The SpaceWire CODEC
International SpaceWire Seminar (ISWS 2003)
4-5 November 2003, ESTEC Noordwijk, The Netherlands

Chris McClements [(1)], Steve Parkes [(1)], Agustin Leon [(2)]

[(1)]University of Dundee, Applied Computing, Dundee, DD1 4HN, Scotland, UK.
Phone: +44 1382 348837, +44 1382 345194, Fax: +44 1382 348838
Email: cmclements@computing.dundee.ac.uk, sparkes@computing.dundee.ac.uk.

[(2)]European Space Agency

ABSTRACT

A SpaceWire system comprises several units connected together with SpaceWire links, either directly or indirectly via one or more SpaceWire routers. The SpaceWire links are high-speed, bi-directional, point-to-point communication links operating at a baud rate of between 2 and 400 Mbits/s. SpaceWire runs over a cable containing four twisted pairs. At each end of a SpaceWire link is a coder/decoder (CODEC) which encodes packets of data to be transmitted into a serial bit-stream and decodes an incoming serial bit-stream into a data packets. The serialised data is encoded using a data-strobe technique where the strobe changes state at a bit interval whenever the data remains constant. This allows simple clock recovery in the receiver by XORing the data and strobe signals together and provides better skew tolerance than data-clock encoding. The data-strobe signals are transmitted using low-voltage differential signalling (LVDS). SpaceWire CODECs are used in both nodes and routers and consequently form an important element of any SpaceWire system.

Recognising the key role of the SpaceWire CODEC in future on-board data-handing systems ESA contracted the University of Dundee, Austrian Aerospace and Astrium to develop a high-performance SpaceWire (CODEC) in VHDL. Dundee were responsible for the design and implementation of the CODEC. The resulting VHDL code has been extensively tested by Austrian Aerospace and ESA. It is expected to operate at a maximum data rate of 200 Mbits/s when implemented in a suitable radiation tolerant ASIC technology.

INTRODUCTION

The SpaceWire system is based on nodes connected together indirectly through routers or directly node to node via SpaceWire links. The role of a SpaceWire CODEC in a system is the physical device which encodes and decodes the serial bit-stream over the SpaceWire links. The CODEC described in this paper is a high speed serial transmitter/receiver implementation of the SpaceWire CODEC compliant with the ECSS-50-12A SpaceWire standard [1]. The CODEC is implemented in technology independent RTL VHDL code. The aims of the VHDL code implementation are re-usability, technology independence, high-speed operation, low area footprint and configurability. The VHDL design was not approached as a one size fits all implementation, rather a configurable solution was implemented to allow the user maximum control over the final gate level representation of the CODEC.

This paper outlines the SpaceWire CODEC function and interfaces. An overview of the CODEC architecture and the function of the internal logic blocks is presented. The CODEC input and output interfaces are presented, and the configuration interface is discussed also. A brief overview of the verification strategy is then presented.

ARCHITECTURE OVERVIEW

The CODEC comprises of a transmitter, receiver and an initialisation state machine as shown in “Fig 1”. Data packets and timecodes are accepted from a host system through parallel interfaces to the transmitter. The data is serialized in the transmitter and encoded through the SpaceWire link using DS (Data-Strobe) encoding [1] [2]. Data is only transmitted when the CODECs at each end of the SpaceWire link have reserved buffer space using the SpaceWire flow control mechanism. When space for eight more data characters in the receive buffer is available then a FCT (Flow Control Token) is transmitted to allow eight data characters to be transferred. In this way buffer overruns are detected as a link error by the CODEC. The serial data-strobe encoded bit-stream is received, error checked and decoded into SpaceWire characters in the receiver. Received data characters which were requested using FCTs are placed in the receiver buffer and timecodes are output directly to the host system. Link control functions are implemented by the initialisation state machine, in particular the link can be started, disabled or left to auto-start on reception of an input bit-stream.

