Testing of an Optimised Data Bus for Pico- and Nanosatellites

Making CubeSats Future-Proof

by

Stefan van der Linden

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Thursday January 26, 2017 at 14:00.

Student Number:	4082346	
Project Duration:	April 12, 2016 – January 26, 2017	
Thesis Committee:	Ir. B.T.C. Zandbergen,	TU Delft, SSE (Head of Committee)
	Ir. J. Bouwmeester,	TU Delft, SSE (Supervisor)
	Ir. C. De Wagter,	TU Delft, C&S
	J. Carvajal Godinez,	TU Delft, SSE

TU Delft

Abstract

The definition of the CubeSat is already nearly two decades old. This results in the trend that a large fraction of CubeSat missions are leaving the domain of (educational) technology demonstration and entering that of commercial applications. Therefore a larger focus is put on the reliability and compatibility of commercial of the shelf systems and the spacecraft containing them.

One subsystem that is more or less fixed since the first CubeSat missions is the serial data bus. This internal network is essential for interconnecting subsystems. However, recent investigations have shown that the industry standard I^2C must be re-evaluated due to performance and reliability restrictions. The research contained in this thesis sets off to evaluate these characteristics of other data bus standards to propose a possible future-proof data bus architecture.

By splitting up the analysis cases into two parts, the choice and design of both parts can be optimised. The Telemetry and Command (TC) bus carries essential commands and housekeeping data between systems, while the Payload (PL) bus is dedicated to high speed bulk data transfers. An initial requirement-based selection of bus standards reduced the selection of standards to several options for both bus cases; for the TC bus, I^2C, CAN and RS485 were selected. For the PL bus, CAN, RS485, SPI and USB were selected.

The selected standards were all implemented in a data bus testing suite, comprising of up to nine simulated subsystems providing pseudo data to be communicated over the bus. Measurements were conducted of the buses' power consumption and data rates in several realistic test cases. Furthermore, the complexity and ability to withstand noise and voltage transients were evaluated. Ultimately, this resulted in a recommendation of using RS485 in future TC bus configurations and SPI in the PL bus configurations. However, this conclusion must be regarded a recommended direction of future research for several reasons. Firstly, more investigations are needed in the ability of these buses to work when subjected to large amounts of noise and other extreme environments. Secondly, the performed trade-off does not apply weighting to its criteria, as these can and will vary wildly for different missions. Finally, the test setup was limited in terms of processor ability for the PL bus case. These specific tests should therefore be redeveloped with more powerful equipment, allowing an even more realistic simulation of future CubeSat subsystems.

Acknowledgements

This thesis, and hence my educational career, would never have been completed without the help and support from a large number of people.

First of all, I am extremely thankful to my daily supervisor Jasper Bouwmeester for the continuous help and support during the entire thesis project. Many thanks also go out to Stefano Speretta, Tatiana Pérez Soriano, Aleš Povalač and Johan Carvajal Godinez for their help, patience and the many fruitful discussions we had while I was finding my way into the diverse world of the many electrical and electronics engineering concepts.

The people at two different companies have also played large roles in the development and steering of my interests into space system engineering in general and specifically in CubeSat engineering. Firstly, David Gerhardt, Jesper Abildgaard Larsen and the rest of the great team while I was at GomSpace provided an incredible amount of motivation and guidance during and after my internship. Secondly, the many people at S[&]T also provided vast amounts of inspiration and largely introduced me to the art of space and data engineering.

Finally, the countless other people I want to thank, of which there are way too many to list here, are all my family and friends. My entire studies at the TU Delft would have especially been impossible without the huge amount of support from my parents, my younger brother Tristan and of course Annemiek.

Thank you.

Stefan van der Linden
Delft, January 2017

Contents

List of Figures

List of Tables

List of Symbols

Symbol	Description	Unit
λ	Poisson parameter	–
I	Electrical current	A
N	The number of connected nodes on a bus	–
N_{bits}	The (integer) number of bits transmitted in a certain transmission or timeframe	–
N_{err}	Integer size of the subset of bit errors in a given transmission of size N_{bits}	–
N_{lines}	The number of SPI chip select lines	–
P	Electrical power	W
P_I	Probability of getting a Type I error	–
P_{II}	Probability of getting a Type II error	–
P_{err}	Overall probability for getting a bit error	–
s	Scaling exponent	–
U	Electrical potential	V

List of Abbreviations

Abbreviation	Description
ACK	Acknowledge
ADCS	Attitude Determination and Control System
API	Application Programming Interface
BER	Bit Error Ratio
CAN	Controller Area Network
CAN FD	Controller Area Network Flexible Datarate
CANH	CAN-High
CANL	CAN-Low
CDF	Cumulative Distribution Function
COTS	Commercial Of The Shelf
CRC	Cyclic Redundancy Check
CS	Chip Select
Delfi-PQ	Delfi PocketQube
dI^{2}C	Differential I^{2}C
EMI	Electro-Magnetic Interference
EOF	End-Of-Frame
EOP	End-Of-Packet
EPS	Electrical Power System
eUSCI	Enhanced Universal Serial Communication Interface
FIFO	First In, First Out
FPGA	Field Programmable Gate Array
GNC	Guidance, Navigation and Control
GPIO	General Purpose Input and Output
GPS	Global Positioning System
HK	Housekeeping
I^{2}C	Inter-Integrated Circuit
IC	Integrated Circuit
IDC	Insulation-Displacement Contact
IDE	Integrated Development Environment
IO	Input-Output
IP	Internet Protocol
ISO	International Organisation for Standardisation

Abbreviation	Description
LIN	Local Interconnect Network
LEO	Low Earth Orbit
MAC	Medium Access Control
MISO	Master-In-Slave-Out
MM	Mass Memory
MOSI	Master-Out-Slave-In
MSB	Most Significant Bit
NAK	No-Acknowledge
NRZ	Non-Return to Zero
OBC	On-Board Computer
OSI	Open System Interconnect
PCB	Printed Circuit Board
PDF	Probability Density Function
PER	Packet Error Ratio
PID	Packet Identifier
PL	Payload
RAM	Random Access Memory
LDO	Low-Dropout Regulator
RMS	Root-Mean-Square
ROM	Read Only Memory
RS232	Recommended Standard 232
RS422	Recommended Standard 422
RS485	Recommended Standard 485
RTOS	Real-Time Operating System
RX	Receive
SCL	Clock
SCLK	Clock
SDA	Data
SPI	Serial Peripheral Interface
TC	Telemetry and Control
TCP	Transmission Control Protocol
TI	Texas Instruments
TX	Transmit
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
UTMI	USB 2.0 Transceiver Macrocell Interface
VHF	Very High Frequency

1

Introduction

"Everything went great right up to the explosion."

— Andy Weir, *The Martian*

"Loading new software into new computers and using it for the first time was like playing Russian roulette. It demanded and got a lot of respect."

— Gene Kranz, *Failure is not an Option: Mission Control From Mercury to Apollo 13 and Beyond*

Slowly but steadily, satellites based on the CubeSat standard are leaving the area of experimental technology demonstration and entering the domain of commercial, fully operational missions based on the developed technology. This transition calls for a review of current CubeSat *de facto* technology standards and common design choices.

CubeSats have been around since the early 2000s [1], with the first launch of this spacecraft class in June 2003 [2]. The most unique aspect of the standard is that the spacecraft consist of multiples of 10x10x10 cm^3 units. This form factor has been derived taking into account several aspects, such as the size of already available Commercial Of The Shelf (COTS) technology, launch vehicle restrictions and specific safety standards [1]. Apart from requiring a large drive in miniaturisation of existing technology to enable CubeSat missions, other aspects are also limited due to these dimensions. For example, the restricted size highly impacts the amount of electrical power able to be generated and stored by the spacecraft.

The strict constraints imposed on CubeSats have triggered many different parties to develop novel components and subsystems specifically to be compatible with CubeSats. This technology is generally well optimized to the limited power and spatial requirements. The resulting availability of many different COTS subsystems makes the standard very attractive for technology demonstration purposes and educational missions, although the use in commercial and scientific Earth observation missions is also growing [3]. Furthermore, the relatively low cost of the design, production and operations parts of the missions means these nanosatellites are

of interest for use in constellations. A well-known example of such a system under development is the Planet Labs 'Flock' constellation [4], which is aimed to create a frequently-updated service providing Earth imagery in the visual spectrum.

The increasing (commercial) reliance on CubeSats calls for an improvement in reliability of the individual spacecraft, as previously flight-proven technology is often required to reliably function in missions with a longer lifetime than the first generation of CubeSats. Furthermore, with the increasing number of separate subsystems being included in the same amount of on-board space, it may be expected that the amount of generated data and number of data interfaces will increase as well. This adds significant pressure to the essential backbone of the spacecraft keeping these systems together electronically: the data bus. The Inter-Integrated Circuit (I^2C) [5] standard may be regarded as the current *go-to* data bus standard in CubeSats due to its relative simplicity and wide support in third party subsystems and components [6]. However, there are concerns regarding its reliability [7]. Moreover, many other data bus standards are also capable of delivering much higher data rates, although often at the cost of increased power consumption [8].

Hence, there exists a need for a data bus architectural design which is able to support the higher required performance of CubeSats missions in the near future, while also increasing the reliability but minimising the power consumption of the bus. A full validation of the architecture will result in CubeSat developers to be able to better substantiate their choices for specific data bus standards and designs.

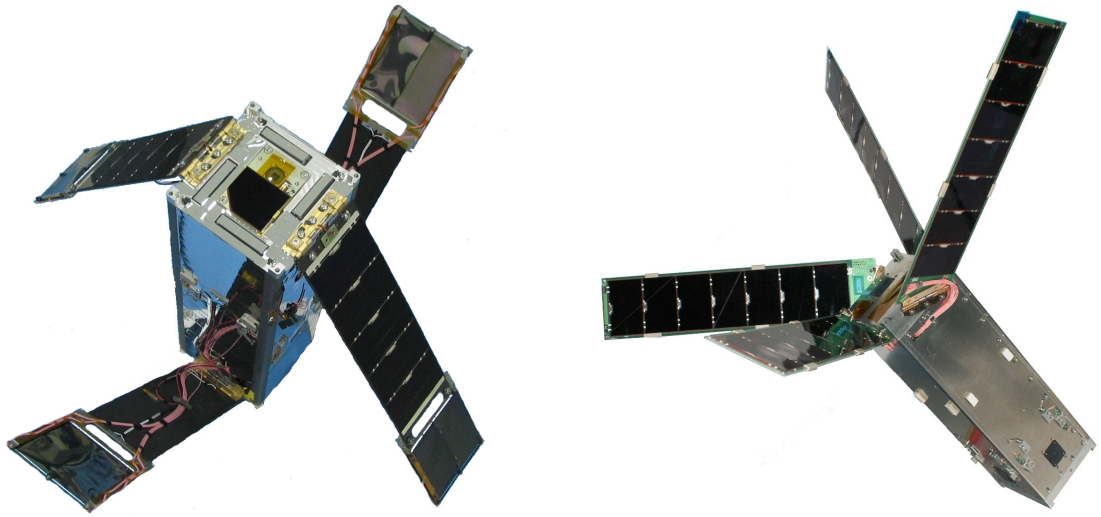
1.1. The Data Bus

Serial data buses are used in most, if not all satellites for communication and commanding between subsystems within the main satellite bus. Despite their small size, CubeSats are no exception. The I^2C serial bus standard is currently the most used bus in CubeSat missions [6], mainly due to its low complexity and wide support in third party commercial subsystems. Moreover and perhaps even more importantly, I^2C requires only a very small amount of power to operate, making it ideal for use in the highly constrained environment of a CubeSat [6][7]. However, the CubeSat industry is slowly outgrowing this bus standard.

Firstly, it has been shown that there are risks to the mission related to the use of this bus. Both CubeSats in the Delft University of Technology's Delfi satellite program, Delfi-C3 and Delfi-n3Xt (Figure 1.1), which were successfully launched several years ago, have experienced serious issues such as locked subsystems and fully locked buses [9]. These problems are reflected in experiences from other missions and is often thought to originate in external influences affecting the addressing of I^2C [10]. Although these issues are generally recoverable after (sometimes significant) effort by ground operators, they do delay mission operations and may even cause catastrophic failures in certain specific cases [10].

Similarly, the increasing amount of announced satellite constellations partly or fully based on CubeSats [3] means that the reliability and fault tolerance of individual units will become of further importance. For example: in interlinking communication constellations, an unexpected sudden failure of one or more CubeSats might have severe consequences for the instantaneous coverage. The same holds for the first CubeSats planned the relative safety of low Earth orbit [12].

Thirdly and finally, the instruments and experiments carried on board CubeSats are also steadily increasing in complexity and generated data load. On the other end of the data han-



(a) Delfi-C3

(b) Delfi-n3Xt

Figure 1.1: The Delfi satellites (image courtesy: [11])

dling chain, high speed space-to-ground downlinks have also been demonstrated [13]. Consequently, future CubeSat missions will require data buses with higher data rates to handle the large amounts of generated information reliably and consistently.

Thus, there is clearly a need for a new data bus architecture design which is both more reliable and allows for a better performance, while still practical when being applied in a CubeSat environment. To stay within the CubeSat mentality of using as many COTS components as possible, it is preferable to use widely available standards to maximise compatibility with existing hardware and minimise the amount of effort going into applying the technology. A related investigation performed by a several US based institutions have partly recognised this need and have proposed a modernised standard bus for US-government-owned CubeSats [14]. Although this proposal additionally encompasses the structural bus and subsystem form factors, it also includes the suggestion to use the Controller Area Network (CAN) bus to tackle the reliability issues. CAN mainly originates from the automobile industry and has been implemented sporadically in CubeSat missions [6]. Just like I^2C, it is one of only several widely available bus standards supporting a linear bus topology. Although completely designed with reliability in mind, a frequently-heard argument is the increased power consumption and increased complexity versus I^2C. Yet hardly any research has been done in validating CubeSat data buses and cross-comparing the results [15], hence it is difficult to assess whether a choice for CAN, or any other bus standard for that matter, is a large impact to a design compared to current implementations of I^2C. It is possible that this lack of information influences choices of bus standard in current satellite design cycles.

In conclusion, a modernised data bus architecture is required to be developed for CubeSat missions in the near-future. There is a need for a data bus with higher reliability and higher performance than is possible with the currently implemented architecture based on I^2C. An architecture consisting of a main command bus and a payload bus has been proposed previously. This idea has its foundations in bus standards chosen for several previous CubeSat missions. Nevertheless, there is next to no documentation available on the integration of CAN (or other data buses) in CubeSats, and how well the performance and reliability of the buses

were found. Thus, research is required to validate the use of CAN and possibly the combination with Universal Serial Bus (USB) in a CubeSat data bus and to thoroughly document the design and results.

1.2. Thesis Objective

The applied method within this thesis research is to perform a complete reselection of existing data bus standards from the ground up, mainly disregarding the results from earlier trade-offs so that the current state of the data bus technology can be evaluated for use in upcoming nano- and picosatellite missions.

This in turn is done by investigating the compatibility and practicality of implementing one of several serial data bus standards in a representative CubeSat environment. Furthermore, the performance and reliability of the data bus standards will be compared to the I^2C standard in various representative practical setups. This is aimed to lead to a recommendation of an optimal data bus standard and a recommended bus architecture.

A secondary objective is to provide well documented results on the implementation and integration of subsystems using the proposed data bus architectures. The design must be well supported and its limits explored. Furthermore, the integration process must also be evaluated. It is believed that this information will benefit designers of future CubeSats and provide a better foundation for the choice of data bus and the knowledge of its characteristics and behaviour.

A minor tertiary objective is to support the development of the new and even smaller PocketQube standard at the department of Space System Engineering of the Delft University of Technology. This is done by creating documentation for the data bus design and operation, and by creating tools for working with the Texas Instruments (TI) MSP432 microcontroller. Previous analyses and work have been performed with the older TI MSP430 series, but the current aim is to use the new series as its 'go-to' microcontroller in future satellites.

Thus, the main research question to be explored and answered is:

> Is there an optimal data bus or combination of data buses to be used in future CubeSat missions?

This question is to be answered by this thesis project through experimental comparisons between selected data bus formats. Therefore, the following sub-question is relevant:

> How do the proposed bus standards compare to I^2C in performance, reliability, power consumption and practical implementation?

It is quite possible there are practical or theoretical limits in some kind of way preventing the use of a specific data bus in certain situations. It is important to identify these:

> What are the practical limitations of the proposed bus architecture?

The main underlying goal of the thesis is to answer the main questions and the corresponding sub questions.

1.3. Thesis Outline

The goal of this thesis is thus to compare different data bus standards to each other to try and reach a conclusion on an optimal configuration for use in future CubeSat missions.

To kick off this process, first chapter 2 defines terminology, conventions and concepts used further up in the report. The next chapter presents the selection of data buses to be considered and tested experimentally. This is necessary because a virtually limitless amount of standards exists. It is of course impossible to test them all, hence a well considered selection must be made. With this selection made, the test metrics and test setup are described in chapter 4 and chapter 5. Chapters 6 through 10 then look at the actual implementation of all the buses, including descriptions of the software and the hardware, and provides the measurement results and initial conclusions for each bus. All these results are then combined in chapter 11. By using a small, generic trade-off process, a general discussion of the results are made. The report's conclusions and recommendations are then given and summarised in chapter 12.

2

Data Bus Design

The basis of all the data buses discussed within this document are the concepts of serial data transfer. To provide a context in which such a system operates, this chapter explores several basic concepts from information theory and electrical engineering. Starting from the lowest levels of digital communication and continuing to upper-level parts such as network topologies, many conventions and assumptions will be stated to avoid possible ambiguities later on in the paper.

2.1. Digital Information

Virtually every electronic circuit consists of a combination of many electronic components. In many cases within modern electronics, a central processor, or in smaller and more lightweight systems, a microcontroller, provides the central node of a certain subsystem. A microcontroller is usually required to perform many different roles or functions. For example, the microcontroller reads out data from connected components such as sensors, performs low-level calculations with these values and then acts by providing output. At the same time, the microcontroller is often supported by other secondary microcontrollers, requiring communication to coordinate between the two stations.

All communication must be reliable and efficient for a microcontroller to function properly. Therefore the vast majority of communications in modern electronic is performed digitally, which means the information being shared between components and systems comprises of binary data. This document follows the following conventional structure of binary data:

1. **Bit**: a single binary digit. This digit is either *true* (1) or *false* (0), or HIGH (1) or LOW (0).
2. **Byte**: a sequence of eight bits, together forming a number or part of a number (if used as part of a combination of bytes). This document assumes bytes are ordered Most Significant Bit (MSB) first, e.g. $0001\,0001_{\text{bin}}$ equals 17_{dec}. Furthermore, bytes are considered to be unsigned: e.g. $1000\,1000_{\text{bin}}$ equals 136_{dec}.
3. **Kilobyte, megabyte, ...**: two common standards commonly apply to SI prefixes of bits or bytes: either a kilobyte equals 1000 bytes, or it equals 1024 bytes. This thesis assumes the former, mainly to simplify computations. This further implies that one megabyte (MB) equals 1000 kB.

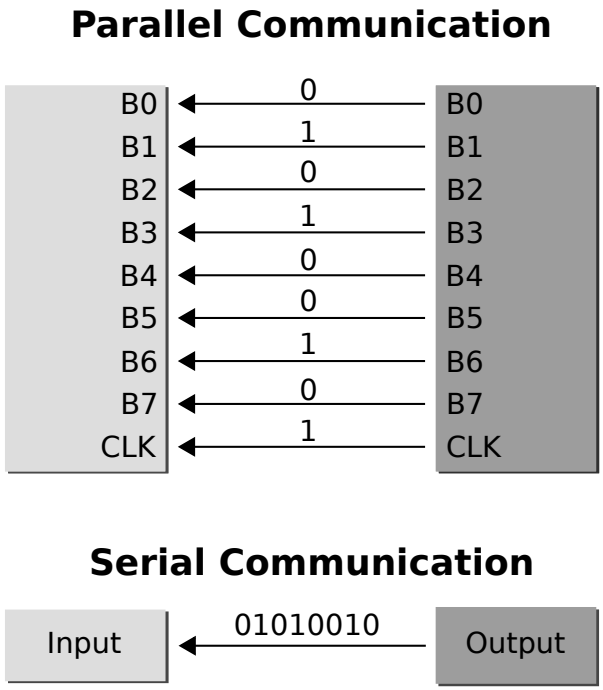


Figure 2.1: Parallel communication (top) versus serial communication (bottom)

Communication of binary data can be performed in one of two ways: parallel or serial [16]. Examples of both are shown in Figure 2.1, where both methods communicate the same byte. With parallel communications, the data is transferred over a large set of lines (one line for each bit). Usually, a separate clock line signals to the client when to read the individual lines. A serial connection uses a single line to transfer all the data in a sequential stream of bits. As will be seen in the following sections, there are different ways of providing the clock information with the line.

Parallel communication will often be faster than serial communication, since multiple data-carrying lines will always be capable of transferring more data per unit of time than an identical single line. However, the apparent major drawback of parallel communication is the high number of separate lines required to transfer the information. To transfer a single byte at a time, eight lines plus one clock line are needed. Apart from the resulting complexity in the design of such networks, this also means an equal amount of General Purpose Input and Output (GPIO) pins on a microcontroller are required to drive the lines. Since the amount of available pins is often very limited on a microcontroller, the use of parallel communication is simply impossible.

Due to these restrictions, parallel buses are hardly ever used when the information must be carried off a Printed Circuit Board (PCB). Serial communication is used in these cases.

2.2. Open System Interconnect (OSI) Layers

To standardise the different parts of an interface, the Open System Interconnect (OSI) standard layers were introduced by a work group part of the International Organisation for Standardisation (ISO) [17]. The model defines seven different layers describing how nodes on a network communicate to each other. The hierarchy is defined in a very system-engineering-like form.

The OSI layers are as follows (from highest to lowest):

7. Application
6. Presentation
5. Session
4. Transport
3. Network
2. Data Link
1. Physical

Each layer will be discussed concisely below. To help illustrate the concepts, the different layers will be explained using the architecture of the Internet, to which it is expected that many readers will be accustomed.

2.2.1. Layer 1: Physical Layer

The lowest layer, the physical layer, defines the electrical characteristics and the physical medium through which all signals pass. For the internet, this typically describes Ethernet when used in a person's home or workplace, but also the various glass fiber and copper connection connecting the different Internet endpoints to an Internet Service Provider.

2.2.2. Layer 2: Data Link Layer

The data link layer describes how binary data is moved between nodes on a network. This handles parts such as the (local) addressing and basic access control. The Medium Access Control (MAC) address, which is unique to each Internet-enabled device, is used for this by routers and switches handling relatively small network segments [17].

2.2.3. Layer 3: Network Layer

The network layer is responsible for establishing, maintaining and closing off connections to other nodes on the network. This is the layer in which the Internet Protocol (IP) and IP-address system works [17].

2.2.4. Layer 4: Transport Layer

The transport layer describes how packets are handled between nodes. For example, if packets are lost, then the technology in this layer ensures retransmission of the packet. The Transmission Control Protocol (TCP) performs this task within the Internet [17].

2.2.5. Layer 5 through 7: Session, Presentation and Application Layers

The upper three levels describe how the information is handled by the applications running websites and those reading them on the user side. For the internet, this includes the web

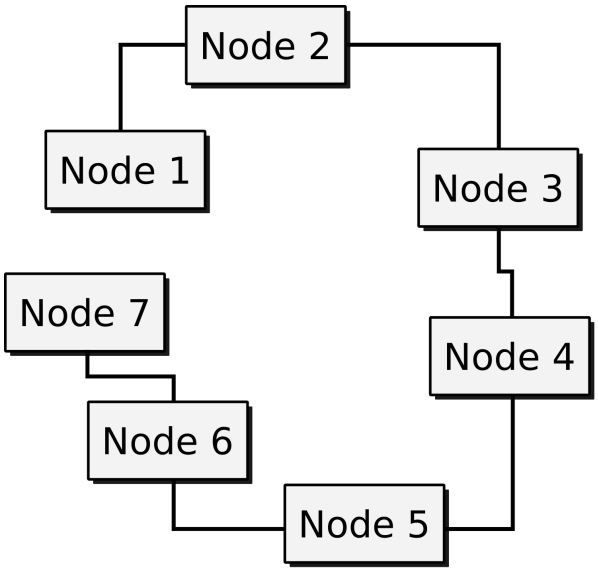


Figure 2.2: A point-to-point/daisy-chained bus topology

servers handling user requests and the web browser used by those user to display the transmitted information such as websites.

2.2.6. Scope of this Research

This research will mainly focus on the first two layers: how to get basic binary data from one point to another over a specific bus. The layers above these two are typically more application-dependent and might therefore vary between different satellite missions. For an example of related research to higher OSI layers at the TU Delft, the reader is referred to [18].

2.3. Bus Topologies

The architecture of how a serial data bus connects nodes together is called the network topology, and can vary wildly. In the most basic forms, three different main topologies can be identified: the ‘straight’ point-to-point topology (also known as daisy-chaining), the star topology and finally a linear topology [19].

The straight point-to-point topology is the most basic topology, where the bus will only connect up to two nodes together. Additional nodes must be added in series. This is called daisy-chaining. Figure 2.2 shows a simple implementation connecting seven nodes. An apparent drawback is the additional work added to each individual mode: a message sent from the first node to the last node must be passed on by each node in between. This also means that individual nodes must remain active, as the network is blocked once a node switches off. A ring topology can be made by connecting node 7 to node 1, but this increases the complexity of the node design due to the extra direction in which data can travel.

A different manner of using point-to-point buses is through a star topology as shown in Figure 2.3. The same nodes as in the previous case are connected to a single central node, here shown as a hub. Of course, the hub can be any node on the network. However in practice, the central node/hub must be capable of handling the routing of traffic over the entire bus. The

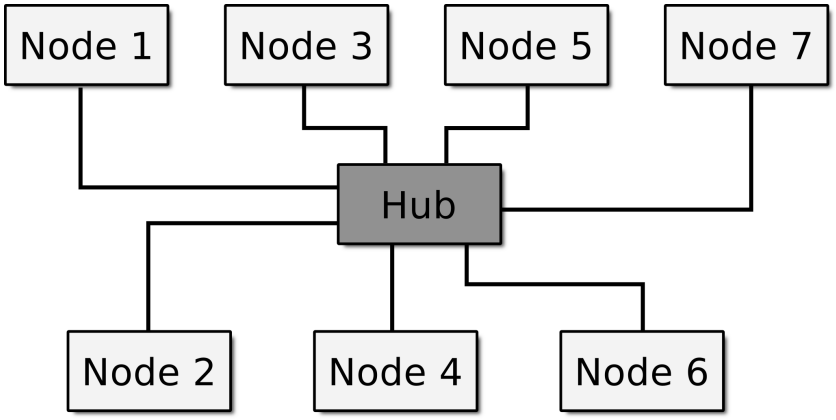


Figure 2.3: A star bus topology

star topology solves the problem where nodes must stay on all the time, at the cost of requiring a dedicated node for routing.

The aforementioned two topologies are the most common topologies connecting the internet, although that network is a highly complex combination of the two. However, a third topology commonly used within electronic systems for small scale networks: the linear bus topology.

An example of the linear bus topology is shown in Figure 2.4. The figure shows the same seven nodes as before, but now connected through a single communication channel. This design is useful in that nodes can be switched off without directly influencing the bus. It also allows a more equal distribution of work load per node. Moreover, two nodes can communicate together without disrupting the operation of the other nodes. The main drawback is the added complexity and overhead to the software protocol: in most cases node addressing is required to ensure correct routing.

As will be seen in the coming chapters, the linear bus is preferred as its benefits outweigh the drawbacks in practice, mainly affecting their power consumption.

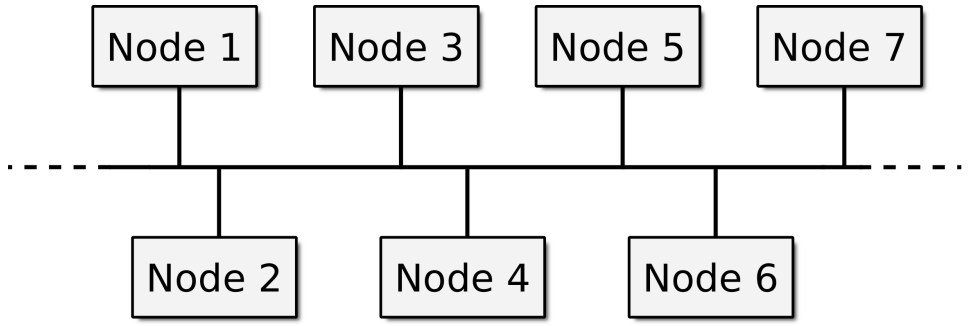


Figure 2.4: A linear bus topology

3

Data Bus Selection

(Note: the process and results of the trade-off included in this chapter have partly been presented at the 14th Reinventing Space Conference (2016) organised by the Royal Interplanetary Society in London, UK [20]. See also Appendix D).

To provide a solid basis for a practical experiment and to avoid having to test a very large number of bus standard and designs, a selection of 'data buses of interest' is performed using purely theoretical analyses. The goal of the selection process is not to result in a single recommended bus architecture, but rather in a 'shortlist' with several bus options. These options are analysed in more detail to yield a prediction of the various strengths and weaknesses between the different options. Practical implementation of the bus hardware and software as well as measurements of the main bus characteristics will provide a secondary independent review. Final conclusions and recommendations on future CubeSat data bus design will hence take into account both individual evaluations.

Due to the limited availability of both electrical power and physical space in most CubeSats [6], a straightforward choice for the data bus would be a single linear bus which connects all subsystems over a single communication channel. Indeed, the majority of CubeSats are implemented with such a (simple) architecture [21]. A similar data bus design was applied in Delfi-n3Xt, the second CubeSat launched by the TU Delft. Figure 3.1 shows a histogram containing the sizes of transactions versus the amount of these transactions during a single 2 s cycle of the Delfi-n3Xt On-Board Computer (OBC) polling for subsystem statuses and other housekeeping data. The figure shows a clear group of transactions with sizes of up to only several bytes, and furthermore a small amount of significant peaks at larger transaction sizes of several hundred bytes. The large gap between the two categories is due to different types of housekeeping data requests: because Delfi-n3Xt was a technology demonstration mission, the elaborate housekeeping data contained in the large transactions may be considered to be payload data.

Although Delfi-n3Xt does not contain a purely scientific or otherwise large-data-generating payload, it may be assumed that other CubeSats show similar data size distributions. Moreover, it is expected that the size of the gap between the small and large will increase when payloads are included which do create large amounts of data. To account for this effect and to allow for optimisation of the data bus, it is decided to perform the trade-off and experimental tests with two separate bus cases:

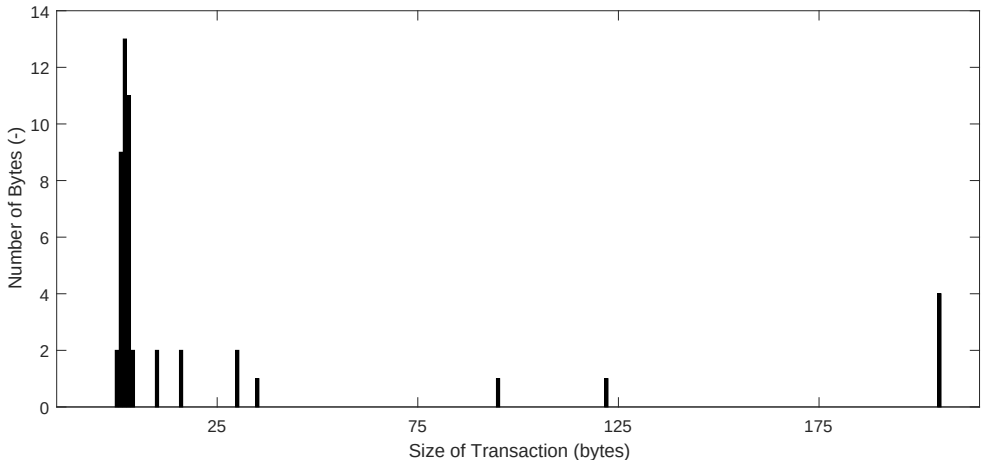


Figure 3.1: The distribution of transaction size versus amount of transactions of the Delfi-n3Xt satellite

1. The Telemetry and Control (TC) bus: all small commands and housekeeping polling is performed over this bus. This bus is used to connect all subsystems to the OBC for basic operation.
2. The Payload (PL) bus: this bus is meant to transport large sets of bulk data (e.g. payload data) between two data points, for example a payload and mass memory unit.

Separating the PL bus from the TC bus gives several benefits. First, buses which allow the interconnection of many nodes often do not feature relatively high data rates (as will be seen in section 3.3), thus a separate bus can be more optimised in terms of data throughput. Secondly, when a large amount of data is transferred over the PL bus, it does not block the critical operation of the TC bus by occupying the bus. Thirdly and finally, since the PL bus is typically not necessary in many critical system modes, it can simply be turned of when in one of these modes.

The system-level requirements will be defined and subsequent selection of the TC bus and PL bus will be performed in the next sections, but first several general top-level requirements are defined to ensure a clear objective of the trade-off process.

3.1. General Data Bus Requirements

As discussed in the introduction, the data bus is normally included in an electronic system when there exists a need to transfer information or data between multiple stations or nodes. This is no different in CubeSats, where many different components such as microcontrollers and sensors need to share information. This research will focus on the former: the main internal bus required to connect the various independent subsystems to one and other. The information shared between the system is, like in any digital system, in the form of binary data. This initial and main need can be described in the form of a top-level requirement:

REQ-01 - The data bus shall provide the means to communicate binary data

Binary information can be transferred through either parallel communication or through serial communication. Although the former is perhaps an intuitive method of communication, parallel

data transfer brings along several issues when implemented with bus lines longer than found on a single PCB, not including the added complexity caused by the large number of communication channels [16] (at least 9 when communicating with single bytes: 8 bits plus clock). Furthermore, cross-talk between the communication lines and typically higher power consumption are reasons for preferring serial communication over parallel communication over longer relatively longer distances (i.e. between PCBs) [22]. Therefore, the communication is expected to be in serial form:

REQ-02 - The data bus shall use a form of serial communication

The fact that there is a very limited amount of electrical power available in CubeSats must be taken into account as well:

REQ-03 - The data bus shall keep its electrical power consumption to a minimum

What these minima mentioned in REQ-03 exactly mean will be explained and defined in the next chapters.

Apart from only acting as a medium for information, the data bus must also ensure reliable transmissions. Especially in the hostile environment of space where electromagnetic radiation effects, partice radiation effects and extreme temperature effects are all much more frequent than in terrestrial applications. Thus:

REQ-04 - The data bus shall minimise the amount of errors in the data caused during transmission of the data

One of the basic ideas behind CubeSat design is the creating and making use of the large amount of COTS components [1]. One must also take into account that CubeSats are still often tools used for educational purposes, scientific projects and commercial, non-military entities. This leads to a requirement describing the public availability of components and general technology, as well as a separate requirement for the relevant documentation and licensing:

REQ-05 - Components used in the data bus shall be readily available as COTS
REQ-06 - Documentation and licensing of a bus standard shall be readily available at no significant financial costs

Somewhat related to the point of availability is the cost in terms of complexity with regards to the implementation of a bus. This highly subjective aspect is related to the amount of hardware and software that must be designed to get a specific bus to work.

REQ-07 - The complexity of implementing a data bus shall be kept at a minimum

Again, the exact definition of this requirement will be defined per bus case (TC or PL).

Apart from the limited power requirements in a CubeSat, the relatively small form factor means that physical space is also severely limited. Apart from a highly constrained intra-board spacing, meaning a low amount of room for connectors and wiring, the space on PCBs is also

severely limited [21]. Therefore the number of physical wires used to operate a bus must be low to allow for inclusion of the bus without major space issues. Because the number of wires was an essential (but not the only) reason to disregard parallel buses, these are used as a reference. As an upper limit, the requirement is stated that the number of separate bus lines must not exceed the number of bus lines for a parallel communicating bus:

> **REQ-08** - The number of data lines shall be less than a parallel communicating alternative

Finally, to ensure maximum compatibility between subsystems, it must be possible to implement the chosen data bus(es) in a universal manner: regardless of chosen microcontroller or bus node hardware.

> **REQ-09** - The data bus shall be universally compatible with typical microcontrollers

Having defined the top-level requirements, the following sections will take these main requirements to define specific system requirements. In turn, the system requirements are then used to select the shortlist of bus options to be tested experimentally.

3.2. TC Bus Requirements

To recap, the TC bus is used to communicate commands and essential housekeeping data between subsystems. This section will define the system requirements of the TC bus based on the top-level requirements defined in section 3.1. Moreover, the layout and characteristics of the network used in an analytic case (for the trade-off) and the practical case are worked out.

The system-level requirements which will be defined in this section act as the absolute minimum criteria each bus option has to meet, and does not necessarily describe the optimal or favoured bus design.

