Design and Validation of an Innovative Data Bus Architecture for CubeSats

van der Linden, Steven; Bouwmeester, Jasper; Povolac, A.

Publication date
2016

Document Version
Final published version

Published in
Proceedings of the Reinventing Space Conference 2016

Citation (APA)
van der Linden, S., Bouwmeester, J., & Povolac, A. (2016). Design and Validation of an Innovative Data Bus Architecture for CubeSats. In *Proceedings of the Reinventing Space Conference 2016: London, UK* (pp. 1-13). Article BIS-RS-2016-10

Reinventing Space Conference
BIS-RS-2016-10

Design and Validation of an Innovative Data Bus Architecture for CubeSats

Stefan van der Linden
Delft University of Technology

Jasper Bouwmeester
Delft University of Technology

Ales Povalac
Brno University of Technology

14th Reinventing Space Conference
24-27 October 2016
London, UK

BIS-RS-2016-10

Design and Validation of an Innovative Data Bus Architecture for CubeSats

Stefan van der Linden, Jasper Bouwmeester
Delft University of Technology
Kluyverweg 1, 2629 HS, Delft, The Netherlands
s.p.vanderlinden@student.tudelft.nl, jasper.bouwmeester@tudelft.nl

Ales Povalac
Brno University of Technology
Technicka 12, 616 00, Brno, Czech Republic
povalac@feec.vutbr.cz

ABSTRACT

Since the first successful CubeSat missions in the early 2000s, payloads for this form factor have emerged and have increased in technical performance level. This trend is likely to continue in the near future. However, despite the subsequent increase in data load and the increasing modularity of components, there are no clear trends for a new electro-mechanical interface standard. The only widely adopted data bus in CubeSats, I^2C, is limited in terms of data rate and reliability. Custom solutions overcoming these limitations are generally not well documented, and especially implementation results in CubeSats are lacking. Therefore, there exists a need to increase the performance and reliability of the CubeSat bus platform. This paper proposes an innovative CubeSat data bus architecture, including performance and reliability tests.

The first need is to increase the feasible data rates to be compatible with future large-data payloads. Secondly, the system's reliability must be increased compared to the current I^2C standard, as many launched CubeSats using this data bus experience severe problems such as bus lockups and even a few catastrophic failures.

The concept proposed in this paper is to separate the main bus used for telemetry and command from the data bus used for payload data, which provides room for optimising the performance of both buses. The selected bus technology standards used to drive the hardware were found through a survey and subsequent trade-off of available serial bus standards. For the main bus, this trade-off results in either a CAN bus or RS485 bus to both increase the robustness of the internal network and potential data throughput. For the payload bus, a USB-based bus is selected to provide a high data rate with increased reliability compared to often-used standards. The combination of both options optimises performance while keeping electrical power consumption at a minimum. Making use of the common modular designs of CubeSats and recent developments in the respective data bus technologies, a flexible, robust and high performance data bus architecture is devised. A practical setup simulating a CubeSat with multiple realistic subsystems generating pseudo data is used to validate the operations of such a data bus. To find the maximum capacity of the network, multiple subsystems are connected with varying high and low data rates, thereby simulating current typical CubeSat subsystems and potential future payloads requiring high capacity data networks. Furthermore, methodologies are developed for implementation and qualification of the proposed bus in future CubeSat missions.

KEYWORDS: **CUBESAT, DATA BUS, CAN, I^2C, USB, RS485**

INTRODUCTION

Ever since the introduction of the CubeSat standard in 2000[1], it has been proven to be a versatile platform suitable for many different types of missions. Starting as primarily an educational standard, it has seen a growth in commercial applications as well as use in scientific missions, mainly starting in the late 2000s.

However, despite the large growth in both complexity and scientific data output from both technology demonstration missions and operation scientific missions[2], development in Cubesats' on-board data handling has been limited. Especially the number of changes to the serial data bus, the electronics carrying commands, telemetry and data internally between subsystems, has been low. Bouwmeester and Guo[3]

have found that the Inter-Integrated Circuit (I^2C) bus standard has especially been used as the main data bus on the majority of CubeSat missions to date.

I^2C, originally developed by Philips and now maintained by NXP Semiconductors[4], has been noted for its simplicity and high level of support in both integrated circuits (ICs) and microcontrollers. Moreover, the power consumption of I^2C is typically assumed to be much lower than other bus types[3].

Nevertheless, the simplicity of I^2C also means that no failure tolerance or error detection is built into the lowest Open Systems Interconnect (OSI) model layers of the protocol[5]. This has resulted in at least one catastrophic system failure and it is hypothesized that several other Cubesats have failed due to issues with this data bus[5,6]. To enable the creation of large CubeSat-based constellations or even more demanding scientific missions such as interplanetary CubeSats, the data bus must be reliable. Also the emergence of highly distributed subsystem architectures[7,8] puts severe strain and importance on the data bus due to multi-master configurations becoming more common.

Furthermore, the relatively low data rate of I^2C (100 kbit/s for *standard mode* and 400 kbit/s for *fast mode*) means that the payload data rates are also severely limited. Already, missions are omitting or complementing I^2C in favour of other higher speed buses due to the limits simply being too low[9]. It may be expected that the required data rate by CubeSat payloads will only increase. Moreover, there are technology demonstration missions being performed with high speed downlinks up to 100 Mbit/s[7].

Hence, the CubeSat data bus must be re-evaluated to ensure compatibility with the next generation of CubeSats, both in terms of reliability and performance.

This paper, consisting of two main parts, investigates a novel data bus based on other data bus standards: the Controller Area Network (CAN) bus, the RS485 bus (Recommended Standard 485) and the Universal Serial Bus (USB). The former two provide a bus to ensure command and control of subsystems (denoted as the *command and telemetry bus* or *TC bus*), while the last option is used as a high performance bus to transfer payload data (denoted as the *payload bus* or *PL bus*). The first part of this paper explains the underlying trade-off explaining the selection of CAN, RS485 and USB. The second part presents a small-scale practical realisation of the two buses in a simulated CubeSat to validate the selection and describe ways of implementing the data buses in actual missions.

SELECTION CRITERIA

The process of selecting the data bus standard for a generic future CubeSat case revolves around a central trade-off. This trade-off takes a large set of data bus standards and aids in finding the optimal architecture. Before these criteria are outlined, a couple of reference cases are first defined to help in the analysis of the different options.