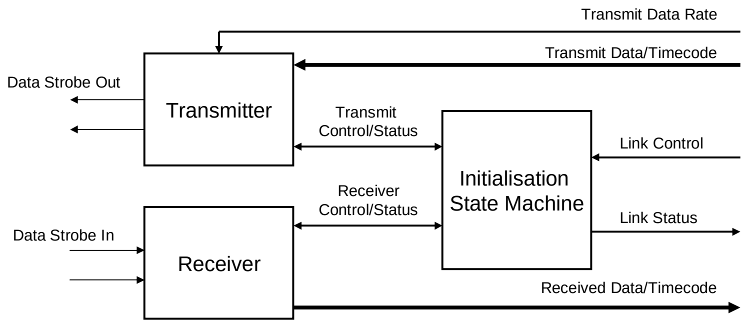

Fig 1 SpaceWire CODEC Architecture Overview

The error recovery scheme described in the SpaceWire standard [1] is implemented internally in the CODEC. Errors detected in the CODEC include incorrect parity bit received, escape character sequence error, receiver disconnection error, receiver credit error (data character received when not expected), transmit credit overflow and character sequence error at startup. On error detection the initialisation state machine disables the transmitter and receiver therefore causing a disconnection and error recovery at the other end of the SpaceWire link.

When an error is detected the SpaceWire link is disconnected and the tail of the packet currently being received is assumed to be lost. To preserve the packet structure used in SpaceWire systems an EEP (error end of packet) character is added to the tail of the packet. This indicates the packet was partly received but an error occurred before the normal EOP (end of packet) marker was received. As the SpaceWire link has been disconnected then the FCT characters which were transferred to the other end of the link, denoting receive buffer space, are assumed to be invalid and the internal receiver FCT pointers are updated to free any reserved buffer space. The transmitter error recovery scheme clears the tail of the packet being transmitted when the error was detected from the transmitter FIFO. Therefore when the next link start-up and connection occurs the next data character to be transmitted is the head of the next SpaceWire packet. In this way the error recovery system is transparent to the host system which needs only to perform the SpaceWire packet level protocol.

Transmit FIFO and receive FIFO are not implemented internally in the CODEC as they are typically implementation technology dependent structures. The CODEC implements the most common synchronous FIFO interface for the transmitter, using an empty flag and read enable signal to read data into the transmitter. The receive FCT operations are implemented internally by the CODEC using internal receive buffer read and write pointers and an internal FCT pointer. The receive buffer pointers can be used to address an external buffer space with synchronous flags and write enable signals output by the receiver.

SERIAL INTERFACE

The SpaceWire CODEC uses the IEEE 1355 DS-DE [2] (Data-strobe, Differential Ended) encoding method to serialize SpaceWire characters for communication over the SpaceWire link.

Data Strobe Encoding

In data-strobe encoding two signals are used in each direction, data and strobe, with four cable wires are required in each direction due to the differential signaling method used. When transmitting, at each bit interval the strobe signal retains its value when data changes and in turn strobe changes when data retains its value. In this way the receiver can recover the transmitter clock by an exclusive-or operation of data and strobe signals. The recovered clock is half the transmitter bit-rate therefore rising and falling edges of the recovered clock are used to sample the data input. This relationship can be seen in “Fig 2”.

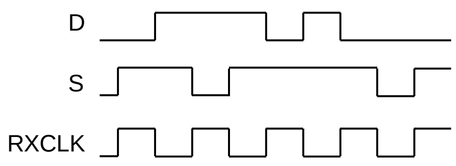

Fig 2 Receiver clock recovery

Single Data Rate (SDR) and Double (or Dual) Data Rate (DDR) bit-stream encoding

The SpaceWire CODEC allows the user to select SDR or DDR encoding for the transmitted bit-stream. The term SDR applies to an encoding method whereby one bit of data is output for each transmit clock period. SDR encoding is the simplest method where the data and strobe outputs of the transmitter are connected directly to the output buffers. DDR encoding outputs two bits of serial data for each transmit clock period, one bit which will be output on the rising edge of the clock and one bit which will be output on the falling edge. The serial bit-rate is double the transmitter clock period rate therefore reducing power consumption in the transmitter by half and reducing logic placement and routing effort on the target technology. DDR encoding requires extra complexity at the output buffers as the two bits of data available on each transmit clock period must be multiplexed onto the output pin in such a way as to avoid glitches occurring at the outputs. Depending on the users target technology this may be implemented as a logic function or as a technology specific DDR output buffer.