The system requirements are defined to describe the reference case as shown in Figure 3.2. The figure shows nine nodes including the OBC as central bus node. All subsystems are thus at least able to communicate with the OBC. Although it is not necessary to communicate in between, this might theoretically be possible depending on the specific bus standard. However, this functionality will not be taken into account any further: a single-master bus is assumed in all cases.

The first requirement derives from REQ-01 and REQ-02: to ensure continuous and fluent transmission of the data, a minimum data rate must be defined. However, this is the point where one enters a very grey area with a variety of ambiguous definitions. For example, I^2C Fast Mode is defined with a 'bit rate' of 400 kbit/s. However, protocol overhead in the data link layer of I^2C means that the actual rate of information is less than 400 kbit/s. In a similar case, CAN of course also contains overhead within the protocol. However, this overhead is defined to, for example, contain message priority information and a marker indicating whether (more) information is requested from a bus node. Hence, the use or uselessness of parts of the protocol overhead will vary between different bus standards and even different ways of implementing the standards.

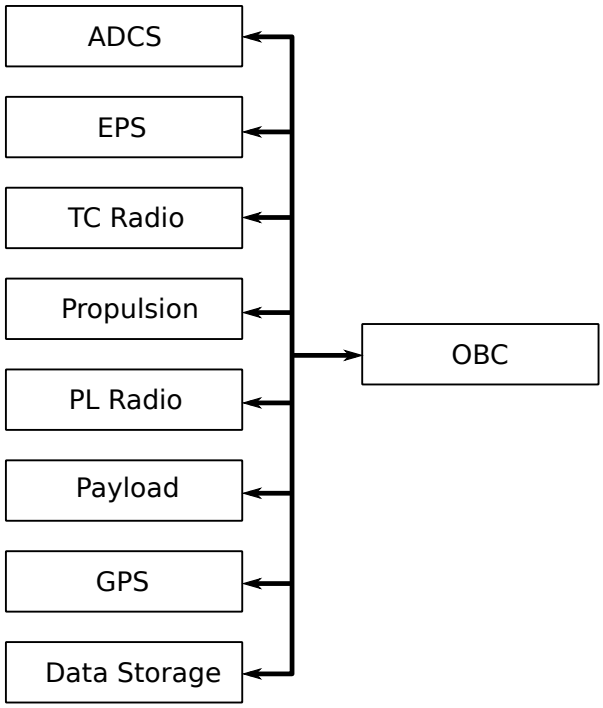


Figure 3.2: The TC bus reference case. Note that the indicated subsystems are examples only. The arrows indicate that data can travel in all directions.

To still be able to compare the different bus standards, the baud rate is used, which is defined as the number of signal events per second [16]. In other words, the baud rate is the base clock speed (in Hz) at which the bus operates at (e.g. 400 kHz for I^2C). As a minimum, a bus type must at least meet the value for I^2C's *Fast Mode* [5]. The choice for this mode and not the *Fast Mode+* with a 1 MHz baud rate is the fact that the baud rate on a bus will always default to that of the device with the lowest baud rate [23]. As the (slightly) older *Fast Mode* is more widely supported, this is taken as a baseline.

TC-01 - The baud rate shall be at least 400 kHz

The power consumption, as mentioned by REQ-03, must be kept to a minimum. This is especially necessary as the data bus is usually only seen as a subsystem supporting other subsystems. As the available power in a CubeSat is typically only on the order of several Watts [6], a (slightly arbitrary) peak power consumption of 1.0 W is chosen for the entire bus comprising of the reference case with nine nodes as shown in Figure 3.2:

TC-02 - The peak power consumption of the data bus according to the TC reference bus shall not exceed 1.0 W

The next requirements are related to REQ-04, which requires reliable transmission of data over the data bus. Especially for the TC bus, which requires a high confidence that commands actually reach their target subsystems, this is a critical aspect. Although the effect of typical measures assuring this will be discussed in more detail later on in this paper, two basic requirements are already defined:

TC-03 - The data bus shall contain a watchdog mechanism
TC-04 - The protocol shall contain a way of detecting bit errors

TC-03 requires as a bare minimum the usage of a watchdog. Such a mechanism monitors the activity of a bus and the connected nodes, and resets (typically through power cycling) the bus hardware when activity drops unexpectedly. TC-04 is added to ensure the detection of transmission problems. However, every bus able to transmit data already meets this requirement, as simply adding a Cyclic Redundancy Check (CRC) checksums to the data link layer is sufficient.

Complexity (REQ-07) is, due to its subjectivity, difficult to quantify. Nevertheless, an attempt is made here by describing the complexity in terms of the number of (dedicated) components required per bus node to connect it to the bus, not including the PCBs on which the components are added. An assumption is made here that the amount of effort needed to design the hardware and software drivers to run the bus scales proportionally with the number of components.

TC-05 - The data bus shall require less than 10 dedicated components per node

Finally, to make REQ-08 regarding the number of data lines more exact, TC-06 is introduced.

TC-06 - The data bus shall not have more than 7 main data lines

The requirements defined in this section, together with the top-level requirements defined in section 3.1, will be used to find the bus options to be used in the practical experiment.

3.3. TC Bus Selection

A large amount of different serial data bus standards have been gathered in Table 3.1 as originally collected in [8]. Even though a list such as this one will never be fully exhaustive, it does take into account as many standards as possible. To avoid having to add an near-infinite amount of options to the initial list, only bus options as defined per their official definitions are taken into account, describing their physical layer. Hence, this excludes all possible variations of bus options with regarding to differing resistor values, bus voltages and data link layers. Table 3.1 acts as a base for the next couple of sections where several bus types (included the eventually selected options) are analysed in more detail, mainly focusing on the power consumption requirements.

3.3.1. Controller Area Network (CAN)

CAN is regularly disregarded for use in a CubeSat due to its high expected power consumption [6]. Nevertheless, its roots within the automotive industry and its design with reliability in mind means it meets all requirements regarding this preliminary bus selection.

For a first order analysis of the power consumption of the CAN bus, each node is assumed to be comprised of an MCP2515 CAN controller [29] and an SN65HVD23319 [30] transceiver. The base (idle) current draw on one node then equals 16 mA. As CAN is essentially a half-duplex multi-master system, at most a single node will always be transmitting at the same time. Thus, when this is the case, this adds another maximum of 50 mA to the total current according to the datasheets of the two main components. For the 3.3 V bus consisting of nine nodes, this results in 640 mW. This value is compliant with TC-02.

Table 3.1: Overview of TC bus options and whether each option is rejected for the experiment or not. Options which are not rejected have been highlighted for clarity

Bus Standard	Rejected	Relevant Req.	Notes
CAN	No		See subsection 3.3.1
CAN FD	Yes	REQ-05	Officially not released yet at time of writing. Low availability of components
Ethernet	Yes	TC-02	Power consumption exceeds 1.0 W (See subsection 3.3.2)
Firewire	Yes	REQ-09	Not able to interface with microcontrollers (only through e.g. PCI/PCIe)
FlexRay	Yes	REQ-05	Low COTS availability: possibly discontinued
I^2C	No		See subsection 3.3.3
Infiniband	Yes	REQ-09	Not compatible with embedded systems
LIN	Yes	TC-01	Baud rate (20 kHz) too low [24]
MIL-STD-1553	Yes	REQ-05	Military standard: non-trivial availability
OneWire	Yes	TC-01	Baud rate (~15 kHz) too low [25]
RapidIO	Yes	REQ-05	Low COTS availability: possibly discontinued
RS232	Yes	TC-01	Baud rate (~110 kHz) too low [26]
RS422/RS485	No		See subsection 3.3.4
SpaceWire	Yes		Low availability of components for microcontrollers [27]
SPI	Yes	TC-06	Number of CS lines becomes very large [28]
Thunderbolt	Yes	REQ-06	High licensing cost
USB (2.0)	Yes	TC-02	Power consumption (with hub) exceeds 1.0 W (See subsection 3.3.5)
USB (3.0 / 3.1)	Yes	REQ-05	Host controller ICs are unavailable (See subsection 3.3.5)

3.3.2. Ethernet

Ethernet, the basic transmission system behind the Internet, is mainly design for use in industrial and consumer electronics. Nevertheless, several ICs acting as Ethernet-to-parallel controllers are available on the market. As a baseline, the Wiznet W5100 is assumed [31]. This IC, supports several basic Ethernet modes. However, the maximum power consumption for a single W5100 already equals 183 mA (604 mW at 3.3 V), which implies a total power consumption of approximately 5.5 W for the full nine node reference case, roughly equalling the total power output of a typical CubeSat [6]. Therefore, Ethernet is rejected for not meeting TC-02.

3.3.3. I^2C

As the main requirements have all been based on the performance and abilities of I^2C, it more or less automatically meets these requirements. It must be noted that although I^2C supports higher speed modes, these often require additional Input-Output (IO) buffers [28]. Therefore, in this analysis, only Fast Mode (with a baud rate of 400 kHz is used.

To estimate the power consumption of an implementation of this bus in the nine node TC bus case, several assumption have to be made. First of all, it is assumed that every node has one PCA9514 I^2C isolator/buffer[32]. 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. 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 [33]. 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 Clock (SCL) and Data (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. Nine nodes plus the main bus lines give a total of 46 mW. Adding up the expected power consumption of the bus buffers (approximately 3.5 mA $=$ 11.55 mW) gives a total of 150 mW. This value is easily compliant to TC-02, as expected.

3.3.4. RS422/RS485

Recommended Standard 485 (RS485), a half-duplex version of the simplex (but otherwise identical) Recommended Standard 422 (RS422), is actually only a definition of the physical layer [26]. Nevertheless, it is normally implemented using the built-in Universal Asynchronous Receiver/Transmitter (UART) of microcontrollers and acts as a medium for the UARTs to communicate. The lack of predefined data link layer and other protocols means that the functionality dictated by the REQ and TC requirements has to be achieved through software. Nevertheless, no limits are known which might cause non-compliance to the requirements.

During the design of Delfi-n3Xt, it was mistakenly assumed by the engineering team that RS485 is not able to operate in a linear bus architecture. Therefore, this bus was not included in the main data bus trade-offs performed at the time. It must be stressed that although RS422 is not able to operate linearly, RS485 is.

Concerning the power consumption of RS485, a typical configuration is assumed: the built-in UART of the microcontroller driving the node provides the data to a dedicated driver/-transceiver. This driver then electrically drives the data onto the differential bus lines. In this case, the ST348517 [34] is used as the reference component. Similar to CAN, only one node will always be transmitting at the same time. Therefore, the power consumption of a single transmitting node equals the overall bus power consumption. For the chosen TC bus reference case, it is assumed that the output of the driver is continuously equal to its default state value of 1.5 V [34]. 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 38.6 mW to the total figure. Thus, for the full TC bus, a total of 76.1 mW is found.

3.3.5. USB (2.0 and 3.0/3.1)

The Universal Serial Bus (USB) is well-known as the industry-standard peripheral bus used in consumer electronics connecting to a variety of systems, especially personal computers. Its popularity results in a high availability of ICs enabling connectivity between embedded systems and other USB devices. However, to enable straightforward implementation of USB on a peripheral device, the standard has been made very host-centric. In other words, the bulk of the bus' responsibilities is for the dedicated host controller [35].

To have a functional USB connection a separate host controller is therefore necessary on the bus. These are available for the older 2.0 standard, however the relatively new 3.0 and 3.1 versions of the standard do not have any host controllers available in the form of a COTS IC. This automatically causes the versions 3.0 and 3.1 to be rejected from the test.

Still, USB 2.0 is not yet rejected, as the required components are readily available. However, one must note that since USB is a point-to-point protocol, a hub or switch is required to connect more than two nodes to a single bus. Unfortunately a typical hub uses approximately 1 W just by itself when fully active [8]. Therefore version 2.0 is also rejected following TC-02.

3.3.6. TC Bus Result

Thus, the list of buses has been reduced from 19 options to just three:

1. CAN
2. I^2C
3. RS485

These options will be the buses considered in more detail with the practical experiment.

3.4. PL Bus Requirements

The PL bus is meant to carry bulk data from the point of origin, most notably as generated by on-board payloads or other types of instruments, to a second point handling the data: for example a high speed radio downlink or mass memory unit. The reference case as shown in Figure 3.3 is used for all theoretical analyses.

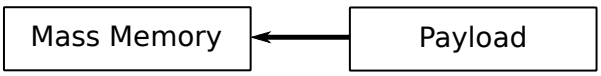


Figure 3.3: The PL bus reference case as used for the analysis and reduction of bus options. Note that the indicated subsystems are examples only. The arrow shows the typical direction of data flow.

For the PL bus, the approach to reducing the number of buses to consider is identical as the one used for reducing the number of TC buses. Naturally, the general top-level requirements as defined in section 3.1 remain the same and valid. This section will define the lower-level system requirements. Even so, as the basic functionality of the PL bus is the same as for the TC bus, all of the requirements apart from one as defined in section 3.2 also remain the same.

The only two system requirements that has changed is PL-01. For the former: instead of a minimum baud rate of 400 kHz, a minimum baud rate of 1 MHz is now required:

PL-01 - The baud rate shall be at least 1 MHz

The higher baud rate is required to provide a high data throughput for the bulk data.

All other system requirements are kept the same (but renumbered for the PL bus) and listed below:

PL-02 - The peak power consumption of the data bus shall not exceed 1.0 W
PL-03 - The data bus shall contain a watch dog mechanism
PL-04 - The protocol shall contain a way of detecting bit errors
PL-05 - The data bus shall require less than 10 dedicated components per node
PL-06 - The data bus shall not have more than 7 main data lines

These requirements can now be used to repeat the analysis as also performed on the TC buses.

3.5. PL Bus Selection

Table 3.2: Overview of PL bus options and whether each option is rejected for the experiment or not. Options which are not rejected have been highlighted for clarity

Bus Standard	Rejected	Relevant Req.	Notes
CAN	No		See subsection 3.3.1
CAN FD	Yes	REQ-05	Officially not released yet at time of writing. Low availability of components
Ethernet	Yes	PL-02	Power consumption exceeds 1.0 W (See subsection 3.5.1)
Firewire	Yes	REQ-09	Not able to interface with microcontrollers (only through e.g. PCI/PCIe)
FlexRay	Yes	REQ-05	Low COTS availability: possibly discontinued
I^2C	Yes		(Typical) Baud rate (400 kHz) too low [5]
Infiniband	Yes	REQ-09	Not compatible with embedded systems
LIN	Yes	PL-01	Baud rate (20 kHz) too low [24]
MIL-STD-1553	Yes	REQ-05	Military standard: non-trivial availability
OneWire	Yes	PL-01	Baud rate (~15 kHz) too low [25]
RapidIO	Yes	REQ-05	Low COTS availability: possibly discontinued
RS232	Yes	PL-01	Baud rate (~110 kHz) too low [26]
RS422/RS485	No		See subsection 3.5.3
SpaceWire	Yes		Low availability of components for microcontrollers [27]
SPI	No		See subsection 3.5.4
Thunderbolt	Yes	REQ-06	High licensing cost
USB (2.0)	No		See subsection 3.5.5
USB (3.0 / 3.1)	Yes	REQ-05	Host controller ICs are unavailable (See subsection 3.3.5)

3.5.1. Ethernet

As previously found in subsection 3.3.2, the minimum power consumption of a single Ethernet node equals 604 mW [31]. Hence, for the two nodes in the PL bus reference case, the total power consumption will still exceed the maximum allowed value of 1 W (PL-02).

3.5.2. Controller Area Network (CAN)

Apart from the apparent suitability of CAN as a TC bus option, it also meets all basic requirements for the PL bus. Using the values from subsection 3.3.1, it can be concluded that implementing the (linear) CAN bus according to the PL bus reference case will result in a maximum power consumption of 271 mW.

As other bus characteristics of the bus design such as the number of bus lines does not change, CAN still complies to all the other requirements.

A more detailed discussion of CAN including component selection and the measurement results can be found in chapter 7.

3.5.3. RS485

Just like CAN, RS485 also meets all requirements of both the TC bus and the PL bus. The expected power consumption is, using the earlier derived values, equal to 46.1 mW, which is still very low compared to requirement PL-02.

A more detailed discussion of RS485 and the measurement results can be found in chapter 8.

3.5.4. Serial Peripheral Interface (SPI)

Serial Peripheral Interface (SPI) was not considered before for the TC bus due to its high number of data lines. When implemented as a bus between two nodes, SPI consists of five lines [28]:

1. Chip Select (CS)
2. Clock (SCLK)
3. Master-Out-Slave-In (MOSI)
4. Master-In-Slave-Out (MISO)

The design of SPI allows sharing of the SCLK, MOSI and MISO lines, but CS lines need to be routes to each bus node. Thus, for a TC bus SPI requires eight CS lines to be able to select each subsystem. This drawback is negated when SPI is applied in a PL bus, where it only requires one CS or even none at all.

The expected power consumption of SPI is negligibly small: as there are no external components required to drive SPI apart from the wiring, all power is consumed by the microcontrollers on each bus node.

Chapter 9 will elaborate on the SPI implementation.

3.5.5. Universal Serial Bus (USB)

Because no additional bus hub is required to connect only two USB nodes, the power consumption becomes considerably less. The MAX3421E [36] (see chapter 10) is assumed for the electrical power calculations for the two node bus. This IC is capable of acting as both a USB peripheral (slave) and host.

The MAX3421E's datasheet [36] states a maximum power consumption of 150 mW when actively transmitting. It does not state a figure for when it is running idly as is assumed to be case when it is only receiving data, so it is assumed that that value is half the value of when it is transmitting. This gives a final approximation of 225 mW, significantly less than 1 W (PL-02).

As with the other selected bus options, the implementation and tests of USB will be discussed in more detail in chapter 10.

3.5.6. Result

This section has selected the PL bus options in the same way as for the TC bus. The resulting options are:

1. CAN
2. SPI
3. RS485
4. USB 2.0

These options will be tested together with the bus options as found in section 3.3.

3.6. Conclusion

This chapter has provided an initial set of data bus standards to be used in the creation of the data bus experimental suite. The next steps will be the definition of two parts: the aspects to measure and compare between the various options, and the definition of the subsystems.

Although there are many serial data bus standards available on the market (both open and proprietary), it is clear that when several basic requirements are stated to limit compatibility to CubeSats and other nanosatellites, the number of options drops drastically. Still, splitting up the main bus architecture into the TC bus and the PL bus allowed seriously consideration of point-to-point bus standards for the latter. This allows for optimization of both buses in terms of power consumption, data throughput and selection of secondary bus features.

The practical experiments with the buses will verify whether there are any major differences between the selected options and validate their analyses.

4

Experimental Comparison

As mentioned in the introduction to this paper, an experimental setup is used to test and compare the different buses in two different architectures. This chapter will go into detail in which tests are performed under what circumstances. The results from the tests will be presented and discussed in chapters 6 through 10, which will be used to populate a final trade-off matrix.

All metrics are designed in such a way that they are measurable on all bus types and architectures. Thus, the main tests will be performed on the following bus architectures, as selected in chapter 3.

For the **Telemetry and Control (TC) buses**:

- **Inter-Integrated Circuit (I^2C)**: a basic I^2C bus similar as used in the previous Delfi missions and to be used on the Delfi PocketQube (Delfi-PQ), performing command and payload data handling (chapter 6).
- **Controller Area Network (CAN)**: a bus originating from the automotive industry, mainly built around providing maximum reliability (chapter 7).
- **Recommended Standard 485 (RS485)**: the oldest standard of all, only defining a physical layer. The data link layer is provided by the microcontrollers' Universal Asynchronous Receiver/Transmitter (UART) (chapter 8).

Regarding the **Payload (PL) buses**:

- **CAN**: a similar implementation as for the TC bus (chapter 7).
- **Serial Peripheral Interface (SPI)**: this is available on most if not all microcontrollers, just like I^2C. The big difference is the significantly simpler data link layer, full duplex operation and higher data rates (chapter 9).
- **RS485**: implemented in point-to-point, more or less making it equal to RS422 (chapter 8).
- **Universal Serial Bus (USB)**: a bus well-known to consumers (chapter 10).

Each chapter mentioned in the lists above describes the exact implementation of each bus. Moreover, in some cases slight variations on the physical and data link layers from the defined standards will be evaluated on their effectiveness in the respective chapters. These include:

- **Differential I^2C**: an implementation of I^2C with differential signalling.
- **RS485 without termination resistors**: termination resistors are necessary in RS485 to avoid bus reflections. However, the relatively short bus lines in nanosatellites should not necessarily require them.

The following sections will look into the tests and corresponding metrics.

4.1. Tests and Metrics

To quantify the performance of a bus network, several metrics must be defined. These metrics are also used as the criteria in the final trade-off. The chosen metrics are as follows:

- Bit Error Ratio (BER) (for the PL bus only)
- Packet Error Ratio (PER) (for the TC bus only)
- Data Throughput
- Power Consumption
- Noise Immunity
- Complexity

The following sections will discuss the individual metrics in more detail.

4.2. Bit Error Ratio

The most common metric for the reliability of a bus is the Bit Error Ratio (BER). Measuring the BER requires transmitting a data set of a certain predefined size and verifying the number of bit errors in the data after the transmission. A test like this can be performed by the PL bus, as its purpose is transmitting a large amount of data in one direction only. For the TC bus however, this is problematic as it will operate in a packet-based manner. Therefore, the Packet Error Ratio (PER) is used for this case and will be described in section 4.3. The current section will describe the basic mathematical theory behind the BER which is required for the PL bus test and for developing a PER test.

The BER's definition is relatively simple, however it is difficult to determine the value for a specific communication link. It is hard to determine whether a ground-based test is representative for typical in space-based communications, as electronics have to operate in a multitude of highly varying electromagnetic and thermal environments, it may be difficult to determine a reliable value for the BER.

When transmitting a certain given bit, the outcomes as shown in Table 4.1 are possible, similar to the conventional Type I/Type II errors in hypothesis testing [37].

Table 4.1: Possible error types when transmitting a single bit

	Transmit '1'	**Transmit '0'**
Receive '0'	Error (Type I)	Correct
Receive '1'	Correct	Error (Type II)

The BER is defined as the ratio of the number of wrong bits N_{err} over the total number of transmitted bits N_{bits} [38], [39], as shown in Equation 4.1:

$$\text{BER} = \frac{N_{Err}}{N_{Bits}} \tag{4.1}$$

This equation presents the main problem for determining the BER: the value can only be found as $N_{Bits} \to \infty$ [38].

Fortunately, basic statistical methods exist to estimate the value for the BER. Analysing the different outcomes in Table 4.1, one might note the probability of each outcome may and probably will vary wildly in practice. For example, the occurrence of an error (Type I or Type II) may be due to a bit flip caused by some kind of energetic radiation interference event having an equal probability to yield both errors. When extending this example to one with I^2C, where the default state of the bus is HIGH (1), a Type II error might occur due to the transmitter not pulling the line LOW (0) correctly when required. Thus, in this example case the probability of getting a Type II error will be higher than a Type I error, as the sum of the probabilities to get a Type II (P_{II}) is greater than the sum of those for a Type I (P_I).

When each transmission of a bit is viewed as an experiment where the outcome is either an error or a correct bit, then the corresponding probabilities are $P_{err} = P_I + P_{II}$ and $P_{Corr} = 1 - P_{err}$. This statistical system is a classic example of a binomial process [37], [38]:

$$(P_{err})_{Binomial}\,(N_{err}, N_{bits}, \text{BER}) = \frac{N_{bits}!}{(N_{bits} - N_{err})\,!} \cdot \text{BER}^{N_{bits}} \cdot (1 - \text{BER})^{N_{bits} - N_{err}}$$

which can be approximated using a Poisson distribution as long as $N_{err} \ll 1$ and N_{bits} is very large [38]:

$$(P_{err})_{Poisson}\,(N_{err}, \lambda) = \frac{\lambda^{N_{err}} \cdot e^{-\lambda}}{N_{err}!} \tag{4.2}$$

where $\lambda = \text{BER} \cdot N_{bits}$. To compute the actual BER from the known quantities N_{err} and N_{bits}, λ may be approximated as equal to N_{err} [38]. Together with the corresponding Probability Density Function (PDF) of the Poisson distribution, an approximation of the BER can be found, which contains a certain error with respect to the actual BER. Figure 4.1 shows several plots of the PDF for various values of λ.

Unfortunately, the absolute value of the BER approximation's error can not be determined exactly. This reduces the overall confidence in the resulting value. However, knowing the probability distribution allows one to find the confidence levels associated with an approximated BER.

One may observe the Poisson PDF is able to give the probability of any arbitrary combination of N_{bits} and N_{err} to occur. For example, if 5×10^6 bits are transmitted resulting in a single bit error, then the 'measured' BER simply equals 2×10^{-7}. If it is of interest to know whether the actual statistical BER is under 1×10^{-6}, then Equation 4.2 can be used to determine the

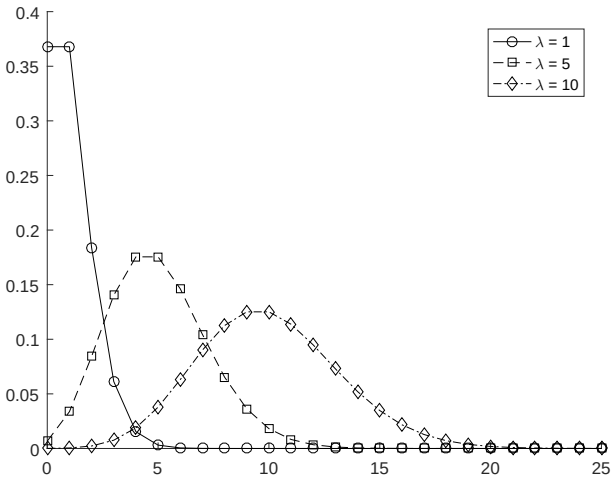


Figure 4.1: Poisson Probability Density Functions corresponding to various values of λ. Although the distributions are discrete, they are shown as continuous lines in this plot for clarity.

probability of having a single bit error in 5×10^6 bits. Since the Poisson distribution is a discrete distribution, its Cumulative Distribution Function (CDF) equals the sum of the PDF from 0 to the given index (in this case N_{err}) [37]. Hence, in this case [38]:

$$(P_{err})_{Poisson}(N_{err} = 0, \lambda = 5) + (P_{err})_{Poisson}(N_{err} = 1, \lambda = 5) = \\ = 0.0067 + 0.0337 = \quad 0.0404$$

Therefore, the probability of having a BER in this case of one error in 5×10^6 of less than 1×10^{-6} equals $1 - 0.0404 = 0.9596 \approx 96\%$.

Generalising, the upper bound of the 95% confidence interval can be found by solving the CDF in Equation 4.3 for λ [38]:

$$\sum_{0}^{N_{err}} (P_{err})_{Poisson}(N_{err}, \lambda) = (1 - 0.95) \tag{4.3}$$

And similarly for the lower bound:

$$\sum_{0}^{N_{err}} (P_{err})_{Poisson}(N_{err}, \lambda) = 0.95 \tag{4.4}$$

Both equations can be solved numerically.

It is important to note that the result of Equation 4.3 is tied to transmitting a set of bits while not exceeding a certain level of bit errors. Equation 4.4 contrastingly requires at least a certain number of bit errors in the given set of bits. The required number of bits to transmit is then simply found by dividing λ by the target BER [38], bringing the method back full circle:

$$N_{bits} = \frac{\lambda}{BER} \tag{4.5}$$

To conclude, the methodology presented in this section provides a powerful tool to approximate the BER while reducing the number of total bits to transmit. Furthermore, it allows a

Table 4.2: Computed values for number of tested bits versus number of measured bit errors following the Poisson distribution. Unless otherwise noted, within this project, the 'scaling exponent' $s = 6$. Table adapted from [38].

95% confidence interval BER >10^{-s}		95% confidence interval BER <10^{-s}	
Minimum number of errors	Maximum number of bits ($\times 10^s$)	Maximum number of errors	Minimum number of bits ($\times 10^s$)
1	0.05129	0	2.996
2	0.3554	1	4.744
3	0.8117	2	6.296
4	1.366	3	7.754
5	1.970	4	9.154
6	2.613	5	10.51
7	3.285	6	11.84

determination of the minimum number of bits to transmit over a bus to allow reliable conclusions on the reliability and performance of the bus. Although the critical values can only be determined through numerically solving the resultant equations, Table 4.2 shows some common values for reference, as a function of a 'scaling exponent' s. In practice, the test can be designed to make use of these values.

The BER test will transmit the computed number of bits over the bus to estimate the BER of the bus standard in question.

An important cut-off point in the BER is when it exceeds 10^{-6}, as this equals the maximum BER in Delfi-n3Xt [40], taken here as a benchmark value. Using the theory derived in this section and the values in Table 4.2, one may find that to determine (with a 95% confidence interval) whether the BER is less than 1×10^{-6}, at least 2.996×10^6 bits must be transmitted without an error. This value scales in the same way as the BER, hence for a BER of 1×10^{-8}, 2.996×10^8 bits must be transmitted. Since the bus standard with the lowest baud rate (400 kbit/s for I^2C) only requires just over seven seconds to transmit the 2.996×10^6 bits of data, the experiment's approach is simply to transmit this full data set. Although practical issues and various overheads will most like reduce the actual achievable baud rate, it still shows that overall running time is significantly reduced over 'traditional', brute-force BER testing.

To keep the running time of the test practical, the BER test is ran to show that each bus is able to meet a minimum required value of 10^{-6} for the BER, based on Delfi-n3Xt.

4.3. Packet Error Ratio

The Packet Error Ratio (PER) is quite similar to the BER and checks for missing packets. As the TC bus splits up all the transmitted data over multiple subsystems in packets, determining the BER becomes highly complex in design. A second factor is the fail-safe design of many bus' driving components: these often include automatic error detection for the payload part of a received message and automatically solve these problems by requesting a retransmission of the message. It is found that getting notified when a retransmission happens is not possible for every bus. For these reasons, it is chosen to compare the PER of the TC buses, as completely dropped packets are easier to detect and are often not handled by peripheral hardware.

How the packets are built-up vary per bus and will be discussed in each respective chapter. Nonetheless, the payload of each packet is assumed to be the same, and is presented in section 5.4.

The packets should not be influenced by a bit error in the message itself. However, serial communication protocols contain much more than just the data: parts like addressing and other meta data are critical for the correct nodes to receive the correct packets. For example, a bit flip in the address segment of a CAN or I^2C message can cause the wrong bus node to ACK a given message, potentially causing the loss of data and functionality. Therefore, measuring the PER takes into account many other factors.

The mathematics behind the PER are assumed to be identical to those of the BER described in section 4.2, thus resulting in certain confidence levels. Again, a 95% confidence level is assumed.

A slightly conservative figure for the PER is found when assuming that a single bit error in a packet will cause a packet error (this will be enforced in the test software). Noting the total amount of data that is transmitted (section 4.2) equals 748 bytes, or 5984 bits per polling cycle divided over 18 packets. Assuming a BER of 10^{-6}, this means that one error would be expected in every 167 cycles, or one in every $167 \times 18 \approx 3000$ packets. Hence, to ensure a 95% confidence of having an approximate BER of 10^{-6} according to the Poisson ratio, $2.996 \times 3000 \approx 9000$, which is rounded up to $10\,000 = 10^4$. In other words, the PER must be below 10^{-4} to be equivalent to a BER of 10^{-6}.

To keep track of which packets are dropped or not, a quasi-unique identifier is necessary. Since the packets will be filled with pseudo-data anyway, the first byte in the packet will be set to contain a simple incrementing (unsigned) integer. Hence, a missing frame will create a gap when transmitted between two subsequent correctly received frames. This is relatively straightforward to be checked by any bus node.

4.4. Data Throughput

The initial selection of the data bus standards looked at the theoretical maximum data rate, here defined as the baud rate. This value is typically specified by the bus standards. Measuring the value experimentally does not add new information: the clock speed of the bus will always follow the prescribed value: if not, then the bus is not implemented correctly.

Even though the baud rate is the maximum rate at which all information will transmit over the bus, the rate at which the ‘usable’ information is transmitted will differ. The rate at which this data is transmitted will be referred to as the data throughput. The difference between the baud rate and the data throughput is because protocol overhead increases the actual amount of data transmitted over the bus. Furthermore, latencies caused, for example, by nodes waiting for each other or by additional buffering of data will also lower the effective throughput.

To investigate how large these effects are, the data throughput will be measured for the bus options in the TC bus case and the PL bus. For the former, the number of packets in a certain amount of time will be counted, from which it is straightforward to deduce the amount of data transmitted. For the latter, the data is not transmitted in packets, hence the data throughput can be measured directly.

It is expected that the data throughput for TC bus cases will be lower than for each PL bus,

Figure 4.2: The Agilent 33210A signal generator

even if it concerns the same bus in both cases. This is because in the former case the focus of the bus keeps switching between subsystems, while for the latter this focus remains with the same system(s).

4.5. Power Consumption

Measuring the electrical power consumption is a relatively simple test, mainly due to the choice for using the Texas Instruments (TI) Launchpad. Having a separate single power source makes it straightforward to accurately measure the power consumption of a single node or of the entire system (see section 5.1) [41]. Since the controllers and other components powering the bus are powered through the Launchpad, it is possible to compare the different consumptions when using varying bus architectures.

The power consumption is traditionally a critical point of comparison between I^2C and CAN, as it is usually states as one of the main reasons to continue using I^2C [6]. It is therefore of interest to see the differences in power consumption between the two standards as well as the other selected buses in a controlled but realistic environment.

The measurements will be compared versus their expected values to determine the accuracy of the theoretical analyses.

4.6. Noise Immunity

By injecting noise into the main bus lines, it is investigated whether the use of differential data lines, such as used by CAN and USB, does result in significant gains in noise immunity compared to a I^2C, which uses a single bus lines. The (white) noise is to be generated by the Agilent 33210A signal generator, which has a 10 MHz bandwidth [42]. The maximum output of the generator's white noise signal equals 2 V Root-Mean-Square (RMS) or 20 V peak-to-peak.

The generated noise is connected to the main bus lines, thereby simulating noise caused by Electro-Magnetic Interference (EMI). The exact connection varies per bus type and will be discussed in more detail in each bus types' respective chapter.

A related test is to see the effects of transient effects on the bus lines. In real spacecraft, these are typically caused by the power switching of secondary subsystems, similar to connecting an audio-jack to an audio amplifier which is already turned on. The tests are performed by introducing transient peaks of two bit times length into the lines using the signal generator at increasing peak-to-peak voltage. Both positive and negative peak are tested.

The metric's value is the voltage level at which the PER of the tested architecture reaches

the threshold value of 1×10^{-4}. In these tests, the actual measured PER will be used and compared to the requirement value.

A secondary test is by using the signal generator to generate ‘transients’, i.e. voltage pulses, on the bus lines. Transients are common in an environment where electrical systems vary a lot in their power consumption [43]. The on/off switching of subsystems/components and highly varying currents under different system loads mean that since the Electrical Power System (EPS) can not respond with different loads instantly, the transients occur. For the test, these (positive) transients are set to be generated with a frequency of 100 Hz with a duty cycle such that their length equals approximately two bit times. Hence, the length of the transients varies for each bus standard. The eventual metric is similar as for the white noise: the peak-to-peak voltage at which the PER exceeds 1×10^{-4}.

4.7. Complexity

The ’complexity’ mainly describes how difficult a specific bus standard or architecture is to set up and integrate. This metric can not be assessed in a fully objective way. Thus, the different cases being compared will be ranked and the resultant scores normalised for use in the final trade-off.

4.8. Conclusion

This chapter has defined the main measurements and tests to be performed on each bus standard. The test setup as defined in the next chapter will be used to perform these tests, comparing the different buses on a level playing field.

5

Generalised CubeSat Data Bus Simulator

The main part of this thesis is to experimentally test the proposed bus architecture, log the results and draw conclusions regarding its implementation (chapter 4). Thus, the main purpose of the experiment is to test and validate the proposed data bus set up, and to provide conclusions on the practical implementation of the set up. A second purpose is to provide insight into applying the bus into the spacecraft design process and actual hardware. This is achieved by creating a realistic hardware-in-the-loop simulation of the data bus.

Several data bus architecture cases are compared. This includes the proposed architecture, but also a 'control case' based on I^2C. By measuring specific metrics (chapter 4), a pre-defined trade-off table will be populated to provide an objective comparison between the cases.

As an important baseline, multiple realistic subsystems must be simulated. It is impractical and unnecessary to use actual (legacy) subsystems from the Delfi program, as this shifts the focus of hardware development within the thesis project away from the data bus. The actual pseudo-subsystems are described in detail in section 5.4 and simply consist of identical microcontrollers acting as data sinks and sources. A practical benefit of this choice for the data bus test suite causes the set up to be as modular as possible.

Lessons learnt from Delfi-C3 show that using different microcontrollers for different subsystems means the most suitable microcontroller can be chosen for each specific job. However, bus standards are often implemented slightly different between various manufacturers or even component versions. These differences cause much of the development time of a bus being put into the fixing of bugs and other issues, rather than into optimisation of the bus performance. Selecting the same microcontrollers in the test suite thus avoid this problem and provides an equal comparison between the buses without focusing too much on effects introduced due to small incompatibilities between systems.

5.1. Bus Node Hardware

As noted in the introduction, the experiment consists of multiple connected subsystems. Each subsystem is identical to others, being based on the same hardware.

