Application and experience of CAN as a low cost OBDH bus system MAPLD 2004, Washington D.C. USA, 8th – 10th September, 2004

A. M. Woodroffe and P. Madle

Surrey Satellite Technology Ltd, University of Surrey, Guildford, GU2 7XH, UK.
Email: a.woodroffe@sstl.co.uk p.madle@sstl.co.uk
Phone: +44 1483 689278
Fax: +44 1483 689503

Abstract

This paper gives an overview of Surrey Satellite Technology Ltd. (SSTL) use of CAN bus on its recent missions. It gives a description of the SSTL CAN topology and goes into depth concerning software protocols. The paper also covers chipsets (including COTS) used in order to implement CAN in space.

1. Introduction

The Controller Area Network (CAN) is a bus system used for the communication of discrete packets of information across a distributed electronic system. It is ideally suited for real time commanding.

CAN was adopted by the automotive industry in the late 80's to handle the increasing connectivity requirements of in-car electronic systems. CAN was originally developed by Bosch and is now formalised in ISO-11898 Parts 1 and 2.
CAN features include prioritization of messages, configuration flexibility, multicast reception with time synchronization, system wide data consistency, multi-master and automatic retransmission of corrupted messages.

2. SSTL CAN Implementation

SSTL has been using CAN as an on-board telemetry/telecommand bus since 1995. Initially CAN was used with a centralised TTC system on FASAT-Alpha/Bravo. SSTL then migrated to a distributed CAN solution, as commonly used in the automotive industry. All TTC functions have been exclusively implemented using the distributed CAN bus on 6 LEO missions (UoSAT-12, SNAP-1, AISAT-1, UKDMC, NigeriaSAT-1 & BilSAT-1). The CAN topology featured in figure 1 has been tried and tested:

Nodes within the spacecraft connect to both primary and redundant buses via a latching relay. On power up, the relay of a node switches to communicate on the primary bus. If a node does not receive a CAN message for 5 minutes, it assumes bus failure and switches to the redundant bus. Modules connected to the bus fall into 2 categories:

- A data processing module such as an On Board Computer (OBC) has 2 connections to the bus. One is through an 8-bit CAN microcontroller which provides telemetry when the main processor is off. The other is via a CAN peripheral connected to the main processor.

- Less intelligent modules are connected to the bus via the 8-bit microcontroller which performs all the control/telemetry gathering needed.

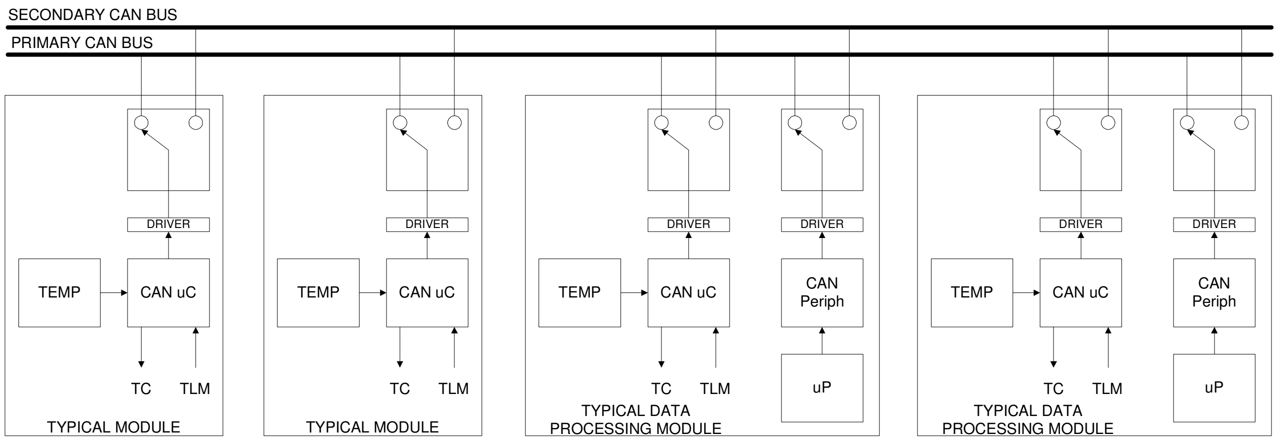

Figure 1 – CAN Bus Architecture

This architecture ensures the survival of the mission upon a single hardware failure anywhere in the system. Failures of the transceivers and other CAN interface hardware are accounted for by incorporating complete hardware redundancy.

