A CCSDS-Based Communication System for a Single Chip On-Board Computer

Daixun Zheng, Tanya Vladimirova and Prof Sir Martin Sweeting

Surrey Space Centre
University of Surrey
Guildford, Surrey GU2 7XH, UK
Tel: +44 1483 689278 Fax: +44 1483 686021
Email: zhengdaixun@ntlworld.com, t.vladimirova@eim.surrey.ac.uk,

Abstract

This paper is concerned with the implementation of a small satellite on-board computer (OBC) on a single programmable logic chip and with the development of a software communication system for it. The communication system is based on the CCSDS protocol, which is a standard communication protocol in the space industry. Integration of soft intellectual property cores forming a main subsystem of the system-on-a-chip OBC (SoC-OBC) is detailed. The structure and functions of the developed CCSDS software package are described. Simulation results verifying the operation of the CCSDS communication system on the SoC-OBC are presented.

1. Introduction

Small satellite development is targeted at achieving affordable and fast access to space. Advantages of Field Programmable Gate Arrays (FPGAs) such as flexibility of design, shorter time-to-market, lower cost, remote reconfigurability, etc. make them a suitable component for use in small satellite on-board systems. High-density FPGAs are becoming the preferred implementation platform in a number of terrestrial applications previously dominated by application specific integrated circuits (ASICs). So far high-density FPGAs have mostly been used in payload systems of small satellites, however introduction of radiation hardened versions of such devices by leading manufacturers as Xilinx and Actel paves the way for their use in main on-board electronic systems.

The Surrey Space Centre (SSC) has a long-term research programme, codenamed ChipSat, which aims to apply advanced micro- and nano- technologies to small satellite design. As part of the ChipSat programme an on-board computer (OBC) of a small satellite is implemented in the form of a system-on-a-chip (SoC) device. The SoC is modelled on a simplified version of an on-board computer designed by Surrey Satellite Technology Limited (SSTL). Soft intellectual property (IP) cores written in the hardware description language VHDL are used to build the system-on-a-chip on-board computer (SoC-OBC). The main blocks of the SoC are – microprocessor, memory error-detection-and-correction (EDAC) unit, bootstrap loader, HDLC controller, controller area network (CAN) interface, network interface, true IDE interface, mathematical co-processor and peripheral bus interface. A Xilinx Virtex FPGA is used for the prototyping of the SoC.

Specification and feasibility assessment of the SoC-OBC were reported in [1]. This paper presents recent research results concerned with the implementation of a main subsystem of the SoC-OBC and the development of a software communication system [2]. The communication

system uses the Consultative Committee of Space Data Systems (CCSDS) protocol, which it is a standard space industry communication protocol employed on numerous missions ranging from relatively simple low-earth orbit missions to deep space probes.

The paper is structured as follows. Section 2 details work on system level integration of IP cores forming a main subsystem of the SoC-OBC. Section 3 describes the structure and functions of the developed CCSDS software package. Section 4 presents results of a simulation experiment carried out to test the operation of the CCSDS communication system on the SoC-OBC.

2. System Level Integration

The system block-diagram of the SoC-OBC is shown in Figure 1. The microprocessor IP core is "LEON" - a SPARC V8 microprocessor IP core developed by the European Space Agency (ESA) [3]. The LEON processor core includes support for the Advanced Microcontroller Bus Architecture (AMBA) protocol, and therefore the AMBA bus is selected as an on-chip bus of the SoC. This section discuses a downsized implementation of the SoC, consisting of the LEON processor core and two peripheral cores - CAN and EDAC.

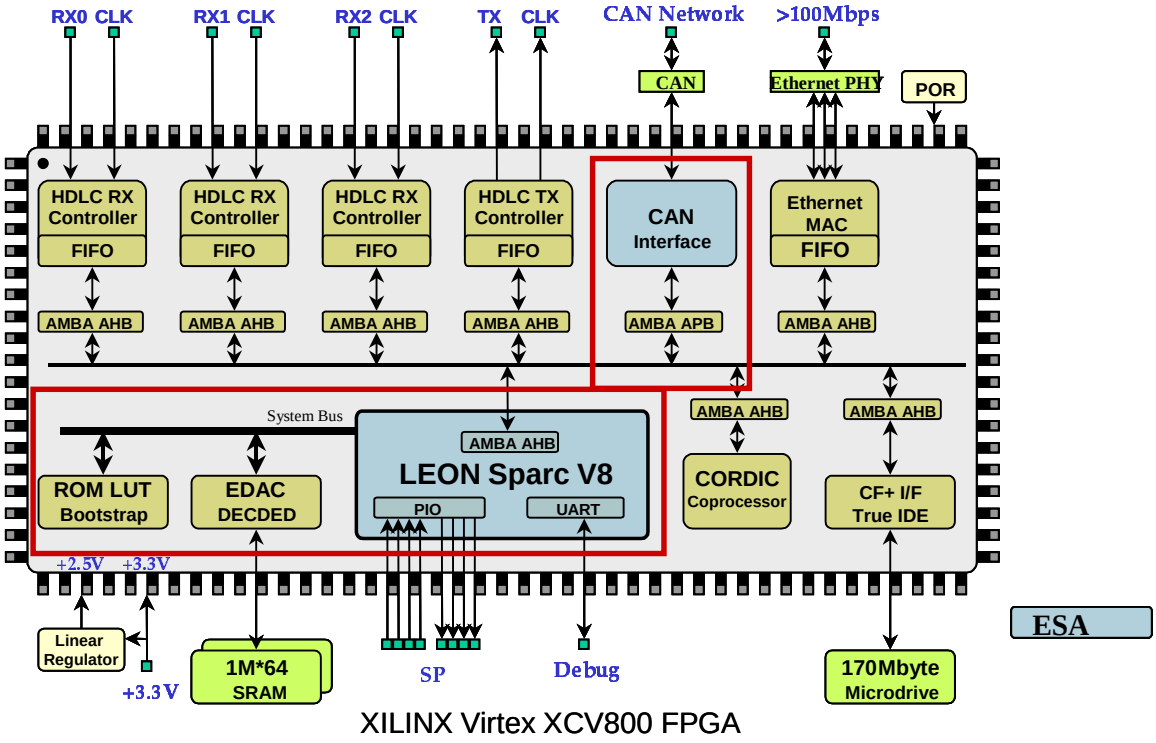

Figure 1. System Block-Diagram of Single Chip OBC [1]

