A VHDL implementation of CANopen Protocol for CAN Bus On Board Spacecraft

F. Tortosa López[1], +31 71 565 5772, francisco.tortosa.lopez@esa.int
P. Roos[2], +31 71 565 3692, peter.roos@esa.int

[1]VITROCISET @ ESA-ESTEC, Keplerlaan 1, 2201 AZ Noordwijk, The Netherlands
[2]ESA-ESTEC, Keplerlaan 1, 2201 AZ Noordwijk, The Netherlands

Abstract

This paper presents the design of a VHDL IP Core developed at ESA (named CANopy) which implements in hardware the basic features of the higher layer protocol for CAN Bus on board spacecraft applications. This includes the basic CANopen features as well as several add-ons that are result of the efforts realised to date by the ESA-Industry Working Group on CAN bus.

1. Introduction

There is an increasing utilisation of *Controller Area Network* (CAN) Bus in spacecraft onboard applications as the main system bus. It has already been successfully used in many missions (SMART-1, MATROSHKA, SSTL Satellites), and will soon be launched in ATV.

CAN Bus provides performances similar to other more traditional buses such as MIL-STD-1553 or ESA OBDH, but with a very low cost, while improving robustness due to its multiple error detection mechanisms. CAN Bus is currently used in virtually all new cars and is one of the most common buses for modern industry automation projects. Thanks to its wide acceptance in the commercial market, numerous development tools are available that support and reduce the cost of all the engineering process, from specifications and design to the final assembly, integration and tests.

There are already some CAN components and Intellectual Property (IP) cores tested and available for use in space projects. New radiation tolerant devices and System-On-Chip solutions will very soon complement them, some of them under development within ESA contracts.

ESA-ESTEC established an ESA-Industry Working Group on CAN Bus, in order to identify the main issues for its use in space applications. The main output of the working group has been a set of guidelines for the use of CAN Bus in spacecraft onboard applications, compiled in the form of a draft recommendation that was presented in the last two DASIA conferences ([1],[2]). This draft is foreseen to be formalised as an ECSS recommendation.

ESA-ESTEC also conceived and developed a demonstrator that implements the proposed recommendations in order to validate them. This demonstrator uses several nodes that communicate through a real CAN bus, emulating the specific traffic demands within a typical spacecraft scenario, taking the SMART-1 example as the reference design. As part of this demonstrator, an IP-Core in VHDL has been designed that implements in hardware the basic features of the higher layer protocol. This paper will focus on the characteristics of this IP-Core.

2. The CAN Working Group Draft Recommendation

The CAN Bus 2.0B specification ([5]) from BOSH GmbH defines with reference to the OSI-7 layer model, the Medium Access Control (MAC) and part of the Logical Link Control (LLC) sub-layers of the Data Link Layer.

The CAN Bus specification left the physical layer undefined. A number of complementary standards have been defined for terrestrial applications: The ISO11898 Part 1 standard ([3]) specifies the Data Link Layer and Physical Signalling for CAN. Parts 2 ([4]) and 3(draft) of ISO11898 specify “high-speed” and “low-speed” CAN respectively. We have selected ISO11898-1:2003 and ISO11898-2:2003 as the normative reference for CAN within the CAN Working Group draft recommendation. Optionally, and in order to ensure to be suitable for a majority of spacecraft missions, implementations of the physical layer based on RS-485 transceivers can be considered.

The working group has also evaluated a number of existing Higher Layer Protocols, coming to the conclusion that CANOpen provided many of the services needed for space applications. The main features being:

- Network management (NMT)
- Method of exchanging process data (PDOs)
- Device configuration (SDOs)
- Emergency Notification (EMCY)
- Time stamping and synchronization (SYNC and TIME)

While the range of services defined in CANOpen is broad, it is scalable, as the number of mandatory elements is reasonably low, and allows for simplified implementations in nodes not requiring the full CANOpen capabilities.

The service provided by CANOpen for segmented data transfers has been complemented with a Large Data Unit Transfer Protocol that can co-exist with CANopen services. It provides higher protocol efficiency than the segmented SDO transfers specified by CANopen.

A complementary Time Distribution Protocol has also been specified. The protocol proposed is based on CANOpen defined services, but uses CCSDS S/C Standard Data Types for describing time.