CONFIGURATION INTERFACE

The ability to configure the CODEC to suit the user's application was a prominent goal in the design. A number of constants are defined in an accompanying top level VHDL package, each with a specific global or low-level effect on the final gate level circuit output from the user's synthesis tool. For each implementation of the CODEC the user should set the configuration constants dependent on the requirements of the SpaceWire system and the constraints of the target technology.

Pipelining Configuration

In a digital system the amount of logic between flip-flops defines the minimum period of the clock and therefore the maximum clock frequency. With pipelining extra flip-flops are inserted into the design to decrease the minimum period and in the case of the transmitter allow a greater serial bit rate to be transmitted. Adding pipelining flip-flops increases the area footprint and adds one cycle of latency to the CODEC when transmitting and receiving.

"Table 1" shows, as an example, the area footprints and the expected bit-rates when synthesized for a Xilinx Virtex-E device in the '-6', slowest, speed grade. The CODEC was synthesized using a standard configuration with an independent transmit clock and a 32 byte receive buffer size. The transmit and receive clocks were not constrained and default analysis was performed by the Xilinx "trce" tool. Typically for a high speed transmitter then the pipelined DDR configuration will be chosen.

The bit-rates given in "table 1" should not be considered as the maximum bit-rates achievable by the CODEC as the clocks were not constrained. Rather they are given as a comparison between the pipelined and non-pipelined configurations.

Table 1 Area Footprint and Bit Rate Pipelining Comparison

Type	Area footprint		Max Transmit Rate		Max Input Bit Rate, Receive Clock*2
	Flip-flops	LUTs	SDR	DDR	
Non-Pipelined	257	352	93 Mbits/s	186 Mbits/s	285 Mbits/s
Pipelined	267	354	141 Mbits/s	282 Mbits/s	345 Mbits/s

Transmit Clock Configuration

The transmit clock frequency determines the maximum bit rate achievable by the CODEC. The CODEC allows a number of transmit clock configurations to ensure the maximum bit-rate can be achieved, while also ensuring that all other possible user configurations are supported. The transmit clock can be independent from the system clock, therefore decoupling the slower link control and status logic from the high speed bit-stream serialisation logic.

The CODEC can be configured to generate variable data rates internally, using either a clock divider or a clock enable generator, or externally using a PLL or other clock multiplexing method. The internal clock divider generates a new transmit clock from the reference bit clock using a programmable divider. The internal clock enable generator uses enable signals to gate the D input of the transmitter flip-flops, therefore varying the data rate.

When the CODEC is making a connection with the other end of the SpaceWire link then a default 10Mbits/s data rate is used. Dependent on the transmit clock configuration the default rate can be internally generated, as above, or an external 10/5 MHz input clock can be used as the reference. If an external 10/5 MHz clock is selected then it is also used to clock the initialisation state machine timeout counter and receiver disconnect detection counter. For a DDR enabled transmitter the rising and falling edges of the transmit clock are used therefore a 5 MHz clock is adequate to provide the 10MHz reference.

The SpaceWire CODEC uses ten valid discrete configurations for the transmitter bit clock as shown in “Table 2”. The configuration symbolic names match the names uses as constants in the top level configuration package.