The resulting subsystem LEON+CAN+EDAC (indicated in red in Figure 1) was implemented on a Xilinx Virtex XCV800 FPGA using an XESS XSV-800 prototyping board. The following CAD software tools were used: VHDL simulator ModelSim, synthesis tool Synplify 6.0+, back-end tools from Xilinx Foundation versions 2.1i and 3.1i.

2.1 Implementation of the Processor IP Core

The LEON processor IP core models a 32-bit RISC processor based on the SPARC V8 architecture. The core is implemented as a foundry independent, synthesisable VHDL model allowing rapid prototyping and porting. The VHDL model is extensively parametrical: register window size, cache size, fault-tolerance functions and clocking scheme can be defined through a single configuration file or through a graphic configuration tool.

The LEON processor is supported by a full set of software development tools, including the LEON Cross Compilation System **(**LECCS**)** and the LEON/SPARC simulator **(**TSIM). The LECCS suite of tools supports cross-compilation of single or multi-treaded C and C++ applications. The implemented LEON processor executes software programs in S-record format. Software applications compiled with LECCS can be converted to an S-record stream with the following command: `sparc-rtems-objcopy -O srec app app.srec .`

The LEON configuration bitstream is downloaded from a personal computer (PC) into the FPGA on the prototyping board through a parallel interface. Communication with the implemented (in the FPGA) LEON processor is carried out via a serial interface between the PC and the board via the standard I/O device of the LEON processor UART A. A terminal program, Tera-Term [4], is used to download software programs to memory in S-record format and to display program results on the PC screen. The interface between the PC and the LEON processor is managed by a CPLD (Xilinx XC95108) on the prototyping board (Figure 2). A Remote Target Monitor (RDBMon) can be downloaded in the FPGA and used for remote target debugging with the GNU Debugger (GDB). The RDBMon communicates with GDB through UART B of the LEON processor, as shown in Figure 2.

The bootloader of the LEON processor is a monitor that is integrated with the processor core in an on-chip boot PROM. On reset, the monitor scans all RAM-banks and configures the Memory Control Register 2 accordingly. The monitor writes a boot message (describing the detected memory configuration) to the transmitter port of UART A and then waits for S-records to be downloaded on the receiver port of UART A. It recognizes two types of S-records: memory contents and start address. A memory contents S-record is saved to the specified address in memory, while a start address record will cause the monitor to jump to the indicated address.

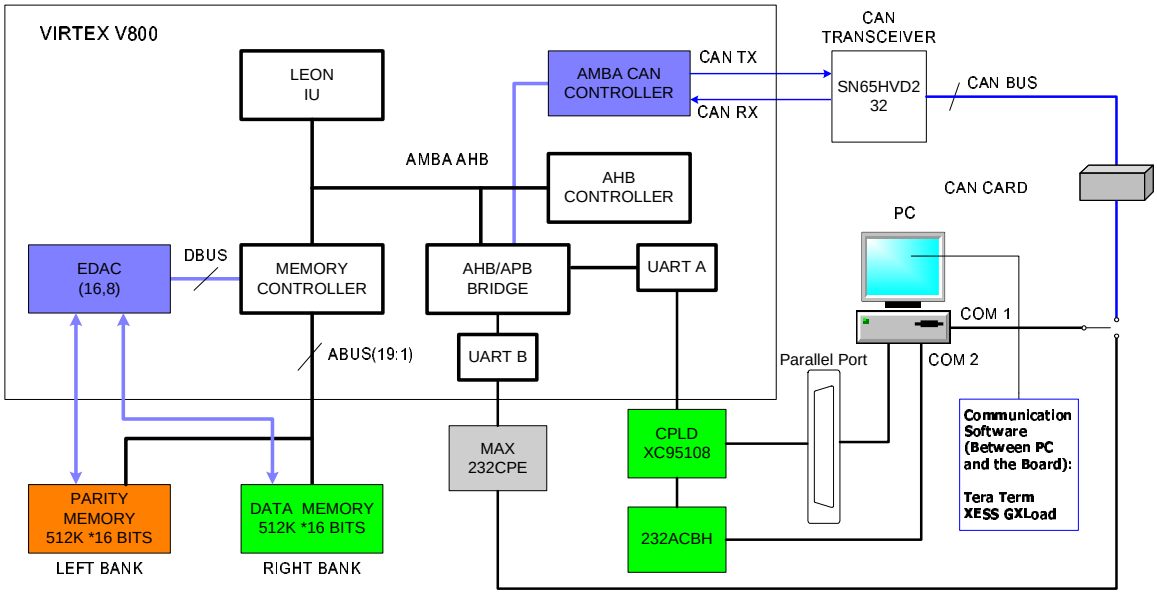

Figure 2. Implementation and Test of the LEON processor with the CAN and EDAC IP Cores

2.2 Integration of the CAN IP Core

The CAN IP core used in this project is the HurriCANe VHDL core developed by ESA. Since the CAN bus is a low-speed peripheral, the HurriCANe core is integrated with the LEON processor core via the Advanced Peripheral Bus (APB) of the AMBA protocol [5], as illustrated

in Figure 2. Therefore, the CAN core registers are mapped as on-chip registers to memory address range 0x800000C0 – 0x800000E8 and the CAN core is set as one of the APB slaves in the LEON processor VHDL model:

```
(“0011000000", "0011101000", 10, true) -- CAN Configuration
```

The CAN core and the LEON processor are connected as shown in Figure 3.