Finally, the Bus Redundancy Management is addressed. Two alternative architectures have been proposed (see *Figure 1*): A *selective* bus access architecture that implements a single CAN controller and redundant CAN bus via two transceivers, and a *parallel* bus access architecture with two CAN controllers that allow simultaneous communications on both buses.

A baseline procedure for bus monitoring and reconfiguration management is also described. It makes use of CANopen Heartbeat message to determine the active bus. The redundancy master sends its heartbeat message periodically only on one of the buses, which is considered the active bus. The slave nodes must monitor the presence of the master hearbeat message to determine the active bus. The slave node can toggle between the nominal and the redundant buses when searching for the Heartbeat of the master.

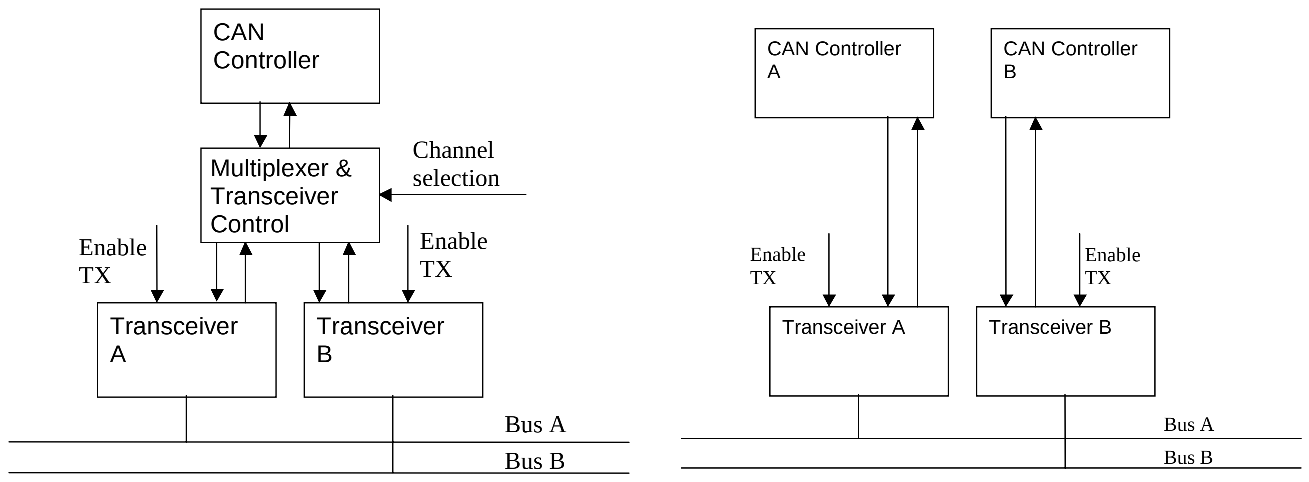

Figure 1: Selective and parallel bus access architectures

All these recommendations and guidelines for the use of CAN Bus on board spacecraft have been compiled by the CAN Working Group in the form of a draft proposal for an ECSS recommendation ([6]).

3. CAN Building blocks available for Space

If we consider the elements required for the use of CAN Bus in Space, we can see that the basic range of components (transceivers and controllers) is quite well covered and that many system-on-chip developments and on-board computers are providing more and more CAN Bus interfaces.

Considering the transceivers, ISO11898-2 compliant options are available, both commercial and Rad Tolerant (15KRad) which are suitable for a variety of applications. Whenever the radiation requirements cannot be achieved with ISO11898-2 standard components, “modified RS-485” based solutions are also possible, which are implemented utilizing Rad-hard RS-485 transceivers in a special way that emulates the ISO11898-2 behaviour. This solution has already been implemented in several space missions.

In terms of CAN Controllers, the variety and diversity of options is even bigger, ranging from commercial components (many available: Philips, Infineon, etc.) to Rad-Hard ASICs (like CASA2). There is also a wide offer of CAN controller VHDL IP Cores available (HurriCANe, OpenCores, BOSCH, etc) that can be implemented in Rad-hard/rad-tolerant FPGAs or ASICs depending on the application requirements or integrated with other IPs in more complex designs.