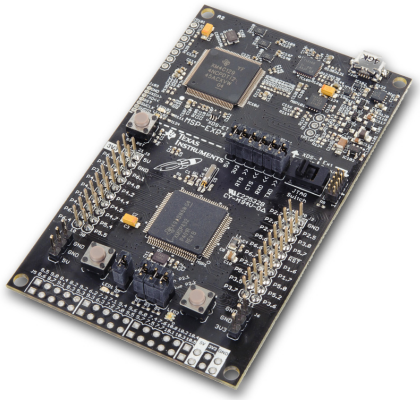


Figure 5.1: The MSPEXP432P401R Launchpad evaluation board. (image courtesy: Texas Instruments)

The cornerstone of the pseudo-subsystems is the TI MSP432 microcontroller, selected as the standard microcontroller in the Delfi-PQ program. Because this microcontroller is still relatively new at time of writing, only the MSP432P401R and MSP432P401M [41] are available. The main difference between the two is a lower amount of flash and Random Access Memory (RAM) memory in the 401R version. In the rest of this document and project, the MSP432P401R is assumed. This microcontroller is the follow-up product of the MSP430 family of low-power microcontrollers. The MSP430 has previously been the main microcontroller used on the Delfi-C^3 and Delfi-n3Xt CubeSats, as well as used in several other CubeSat missions or products [44], [45]. The MSP432 is based on the ARM 32-bit architecture and boasts an increased maximum clock frequency of 48 MHz over 25 MHz for the MSP430 [46] as well as increased flash memory and RAM space.

To reduce the complexity of the design, use is made of the Launchpad standard. The MSP432's Launchpad evaluation board, the MSPEXP432P401R, as shown in Figure 5.1, is available at low cost. Using the Launchpad omits the need for the development of a separate dedicated computer 'board'. It also comes with a built-in emulator used to program and debug the microcontroller's software. Internally, the emulator is connected to the microcontroller via a JTAG interface [41]. Finally, the emulator is also able to measure voltage and current use of the board, which is to be used during the tests to measure relative power consumption of the various subsystem boards.

All custom hardware driving the various buses is placed on a Printed Circuit Board (PCB), which in turn is connected to the fanout of the MSP432 Launchpad. There are several of these 'daughterboards' designed for various bus standards. The Launchpad's pinout connecting the daughterboard to the MSP432 is standardised by TI and branded the BoosterPack [47] [47]. The pinout (shown in Figure 5.2) features several serial connections to the microcontroller itself such as I^2C and SPI, but also clock outputs and General Purpose Input and Output (GPIO) pins. To simplify the test setup, the custom hardware is designed in such a way to be compatible with the pinout and the mechanical specifications.

Included on each daughterboard PCB is an Insulation-Displacement Contact (IDC) header for a ribbon cable connecting the subsystem board to a central bus. The ribbon cable can be extended or shortened if necessary. Moreover, as may be seen in Figure 5.3, the connecting wires for each respective bus are placed alongside each other for realistic implementation.

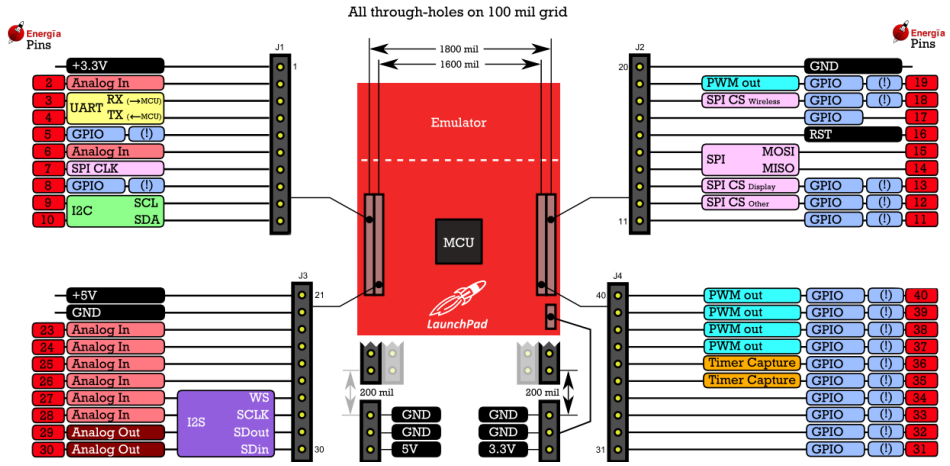


Figure 5.2: The BoosterPack standard pinout. Note that most pins are reconfigurable in software. (image courtesy: [47])

5.2. Physical Configuration

To have the physical set up closely resemble an actual CubeSat, the subsystems are stacked. The spacers used to achieve this add approximately 3 cm of spacing between the boards. Although double the minimum spacing in the PC/104 standard [48], this amount of room enables switching out the different daughter boards without disassembling the entire stack.

Although all boards can be power through their USB connectors, it is found to be highly impractical. Furthermore, measurement of the entire bus' power consumption becomes complicated and inaccurate. Therefore, the Launchpads were slightly modified to add a connector to the boards' internal 5 V and grounding rails, as can be seen in the circled area in Figure 5.4.

The overall test setup is shown in Figure 5.5.

5.2.1. Electrical Power Measurement

One of the metrics is the overall bus power consumption.

The electrical power measurements are performed using the Keysight Technologies 34401a

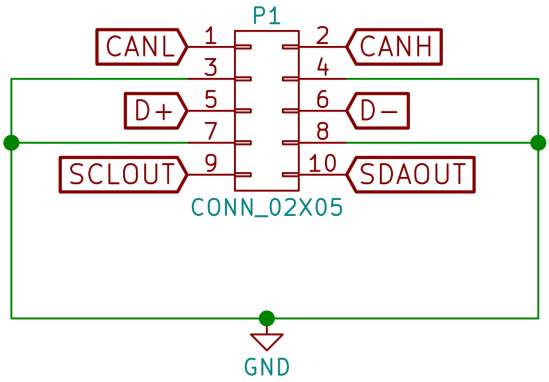


Figure 5.3: An example of the IDC pinout of the daughterboard featuring I^2C, CAN and USB hardware. The ten wires included in the ribbon allow enough room for grounding wires and the pairs carrying the data.

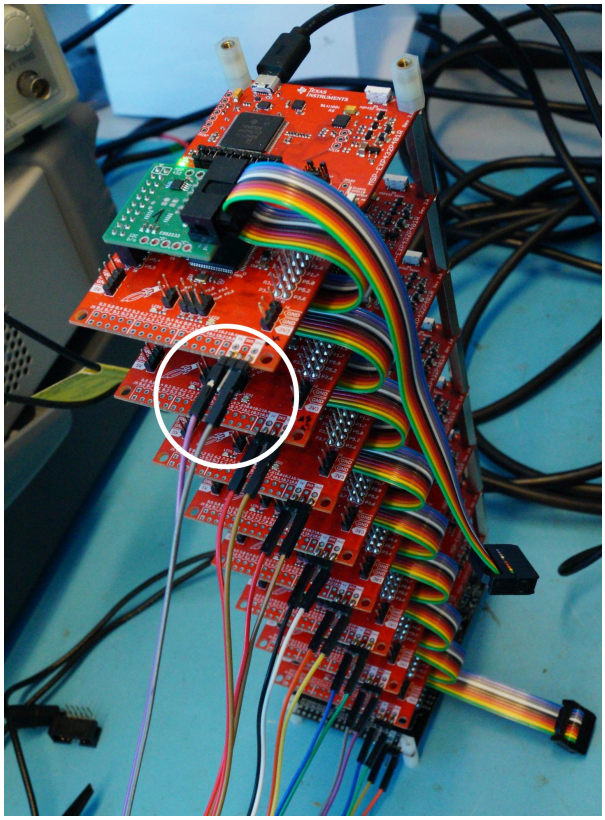


Figure 5.4: Photos showing the physical configuration of the Launchpad, including the stacking components and electrical harnessing. The rainbow harness is the interconnecting ribbon cable.

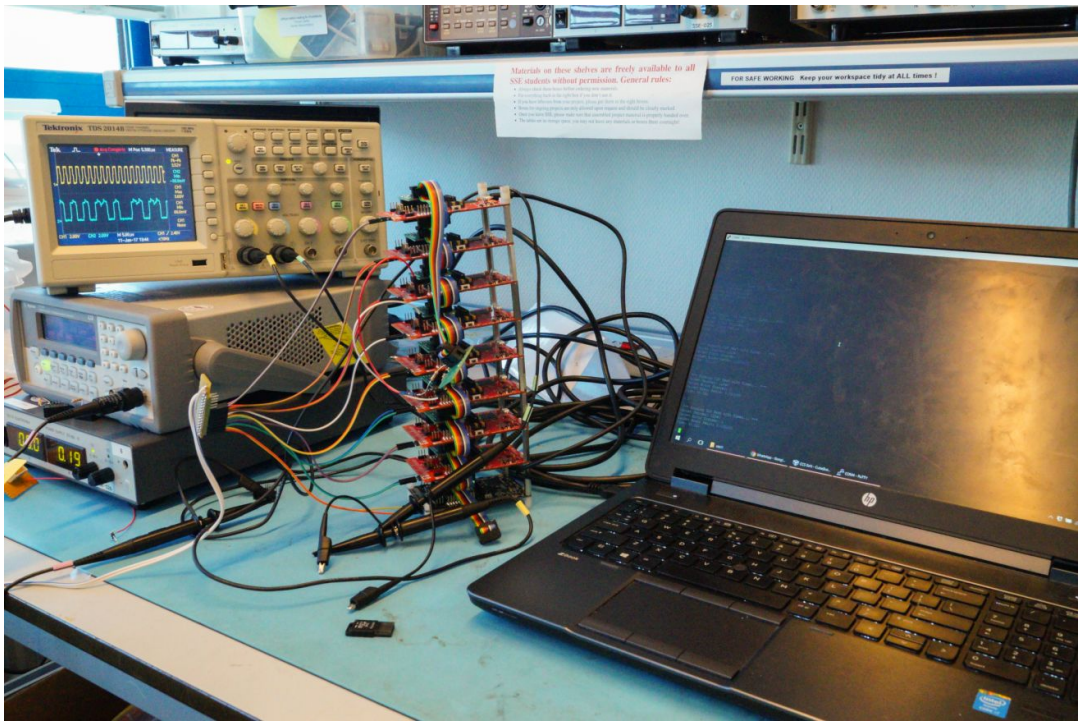


Figure 5.5: The experimental setup on the workbench. On the left are (from top to bottom) the oscilloscope, signal generator and power supply. On the absolute right hand side is the main computer. The data bus stack is in the centre.

Digital Multimeter [49]. The multimeter is placed in series in the main power supplying line to all the Launchpads, allowing accurate measurement of the overall power consumption of the bus. In an attempt to isolate the power consumption of each individual bus from the consumption of the Launchpad, 'null measurements' are performed of only the Launchpads without any attached daughterboards. The power consumption of a bus is then defined as the difference between the total power consumption with and without the attached bus daughterboards.

The MSP432 uses an Low-Dropout Regulator (LDO) regulator to convert from 5 V to 3.3 V. Therefore, to compute the power consumption of the bus, the measured current is simply multiplied with 3.3 V according to Equation 5.1 [26]:

$$P = U \cdot I \tag{5.1}$$

The power measurements will be performed five times for each bus in ten second intervals. This gives a statistical base for the measurements providing both an average value and a indication of the spread.

5.3. Central Controlling Computer

For logging and control purposes, a central computer is added to the simulator network. A UART serial connection is established between the bus master (a simulated on-board computer) and the controlling computer. To reduce delays due to the bus waiting for the UART, a slot is given for the transmission to take place. The transmission will then contain short reports of the status or changes to the status since the last transmitted report. This way the main bus drivers are not influenced by the diagnostic interface.

5.4. Simulated Subsystems

The basis of the test suite is a realistic representation of subsystems typically found in CubeSats. This is especially of importance in testing the TC bus, where data streams are usually not fully continuous as for the PL bus.

The selection of the subsystems for the TC bus case is based on Delfi-n3Xt, as a detailed description of its Housekeeping (HK) polling cycle is still available. The base composition consists of nine nodes:

1. **On-Board Computer (OBC)**: the OBC is the central node of the network. Its function in the simulator is essentially running the test: it requests HK data in a predefined cycle from the various subsystems, counts data errors and handles the serial communication to the central computer monitoring the tests. In all cases, the OBC is the central node in the bus architecture.

2. **TC Radio**: similar to having two separate data buses, it is assumed that a low-speed, highly reliable TC radio is included for basic communication between the spacecraft and the ground segment and a high-speed radio for payload (bulk) data.

3. **EPS**: in an actual satellite the EPS is responsible for switching the power supplied to subsystems. Although this functionality is not possible in the current setup, it is still included as a source of HK data.

4. **Attitude Determination and Control System (ADCS)**: the ADCS is usually a system demanding a lot from its connections to the various sensors and actuators present in the spacecraft. In this case, it is assumed that the ADCS uses a separate bus from the main bus, although using the same bus standard. The ADCS node is thus only providing basic telemetry.
5. **Payload**: although the payload data is not transported by the TC bus, it still requires separate commanding and housekeeping requests via the TC bus.
6. **Global Positioning System (GPS)**: used as a basic but accurate solution for Guidance, Navigation and Control (GNC), it is assumed the GPS provides near-continuous location and orbit data to the OBC.
7. **Propulsion**: as Delfi-n3Xt included an experimental propulsion system [50] and one is also targeted to be included in the Delfi-PQ, a propulsion system is added to the bus.
8. **PL Radio**: complementing the TC radio, this system downlinks bulk data over a high speed link. For example, an S-band or X-band link in place of a Very High Frequency (VHF) link may be used to enable a higher bandwidth.
9. **Mass Memory (MM)**: a MM systems is added to the bus, representing a system used to store payload data until it can be downlinked to the ground segment.

The order of the list shown above is according to their importance and likelihood of inclusion. Thus, when a test is performed on a spacecraft consisting of five nodes, the first five subsystems (OBC, TC radio, EPS, ADCS and Payload) in the list are included in the bus.

Note that similar systems to the Propulsion and PL Radio nodes are used for the PL bus simulations, as will be discussed in section 5.7.

Each subsystem is not simulated in a comprehensive way, but rather only as a source and sink of data. To enable the data transmissions, two types of transactions are defined: the data request and the data transfer. The centralised architecture of the TC bus means the bus transactions are coordinated by the OBC. A data request involves a 2 byte command packet transmitted by the OBC to the subsystem in question readying it. The subsystem then responds with the correct number of bytes. For a data transfer, the OBC simply transmits the data.

The ordering, size and direction (as based on the Delfi-n3Xt housekeeping poll cycle) of the different data transactions is presented in the list below. One must note that the majority of transactions consist of a 2 byte command and a larger response assumed to contain (pseudo) HK data. Only the last two transaction are simple data transfers.

- **OBC** → **EPS**: 2 byte command
- **OBC** ← **EPS**: 30 byte data package
- **OBC** → **ADCS**: 2 byte command
- **OBC** ← **ADCS**: 120 byte data package
- **OBC** → **GPS**: 2 byte command
- **OBC** ← **GPS**: 30 byte data package
- **OBC** → **Propulsion**: 2 byte command
- **OBC** ← **Propulsion**: 10 byte data package

- **OBC → TC Radio**: 2 byte command
- **OBC ← TC Radio**: 10 byte data package
- **OBC → Payload**: 2 byte command
- **OBC ← Payload**: 10 byte data package (note: this is only HK data)
- **OBC → MM**: 2 byte command
- **OBC ← MM**: 10 byte data package
- **OBC → PL Radio**: 2 byte command
- **OBC ← PL Radio**: 10 byte data package
- **OBC → MM**: 250 byte data frame
- **OBC → TC Radio**: 250 byte data frame

The total amount of data transmitted during each cycle when all nine nodes are included equals 748 bytes, or 5984 bits. Each transmission as shown above is defined as a packet.

The data as used in the test suite is a simple array defined in Read Only Memory (ROM) containing 250 random pre-generated bytes. How each driver transmits this data between the nodes is up to the driver itself. Thus, the transaction sizes mentioned above do not include any additional overhead data such as checksums and addressing.

5.4.1. Bus Duty Cycles

Three different duty cycle modes will be defined:

1. **Idle**: all bus hardware is powered and all drivers initialised, but no data is communicated over the bus
2. **Once-per-second**: the once-per-second case performs a full housekeeping cycle once per second
3. **Continuous**: the housekeeping cycle is repeated continuously

The way these cycles are implemented in software will be discussed in the subsequent sections. Regarding the metrics for the tests, the maximum data throughput will be measured using the continuous case, which attempts to come as close as possible to a 100% duty cycle. For the power consumption, all three cases are considered.

5.5. OBC Software Architecture (TC Bus)

Having defined which subsystems are included in the test suite, a more detailed description of the software is given in the following sections. Starting with the OBC, this system is significantly more complex than the other subsystems due to its importance within the bus architecture.

The OBC has several main tasks. Firstly, the OBC coordinates the flow of information over the bus, requesting and transferring data to and from each subsystem. Its second task is counting the errors and other types of failure during each test to provide an overview of a test containing specifically selected telemetry. The third and final task is providing an interface for the test operator to start or stop a test and view its results, in this case achieved through a serial monitor.

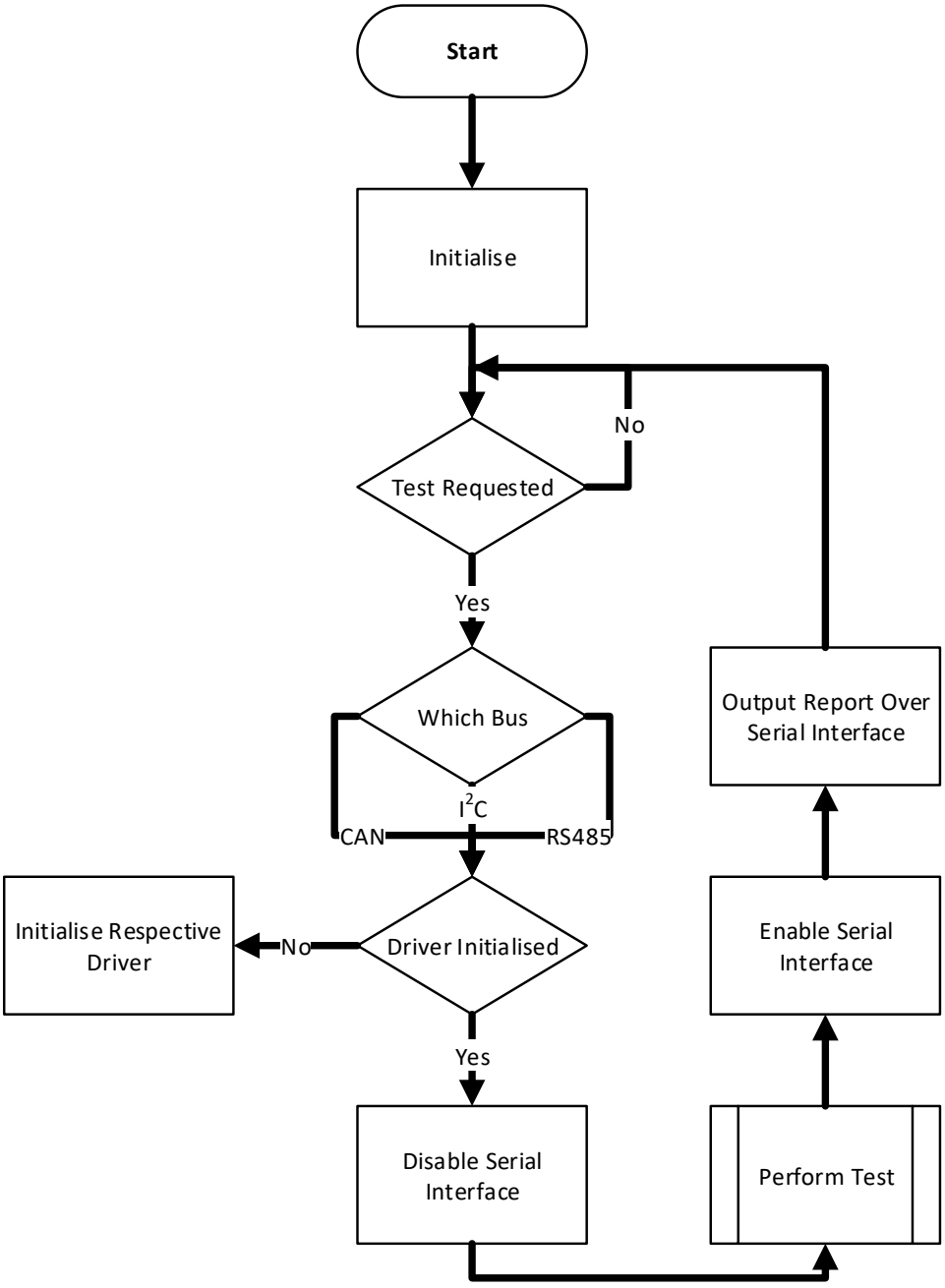


Figure 5.6: The top-level OBC architecture. Note that the 'Perform Test' block is described in subsection 5.5.1

```
**************************************************

   CubeSat Bus Simulator v0.1 (Nov 23 2016)

   Boot-count: 4

**************************************************

Menu:

 1) Display Boot Counter
 2) Reset Boot Counter

 3) Start I2C Test with Timer
 4) Start CAN Test with Timer
 5) Start RS485 Test with Timer

 6) Start I2C Test with Packet Limit
 7) Start CAN Test with Packet Limit
 8) Start RS485 Test with Packet Limit

 A) Start I2C Test for Power Measurement
 B) Start CAN Test for Power Measurement
 C) Start RS485 Test for Power Measurement

 9) Reboot

 0) Display this menu
```

Figure 5.7: The serial-based menu showing the different available test options and a boot counter

The top-level architecture of the OBC's software is shown in Figure 5.6. See Appendix B for a brief description of the flowchart's symbols. After booting the OBC, it first initialises the hardware shared by all buses (the necessary pins and peripherals), all variables and finally all the data sets. At this point, no bus drivers are yet initialised. One important part that is initialised is the serial monitor.

The menu offered to the user over the serial connection is extremely basic. An example of the main menu is shown in Figure 5.7. The MSP432's Enhanced Universal Serial Communication Interface (eUSCI) peripheral provides a UART, of which the output can be routed to the Launchpad's USB connection. The user or operator is able to select different tests by simply entering a single character representing each menu option. Once a test commences, the menu is disabled to prevent the user from interfering with the test by keeping the UART active.

To keep the system in a deterministic state at all times, use of the MSP432's low power modes are avoided. Therefore, the main part of the OBC's software is a simple loop continuously checking whether one of the tests has been requested, and running the test when that is the case. Once a test has been started, a check is performed whether the driver for the respective bus has been initialised previously. This makes it possible to have only a single bus driver ready at any time to give more accurate power consumption values.

Having confirmed the bus driver has been initialised, the actual test is performed. After finishing the test, the serial connection is re-enabled and a short report is transmitted to the serial monitor. This report, as shown in Figure 5.8, includes information on the number of packets transmitted, the amount of errors encountered and the time it took to perform the test. Naturally, this report is completely generated and transmitted after the test to avoid interference with the measurements.

```
> 3

 *** Running I2C Test with Timer... ***
Packet Counter: 28890
Packet Error Counter: 23112
Packet Error Ratio: 0.800000
Time: 30.006
```

Figure 5.8: An example of the report shown after finishing a bus test

5.5.1. Performing the Tests

Figure 5.9 shows the loop that runs each test on demand. Once the OBC enters the test, it is known which bus the test needs to be performed on. Furthermore, this bus is also already initialised. The test parameters for the test are derived from the test demanded by the user, and describe how long the test must run and/or how much data is transmitted over the bus. The main loop then runs the polling cycle as defined in section 5.4 while counting errors. At the end of each cycle, it checks whether the final criteria describing when the test must end are met. If so, then the OBC finishes the test. Three different tests are defined for each bus,

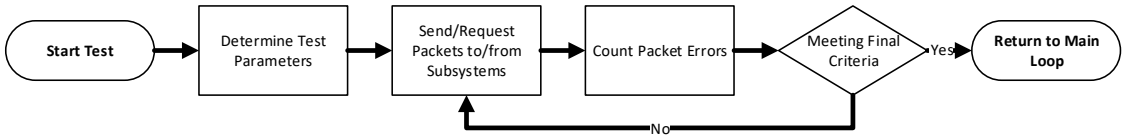


Figure 5.9: The OBC's test loop

all of which define different test parameters. Each test is essentially based on the same polling cycle, but how frequent this cycle is performed varies per test case.

The first case simply runs the polling cycle without intermittent delays until a set number of packets has been transmitted. This case is typically only used for verifying correct functioning of the bus, as it provides a short test of the bus.

The second case is similar: a 32 bit timer is set to trigger an interrupt. Until this happens, the test will run continuously. In most cases, the timer was set to trigger after 30 s to provide enough time to perform power consumption measurements and to provide a large enough interval to accurately average out the data throughput.

The third and final case does not run the polling cycle continuously, but rather when a 16 bit timer triggers. This timer is set to a 1 s interval. As a single polling cycle takes significantly less than this, the entire bus is polled reliably and accurately every second. This is the most realistic case as it comes closest to the actual operation of Delfi-n3Xt.

As the polling cycles may run in quick succession with identical (but random) packets, a check must be added to rule out the case where the OBC receives a packet in one cycle which was intended for a previous cycle. For this, the first byte in the data set used in each polling cycle is incremented each cycle. This effectively transmits a semi-unique identifier to each subsystem. The subsystem must read this byte, and transmit the response containing the same byte. The OBC verifies whether the byte in the response equals the current value of the identifier. If not,

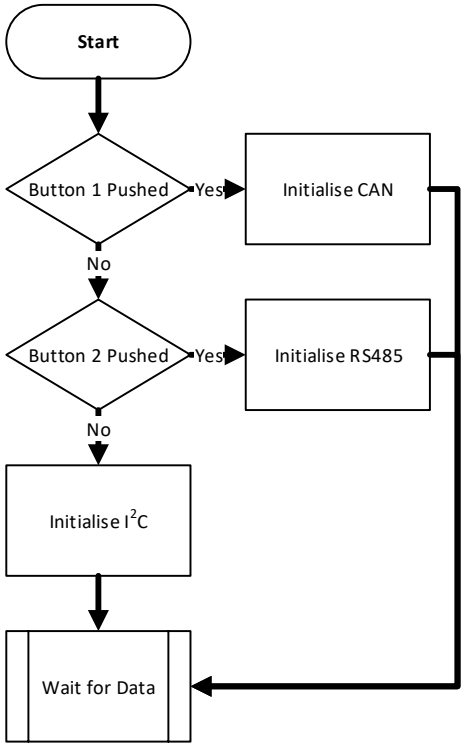


Figure 5.10: The subsystems' initialisation procedure

then it is marked as a packet error.

The drivers can also mark packets as a packet error. How this is done varies for each driver and is therefore described in each respective chapter. The same holds for how the drivers handle the transmission of the data packets.

5.6. Subsystem Software Architecture (TC Bus)

In most aspects each subsystem is simply a less complex version of the OBC, as it does not have to provide a user interface or timing mechanisms. However, a problem arises concerning the initialisation of only a single bus at a time. As initialising all buses is unwanted and causes problems due to missing hardware, each subsystem must be somehow signalled of which bus to initialise. As it is highly impractical to have a serial connection between the central monitoring computer and each subsystem, use is made of the two buttons which are added by default to the MSP432 Launchpad. The procedure describing how these are used is shown in Figure 5.10. One may note that each subsystem will by default initialise the I^2C driver. This choice was made to handle a bug in the MSP432 hardware which arised during testing and is described in chapter 6.

After initialising the requested driver, the subsystem simply waits until interrupts signal that a packet has arrived.

Once a message does arrive, the subsystem enters the process as shown in Figure 5.11. The respective bus driver is responsible for receiving the data and passing it on the the main

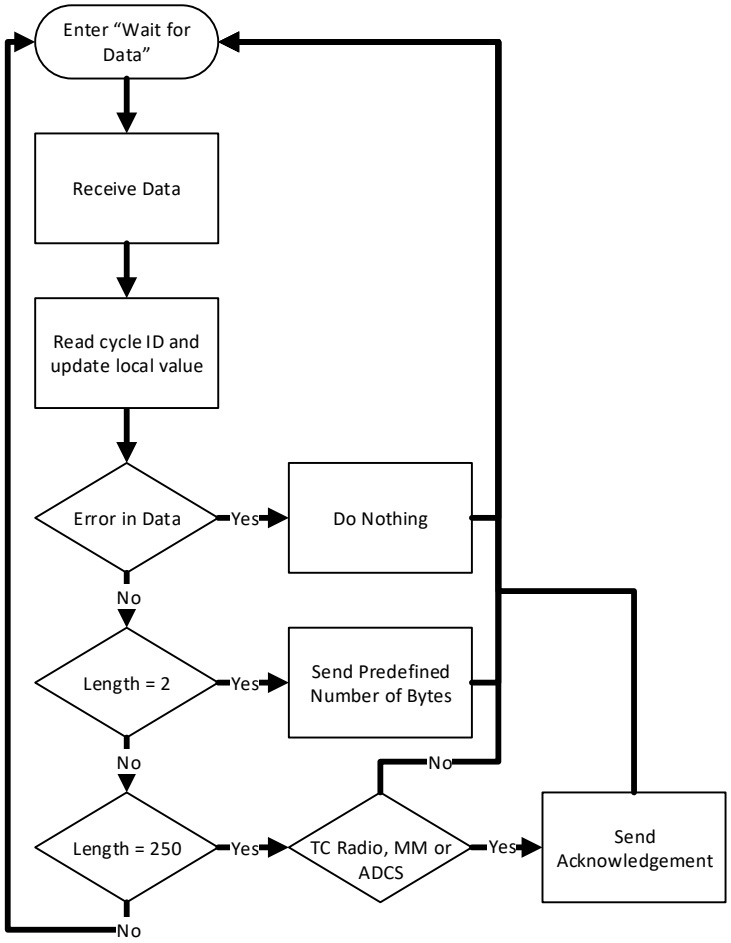


Figure 5.11: The subsystem handling an incoming packet

program. Once this is done, the local value of the polling cycle 'identifier' is updated to the received value (the first byte in the message, not including any overhead). If the bus driver reports some kind of error with the packet, such as a checksum error, then no packet is returned. The absence of a response will trigger a packet error in the OBC.

If there is no packet error, then the program starts analysing the message. Because no actual commands have been defined, the length of the message is verified. Each subsystem expects a 2 byte message from the OBC, indicating a data request. The MM and TC radio have an extra trigger: if the length equals 250 bytes, the transaction is a one-way data transfer from the OBC. Either way, a look-up table is used by the subsystem to find the amount of data it needs to reply to the OBC. In the case of a data transfer, a simple acknowledgement is sent to the OBC. How this is done varies for each driver.

5.7. OBC Software Architecture (PL Bus)

The previous sections have looked at the software architecture regarding the TC bus. However, the different setup in the PL bus case requires different software to run the tests. The

main difference between the two cases is that the former requires the data to be transmitted in packets with addressing overhead. The PL bus case, since it only consists of data moving in one direction, does not require either to function. All data is assumed to be transmitted in bulk, meaning that the BER can be measured directly according to section 4.2.

Similar to the TC bus case, a multitude of subsystems is defined. Nonetheless, as the PL bus consists of only two bus nodes, only two need to be defined. The first node is the OBC, which also runs the test in this case, requesting data from the second 'payload' node. However, the OBC in the test setup does not necessarily represent the actual OBC, but rather a subsystem such as a mass memory or payload data link. Still, the naming is chosen simply for consistency.

The data transmitted over each PL bus is an array of random bytes defined in the ROM of both the OBC and the payload. When the OBC requests data, a dedicated line is pulled low signalling the payload node to start transmitting as noted in Figure 5.12. Of course, in an actual spacecraft this signal would most likely be passed through the TC bus. The high pullup resistance of the MSP432 (30 kΩ [41]) only consumes at most 363 µA based on the default 3.3 V, thus its contribution is not significant to the overall power consumption.

Once data is received from the payload node, the data is first verified against the internal copy of the same data. Thus, no checksum or other similar indirect methods are employed to check the integrity of the data. The main benefit here is that the exact amount of bit errors can be counted during the entire test.

5.8. Subsystem Software Architecture (PL Bus)

The simplicity of the PL bus bus test is also reflected in the design of the payload node transmitting the data. As can be seen in Figure 5.13, the only task being performed by the subsystem is simply waiting until the data request line is pulled low. The microcontroller simply keeps on transmitting data until the line is pulled back high.

5.9. Conclusion

This chapter has looked at the hardware comprising the test setup and the software running the tests. What has been described here are the parts shared by all buses and bus drivers. However, the way binary data is transmitted between nodes and the necessary additional hardware varies wildly between the different buses. The next several chapters will define these parts and present the results from the performed tests.

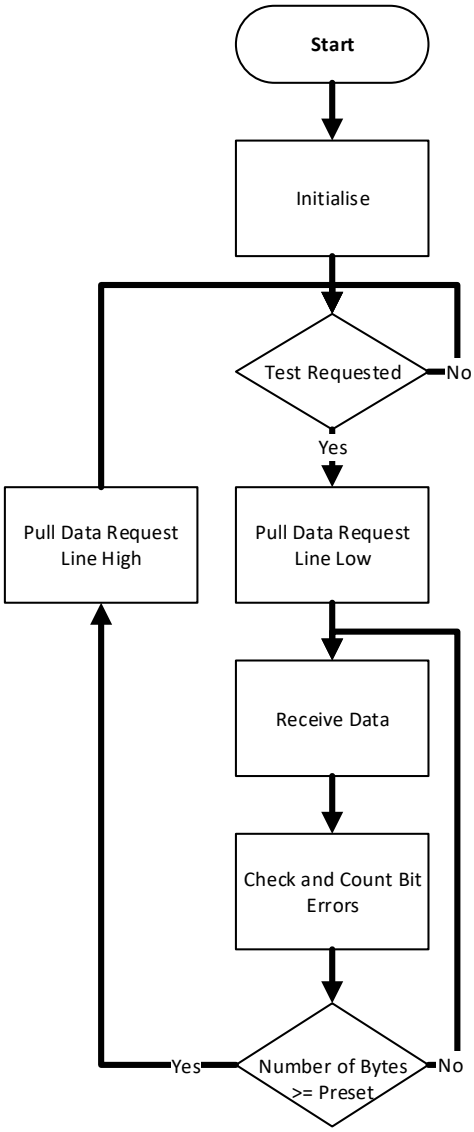


Figure 5.12: The software architecture of the OBC in the PL bus test cases

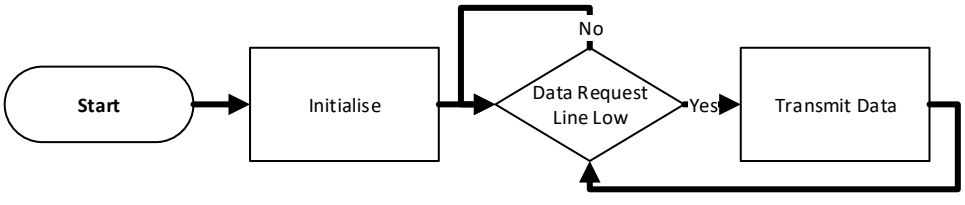


Figure 5.13: The software architecture of the payload node in the PL bus test cases

6

Inter-Integrated Circuit (I^2C)

For the first bus analysis, the Inter-Integrated Circuit (I^2C) standard is chosen. As I^2C is the most common choice for data buses in CubeSats, this standard essentially acts as a control group for the experiment, providing a minimum reference to which the other bus standards are to be compared.

This chapter will first give an introduction into the I^2C protocol, followed by selection of the required bus hardware and a description of the novel driver developed for the Delfi-PQ: DWire.

6.1. Introduction

The synchronous I^2C protocol was developed in the early 80s with its main purpose the inter-connection of Integrated Circuit (IC) on PCBs [28]. Although started by Philips Semiconductor (now known as NXP Semiconductors), the standard is now available under a “fair open policy”.

The original specification defined a baud rate of 100 kHz, but the *Fast Mode* introduced later on already increased it to 400 kHz. Even newer versions of the standard even improved the baud rate to over 1 MHz, however these speeds are not yet widely supported [5].

I^2C has only two distinct transmission lines in its physical layer, where it is possible to operate the bus in a linear topology. The Clock (SCL) line provides clock information and the Data (SDA) line carries the shifted out binary data. Both lines are pulled up to the reference voltage (Vcc) when the bus is inactive. Bits and clock ticks can then be transmitted by pulling the line(s) low, as may be seen in Figure 6.1. The designated bus master is responsible for all the bus timing and control. A bus slave is only able to transmit information on request and must shift out the serial data following the clock information provided by the master.

Regarding the data link layer of I^2C, a simple example is shown in Figure 6.2. Both lines start in a high state due to the connected pullups. The master starts a transaction by pulling SDA low, signalling the so-called START condition [5]. After a certain amount of time, the master will start providing clock ticks and shifting out data. The transaction is ended when a STOP condition is signalled, which is simply the opposite of the START condition and returning both bus lines to high.