2.1 CAN SU Protocol

SSTL has developed its own relatively simple higher layer protocol on top of CAN. CAN for spacecraft usage (CAN-SU) forces peer to peer addressing and is optimised for telemetry, telecommand and buffer transfer. The development of CAN-SU was driven by SSTL practices and had the following requirements:

- Repeat build of spacecraft sub-systems for different missions
- Large volume of telemetry available
- Scalable platforms

At the time of CAN-SU's creation, 29 bit CAN identifiers were being introduced and therefore it is limited to an 11 bit identifier. This has proven to be a limiting factor to the way the CAN is used on SSTL spacecraft.

The CAN-SU protocol splits a CAN frame up in the following way (with the exception of the buffer transfer) as shown in Figure 2 below.

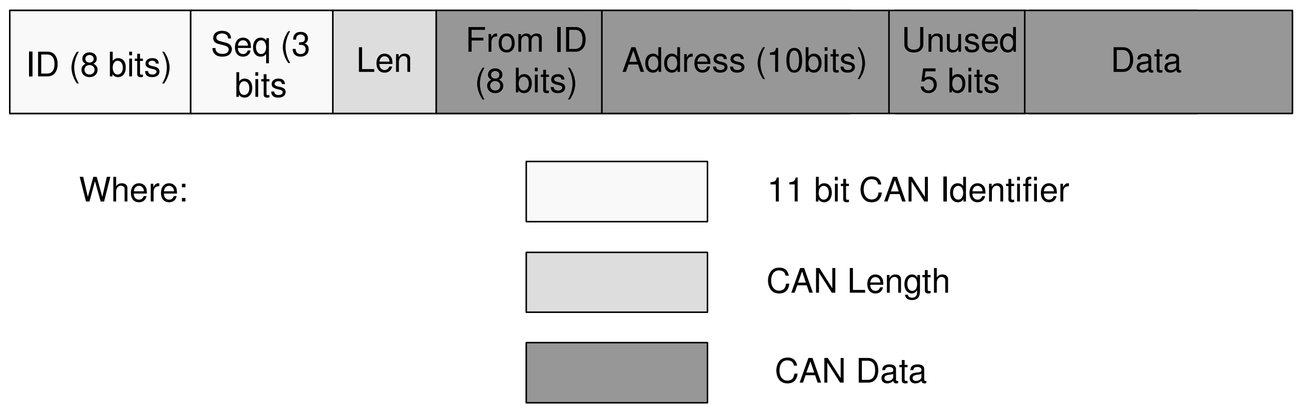

Figure 2 – CAN-SU protocol in a CAN frame

The 11 bit CAN identifier field is split into 2 parts. The first part is the ID of a task running on a processor (eg AOCS task), the other being a sequence number that is used in the buffer transfers. Since the CAN-SU identifier field is only 8 bits long, the maximum number of addressable processes in the system is 256. Not wanting to limit the amount of telemetry channels on the spacecraft to 256 the decision was made to use the CAN data field to hold an address field. This specifies a telemetry/telecommand channel giving a total of $256*2^{10}$ possible telemetry channels across the spacecraft and 2^{10} channels per task. Since the addressing of telemetry and telecommands across the CAN uses not only the CAN ID field but also part of the CAN Data field, the use of remote frames[2] is not possible. CAN-SU therefore only makes use of CAN Data frames[2].

The protocol forces peer to peer addressing by featuring the “From ID” in the CAN data field, this allows the receiving node to know where to acknowledge the request.

The “C” field specifies the CAN-SU message type (eg. telemetry request, telemetry response etc). It is excluded in some packets of a buffer transfer in order to maximise data throughput. The packet type is then determined by the sequence number within the CAN ID field. The protocol features several message types some of which are detailed in the following sections.

