



SpaceWire

Design of the SystemC model of the SpaceWire-b CODEC

Dr. Nikos Mouratidis
Qualtek Sprl.
36 Avenue Gabriel Emile Lebon
B-1160, Brussels, Belgium
19/09/2011



- **Activity objectives:**

- High-level modeling of the ESA SpaceWire using SystemC TLM 2.0
 - SystemC  Simulation speed
 - TLM 2.0  Interoperability
- Functional validation and timing accuracy analysis of said model
 - Model functions identical to RTL IP core
 - Reported timing is useful for system performance evaluation

- **Intended uses of activity outputs:**

- Allow ESA to develop new VPs based on the available SystemC IP Models and distribute them to contractors without being subject to any fee or restriction, allowing software development before SoC hardware is ready
- ESA will license the SystemC IP Models without being subject to any fee or restriction and allow contractors to develop new VPs using the guidelines in the model documentation and perform design space exploration for future SoCs to be implemented either as ASICs or on FPGA

Project Overview

- Why is the model of an existing IP necessary?
 - Royalty-free access
 - Development of VPs prior to commitment / licensing
 - Actual IP licensed only after grounds for commitment
 - Licensing process does not stall development
 - Simulation speed
 - x10s to x1000s improvement over RTL simulation
 - Simulation kernel constitutes part of model
 - Linked library
 - No specialized simulation tools necessary
 - Interoperability
 - Compliant models utilise standardised interface
 - TLM 2.0 implements interoperability layer
 - Model may be directly used in compliant platforms
 - No interface adaptations necessary
 - Different use cases demand different level of detail
 - Software development – LT (b_transport)
 - Design space exploration – AT (nb_transport)