Table 2 Transmit clock configuration

Symbolic name	Input clock	Default 10/5MHz	Variable data rate
SYS_DEFAULT	System Clock	System Clock	System Clock
SYS_SLOWCLK	System Clock	External 10/5 MHz	System Clock
SYS_SLOWCLK_DIV	System Clock	External 10/5 MHz	Clock divider
SYS_DIV	System Clock	Clock divider	Clock divider
SYS_EN	System Clock	Clock enable	Clock enable
TXCLK_DEFAULT	Transmit Clock	Transmit Clock	Transmit Clock
TXCLK_SLOWCLK	Transmit Clock	External 10/5 MHz	Transmit Clock
TXCLK_SLOWCLK_DIV	Transmit Clock	External 10/5 MHz	Clock divider
TXCLK_DIV	Transmit Clock	Clock divider	Clock divider
TXCLK_EN	Transmit Clock	Clock enable	Clock enable

When the variable data rate and default 10Mbits/s data rate are generated internally, then “(1)” is used to determine the default data rate and “(2)” is used to determine the variable transmit bit rate.

$$\text{10MbitRate} = \frac{TxMaxBitRate}{(CFG\_SLOWRATE\_TXCLK + 1)} \tag{1}$$

$$\text{TxBitRate} = \frac{TxMaxBitRate}{(TXRATE + 1)} \tag{2}$$

Where CFG_SLOWRATE_TXCLK and TXRATE are port signals to the SpaceWire CODEC.

Receive Buffer Configuration

The receive buffer is used by the CODEC to store received data characters until they can be read by the host system. A buffer size of at least eight bytes must be present to allow at least one FCT character to be transmitted, enabling link start-up and data transfers. The receive buffer size is configurable in the CODEC, allowing the user to select the size of the receive buffer from 8 bytes to the maximum size available in the users implementation technology. Valid sizes are on a 2^N boundary, i.e. buffer sizes of 8, 16, 32, etc., as this allows the most efficient implementation of the receive buffer read, write pointers and the receive buffer status flags. FCT credit operations are automatically performed internally by the CODEC dependent on the size of the receive buffer. For example if the CODEC is configured with a receive buffer size of 1K bytes, then 1K of data can be received and stored from the other end of the link before any reads are performed by the CODECs host controller.

The bit length of the read and write pointers is dependent on the size of the receive buffer, therefore the area footprint will increase with larger buffer sizes. The user also has control of the maximum number of outstanding data characters expected by the CODEC, between 8 and 56. Typically this value will be set to the maximum amount of receive buffer space available up to the maximum value of 56.

Receiver Data Character Re-synchronization configuration

Data characters in the serial input bit-stream are decoded in the recovered receiver clock domain and resynchronized into the receive buffer clock domain before being written to the receiver buffer as shown in “Fig 3”. Re-synchronising the data characters to the receive buffer clock decouples the receive buffer from the unstable receiver clock domain, which may be exhibit erratic behaviour and cause the receive buffer pointers to become corrupted. When receiving data at the maximum rate and using the slowest possible receive buffer clock then the time taken to re-synchronise one data character is longer than the time taken to receive it. Therefore the first data character received will be overwritten by the next data character before it can be stored in the receive buffer, if temporary storage is not allocated.

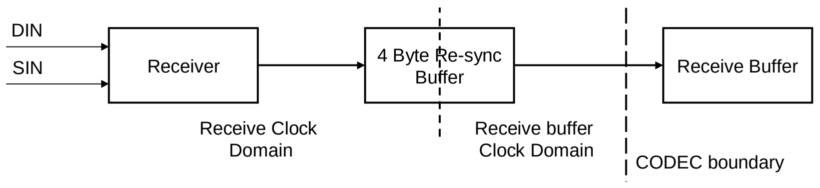

Fig 3 Receive Re-synchronisation Buffer

The CODEC uses an internal 4 byte storage buffer to ensure that enough time is available for each data character to be synchronized to the receive buffer clock domain. An important goal of the SpaceWire CODEC was to allow the transmit and receive data storage to be implemented by the user dependent on the target implementation technology, typically latches for ASIC vendors and on-chip RAM resources for gate array users, such as FPGAs. The CODEC allows the user to select the storage method, latches or flip-flops, used for the internal 4 byte buffer space, therefore ensuring the most efficient implementation of the CODEC is performed for the user’s technology.