2.1.1 Telecommand Exchange

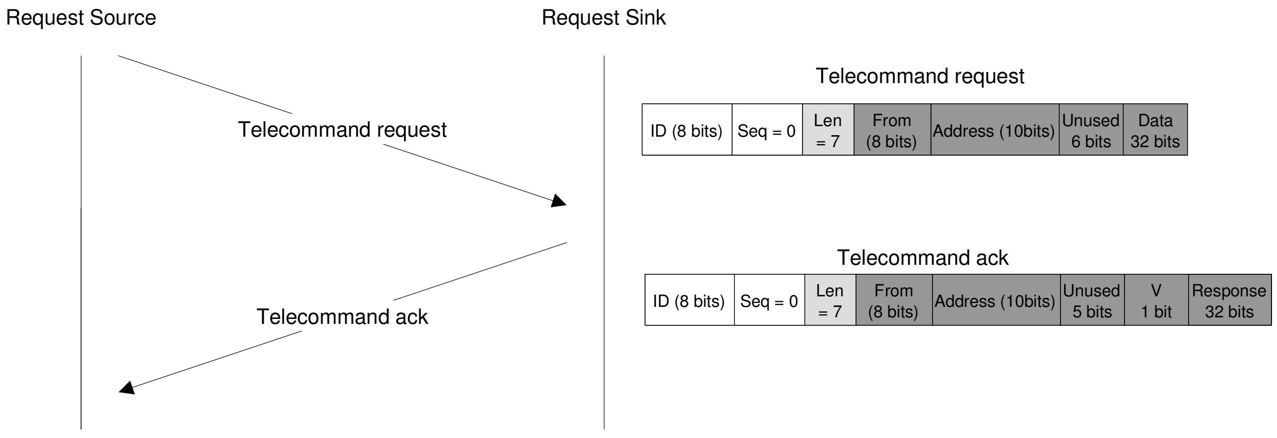

Figure 3 – Example of CAN-SU Telecommand Exchange

The telecommand exchange (Figure 3) is a simple request/acknowledge transfer using the CAN Data frame2 message format. Since the address of the telecommand is comprised of both the ID field and the address in the data field and that a processor task typically uses 1 ID, overruns of the receiving hardware buffer can occur, therefore an acknowledgement message is necessary. The data field of the telecommand request comprises of an address and the data or argument related to that telecommand address. The telecommand acknowledgement includes a response field featuring any necessary data response.

2.1.2 Telemetry Exchange

The exchange of telemetry is illustrated as shown in the following diagram:

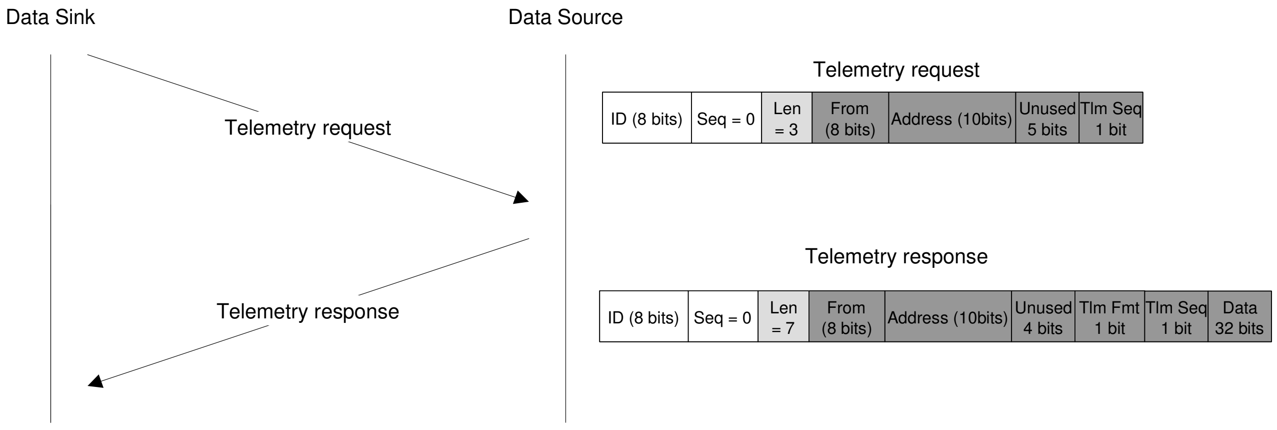

Figure 4 – Example of CAN-SU Telemetry Exchange

Telemetry is retrieved using a simple request/response method (Figure 4). Once again using CAN Data frames2, the request packet features a 10 bit address and the response features up to 32 bits of data.