Reference Cases

Figure 1 shows the distribution of transaction sizes during a single 2 s cycle of the Delfi-n3Xt On Board Computer (OBC) polling various subsystems for housekeeping data.

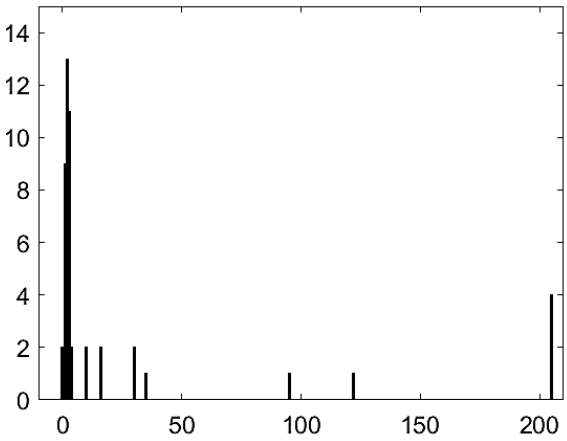

Figure 1. A histogram showing sizes of transactions in bytes during a normal housekeeping information polling cycle of Delfi-n3Xt.

It may be seen that the majority of transactions are only several bytes in size. Nevertheless, a small amount of transactions consist of a relatively large amount of bytes, going up to 205 bytes. Thus, a clear distinction can be found between small command transactions and larger, data-filled requests. Although Delfi-n3Xt did not contain payloads with an exceptionally high data load, it is expected that many in-orbit demonstration CubeSats will exhibit a similar data size distribution.

The gap between transaction sizes provides the reasoning to split up the overall data bus architecture into two main branches: one bus for telemetry and command (TC) and a second bus for any payload (PL) producing a significant amount of data.

The TC bus case (Figure 2) is defined to consist of five different subsystems, including a main OBC. To take into account an increased modularity of the subsystems, it is assumed that the bus does not feature a single central node, but rather operates in a multi-master setup.

The PL bus (Figure 3) is much simpler in setup than the TC bus, as it consists of only two nodes. Its objective is simply to take the data from a subsystem generating a relatively large data load, such as an imaging payload, and transmit it to a second subsystem handling the data. For example, the second subsystems could be a high speed radio or mass data storage. To investigate the differences between the two bus types, two trade-offs are performed: one for each bus reference case.

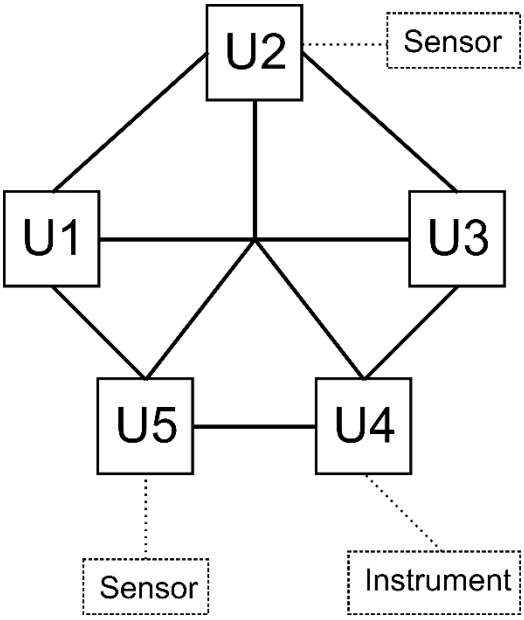

Figure 2. The TC bus reference case. Note that secondary buses connecting sensors and instruments (dashed) are not assumed to be part of the TC bus

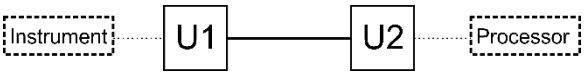

Figure 3. The PL bus reference case. As with the TC bus, the secondary nodes (dashed) are not part of the bus layout

To perform the trade-off process, requirements must be specified as a reference target for the trade-off process.

Several high level requirements can be stated, describing the criteria taken into account by the trade-off. The criteria are as follows.

Baud Rate

In terms of performance, the bus must be able to transmit both payload data and command or house-keeping data rapidly. For the former, this is naturally required due to the expected increase in the amount of data generation. For the latter, a higher data rate may be beneficial in a highly distributed bus layout. To compare the different trade-off options, the expected baud rate is selected and not the raw data rate. The baud rate is defined here as the maximum amount of signals per second and should equal the raw data rate without taking into account any protocol-layer overhead and other latencies. To emphasize the difference between the baud rate and the data rate, the baud rate is expressed as a frequency. When referring to an (effective) data rate, the units for data per unit time (e.g. kbit/s) will be used. The reason to choose the baud rate over the data rate is because the protocol-layer overhead can, in some cases, be used to transfer information (such as commands) itself. Hence, the difference between true 'useless' overhead and 'useful' overhead would become a very grey area.

I^2C is able to provide a baud rate of 400 kHz, which proved to be enough for Delfi-n3Xt. However, to provide a large enough margin for large new developments, a higher (and slightly arbitrary) minimum baud rate requirement of 800 kHz is chosen, providing double the value of I^2C. For the higher speed PL bus case, a minimum value of 1 MHz is chosen to signify the larger amount of data passing through the bus.

Power Consumption

A second measure of the bus performance is the power consumption to operate. This value comprises of an estimate of the electrical power used to power all bus hardware, with the exception of the microcontrollers of the bus nodes themselves. Both bus reference cases (TC and PL) will be analysed in separate trade-offs. The total available electrical power is severely limited in a CubeSat: typically in the order of only several Watts[3]. Accounting for this limitation, an upper limit of 1 W is set to the bus' required electrical power.

Reliability

In contrast to I^2C, the bus must have one or more built-in mechanisms to ensure reliable transmission of the data. This is especially important when considering CubeSats with longer lifetimes or higher altitudes, where single event upsets (SEUs) and other externally sourced effects are significantly more common. However, the added complexity and effort going into designing such as bus may not be worth it when other bus standards are able to provide more robust measures.

The bus' reliability can be assured, for example, by providing protection against physical effects through differential signalling and through active error or fault detection.

Table 1: The reference rubric used for the trade-off. Note the different values regarding the baud rate for the TC and PL buses.