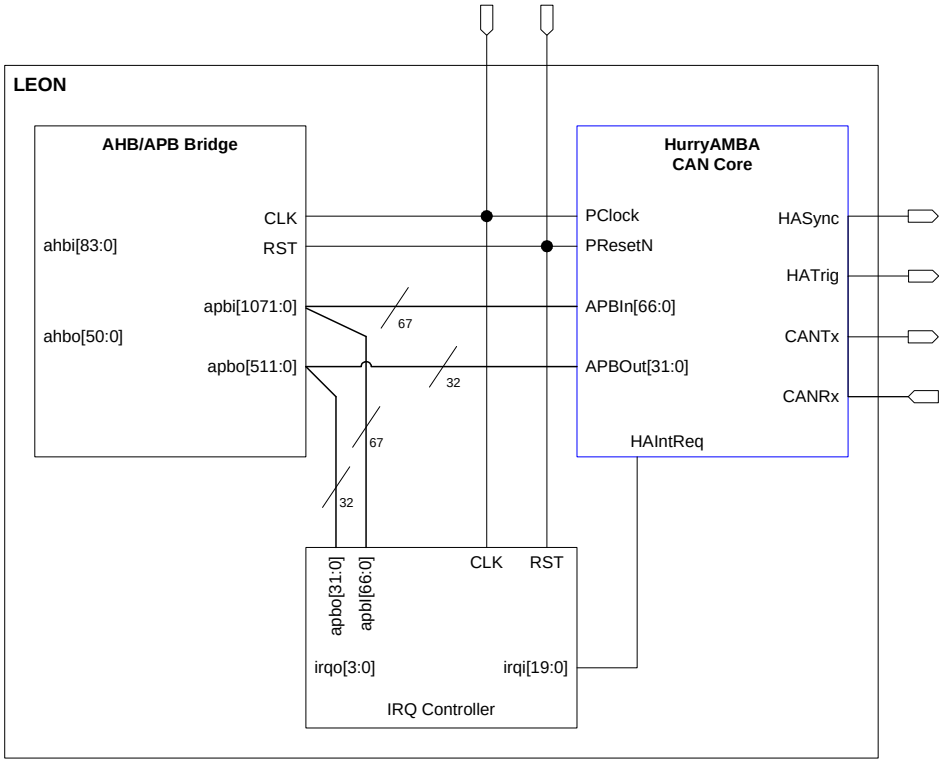

Figure 3. Connection of the CAN core to the LEON Processor IP Core

In addition to the prototyping board an external CAN transceiver and a CAN controller card are employed to test the CAN core, as shown in Figure 2. The transceiver is SN65HVD232 from Texas Instruments, designed for use with 3.3-V output CAN controllers. The CAN IP core and the external CAN card act as two equivalent devices or nodes on the CAN bus. The CAN card communicates with the PC through a serial port and is controlled by the software package CAN-PCS developed by SSTL. A test program for the CAN core is compiled and downloaded to the board. The test program is compatible with the SSTL protocol for on-board CAN communication. The test scenario is as follows. The CAN card sends a telemetry request message. After the CAN core receives this request correctly, it sends a CAN frame with a telemetry respond message back to the CAN card. The CAN-PCS software is used to monitor the transmission of the CAN messages from the PC screen.

In the CAN core test, it is necessary that the CAN core and the CAN card have matching baud rates. Achieving an exact match required adjustments of the equipment and therefore was not pursued. Instead, it was decided to use an approximating approach, which yielded the following best data rate values:

$$BaudRate_{CANCORE} = 312500\ bps \quad \text{and} \quad BaudRate_{CANCARD} = 307197\ bps$$

Despite the slight difference between the two baud rates (around 1.7%), the communication process was not affected and the two CAN nodes exchanged messages correctly throughout the simulation experiment described in Section 4.

2.3 Integration of the EDAC IP Core

A memory error-detection-and-correction (EDAC) unit is integrated with the LEON processor via the system bus. The EDAC IP core used in this experiment is based on a double-bit correcting Quasi-Cyclic (16,8) shortened EDAC code [6] rather than the traditional single-bit correcting Hamming code [7]. The core was developed in-house by SSTL. The interface between the EDAC core, the LEON core and the RAM memory on the prototyping board is shown in Figure 4. The left memory bank on the board is used as 'parity memory' while the right memory bank is used as 'data memory'.

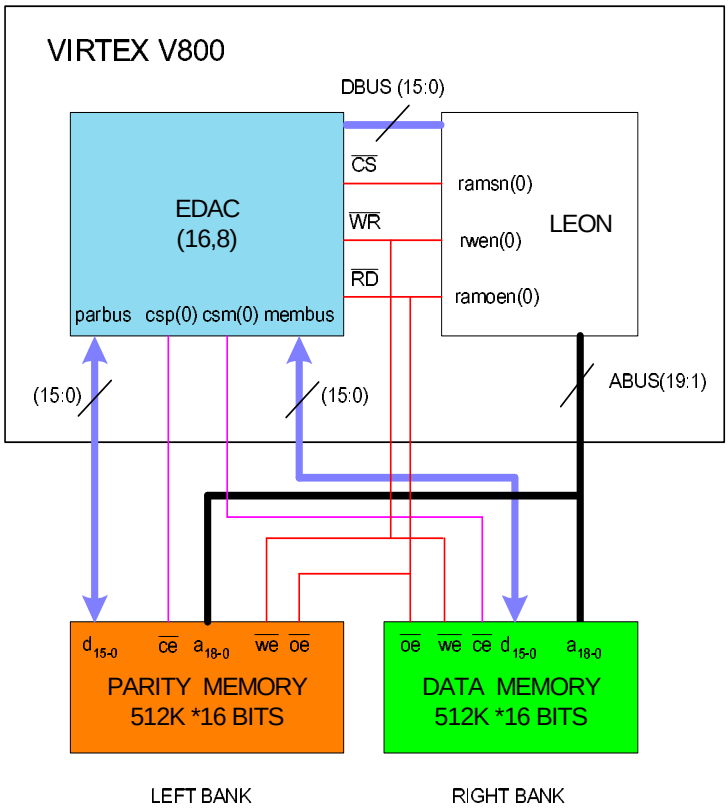

Figure 4. Integration of the EDAC Core with the LEON Processor

During a write cycle, the EDAC core generates parity bits which are stored at the same address as the data. During a read cycle, parity is generated again and compared with the stored parity. If the resultant syndrome is not equal to zero, a Look-Up Table (LUT) is used. The data is then corrected and passed on to the CPU.