Finally, several new system-on-chip developments are ongoing, that provide one or several CAN Bus interfaces integrated with diverse microcontrollers and microprocessors and other types of interfaces, in order to perform several functions within the spacecraft, to be used in the main control computer, in remote terminals, payloads or micro-RTUs.

All these elements provide a good and sustainable platform for the use of CAN Bus on board spacecraft, especially interesting for ESA if we also consider that all the building blocks mentioned are fully European technology.

4. Why a VHDL implementation of CANopen?

We must not forget that any given project that wants to use CAN Bus as the main spacecraft command and control bus needs also to implement a higher layer protocol on top of it. For that reason, the CAN Working Group provided a set of recommendations (briefly explained in section 2 of this paper) which are based in the well-known commercially available CANopen higher layer protocol, in order to standardize the use of CAN Bus on board spacecrafts.

The adoption of a Higher Layer Protocol like CANOpen, leads frequently to some opposing opinions that suggest that the protocol is heavy and would require a significant processing power in all the nodes of the network, leading to an increased complexity of the nodes.

One of the main reasons for this thought is that CANopen has traditionally been implemented in software. This has been a good approach for terrestrial application, where there is mass production and low-cost microcontrollers are widely available and can be used almost anywhere in the design. The flexibility of a software implementation is also an asset, as it allows for simple evolutions of the control system.

However, for space applications, a hardware-only implementation can provide some advantages. As the availability of microcontrollers/processors is limited, the possibility of having CPU-less implementations for use in simple remote terminals is quite attractive. It would be also a suitable approach for system-on-chip implementations, reducing the complexity of the software and the processor load in CPU-based nodes.

Due to the scalability of CANOpen, and the scalability of the draft recommendation, each node in the network can implement only the services required for its operation, which allows for very simple hardware implementations. Considering the benefits of such a design, ESA-ESTEC has internally initiated this development. The IP-Core developed (named “CANopy”) is scalable and allows for simplified implementations of the protocol.

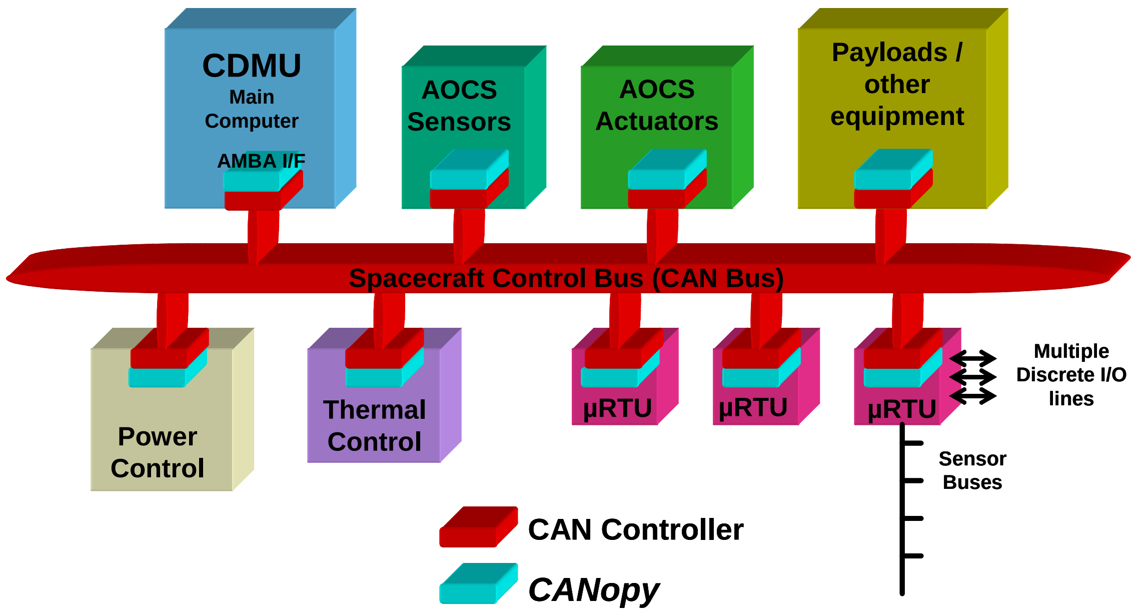

Figure 2. CANopy in system context

5. CANopy: CANopen Core for Space

CANopy implements the basic features of CANopen protocol and the extra features defined in the CAN Working Group draft recommendation. Its main intention is to serve as the higher layer protocol engine in CPU-less nodes like the µRTUs shown in *Figure 2*. However, as it can be seen also in *Figure 2*, the core can also be integrated in other nodes of the spacecraft data handling system.