Score	Base Baud Rate	Power	Reliability	Availability	Complexity	Data Lines
Reject	Less than 400 kHz (*TC bus*) Less than 1 MHz (*PL bus*)	The maximum total bus power is more than 1000 mW	No safety mechanisms possible	Components / documentation not available	Not possible to interface to (typical) microcontrollers	More than 7
1	400 kHz - 800 kHz (inclusive) (*TC bus*) 1 MHz - 10 MHz (inclusive) (*PL bus*)	The maximum total bus power is 500 mW to 1000 mW (inclusive)	Simple mechanisms (watchdogs) possible through modification	Components and documentation only available through highly dedicated suppliers	Large amount of additional electronic components necessary (more than 10 per node)	5 - 7 lines required
2	800 kHz - 5 MHz (inclusive) (*TC bus*) 10 MHz - 100 MHz (inclusive) (*PL bus*)	The maximum total bus power is 100 mW to 500 mW (inclusive)	Differential signalling and/or simple error detection	Components and documentation easily/widely available	Additional electronic components required for operation (less than 10 per node)	3 - 4 lines required
3	More than 5 MHz (*TC bus*) More than 100 MHz (*PL bus*)	The maximum total bus power is 0 mW to 100 mW (inclusive)	Differential signalling, error and fault detection/correct-ion	Typically built-in feature of microcontrollers	No additional components required	2 lines or less required

Availability

Staying with the original philosophy of CubeSats[1], the commercial availability and low cost are key requirements. This means the necessary hardware (and potentially software) must be available Commercially of the Shelf (COTS) and not limited under expensive licensing costs.

In a related note, relevant documentation must be publicly available at low or no cost to aid in the implementation and development of the bus.

Complexity

Since CubeSats are often implemented in the form of educational tools or as a supporting platform for small scale scientific projects, the complexity of the bus hardware and software is of importance. It is generally difficult to specify a metric for something as subjective as the complexity. Nevertheless, within this trade-off the amount of additional electronic components is used. When a system contains more components, it becomes more difficult to populate boards and integrate a satellite. Furthermore, it may be assumed that the amount of programming code increases with more components to control, especially when a data bus requires several relatively complex ICs.

Number of Data Lines

Finally, taking into account the low amount of space available inside CubeSat buses for wiring harnesses, the amount of (data) lines required by the standard is taken into account.

Trade-off Rubric

The seven criteria defined in the previous section are reflected in the selection of the trade-off criteria as shown in Table 1. Moreover, this table also defines the trade-off's rubric: the rules governing which score to assign to which characteristic of a bus. The values in the rubric have been set and fixed before the actual trade-off to prevent biasing of the values. Predefining this rubric avoids the need of defining weights for each criterion, which would introduce possible biasing. The rubric inherently includes the weighting in its design.

It must be noted that Table 1 contains a 'reject' row. If a certain data bus option meets one or more of the characteristics of this row, then the option is removed from the trade-off.

TRADE-OFF

It is impractical and mostly unnecessary to create an exhaustive list of all possible serial data bus standards. Therefore, this trade-off is limited to data bus formats and architectures as defined by their official standards and excludes off-standard modifications and configurations.

To keep the trade-off concise, the majority of data bus standards are evaluated in terms of the 'reject' criteria from the trade-off rubric.

Initial Reduction of Options

Table 2 lists a selection of serial bus standards from a multitude of sources, several of which have previously

been implemented in CubeSats, but also many have not. It shows which buses have been rejected for not meeting the minimum acceptable requirements and for what reason.

From Table 2 it may be noted that for the TC bus case, only I^2C, RS485 and CAN are left. For the PL bus, RS485, SPI, CAN, USB 2.0 remain. These remaining options are compared in more detail in the following sections.

Telemetry & Command (TC) Bus

To compare RS485 and CAN, the predefined rubric in Table 1 is used. The results are shown in Table 3.

With regards to the baud rate of I^2C (the generally well supported fast mode) equals 400 kHz, equalling 1 point in the trade-off. For RS485, this is officially defined at 1 MHz, although it can be increased to a multiple of that value when the bus lines are kept short[10]. A similar situation is found for CAN: the official standard is limited to 1 MHz, although it may be increased for short wire lengths. However, it is expected that the resultant increase will be less than with RS485. Nevertheless, this means a score of 2 for both options.

Table 2: A list of bus standards and reasons for rejection from the trade-off.

Bus Standard	Rejected for TC Bus	Rejected for PL Bus
I^2C (Fast Mode)	No	Baud rate (400 kHz) too low[4]
RS232	Baud rate (≈110 kHz[10]) too low	
RS422	Simplex version of RS485, so assumed identical to RS485	
RS485	No	No
SPI	Large amount of chip select lines required for multi-master	No
CAN	No	No
USB (2.0)	Power consumption with hub exceeds 1 W	No
USB (3.0/3.1)	Necessary COTS USB 3.x Host controller not yet available	
Firewire	Not able to interface with microcontrollers (e.g. PCI/PCIe)	
Spacewire	Low availability of COTS components, large amount of lines (8)[11]	
MIL-STD-1553	Very low COTS availability	
Ethernet	Power consumption exceeds 1 W for TC and PL bus (using the WIZnet W5100[12])	
RapidIO	Very low COTS availability	
Infiniband	Not compatible with embedded systems	
Thunderbolt	Very low COTS availability (high licensing costs)	
FlexRay	Very low COTS availability	
Local Interconnect Network	Baud rate (20 kHz) too low[13]	
OneWire	Baud rate (≈15 kHz) too low[14]	

To analyse the maximum power consumption of I^2C, several assumption have to be made. First of all, it is assumed that every node has one PCA9514 I^2C isolator/buffer[15]. A buffer or isolator is often necessary in I^2C networks consisting of many nodes to reduce the total bus capacitance, which is limited to 400 pF[4]. Moreover, the isolator function of these components makes it possible to safely remove a subsystem from the bus (i.e. powering down redundant systems) without affecting the main bus[16]. The worst case power consumption is assumed where the bus is continuously in a logic LOW state. This means all bus lines are completing a circuit and thus consuming power through its pull-up resistors. All separate SCL and SDA lines require their own pull-up resistors, meaning that when assuming typical pull-up values of 4.7 kΩ (at 3.3 V), each individual line consumes 2.3 mW. Five nodes plus the main bus lines give a total of 27.6 mW. Adding up the expected power consumption of the bus buffers, this makes a total of 85.4 mW.