To test the EDAC IP a number of application programs were downloaded in S-record format (e.g. hello.srec and float.srec) to the right memory bank of the XESS board and subsequently executed on the LEON processor. In all cases the programs ran as expected, showing that the EDAC core engaged correctly in memory read and write operations. In addition, the contents of the left and right memory banks were read-back during execution of test programs and verified by manual inspection. The read-back operation was performed with the GXSLoad tool supplied by XESS Corporation [8], the manufacturer of the prototyping board.

2.4 Performance and Chip Area

This section gives details about the performance and size of the SoC-OBC design. Table 1 presents a summary of results showing estimated frequency and area requirements of the microprocessor core and the total flattened main subsystem module of the SoC.

Table 1. Implementation Results

SoC Modules	*Virtex XCV800-4*				
	CLB Slices	**BlockRAMs**	**IOBs**	**MHz (P&R[1])**	**MHz (Sim[2])**
LEON 1-2.2.2	2572 out of 9408 (27%)	14/28(50%)	61 out of 166 (36%)	23.226	26.67
LEON 1-2.4.0	3640 out of 9408 (38%)	14/28(50%)	58 out of 166 (34%)	24.79	26.67
LEON 2-1.0.2a	3754 out of 9408 (39%)	14/28(50%)	60 out of 166 (36%)	24.03	26.67
LEON 1-2.4.0 + CAN+EDAC	4694 out of 9408 (49%)	14/28(50%)	100 out of 166 (60%)	25.22	—
LEON 2-1.0.2a + CAN+EDAC	4749 out of 9408 (50%)	14/28(50%)	99 out of 166 (59%)	25.186	—

Notes to Table 1:

1 – Frequency predicted by the P&R tool

2 – Frequency obtained by simulation of the back-annotated design.

Three versions of the LEON processor IP core (1-2.2.2, 1-2.4.0 and 2-1.02a) have been implemented and verified at 25 MHz on the XESS prototyping board. Synthesis results of the LEON processor IP core are affected by the configuration type. We have used the pre-defined configuration (`constant conf : config_type := fpga_2k2k_v8_softprom`) [9].

Table 1 shows that the implemented subsystem of the SoC-OBC (LEON+CAN+EDAC) requires about 50% of the capacity of the target FPGA Virtex XCV800. The bitstream file of the implemented subsystem measures 576 Kbytes. Estimates of the area of the entire SoC-OBC (without the co-processor) indicate that it fits in about three quarters of the chip [1]. The 32-bit floating-point mathematical co-processor is based on the CORDIC algorithm and is able to evaluate 17 floating-point operations, such as addition, multiplication, division, square root, trigonometric and hyperbolic functions as well as log and exp. The co-processor, which is still in a process of testing, takes approximately half of an XCV800 chip. Hence the complete OBC system will not fit in a Virtex XCV800 chip, however, a larger Virtex chip, for example, Virtex XCV2000E, will be able to house the implementation of the whole system.

3. The CCSDS Software Package

This section presents a software communication system for transmission of telemetry (TLM) and telecommand (TC) data that satisfies the needs of a single chip on-board computer. The complete CCSDS TLM and TC implementation is very complex for low-cost small satellites and hardware implementations are expensive. Therefore, the CCSDS software package developed at SSC focuses on a subset of functions such that it represents a simplified yet reliable standalone alternative software implementation of the CCSDS TLM and TC Command

Operation Protocol (COP-1) [10]. The structure of the CCSDS software package is shown in Figure 5.