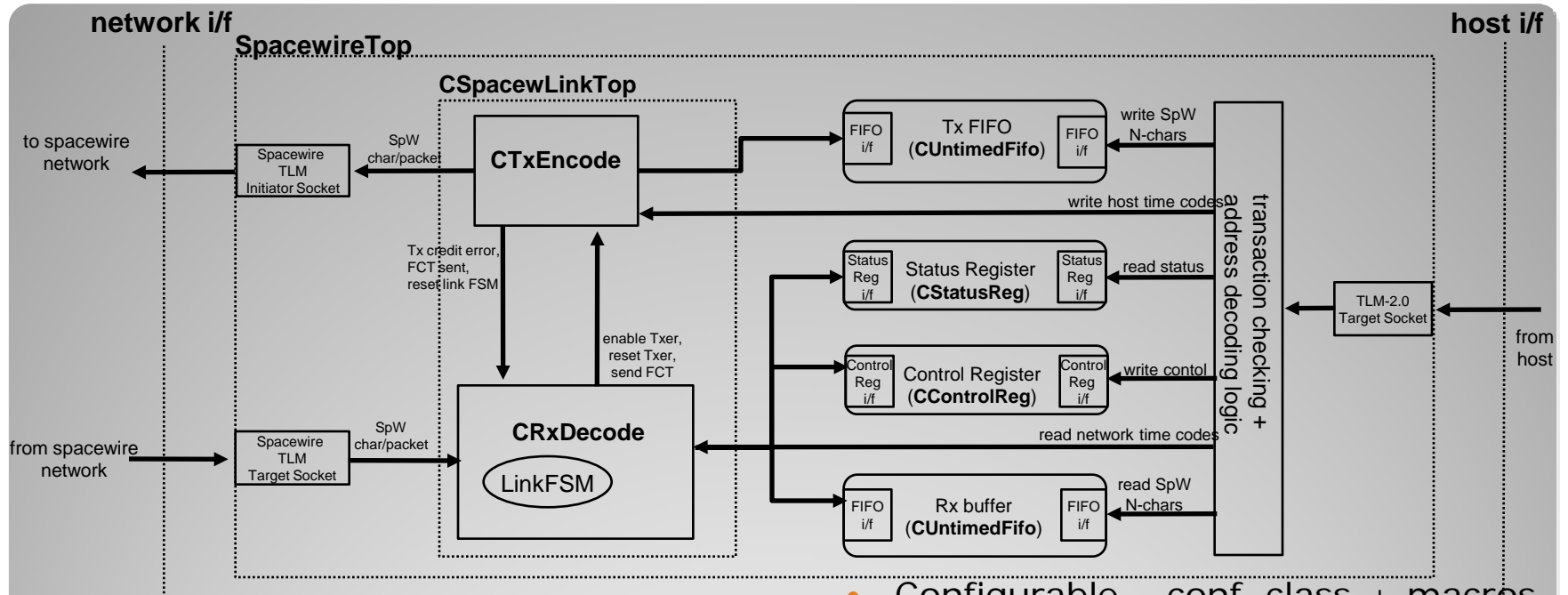
Motivation

- Point-to-point data connection
 - Lightweight protocols (w.r.t to alternatives)
 - Low-latency data transfer (minimal buffering)
 - Flow controlled (data loss avoidance)
- low-complexity technology for building scalable, fault tolerant networks
 - using routing switches connected by point-to-point links
- Aims at equipment compatibility and reuse
- High speed data link
 - Minimum speed is ~2Mb/s, maximum 400Mb/s
- Widely adopted

SpaceWire standard*



*Available at <http://www.ecss.nl>




- CODEC to manage connection and data flow across SpW link
- L-chars (link) and N-chars (normal)
 - L-chars - flow control
 - N-chars - information (data, EOP, timecodes, etc.)
- Configurable – conf. class + macros
 - Pipelined / non pipelined
 - DDR or SDR outputs
 - Transmission clk configuration options
 - Configurable receive buffer size
 - Discard empty packets
 - Reserve timecodes when link inactive
 - Observation points, module names, addresses

SpaceWire IP overview

- Timing report target examples ($\pm 20\%$)
 - Tx 1024 bytes@200mbits, 1024 byte packet: 51.22 us (159.937 Mbps effective rate)
 - Tx 1024 bytes@200mbits, 1 byte packet: 103 ns (114.286 Mbps effective rate)
 - Delay between ErrorReset and ErrorWait: 6.4 us
 - Delay between ErrorWait and Ready: 12.8 us
- Several times faster on same machine
- Internal accuracy
 - Character level
- Interface accuracy
 - LT: transfers at the packet level \Rightarrow no flow control
 - AT: transfers at the character level \Rightarrow all control in place

Implementation constraints

- Dual coding style targets - Interfaces at two different accuracy levels implemented (AT and LT)
- Dual data abstraction layers – Exchange (Character) and packet level
- General Modelling Style – TLM 2.0
 - TLM Interfaces – Base protocol/generic payload host side, custom net side
- Functional compliance (TLM ↔ RTL)
 - Same input  same output
- Timing accuracy - Max Divergence from RTL: 7% (vs. 20% target)
 - Timing behaviour – Matched to RTL (AT, exchange level)
- Simulation Performance - Assessed against RTL IP
 - Up to 1000s times faster than RTL
- Scalability
 - Memory footprint / CPU load - minor increase when multiply instantiating
- Self containment – Complete package delivered, no 3rd party deps.
- CAD tool independence
 - gcc tested on Linux, cygwin
 - Build system – cmake based

Implementation characteristics

- Separate computation and communication
 - Communication class
 - TLM 2.0 forward/backward interface
 - Read and write wrappers for computation callbacks
 - Computation class
 - Data operations
- Definition of four base classes
 - Multiply inherited by most blocks
- Inter-process communication using callbacks
- Data represented as arrays of bytes
 - Assure compatibility with any host, regardless of endianness
- Selection through constructor argument
 - Coding style
 - Interface data abstraction level