Concerning RS485, a typical configuration is assumed: the built-in UART of a microcontroller provides the data to a driver/transceiver. This driver then drives the data onto the bus lines. In this case, the ST3485[17] is used as a reference, where it is assumed that its results are typical for other drivers. Since only one node will always be transmitting at the same time, the power consumption of a single transmitting node equals the overall bus power consumption. For the chosen TC reference case, it is assumed that the output of the driver is continuously equal to its default state value of 1.5 V[17]. Further assuming a standard termination load of 60 Ω (two 120 Ω resistors in parallel), the total power lost over the bus lines becomes 37.5 mW. Added to this is the passive current draw by all other nodes: 1.3 mA, adding another 21.5 mW to the total. Thus, for the full TC bus, a total of 59 mW is found.

For the CAN bus, each node is assumed to be comprised of an MCP2515 CAN controller[18] and an SN65HVD233[19] transceiver. The base current draw on one node then equals 16 mA. When a single node is transmitting, this adds another maximum of 50 mA to the total current. For the 3.3 V five node bus this results in 429 mW. These figures translate to scores of 3 and 2 for RS485 and CAN respectively.

One of the main reasons for this trade-off is the observed unreliability of I^2C which is also reflected here: only 1 point can be assigned due to its physical layer. Conversely, CAN is well-known for its reliability: it defines a physical layer with differential signalling and several fault tolerant measures. Furthermore, the CAN standard defines a protocol layer with features like built-in error detection. Although RS485 comes with differential signalling,

Table 3: The TC bus case trade-off. A higher score is better.

	Baud Rate	Power	Reliability	Availability	Complexity	Harness Lines	Total
I^2C	1	3	1	3	2	3	**13**
RS485	2	3	2	2	2	3	**14**
CAN	2	2	3	2	2	3	**14**

the lack of other pre-described safety mechanisms means it scores 2 points versus 3 for CAN.

RS485 and CAN both score the same for availability, complexity and the number of bus lines: the bus protocols both need external drivers or transceivers to function, and they both use two differential data lines to communicate. I^2C scores high for the availability as it is built-in in nearly every microcontroller. It also scores 2 points for complexity, because bus buffers/isolators are practically always required to have a functional bus.

Thus, in the end, I^2C scores 13 points, and both RS485 and CAN score 14 points. This draw is remarkable and underpins the similarity between the two standards.

Payload (PL) Bus

The PL bus is evaluated in a similar manner as the TC bus, taking the rubric table as a basis for the comparison. RS485 and CAN are again assessed as options together with SPI (Serial Peripheral Interface) and USB 2.0.

For both CAN and RS485, the configurations are assumed to be identical as in the TC case. Hence the baud rate remains the same. For SPI, there is no maximum data rate specified. However, applications are typically able to reach at least 10 MHz[20]. In turn, even though the baud rate of USB 2.0 has been standardised to 480 MHz[21], it is only found possible to interface to standard microcontrollers with 'Full Speed' devices, capable of providing a baud rate of 12 MHz[21].

With regards to power consumption, it is assumed that RS485 and CAN are applied in the same configuration as in the TC bus case. This results in the same value as in the aforementioned analysis minus the power for three subsystems, giving 46.1 mW for the former and 271 mW for the latter. The next option, SPI, is somewhat of a special option: it is supported by the vast majority of microcontrollers[20] and hence does not require any external ICs to operate. Thus, all power is consumed by internal microcontroller operations. A very low power consumption can therefore be assumed for this option. For the final option, USB 2.0, it must be noted that at least one host controller is required in the bus and one so-called peripheral (slave) controller. To be able to have a microcontroller fulfil both roles, the MAX3421E[22] is selected. This integrated circuit is able to act as either a host or peripheral in a USB connection. From the datasheet, it is found that the maximum expected power consumption of a transmitting node equals approximately 150 mW. No data is given for a listening node, but it is expected that it will be around half the value for a transmitting node, resulting in approximately 225 mW for the total bus.

Because RS485 and CAN are in the same configuration as for the TC case, the scores for all other criteria are deemed the same. For SPI, the reliability is set to low, as there are no built-in safety mechanisms, putting it on the same level as I^2C. The availability is very good however, as it is a standard interface for most microcontrollers. This also implies a high score for complexity. However, the large amount of lines required (SCLK, MISO, SOMI, CS) means it scores low on the final criterion: the number of bus lines. USB scores slightly lower on availability and complexity for similar reasons as RS485 and CAN. It scores well on reliability however as the protocol is designed around hot plugging peripherals which requires robust communications. Disregarding the two additional power lines (which are not necessary to transfer data), USB 2.0 requires only two data lines.

To finalise, USB 2.0 shows a higher score (14 points) than the other standards (all 13 points) when applied in the PL bus case.

Table 4: The PL bus case trade-off. A higher score is better.

	Baud Rate	Power	Reliability	Availability	Complexity	Harness Lines	Total
RS485	1	3	2	2	2	3	**13**
CAN	1	2	3	2	2	3	**13**
SPI	1	3	1	3	3	2	**13**
USB (2.0, Full Speed)	2	2	3	2	2	3	**14**

VALIDATION: DATA BUS SIMULATOR

As a small-scale proof of concept and to compare the estimated characteristics of the trade-off options, a data bus 'simulator' is developed. The simulator, one example configuration of which is shown in Figure 4, is setup in a similar way as a CubeSat with multiple subsystems, including an (OBC). Each subsystem consists of a Texas Instruments (TI) MSP432 low power microcontroller 'Launchpad' (MSP432P401R). The MSP432 has been chosen as the standard microcontroller for all new subsystems and satellites within the Delfi program[23]. Its 'Launchpad' configuration features an emulator/debugger and a breakout of a large collection of microcontroller pins, including pins used for I^2C, SPI and Universal Asynchronous Receiver/Transmitter (UART) communication. As the choice for microcontroller is considered to be fixed, all data bus-related components must be able to interface directly with the MSP432, resulting in a universally applicable bus. Electronically, each considered bus is implemented using the components mentioned earlier during the trade-off.