When the latch implementation is chosen data is written to the latches using a three cycle setup, write and hold process to ensure race conditions and data loss do not occur. In stage one the data and write address is set up, in stage two the latches are written to using a write strobe to the gate enable of the latch and finally in stage three the data and address are held for one clock cycle to ensure each latch is written to cleanly.

The CODEC receiver adds a one cycle delay when receiving an EOP marker to ensure the three cycle latch write operation is not violation. EOP markers are only four bits in length compared to ten bits required for a data character. Therefore an EOP marker would be output from the receiver two clock cycles after a data character if the delay operation was not performed. The reception of a double or constant stream of EOP or EEP markers can also cause the three cycle latch write operation to be violated. In this case the user can configure the CODEC receiver to discard empty packets, packets consisting of no cargo and only an end of packet marker, as defined in the SpaceWire standard [1]. The CODEC receiver can also be thought of as a stand alone IP component, for example in a SpaceWire link monitoring application [6] where configuring the receiver to output empty packets can be used to detect a malfunctioning transmitter or host controller.

HOST INTERFACE

The host interface allows the host system to send and receive data or timecodes over the SpaceWire link and initiate link control functions such as starting the link.

Link Control and Status Interface

The initialisation state machine implements the SpaceWire exchange level state machine controller as shown in [1]. The interface to the host system comprises a Start signal, a Disable signal and an Auto-Start signal. A 19.8 µs exchange of silence protocol is used at startup and after link disconnection to ensure both link interfaces are ready to start-up.

After the exchange of silence protocol is performed the initialisation state machine is ready to make a connection and is enabled to do so dependent on (3).

$$\text{LinkEnabled} = (\text{not Disable}) \text{ and } (\text{Start or } (\text{Auto-Start and GotNULL})) \qquad (3)$$

When Start is asserted and disable is not asserted then the CODEC will always try to make a connection. If a connection is not established after 12.8 µs then the exchange of silence protocol is re-initiated and the connection attempt will be made after the 19.8 µs silence time. If a NULL character is received and then an FCT character the link will move to the run state and data characters can be transmitted. When the link is set to auto-start then the CODEC remains in the ready state until a connection attempt is made by the other end of the link. In this case the receiver is enabled and the transmitter disabled until the SpaceWire interface at the other end of the link tries to make a connection. Therefore the LVDS drivers at the output buffer for data and strobe can be tri-stated to save power. This approach is implemented successfully in the SpaceWire router [4] [5] where the CODEC status outputs are used to externally tri-state the LVDS drivers. If disable is asserted then the CODEC will not attempt to make a connection and will disconnect the link if a connection has already been made.

The control interface outputs a number of status bits shown in “Table 3”. The status bits are synchronised to the system clock and can be used as snapshot to the state of the CODEC interface.

Table 3 CODEC Status Outputs

Signal	Description
STATUS(0)	Disconnect error.
STATUS(1)	Parity error.
STATUS(2)	Escape error.
STATUS(3)	Receiver credit error.
STATUS(4)	Transmitter credit error.
STATUS(7:5)	Interface state encoded into three bits (6 states)
STATUS(8)	Interface state machine is in the *Run* state.
STATUS(9)	Receiver got NULL. Remains asserted after first NULL.
STATUS(10)	Receiver got FCT. Remains asserted after first FCT
STATUS(11)	Receiver got N-chars. Remains asserted after first N-Char
STATUS(12)	Receiver got Timecodes. Remains asserted after first Timecode
STATUS(13)	Transmitter has credit to send one more data character
STATUS(14)	N-char sequence error (N-char received before link state is Run)
STATUS(15)	Timecode sequence error (Timecode received before link state is Run)