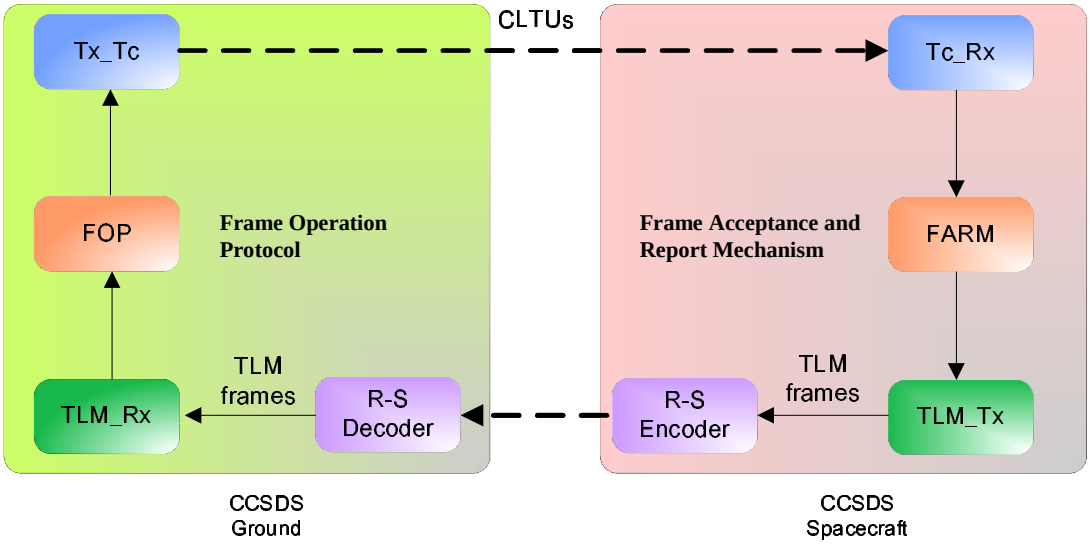

Figure 5. Structure and Interfaces of the CCSDS Software Package

The specification of the CCSDS software package follows strictly the CCSDS TLM and TC recommendation documents [11,12,13,14,15]. The COP-1 service specification is based on a simplified version of the CCSDS Frame Acceptance and Report Mechanism (FARM) and Frame Operation Protocol (FOP). Both the FARM and FOP systems satisfy the essential requirements for a reliable CCSDS communication system. The software imposes minimal memory footprint and performance requirements on the On-Board Computer. The CCSDS communication system implements the Packetisation Layer, the Transfer Layer and the Coding Layer of both the TC and TLM Systems. A CCSDS Reed-Solomon (R-S) encoder and decoder are used for the TLM channel coding, while the Bose, Chaudhuri and Hocquenghem (BCH) cyclic redundancy check (CRC) error detecting code is used for the TC coding.

The functions of the ground segment are to format CCSDS TC packets and CCSDS TC frames; calculate the BCH check; insert TC packets into the data field of TC frames; insert TC frames into codeblocks; format Control Link Transfer Units (CLTU) to ensure synchronisation; implement the R-S or Turbo decoding; receive CCSDS TLM frames; subtract CCSDS TLM packets from the Data Field of the TLM frames; decode the CRC and if no error is detected, the raw TLM data is passed to be analysed and displayed. The functions of the spacecraft segment are to format CCSDS TLM packets; format CCSDS TLM frames; insert TLM packets into the data field of the TLM frames; calculate the CCSDS recommended CRC code of the frame inserted at the end of TLM frames; format the Attached Synchronized Marker (ASM); implement R-S or Turbo encoding; receive and decode CCSDS TC frames; detect whether a transmission bit error has been identified by the BCH code. If there is no error, the telecommand will be passed to the on-board CAN bus and accepted by the corresponding CAN node. The TC retransmission system utilises the COP-1 "go-back-n" automatic retransmission protocol, which consists of the pair of synchronized procedures, FOP (in the ground segment) and FARM (in the spacecraft segment). The FOP transmits TC transfer frames to the FARM. The FARM returns Command Link Control Words (CLCW) within the TLM transfer frames.

The R-S encoder/decoder program used in the TLM coding system is derived from a public domain program (`rs.c` written by Phil Karn). The R-S code is compliant with the coding algorithm in the CCSDS recommendation [12,16] and has the following main features:

- uses 8 bits per symbol;

- uses 255 symbols per codeword and is a (255, 223) code (the first 223 symbols are information symbols, and the last 32 symbols are check symbols);
- corrects and detects 16 symbol errors;
- the field polynomial is $f(x) = x^8 + x^7 + x^2 + x + 1$;
- the interleaving is chosen as 1.

The size of the CCSDS software is as follows: Ground segment (including the R-S decoder) - 177 Kbytes; Spacecraft segment (including the R-S encoder) - 176 Kbytes; R-S code: 21.7Kbytes.

4. Simulation Experiment

This section describes a simulation experiment carried out to test the operation of the CCSDS package in conjunction with the SoC-OBC. Figure 6 outlines the experimental set-up used for the simulation. The operation of the on-board segment was emulated on the SoC-OBC within the XESS prototyping board, the operation of the ground station segment - on a personal computer. The communication between the two segments was simulated by a serial link. An external CAN controller card was used to emulate an on-board payload.