2.1.3 Buffer transfer

CAN has a small data payload per packet (8 bytes max) which minimises bus latency. Therefore, it is not well suited to large data exchanges between nodes. Therefore, CAN-SU incorporates a buffer transfer scheme that speeds up large data transfers. This is illustrated below:

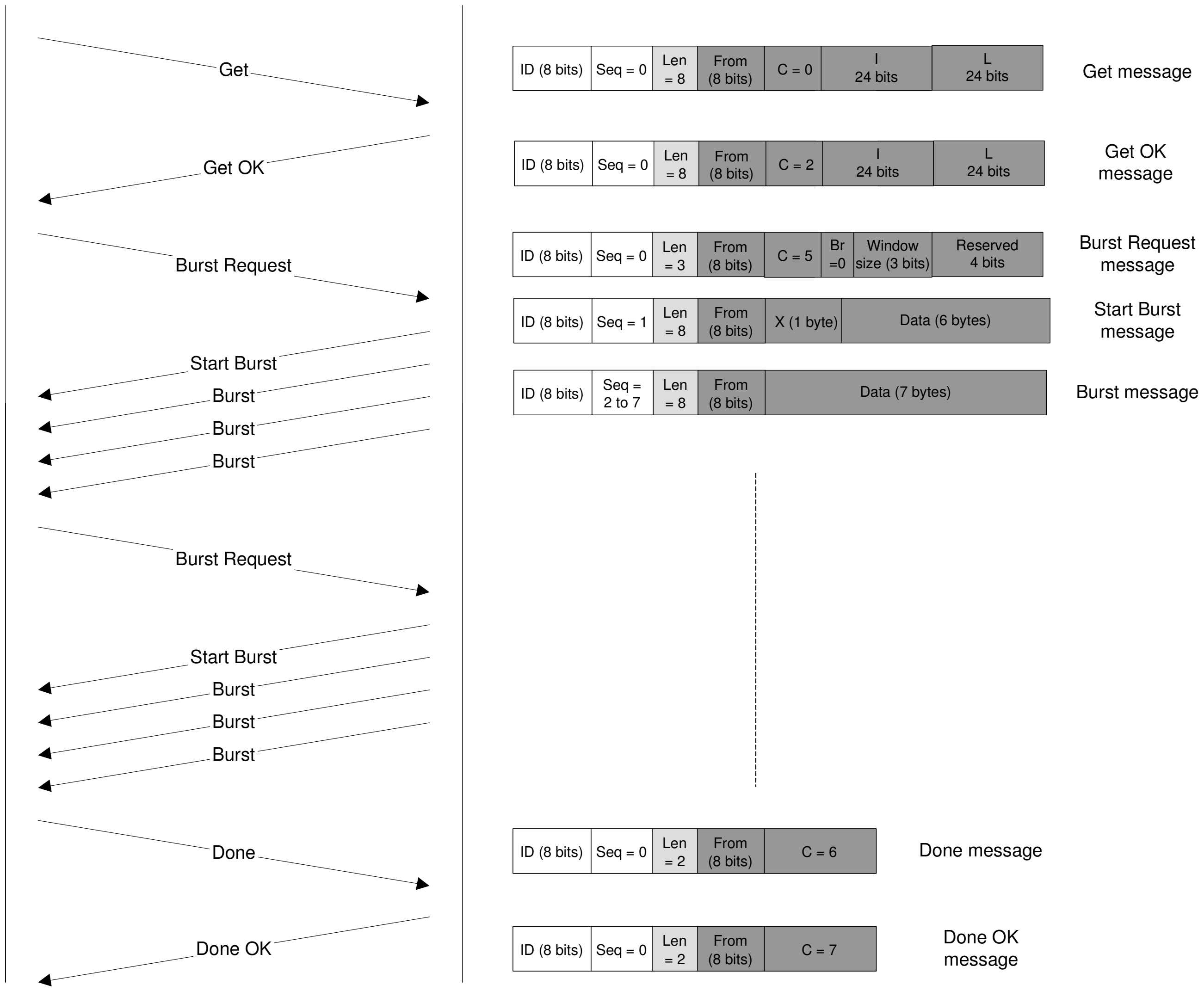
Get
Get OK
Burst Request
Start Burst
Burst
Burst
Burst
Burst Request
Start Burst
Burst
Burst
Burst
Done
Done OK
ID (8 bits) | Seq = 0 | Len = 8 | From (8 bits) | C = 0 | I 24 bits | L 24 bits
Get message
ID (8 bits) | Seq = 0 | Len = 8 | From (8 bits) | C = 2 | I 24 bits | L 24 bits
Get OK message
ID (8 bits) | Seq = 0 | Len = 3 | From (8 bits) | C = 5 | Br =0 | Window size (3 bits) | Reserved 4 bits
Burst Request message
ID (8 bits) | Seq = 1 | Len = 8 | From (8 bits) | X (1 byte) | Data (6 bytes)
Start Burst message
ID (8 bits) | Seq = 2 to 7 | Len = 8 | From (8 bits) | Data (7 bytes)
Burst message
ID (8 bits) | Seq = 0 | Len = 2 | From (8 bits) | C = 6
Done message
ID (8 bits) | Seq = 0 | Len = 2 | From (8 bits) | C = 7
Done OK message