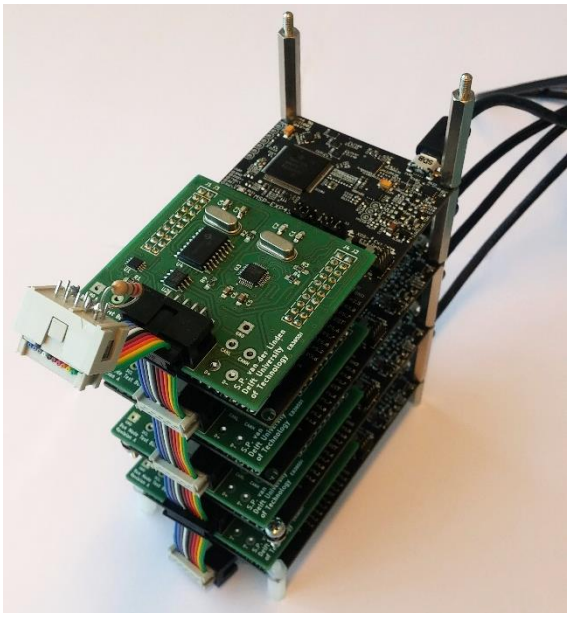

Figure 4. An example of the data bus testing setup showing four connected nodes over CAN, I^2C and USB (RS485 is on separate but similar boards).

Firstly, the I^2C bus is directly controlled by each microcontroller itself, as it is a standard serial output supported by the MSP432. To limit the total bus capacitance as required by the standard, bus buffers (PCA9514A) are added. The bus lines between the microcontroller and the buffer are powered by the local microcontroller. The main bus lines are powered by the single OBC.

Secondly, the CAN bus is implemented using the MCP2515 CAN controller with SPI interface. Additionally a bus transceiver is required to drive and buffer the differential bus. For this, the TI SN65HVD233 is used, because it supports 3.3 V operation. The MCP2515 is configured and controlled by the MSP432 through its SPI interface.

Thirdly, for USB the MAX3421E is used, which also has an SPI interface and a multi-role option: it can act as either the bus host (master) or peripheral (slave). It is impossible to communicate over USB without a host controller, thus the MAX3421E solves this problem.

Finally, although RS485 is essentially direct UART output, it requires a driver to drive the differential data lines and handle the inverse. For this purpose, the ST3485 driver is used.

The power of each individual bus was measured at idle to be able to isolate irrelevant power drains. These values are shown in Table 5 and will be used together with the values discussed in the next section to model the full buses. Note that the minimum and maximum adjusted values are assumed to be the absolute minimums or maximums, e.g. the minimum adjusted power consumption is the minimum measured consumption of each bus minus the maximum measured power consumption of the MSP432.

Table 5: Power consumption per unit of one bus node measured at idle (no bus traffic), and adjusted for the power consumption of the MSP432 Launchpad

Microcontroller Base Power Consumption [mW]			
System	**Minimum**	**Mean**	**Maximum**
MSP432	11.6	14.0	17.0
Power used by each bus when idle, adjusted for MSP432 consumption			
System	**Minimum**	**Mean**	**Maximum**
I^2C	5.2	9.1	12.3
CAN	14.1	19.5	24.5
USB	33.0	38.0	42.7
RS485	-0.4*	3.3	9.4

* The negative value is due to summing noise extremes with the power adjustment.

Measurements

This paper will look at two different metrics of each bus option:

1. The actual power consumption of each bus when nearing the maximum capacity
2. An estimate of the effective data throughput of each bus

For the power consumption, the TI EnergyTrace tool is used. This tool connects to the onboard emulator and is able to measure the current consumption with approximately 5% relative accuracy[24].

The effective throughput is measured in two ways, depending on whether the bus in question is being applied as a representation of the TC bus or of the PL bus. When the former is the case, the system designated as OBC sends special command packets: 8 bytes containing a command byte, the origin node, the target node, three parameters (default 0) and finally a CRC-16 of the preceding 6 bytes. When a subsystem receives such a packet, it first checks the CRC: if correct, it replies with an 8 byte 'ACK' command packet or a 'NAK' command packet if incorrect. During each TC bus test, the OBC will simply send a 'ping' command packet and wait for a reply. This will then be repeated without any artificial delays in between. All messages transmitted by the OBC and subsystems are generated dynamically to ensure realistic response times and processor activity. For the PL bus case, a simpler test case is considered: static but pseudo-random data is transmitted by the OBC to the other connected subsystem in one direction only to maximize the data rate.

Each individual test is only 15 s long, but repeated at least ten times to verify the values. The first 5 s is a delay where all hardware and software is initialized, but not actively transmitting or receiving. The following 10 s the data throughput test is performed. Using the EnergyTrace tool, the power consumption of the board is measured during both stages to estimate the power actually required by the bus activity. Furthermore, by counting the number of packets transmitted, the effective number of bytes may be computed in the TC bus tests. For PL bus tests, the transmitted bytes are counted directly.

Results: Telemetry & Command Bus

The first bus case being tested is the I^2C bus, which is considered only for the TC bus case. Figure 5 shows the power consumption of the OBC performing the ping requests and the generic subsystem responding to those requests. The plot shows a large difference between the OBC (blue) and the subsystem (orange) when active, but highly similar values when idle.

The minimum, maximum and mean values of both the idle state and active states of both subsystems have been computed and are shown in Table 7. Taking the mean values and applying it to the bus architecture of the TC case, a total power consumption of 121.5 mW is found, approximately 40 mW more than calculated in the simplified analysis for use in the trade-off. The average measured bit rate (Table 6) is approximately 200 kbit/s: about half the maximum capacity (baud rate) of the bus (400 kHz).

Table 6: Average measured effective bit rates excluding overhead over the 10 s intervals of the different tests with 95% confidence interval.

Bus	Average Measured Bit Rate (TC Bus) [kbit/s]	Average Measured Bit Rate (PL Bus) [kbit/s]
I^2C	200.64 ± 0.0025	*N/A*
RS485	607.17 ± 0.017	781.25 ± 0
CAN	94.90 ± 0	*N/A*
USB	*N/A*	999.50 ± 0.050

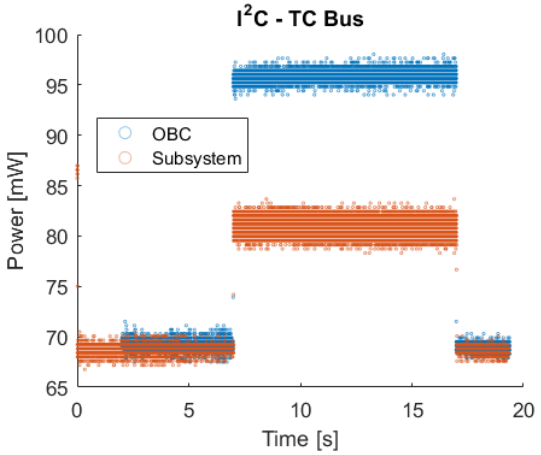

Figure 5. The raw measured power consumption of both the OBC and the generic subsystem when using I^2C, including MSP432 and other supporting systems.