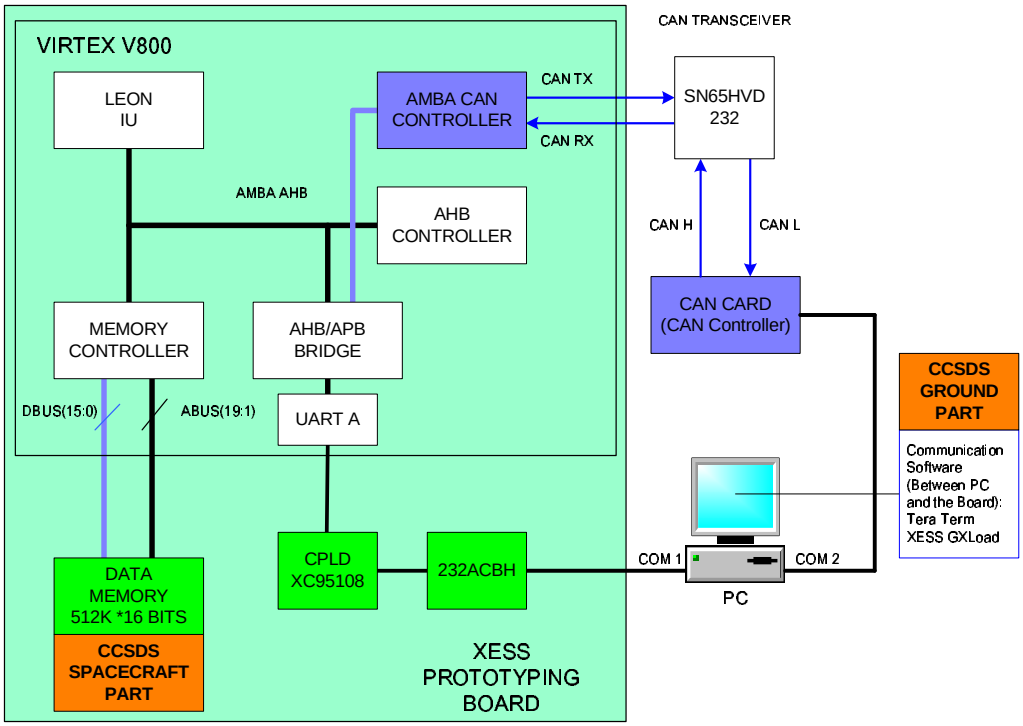

Figure 6. Experimental Set-up

Figure 7 shows the data flow for the simulation of the CCSDS communication system, where the on-board software is run by the subsystem of the SoC-OBC described in Section 2 above. The CCSDS spacecraft programs are downloaded to the RAM memory on the prototyping board and are executed by the LEON processor. The operating frequency of the implemented LEON processor is 25MHz. The “on-board” CAN bus system consists of two nodes – the HurriCANe IP core and a CAN card, where the latter emulates the responsibilities of an on-board payload. The CAN system closes the on-board communication loop receiving telecommand data and generating telemetry data. The SoC-OBC communicates with the PC via an asynchronous RS-232 serial port by using the UART A of the LEON processor.

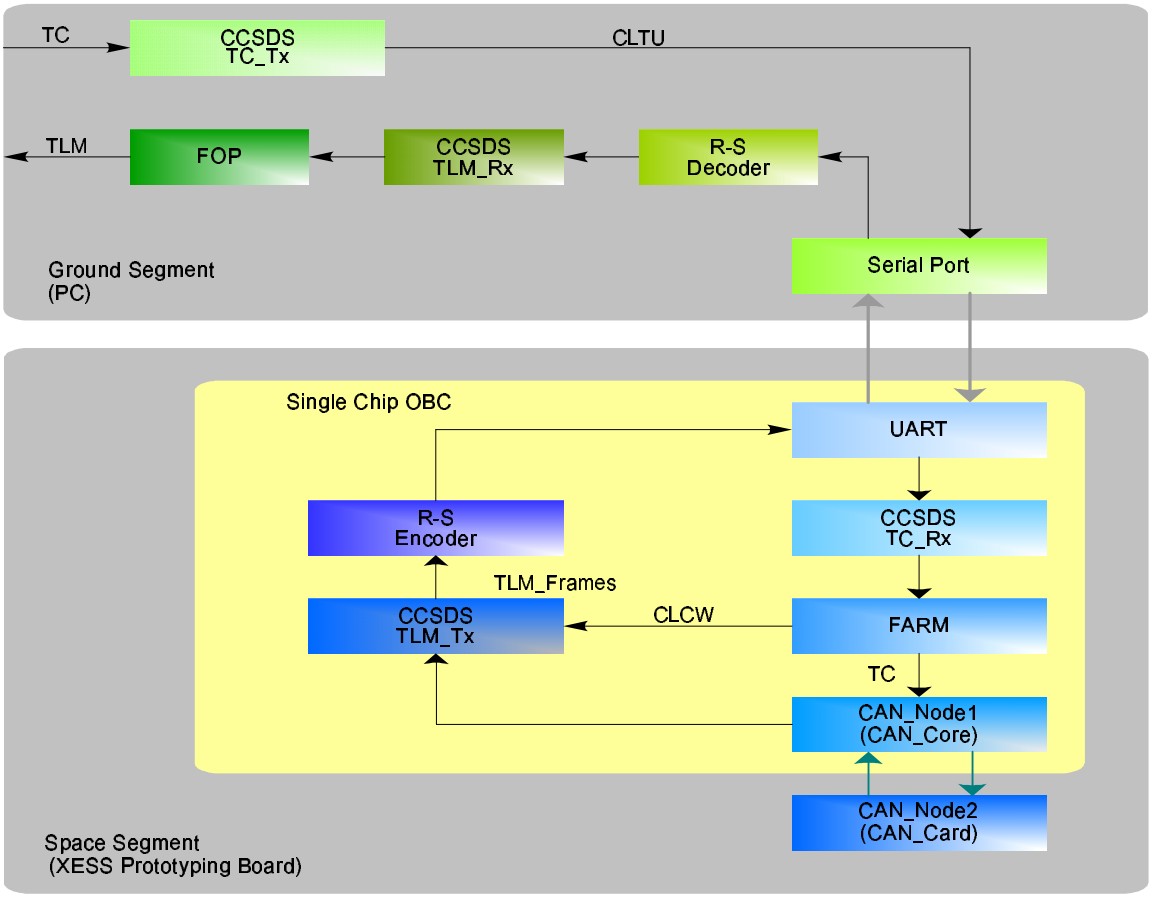

Figure 7. Simulation Data Flow of the CCSDS Communication System

4.1. Simulation Results

This section presents simulation results covering the entire communication data-flow cycle – sending of TC data to the spacecraft, on-board processing of TC data, generation and sending of TLM data to ground.

Figure 8 illustrates the transmission of telecommand data from ground to the spacecraft segment. The TC_TX module sends a CLTU TC frame (338 bytes, two tails) to the SoC-OBC (CCSDS requires MSB to be sent first). The TC_RX module (which runs on the LEON processor implemented in the FPGA) receives this CLTU and decodes it by the BCH decoder. In the CLTU frame, the first two bytes ‘eb 90’ (hexadecimal) are ‘Start Sequence’, followed by ‘0 1 5 17 0 0 1 be 0 1 1 c 0 0 0 d8’, where ‘be’ and ‘d8’ are the BCH parity bytes. So the TC frame contents (without BCH) are ‘0 1 5 17 0 0 1 0 1 1 c 0 0 0’. The rest of the bits are ‘0’ – IDLE TC frame. The tail sequence is: c5 c5 c5 c5 c5 c5 c5 79 (8 bytes) - two tails.

Figure 9 illustrates the transmission of telecommand data from the OBC to the payload and telemetry data from the payload to the OBC. The telecommand to be sent to the CAN card is “idle” (all 0s), therefore the CAN core sends the message ‘50 0 0 0 0 0 0 0’. The CAN card receives this TC and sends back a TLM message ‘18 0 0 8d dd 78 0’, which will be included in the TLM frame.