Transmit Interface

The transmit interface allows the host controller to transmit data and timecodes to the other end of the SpaceWire link. Data to be transmitted is read directly from the external transmit FIFO to the transmit shift register, therefore ensuring the maximum data rate can be achieved. The interface is synchronous to the transmit bit clock and consists of an empty flag, which indicates if there is more data to send, an eight bit wide data input and one control bit indicating EOP or EEP and a read enable signal which reads one data character from the FIFO. A good implementation of the transmit FIFO will use independent read and write clocks to decouple the high speed transmit clock logic from the slower parallel data processing logic.

Timecodes are accepted by the transmitter using a tick signal and an eight bit timecode input. When the tick signal is asserted and the CODEC has made a connection with the other end of the SpaceWire link then the next character to be transmitted will be the input timecode. Timecodes and FCT characters always have priority over data characters.

Receive Interface

The receive interface allows the CODEC to output received data characters and timecodes to the host controller. As described previously the receive interface performs FCT credit operations internally dependent on the size of the receive buffer. To perform efficient error recovery and credit operations the read and write pointers for the receive buffer are implemented internally and the user simply has to supply a memory buffer for the data characters and connect the read and write pointers to the buffer. When a data character is received and no errors are detected then the data character is written to the receiver buffer at the write pointer location. If the host controller performs a read operation from the receive buffer then the read input to the CODEC should is asserted and the read pointer is incremented if the read was valid, i.e. the receive buffer was not already empty. The CODEC implements two receive buffer interface flags, an empty flag and a programmable level flag. The programmable flag can be used to detect when the receive buffer has filled to a pre-defined level, half full etc., and for example wake up an external DMA or processor controller to start reading data characters from the receive buffer.

If a timecode is received by the CODEC and no errors are detected then a tickout signal is asserted and the timecode is placed on the timeout eight bit data output.

Receive Buffer Clock Frequency

Typically the parallel data output from the CODEC will be processed at lower clock rate than the bit-stream clock rate. As shown below the maximum input bit rate which the CODEC can accept, dependent on the users target technology, and the minimum size of received packets determines the minimum frequency of the receive buffer clock. The examples below show two scenarios

Example 1 An input bit rate of 200Mbits/s and a constant flow of packets of one data character, with an EOP marker, causes a data character, including EOPs, to be output from the receiver on average every 35 ns. In this case the absolute minimum receive buffer clock frequency is 28.571 MHz (1/35ns).

Example 2 The same data rate as above is used but the minimum packet size expected by the CODEC is four data characters and an EOP marker. Therefore a constant flow of packets will cause one character to be output from the receiver every 44 ns on average. In this case the absolute minimum receive buffer clock frequency is 22.727 MHz.

Due to variations in the transmit clock period and possible jitter on the serial data-strobe inputs then a tolerance should be added to the absolute minimum receive buffer clock frequency. From the examples above it can be seen that (4) should be used to determine the absolute minimum receive buffer clock frequency.

$$\text{Minimum read clock frequency} = \frac{1}{\left(\frac{(MinNumData \times Tdata) + Teop}{MinNumData + 1}\right)} \tag{4}$$

$$\text{Where Tdata} = \left(\frac{1}{BitRate}\right) \times 10$$

$$\text{and Teop} = \left(\frac{1}{BitRate}\right) \times 4$$

Equation (5) defines the maximum input bit-rate which can be accepted by the CODEC in terms of the minimum receive buffer clock frequency and the number of data characters in a packet.

$$\text{MaxInputBitRate} = \frac{1}{\left(\frac{Trxbuf\_clk \times NumPktNchars}{NumPktBits}\right)} \tag{5}$$

$$\text{where Trxbuf\_clk} = \frac{1}{RxBufClkFrequency}$$

$$\text{and NumPktNchars} = MinNumData + 1$$

$$\text{and NumPktBits} = (MinNumData \times 10) + 4$$

In a typical system it is likely that the system clock will be adequate to resynchronize data characters to the receive clock buffer. If a very high input bit rate is required and the system clock is already tied to an external control logic source then the use of an independent receive buffer clock is supported by the CODEC.

