_TRANS_DEBUG	enables the print out of host system log messages relating to the TLM transactions – messages originating from the b_transport, nb_transport_fw, and nb_transport_bw calls.

Table 11: Host emulator observation points

A.4. PACKET ERROR INJECTION MECHANISM

When the test-bench operates at the packet level, the test-bench user has the capability to define if he/she wishes to inject errors in the packets transmitted, as well as the frequency of the packets containing an error, via the packet error injection interval parameter. If the user wishes to not use packet error injection, this parameter should be set to zero, otherwise if set to k it forces the injection of a random error type in every k^{th} packet. The error type is selected between the 5 available error types of the packet level - credit error, parity error, character sequence error, escape sequence error and disconnection error. The random error type selection is ensured by calling the rand() method and performing a modulo 5 operation on the rand() return value.

A.5. TEST-BENCH OPERATION

Upon start-up the Spacewire data generation SystemC process – TxCharsTh – writes the control register, setting the RESET and LINK_START bits. The RESET bit is set to one to emulate the active low reset deassertion and allow for normal CODEC TLM operation, whereas the LINK_START bit enables the FSM to move in the STARTED state once it enters the READY state. Thereafter the process waits for the link to enter the RUN state. The ReadStatusReg() SystemC process polls the CODEC status registers, and once the relating byte is set, it notifies an event that enables both the time code generation process and the data generation process to start sending traffic to the Spacewire CODEC. On the host system transmit side to the CODEC, the TxCharsTh() polls the Tx FIFO full flag according to the time interval set by the test-bench user, reading its value using the TLM interface; if the FIFO is not full, it writes a Spacewire character/Spacewire packet pointer, utilizing the TLM forward blocking/non-blocking interface. The actual interface to be used, is indicated to the host system by the CODEC configuration structure, i.e. if the CODEC is configured to use the AT model, the host system will use the non-blocking forward interface. Similarly, the time codes generation process waits for the time interval configured by the user, and then sends a time code to the CODEC.