Model structure

- Timing annotations instead of explicit timing
 - Communicating processes exchange timing related information instead of actually delaying simulation
 - WAIT statements are aggregated, thus avoiding multiple context switches
- Use of callbacks rather than events
- Temporal decoupling
 - Individual SystemC processes are permitted to run ahead without actually advancing simulation time until they reach the point when they need to synchronize with the rest of the system
 - SystemC processes run ahead of simulation time for an amount of time known as the time quantum
- Simulation control
 - Details like transaction tracing and value recording are done through SystemC coding
 - Unlike vendor specific simulation tools that use script languages and GUI commands to control simulation details

Implementation decisions - I

- Runtime vs Compile-time configuration
 - Central point of configuration for the whole model, giving the end user the option to select simulation parameters, even the data abstraction level (packet vs character), and the coding style (LT vs AT). These features would typically be implemented in #ifdef statements.
- LT/AT switching
 - Style switching during model execution
 - Compliance checking for communicating instances
 - Loosely Timed – helps in advancing simulation faster during sections not important for particular use case
 - Approximately Timed – provides increased timing detail and data accuracy, but slows simulation down
 - LT -> AT - Switching has to delay for the length of the maximum 'wait' within all outstanding b_transport calls
 - AT -> LT - If outstanding nb_transport calls exist, issuing of b_transport calls is stalled until every single nb_transport completes

Implementation decisions - II

- AT for accuracy + LT for speed in same model
 - Independent Character or Packet level abstraction
- BUT
- Partial model switch
 - Host side fixed at character level
 - Network side switchable between character and packet levels
 - Run-time switch of network side possible
 - 4 possible combinations
 - Character/packet – AT/LT