Figure 10 shows the transmission of telemetry data from the spacecraft segment to ground. The TLM_Tx module generates a TLM frame, which includes a CLCW, and TLM data obtained from the CAN card. The R-S encoder module encodes the TLM frame after which the encoded TLM frame is sent to the PC via the serial port.

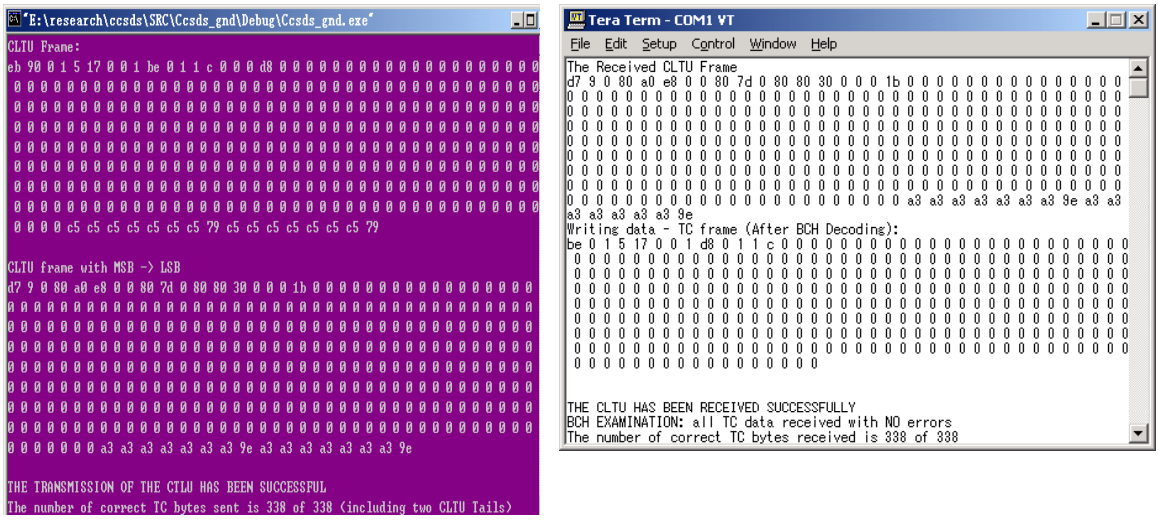

Figure 8. Transmission of a CLTU TC Frame from TC_Tx to TC_Rx

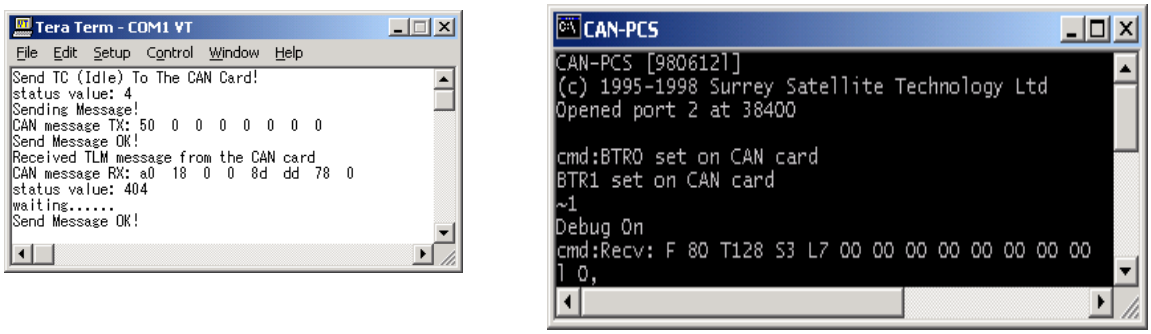

Figure 9. Transmission of TC & TLM Between the CAN Core and the CAN Card

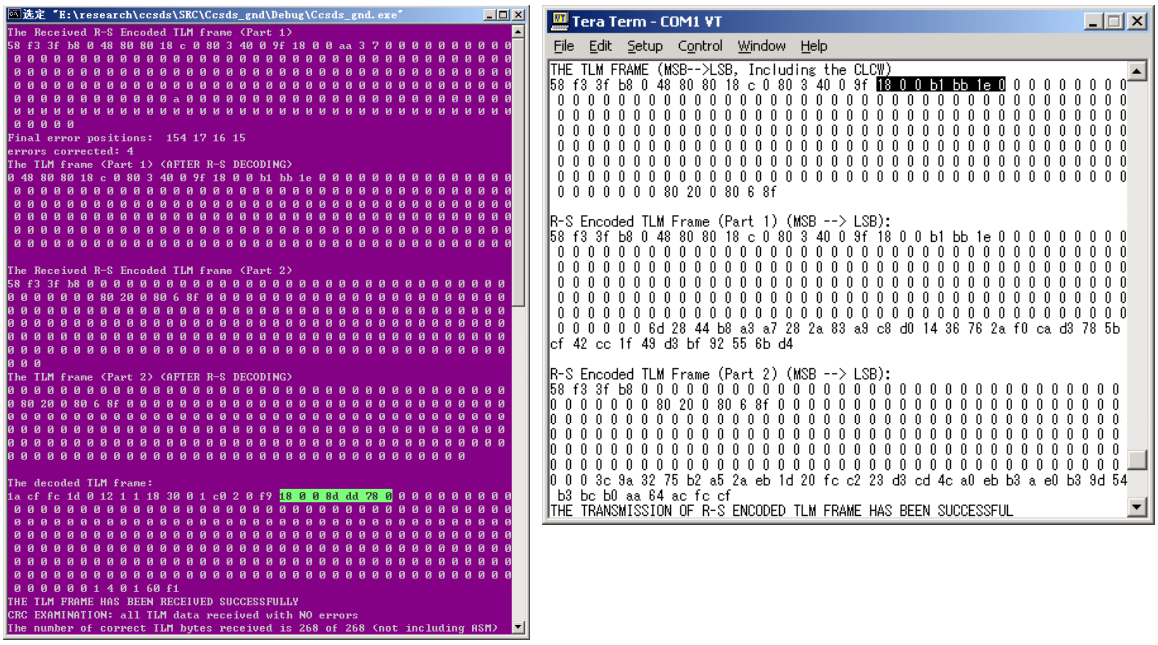

Figure 10. Transmission of a TLM Frame from TLM_Tx to TLM_Rx

The result of the R-S decoding can be seen in Figure 10. The example illustrates a case when three bytes of the TLM frames have errors. The R-S decoder corrects the errors and recovers the correct codeword as a result of which the TLM frame is received and decoded correctly. In the TLM frame, the first 4 bytes '1a cf fc 1d' are the ASM; after conversion MSB→LSB, they become '58 f3 3f b8'. In the CCSDS standard, these bytes form the header and are not R-S encoded. The header bytes are followed by a frame header of 6 bytes: '0 48 80 80 18 c'; a package (source) header of 6 bytes: '0 80 3 40 0 9f'; and TLM data from the CAN card:'18 0 0 b1 bb 1e 0' (obtained MSB→LSB from '18 0 0 8d dd 78 0'). The tail consists of a CLCW of 4 bytes: '80 20 0 80'and the CRC bytes of the TLM frame: '6 8f'.

5. Conclusions

The work presented in this paper is part of a research project aiming to reduce the size of an on-board computer to a single chip. A downsized implementation of a single-chip OBC, consisting of the LEON processor core and two peripheral cores - CAN and EDAC - has been developed. Area and performance estimates indicate that the SoC-OBC without the mathematical co-processor fits in a single XILINX Virtex XCV800 FPGA chip, the complete system requiring a larger Virtex chip, for example Virtex XCV2000E. These findings serve as an illustration of the increased capacity of high-density FPGAs, which are nowadays large enough to house complex digital SoCs.

A simplified communication system, specifically designed to meet the needs of a single-chip on-board computer has been developed. The system represents a streamlined, yet reliable and automated, standalone software implementation of the CCSDS protocol. The software package features a modular structure, which can facilitate easy expansions of functionality to suit specific mission requirements. The software imposes minimal memory footprint and performance requirements on the OBC. The functionality of the package has been verified via simulation.

The combination of a single chip OBC and a software CCSDS-based communication system supported by a thin-layer hardware interface can provide a cost effective and flexible communication solution for small satellites.