To provide for address handling and defining the direction of flow of data, a basic protocol

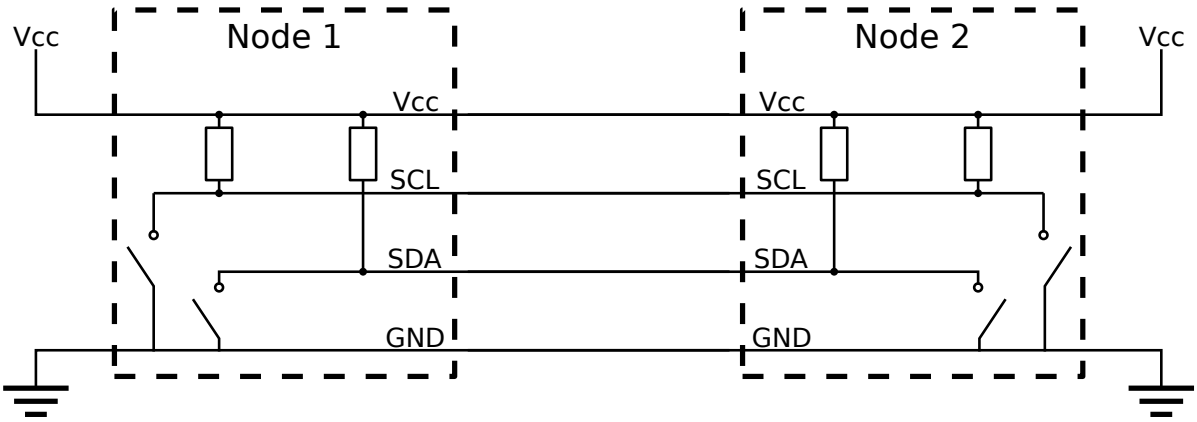


Figure 6.1: A basic schematic of driving the I²C lines

has been defined. This protocol is summarised in Table 6.1. By default, a 7 bit address is transmitted after the START to activate the correct bus node (although this can be increased to 10 bits in case more than 127 node addresses are required). Following the address, an additional bit (R/W) is transmitted telling the node whether the bus master will be writing data to or requesting data from the node. This is followed up by an Acknowledge (ACK) from the node, or not if something is wrong. This ACK requires the transmitting node to release the SDA line. The other node must then pull the line low, signalling the ACK. If this does not happen, it is defined as a No-Acknowledge (NAK). If the bus slave does indeed ACK, the data is sent in multiples of 8 bits plus an additional ACK from the receiving node (either the master or slave). Finally, the STOP finishes the transaction.

Table 6.1: The I²C protocol

Function	START	Address	R/W	ACK	Data + ACK	STOP
No. of Bits	1	7	8	9	$N \times 9$	1

Several special cases exist within the protocol, adding extra features to the bus. First of all, when a bus node cannot keep up with the data flow, it can pull the SCL line low after a clock tick. This so-called *clock-stretching* allows a bus node to temporarily halt the bus to prevent it from missing data. The second added feature is the *repeated start*: by sending a START instead of a STOP condition, the bus can be reset. This is useful when first a bus master writes data to a node and then wants to request data from the same node, reducing the total duration of the transaction.

The oscilloscope capture in Figure 6.3 shows an example of an actual I²C data transfer. The

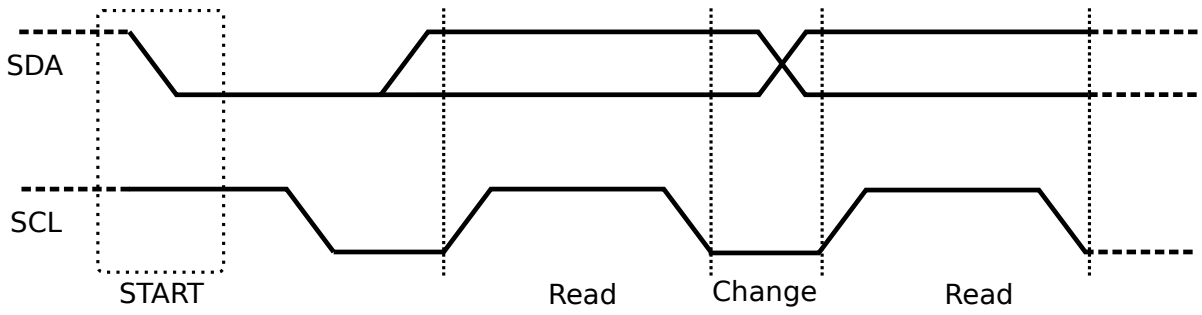


Figure 6.2: A basic I²C data transmission, starting with the START condition and the shifting out of two bits

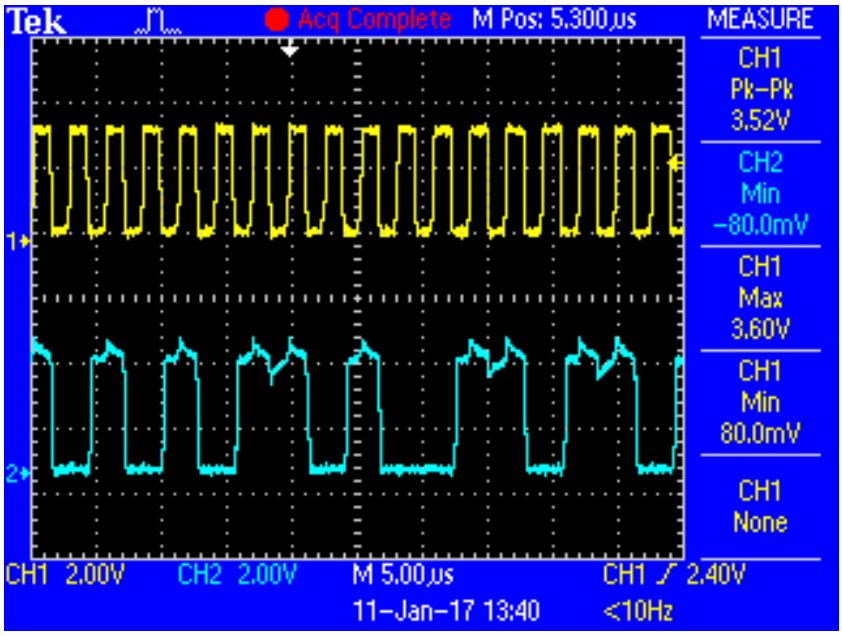


Figure 6.3: Oscilloscope capture of I²C signals. The top measurement shows SCL, the bottom curve shows SDA, synchronised to SCL.

SCL line can be identified from the highly consistent signal at the top of the plot. The SDA signal is changing much more irregularly. Cross-talk can be identified in SDA: the small peaks and troughs seen when the signal is high are perfectly synchronised to the rising and dropping of the SCL signal. Care must be taken in the design of a wiring harness or other signalling medium to reduce the amount of cross-talk between different communication wires.

6.2. I²C Daughterboard

The MSP432 microcontroller is able to drive three serial bus protocols through its built-in eUSCI modules: SPI, I²C and the UART. Thus theoretically, no external hardware is required to drive the bus. Nevertheless, experience from the previous Delfi missions has shown the need for I²C buffers [33], [51]. The buffer effectively removes loading effects in case of bus power-down or power-up events [16]. The buffer isolates a subsystem from the main bus, significantly reducing the amount of capacitance added to the overall bus: only the capacitance of the outbound side of the buffer is added. This results in increasing the possible number of bus nodes added to the network in regards to the electronic capacity. However, using buffers does of course not remove the limitation in number of addresses (127) available on the network, although it is not very likely CubeSats will reach this limit. Furthermore, when a bus node is power down, it is isolated from the rest of the bus. This avoids a blocked bus due to bus lines accidentally pulled to ground.

As the physical IC, the PCA9514A [32] manufactured by NXP Semiconductors is chosen. This component has previously been chosen for the Delfi-PQ and is relatively basic. Nevertheless, apart from normal I²C buffer behaviour it also supports *hot swapping*. Although actual physical hot swapping will never be necessary in a spacecraft due to the typical inaccessibility of the systems, it does mean that the bus is most likely more capable of handling sudden halts, reboots and power ups of bus nodes.

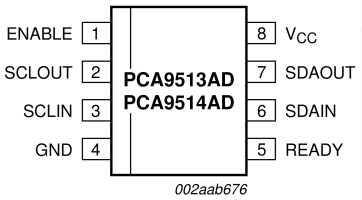


Figure 6.4: The pinout of the PCA9514A.

Figure 6.5 shows the I²C mounted on top of an MSP432 Launchpad.

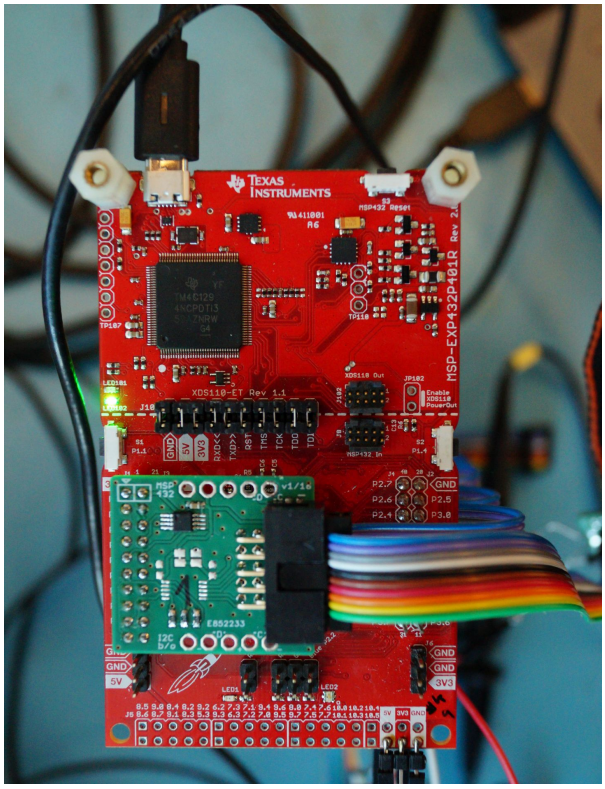


Figure 6.5: The I²C daughter board shown on top of an MSP432 Launchpad

6.3. Differential I²C

Apart from the standard, completely 'on-spec' implementation of I²C, an additional test is performed using differential I²C (dI²C). Although the protocol and the basic functioning of the bus driver are the same as with 'regular' I²C, the main SCL and SDA are transmitted in a differential form of larger distances (i.e. between PCBs).

No official standards exist for this type of implementation, but as a basis the NXP PCA9615 [52] is used as a driver. Figure 6.6 shows how two PCA9615s are connected. Even as the image shows a direct connection between the two nodes, dI²C can be applied in a linear topology just like regular I²C. The other characteristics of the PCA9614 are more or less identical to the (regular I²C) PCA9514A

A differential signal is defined as the signal resulting from the difference between two opposite signals, as shown in Figure 6.7. The figure shows a positively biased signals (A), a negatively

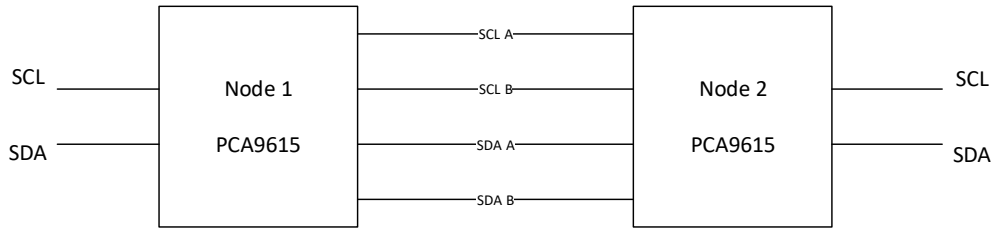


Figure 6.6: The basic architecture of the PCA9615 differential I^{2}C drivers

biased component (B) and the resulting signal. This has several benefits [16]: firstly, assuming that the wires carrying the two components are close together, any external noise or EMI will be cancelled out. Secondly, as the total voltage swing of the two components should equal the voltage swing of the main signal, that of the former will be smaller. This may lower power consumption and increase the maximum baud rate by shortening the signal's rise time.

An important addition to differential bus lines is the termination resistance. When the rise time of a signal comes close to the time it takes for the signal to travel from one end of the bus to the other, reflections start to occur [26]. Termination resistors reduce the effects significantly, especially when the resistors match the impedance of the bus lines. For regular I^{2}C, the pull-up resistors act as bus termination. As these are not in place with dI^{2}C, additional resistors must be added to the bus lines. Following CAN and RS485 standards [53], two 120 Ω resistors, one at each end of the main bus lines, are used as termination. Thus, the effective bus impedance is then equal to 60 Ω.

All tests performed on I^{2}C will also be performed on dI^{2}C.

6.4. Bus Schematics

To connect the signal generator necessary for quantifying noise effects to the dI^{2}C bus, the generator's output is connected through a very low-capacitance capacitor (1 µF)to a single bus line. The noise signal is still able to reach the other line of each pair through the termination resistors, simulating the coupled noise which is expected in a real spacecraft environment. A

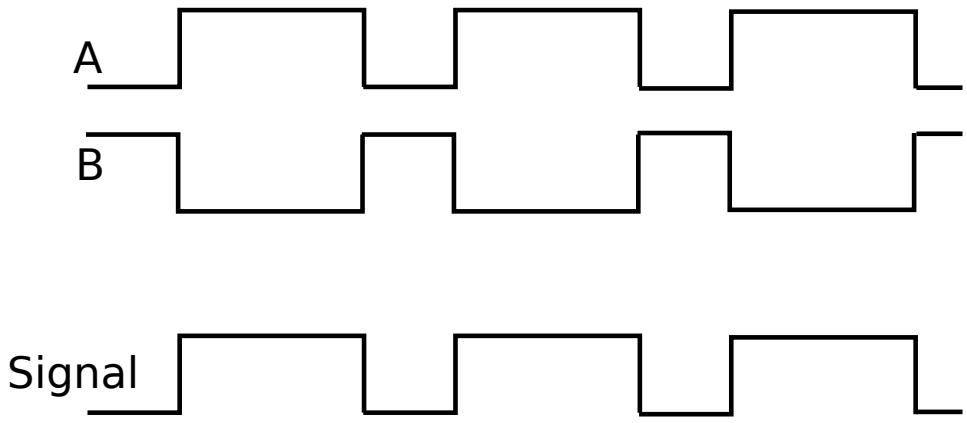


Figure 6.7: Differential signalling: A is the positively biased line, B is the negatively biased line. The difference between the two signals gives the actual signal

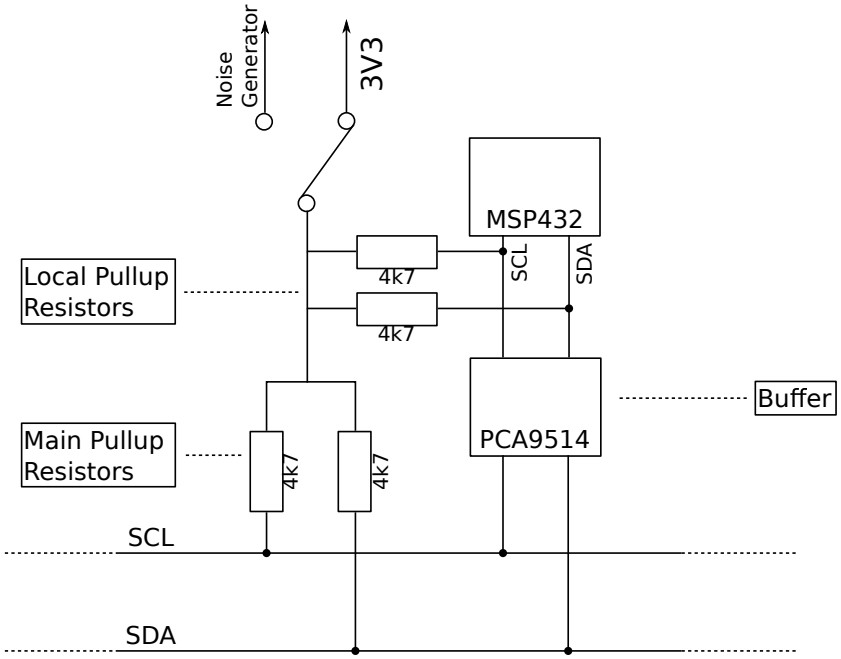


Figure 6.8: Schematic of a single node connected to the I²C bus with power source and pullups. Note the two sets of two pullups: one between the microcontroller and the bus buffer, and a second set to run the master bus lines.

slightly different design had to be applied to the regular I²C bus, because the bus is biased to 3.3 V by default. Applying a capacitance-coupled noise to this bus would result in a current flow to the signal generator when both the bus and the signal generator are idle. Therefore, a 3.3 V bias was introduced to the signal generator's output and used as a power source for the bus, as can be seen in Figure 6.8. When the noise function is switched off, the signal generator will still enable normal functionality of the bus. This does limit the (relative) peak-to-peak voltage of the white noise to approximately 5 V.

Note that the 4.7 kΩ pullup resistors were used in the regular I²C case and 600 Ω resistors in the dI²C case.

6.5. Bus Software: DWire

Before starting the main development on the I²C bus for the main experiment, an initial exploration of using I²C for the Delfi-PQ was performed. One goal of making the project more accessible to students of the faculty of Aerospace Engineering is making the software of the Delfi-PQ using Arduino projects. Energia [54] is an Arduino port for TI microcontrollers and is thus used for this purpose.

Like Arduino, Energia has a built-in I²C library, called Wire, to provide basic communication over an I²C bus. A simple example of using this library is shown below. The code snippet initialises the Wire library, starts a transmission to the node with address 4, sends five bytes of data in total, including the value of a predefined variable `x`, and finally ends the transmission.

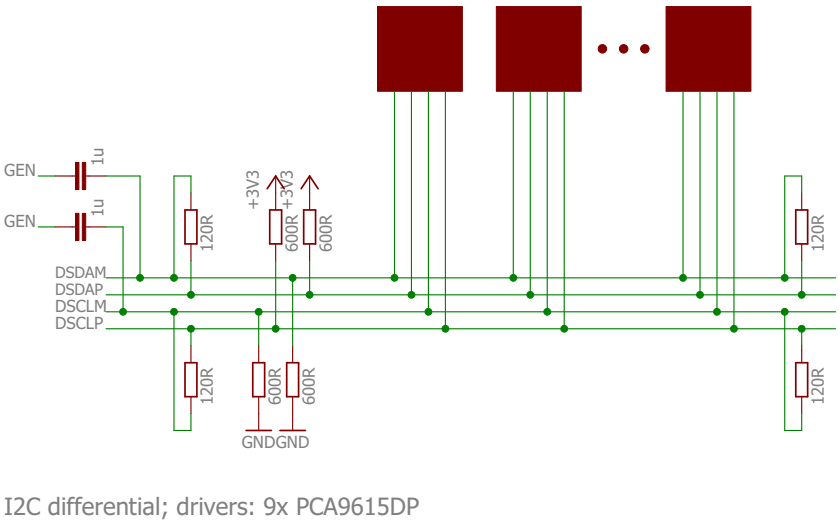


Figure 6.9: Schematic of the dI^{2}C bus showing the additional number of pullup resistors required. Nodes are shown as blocks, but have a highly similar architecture as shown in Figure 6.8.

```
Wire.begin();                    // initialise Wire
Wire.beginTransmission(4);       // transmit to device #4
Wire.write("x is ");             // sends five bytes
Wire.write(x);                   // sends one byte
Wire.endTransmission();          // stop transmitting
```

Unfortunately, the high simplicity of the Wire library means it is not customisable in use: even though the MSP432 has two eUSCI modules capable of driving I^{2}C, Wire uses neither but rather performs the protocol through software. This results in the library using an arbitrary baud rate of approximately 250 kHz instead of the standard 400 kHz. Furthermore, a node can only function as a bus master, regardless of the availability of slave-related functions in the library. There is also no way of connecting a single node to multiple I^{2}C buses.

To address these shortcomings and optimise the performance, the DWire (*Delft-Wire*) library [55] was developed. An important requirement of DWire is to provide backward compatibility with Wire. Furthermore, the library can be used in both Energia and as a normal stand-alone C++ library for use with for example TI's Code Composer Studio. Hence, the code snippet above will function identically when `Wire` is swapped with `DWire`. The extended functionality may be seen in the following code snippet:

```
DWire.begin( EUSCI_B0_BASE );  // initialise DWire with
                               // custom eUSCI module
DWire.beginTransmission(4);    // transmit to device #4
DWire.write("x is ");          // sends five bytes
DWire.write(x);                // sends one byte
DWire.endTransmission(true);   // end the transmission with a
                               // STOP condition. Passing false
                               // would allow a repeated START
```

The simplicity of the original library's interface is kept while also offering optional customisation. Other parts of the library enable interrupts to have a node function as a slave, and add accurate

customisation of the bus' baud rate as bus master. As may be seen in the code snippet, repeated starts are now also supported.

A small scale test was performed to help in the development of DWire using several I^2C-based sensors and to verify reliable functionality of the software. Eventually, the bus was able to run without lock-ups or bit flips for exactly two months before it was shutdown.

6.5.1. Issues Encountered During DWire Development

During the approximately five weeks required to develop the bulk of the code that would become DWire, many minor and several major issues were encountered. Some issues were due to the library on which DWire is built, the DriverLib developed by TI, was still being ported to the MSP432 platform. Hence, on more than one occasion, the available documentation did not match the actual Application Programming Interface (API). However, these issues were usually solved through analysis of the underlying code.

Larger issues were caused by the double buffer used internally by the eUSCI. A double buffer is usually implemented to avoid corruption of data: when the first byte of a data set is written to the buffer, it is immediately copied to the second buffer. The hardware performing the transmission (flushing) of the data uses the data from the second buffer. The software driver is then able to load in the second byte of data into the first buffer without disturbing the transmission of the first byte. However, the design of the MSP432 is designed slightly differently. The double buffer in the MSP432 is not automatically flushed when transmitting bytes, but only commences once the first buffer is filled. This more or less means the double buffer acts as a single buffered system with a delay of the duration it takes to transmit one byte. This also adds a secondary problem, as it is not possible to transmit only one single byte: this byte would become 'stuck' in the secondary buffer until a second dummy byte is written to the buffer.

A matter which complicates the double buffer problem is that when a STOP (or START in a repeated START) is requested, this is handled after the current byte has been fully transmitted, regardless of whether the first buffer is full. This requires careful timing when setting the STOP condition to avoid cutting of a transaction prematurely.

The double buffering issue proved to be a source for many different problems and inefficiencies in the library. At several points one problem would be solved when communicating with an MSP432 slave connected to the bus, but unfortunately breaking the ability to communicate with sensors (which do not have a double buffering system). In the end, tweaking the reaction speed of interrupts and adding a 'smart' delay (which is able to wait for a single bit-time regardless of clock speed) made the library functional.

6.6. DWire in the Test Suite

Having established a suitable library for running I^2C, it must be implemented within the TC bus test suite as defined in chapter 4. As the test suite software is responsible for creating the correct packets and data, DWire is only necessary to actually carrying the binary data between bus nodes.

One requirement of a realistically implemented databus is to be able to verify the integrity of the transmitted data. As noted in Table 6.1, I^2C does not have a built-in mechanism to check for bit errors. Therefore, a 16 bit (two bytes) CRC (CRC-16) checksum is added to each

transmission. The reason for the choice for CRC-16 is that it can be computed by default using the Cyclic Redundancy Check (CRC) peripheral onboard the MSP432. Moreover, an 8 bit CRC only has $2^8 = 256$ combinations, while CRC-16 has $2^{16} = 65\,536$ combinations. Thus, CRC-16 significantly decreases the probability of missing a bit error due to colliding CRC checksums.

6.6.1. Handling Data Transfers

As mentioned in chapter 4, two transaction types can be identified: data transfers and data requests. The former is a straightforward transfer of data from the OBC a second subsystem. A data request is a data transfer in the opposite direction. The master-centric nature of I^2C means that both types are completely handled by the OBC.

Figure 6.11 contains the process flow of the OBC during a data transfer. As an input, the data from the testing software is received. The first step is to compute the CRC-16 of the given data and combine this into a packet with the data itself. Using the standard DWire `beginTransmission` and `write` functions, the data is transmitted directly to the targeted subsystem node with an address obtained from a lookup table. If an error occurred during transmission, such as a locked bus line or no ACK after the address segment of I^2C, then an error is returned. After transmitting the data, an ACK is requested from the subsystem to verify the transaction result. Note that this ACK is separate from the ACK already present in the I^2C protocol. If this ACK is not received correctly, then the transaction is marked as a packet error.

The workflow of the subsystem handling a data transfer is shown in Figure 6.11 and can be viewed as the opposite of the workflow of the OBC. The data is received by DWire and passed on to the higher driver layer. The packet is read, the CRC-16 is verified and an ACK is made ready to be transmitted to the OBC if the data was received correctly. This is the point where in a real data bus, if a packet error is detected, the failure would be handled. In most cases, this would imply a simple retransmission of the packet, although the exact way of handling can be different depending on the context.

6.6.2. Handling Data Requests

For a data request, most of the workflow of the OBC and the subsystem are identical to that of handling a data transfer, and is thus not described in detail through flowcharts. The main difference between the processes of the data transfer and request is that in the latter case, the OBC performs an I^2C read action of a known amount of bytes plus two bytes for the CRC. If an I^2C error occurs or a bit error is detected in the packet, then the packet is marked as a packet error immediately. Because all actions are performed on the OBC side: there is no need for a separate ACK.

6.7. Results

The various metrics as described in chapter 4 have been tested on both the regular I^2C bus and the dI^2C bus. The results will be discussed in this section.

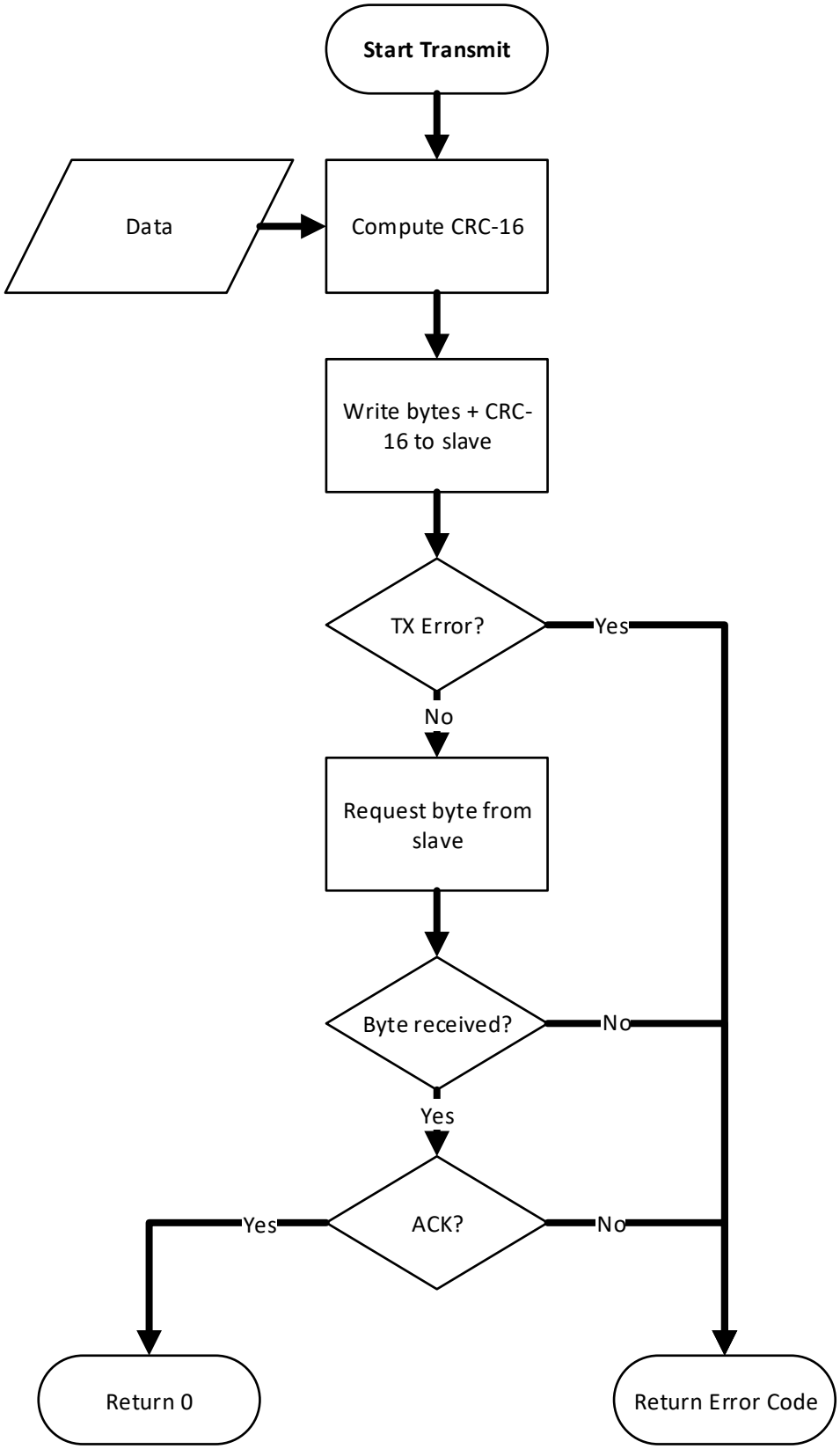


Figure 6.10: The workflow of the OBC during a data transfer over I^2C.

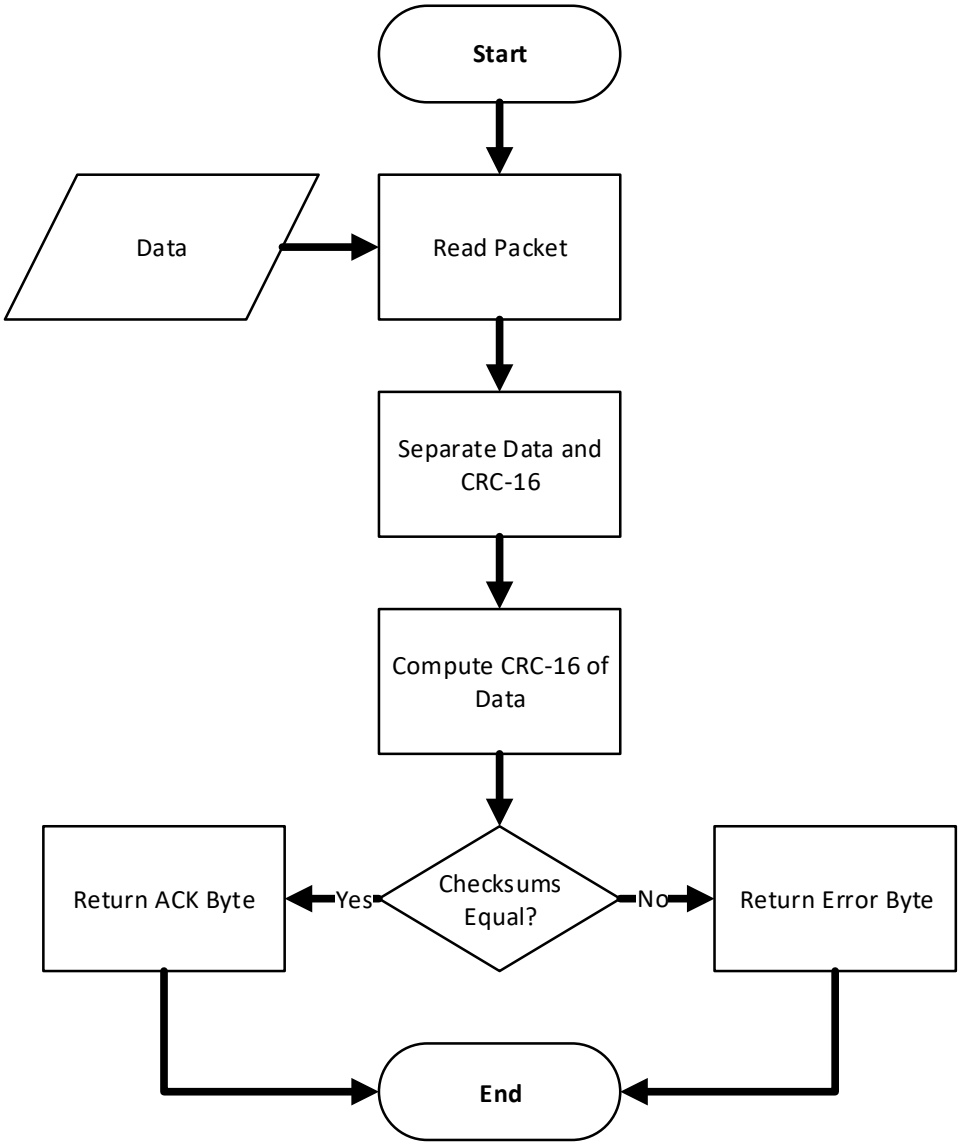


Figure 6.11: Functional breakdown of the OBC processing a data request

6.7.1. Data Throughput

Perhaps the most basic test is the data throughput test. For I²C, the maximum throughput was measured to be 250.4 kbit/s $\pm$ 0.008 (3σ). For dI²C, it was measured to be 256.9 kbit/s $\pm$ 0.007 (3σ). These values mean a 62.6% efficiency and 64.2% efficiency respectively. The difference is most likely due to the lower overall voltage swing of I²C, slightly reducing the length of delays between transactions and bytes on both the SCL and SDA channels. Note that this would not reduce the bit time, as this is coupled to the clock frequency of the microcontrollers.

6.7.2. Packet Error Ratio and Noise Immunity

In a normal ambient environment, the PER of both I²C and dI²C has been determined to be less than 10^{-4} with 95% confidence according to the method described in section 4.3, compliant to the predefined threshold value.

It was expected that the behaviour of the PER would be linear compared to the white noise RMS voltage. However, a very clear 'cliff' is apparent in the noise measurements. I²C did not show any persistent bit errors happening at realistic RMS voltages. At no particular point did the PER of I²C exceed the minimum of one error in 10 000 packets to say that the BER is less than the required 10^{-4}. It is suspected that each single bus line, with its specific bus capacitance and pull-up resistors, acts as a band-pass filter.

Exposing I²C to positive transient peaks into both bus lines, no packet errors were detected. With negative transients however (pulling the lines to ground, to which the I²C is expected to be more susceptible), 3.62% of the packets failed to be transmitted successfully, starting at 2 V peak-to-peak (with 24750 tested packets).

Regarding dI²C, this bus did not show any packet errors with injected white noise until 0.8 V RMS, where the bus completely locked up on each try. However, with this noise level, the peaks of the signal already reach around 10 V, exceeding the maximum allowed voltage on the PCA9615 by at least 4 V. Applying the transient peaks to the dI²C bus gave similar results: no errors until the bus completely locked up at 5 V peak-to-peak.

The lock-ups experienced with the dI²C bus were found to be quite difficult to fix: the lock-ups were found to be the MSP432 actually entering a hard fault mode. This implies that these tests exposed a bug in either the MSP432's firmware or hardware. Solving such a lock-up situation in a satellite would only be possible by power-cycling the respective board, either by detecting the hard fault or through a watchdog mechanism. The board's design means a power cycle also resets the I²C/dI²C buffer, possibly resolving it from a locked state.

The microcontroller's hard fault is the reason why each subsystem board automatically boots into I²C mode: when a hard fault is detected, the board is completely rebooted. As these hard faults were only found with I²C, it was sufficient to have the subsystems simply boot into I²C.

One possible explanation for the difference between I²C and dI²C with the noise effect tests is the design of the rise time accelerator included with the PCA9514 (regular I²C): according to its datasheet [32]: *"the rise time accelerator will clamp the [excessive] voltage to the positive supply rail."* In other words, the rise time accelerator is able to counter sudden spikes in the driver's inputs. The PCA9615 on the other hand, does not have a rise-time accelerator/regulator mentioned by its datasheet [52].