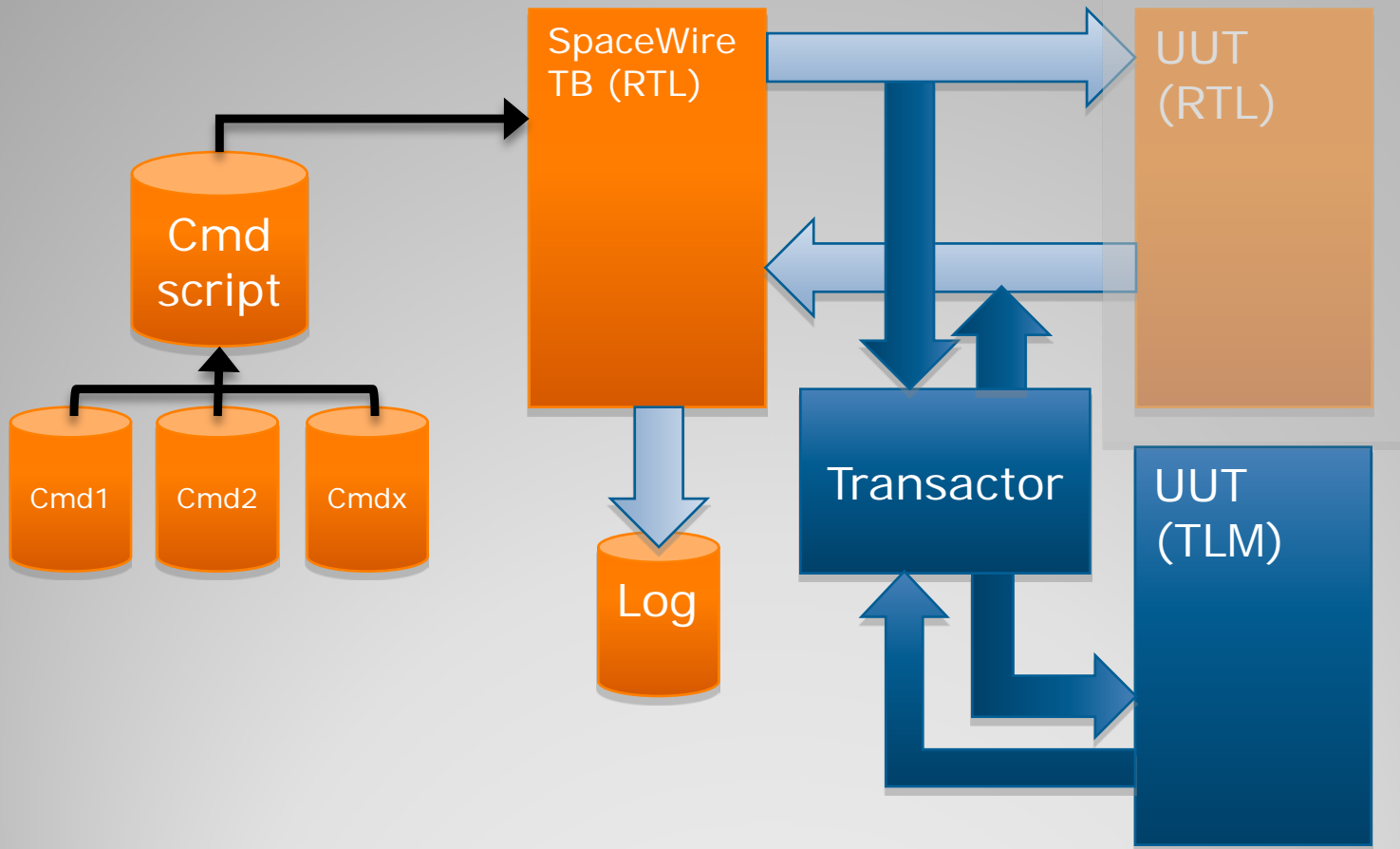
Abstraction levels combination

- TBV - Transactor Based Verification
 - Original testbench used - Mixed RTL-TLM simulation performing the complete set of testbenches supplied with RTL
 - TLM model interfaced through transactors
 - Automated verification mechanisms from RTL maintained to great extent
 - Satisfy assertions of testbench
 - Possibility to adopt incremental block replacement using some transactors internally
 - Variable and Transaction Recording
 - Values of a variable across time recorded using value-change callback functions
 - Recording timing information and attribute information associated with transactions
- SystemC only vs RTL only