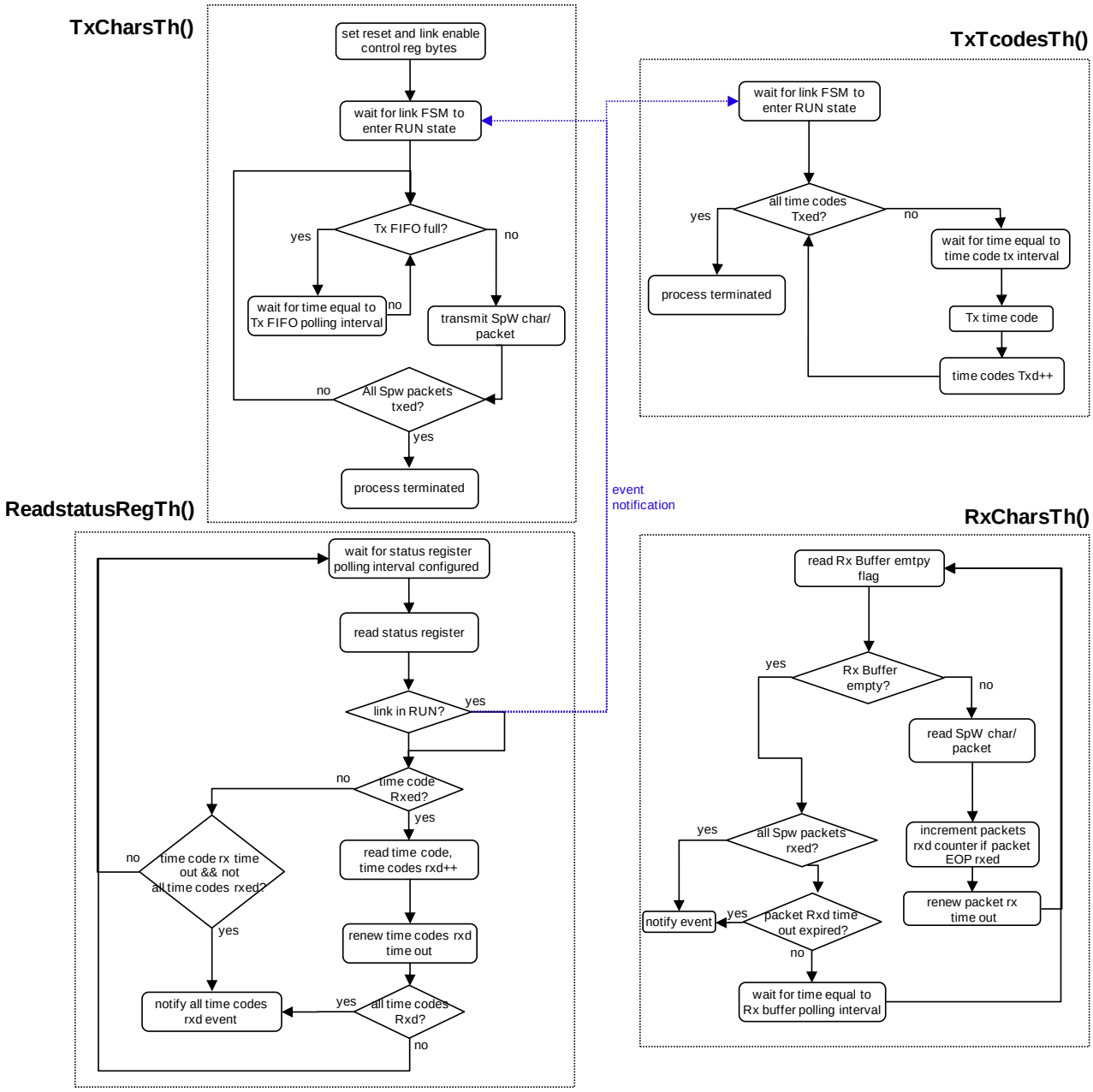

Figure 16: CHostEmulator processes and relating logic

On the host system receive side, the RxCharsTh() polls the receive buffer empty flag, reading its value using the TLM-2.0 interface. If the buffer is not empty, it fetches the Spacewire character/packet pointer structure from the buffer and verifies the packet contents. Additionally, the process polling the status register reads in the status register value according to the polling interval set by the test-bench user. In case the flag that indicates the reception of a new time code is set, the process also reads in the time code value.

A.6. TEST-BENCH TERMINATION

The test-bench termination is based on the packet numbers expected by the host system, which are configured by the test-bench user. Specifically, the CDeltaTestbench module which instantiates the host system emulation and the Spacewire CODEC objects, also registers a SystemC process which is triggered by events generated by the host system instances; one event is generated by the host system once it has received all packets it expects to receive. The test-bench SystemC process then examines the number of packets both host systems have received and if it matched the number of packets each host system expects, it terminates the simulation by calling sc_stop().

However, there are cases where all packets and time codes expected will not be received. Such cases are very likely to occur at the packet level: if the maximum packet size configured by the test-bench user is very large or the transmit clock period is too slow, the CODEC data transmission thread will have long wait() intervals. This, combined with a small time code transmission interval from the host side, will result in time code losses, as the 2nd time code shall be written to the CODEC before the 1st one has been transmitted. Furthermore, errors injected at the packet level cause the CODEC link FSMs to be reset and the link establishment process to be reinitialized. Therefore if the CODEC data transmission thread had called wait() to account for a packet transmission time and the link FSM is reset, this packet is discarded by the CODEC, thus emulating the disconnection error that would occur in the RTL level model.

In order to ensure the test-bench termination in such cases, conditionally compiled code has been included in the data generation processes of the host emulator classes, which forces the data generation processes to keep sending packets, even if they have completed the transmission of the number of Spacewire packets requested by the model user via the host configuration class. Specifically, if the model user plans to execute a simulation using the test-bench supplied in which erros are expected to be injected, he/she should compile the model with the

PACKET_LEVEL_WITH_ERRORS_SIM

compiler directive. The aforementioned directive enables the modification of the data generation process in the host emulator classes, so that they keep sending Spacewire packets to the CODEC TLM transmission FIFO, even if they have completed the transmission of the Spacewire packets requested.

A.7. DELTA TEST-BENCH SYSTEMC PROCESSES

The CDeltaTestbench class registers two process with the SystemC simulation kernel, the TestbM() and the TbProgressM() methods. The former is the one waiting the event notifications originating from the host system classes that indicate the completion of the reception or the expiration of the relating time outs. Once all four event are notified, the process signals the simulation termination by calling sc_stop().

The latter is a reporting utility which in effect denotes the test-bench progress by printing a log message that states for each host the number of time codes/Spacewire packets expected and the number of those actually received. The reporting period is currently set to 10us.

LIST OF ACRONYMS

API	Application Programming Interface
AT	Approximately-Timed
ASIC	Application-Specific Integrated Circuit
CA	Cycle-Accurate
CODEC	CODer DECoder
EDA	Electronic Design Automation
FCT	Flow Control Token
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
IDE	Integrated Development Environment
IP	Intellectual Property
LRM	Language Reference Manual
LT	Loosely-Timed
OSCI	Open SystemC Initiative
RTL	Register Transfer Level
SCV	SystemC Verification Library
SoC	System on Chip
TLM	Transaction Level Modelling

REFERENCES

[1] OSCI TLM-2.0 LANGUAGE REFERENCE MANUAL, Software version: TLM 2.0.1 Document version: JA32, July 2009

[2] Space engineering, SpaceWire – Links, nodes, routers and networks,

ECSS-E-ST-50-12C_31July2008

[3] SpaceWire CODEC IP - User Manual, Revision. 2.4, March 27, 2009

[4] SpaceWire CODEC IP VHDL Verification, Revision 2.4, April 1, 2009

[5] GRLIB IP Core User's Manual, Version 1.0.22, April 2010

[6] SpaceWire CODEC TLM – Development Manual