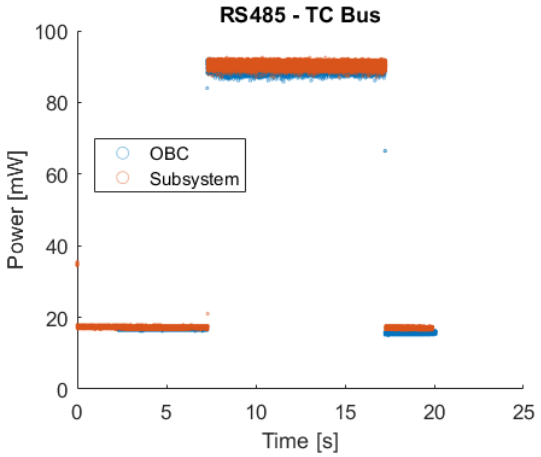

Figure 6. The power consumption of both the OBC and the generic subsystem when using RS485. The bus termination resistors are 2 x 120 Ω.

Table 7: The computed minimum, maximum and mean differences between idle and active states, and total derived power consumption by summing the difference with the fully idle power consumptions included in Table 5. All units are in milliWatts.

Bus	State	Power TC OBC			Power TC Subsystem			Power PL OBC			PL Subsystem		
		Min	Mean	Max	Min	Mean	Max	Min	Mean	Max	Min	Mean	Max
I²C	*Difference*	22.1	26.7	30.5	7.8	12.3	20.2	**N/A**					
	Total	27.3	35.9	42.8	13.0	21.4	32.5						
RS485	*Difference*	66.4	72.5	76.0	66.1	73.1	75.8	148.7	160.9	165.1	5.4	7.2	9.1
	Total	66.0	75.8	85.4	65.7	76.4	85.2	148.3	164.2	174.5	5.0	10.5	18.5
CAN	*Difference*	17.3	34.7	43.8	-0.4	20.6	31.0	**N/A**					
	Total	31.4	54.2	68.3	13.7	40.1	55.5						
USB	*Difference*	**N/A**						6.3	15.9	30.5	7.9	15.6	21.2
	Total							39.3	53.9	73.2	40.9	53.6	63.9

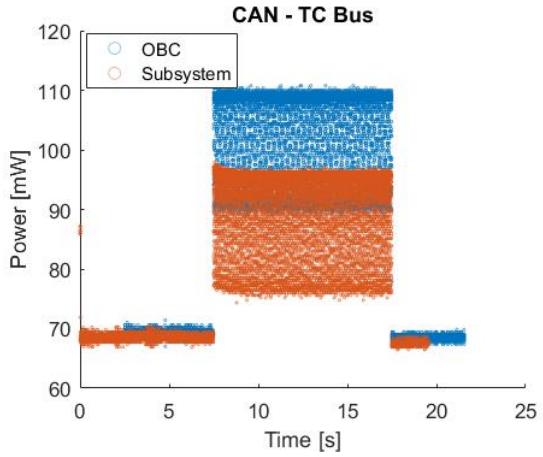

Figure 7. The power consumption of both the OBC and generic subsystem when using CAN. The bus termination resistors are 2 x 120 Ω.

For RS485, of which the power consumption during the TC tests is shown in Figure 6, shows highly similar values between the OBC and subsystem over the entire test cycle. However, this is to be expected because RS485 is implemented as a purely serial output in the bus simulator: both systems are performing the same amount of work by repeatedly sending a single command packet back and forth. This is different in for example I²C, where the bus master (OBC) has more responsibilities with arbitrating the bus, causing a difference in amount of activity between the OBC and subsystem. This behaviour of RS485 is mirrored in Table 7, where the mean power consumption of the OBC and subsystem are within 1 mW of each other. This must also be kept in mind when determining the power consumption of a full five-node TC bus: simply multiplying the computed values with five will result in a highly exaggerated overall bus consumption. Instead, one must note that the sum of the two mean power consumptions equals a 100% bus duty cycle. Thus, the overall bus power with more nodes will be the same, but including the idle power for three more bus nodes: approximately 151 mW. This value is much higher than the predicted value of 59 mW. It is thought that the majority of this power is used by the bus termination resistors: two (parallel) 120 Ω resistors[10]. To test this hypothesis, a small test was performed using two 2 kΩ resistors, which resulted in a reduction of 51 mW of the OBC's mean power consumption.

The effective data rate of RS485, its baud rate configured as the standard value of 1 MHz[10], equals just over 600 kbit/s as shown in Table 6.

Regarding CAN, the power consumption in Figure 7 shows much more noise than for the other bus types. This is most likely due to the SPI switching between active states and waiting for any actions. Surprisingly, the approximated power consumption of CAN (also included in Table 7) is significantly lower than for RS485 using the same termination resistors: sitting approximately in the middle between I²C and RS485. For the total TC system, between 86.2 mW and 290.3 mW with a mean of 214.6 mW is required. Unfortunately, the relatively low power consumption is probably caused by the low effective data rate seen during the tests: only 94.9 kbit/s. It is suspected that the combination of using an external CAN controller requiring the overhead of its SPI interface plus the overhead of the CAN protocol layer itself causes a large reduction of the maximum data rate. Therefore, it is expected that selecting a CAN controller which is included in a microcontroller will significantly increase the data rate. As the minimum overhead of CAN is approximately 42%[25] of an eight byte packet, the maximum achievable effective data rate would be around 580 kbit/s. The amount of extra power consumed caused by the higher data rate could potentially be reduced in the same way as with RS485 by increasing the value of the termination resistors. A test run with two 2 kΩ resistors showed that the power consumption dropped by about 20 mW per bus node without loss in data rate, putting it on similar levels as I²C.