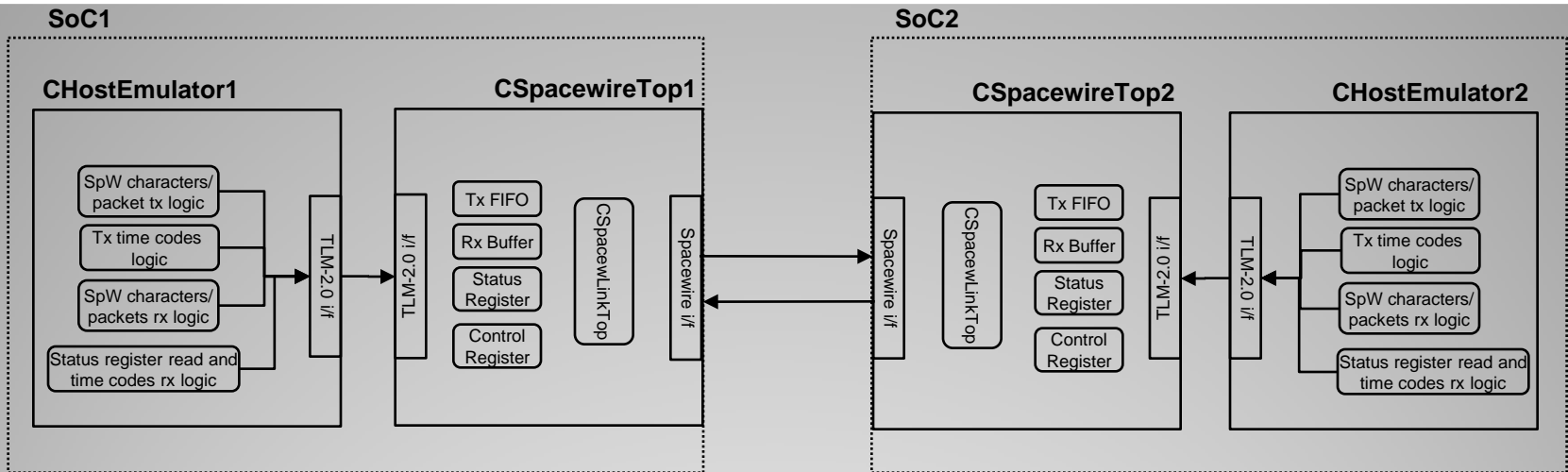
Verification strategy

- Compare TLM and VHDL results using
 - Varying number of fixed-size packets
 - Fixed number of varying size packets
- Simulation time logged in both models
 - Timing accuracy assessed
- Computer time logged in both models
 - Speed of execution assessed
- Single testbench to run multiple scenarios in accordance to configuration

Timing and speed comparisons



Testbench – re-use method



- Completely tests the TLM implementation
 - Normal functionality, corner cases, handling of abnormal situations
- Allows direct speed comparisons
- Each host utilizes 4 SystemC processes to generate data for and receive data from the SpaceWire CODEC model:
 - TxCharsTh()
 - RxCharsTh()
 - ReadStatusRegTh()
 - TxTcodesTh()

SystemC only verification

Simulation duration and time

SpaceWire CODEC Model (400 packets, 10K characters, 40 timecodes)	Simulation Duration (seconds)	Simulation Time (nanoseconds)
RTL	2824	4223644493
AT-exchange level	84	4222377400
LT-exchange level	65	4221724000
AT-packet level	3	4002783700
LT-packet level	2	4221724000

Performance improvement

SpaceWire CODEC Model	Speedup ratio
AT-exchange level	1:33
LT-exchange level	1:43
AT-packet level	1:941
LT-packet level	1:1412

Timing accuracy

SpaceWire CODEC Model	Timing accuracy (%)
AT-exchange level	99.97
LT-exchange level	99.95
AT-packet level	94.77
LT-packet level	99.90

Results

- Protocol constraints dominate implementation choices
 - Flow control in packet level - Error injection mechanisms
 - Phase decomposition in AT – Single phase transactions
- Custom interface on network side (vs. objective)
 - SpW not appropriate for base protocol/generic payload
- Simulation speed improvement (vs. RTL)
 - Iterative code optimisations to attain performance targets
- Dual data abstraction levels and combinations
 - Dual implementation in several places due to:
 - Differences in data handling because of lack of mechanisms (e.g. flow control)
 - Independent to coding style to allow combinations
 - Optimisations to allow streamlined use
- Transactor complexity
 - Interaction between SystemC model and transactor processes in order to produce the least amount of event notifications possible

Encountered issues

- Modeling task has to be finely focused
 - Solid understanding of implementation subject
 - Well defined interfaces
 - Good knowledge of system timing
- Adopt an architecture suitable for a stepped implementation approach
 - Allow for additional details from preliminary spec
- Compromises made have to be fully assessed
 - Timing inaccuracies considered a priori and in retrospect
- Mixed-language models (SystemC + VHDL or Verilog)
 - Pose diversions from pure scenario

Experience gained

- Keep implementation of building blocks as simple as possible
 - Avoid breaking up functionality in many functions
 - Function calls reduce overall model execution speed
- Use module member variables in place of local variables within functions
 - Local variables incur performance penalty - constructed every time the function is entered
- Use callbacks and reduce events usage
 - Callbacks used in place of event notifications for inter-process communication incur lower delays and increase execution speed
- Dual coding style/double data abstraction
 - Complete isolation of data and communication
- Transactor complexity
 - Attributed to complexity of RTL TB
 - I/F signal-by-signal approach