Even though the significance of the results of these noise effect experiments are debatable

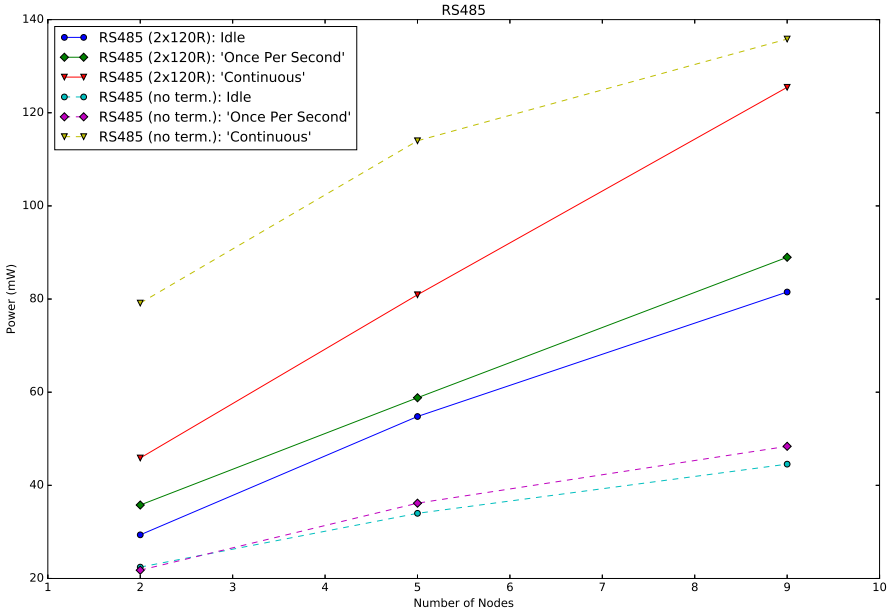


Figure 6.12: Plots of the I^{2}C and dI^{2}C power consumption for two, five and nine nodes. Note that the connecting lines are included to show trends and are not meant for interpolation

due to the extreme voltages until failures occurred, it has been found that they are extremely beneficial to the development of the software drivers. Early experiments exposed flaws in the software which were not found during the two month continuous test of the DWire library. In nearly all cases, problems arose due to loops polling certain register values or interrupt flags becoming stuck. Adding the extreme transients also proved to be a useful tool to find effective time-out strategies. As mentioned above, the only bugs left were in the lowest levels of the MSP432: bugs which most likely would not have been found otherwise.

6.7.3. Power Consumption

The measured power consumption for the entire I^{2}C bus and dI^{2}C bus are shown in Figure 6.12. Error bars have not been included in the plots, as the maximum 3σ error equals 0.6 mW and would simply not be visible.

The general trend of both buses is the same: fairly linear behaviour of the power once more nodes are added to the bus. Because nodes are completely disconnected from the bus when not included in the test, adding them to the bus noticeable increases the bus capacitance. In turn, an increased capacitance will decrease the rise time of the bus lines, meaning the mean power of the signal decreases somewhat. This is visible in the slight downward trend for both I^{2}C and dI^{2}C. The effect is stronger for dI^{2}C due to the lack of a rise time regulator.

Preliminary conclusions that can be made from this plot is that for relatively low duty cycles, dI^{2}C would be the preferred choice, although I^{2}C's power consumption is slightly more stable for varying bus duty cycles.

6.7.4. Complexity

The complexity of I^{2}C is difficult to assess, as one hand the physical layer of both I^{2}C and dI^{2}C are straightforward to implement. Furthermore, since I^{2}C is supported by most microcontrollers, basic software can be developed relatively quickly. However, the first problems will arise once multiple systems developed by different manufacturers must communicate together. The double buffering issue of the MSP432 as experienced during the development of DWire is an excellent example of this phenomenon. Moreover, previous experiences [51] have shown that it is difficult to implement reliable fault detection (watchdog) mechanisms.

6.8. Conclusion

This chapter has discussed the design and implementation of the DWire library, the bus hardware used and the results of the experiments.

It is clear that the use of a rise time regulator which clamps input voltage has a beneficial effect on the bus reliability. The noise and transients applied to the I^{2}C and dI^{2}C were therefore found to have a much larger impact on the functioning of the latter bus than the former.

Especially the use of transients at a relatively high frequency has proved to be useful to develop and debug software for operating a data bus. Even though the applied values during this experiment were quite extreme, it provides ample opportunities for finding bugs in the drivers, and testing failure detection and handling.

Including a rise time regulator in an I^{2}C is found to be beneficial to both the susceptibility of the driver and bus to noise and transients, and has probably stabilised the power consumption at varying bus duty cycles.

The power consumption of dI^{2}C is found to be less at low bus occupancy. However, once the bus approaches a full duty cycle the roles switch around. Together with the additional hardware and ‘off-spec’ hardware drivers required for dI^{2}C, it might provide reason to choose I^{2}C over dI^{2}C. However, other similar implementations of dI^{2}C may prove otherwise.

Benefits of differential signalling were already seen, especially when comparing I^{2}C and dI^{2}C: it is superior in terms of cross-talk behaviour and also theoretically reduces in other external EMI. However, when specifically applying dI^{2}C, the weak link remains the standard I^{2}C bus between the node’s microcontroller and the bus driver. Moreover, no formal standard for dI^{2}C exists, making a guarantee of compatibility between systems developed by different parties difficult.

7

Controller Area Network (CAN)

This chapter will look at the Controller Area Network (CAN) bus, the implementation details and results from the analyses. It will start with a brief introduction on the bus standard and implementation (its physical and data link layers) and a description of the custom software drivers, followed by a presentation and discussion of the results.

7.1. Introduction

In the eighties and early nineties, the steady increase in the number of microcontrollers included in new cars lead to problems: the differences in wirings between unique subsystems and simply the absolute amount of wires began to grow out of control [56]. Furthermore, the time-criticality of several systems, such as engine controls and control of a vehicle's dynamics, means that a reliable interconnecting bus is necessary. Several bus standards were developed, from which CAN emerged as the industry standard [56].

The main reasons for CAN's popularity are its flexibility and low costs, resulting in a large variety of manufacturers offering compatible electronics. CAN features two main speed modes: a low speed mode (up to 125 kbit/s) which supports *time-triggered* applications: messages can be transmitted at highly regular intervals. An example is the speedometer of a car, requiring a steady rate of updated measurements. The second mode is the the 'High Speed' mode (up to 1 Mbit/s) for applications with a larger bandwidth.

The popularity of CAN has led to a large variety of industries adapting the bus. It has also been used in CubeSats and larger spacecraft [6], but results and reflections of the implementation are lacking.

7.1.1. Physical Layer

The physical layer of CAN is based on differential signalling (a basic description of differential signalling is found in section 6.3). CAN requires the same number of wires as I^2C: the two differential lines do not require an additional clock line. Two different voltage states are defined by the CAN bus: the 'dominant' state and the 'recessive' state [57]. The former is where the positively biased signal CAN-High (CANH) is driven high and the negatively biased signal CAN-Low (CANL) is pulled low. This state equals a logical 0. The recessive case is were both

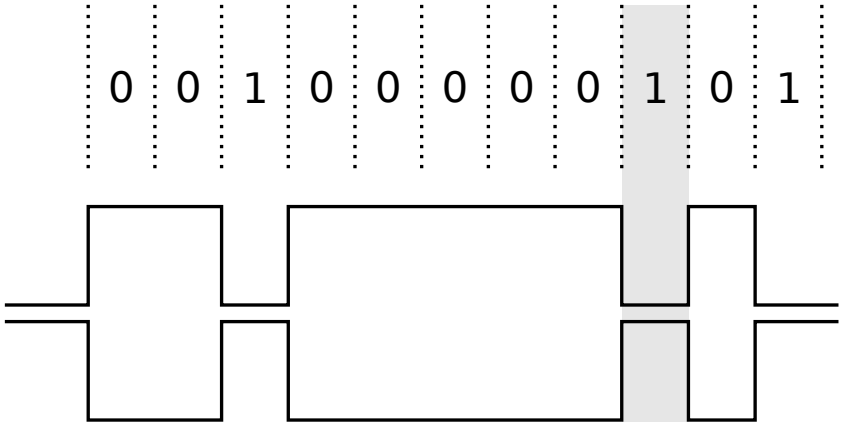


Figure 7.1: A schematic view of CAN's signalling, showing the dominant (0) and recessive (1) states. The highlighted bit is a stuffed bit ensuring correct synchronisation between nodes

CANH and CANL are left idle, essentially staying at the same voltage. The required timing information is included in the signal itself.

The different nodes on a CAN bus network are synchronised to the edges of each bit. However, the Non-Return to Zero (NRZ) scheme employed by CAN, where the bus does not return to an opposite state between identical bits, means nodes might lose synchronisation if the same bit is transmitted in series for too long. As is shown in Figure 7.1, so-called bit stuffing is used to force shifting of the signal to re-synchronise the bus nodes by adding an extra bit opposite in value. A stuff bit is required after every five identical bits [57], and should be removed from the data stream before interpreting it.

A CAN bus can be operated in a linear topology, where the bus lines must be terminated with (typically) two 120 Ω resistors. Each separate node requires a CAN controller to handle the signal generation and receiving. To drive the bus, a separate transceiver is required. The transceiver takes the generated signal from the CAN controller and converts it into a differential signal and vice versa. Furthermore, the transceiver acts as a buffer, electrically isolating a CAN controller from the main network. This is highly similar to the function of buffers in an I^2C bus.

7.1.2. Data Link Layer

The protocol of CAN is based on relatively short messages contained within frames [57]. The contents of one frame is summarised in Table 7.1. The first bit is the synchronisation bit and is used to prime and synchronise the timing of all bus nodes. The next block of bits is the arbitration block. As CAN is a multi-master bus, the arbitration block is used, as the name implies, to arbitrate the bus. When two frames start transmission at the same time, the frame which first reaches a recessive bit wins the arbitration. Apart from the inherent message priority given in the arbitration block, it is also used as the message identifier. Within this thesis research, the message identifier is treated as the address of the recipient. However, the designer might choose to enable broadcasting of the message to multiple recipients, in which case the identifier would tell bus nodes the contents of the message. By default, the message identifier is 12 bits in length, but it can be switched to an 'extended' 32 bits if more values are required.

The control block following the arbitration block contains more information on the type of message [57]. Firstly, it contains two 'reserved' bits used by the controller internals, followed by a

Table 7.1: Summary of the protocol used by CAN, excluding bit stuffing

Function	Sync	Arbitration	Control	Data	CRC	ACK	End-of-Frame
No. of Bits	1	12 (standard) 32 (extended)	6	0 - 64	16	2	7

bit flag indicating whether the message identifier is of the 32 bit or 12 bit type. Thirdly, a bit flag indicates whether the frame is a so-called remote frame or not. The former type is used when a node requests information from another node. The last two bits indicate the number of data bytes in the message, which must be zero when it concerns a remote frame.

After the control block follows the actual message data if applicable. The maximum size of a data frame in CAN is 64 bits, or eight bytes. Thus, larger messages must be split into smaller parts. A CRC checksum of entire frame until the CRC block itself, to be verified by the recipient. The CRC is followed by an ACK, given by the recipient. Finally, the End-Of-Frame (EOF) ends the transmission. The EOF provides a guaranteed idling time between frames, and does not carry any additional information [57].

7.2. Error Handling

One of the main advantages of CAN is the error handling. The error detection and handling consists of several segments: error detection, error handling and fault isolation. The autonomous operation of these segments is one main reason for the CAN bus' reliability.

7.2.1. Error Detection

A CAN controller can detect errors in frames and data using several direct and more implicit methods [56]:

- CRC error: a non-matching CRC is a very direct way of detecting bit errors in the transmitted data and can indicate a bit error in the data, but also, for example, an incorrect message identifier.
- Stuff error: when more than five bytes are detected in sequence without stuff bits, a bit error must have occurred.
- Form error: incorrect placement of blocks within a frame can be detected.
- Bit error: although the naming is very generic, this implies that a transmitting node (which also listens in to its own transmission) detects a different bit value than it transmitted.
- ACK error: a frame has not been ACK'ed.

7.2.2. Error Handling

When an error has been detected by a node, it transmits a predefined error frame [56]. This error frame tells every node on the bus to reject the current frame. The frame in question is then retransmitted automatically.

Table 7.2: The two stand-alone CAN controllers under consideration.

Product	Supplier	CAN Speed	TX Buffers	# RX Buffers	Input Bus	Power (operating)	Min-Max Temp
MCP2515	MicroChip	1 Mbit/s	3	2	SPI	25 mW	-40 to 125 °C
SJA1000	NXP	1 Mbit/s	1	1	Parallel	75 mW	-40 to 125 °C

Table 7.3: The CAN transceivers under consideration.

Product	Supplier	CAN Speed	Power	Min-Max Temp	Remarks
TJA1051	NXP	2 Mbit/s (CAN FD)	250 mW	-40 to 150 °C	5 V supply and logic level
TJA1050	NXP	1 Mbit/s	250 mW	-40 to 150 °C	5 V supply and logic level
MCP2551	Microchip	1 Mbit/s	unknown	-40 to 125 °C	5 V supply and logic level
MAX3051	Maxim	1 Mbit/s	175 mW	-40 to 85 °C	3.3 V supply
SN65HVD23x	TI	1 Mbit/s	33 mW	-40 to 85 °C	3.3 V supply
MCP2561/2	Microchip	1 Mbit/s	225 mW	-40 to 150 °C	5 V supply and logic level
PCA82C251					Disregarded: for 24 V systems
TJA1145					Disregarded: for 12 V/24 V systems

7.2.3. Fault Isolation

Once the error frame has been transmitted, each node affected by the error increments an error counter. Once the counter exceeds a certain predefined amount, a node will enter the *error-passive* mode [56]. In this mode, it will only continue counting errors in its own transmitted frames. If the error counter then exceeds a secondary, higher threshold, the node will enter the *bus-off* mode, in which the CAN controller will completely cut off from the rest of the bus. In this case, the node can only rejoin the bus after some form of reset performed by the node's microcontroller.

7.3. CAN Controller and Transceiver

A survey of some common standalone CAN controllers and transceivers was performed. Table 7.2 contains the resulting controllers, while Table 7.3 contains a list of CAN transceivers. Both tables have been populated using values taken from each corresponding publicly available datasheet.

It is clear that there are many more transceivers available than controllers. This is most likely because there are several commercially available microcontrollers with built-in CAN controllers. Although these could be considered here separately, one is reminded here that the goal of the design is to provide a bus which is independent from the chosen microcontrollers. It is expected that the added complexity and additional development time by selecting a dedicated microcontroller will outweigh any benefits.

Regarding the two CAN controllers in Table 7.2, the MCP2515 is viewed as better performing on nearly every chosen criterion: it contains more Transmit (TX) buffers, more Receive (RX) buffers than the SJA1000 and consumes less power. Finally, using a (serial) SPI interface requires less pins than using a dedicated parallel interface, with no real expected loss in performance. Therefore, the **MCP2515** is the most straightforward choice.

The second task is to select a CAN transceiver from the list in Table 7.3. The PCA82C251 and TJA1145 are both designed for use with a 12 V or 24 V power supply, which is a standard voltages commonly found in cars and trucks, derived from the operating voltages of car batteries. This relatively high voltage is impractical in nanosatellites. Several other options

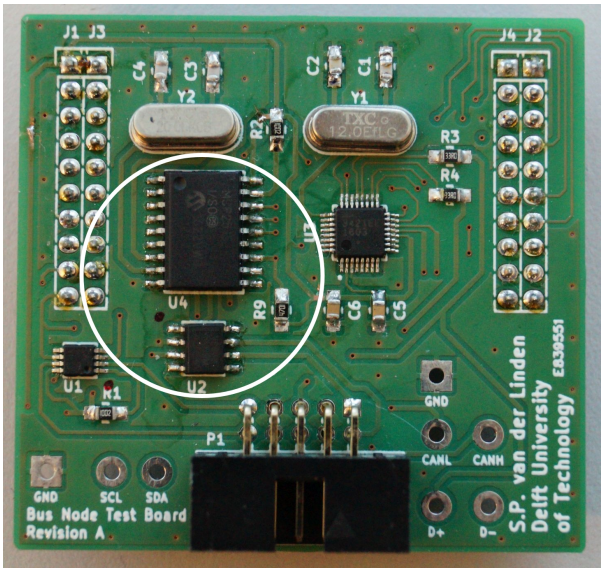


Figure 7.2: The CAN controller and transceiver as placed on the combined CAN/USB daughter board

require a 5 V supply. This voltage is already much more practical and can be supplied through the TI Launchpad. However, since the MCP2515 and MSP432 both have 3.3 V logic voltage levels, a level translator is necessary to drive these transceivers. Since the addition of the extra IC will only increase the power consumption, which is already higher than the two other 3.3 V transceivers, the 5 V transceivers are removed.

A special mention must be made of the TJA1051, which supports Controller Area Network Flexible Datarate (CAN FD). Although this is not yet supported by the CAN controllers themselves, it is expected that CAN FD will increase in popularity once it will be fully adopted by the automotive industry.

The last two transceiver options left, the MAX3051 and SN6565HVD23 series, both consume considerably less power than the other options, at the cost of a lower maximum operating temperature. This difference is not considered to be a problem for this databus test, but might be a killer requirement when used in actual spacecraft. According to its datasheet, the SN65HVD23 has a lower typical power consumption, leading to the choice of the **SN65HVD23** transceiver to be used in the bus hardware.

7.4. Bus Schematics

The schematic in Figure 7.3 shows the architecture of the CAN bus. This same architecture is used for the TC bus case as well as the PL bus case.

Noise and transients are again injected into a single bus line, as the electrical context is identical to that of the dI^{2}C test case.

7.5. Software Driver Architecture

The MCP2515 features an SPI driven interface used to configure the controller. This interface provides access to the register system used by the CAN controller, and is complemented with several dedicated commands for retrieving status and result codes. The SPI clock speed is

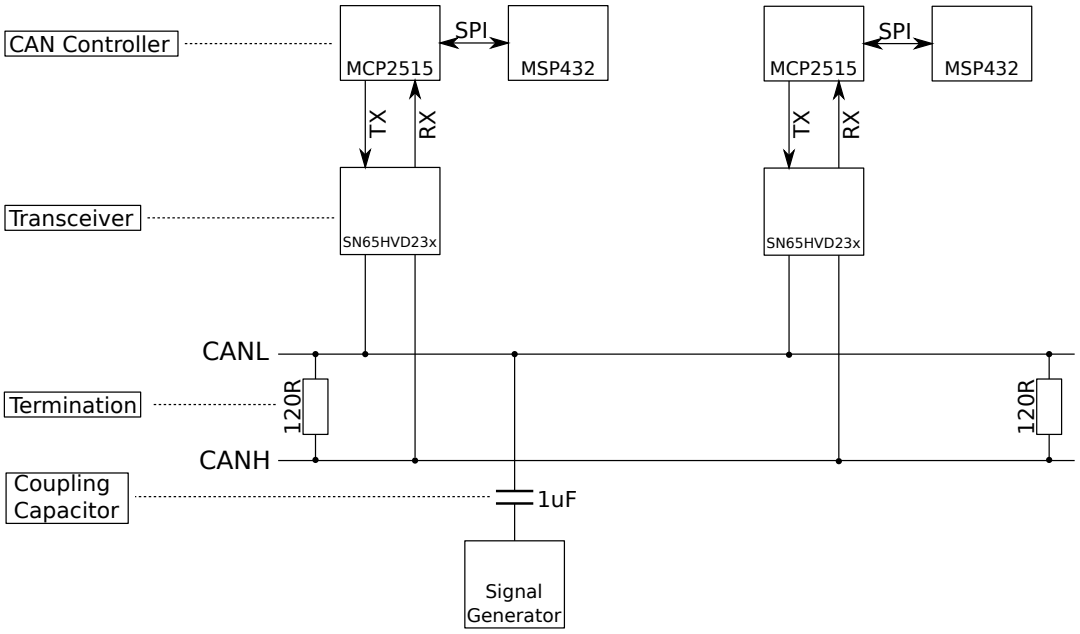


Figure 7.3: Simplified schematic of the CAN bus showing two nodes (with microcontroller, CAN controller and CAN transceiver), a connected signal generator and termination resistors

set to the maximum supported speed of 10 MHz, to reduce the effect the SPI connection has on the overall data rate of the bus.

The high level of fault and error detection included in the default physical and data link layers of CAN and the large amount of autonomy shown by a controller following the CAN standards means once a basic driver has been developed for the MCP2515, it is fairly straightforward to send messages. Furthermore, the built-in error detection omits the need for including additional CRC checksums to the messages.

The only complexity of the CAN bus not shared with the other analysed bus standards is the maximum size of 64 bits of the data segment within a frame. This limit means that any message larger than eight bytes has to be split up in multiple parts.

The functional breakdown of the CAN driver transmitting a single packet over either the TC bus or PL bus is included in Figure 7.4. An important thing to notice is the CAN frame containing the length of the total packet. This value is used on the receiving side to reconstruct the packet, as seen in Figure 7.5.

The transmit and receive errors in both flowcharts are retrieved from the MCP2515's buffer status bytes and error codes. If for example no subsystem responds to a certain system, even after several tries, then the transmission will time-out.

Each bus node has a filter configured within the CAN controller to only acknowledge and read frames directed to a single unique address via the message identifier. When such a packet is fully received by a bus node, then the received data is analysed in the same way as with the I^2C driver: if the length of the packet equals 250 bytes, then the packet is a data transfer. This data transfer is concluded by sending back an ACK. If the length of the packet is equal to two bytes, then it concerns a command from the OBC requesting data, which is passed on to the overhead test program. The reply is sent in a similar manner by the subsystem as described in Figure 7.4.

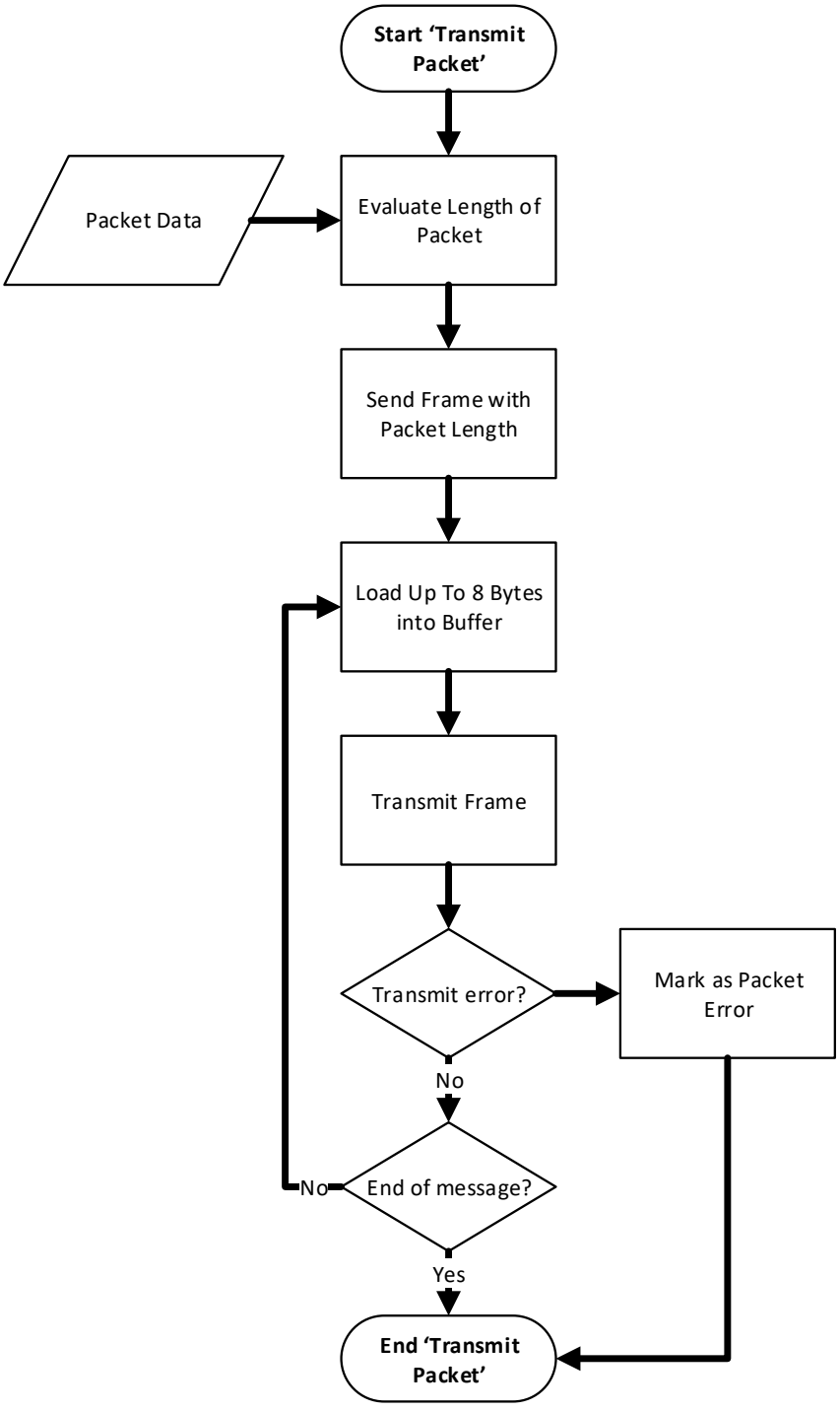


Figure 7.4: The functional breakdown of transmitting a packet over CAN

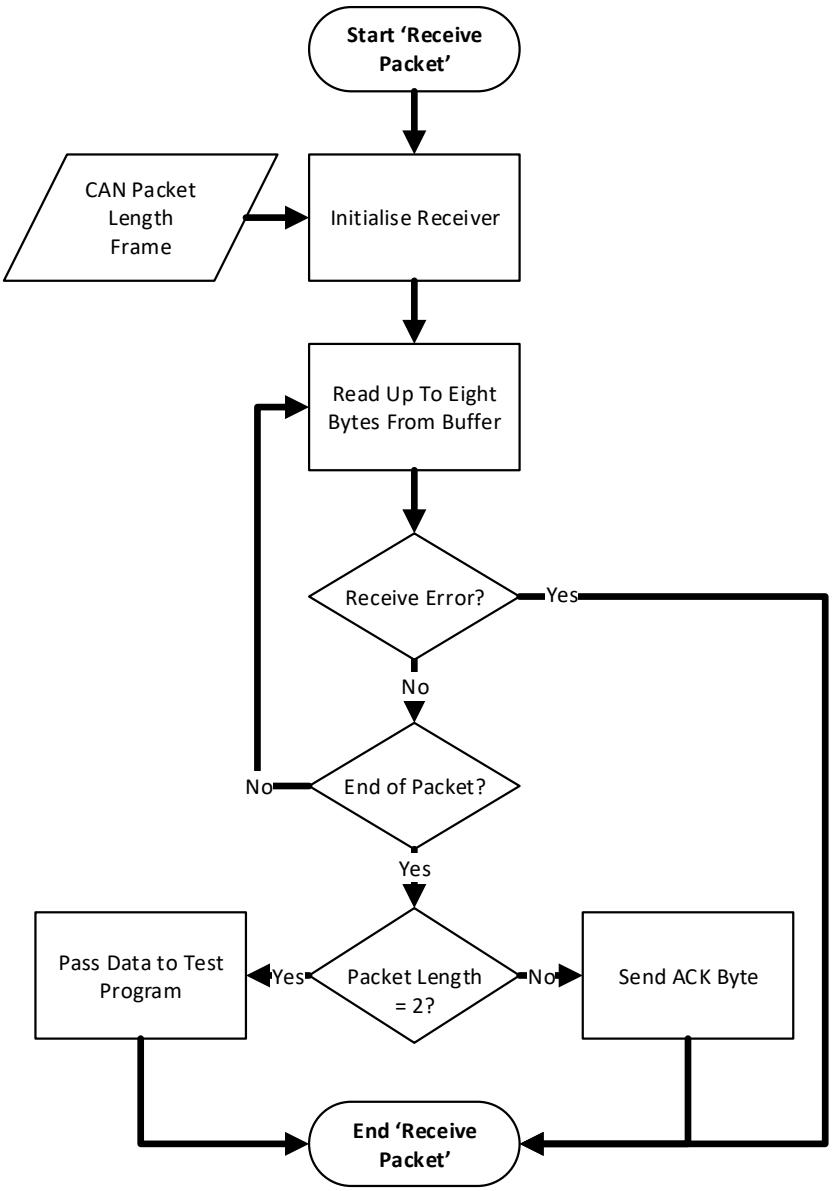


Figure 7.5: The functional breakdown of a subsystem receiving a packet over CAN

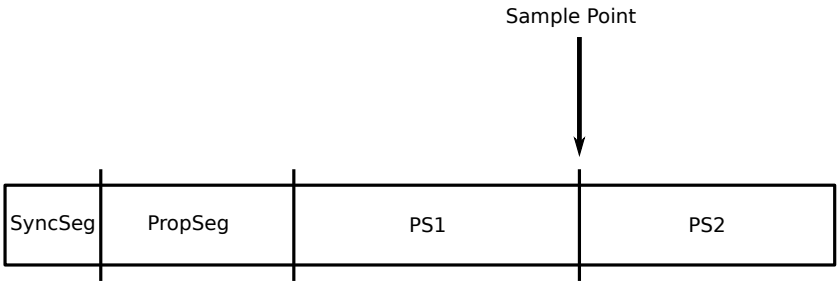


Figure 7.6: Timing of a single CAN bit (not to scale)

Most of the functional breakdown is very similar to that of I^2C.

7.5.1. Bit Timing

The bit timing of CAN is highly configurable to suit different wire lengths and microcontroller clock speeds. The basis of the configuration is the Time Quantum (TQ). For the MSP2515, the minimum length of the TQ is two ticks of the reference oscillator [29]. As may be seen in the electrical diagram included in Appendix A, a 20 MHz reference oscillator is used. Thus, the minimum TQ length (in time) is 100 ns. Still, this value is configurable through a simple clock divider.

A single CAN bit consists of several segments, each in turn consisting of a multiple of TQ:

1. The synchronisation segment (SyncSeg): used to allow time for synchronisation, fixed to one TQ in length
2. The propagation segment (PropSeg): used to account for the propagation of the signal over the bus. This is at least one TQ in length.
3. The first phase segment (PS1) and
4. The second phase segment (PS2) are used to set the location of the sample point. Both must be at least one TQ in length.

The sample point of the bit in between the two phase segments, as shown in Figure 7.6.

To achieve a certain bit rate (baud rate), the sum of the lengths of the different segments must equal the targeted bit time. The MCP2515 used in this thesis is configured to achieve a baud rate of 1 MHz, the maximum supported by the CAN standard. This baud rate implies a required bit time of 1 µs, equalling ten TQ in length. The exact segment configuration is then set to:

- SyncSeg: 1 TQ
- PropSeg: 2 TQ
- PS1: 3 TQ
- PS2: 4 TQ

Although there are many different configurations possible, this configuration is thought to provide enough margin on both sides of each bit to account for rise time of the signal.

7.6. Results

This section presents the results from the experiments performed using the CAN bus, both using the TC bus case in two, five and nine-node configurations, and the PL bus case in a two-node configuration.

7.6.1. Data Throughput

The data throughput was measured fivefold in each bus reference case to obtain average values. For the TC bus case, the average measured throughput is equal to 136.6 kbit/s $\pm$ 0.007 (3σ). For the PL bus, the average value is 158.8 kbit/s $\pm$ 0.015 (3σ).

These values are deemed extremely low: much lower than expected. Assuming the 1 MHz baud rate, the data efficiencies of the TC bus and the PL bus cases are only 13.7% and 15.9% respectively.

7.6.2. Packet Error Ratio and Noise Immunity

The 'cliff' behaviour as seen with the I^2C tests was again very apparent during the CAN white noise tests. No significant issues were detected until 0.5 V RMS, at which point more and more error frames were being transmitted until all nodes on the bus entered the *bus-off* fault isolation state. Because the test suite does not feature an actual EPS capable of resetting the microcontrollers, this state was unavoidable. Nevertheless, it does indicate that the (packet) error rate was of a sufficient level to make communication practically impossible.

For the tests using the transients, the bus was able to recover more readily, only entering a bus-off state at 10 V peak-to-peak, which is significantly more than the maximum allowed voltage of 4.3 V [29]. Nevertheless, at 2 V peak-to-peak, the PER already exceeded 5.3×10^{-3}, for both positive and negative transients.

7.6.3. Power Consumption

The power consumption of the CAN bus was determined for both the TC bus case and the PL bus case. Figure 7.7 shows the of the former. The maximum confidence in this figure equals 0.8 mW (3σ).

One may note that the idle power consumption and the power consumption of the case where the polling cycle is repeated once per second are extremely close together. The power consumption of the continuous polling test case does appear to be much higher, as one would expect.

For the PL bus case, no plots were made as there is only a single data point (also measured five times), namely with the two node case. The power consumption in idle state was measured to be 41.91 mW, and in the active state it was measured to be 91.5 mW. Both values correspond very well to the two-node values in the TC bus case, which is to be expected for an actively driven bus. It should not matter much whether one node is transmitting or another: the amount of effort distributed between the nodes is fairly equal, in contrast to the very master-centric I^2C.

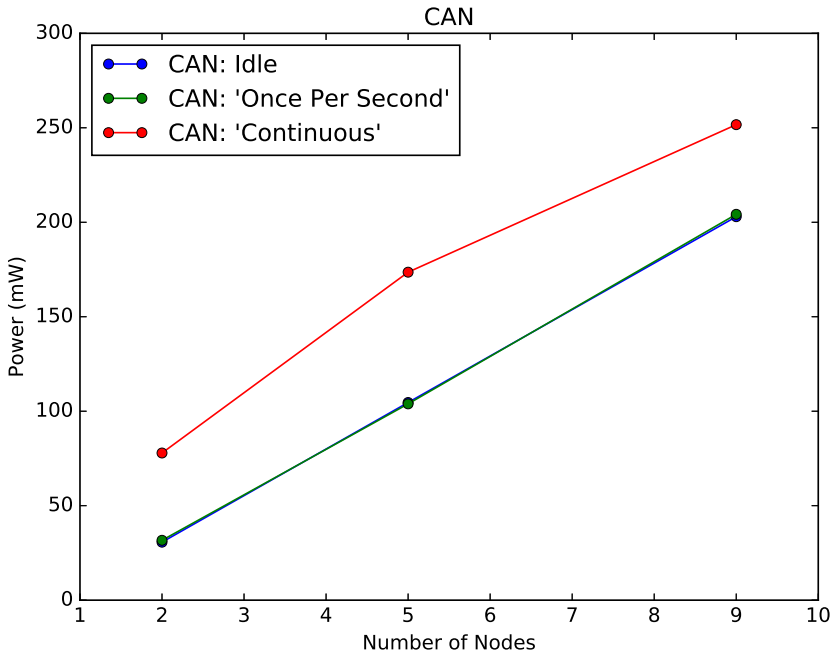


Figure 7.7: Power consumption measurements of the CAN TC bus. Note that the idle power consumption and the 'once-per-second' power consumption nearly completely overlap.

7.6.4. Complexity

To implement CAN, many microcontrollers include an internal CAN controller as an included peripheral. Because this is similar to how, for example, the eUSCI is included in the MSP432 to support I^2C and SPI, it is suspected that this sort of implementation is of a similar complexity as those on the MSP432. Especially as CAN contains a fairly extensive protocol, many standard tasks to perform on a bus, such as transferring or requesting data, are quite easy to perform.

Unfortunately, because the MSP432 does not include an internal CAN controller, an external controller must be used. The MCP2515 selected for this experiment is quite popular amongst developers, ensuring widely available programming examples and support. The extra (SPI) interface required to communicate to the CAN controller did prove to add a significant amount of effort to the development of the CAN driver: the extra layer made debugging and testing the software relatively difficult.

On the other hand, the bus hardware was relatively straightforward to implement, with only two wires and nine components to have a functional bus (including passive electronic components).

7.7. Conclusion

This chapter has looked at how the CAN bus was included in the experiments and how it performed. The lack of an integrated CAN controller in the MSP432 meant that an external controller had to be chosen: the MCP2515. This is in line with the target of evaluating buses which can be implemented universally.

The data throughput of CAN was found to be significantly lower than expected. Although the

large amount of overhead in the CAN's data link layer partially explains this discrepancy, it is impossible it is the only one. It is thought that a very large part of the lower throughput is due to the additional SPI interface between the MCP2515 and the MSP432 and the second layer of protocol overhead that it brings to a single CAN transmission. Furthermore, the additional delays caused by the extra physical layer also lower the overall data rates.

The tolerance to white noise is quite disappointing. It was expected that the differential signalling would cancel out the majority of the noise, but this does not appear to be the case. This is probably due to the excessive voltage levels generated by the signal generator: at 0.5 V RMS, the peak voltages already approached 10 V, probably causing low-level hardware issues due to the voltage spikes rather than real interference to the signal. It is possible to have the SN65HVD23 include a rise time regulator, although this was not included on the actual component used in the test. It is though that these excessive spikes can be clamped down when one is used.

A major obstacle during the tests with CAN is that the CAN controllers quickly entered bus-off states, following the CAN standard. Although the CAN bus is fault tolerant in design, the incidents of the bus-off state must be handled quickly and effectively to ensure correct operation of the bus. As the handling of this is very limited within the data bus test suite, the problems occurred. It is important to incorporate these fault handling mechanisms into the software design from an early stage of the development.

8

RS422 / RS485

This chapter will look at the Recommended Standard 485 (RS485), a basic bus standard based on a microcontroller's UART. First a description of the physical layer is given, together with a definition of the protocol used by the different bus nodes. As in the previous cases, the chapter will end with the results obtained from the practical tests with RS485.

8.1. Introduction

RS485 is somewhat of a different case than the other bus standards treated within this thesis. First, it is an *asynchronous* serial standard instead of a *synchronous* standard. Instead of including clock and timing information, either in the signal or as a separate signal, all bus nodes must be synchronised beforehand. Nodes can fine tune the synchronisation through detecting bit changes, although there is no guaranteed level shifting through bit stuffing as is the case with CAN. Secondly, RS485 only defines a physical layer, no data link layer. Thus, how data is transferred over the bus is completely up to the bus designer.