Results: Payload Bus

The payload bus test was performed separately with both RS485 and USB. Similar to the TC tests, Figure 8 and Figure 9 show the accumulated measurements of the power consumption of the RS485 tests and USB tests respectively. Moreover, Table 6 presents measured data rates and Table 7 includes minimum, maximum and calculated mean power values for both PL test cases.

With regards to power consumption, RS485 shows slightly higher values as per its TC bus test: mean values of 174.7 mW versus 152.2 mW (two nodes), which corresponds to the slightly higher data rate of 781.25 kbit/s versus 607.17 kbit/s. Again, this confirms the hypothesis that a large contribution of the bus power is the loss in the termination resistors, as the only major difference between the TC and PL applications is a higher duty cycle of the bus.

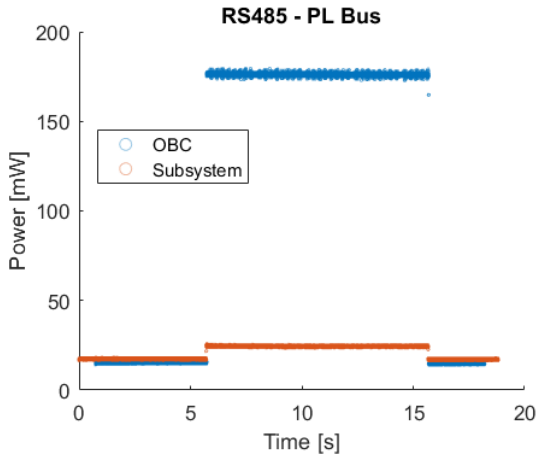

Figure 8. The power consumption of the OBC and generic subsystem when performing a large data transfer from the OBC to the subsystem over RS485.

USB, which is only tested in PL configuration, consumes 107.5 mW for the total two-node bus, providing nearly exactly 1 Mbit/s of effective data rate of the theoretical maximum of 12 Mbit/s. Similarly to CAN, it is thought that the use of an external SPI-driven USB controller is the cause for the relatively low effective data rate of the bus.

The differences in power consumption between the theoretical analysis for the trade-off and the actual measured values are summarised in Table 8.

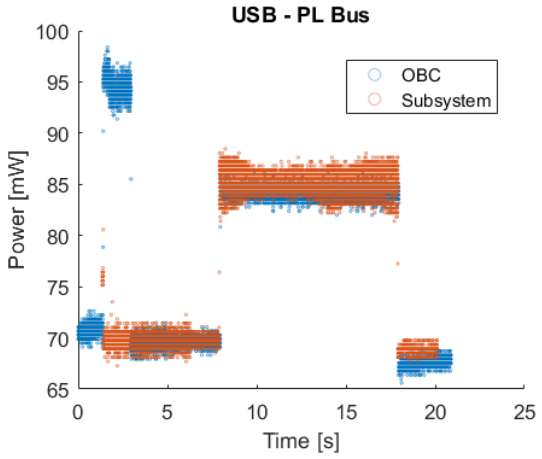

Figure 9. The power consumption of the OBC and generic subsystem when performing a large data transfer from the OBC to the subsystem over USB. Note the extra (peak) power required for enumeration, required once to establish the connection.

Table 8: A summary of the expected power consumptions of the TC and PL buses, the values based on the measurements and their differences.

Bus	Expected Power Consumption	Measured Power Consumption	Difference
I^2C	85.4 mW	121.5 mW	36.1 mW
CAN	429 mW	290 mW	-139 mW
RS485 (TC)	59.0 mW	160 mW	101 mW
RS485 (PL)	46.1 mW	174.7 mW	128.6 mW
USB	225 mW	107.5 mW	-117.5 mW

CONCLUSIONS AND RECOMMENDATIONS

This paper has investigated a new serial data bus architecture for use in CubeSats and other nano-satellites of similar size. First a theoretical analysis was performed, after which a validation was performed of the results.

A trade-off was performed, reducing a list of potential bus standards to only a handful of options to be fully analysed. The detailed analyses were performed with the assumption of splitting up the main bus into two separate buses: the telemetry and command bus and the high speed payload bus. This trade-off results in a novel and optimised data bus architecture.

Using the theoretical trade-off for the TC bus, both CAN and RS485 are equally recommended. The exact choice will be dependent on the exact mission and will most likely be between two factors: if reliability is of primary concern, then CAN is the most likely choice. If on the other hand low power consumption and high

data throughput is of high importance, then RS485 is the better choice.

For subsystems requiring dedicated and high speed bus connections, the separate PL bus is added. USB is given as the theoretical optimal choice, as it provides several fault detection methodologies as well as providing a high data throughput. It must be noted however that USB cannot be implemented in a linear bus topology, but only in a point to point style design. Careful design of the bus and subsystem layout is therefore required to omit the need for extra high speed bus nodes.

The data bus simulator was developed to provide a platform to implement the different selected data bus options resulting from the trade-off and to validate their performances. Although many of the main features of the trade-off options are well defined in datasheets and general descriptions of the technology, two essential networking characteristics, the power consumption and data throughput, are difficult to estimate and assess. For I^2C and RS485 in the TC bus, the measured power consumption was higher than initially expected. For CAN, the opposite is the case, as it only consumes approximately half of what was expected. Still, when implementing all subsystems according to their standards, CAN still consumes approximately double that of I^2C. However, by increasing the value of the termination resistors, the power consumptions of CAN and RS485 will reduce to roughly the same values as I^2C.

The effective data rate also featured differences, where all tested buses showed significant deviations from the ideal. Especially CAN, which was implemented through an external integrated circuit, showed relatively low data rates: managing only half of the effective data rate of I^2C. This shows that to efficiently implement CAN at higher speeds (up to 580 kbits/s), microcontrollers with built-in CAN controllers must be selected. Such a controller is unfortunately not part of the MSP432. Therefore, for a truly universal and microcontroller-independent TC bus, RS485 would be the preferred choice.

In the PL bus analyses, USB consumed only approximately half the amount of power of what was expected while also providing a higher effective data rate than RS485 in the same test. The latter also used more power. However, again in this case, tweaking the termination resistors could prove positive for the performance of RS485.

As RS485 does not define any protocol (part of the OSI link and network layers), hence no software-based safety mechanisms are built in by default. Further research might investigate which standard protocols (e.g. KISS, Modbus) could add robustness to the standard. The relatively low power consumption, high achievable data rate and simplicity in combination with a reliable protocol could make it the clear preferred option over CAN and USB. Since it is also a decent contender for use as PL bus, RS485 could still be an option for the next single universal data bus.