Figure 5 – Example of CAN-SU Buffer Transfer

The transfer is initiated by either a put or a get request to/from the data source. This is acknowledged by the corresponding sink/source. The get request features a 24 bit buffer identifier and a 24 bit length. Next in the transfer is the burst request. This features the window size (number of packets in a burst). This packet also features the 1 bit “Br” burst number used to keep sync of the bursts (in the “X” byte). The burst features a start burst and up to 6 burst packets. A burst packet features 7 bytes of data and the from identifier. After as many bursts are transferred as are necessary for the transfer, the session is terminated by a Done and Done OK handshake.

2.2 CAN Hardware

2.2.1 COTS CAN Solution

SSTL specialise in providing low cost, predominantly LEO missions to a variety of civil and military markets. It has been the ongoing philosophy to fly exclusively commercial off the shelf technology (COTS) while mitigating the risk of radiation effects by module level redundancy and passive fail-safe system design. This is possible as the LEO orbits used range between 600Km and 1000Km where the radiation effects are relatively benign.

The total ionising radiation dose is in the region of 1Krad per year (at the component level with 5mm of aluminium shielding), SEUs occur at a rate of approximately 1 SEU per Mbyte of SRAM per day (from empirical observations) and observed SEL events are very rare, with only 3 or 4 suspected cases in SSTL's history.

It has been found that almost all CMOS COTS electronics will survive 10Krads with no significant degradation in performance. SELs are mitigated by over current trips on the power system and cold redundant modules. And SEUs are mitigated by EDAC where possible. Further SEU protection provided by the design of the satellite system to be safe should the on-board computer crash. For example, the thermal control is passive and will work with the satellite in any attitude.

The use of COTS components allows SSTL to rapidly utilise new technology as it enters to commercial market. SSTL first flew CAN bus technology in 1996 and now uses CAN as the telemetry and tele-command bus for all missions. See table 1 for components flown:

Components flown	**Number of Missions Flown**
Philips CAN Transceiver: Physical CAN driver, current production	4
Phillips PCA82C250: Physical CAN driver, going obsolete	10
Philips P87C592: CAN microcontroller, obsolete	10
Philips CAN 8-bit peripheral, current production	4
Philips PCA82C200: CAN 8-bit peripheral, obsolete	6
Infineon: 8-bit CAN microcontroller (A/D, PWM etc. 8051), current production	6
Microchip CAN SPI peripheral	4

Table 1 – COTS CAN components used on recent missions

Total numbers of a COTS component flown can be substantial. For example, the 8-bit microcontroller is used on all 30 to 40 modules in a spacecraft, so over the five year lifetime of a typical SSTL satellite, several hundred orbit-years of use is collected. This rapid accumulation of failure-free orbit years suggests that COTS is a viable approach to benign LEO missions.

2.2.2 Radiation Tolerant CAN Solution (RadCAN)

Recently SSTL has started moving out of the benign environment LEO missions. As part of the British National Space Centre funding, a small GEO platform program is being developed under the project GEMINI. In order to meet the tougher radiation environment and higher reliability (as GEMINI has deployable panels and therefore no passive fail-safe attitude) A COTS approach is no longer applicable and a more traditional space industry solution was required.