RS485 and its 'sister standard' Recommended Standard 422 (RS422) are the logical successors to Recommended Standard 232 (RS232), the serial standard often used to communicate between computers and embedded equipment [26]. However, it supports significantly higher baud rates (the default is defined to be 1 MHz, but is often applied at higher levels [26]). RS422 is To enable the transfer of information over longer distances and in more aggressive (industrial) environments, RS422 and RS485 describe the use of differential signalling (for a brief introduction, see section 6.3). The only difference between RS422 and RS485 are that the latter is designed with implementation in linear bus topologies in mind. However, as the RS485 is also tested in the PL bus case in a point-to-point topology, it is technically RS422. Yet, pedantics aside, this chapter will refer to both as RS485 for clarity and since it is the more universal case.

The physical layer is fairly simple, with only a single driver required for each individual bus node. The design of an RS485 driver allows for half duplex communication. Hence, the driver must feature a dedicated pin to switch the direction of communication. This pin is not present in RS422, thus practically resulting in it being a simplex communication format. Although the standard does not otherwise define the source of the signal [26], the internal UART included in most, if not all microcontroller is used.

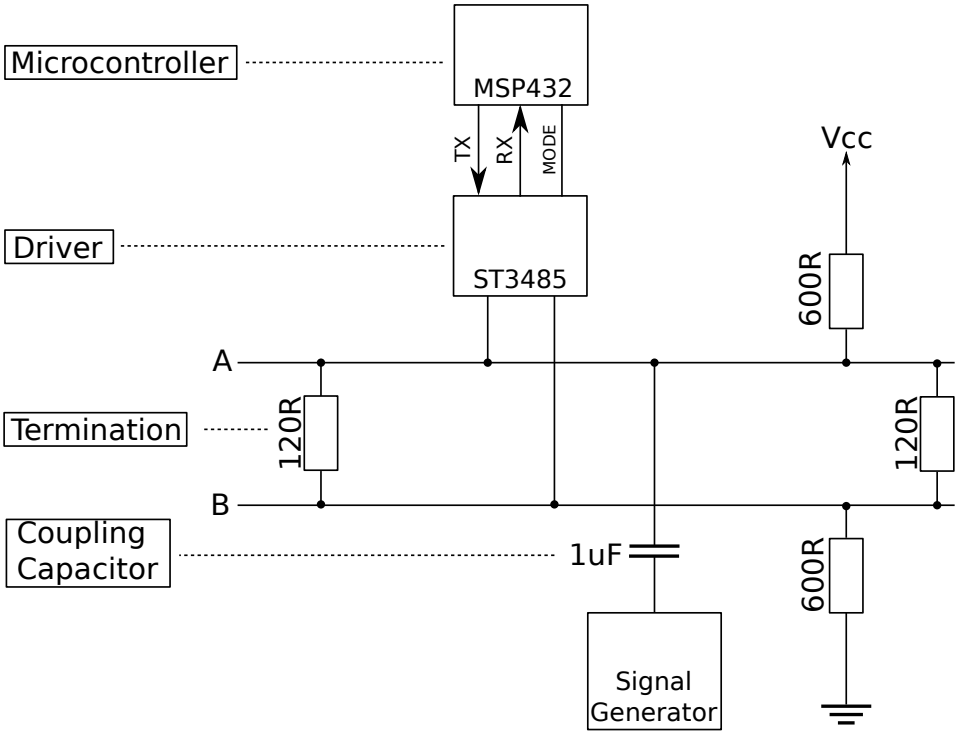


Figure 8.1: Schematic of the RS485 test setup including the noise generator (GEN)

The maximum number of nodes connected to a standard RS485 bus is at least 32 [58]. Nevertheless, by carefully selecting the RS485 drivers according to their specific *Unit Load*, which is defined as the equivalent resistance of a single node normalised to that of 32 nodes, the number of bus nodes can be increased to at least 256 [58].

The details given in this section are more or less the full definition of the RS485 standard. Therefore, following the hardware selection in the next section, a simple protocol will be presented fulfilling basic communication.

8.2. Physical Layer Design

The popularity of RS485 in industrial applications means that there are a lot of different driving components available. For this test, a fairly generic low-power driver is chosen: the ST3485 [59].

Figure 8.1 shows the bus architecture of the RS485 bus used in the experiments. Included are pullup/pulldown resistors to keep the bus lines in a defined state when the bus is idle, similar to those seen for dI^{2}C. Furthermore, two 120 Ω termination resistors are included.

8.2.1. Analysis of the Bus Termination

As mentioned in section 6.3, termination of a differential bus is required to remove signals from reflecting on the bus. However, on a short transmission line, the probability of these reflections influencing the signal are small. One of the main reasons is that since the signal travels at approximately the speed of light, the signal delay is roughly 3.3 ns/m assuming the speed of light to equal 3^8 m/s. As a 1 MHz signal has a 1 µs bit time, the signal's travel time over the

bus is two orders of magnitude smaller. Therefore, even if reflection would occur, they would hardly interfere with the main signal as long as the line length stays in the order of a couple of meters or less.

To investigate the effect of removing the bus termination, the RS485 experiment is also performed on the same bus without the resistors. To see that removing the termination resistors, an analysis of the circuits is required when they are either active or idle (the two states with differential signalling).

The schematic drawing of the case where the bus is idle is shown in Figure 8.2. The pullup and pulldown resistors make sure the transmission lines stay in a deterministic state. As there are two 120 Ω termination resistors, the equivalent resistance between the two transmission lines equals 60 Ω. Furthermore, because all bus drivers are in a high impedance state following the RS485 standard [26], the A and B lines effectively do not affect the overall circuit. The equivalent circuit that remains is given on the right of the figure, with three resistors in series. Assuming the default 3.3 V, the overall power consumed by the bus in the idle case equals approximately 9 mW. The voltages of the main bus lines A and B can be shown to be 1.73 V and 1.57 V respectively.

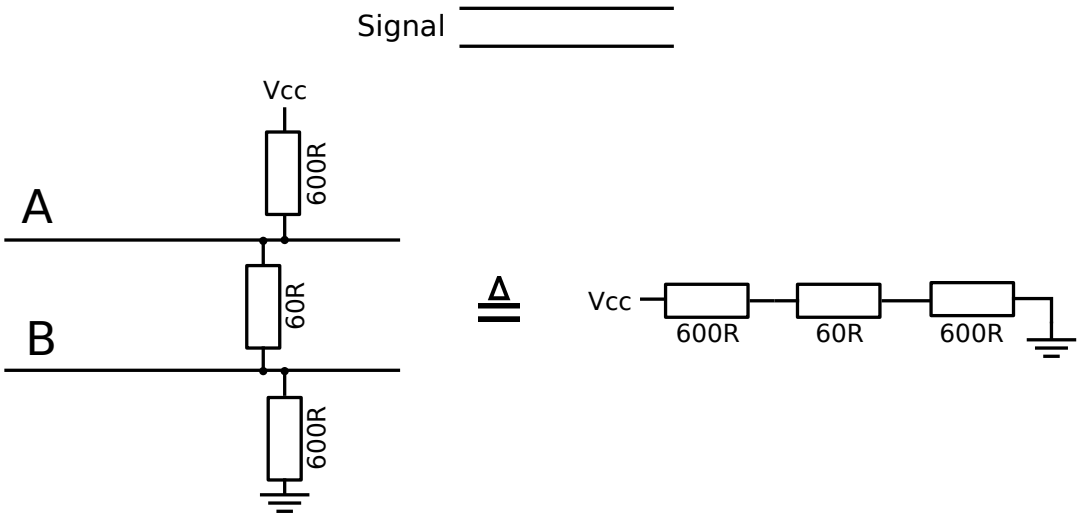


Figure 8.2: A schematic of the idle RS485 circuit: transmitting a logic low

When the bus is in an active state (transmitting a logic high), the case as in Figure 8.3 holds. Because the positively biased A line is now driven to the source voltage Vcc and the negatively biased B line pulled to ground, the two 600 Ω do not experience a voltage change across the resistor anymore. Therefore, the only resistor still consuming power is the effective termination resistance of 60 Ω. In this case, the bus power consumption equals 182 mW.

Assuming an equal distribution between the high and low states in the data stream, the mean power consumption would be the average of the two power figures: 96 mW.

As seen in both the active and idle cases, when the bus termination is removed, the effective resistance between the A and B lines becomes infinite, reducing the bus' power consumption to zero. Because removing the termination will make the bus more sensitive to noise altogether, a middle ground must be found for the value of the termination, which is application specific.

The oscilloscope captures in Figure 8.4 show the differences between terminated bus lines and unterminated bus lines in the actual RS485 test setup. In Figure 8.4a, the bus lines are

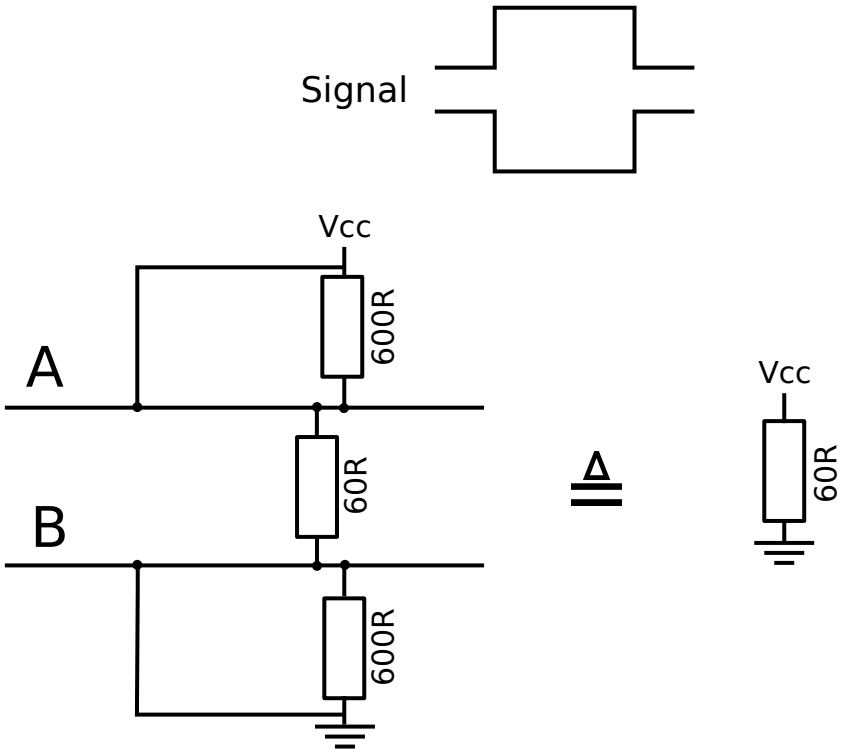


Figure 8.3: A schematic of the RS485 circuit in active mode: transmitting a logic high

terminated. The idle state of the bus lines (roughly the first half of the plot's x-axis) shows the signals at their computed voltages (1.73 V and 1.57 V) before starting the actual transmission, in this case the packet preamble byte. In Figure 8.4b, the idle state shows the A and B lines pulled to Vcc or ground respectively. This is because the lack of termination resistor means no path over the bus lines exists between Vcc and ground.

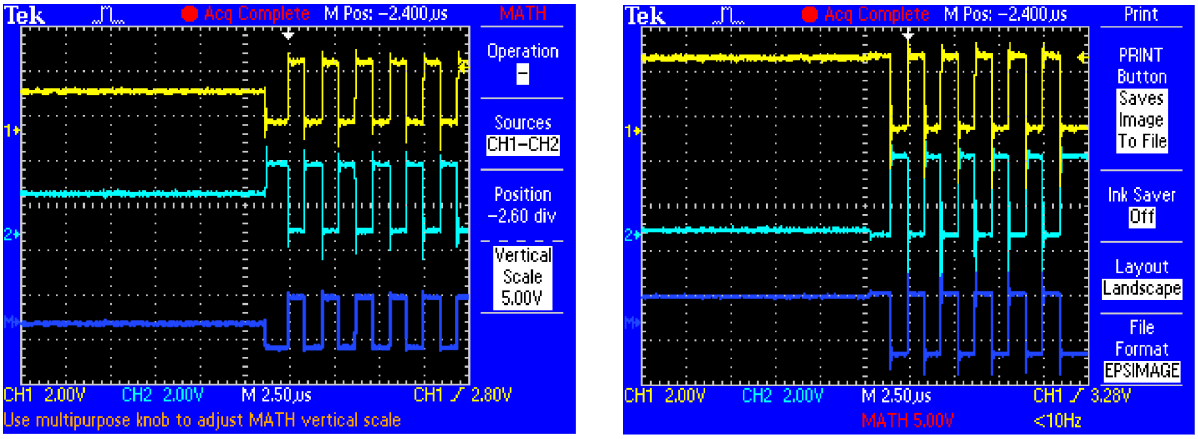
Because the idle state of the unterminated differential bus lines results in a logical high signal, this might be problematic in some bus configurations as it could be read as part of a message. Fortunately in the case of RS485, the selection of the start and stop bits (see section 8.3) are such that this is not an issue.

One may observe that the signal seems to overshoot the target voltage in both Figure 8.4a and Figure 8.4b. This is actually the reflection of the signal returning over the bus lines. Furthermore, the reflection peaks are significantly larger on the unterminated bus lines than the termination ones, whereas the in the former the reflections even show up in the resultant signal. Although these reflections are not found to be problematic, it shows the measurable effect the terminations has, even at these relatively short bus lines.

8.3. Data Link Layer

As mentioned in the introduction to RS485, there is no data link layer defined for RS485. Thus, one must be designed.

First and foremost, the function of the UART must be clear. This component, part of the eUSCI in the MSP432, is similar to a shift register, simply taking a parallel bus signal and serialising it. The only major addition added by the UART are the start and stop bits: a 1 and



(a) With bus termination

(b) No bus termination

Figure 8.4: Oscilloscope captures of an RS485 bus with and without termination. Both images show the two main bus lines (top two) and the resulting computed differential signal (bottom). Note that the 'decoded' signal is actually the inverse of the signal as interpreted by the UART. The data transmitted is clearly identified as the defined packet preamble ($0101\,0101_{\text{bin}}$).

a 0 respectively [60]. Thus, to transfer a single byte, ten bits are required.

To transfer a data set, some additional (custom written) message overhead is required to be defined for the RS485 data frames to arrive correctly at their correct location. The frame format is shown in Table 8.1, with the basic functional breakdown to transmit a frame included in Figure 8.5.

The preamble is used to signal a new frame. Because this byte can be set to any arbitrary value, the value $0101\,0101_{\text{bin}}$ (55 in hexadecimal) is used. The next byte indicates the size of the frame. As its maximum value equals 255, the maximum frame size is also 255. Of course, this value can be increased if necessary. Fortunately, the longest packet to be transmitted during the test (chapter 4) is 250 bytes in length, hence it can be contained in a frame with a single 'size' byte. The size byte is followed by the address byte indicating the recipient of the frame, in turn followed by the data bytes. The last two bytes of the message are a CRC-16 checksum to be verified by the recipient. If everything goes well on the recipient side, including a correct CRC checksum, then the recipient ACKs the message by responding with a single byte containing its address.

One must note that the ordering of these parts of the frame is completely arbitrary, as RS485's design causes each node on the bus to receive each frame, therefore shifting the decoding of the message to the node's software.

In the PL bus version of RS485, the address and size are omitted, as these are both set to fixed values due to its point-to-point architecture.

As can be seen in Figure 8.5, the transmitter more or less blindly transmits the generated

Table 8.1: The RS485 data frame. Note that the start and stop bits, inherent to the UART, are omitted in the bit count.

Function	Preamble	Size	Address	Data	CRC	ACK
No. of Bits	8	8	8	0 - 2040	16	8

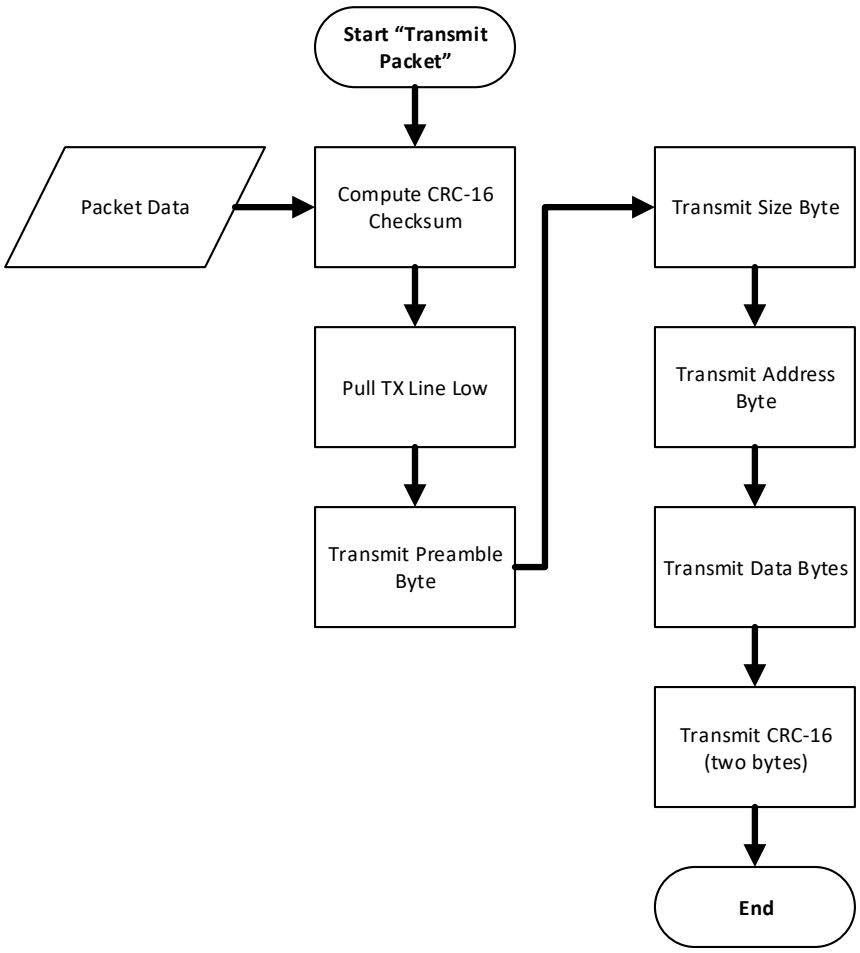


Figure 8.5: Functional breakdown of the RS485 driver transmitting a packet

frame and waits for the ACK. As the UART in the MSP432 passes on the data byte-wise to the microcontroller, more logic is required on the recipient side to reconstruct the original data frame. This logic is shown in Figure 8.6. When the driver receives the first byte of a frame (the preamble), it first verifies whether it has received an actual preamble byte. If this is the case, a boolean variable is set to notify the next instance that a frame has started. It then continues into the frame, first receiving the size of the frame, then the number of data bytes. Using this information, the original data set can be reconstructed. This dataset's CRC-16 checksum is then verified to the one which it has received. After a full frame has been received, the boolean variable indicating 'preamble received' is cleared.

This simple architecture including the preamble solves several failure modes. In case a recipient node misses the preamble, it will ignore the entire frame, which can be retransmitted after the transmitter does not receive an ACK. Even if the data set contains the preamble byte in the rest of the frame somewhere (like the pre-generated dataset does), it will receive gibberish, failing the CRC check and not sending the ACK. At this point, in a real environment, the message would be retransmitted by the transmitting node.

8.3.1. Use of an Address Bit

The driver architecture presented here possesses one elemental flaw: each separate byte has to be read by the main program through the use of interrupt routines. Hence, every message is effectively transmitted to every node on the bus, where only the intended recipient of the message responds to the transmitter. Especially when the bus has a large amount of traffic, it can interfere with the operations of all the nodes on the bus, which have to keep up with all the messages.

Although this was not direct a problem in the testing quite used in this thesis project, a solution is needed to prevent serious problems from occurring within actual spacecraft. One common way is the addition of an extra 'address' bit accompanying each byte output by the UART. When the byte in question contains the address, the address bit is set to 1. Otherwise, the value is always 0. This makes it possible to have a separate interrupt trigger in every node when an address byte is transmitted. The node for which the frame is intended can then enable the mechanism to receive all following bytes, while other nodes simply keep it off. This also omits the need for a preamble byte, possibly adding extra robustness to the protocol.

Unfortunately, this was not implemented in the test suite, but it is deemed absolutely necessary in any real implementation of the bus.

8.4. Results

This section will look into the results obtained from the experiment using RS485 as a TC bus (with two, five or nine nodes), both with and without termination resistors, and in the PL bus test case.

8.4.1. Data Throughput

For the TC bus case, the measured data throughput was 603.6 kbit/s $\pm$ 0.016 (3σ), resulting in a 60.3% efficiency with the configured (standard) 1 MHz baud rate. This value is the same for the bus without termination resistors.

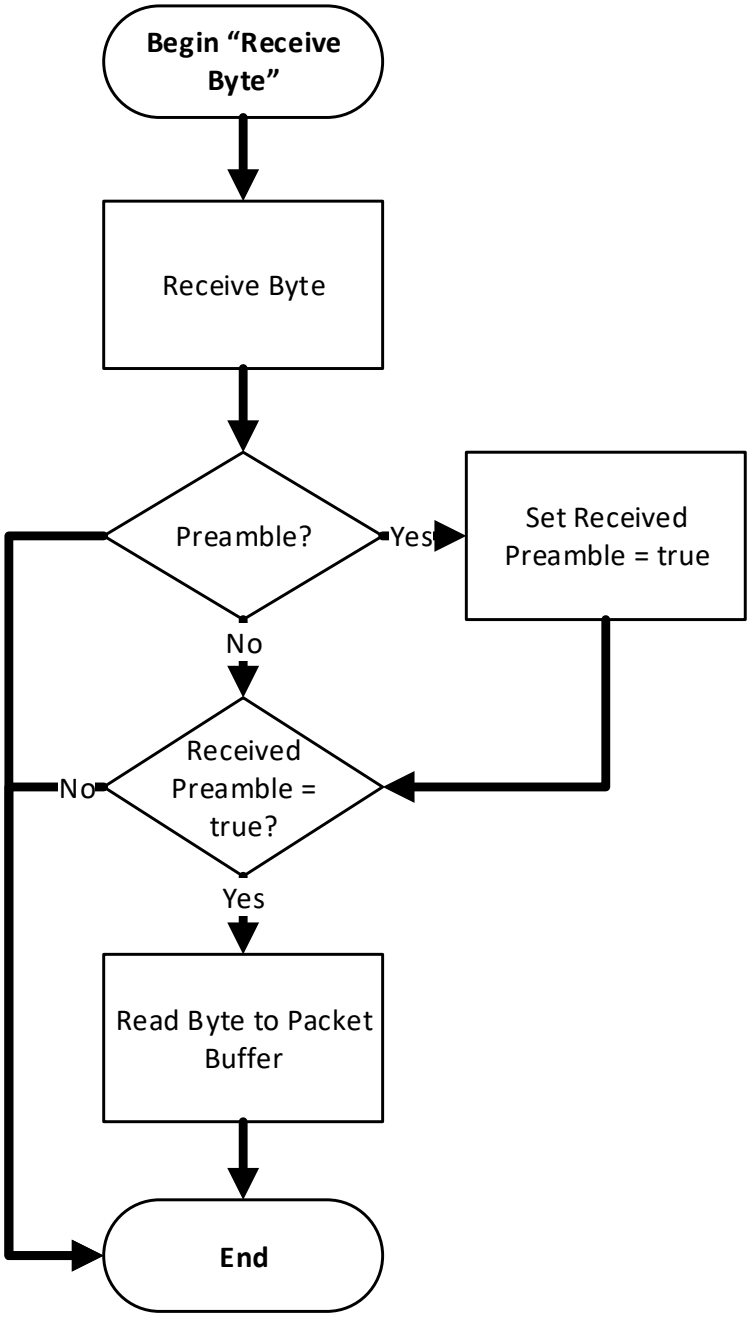


Figure 8.6: Functional breakdown of the RS485 driver receiving a single byte

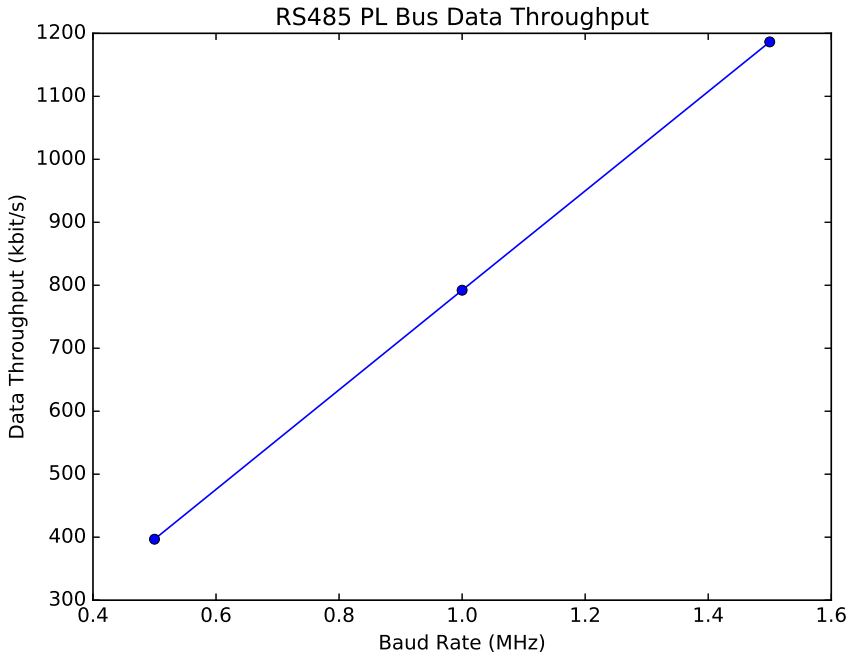


Figure 8.7: Plot of the RS485 effective data throughput versus baud rate

For the PL bus case, an analysis was performed by varying the baud rate to see the resultant behaviour of the throughput. It was tested up to a baud rate of 1.5 MHz, where timing problems in the software caused the test to become unreliable. These problems are however not linked to the data bus itself. The resulting relationship, plotted in Figure 8.7, appears to be perfectly linear between the three measured data points. The highest 3σ value equals 0.3017 kbit/s. The value at where the baud rate equals the standard value of 1 MHz, the effective throughput is found to be 792.2 kbit/s $\pm$ 0.1095 (3σ), considerably higher than the TC bus test. This is most likely not due the lower amount of overhead in the frames, as its size is negligible compared to the large data frames. Rather, it is probably due to the lower amount of logic on the recipient (OBC) side to receive a full data frame.

8.4.2. Packet Error Rate and Noise Immunity

In contrast to the CAN bus setup and dI^{2}C driver, the chosen RS485 driver does feature a rise time regulator. This was clear in the results of both the white noise testing and the transients.

For the white noise testing of the RS485 bus with termination resistors, no packet errors were detected until 0.8 V RMS, where the measured PER equalled 9.8×10^{-2} based on 20 530 measured packets. Figure 8.8 shows an example of the injected noise on an oscilloscope capture and the ability of the differential lines to filter out the coupled noise. It is also apparent that the effects of noise are much less once a signal is driven over the bus. This is probably due to the significant change in overall bus impedance when an RS485 driver leaves the high-impedance state to be able to transmit.

For the injected transients, the first point where the PER exceeds the maximum allowed value of 10^{-4} is at 8 V peak-to-peak, grossly exceeding realistic voltage levels. Here, a PER of 7.2×10^{-2} is measured, based on 25 380 packets.

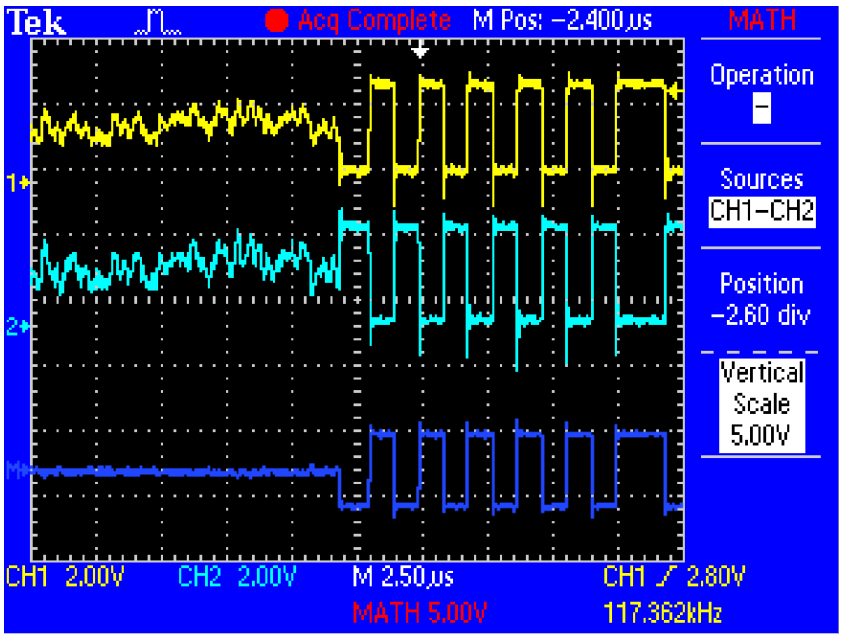


Figure 8.8: Example of the induced white noise into the RS485 bus lines (with termination) (top two curves). Note the nearly complete disappearance of the noise signal in the resultant signal (bottom)

Interestingly, the RS485 bus without termination resistors performed significantly better than the one with resistors, reaching 1.0 V RMS with the white noise testing until the PER reached 8.06×10^{-4} with 59 490 packets. For the transients, it managed to withstand 10 V peak-to-peak until it reached a PER of 2.6×10^{-2} at 20 V peak-to-peak.

8.4.3. Power Consumption

The power consumption of the RS485 in the TC bus case is shown in Figure 8.9. The maximum measured 3σ value is 0.45 mW.

First of all, it must be noted that the measurements of the 'once per second' test cases appear lower than the idle state of the bus. As the duty cycle of the former is still extremely low compared to a continuously occupied bus, it would be expected that the curves would more or less follow the same line. It is thought that this is due to inefficiencies in the software program of the test suite: as the power consumption of the RS485 bus components (including the drivers) is very low, this inconsistency has become apparent.

The measurements shown in the figure correspond fairly well with the simple model discussed in subsection 8.2.1: the difference between the measurements of the idle states of both with and without termination is approximately 9 mW, as predicted. The difference between the measurements of the continuous states is larger: about 120 mW versus the predicted 96 mW. This is most likely due to the increased activity of the rise time regulators in the bus drivers, which have to drive larger currents onto the bus to achieve consistently stable rise times. Moreover, the additional activity of the microcontrollers themselves adds to the power consumption.

For the PL bus, the measured values for the power consumption for different baud rates are shown in Figure 8.10. Note that despite trebling of the baud rate, the power consumption stay relatively flat. This is to be expected, because as found in subsection 8.2.1, the mean power consumption of a bus continuously switching will not vary with different baud rates.

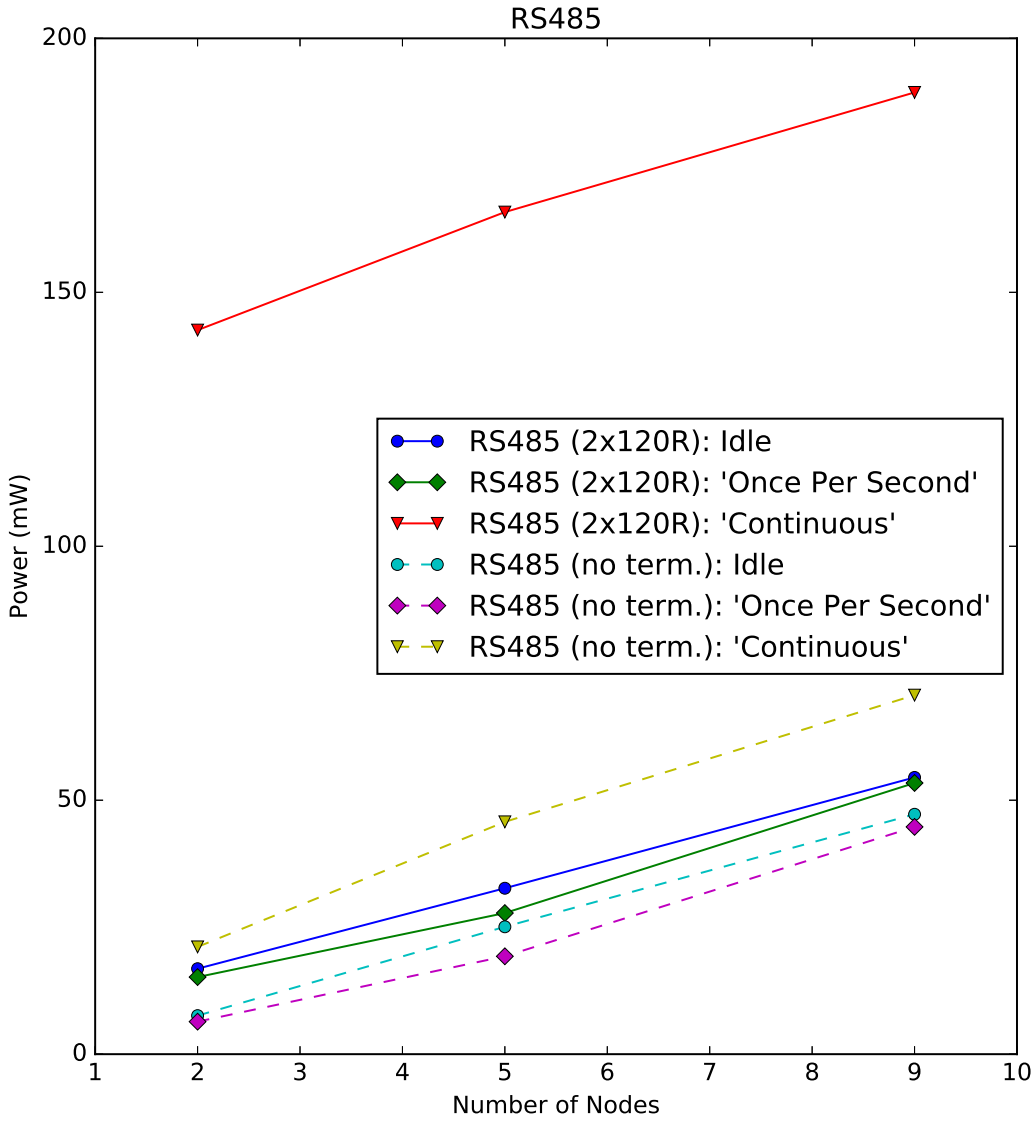


Figure 8.9: Measured values of the power consumption of RS485 in the TC bus

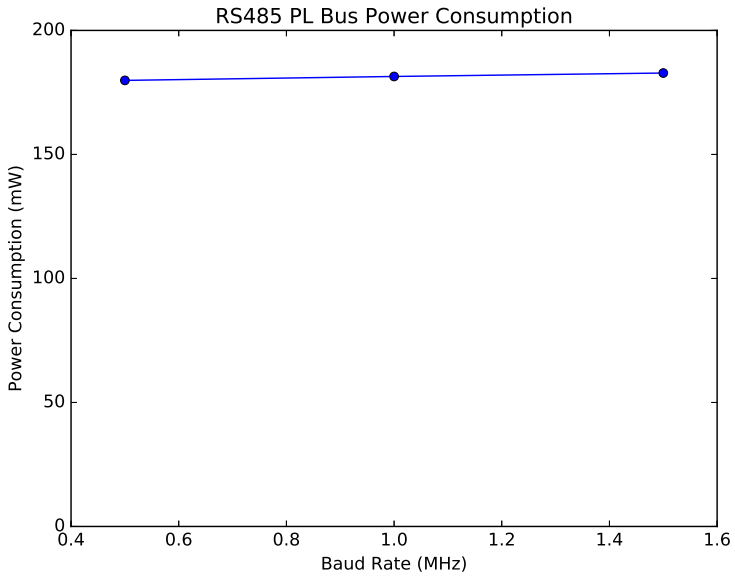


Figure 8.10: Power measurements of the RS485 PL bus as a function of baud rate

Furthermore, despite the increasing data rate, the bus occupancy remains the same (79% as measured with the data throughput). Thus, the mean power consumption should remain stable.

8.4.4. Complexity

The RS485 bus was perhaps one of the least complex buses to implement, despite the increased effort in designing a simple protocol. However, it is suspected that once subsystems acting as bus nodes begin to feature more time-critical functions and external interrupts, problems will start to occur. Overcoming these issues might be very difficult.

The standard placement of a UART in virtually all microcontrollers and the simple design of the bus' physical layer results in a very straightforward implementation of the bus in any subsystem.

8.5. Conclusion

The implementation of RS485 has proved to be very simple and durable. The fact that the standard does not define a protocol means that the developer has a lot of freedom in his or her design. However, this is a double edged blade: on one hand, it makes it possible to customise the data link layer to a high degree, omitting any unnecessary message overhead. On the other hand, the messages can become too simple, possibly decreasing the robustness of the standard dramatically. Moreover, lacking a protocol means that even though the standard is easy to implement for many commercial parties, complete incompatibility between systems is still a real possibility.