Lessons learned



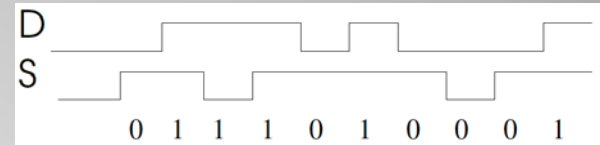
Thank you

Backup slides

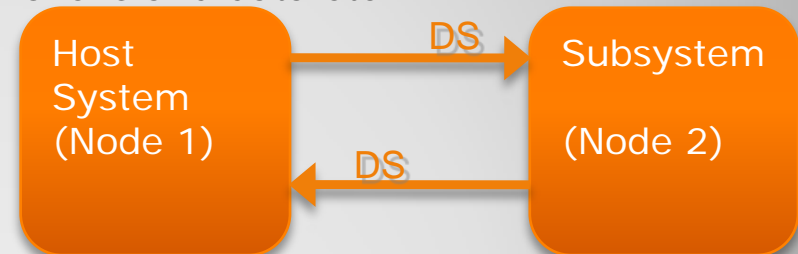
- Tradeoffs: accuracy vs speed
 - Data abstraction
 - Bit level avoided
 - Communication abstraction
 - Temporal decoupling
- Tradeoffs: code elegance vs speed
 - Macros instead of configuration variables
- Tool peculiarities
 - Mixed language simulation constrains TLM
 - Conditional defines in code
 - Catering for tested compilers
 - Compiler option alternatives in makefiles
 - Code adapted when compilation problems existed

Aspects of implementation

- SpaceWire-b (SpW-b) CODEC
 - compliant with SpaceWire standard
 - serial transmitter/receiver
 - full-duplex, bidirectional, point-to-point data link
- Data Tx via 2 (pairs of) wires using LVDS
 - The clock can be recovered as (D xor S)
 - Phase-locked-loops are not needed
 - Accurate control of clock frequencies not needed
 - Link failure detected by loss of the derived clock signal



- Operation:
 - a SpaceWire node sends Tokens to the node on the other end of the link
 - Each Token that a node sends indicates to the receiving node that the sender has 8 bytes of available buffer space
 - Three tokens would indicate there are 24 bytes of available space in the Host System buffer
 - Both nodes are senders and receivers of Tokens and data etc.



SpaceWire CODEC

- Initiative announced 12 years ago
 - 27-9-1999 Open SystemC announcement
 - Originates in Scenic programming language
 - SpecC was chief competitor
- Open SystemC Initiative (OSCI) formed 2000
- TLM 1.0 standard released 2005
- SystemC v2.2 released 2007
- TLM-2.0 LRM, TLM-2.0.1 lib. released 2009
- SystemC AMS extensions 1.0 LRM 2010 released

A bit of history

- Replacement of FCT sending method with corresponding callback
- Optimisation to the “Begin Response” method
- Modification to behavior of control and status registers
 - Eliminated polling
- Reversal of notification between TX/RX methods and FIFO
 - Polling essentially becomes event driven mechanism
- In address decoding: if()-else() replaced by case