Acknowledgements

The research presented in this paper was sponsored by ESA under Contract N 14894 "A System-on-a-Chip for Small Satellite Data Processing and Control".

We would like to express our gratitude to the following people for their help, advice and encouragement –Andrew Dachs, Simon Prasad and Adrian Woodroffe of SSTL, Hans Tiggeler of Mentor Graphics, Sandi Habinc of Gaisler Research, and Ronald Weigand of ESA.

References

1. H.Tiggeler, T.Vladimirova, D.Zheng. "A System-on-a-Chip for Small Satellite Data Processing and Control", Proceedings of Military and Aerospace Applications of Programmable Devices and Technologies International Conference (MAPLD'2000), P20, September 2000, Laurel, Maryland US, NASA.

2. D.Zheng, T.Vladimirova, H.Tiggeler, M.Sweeting. “Reconfigurable Single-Chip On-Board Computer for a Small Satellite”, 52nd International Astronautical Congress, Toulouse, France, October 1-5, 2001, IAF-01-U3.09.

3. J. Gaisler, “A Portable Fault-tolerant Microprocessor Based on the SPARC V8 Architecture,” Proceedings of DASIA (Data Systems In Aerospace) 99, pp 173-178, Portugal, May 1999.

4. Tera Term Home Page http://hp.vector.co.jp/authors/VA002416/teraterm.html

5. AMBA Bus, http://www.arm.com/sitearchitek/armtech.ns4/html/amba

6. M.S. Hodgart, H. Tiggeler. “A (16,8) Error Correcting Code (t=2) for Critical Memory Applications.” DASIA2000, Montreal, Canada, 22-26 May 2000.

7. W.W. Petersen, and E.J. Weldon. Error-correcting Codes, MIT Press. 2nd edition, 1972, pp 256- 261.

8. XESS Corporation, http://www.xess.com

9. J. Gaisler, “The LEON Processor User’s Manual”, http://www.gaisler.com/doc/Leon2-1.0.8.pdf

10. I. Rutter, T.Vladimirova, H.Tiggeler.“A CCSDS Software System for a Single-Chip On-Board Computer of a Small Satellite”, AIAA Small Satellites Conference, 2001, SSC01-VI-4.

11. “Packet Telemetry Recommendation for Space Data Systems Standards”, CCSDS 102.0-B-4. Blue Book. Issue 4. Washington, D.C.: CCSDS, November 1995.

12. “Recommendation for Telemetry Channel Coding” 6, CCSDS 101.0-4. Blue Book. Issue 4. Washington, D.C.: CCSDS, May 1999

13. “Telecommand Part 2.1. Recommendation for Telecommand: Command Operation Procedures”, CCSDS 202.1-B-1. Blue Book. Issue 1. Washington, D.C.: CCSDS, October 1991.

14. “Telecommand Part 2. Recommendation for Telecommand: Data Routing Service”, CCSDS 202.0-B-2. Blue Book. Issue 2. Washington, D.C.: CCSDS, November 1992.

15. “Telecommand Part 1. Recommendation for Telecommand: Channel Service”, CCSDS 201.0-B-2. Blue Book. Issue 2. Washington, D.C.: CCSDS, November 1995.

16. “Telemetry Channel Coding”, CCSDS 101.0-B-5. Blue Book. Issue 5. Washington, D.C.: CCSDS, June 2001”