The fact that most of the protocol must be handled in the main software program might cause problems in systems once real-time applications and dependencies are added. Although adding an extra address bit reduces this problem, it still requires every subsystem to tem-

porarily interrupt what it's doing to handle the single address byte for each message. This address bit also adds one-tenth the overhead to the protocol, which could especially have significant effects to systems transmitting large data sets. Another way to handle these timing effects more gracefully would be through the use of an Real-Time Operating System (RTOS), although this does not necessarily increase the performance of a system, while it does increase the complexity of a system.

Removing the termination resistors from the bus proved to radically decrease the power consumption of the overall bus, while also increasing the robustness of the system to the injected noise and induced transients. This unexpected result regarding the latter is possibly explained by the test setup and the way the noise is induced into the bus. Because the two differential lines are not connected in any way in the unterminated case, the two wires have independent behaviour (disregarding capacitance and inductance effects). Furthermore, as the 33210A signal generator has a 50 Ω load impedance [42], the wire connected to the signal generator will experience a higher line impedance. This reduces the amplitude of the signal and the injected noise compared to the second, unaltered line. Because the unaffected signal has thus a larger voltage swing, the signal is stronger on this side and will simply overpower the negative effects of the noise on the other line. In the terminated bus case, the noise and load impedance affects both lines through the termination resistors, reducing the overall (differential) signal strength. Thus, no conclusions can be made regarding the noise immunity using this particular experiment.

One must note however that in realistic scenarios, where the noise over the bus lines is thought to be highly coupled, the use of termination resistors is still necessary. Furthermore, the relatively short bus lines mean that the effective resistance could be increased to several hundred or even thousands of ohms, although the exact value needs to be determined in each application.

Although the ST3485 RS485 driver used in this test was very suitable for the job and proved to be low in its power consumption, the power consumption can be even more optimised by selecting a driver with a more suitable rise time regulator. There are multiple manufacturers on the market which carry drivers optimised for baud rates from several 10s of kbit/s a couple of Mbit/s. Choosing the correct and optimised driver for the application can reduce the overall power consumption even further. Still, it is recommended a standard (set of) baud rate(s) to be defined to enable cross compatibility between third party systems and components.

9

Serial Peripheral Interface (SPI)

The Serial Peripheral Interface (SPI) bus is discussed in this chapter. As it was only tested in the PL bus case, the amount of results might be considered to be limited compared to the preceding chapters. Nevertheless, this chapter will introduce the bus standard, describe the physical and data link layers and present the measurements.

9.1. Introduction

The SPI interface was briefly discussed in chapter 7, regarding the interface used to communicate to the MCP2515 CAN controller.

SPI was introduced with the Motorola 68xx series of microcontrollers in the early 1980s [28], and was defined to be used as the main bus to connect to external peripherals. No formal standard exists for the bus, hence the datasheets of microcontrollers are usually the 'official' reference for developers.

As is the case with RS485, no data link layer is defined. Therefore, this must be defined separately again.

Note that the noise and transients will not be tested for the SPI, as will be explained in chapter 11.

9.2. Physical Layer

The basic physical layer is shown in Figure 9.1. A unique feature of SPI is the Chip Select (CS) line. The CS functions as the addressing system: the bus master simply activates the line (active low in this case) to signal the corresponding subsystem that it must receive or transmit data. This architecture makes the bus master-centric, similar to I^2C.

The need for the CS is also the reason why SPI was not considered for use in a TC bus configuration. As the OBC would require at least eight individual CS lines to be able to talk to all subsystems. In the extreme case where the TC bus would be extended into a multi-master configuration in the future, the number of chip select lines N_{lines} would increase following

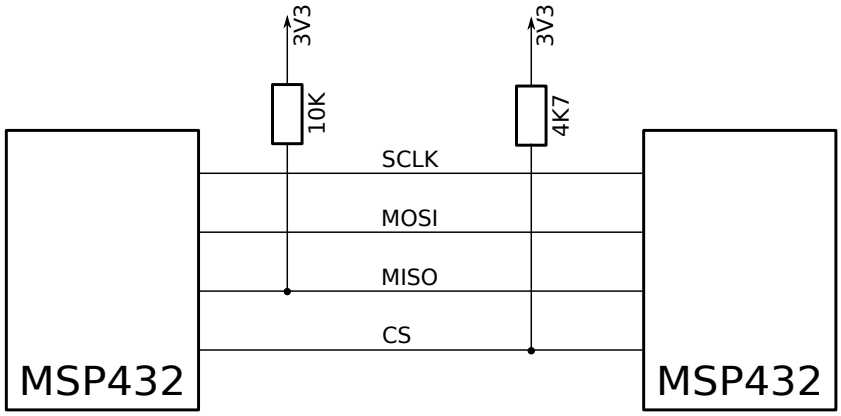


Figure 9.1: A schematic of the SPI architecture as used for the PL bus test

Equation 9.1:

$$N_{lines} = \sum_{n=1}^{N-2} n \tag{9.1}$$

Where N is the number of nodes on a bus. For the case $N = 9$, $N_{lines} = 28$.

The MSP432 contains a built-in SPI peripheral as part of the eUSCI, thus no external drivers or other ICs are required to operate the bus.

The two main data lines, Master-Out-Slave-In (MOSI) and Master-In-Slave-Out (MISO), carry the data in the direction stated in their names. The use of the two lines makes SPI the only full-duplex bus analysed within this thesis project. Both lines are active high, non-differential data lines. Data driven out on these lines is synchronised to the clock signal on the Clock (SCLK) line, making SPI a fully *synchronous* standard.

Two pullup resistors are necessary: one for the CS line, and a secondary resistor pulling up the MISO line. To understand why a pullup is required on the MISO line and not on the MOSI line, one must understand how the lines are driven in SPI. The input/outputs of the nodes connected to the SCLK, MOSI and MISO lines can take on three states: high, low and high-impedance [16]. Because the bus master controls the first two, it can simply keep the lines in a defined state when idle. However, because the MISO line is an input for the master, it can not control its state. When no other bus node is active (idle bus), the MISO will be left floating, as all nodes keep their MISO input at high-impedance. Thus, a weak pullup is required to avoid this situation, which can be overpowered when a node starts communicating.

No default baud rate is defined for SPI, although it has been shown that it may function at speeds over 10 MHz [28].

9.3. Data Link Layer

SPI modules are designed to be triggered by either the rising or falling edge of the clock ticks on SCLK. Which one is used is entirely up to the developer of the system, although it is important to have each bus node configured the same way. Often, components or peripherals supporting SPI are preconfigured in one mode, limiting the SPI mode selection to just one option. Whichever option is chosen, it should not influence the performance of the bus.

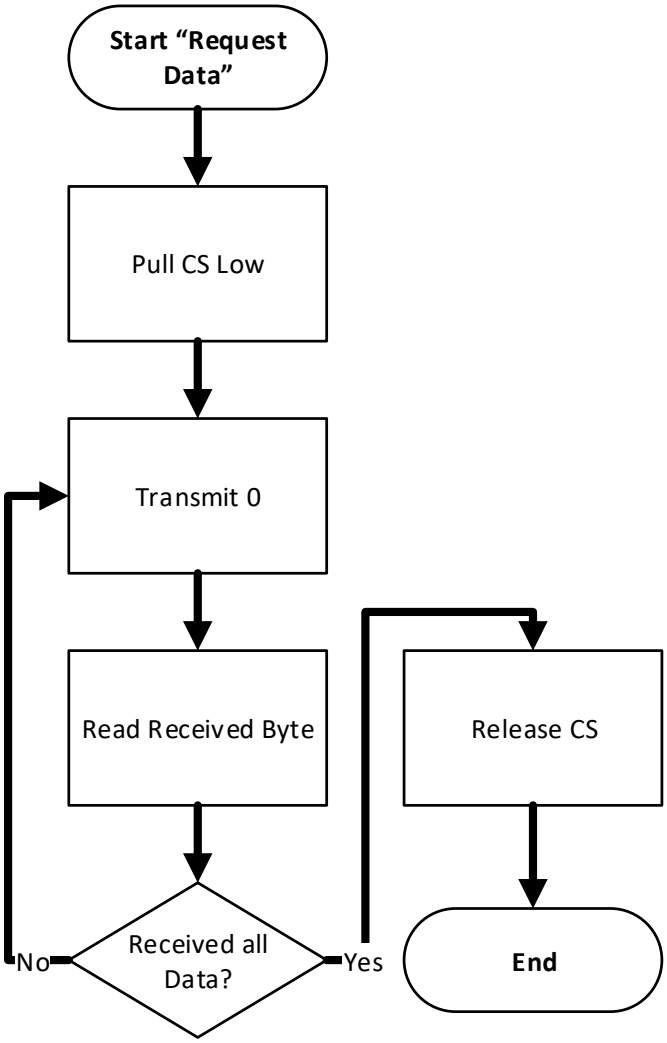


Figure 9.2: Functional breakdown of the OBC requesting data from the payload

The master-centric architecture results in most of the work being done by the master node, in this case the OBC. Nevertheless, the functionality is very straightforward, as seen in Figure 9.2.

The OBC, which acts as bus master, starts by pulling the CS low. This signals the slave (the payload) to prepare the data for transmission. The OBC transmits a zero, keeping the MOSI line low, but providing eight clock ticks on the SCLK line. A the same time, the payload drives out the actual data synchronous to the clock ticks over the MISO line. This continues until the OBC has received all the data (or has filled up some kind of buffer), and releases the CS line.

The corresponding functions performed by the payload (slave) may be seen in Figure 9.3. The order is extremely simple: if the CS is pulled low and the TX buffer is available, the next byte is loaded into the buffer. The logic is all based on interrupts.

As the size of the data set is assumed to be known beforehand on the PL bus, and only a single

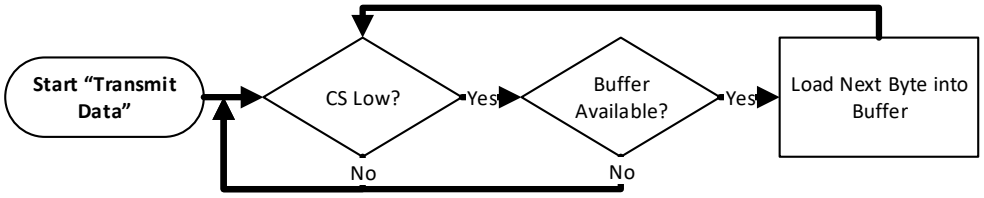


Figure 9.3: Functional breakdown of the OBC requesting data from the payload

node is connected to the master, no additional message protocol is required to accompany the payload data. The single node also means the bus could run without the CS, although in this case it was assumed to be necessary to prepare the transaction on the payload node's side. As the node needs to provide payload data immediately once the OBC start transmitting itself, this small extra margin provided time to fill the TX buffer. Nevertheless, the 4.7 kΩ pullup resistor means it only uses 2.3 mW when pulled low, based on 3.3 V.

9.4. Results

As mentioned previously, the SPI bus has only been tested in the PL bus configuration. This section will go through the results.

9.4.1. Data Throughput

The measured throughput values are shown in Figure 9.4, plotted versus varying baud rates. The maximum 3σ error was 0.090 kbit/s. A similar straight line is apparent as seen with RS485. However, in this case the slope decreases slightly at higher baud rates. This is most likely due to the fact that the delays in between bytes do not reduce with increasing baud rates. Hence, even as the speed of bytes being clocked out is increasing linearly, the overall data throughput is not.

The baud rate in Figure 9.4 is tested up to and including 2.0 MHz. Above this value the bus started dropping a significant amount of packets, which is possibly due to timing problems similar to the problems encountered with RS485. Nevertheless, the figure gives a good indication of the behaviour of SPI, and more dedicated software development of the SPI drivers will ensure higher achievable baud rates.

9.4.2. Bit Error Rate

As mentioned in the introduction and to be discussed in chapter 11, no noise tests were performed. However, the bit error rate was confirmed to be lower (better) than 10^{-6} with 95% confidence using the method as described in section 4.2 transmitting 3×10^6 bits without any detected bit errors.

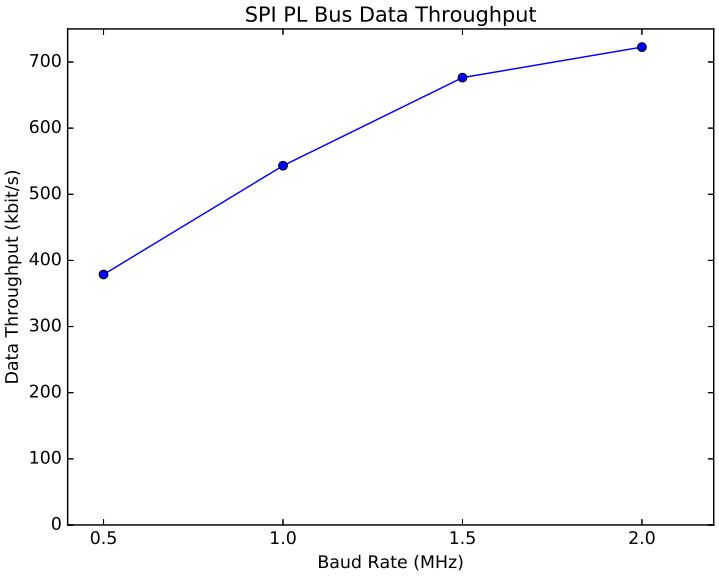


Figure 9.4: Plot of the SPI data throughput versus baud rate

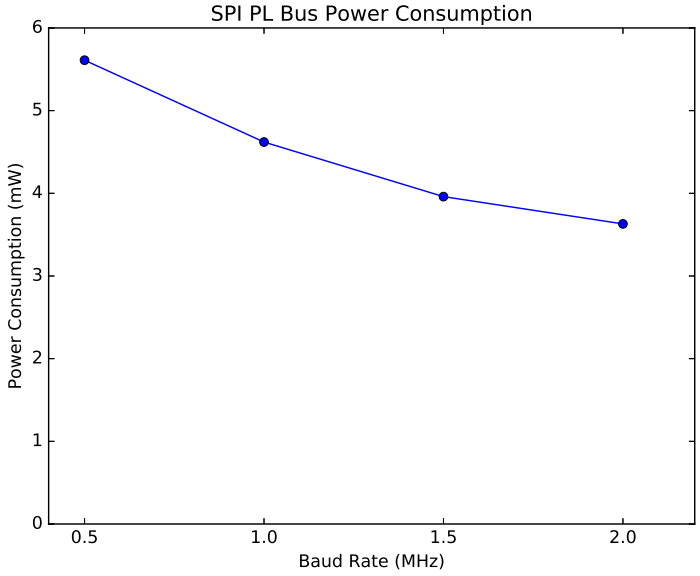


Figure 9.5: Plot of the SPI power consumption as a function of baud rate

9.4.3. Power Consumption

The idle power consumption of the SPI bus was found to be equal to approximately 0.5 mW for all different baud rates.

As one can see in Figure 9.5, the power consumption of the SPI bus when active is extremely low, reaching only several milliwatts in magnitude. The maximum 3σ error of these measurements were 0.18 mW. This is most likely due to the integrated SPI peripheral in the MSP432 and no need for other components such as buffers.

The downwards trend of the curve can be explained using the same reasoning as for the data throughput. Because the delay between the bytes does not decrease, but the time it takes to transmit a single byte does, the bus spends a larger relative amount of time in the idle state. As the bus uses more power when active, the mean power consumption goes up.

9.4.4. Complexity

As is apparent from the description of both the physical layer and data link layer of the SPI bus, the SPI bus is very simple to implement. Although SPI suffers from the same problem as RS485 where every byte must be handled in the main software loop, this is considered less of a problem in the PL bus.

9.5. Conclusion

SPI has proven to be straightforward in both hardware implementation and software development. As SPI is also implemented in nearly every microcontroller, compatibility between systems is not an issue.

One of the possible major drawbacks, like RS485, no full protocol is defined. However, as it is also highly impractical to implement a SPI-based TC bus, no real necessity exists for this. The versatile CS line enables most required functionality, including addressing.

The reducing slope of the data throughput as seen in Figure 9.4 combined with the reducing amount of power as seen in Figure 9.5 indicate some kind of optimum must exist, where the largest amount of data is transferred for the least amount of power. More tests and research might be necessary to find this point. Furthermore, this point is most likely dependent on the microcontrollers used.

10

Universal Serial Bus (USB)

The Universal Serial Bus (USB) is most likely well known to all readers, as it is the industry standard data bus for personal computer peripherals. Just as in the preceding chapters, this chapter will start with an introduction to the bus and its implementation, followed by an overview of the measurement results.

10.1. Introduction

As the development of the personal computer continued rapidly through the 1980s and 90s, hardware developers stuck to the older 'tried-and-proven' data buses such as RS232, amongst others to maximise the compatibility with existing systems. However, the computers and peripherals started to outperform the maximum capabilities of the bus standards. To increase performance and to homogenise the peripheral buses, USB was developed [35]. The first full version of USB, USB 1.0, was released in 1996, and was made part of Microsoft Windows 95. Although containing a lot of stability issues in its first issue, including it in Windows 98 saw a lot of positive changes. This paved the way for USB becoming the *de-facto* industry standard it is today.

As briefly discussed in chapter 3, USB versions 3.0 and 3.1 have been released relatively recently. This version features bus speeds up to 5 Gbit/s [35]. However, the low availability of USB 3.0/3.1 components able to interface to generic microcontrollers is means it is impossible to test it at the current time. Nevertheless, the previous and probably most common version, version 2.0, is widely supported for embedded applications and deemed to have sufficient data rates for a representative test.

USB is well known due to the large amount of consumer products supporting this standard. Apart from a basic physical layer description and protocol definition, the USB standards also define standardised connectors and component layouts. This ensures a high level of hardware compatibility between peripherals, also allowing hot plugging of devices. However, these capabilities are of high concern for consumer electronics, but not necessarily when developing systems running in spacecraft. What parts of the physical layer are implemented, and how this is done, will be explained in the next section.

Figure 10.1: A variety of USB connectors. Note the clear presence of the four pins in all connectors: two differential signalling pins, one +5.0 V pin and one ground pin. (Image courtesy: [61])

10.2. Physical Layer

The architecture of USB is based on a central master (the host controller), with direct point-to-point connections to other nodes. To create a network, hubs are required to act as an intermediate host between different segments [35].

The physical layer of USB is very similar to that of CAN: data is transferred through differential signalling over two dedicated channels. The positively biased line is dubbed the D+, and its complement the D- connection. The bits themselves are transferred as NRZ with additional bit stuffing ensuring correct timing synchronisation. However, one major difference between USB and CAN is how the logic level are defined [35]: a differential 0 means the 'logical' bit (the bit representing data) is equal to the preceding bit. A differential 1 means the logic bit has changed states. This means that when the first logic bit is a 0 and a differential 1 is received, the next logic bit is a 1. If a differential 0 is received next, the subsequent logic bit will also be 1. This encoding ensures better timing synchronisation and less overhead (stuffing bits) [35].

Other extra modes of the differential lines are also defined: when both D+ and D- are pulled to ground, a so-called Single-ended 0 state is signalled (see Figure 10.2). This is for example used to signal an End-Of-Packet (EOP), a node disconnect or a bus reset [35]. The opposite state, where D+ and D- are pulled to Vcc (high), is undefined and often used in error/fault detection.

Apart from the two differential lines, USB 2.0 connectors contain a +5.0 V power line and a ground connection, as may be seen in Figure 10.1. This is included in the standard to provide power to otherwise unpowered bus devices. A common example of such a device is a USB

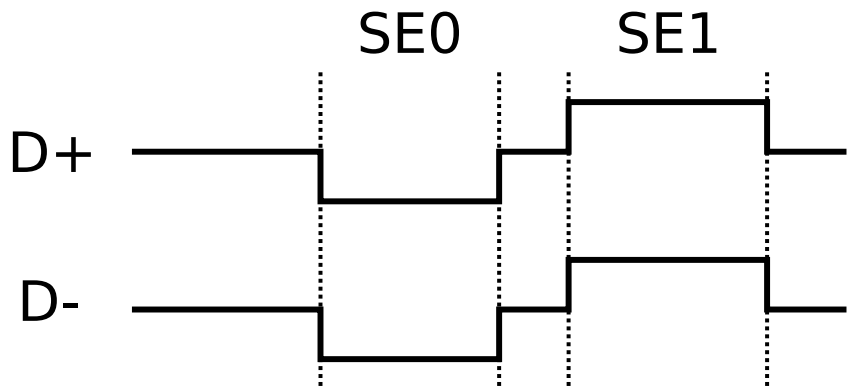


Figure 10.2: USB Single-Ended 0 and Single-Ended 1 states. For SE0, both differential lines are pulled to ground. For SE1, both lines are pulled to Vcc.

Table 10.1: The available USB host controllers.

Product	Supplier	USB Version(s)	# USB Ports	Board Interface	Power (operating)	Min-Max Temp
VNC1L	FTDI	2.0	2	UART, SPI, Parallel	82.5 mW	-40 to 85 °C
FT313H	FTDI	2.0	1	Direct SRAM, NOR Flash, General Multiplex	115.5 mW	-40 to 85 °C
MAX3421E	Maxim	2.0	1	SPI	148.5 mW	-40 to 85 °C
USB3300-EZK	Microchip	2.0	1	UTMI+	95.7 mW	-40 to 85 °C
VNC2-32L1B	FTDI	2.0	2	UART, (2x) SPI, Parallel	82.5 mW	-40 to 85 °C
EZ-Host	Cypress	1.0	4	SPI	246 mW	-40 to 85 °C

memory stick. Naturally, these lines are ignored within the test suite, as nodes are powered separately.

10.2.1. USB Controller Selections

As the MSP432 contains neither a USB peripheral controller nor a host controller, an external IC must be selected which is compatible with this microcontroller. The point-to-point nature of USB combined with its major implementation as a peripheral bus means it is usually implemented in systems as a USB slave, used to communicate to a personal computer. However, since there is no external host available in a closed satellite, a USB host controller must be added to the bus.

Table 10.1 contains the possible USB Host controllers. The EZ-Host by Cypress Semiconductors is discarded immediately due to its relatively high power consumption and no support for USB 2.0 or higher. Furthermore, the FT313H by FTDI and the USB3300-EZK by MicroChip are discarded due to requiring specialist interfaces, which are not supported by th MSP432 by default. The only options left are the VNC1L “Vinculum 1”, the VNC2-32L1B “Vinculum 2” by FTDI and the MAX3421E by Maxim Integrated. Although the former two are outperforming the last mentioned one in terms of power and features, the availability was deemed unacceptable due to an unavailable and a generally unsupported Integrated Development Environment (IDE) and corresponding compilers. Therefore, it was necessary to use the only remaining option in the table: the **MAX3421E** by Maxim Integrated.

The MAX3421E host controller is a ‘Full Speed’ controller, supporting a baud rate up to 12 MHz. It has an SPI interface similar to the MCP2515 CAN controller. However, the SPI clock speed can be configured to go up to 26 MHz.

10.3. Data Link Layer

The flexibility of USB is partially ensured by the vast protocol defined by the official standards. Several different types of packets are defined, all with their own specific purposes [35].

10.3.1. Setup, IN, OUT

The basis of the USB data link layer are the Setup, IN and OUT packets, their structure shown in Table 10.2. The Setup packets are used to configure nodes, assigning parameters such

Table 10.2: Structure of USB Setup, IN and OUT packets

Function	SYNC	PID	Address	Endpoint	CRC
No. of Bits	8	8	8	4	5

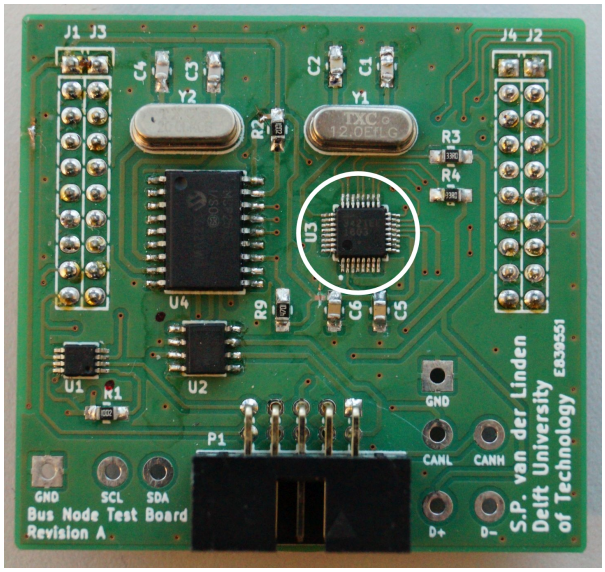


Figure 10.3: The USB controller as placed on the combined CAN/USB daughter board

Table 10.3: Structure of the frame markers

Function	SYNC	PID	Frame Number
No. of Bits	8	8	11

as the node's address. This is mainly important during the 'enumeration' stage: this is the point where the host controller detects the presence of a (new) node on the bus, and starts interrogating it to determine what type of device it is. Moreover, the supported USB modes are discovered. The related IN and OUT packets are used to signal a node to transmit or receive data respectively.

The first segment of the packet is the SYNC field, used to synchronise the bus nodes and acts as a delimiter for the start of the packet. The Packet Identifier (PID) describes the type of packet. The first four bits (out of eight) define the actual PID, the subsequent four bits are the complement of the first four, used in error checking. Several default PIDs are defined for common operations, such as querying the state of a peripheral or assigning an address. It also allows a fairly large number of 'free' PIDs for use by developers. The eight-bit address is designates the recipient of the message. This is mainly used to transport a packet across multiple nodes and hubs. The endpoint designates the target endpoint of the packet. The endpoints act as a multitude of buffers. Endpoint 0 is a required endpoint, used for Setup packets. The MAX3421e has four endpoints, including Endpoint 0. One is used for IN data, one for OUT data, and the final endpoint is used for overflowing data. Finally, a packet is closed with a five-bit CRC checksum.

10.3.2. Frame Markers

Additional features of USB apart from 'simple' data transfer include time-triggered messaging and interrupt packets. To enable this functionality, accurate real-time chronology is required. To do this, frame markers (Table 10.3) are transmitted every millisecond containing an 11-bit counter. Because this type of real-time messaging is not used within this project, this functionality is not analysed any further. However, the frame marker must still be transmitted

Table 10.4: Structure of the data packets

Function	SYNC	PID	Data	CRC
No. of Bits	8	8	0 - 8192	16

reliably according to the standards [35].

10.3.3. Data Packets

Once the host has transmitted an IN or OUT packet, the node (peripheral or host, dependent on the request) will start transmitting data. The structure of the individual data packets are similar to those of the Setup packets, but contain the data (up to 1024 bytes, although the MAX3421e has a 64 byte First In, First Out (FIFO) buffer) and a 16 bit CRC checksum.

Although the structure of the data packet is always the same, two PIDs are used: DATA0 and DATA1. The purpose of the two PIDs is to provide a toggle: the value is switched between the values for each transmitted data packet. If a packet is missed, the recipient will receive either DATA0 or DATA1 two times in a row, triggering a request to resend the data.

Other errors can be detected through the use of the CRC checksums. When errors are found through this, the USB controller must request a retransmit of the data. This is fully handled by the USB hardware without requiring external help from the node's microcontroller.

10.3.4. Additional Functionality

USB contains several built-in mechanisms which complement the packets defined within the protocol. The Single-Ended 0 state on the differential lines can be used to initiate a bus reset: a node connected to the bus must fully reset when this is requested. As this state is detected by hardware, it is thought to be more reliable than other protocols requesting resets through software.

The host controller must provide a weak pull-down voltage to the bus lines (typically with 15 kΩ) [35]. When a device is first connected, it must pull-up either D+ or D- to 3.3 V using a 1.5 kΩ resistor to notify the host it connected to the bus. Which line depends on the speed mode of the device: a full-speed device pulls up D+. A 'High Speed' device pulls up D- [35]. The host will also detect the disconnect of a device when the pull-up is lost in idle.

These additional mechanisms provide means for knowing the exact state of the bus and the peripheral device at any point in time.

10.4. MAX3421E Driver

The implementation of the software driver of the MAX3421E is fully according to the official programming guides [62][63], using the MSP432 SPI interface. In the test suite, the payload node is designated as the peripheral, and the OBC is denoted the USB host. Both nodes use the same MAX3421E IC, as it can take on both the host and peripheral roles.

As seen in the previous sections, the flexibility of USB means the standard protocol is relatively complicated. Fortunately, because the USB nodes are not required to connect to any external USB connector (although this could be a future option), the complexity of the driver can be

reduced by ignoring parts.

The initial data querying needed to be performed by a USB host is the enumeration stage. Amongst others, it must assign an address to the peripheral device and query the newly-connected device for meta-information. Information that is normally requested includes the vendor ID [64]. This vendor ID identifies the manufacturer of a system and is used to determine the correct driver software to use. Such a license must be obtained through the official registrar for a fee.

Since the (USB) test suite will always feature the same hardware, the enumeration stage is limited to just two parts:

1. **Address assign**: assigning an address, its value arbitrary, means a functional data link is created between the host and the peripheral/slave

2. **Check Endpoint 2 status**: this endpoint is used for IN data transfers (from the peripheral to the host). If the status is returned as available, it is clear that the bus is ready to be used

This simple enumeration procedure is thus mainly used to verify the availability and functionality of the attached peripheral. If the verification of endpoint 2 is unsuccessful, this either means that the address assignment failed, or the node is misconfigured/failed. The host then issues a bus reset and tries again.

The data transfers, being in one direction only, are straightforward in implementation. Assuming a successful enumeration phase, the actual bus tests can begin. The OBC, acting as host, will start by sending an IN packet to the payload. This signals the payload to start transmitting data in packets 64 bytes in size. Virtually all parts of the transfer are handled by the MAX3421E, leaving the OBC microcontroller to simply receive and check the incoming data.

10.5. Results

This section will present the results from the practical test of a USB bus. Note that the USB test bus is only implemented in the PL bus case.

10.5.1. Data Throughput

The MAX3421E can only be operated in USB Full Speed mode, so the bus' baud rate is fixed to 12 MHz. Therefore, the number of tests for both the data throughput and power consumption are limited to a single measuring point with the default baud rate. The measured average data throughput over five 10 s tests is 999.50 kbit/s $\pm$ 0.077 (3σ). Although this is the highest data rate seen during the tests, it is still only an 8% efficiency. Similar to the tests with CAN, this is most likely due to the additional bottleneck caused by the SPI interface and protocol. Furthermore, at high speeds, a large TX/RX buffer is necessary to fully achieve high performance. The 64 byte buffer in the MAX3421e is simply too small to reach the full capacity of the protocol.

10.5.2. Bit Error Rate

The BER was verified to be lower than 10^{-6} with 95% confidence by transmitting 3×10^6 bits without bit errors, following section 4.2.

Regarding noise injection, this was not performed on this PL bus. The choice for this is discussed in chapter 11.

10.5.3. Power Consumption

During the tests, the idle power consumption was found to be equal to 83.16 mW with a 3σ value much smaller than 1 mW. This value describes the overall power consumption, including both the OBC and payload. Furthermore, the MSP432's measured idle power has been excluded. For a fully active bus, where the payload is transferring bulk data to the OBC, the measured power consumption is 104.87 mW $\pm$ 0.18 mW (3σ).

10.5.4. Complexity

The rationale of USB's design is to allow for a very universal and flexible data bus. Unfortunately, this comes at a cost of additional complexity due to all the features. Yet, many of these features can simply be ignored, especially considering the fact that the bus nodes will not be used to connect to other third party USB devices. This makes enumeration, a common critical and perhaps problematic point in the USB protocol, much more straightforward.

Once devices are connected and enumerated, 'normal' data transfers are very straightforward to start and operate. The USB defines a lot of error checking to be performed by the Commercial Of The Shelf (COTS) components.

10.6. Conclusion

The USB has found to be fairly equivalent to CAN in terms of built-in reliability features within both the hardware and software. By making the assumption that the host and peripheral nodes are both known, a reduced enumeration stage can be performed, decreasing the overall complexity of the software.

Regarding the performance of the USB, this analysis case is thought to be similar to the CAN analysis. As the MAX3421E is an external IC operated through an SPI interface, the data throughput does not come close to the maximum theoretical value of 12 Mbit/s, only reaching approximately $1/12^{\text{th}}$ of that. To efficiently implement a USB host controller, a controller implemented as a microcontroller-peripheral must be chosen. An example of such functionality is found in the PIC24FJ64GB004 [65]. Unfortunately, this removes the possibility of implementing the bus on any arbitrary microcontroller.

Data bus designers might still choose to deviate from a truly universal bus, as a PL bus will only consist of a very small number of nodes. Therefore, it could be possible to specially select microcontrollers with built-in USB controllers or with other interfaces (e.g. USB 2.0 Transceiver Macrocell Interface (UTMI) [66]) to communicate with other types of USB controllers. Furthermore, it is necessary to select USB with larger built-in (double-buffered) TX/RX buffer, which is thought to be a significant data throughput bottle neck in the USB design utilised in this data bus test suite.

11

Data Bus Comparison

In this chapter the performed measurements will be discussed, leading to recommended data bus configurations and architectures to specific reference cases. Focus will be kept on comparing the different data bus standards to each other.

The preceding chapters have presented and discussed the results from the experimental tests on individual data bus standards and configurations. This chapter will bring all the results together, taking into account the criteria as defined in chapter 4.

11.1. Overview

Based on the measurements performed during this thesis research, a qualitative overview can be made. For the TC bus and PL bus, Table 11.1 and Table 11.2 contain these trade-offs respectively. All bus options have been assigned a score ranging from '- -' to '++', equalling -2 to +2. Rationale to the different scores will be given in the next several sections, but the basis is comparing the different bus options to each other.

Summing the scores results in the total scores shown in the tables. For the TC bus case, RS485 is found to have the highest score. For the PL bus case, SPI is found to be the winner.

Although the results are presented in the form of a trade-off table, limitations do exist to the conclusions that can be made from these tables. These will be discussed in section 11.6. The keen observer will have noticed that the criterion 'immunity to noise and transients' has not been included in the final trade-off. The reasoning behind this will be discussed in section 11.5.

Table 11.1: Trade-off table showing the main findings for the TC bus

Bus	Data Throughput	Power Consumption	Complexity	Total
I^2C	0	+	++	+3
CAN	- -	-	0	-3
RS485	++	++	+	+5

Table 11.2: Trade-off table showing the main findings for the PL bus

Bus	Data Throughput	Power Consumption	Complexity	Total
CAN	- -	-	0	-3
RS485	++	0	+	+3
SPI	+	++	++	+5
USB	++	+	0	+3

11.2. Data Throughput

Table 11.3 shows the different values of the measured data throughput for the nine-node bus cases, as previously discussed in each respective chapter. All data buses were implemented according to standard baud rate settings. In the case a range was defined, the maximum baud range was selected.

Comparing these values, it is clear that RS485 shows the highest effective data throughput, at 604 kbit/s.

For the PL buses, the measured values are shown in Table 11.4.

The trade-off scores in Table 11.1 and Table 11.2 have been determined directly using these results. For the TC bus, a clear difference between the different data buses exists. CAN scores the lowest, followed by I^2C. RS485 reaches an effective data throughput roughly three times higher than I^2C. Therefore, RS485 is assigned the highest score, with I^2C scoring a neutral 0. As CAN reaches just over half I^2C, it is given a single '-'. No single option is given the lowest option, simply because each one was capable of performing the realistic test case of one HK cycle per second without problems.

For the PL bus trade-off, a similar methodology is used. RS485 and USB reach approximately the same data throughput around 800 kbit/s and are therefore assigned the highest score ('++'). Although the third option, SPI, can be operated at higher baud rates (just like RS485), the 1 MHz baud rate is selected to compare the different options. The lower efficiency of SPI (54.3% versus RS485's 79.2%) is here the reason to assign only a single '+' to SPI. Finally CAN, showing the lowest bus efficiency, is given '- -' due to the relatively (and disappointingly) low performance.

Table 11.3: Average maximum data throughput of TC bus tests and their resultant efficiencies with respect to the standard baud rates

Bus	Data Throughput (kbit/s)	Baud Rate (kHz)	Efficiency (%)
I^2C	250.38	400	62.6
CAN	136.59	1000	13.7
RS485	603.55	1000	60.4
dI^2C	256.88	400	64.2
RS485 (no termination)	603.55	1000	60.4

Table 11.4: Average maximum data throughput of PL bus tests and their resultant efficiencies with respect to the standard baud rates

Bus	Data Throughput (kbit/s)	Baud Rate (MHz)	Efficiency (%)
CAN	158.8	1.0	15.9
RS485	792.1	1.0	79.2
SPI	543.2	1.0	54.3
USB	883.7	12	7.4

11.3. Power Consumption

The second criterion evaluated in the trade-off tables is the power consumption.

A comparison of the power consumption of the various data buses is shown in Figure 11.1, all as a function of the number of nodes on the bus. These plots are simply a superposition of the plots seen earlier in the bus' respective chapters. For the raw values, see Appendix C.