PERFORMANCE AND LATENCY

The bit-rate performance of the SpaceWire CODEC is largely dependent on the users implementation technology. The CODEC is implemented in the SpaceWire router ASIC [5] and is expected to achieve a data rate of 200Mbits/s. Complex high speed devices can expect data rates up to the maximum 400Mbits/s achievable over a SpaceWire cable. The minimum transmitter latency is two transmit clock cycles measured from the time taken to accept the data or timecode character until the first bit of the character is output from the transmitter. The minimum receive latency four receive clock cycles plus an additional two receive buffer clock cycles to resynchronize characters to the receive buffer clock. One cycle of latency is added when the pipelined configuration is used. The maximum transmit latency depends on the current character being transmitted (timecodes and FCTs have priority over data characters) and the time taken to start the link when the first data character is ready to be transmitted.

VERIFICATION

The SpaceWire CODEC was verified in accordance with sub-clause 12.2.4 of the SpaceWire standard. Test-cases for the CODEC were derived directly from the SpaceWire standard, and the CODEC user manual for the added configurability and functionalities. Functionality testing was performed by a VHDL testbench implemented using the Austrian Aerospace SpaceWire CODEC testbench model. The testbench was modified to perform automatic verification of the CODEC.

AVAILABLITY

The SpaceWire CODEC VHDL IP will be made available for European space projects by ESA. It is expected that this IP will be widely used for both ASIC and FPGA designs where a SpaceWire interface is required. A comprehensive user manual has been included with the SpaceWire CODEC which details the function, configuration and interfaces which have been outlined in this paper.

CONCLUSIONS

This paper has given an overview of the work done on researching and developing a SpaceWire CODEC which will provide a platform for SpaceWire capable devices to communicate in the present and future.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the support of Pierre Fabry and Agustin Leon of ESA as technical managers. Specific thanks are due to the support and time given by Gerald Kempf, Austrian Aerospace GmbH, and Uwe Liebstueckel and Stephan Fischer, EADS Astrium GmbH, in preparing the SpaceWire CODEC VHDL code, verification and documentation. The SpaceWire CODEC was derived from the SpaceWire standard which resulted from the work of the SpaceWire working group comprising many people across many countries, companies and organisations

REFERENCES

[1] S.M. Parkes et al, “SpaceWire – Links, Nodes, Routers and Networks”, *European Cooperation for Space Standardization, Standard No. ECSS-E50-12A, Issue1, January 2003.*

[2] IEEE Computer Society, “IEEE Standard for a High Performance Serial Bus”, *IEEE Standard 1394-1995, IEEE, August 1996*.

[3] IEEE Computer Society, “IEEE Standard for Heterogeneous Interconnect (HIC) (Low-Cost, Low-Latency Scalable Serial Interconnect for Parallel System Construction)”, *IEEE Standard 1355-1995, IEEE, June 1996.*

[4] S.M. Parkes, “High Speed, Low power, Excellent EMC: LVDS for onboard data handling” *Proceedings of the 6th International Workshop on Digital Signal Processing Techniques for Space Applications, ESTEC, Sept 1998.*

[5] S.M. Parkes, C. McClements, G. Kempf, S. Fischer and A. Leon, “SpaceWire Router,” *International SpaceWire Seminar, ESTEC Noordwijk, The Netherlands, November 2003*

[6] S.M. Parkes, C McClements, “SpaceWire Networks”, DASIA 2002, ISSN 1609 042X, ISBN 92-9092-819-0.

[7] S.M Parkes, C McClements, S.J. Mills, I Martin, “SpaceWire: IP, Components, Development Support and Test Equipment”, *DASIA proceedings 2003.*