Performance improving modifications

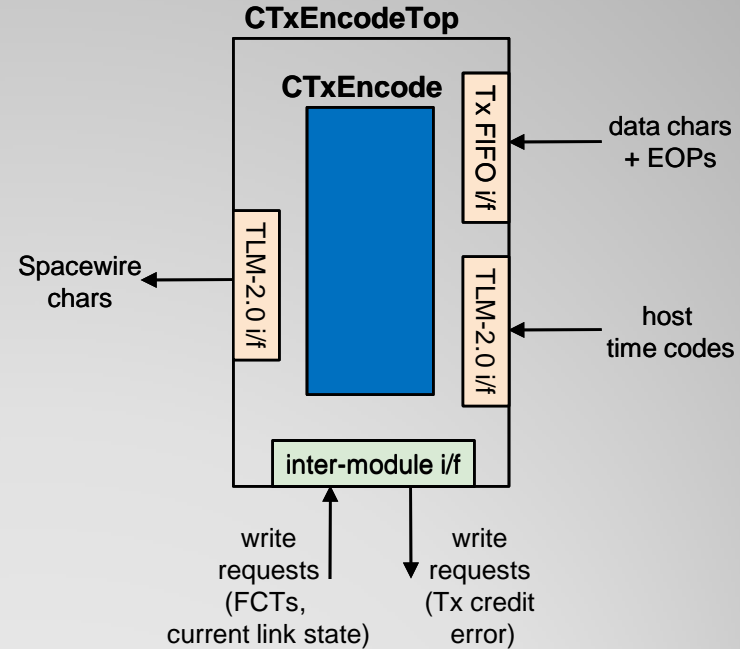
- RTL core provides meticulous testbench
 - Re-use methodology desirable
 - Interface (e.g. transactors)
 - Timing – satisfy assertions of testbench
- Verification coverage
 - $\geq 90\%$ of code

Verification challenges

- Build platform
 - SystemC 2.2.0
 - TLM 2.0.1
 - SCV 1.0e
 - GCC version $\geq 4.3.4$
 - Cmake version $\geq 2.8.1$
- Waveform viewer
 - GTKWave

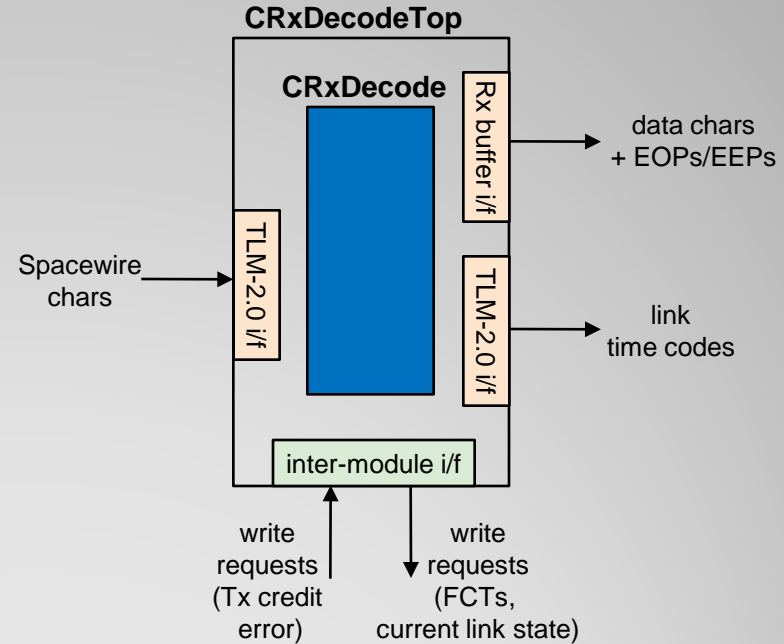
Tools

- Target for the host side
- Initiator for the network side
- Read data from Tx FIFO
- Credit counter maintenance
 - Credit Error indication
- Execution thread
 - CTxEncodeInitTh()