FURTHER WORK

The data bus simulator as described in this article is still under development. The goal is to provide a realistic CubeSat-networking testing platform which shall be capable of performing more in depth analyses of the various data bus options such as:

- Reliability analysis, amongst which bit error rate, and electro-magnetic interference and radiation effects.
- Performance characterisation, with full simulations of CubeSat systems and their behaviour. This potentially includes differences between external (CAN) bus controllers and equivalents built into microcontrollers.
- Ratings of the complexity of software and the necessary drivers.

Using the information gained from these simulations, the trade-off will be revisited with more accurate estimates for the bus characteristics. The main objective is to look at the results from slightly deviating from the established bus standards for further optimisation. For example, increasing the termination resistors' values on the CAN and RS485 bus tests showed a sharp decrease in the amount of power used to run the buses. Furthermore, the relatively short bus line lengths in CubeSats might allow overclocking of those same buses. In the end, a small and limited spectrum of bus architectures should be designed, each with a corresponding performance envelope. Ensuring clear definitions of these bus architectures allows simplified design and optimal performance of future CubeSat data buses.

REFERENCES

1. Heidt, H. et al. 2000. "CubeSat: A New Generation of Picosatellite for Education and Industry Low-Cost Space Experimentation," *Proceedings of the 14th Annual AIAA/USU Conference on Small Satellites*, Logan, USA

2. Woellert, K. et al. 2011. "Cubesats: Cost-Effective Science and Technology Platforms for Emerging and Developing Nations," *Advances in Space Research*, Vol. 47, No. 4. (DOI: 10.1016/j.asr.2010.10.009).

3. Bouwmeester, J. and J. Guo. 2010. "Survey of Worldwide Pico- and Nanosatellite Missions, Distributions and Subsystem Technology,"

Acta Astronautica, Vol. 67, No. 7–8. (DOI: 10.1016/j.actaastro.2010.06.004).

4. NXP Semiconductors. 2014. *UM10204 I²C-Bus Specification and User Manual*, URL: http://www.nxp.com/documents/user_manual/UM10204.pdf

5. Bouwmeester, J., M. Langer, and E. Gill. September 2016. "Survey on the Implementation and Reliability of CubeSat Electrical Bus Interfaces," *CEAS Space Journal*. (DOI: 10.1007/s12567-016-0138-0).

6. Manyak, G. and J.M. Bellardo. 2011. "PolySat's next Generation Avionics Design," *Proceedings - 4th IEEE International Conference on Space Mission Challenges for Information Technology, SMC-IT 2011*. (DOI: 10.1109/SMC-IT.2011.13).

7. Gerhardt, D. et al. 2016. "GOMX-3: Mission Results from the Inaugural ESA In-Orbit Demonstration CubeSat," *30th Annual AIAA/USU Conference on Small Satellites*, Logan, UT

8. Mitchell, C. et al. March 2014. "Development of a Modular Command and Data Handling Architecture for the KySat-2 CubeSat," *2014 IEEE Aerospace Conference*, Big Sky, MT (DOI: 10.1109/AERO.2014.6836355).

9. Nohka, F., M. Drobczyk, and A. Heidecker. 2012. "Experiences in Combining Cubesat Hardware and Commercial Components from Different Manufacturers in Order to Build the Nano Satellite AISat/Clavis-1," *Proceedings of the AIAA/USU Conference on Small Satellites, Mission Lessons*

10. Horowitz, P. and W. Hill. 2015. *The Art of Electronics*, Cambridge University Press, New York, NY.

11. Saponara, S. et al. 2007. "Radiation Tolerant SpaceWire Router for Satellite On-Board Networking," *IEEE Aerospace and Electronic Systems Magazine*, Vol. 22, No. 5. (DOI: 10.1109/MAES.2007.365328).

12. Wiznet. 2008. *W5100 Datasheet*, URL: http://www.wiznet.co.kr/wp-content/uploads/wiznethome/Chip/W5100/Document/W5100_Datasheet_v1.2.6.pdf

13. LIN Consortium. 2010. *LIN Specification Package (2.2A)*

14. Atmel. 2004. *AVR318: Dallas 1-Wire Master*, URL: http://www.atmel.com/images/doc2579.pdf

15. NXP Semiconductors. 2009. *PCA9513A / PCA9514A (Datasheet)*, URL: http://cache.nxp.com/documents/data_sheet/PCA9513A_PCA9514A.pdf

16. de Jong, S., G.T. Aalbers, and J. Bouwmeester. 2008. "Improved Command and Data Handling System for the Delfi-n3Xt Nanosatellite," *International Astronautical Congress*, Glasgow, Scotland

17. ST3485. 2016. *ST3485 RS-422/RS-485 Transceiver Datasheet*, URL: www.st.com/resource/en/datasheet/CD00003137.pdf

18. Microchip Technology. 2012. *MCP2515 - Stand-Alone CAN Controller With SPI Interface (Datasheet)*, URL: http://ww1.microchip.com/downloads/en/DeviceDoc/21801e.pdf

19. Texas Instruments. 2015. *SN65HVD23x 3.3-V CAN Bus Transceivers Datasheet*, URL: http://www.ti.com/lit/ds/slls933g/slls933g.pdf

20. Leens, F. 2009. "An Introduction to I2C and SPI Protocols," *IEEE Instrumentation & Measurement Magazine*, Vol. 12, No. 1. (DOI: 10.1109/MIM.2009.4762946).

21. Axelson, J. 2009. *USB Complete: The Developer's Guide*, Lakeview Research LLC, Madison WI, USA.

22. Maxim Integrated. 2007. *MAX3421E (Datasheet)*, URL: https://datasheets.maximintegrated.com/en/ds/MAX3421E.pdf

23. Delft University of Technology. 2016. *DelfiSpace - TU Delft Small Satellite Program*, URL: http://www.delfispace.nl/

24. Texas Instruments. 2016. *Energy Trace for MSP432*, URL: http://processors.wiki.ti.com/index.php/Energy_Trace_for_MSP432

25. Tindell, K.W., H. Hansson, and A.J. Wellings. 1994. "Analysing Real-Time Communications: Controller Area Network (CAN)," *Proceedings - Real-Time Systems Symposium*, Pisa, Italy (DOI: 10.1109/REAL.1994.342710).