5.1. CANopy interfaces

In order to allow both stand-alone and integrated system-on-chip uses of the core, two different interfaces have been developed as shown in *Figure 3*.

An AMBA APB interface, allows the integration of the CANopy within system-on-chip developments based on the AMBA architecture (like LEON processor). The Object Dictionary of the node is accessed as a memory-like interface, mapped in the processor memory map. The OD can be accessed to read or change the device configuration, and to read or write process data.

A general I/O Interface is also available for the use of the core in stand-alone CPU-less implementations. In this case, the process data is mapped directly from the object dictionary to these digital I/O lines, with no software involved.

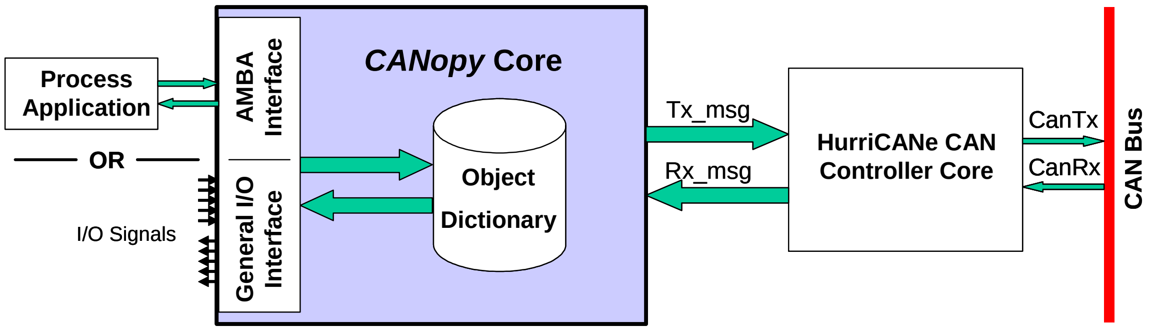

Figure 3. CANopy Interfaces

On the CAN Controller side, the core is designed to interface with the ESA HurriCANe CAN Controller IP-Core, as shown in *Figure 3*. However, it can be adapted to other CAN controller interfaces. This interface is quite simple and it basically allows sending CAN messages over the bus one by one via the tx_msg port, and getting the received message in the rx_msg port whenever the rx_completed flag is activated.

5.2. CANopy vs. CANopen features

CANopy implements the following CANopen features:

- Object Dictionary. With data types limited to 32 bits size or smaller.
- SDO Server. Expedited transfer only (not segmented or block).
- Configurable number of Transmit and Receive PDOs
    - Synchronous/Asynchronous
    - Cyclic/Acyclic
    - Timer and Remote Request
- NMT State Machine
- Heartbeat Producer/Consumer
- Sync Producer/Consumer

These extra non-CANopen features are defined in the CAN Bus draft recommendation and will also be implemented by CANopy:

- Redundancy Management (Slave)
- Time Distribution
- Large Data Unit Transfer

The core also implements an output buffer, where the CAN messages are inserted ordered by their priority. This should avoid the typical priority inversion problem that occurs in some CAN controllers.

5.3. Architecture

The internal architecture of CANopy is shown in *Figure 4*. The core is in charge of preparing all messages to be transmitted according to the current communication profile defined in the object dictionary. It will also store all the data received in the corresponding data area of the object dictionary according to the defined communication profile.