As SSTL has a lot of experience with the CAN bus, it was decide to develop a Radiation tolerant CAN node – RadCAN. This was to allow quicker migration of current designs between the various environments as required.

Therefore, the single chip CAN microcontroller functionality had to be re-created with a discrete 8051 microcontroller, memory, EPROM, ADC and FPGA, as shown in figure 5.1 below.

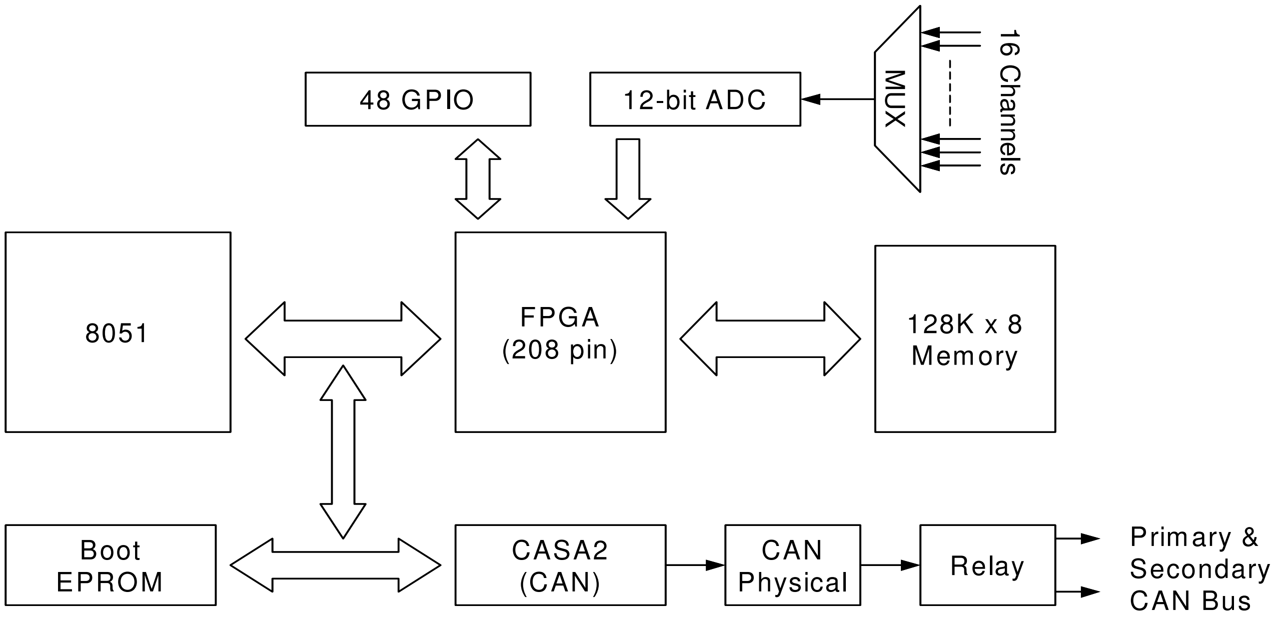

Figure 6 – Discrete elements of the RadCAN solution

Using traditional space approved components a Latch-up immune, highly SEU tolerant, 100Krad solution was produced. However, there is a significant increase in design complexity and size. The physical impact of migrating from COTS to radiation tolerant is shown in Table 2.

	COTS microcontroller	**RadCAN**
Size	1”sq	>6”sq
Mass	<5g	>50g
Power	<0.75W (max), 0.20W (nominal)	Estimate 1W
Total Dose	<10Krad	100Krads
SEE Protection	None	Highly SEU tolerant, latch-up immune
Component Cost	<$10	>$15000
Export Issues	None	Full ITAR

Table 2 – Comparison of a Radiation Tolerant to COTS CAN solution

3 Future Work

3.1 Protocols

CAN-SU has been tried and tested over a large number of missions and has been proved robust. It does not however make the most of CAN. The use of 29 bit identifiers could increase the amount of data available per telemetry request and also make use of the RTR bit and other features unavailable through CAN-SU. Future work on CAN protocols at SSTL could also concentrate on CANOpen.

3.2 Hardware

SSTL is currently looking at the latest generation of 8-bit microcontrollers to replace the current infineon device. Initials tests of a Microchip PIC device have looked promising with the 3.3V core providing a significant power saving. The processor can also be clocked up to 40MHz providing up to 3.5MIPS if required (compared to the infineons 1MIPS).