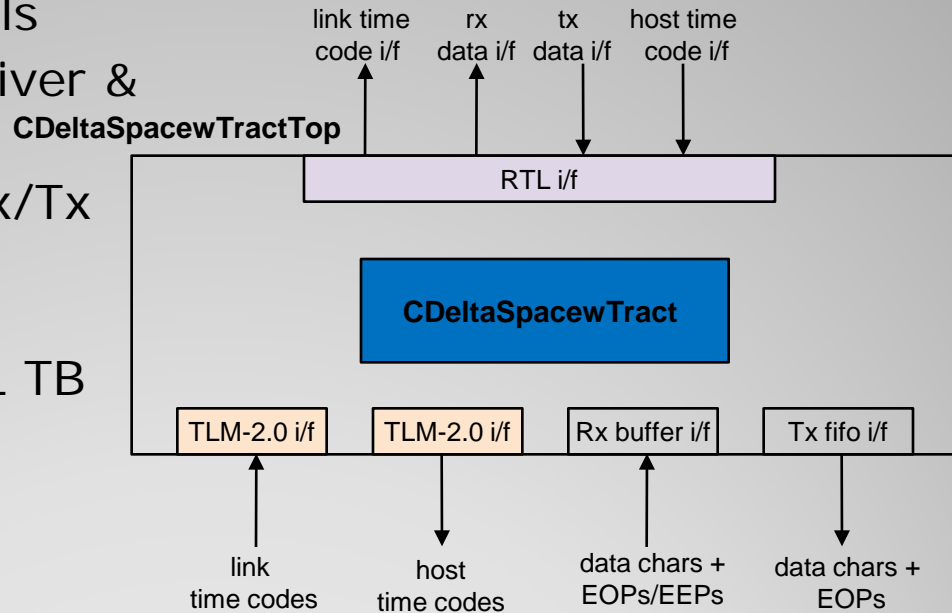
Transmitter

- Process target transactions
- Extract characters
 - Store into Rx buffer
- Process time codes
- Maintain flow control
 - Request FCT transmission
- Realise LinkFSM
- Execution thread
 - RxDataTh()

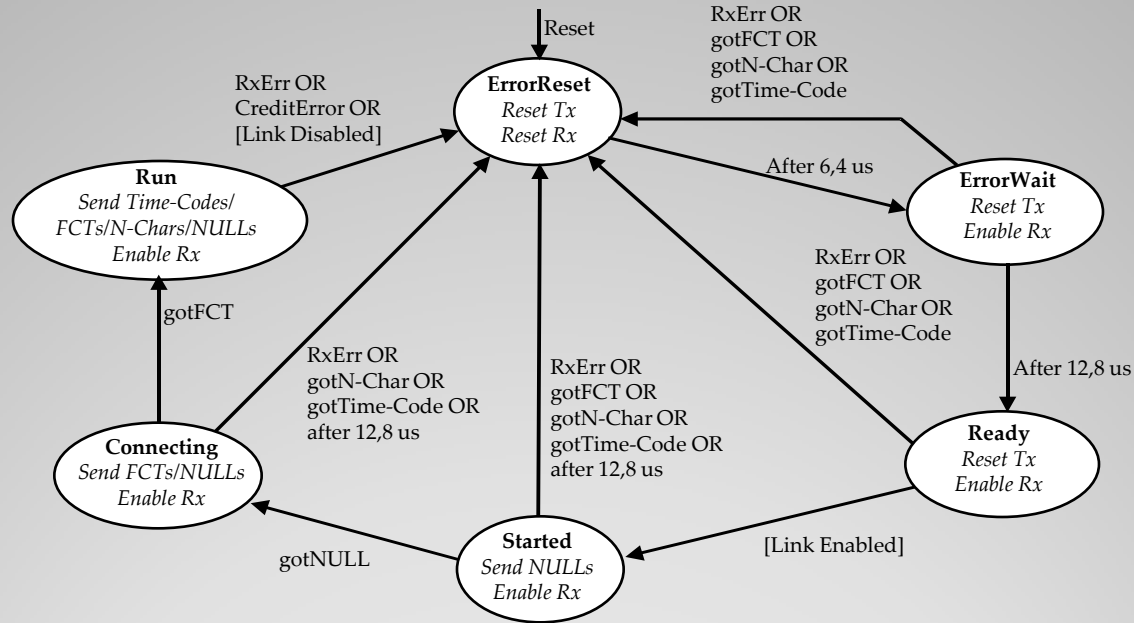


Receiver

- TLM interface for VHDL TB
- Translate between transactions and signals
- TLM interface for Receiver & Transmitter
- Custom interface to Rx/Tx buffers
 - To be converted to TLM
- RTL interface for VHDL TB



Transactor



LinkFSM