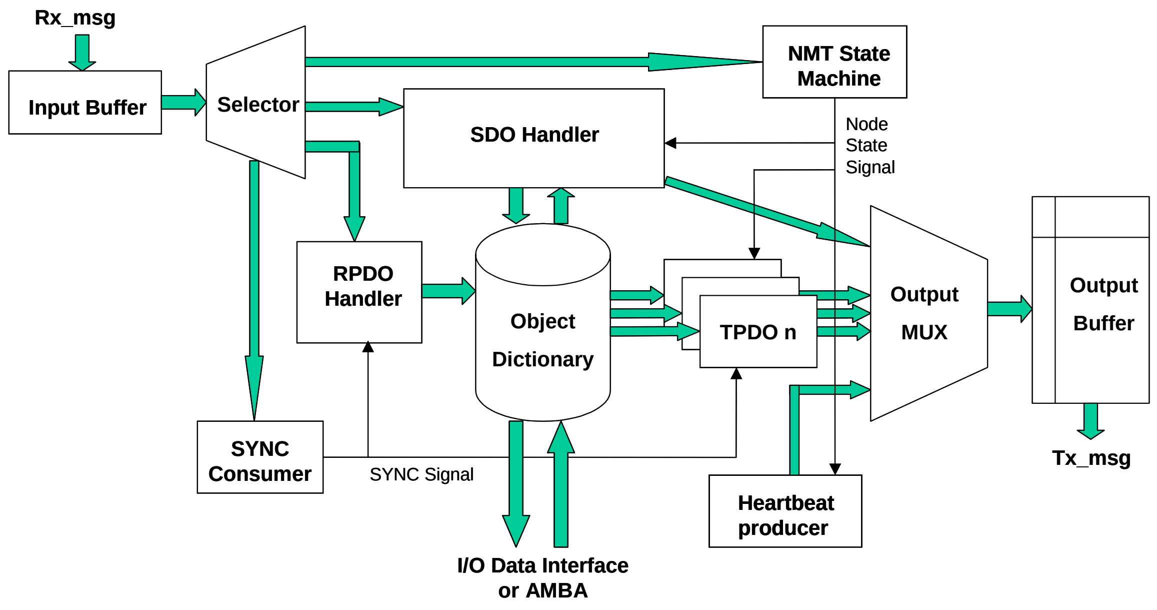

Figure 4. CANopy Architecture

For example, when an RPDO message (*Rx_msg*) is received by the CAN Controller, it is passed on to the CANopy *Input buffer*. The *Selector* will verify that the Identifier of the received message actually corresponds to an RPDO that has to be handled by this device and passes it on to the *RPDO Handler*. This entity will finally update the corresponding object values in the *Object Dictionary*. Depending on the device configuration, the data received will be passed directly to the I/O Data Interface, or just remain in the *Object Dictionary* until it is read by the application (e.g. via the AMBA interface).

If instead, the message received is a SYNC, the *Selector* will pass it on to the *SYNC consumer*, which will activate an internal SYNC Signal that will trigger any event required. In particular, if we have two TPDOs programmed to be transmitted when the SYNC is received, they will be triggered and the TPDOs messages will be queued for transmission in the *Output Buffer*, ordered by their priority.

5.4. Priority ordered output buffer

CANopy implements an output buffer with capacity for several CAN messages (configurable), that are inserted ordered by their priority. The objective of having such a buffer is to avoid a potential issue that can occur with certain CAN controllers called “Priority Inversion”. Priority inversion may occur in one node when the CAN controller implements only one transmit buffer. A low priority message stored in the buffer could prevent a message of higher priority from being transmitted over the bus.

However, implementing this buffer outside the CAN controller is not enough. The controller must provide the possibility to clear the transmit buffer if the arbitration was lost. CANopy would then put the low priority message back in the queue and pass the highest priority message waiting to the CAN controller.