As SSTL moves out of purely LEO missions, the need for more robust, radiation tolerant systems increases. The next generation of RadCAN is being developed as a System on a chip solution (SoC), integrating VHDL IP cores for a micro-controller, CAN core, memory and EDAC in a single FPGA as shown in figure 7. The current FPGA baseline is the million gate radiation tolerant version of the Actel AX.

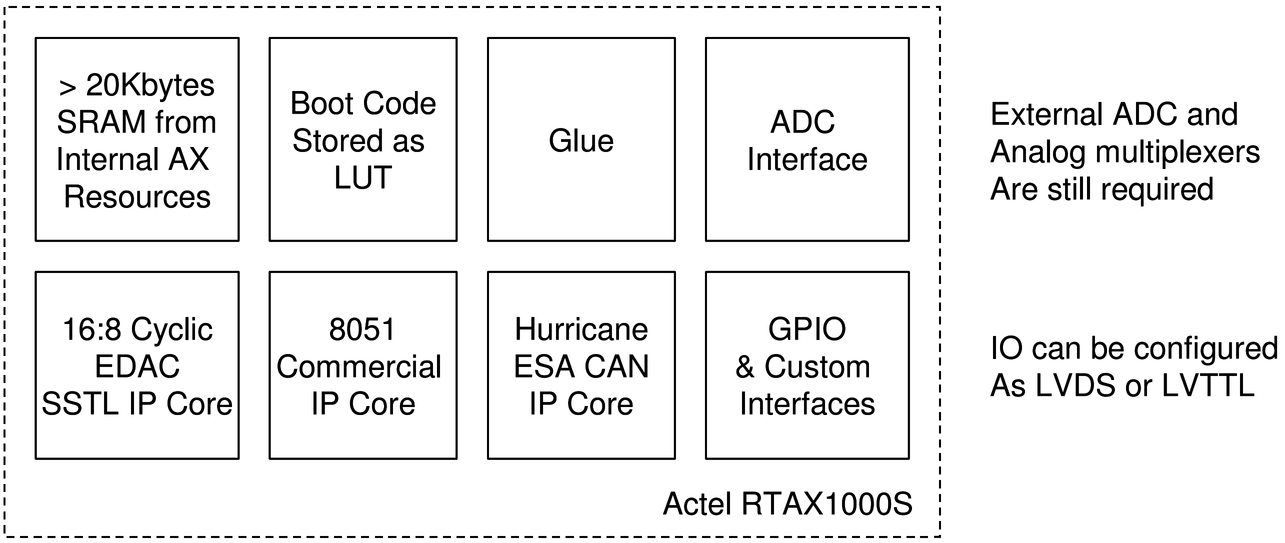
> 20Kbytes SRAM from Internal AX Resources
Boot Code Stored as LUT
Glue
ADC Interface
External ADC and Analog multiplexers Are still required
16:8 Cyclic EDAC SSTL IP Core
8051 Commercial IP Core
Hurricane ESA CAN IP Core
GPIO & Custom Interfaces
IO can be configured As LVDS or LVTTL
Actel RTAX1000S

Figure 7 – IP cores to replace RadCAN

For applications not requiring microcontroller functionality, RadCAN Lite is being developed as just the CAN core with a simple state-machine for telemetry and telecommand services. This design would be targeted at the smaller and cheaper Actel RTSXA range of FPGAs.

4 Conclusion

With 10’s of orbit years of use on SSTL satellites, COTS CAN has been proven to be a suitable architecture for the radiation benign LEO missions. The combination of module redundancy and fault tolerant system design has resulted in no observed failures of the CAN bus or it’s constituent components.

With the developments outlined above, the CAN bus will be an attractive alternative to more traditional space bus architecture for all missions.

5 References

1. http://www.caen.it/micro/rd_casta.html: a Rad-Hard 8051 microcontroller with on chip CAN hardware.
2. http://www.can.bosch.com/: bosch’s CAN page
3. http://www.can.bosch.com/docu/can2spec.pdf : the Bosch CAN 2.0 B specification
4. http://www.can-cia.org/: CAN in automation
5. http://www.vector-informatik.com/canlist/: an informative CAN mailing list