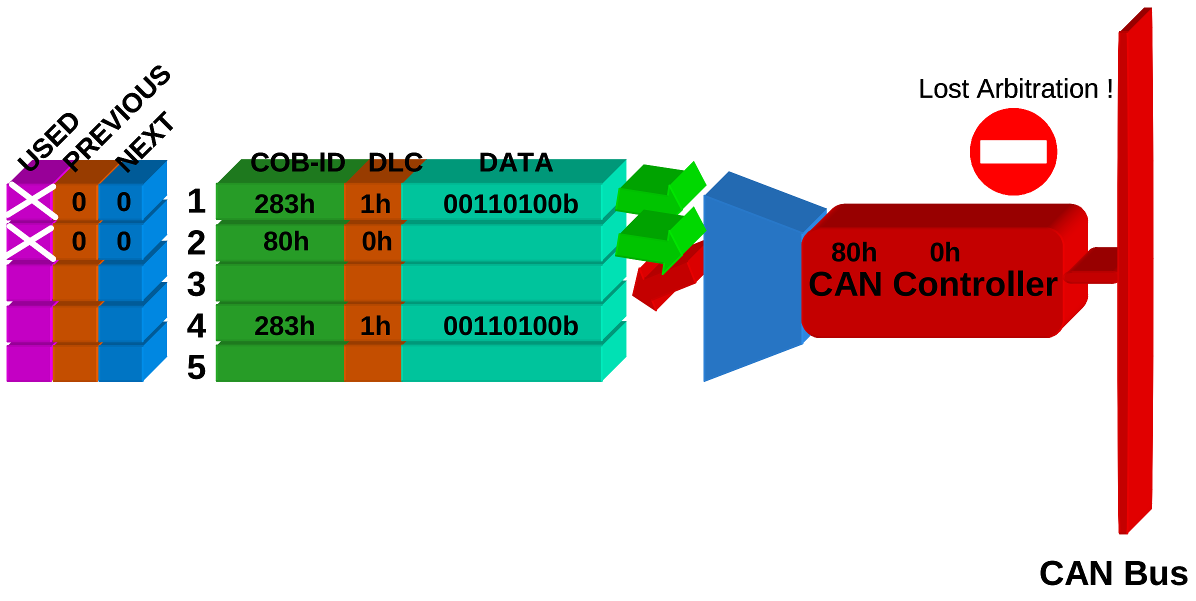

Figure 5. CANopy priority ordered output buffer

In *Figure 5* an example is shown where message with COB-ID 283h is attempted for transmission first, but it lost arbitration. In the meantime, a higher priority message with COB-ID 80h arrived in the buffer. The lower priority message must then be taken back from the CAN controller into the buffer again, and the higher priority message is given to the CAN controller for its transmission, avoiding the priority inversion issue.

5.5. NMT state machine

CANopy also implements the standard CANopen NMT (network management) state machine as shown in *Figure 6*. The CANopen node control services permit to switch between the states of the node’s state machine. After a reset, the node will go through the Initialisation state directly to Pre-Operational. This is the moment for the device configuration via SDOs. PDOs are not yet permitted. Via the NMT messages, that are sent by the NMT master, the node can start and go to Operational (normal device operation state), stop and go to Stopped state (no communication allowed, except NMT), switch back to Pre-operational state, or reset and go back to Initialisation.

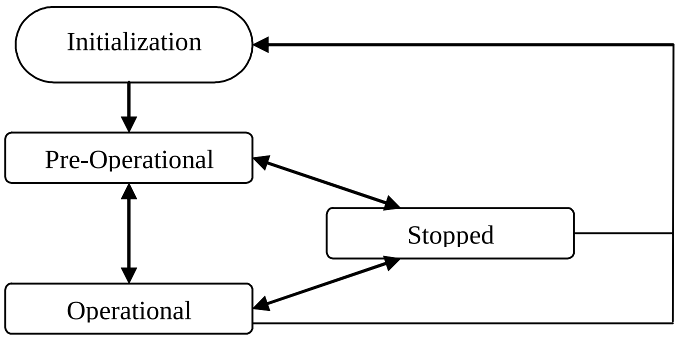

Figure 6. CANopen NMT State diagram

5.6. Configuration options

In CANopy, a VHDL package called "CanOpen" is used to define all the configuration parameters of the core using constants. This permits a very easy configuration of the core.

All the sizing parameters are defined there: the size of buffers, sizes of data types used, number of objects, basic COB-Ids, etc. Also the whole Object Dictionary is defined in that package: Objects Index, Sub-index, Initial values, and access type; As well as the number of TPDOs and RPDOs available.

The required number of blocks needed for the management of TPDOs and RPDOs, etc. are generated according to those parameters.

This configuration file is defined in a way that it can be easily generated automatically (e.g. with a script) from the device Electronic Data Sheet (EDS).

6. Current status and preliminary synthesis results

Current version (1.3) is verified with a simulation testbench, and has been synthesised using Synplify Pro for Actel FPGA RT54SX72S-1. It is suitable for use as stand-alone in small RTUs. It has the following characteristics implemented:

- Object Dictionary of 32 objects
- SDO Server
- NMT State machine
- 2 TPDOs, no RPDOs implemented
- 5 messages Priority ordered output buffer
- HurriCANe CAN Controller integrated

The full design (CANopy+HurriCANe) occupies 59% of the Actel FPGA. 23% corresponds to HurriCANe controller and 36% to CANopy.

A newer version (1.4) including RPDOs, 60 objects in the object dictionary, timer implementation to allow heartbeat generation, time triggered PDOs, a SYNC producer, etc. is currently under simulation tests, and no synthesis results are still available.

7. Conclusions and future work

This development has demonstrated that a basic implementation of CANopen in hardware is feasible and does not require very big size FPGAs (while the actual occupation can be highly dependant on the size of the object dictionary implemented). Simple implementations with a small object dictionary can fit in small (Actel 54SX72) Rad-Hard FPGAs. More complex implementations may require the object dictionary to be implemented in an external memory.

It is clear is that a development like CANopy needs to be highly configurable in order to fit all the different needs. Scalability is a key factor for CANopy success.

CANopy can be useful for Space projects using CAN bus allowing simple remote terminals to run the higher layer protocol without any software, and reducing the processor load for handling communications when integrated in system-on-chips.

The future work to be performed will include the LDUT protocol, the Time Distribution protocol and the Redundancy management. The final core is expected to be publicly available in Q4-2005.

In this paper we have focused on some particular interesting points of the design. The datasheet of CANopy will provide more details ([8]).

8. References

[1] C. Plummer, P. Roos and L. Stagnaro. CAN Bus as Spacecraft Onboard Bus. DASIA 2003 Conference proceedings.

[2] F. Tortosa López, P. Roos, L. Stagnaro, C. Plummer, B. Storni. The CAN Bus in Spacecraft On Board Applications. DASIA 2004 Conference Proceedings.

[3] ISO11898-1:2003. Road vehicles – Controller Area Network (CAN) Part 1: Data link layer and physical signalling.

[4] ISO 11898-2:2003. Road vehicles – Controller Area Network (CAN) Part 2: High speed medium access unit.

[5] Controller Area Network (CAN) Specification version 2.0 Part B, 1991, BOSH GmbH, Stuttgart, Germany.

[6] Draft proposal CAN ECSS-E50-xx-2.1. CAN Working Group.

[7] CAN Working Group website: ftp://ftp.estec.esa.nl/pub/ws/wsd/CAN/CAN-WG/index.htm

[8] CANopy Website: ftp://ftp.estec.esa.nl/pub/ws/wsd/CAN/CAN-WG/canopy.htm

[9] CAN in Space discussion forum: http://groups.yahoo.com/group/CAN_Space/

[10] CANopen Application Layer and Communication Profile. CiA Draft Standard 301. Version 4.02. CAN in Automation e. V. http://www.can-cia.de

[11] CANopen Device Profile for I/O Modules. CiA Draft Standard 401. Version 2.1.

[12] CANopen Device Profile for Drives and Motion Control. CiA DSP 402. Version 2.0.

[13] KVASER CAN Education: http://www.kvaser.com/index.htm

[14] Etschberger, K.: Controller Area Network; IXXAT Pr.; 2001; ISBN 3-00-007376-00

[15] Etschberger, K.: CAN-based Higher Layer Protocols and Profiles; Proceedings of the 4th International CAN Conference; CAN-in-Automation; Berlin; 1997.

[16] Lawrenz, W.: CAN System Engineering, from theory to practical applications; Springer-Verlag; 1997; ISBN 0-387-94939-9.

[17] Barbosa, M.: CANopen Implementation; Research Studies Press ltd.; 2000; ISBN 0-86380-247-8.

[18] ESA IP Cores web: http://www.estec.esa.nl/microelectronics/core/corepage.html

[19] Leen, G. and Heffernan, D. : Time-triggered Controller Area Network, IEEE Computing and Control Engineering Journal. Vol. 12, Issue 6, Dec. 2001, pp. 245-256.