A problem with these straightforward plots is that they do not take into account the different data throughput values. For example, CAN is found to use less power than RS485 in a two-node bus configuration in the 'continuous' test case (Figure 11.1). However, CAN is only able to provide one sixth the data throughput of RS485 at that same measurement point. Therefore, a different approach must be taken to fairly compare the different buses.

Figure 11.2 shows a plot of each TC bus' power consumption, normalised to the measured throughput in the continuous test state. This results in an absolute energy spent per bit, a common measure in (wireless) communication [16]. A clear hierarchy is visible between the three options.

A similar figure can be created for the PL buses. The result is shown in Figure 11.3. The figure also shows a clear, but also equally distributed spread between the buses with the highest and lowest energy-per-bit ratings. Again, CAN uses up the most electrical energy per bit, using up nearly three times that of RS485, the second highest. USB and especially SPI are significantly more efficient.

These two figures are used to determine the trade-off scores shown in Table 11.1 and Table 11.2.

For the TC bus: as expected, CAN results in the lowest power efficiency with nearly 2.0 mJ/bit. On the other end, RS485 requires approximately 0.3 mJ/bit and I^2C uses slightly more. Thus, RS485 is assigned '++', I^2C a '+' and CAN the score '--'.

For the PL bus, SPI clearly has a very low power consumption, with the bar hardly visible in Figure 11.3. Therefore, SPI is given the highest score ('++'). Next, as can be observed in

Table 11.5: Comparison of PL bus power consumption

Bus	Idle Power (mW)	Active Power (mW)	Baud Rate (MHz)
CAN	41.91	91.476	1
RS485	16.2	181.5	1
SPI	1.7	4.6	1
USB	83.2	104.9	12

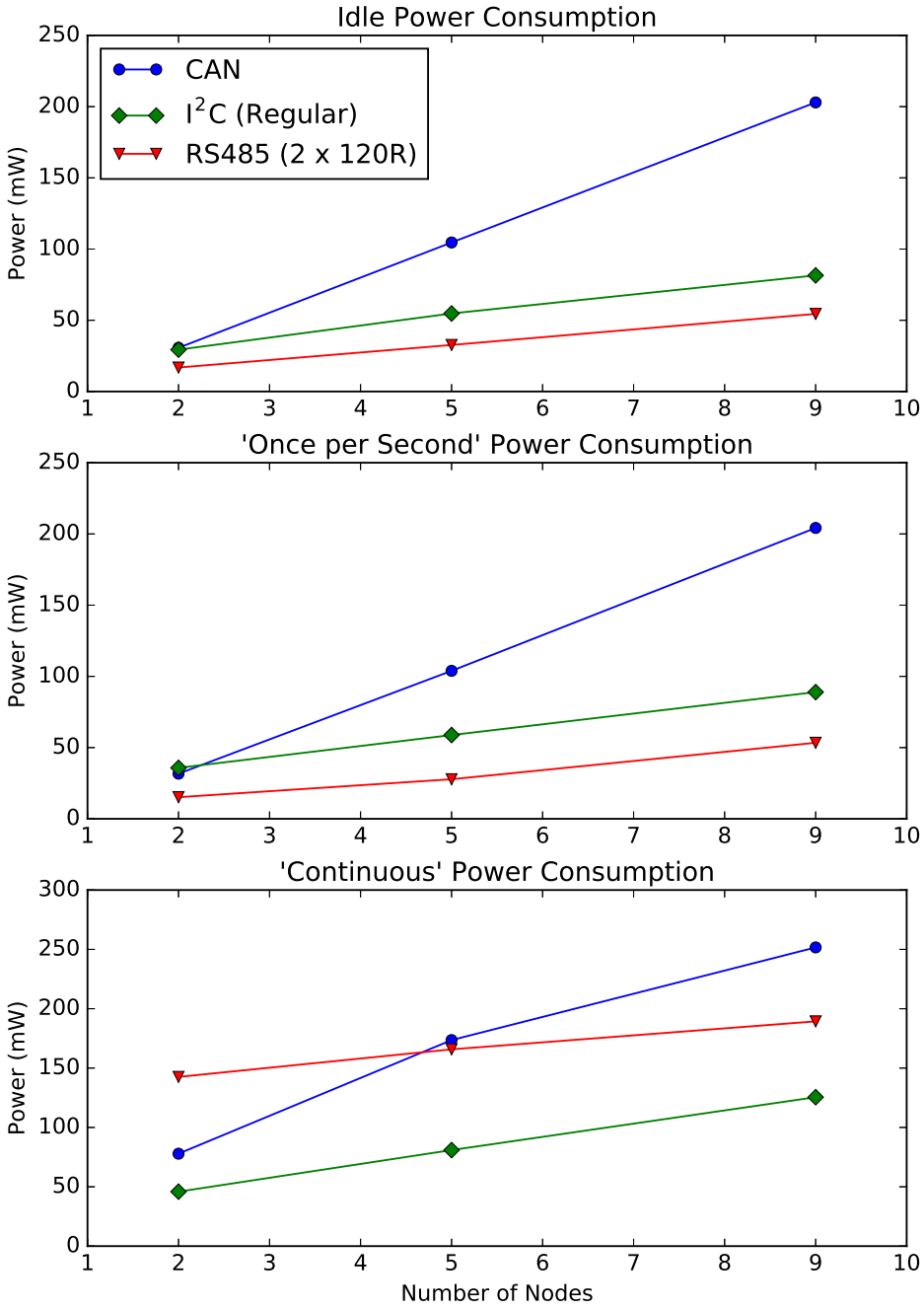


Figure 11.1: Plots comparing the three different test cases showing the behaviour of the different TC data buses with increasing number of nodes. All raw data may also be found in Appendix C.

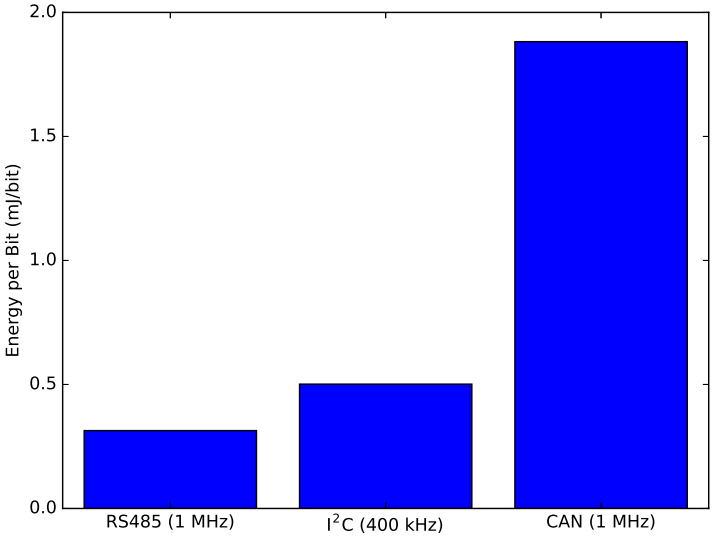


Figure 11.2: Normalised power consumption (mJ/bit) of the TC bus options.

Table 11.5, USB shows consistent behaviour: its idle power consumption does not vary much from its peak power consumption. This results in a fairly high idle power consumption, but a relatively low energy-per-bit value. It is assumed that in an actual spacecraft, the high speed bus would simply be switched off when not active, thus the high idle power is not deemed critical here. Therefore, a final score of '+' is given to USB. Finally, RS485 and CAN are given scores of '0' and '-' respectively reflecting their relative differences in energy-per-bit figures.

11.4. Complexity

The data throughput and power consumption are both criteria which can be assessed quantitatively. The complexity however, is something that can only be described qualitatively. The scores assigned to the different options in Table 11.1 and Table 11.2 are therefore derived from the experience gained during this research project and represent the amount of work expected to be required to implement the systems in different CubeSat missions.

CAN and USB were implemented using the MCP2515 and MAX3421E respectively, ICs external to the microcontroller. This additional layer of communication makes implementation and software development significantly more complex, although major parts of the protocol are fully handled by the IC. Thus, as the external IC adds both benefits and drawbacks, a neutral score of 0 is assigned to both CAN (both TC and PL) and USB.

Because SPI and RS485 are both part of most (if not all) microcontrollers, it is not deemed to be very difficult to implement in arbitrary CubeSats. However, SPI is given a higher score ('++') than RS485 ('+'), because RS485 has the problem that every node on the bus has to receive every packet completely. This problem requires additional software engineering to solve or reduce the effect it has on the overall system.

The final option, I^2C, is similar to SPI and RS485 that it is implemented within the microcontroller. As it does provide an internal addressing system in contrast to RS485, it is assigned the highest score ('++').

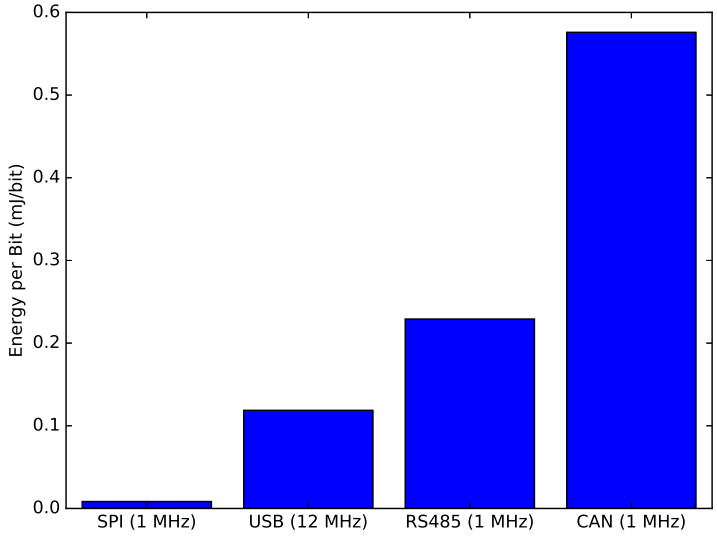


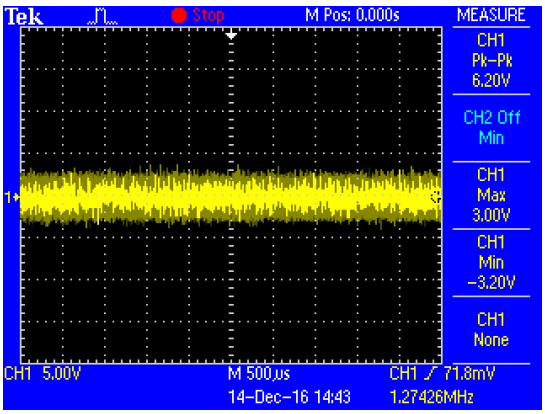
Figure 11.3: Normalised power consumption (mJ/bit) of the PL bus options.

11.5. Noise and Transient Effects

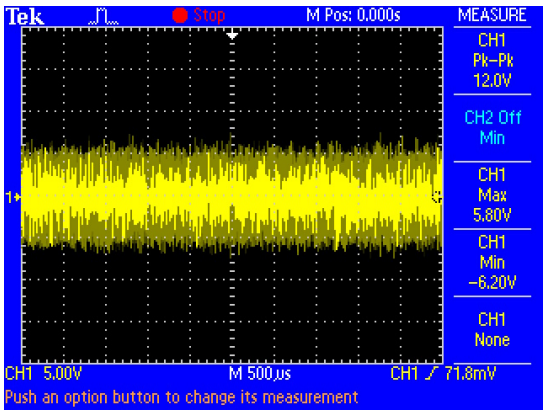
As mentioned in the introduction, the noise susceptibility and resultant BER/PER tests were not included in Table 11.1 and Table 11.2. This is because the results from these tests are not deemed to be representative for a real space environment.

As the generator is practically limited in both output voltage and frequency, the PDF will not equal an ideal Gaussian distribution. Therefore, its signal is measured to quantify its characteristics. Figure 11.4 shows two images (with the same scales) obtained from an oscilloscope connected directly to the signal generator. The signal is purely generated over the 50 Ω internal impedance of the generator. The left-hand image is of an 0.5 V RMS white noise signal, and the right-hand signal of a signal with 1.0 V RMS. As may be seen with the associated measurement values on the right hand side of each image, the peak-to-peak voltage is approximately 12 times larger than the RMS value. As most noise-induced problems started occurring between these two values, the peak-to-peak values equal roughly the maximum voltages for which the respective electrical components were designed. It was very apparent from the white noise tests using drivers/buffers featuring rise time regulators that these performed significantly better. This is mostly because the regulators act as a voltage clamp, effectively filtering out the problematic peaks. The injected transients, varying up to 10 V peak-to-peak result in a similar situation.

A second problem with the direct injection of the noise is the fact that the internal impedance of the generator changes the overall (system) impedance of the bus. This was most apparent when injecting noise in the unterminated bus lines during the RS485 tests. This resulted in the result that the unterminated bus performed better with induced noise than the terminated one, even though realistically the opposite is to be expected. The impedance of the generator reduced the signal's amplitude on the line with the noise, causing the unaffected line to overpower the noisy signal (see subsection 8.4.2). Realistically, (EMI) noise will be injected as coupled between the two differential bus lines, then cancelled out by the differential operation. Moreover, bus termination would increase the minimum energy needed by an interfering



(a) 0.5 V RMS



(b) 1.0 V RMS

Figure 11.4: Oscilloscope image of the voltage signal of two different white noise amplitudes, averaged over a two-second interval. Although the RMS values seem relatively small, the peak-to-peak voltages are more than ten times larger

signal to actually interfere [16].

Finally, the base BER/PER in a normal terrestrial environment. For all buses, it was determined that the BER and PER were less than their maximum values of 10^{-6} and 10^{-4} respectively[1] with 95% confidence.

Hence, because the results from the noise tests are therefore not deemed to be representative of realistic EMI or single event upsets, they are therefore omitted from the final 'trade-off'.

A different and more realistic way of assessing the susceptibility of the data buses to noise and other external influences is by injecting the noise and transients into the bus in the form of actual EMI by using radio transmitters and other (background) systems acting as sources. This makes it possible to assess the effects of choosing differential wiring over non-differential wiring as well as testing the various fault detection and isolation schemes used by the different bus components.

11.6. Discussion of Results

The tables given in section 11.1 give indications for recommended bus standards to use in a TC bus-like implementation and a PL bus case. For the former, RS485 is clearly regarded the preferred option. For the PL bus, the SPI comes forward as an optimum using these test conditions. However, these results are more intended to be as an indication of the direction future research into this subject must take rather than an absolute recommendation for a standard.

Firstly, this is due to having to omit the critical criterion of noise and transient effects from the final trade-off. It is still thought that this is the area CAN would excel at, as it is designed according to strict standard to operate in harsh environments. Also, other differential bus standards should theoretically provide sufficient benefits in this area.

Secondly, all data buses evaluated in the trade-off are implemented completely *on-spec*. As

[1]Note: the real values are definitely much better than these values, as the respective tests were performed many times during the software development, effectively increasing the size of the data set. Although it was not tracked by how much it increased.

was seen with the tests with dI^{2}C and removing the termination resistors from RS485, many variations exist on the exact implementations possibly increasing the performance of the system in one or more areas without significant drawbacks. Although, for example, the tweaking of the value of termination resistors is highly dependent on the actual application it is going to be used in, recommendations may be made regarding the process. One must also note that CAN and USB were implemented using external ICs. Significantly different results will probably be achieved if these are tested in the form of microcontroller peripherals.

Thirdly, the trade-off criteria are unweighted. The choice to do this was mainly because the trade-off tables were used to summarise the main findings of the research. Very few real-world applications will need to assume equal weighting to all criteria. To illustrate this, one can think of three cases: an educational 1U CubeSat in Low Earth Orbit (LEO), a 6U imaging CubeSat in LEO and another 6U CubeSat in lunar orbit.

For the 1U CubeSat, most likely the most important criterion will be power consumption. The maximum mass of a 1U CubeSat is 1.33 kg [67], hence the available electrical power is between one and two watts, based on historic average power per unit mass figures [6]. Furthermore, 1U CubeSats generally do not contain subsystems generating large amounts of data and a large amount of bus nodes. Therefore, developers of such a CubeSat would most likely find power consumption more important than data throughput and noise susceptibility.

Increasing the scale of the spacecraft, the 6U imaging spacecraft in LEO would require a high data rate, possibly over a separate PL bus. while having relatively more power available due to its larger surface area. Therefore, the data rate would possibly be of higher importance than power consumption.

Finally, the 6U CubeSat in lunar orbit will probably only have a low speed communication link to Earth due to the large distance, requiring large amounts of power. In this case, low power is more necessary over high data rate. Furthermore, the harsher radiation environment will cause larger importance for the noise effects.

To conclude, it is clear that it is hard to standardise one data bus architecture for use in multiple missions using only the information found in this research. Nevertheless, this thesis has looked at a universally implementable data bus regardless of the chosen microcontroller, and RS485 SPI are found to be the preferred choice in the general case, where a bus consists out of current-generation microcontrollers.

A specific discussion is required for the PL bus results. RS485 was the only bus able to exceed a data throughput of 1 Mbit/s. This result is perhaps disappointing, especially considering the fact that one of the original objectives of this thesis was to investigate data buses for future CubeSats, where high speed communications (already data rates up to 100 Mbit/s are being reached [68]) and large data sets are more common. Several different aspects come into play causing the low reported speeds.

Firstly, the clock frequency of 48 MHz of the chosen MSP432 microcontroller is simply too low to test very high bus speeds: a rule of thumb is that the bus clock (baud rate) must be at least ten times smaller than the microcontroller clock for reliable communication [69], limiting the maximum test speed to only 4.8 Mbit/s. Although this rule possibly becomes less strict at high speed point-to-point buses, it is still something to take into account. A possible future test might comprise of testing buses using high speed processors (not microcontrollers), or for a more realistic implementation, a test setup based on a chosen Field Programmable Gate Array (FPGA). Note that COTS CubeSat-specific FPGAs can already reach clock speeds up

to nearly 700 MHz [70].

Secondly, as a PL bus implementation is highly specific, it is often worthwhile to invest in microcontrollers (or FPGAs) capable of handling high speeds. This might include peripherals in the microcontroller that are able to handle specific buses such as USB. As both CAN and USB were tested using external ICs, the number of possible communication bottle necks increased. Especially in the case of USB, which has a minimum baud rate of 12 MHz, it is thought that the major limiting factor is the SPI interface.

Therefore, the PL bus tests are possibly representative for CubeSats developed in the last ten years, but not necessarily for all future CubeSats. However, starting of by splitting the TC bus and PL bus does give a lot of room for optimisation of both cases, with more research required for the latter to give insight into actual high speed performance. For example, the current 'winner' from the PL bus trade-off, SPI, has a single, non-differential signalling scheme. It is therefore possible that implementation at higher baud rates (i.e. more than 20 MHz) will become very difficult due to the larger effect of (increased) bus capacitance, something which is difficult to test with the current setup.

As a side note to the selection of SPI, one must consider that SPI is a non-differential standard, whereas many more recent standards optimised for speed (e.g. USB and Ethernet) are differential buses. Furthermore, the electrical characteristics of these buses are often constrained or prescribed in some way to maximise performance. Therefore, it is expected that SPI will hit a certain limit in terms of raw baud rate and data throughput due to its loosely defined physical layer. At this point, higher speed buses must take over at the cost of a higher power consumption and complexity.

12

Conclusions and Recommendations

Results obtained from during this thesis research are briefly summarised in this chapter, with the main conclusions split up between the TC bus and the PL bus. A final section will discuss the main general conclusions from this research, answering the research questions stated in the introduction.

12.1. TC Buses

Based on the measurements and analyses performed within this thesis, the optimally performing TC bus is found to be RS485. Its power consumption is generally on the same level as I^2C, while yielding much higher data throughput.

It is found that the power consumption of RS485, and any other differential bus standard for that matter, can be lowered through the customisation of the bus termination resistors. Increasing the overall bus impedance results in less current and hence a lower mean power consumption. However, care must be taken that the bus' susceptibility to EMI noise does not increase to critical levels, requiring a trade-off between it and power consumption.

The tested TC buses were selected for their ability to be implemented universally: I^2C and RS485 were possible to be used with the dedicated internal peripherals in the MSP432 microcontroller. In this case, CAN was the only TC bus that required an external IC. It was notable that CAN was also the worst performing bus in terms of both power and data 'efficiency'. Therefore, it is recommended to perform the CAN test again using a microcontroller that contains an internal CAN peripheral.

One useful bit of knowledge gained from the performed noise and transient tests is that they were very useful in detecting and isolating software bugs which would otherwise have not been found operating in a normal terrestrial environment. This is mainly due to the extreme voltages to which the components were subjected, causing indeterministic behaviour of the electronics. It is therefore recommended to subject any new bus software to such a test, avoiding possible bus lockups or loss of subsystems in space, as experienced with Delfi-C3 and Delfi-n3Xt.

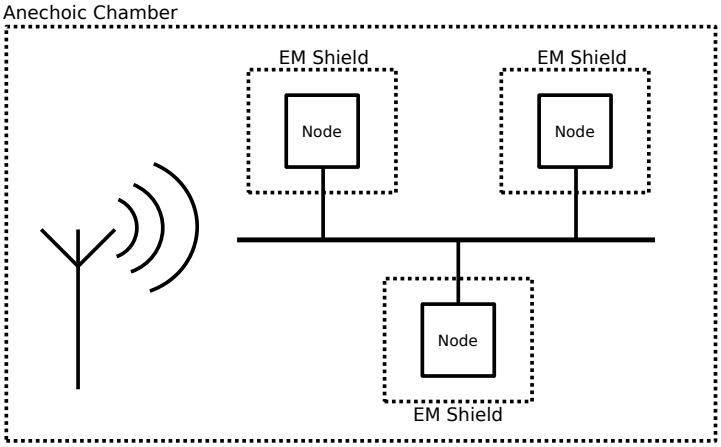


Figure 12.1: A schematic drawing of the proposal for more realistic EMI testing

12.2. PL Buses

From the measurements performed on several PL buses, it is found that SPI provides the optimum solution based on the selected microcontroller(s), mainly due to the low power consumption while still providing relatively high data throughput. It is however unclear how close the tested SPI was to the practical limits of the bus.

Despite aiming for high data rates, RS485 was the only tested PL bus able to reach and go over a data throughput of 1.0 Mbit/s during testing. Although both SPI and RS485 feature customisable baud rates over 1.0 MHz, significant timing issues and other software related problems occurred above the tested baud rates. This is mainly thought to be due to the MSP432's clock rate, which, although relatively high for a microcontroller, still is not high enough to provide the high data rates to test a realistic PL bus. A future test might be performed using an FPGA or even a microprocessor to provide the high data rates. Therefore, the result found in this thesis must be regarded as preliminary and as an indication for future research.

12.3. Recommended EMI Testing

It has been mentioned several times that the 'noise and transient effects' testing performed in this research was, although giving drastic results, not realistic. The resultant large, unrealistic voltage spikes and the fact that the signal generator influences the electronic characteristics of the bus invalidates the necessary comparisons between the different bus standards.

Therefore, a different testing methodology is proposed to circumvent these issues. To realistically inject the (coupled) noise into the bus lines, an actual Radio Frequency (RF) transmitter is required, as seen in Figure 12.1. To avoid the EMI from interfering with the bus nodes' operation, these need to be shielded in separate electromagnetic shields.

An actual test would need to vary two pairs of two parameters. On the bus testing hardware side, the number of nodes and the length of the bus lines must be varied, similar to the test performed in the current research. On the transmitter's side, the frequency and signal amplitude must be simulated.

One must take into account that there are essentially two main types of (continuous) EMI: the low-intensity interference from other electronic systems, and EMI caused by the transmission

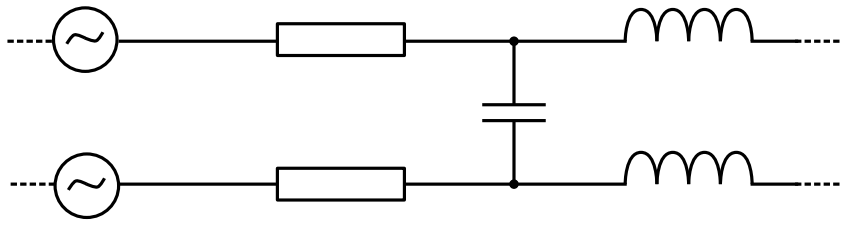


Figure 12.2: Schematic view of the recommended transient testing. The capacitor and inductors signify the induced coupling between the wires and are therefore not physical components

of external electromagnetic signals. It is expected that both types can be simulated using the same RF transmitter by choosing the correct frequencies and intensities. For example, VHF (30 MHz to 300 MHz [16]) or similar frequencies transmitted at several watts simulate a typical CubeSat data link, while slightly lower frequencies (10 MHz to 20 MHz) transmitted at low intensities can simulate EMI caused by systems operating on typical crystal oscillators.

A simple test setup as described, together with a well-defined range of frequencies and signal intensities, should be able to simulate the EMI effects more realistically. However, this test setup needs to be altered slightly when considering transients effects.

As these transients are typically caused by other subsystems within the satellite, and not actual transients driven onto the bus lines, a realistic coupling between the bus wires and some kind of power line must be simulated. Therefore, it is recommended to replace the RF transmitter in the test setup in Figure 12.1 with wires carrying significant transients running parallel to the bus wire(s). The resultant situation would become something like that shown in Figure 12.2, with two electrically coupled wires carrying different signals.

In the end, a reliable noise and transient test, together with the power and throughput measurements performed in this study, should result in additional information to help find a suitable bus architecture for future CubeSat and picosatellite missions.

12.4. General Conclusions and Recommendations

This thesis started off with the following question:

> Is there an optimal data bus or combination of data buses to be used in future CubeSat missions?

and the first subquestion:

> How do the proposed bus standards compare to I^2C in performance, reliability, power consumption and practical implementation?

Based on a bus architecture optimised by taking into account data throughput, power consumption and complexity, the main TC bus segment is recommended to be based on RS485. If a certain system, representing similar data rates and nodes as tested with the PL bus test suite requires a high speed bus, the SPI bus is recommended.

RS485 provides both higher data throughput and lower power consumption than I^2C when implemented as a TC bus, while the basic architecture of the standard based on the microcontroller's UART provides straightforward implementation of the bus. Furthermore, the use of

differential signalling theoretically provides higher robustness to line cross-talk and externally induced EMI.

SPI is actually very similar to RS485, but can reduce power consumption even further and omit the need for a (full) protocol through its point-to-point design and the use of a separate CS line. This also simplifies software design compared to other buses, although it was found SPI is more susceptible to timing issues.

The keyword in the main question is 'optimal': the trade-off performed in section 11.6 is perhaps too generic: focusing on a specific mission and its requirements will likely produce a different result. This is not due to the selected bus standards, but more down to the large variety of missions that are possible. It is unfortunate that no bus standard exists that will provide the 'safety' of CAN, the flexibility of I^2C and the speed of RS485, therefore trade-offs will probably still be necessary for specific cases.

One way of finding out which data bus would be the baseline for future designs is to perform a trade-off using criteria which are assigned a weighting according to input from the CubeSat community. If done correctly, this should result in a data bus standard architecture which is suitable for the majority of missions, making more COTS hardware and software compatible with new missions.

The final subquestion is as follows:

> What are the practical limitations of the proposed bus architecture?

The chosen bus for the TC bus, RS485, has one major drawback: all messages sent over the bus must be handled in software by every node on the bus. This means that the bus has two different factors defining its capacity: the first one is the data throughput of the bus, while the second one is the maximum data throughput the bus may have before it starts slowing down time-critical processes on the bus. Therefore, follow-up research is necessary in this area. First, the effect this has on time-critical systems must be quantified. Secondly, if this is indeed deemed a problem, and it is assumed so, then possibilities for reducing or removing this effect must be explored. One possibility is the use of address bits, although this might not be supported by all microcontrollers and other electronics.

A final general recommendation is to always use electronic components with rise time regulators/accelerators in space applications, as they come with several benefits. First: by reducing the amount of time the signal stays active, the power consumption of a signal can be lowered. However, this must be balanced with the added power consumption of the regulator itself. A rise time accelerator tuned to the selected baud rate is therefore essential. Secondly, the regulators are able to clamp the inputs either up or down, effectively filtering out large spikes. This could increase the tolerance of systems versus single-event upsets without increasing the amount of effort required to implement the bus.

Ultimately, this thesis report has provided an initial recommendation for future CubeSat and picosatellite missions, but more research in the data bus robustness and performance in high speed applications is required. Nevertheless, this research's legacy is a fully functional data bus testing suite which is straightforward to be extended in the proposed future tests, and provides a well-defined test bed and reference for the development of new data bus technologies.

Bibliography

[1] H. Heidt, J. Puig-Suari, A. S. Moore, S. Nakasuka, and R. J. Twiggs, "CubeSat: A new Generation of Picosatellite for Education and Industry Low-Cost Space Experimentation", in *Proceedings of the 14th annual aiaa/usu conference on small satellites*, Logan, USA, 2000 (Cited pp. 1 and 15).

[2] R. Nugent, R. Munakata, A. Chin, R. Coelho, and J. Puig-Suari, "CubeSat: The Pico-Satellite Standard for Research and Education", in *Aiaa space 2008 conference & exposition*, 2008, pp. 1–11, isbn: 978-1-62410-002-4. doi: 10.2514/6.2008-7734. [Online]. Available: http://arc.aiaa.org/doi/pdf/10.2514/6.2008-7734 (Cited p. 1).

[3] E. Buchen, "Small Satellite Market Observations", *Aiaa/usu conference on small satellites*, pp. 1–5, 2015. [Online]. Available: http://digitalcommons.usu.edu/smallsat/2015/all2015/51 (Cited pp. 1 and 2).

[4] Planet Labs. (Jun. 2016). Planet - Home, [Online]. Available: https://www.planet.com/ (Cited p. 2).

[5] NXP Semiconductors, *UM10204 I²C-bus specification and user manual*, 2014. [Online]. Available: http://www.nxp.com/documents/user%7B%5C_%7Dmanual/UM10204.pdf (Cited pp. 2, 17, 22, and 47).

[6] J. Bouwmeester and J. Guo, "Survey of worldwide pico- and nanosatellite missions, distributions and subsystem technology", *Acta astronautica*, vol. 67, no. 7-8, pp. 854–862, 2010. doi: 10.1016/j.actaastro.2010.06.004 (Cited pp. 2, 3, 13, 17, 18, 19, 31, 61, and 108).

[7] K. Laizans, I. Sünter, K. Zalite, H. Kuuste, M. Valgur, K. Tarbe, V. Allik, G. Olentšenko, P. Laes, S. Lätt, and M. Noorma, "The design of fault tolerant Command and Data Handling Subsystem for ESTCube-1", in *Proceedings of the estonian academy of sciences*, 2014, pp. 222–231. doi: 10.3176/proc.2014.2S.03 (Cited p. 2).

[8] S. van der Linden, "A Novel Databus Design for CubeSats", Delft University of Technology, Literature Survey, 2015 (Cited pp. 2, 18, and 21).

[9] R. Schoemaker, "Robust and Flexible Command & Data Handling On Board the Delffi Formation Flying Mission", MSc Thesis, Delft University of Technology, 2014 (Cited p. 2).

[10] K. McCabe, "Enhancements to the CPX I2C Bus", California Polytechnic State University, Tech. Rep., 2007 (Cited p. 2).

[11] Delfi Space. (Jun. 2016). TU Delft Small Satellite Program, Delft University of Technology, [Online]. Available: http://www.delfispace.nl/ (Cited p. 3).

[12] R. L. Staehle, B. Anderson, B. Betts, D. Blaney, C. Chow, L. Friedman, H. Hemmati, D. Jones, A. Klesh, P. Liewer, J. Lazio, M. W.-y. Lo, P. Mouroulis, N. Murphy, P. J. Pingree, J. Puig-suari, T. Svitek, A. Williams, and T. Wilson, "Interplanetary CubeSats: Opening the Solar System to a Broad Community at Lower Cost", *Journal of small satellites*, vol. 2, no. 1, pp. 161–186, 2012 (Cited p. 2).

[13] E. Peragin, H. Diez, F. Darnon, D. Belot, J. Millerioux, J. Issler, T. Dehaene, Y. Richard, G. Guillois, F. Sepot, and D. Simon, "X band downlink for CubeSat", *26th annu. aiaa/usu small satellite conf.*, 2012. [Online]. Available: http://digitalcommons.usu.edu/smallsat/2012/all2012/52/ (Cited p. 3).

[14] V. Riot, L. Simms, D. Carter, T. Decker, J. Newman, L. Magallanes, J. Horning, D. Rigmaiden, M. Hubbell, and D. Williamson, "Government-owned CubeSat Next Generation Bus Reference Architecture", in *Proceedings of the 28th aiaa/usu conference on small satellites*, Logan, UT, 2014 (Cited p. 3).

[15] A. Scholz and J. N. Juang, "Toward open source CubeSat design", *Acta astronautica*, vol. 115, no. June 2003, pp. 384–392, 2015, issn: 00945765. doi: 10.1016/j.actaastro.2015.06.005 (Cited p. 3).

[16] N. Storey, *Electronics: a System Approach*, Fifth. Harlow, United Kingdom: Pearson Education Ltd., 2013, isbn: 978-0-273-77327-6 (Cited pp. 8, 15, 17, 49, 51, 88, 103, 107, and 113).

[17] N. Briscoe, "Understanding the OSI 7-layer model", *Pc network advisor*, no. 120, pp. 13–15, 2000. [Online]. Available: http://enhanceedu.iiit.ac.in/ttp/images/5/5c/Osi.pdf (Cited pp. 8 and 9).

[18] E. Orsel, "Power Modelling and Optimisation of a Communication bus for Small Satellite Missions", MSc Thesis, Delft University of Technology, 2016. [Online]. Available: http://resolver.tudelft.nl/uuid:cef6b8a3-b7e0-4a86-932f-e35027091bb8 (Cited p. 10).

[19] I. Moir, A. Seabridge, and M. Jukes, *Civil Avionics Systems*, 2nd ed. Chichester, UK: John Wiley & Sons, Ltd, Aug. 2013, isbn: 9781118536704. doi: 10.1002/9781118536704 (Cited p. 10).

[20] S. van der Linden, J. Bouwmeester, and A. Povalac, "Design and Validation of an Innovative Data Bus Architecture for CubeSats", in *14th reinventing space conference*, London, UK: The British Interplanetary Society, 2016. [Online]. Available: http://rispace.org/2016/papers/BIS-RS-2016-10-vanderlinden.pdf (Cited pp. 13 and 129).

[21] J. Bouwmeester, M. Langer, and E. Gill, "Survey on the implementation and reliability of CubeSat electrical bus interfaces", *Ceas space journal*, Sep. 2016, issn: 1868-2502. doi: 10.1007/s12567-016-0138-0. [Online]. Available: http://link.springer.com/10.1007/s12567-016-0138-0 (Cited pp. 13 and 16).

[22] R. R. Dobkin, A. Morgenshtein, A. Kolodny, and R. Ginosar, "Parallel vs . Serial On-Chip Communication", in *Proceedings of the 2008 international workshop on system level interconnect prediction*, Newcastle, UK: ACM, 2008, pp. 43–50 (Cited p. 15).

[23] F. Nohka, M. Drobczyk, and A. Heidecker, “Experiences in Combining Cubesat Hardware and Commercial Components from Different Manufacturers in order to build the Nano Satellite AISat/Clavis-1”, in *Proceedings of the aiaa/usu conference on small satellites, mission lessons*, 2012 (Cited p. 17).

[24] LIN Consortium, *LIN Specification Package (2.2A)*, 2010 (Cited pp. 19 and 22).

[25] D. Awtrey, “Transmitting Data and Power over a One-Wire Bus Transmitting Data and Power over a One-Wire Bus”, *Sensors, the journal of applied sensing technology*, 1997, issn: 07469462 (Cited pp. 19 and 22).

[26] P. Horowitz and W. Hill, *The Art of Electronics*, 3rd ed. New York, NY: Cambridge University Press, 2015, isbn: 978-0-521-80926-9 (Cited pp. 19, 20, 22, 37, 51, 73, and 75).

[27] A. Baron, I. Walter, R. Ginosar, I. Keslassy, and O. Lapid, “Benchmarking SpaceWire Networks”, in *International spacewire conference*, Dundee, Scotland, 2007 (Cited pp. 19 and 22).

[28] F. Leens, “An introduction to I2C and SPI protocols”, *Ieee instrumentation & measurement magazine*, vol. 12, no. 1, 2009, issn: 1094-6969. doi: 10.1109/MIM.2009.4762946 (Cited pp. 19, 23, 47, 87, and 88).

[29] Microchip Technology, *MCP2515 - Stand-alone CAN Controller With SPI Interface (Datasheet)*, 2012. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/21801e.pdf (Cited pp. 18, 69, and 70).

[30] Texas Instruments, *SN65HVD23x 3.3-V CAN Bus Transceivers Datasheet*, 2015. [Online]. Available: http://www.ti.com/lit/ds/slls933g/slls933g.pdf (Cited p. 18).

[31] Wiznet, *W5100 Datasheet*, 2008. [Online]. Available: http://www.wiznet.co.kr/wp-content/uploads/wiznethome/Chip/W5100/Document/W5100%7B%5C_%7DDatasheet%7B%5C_%7Dv1.2.6.pdf (Cited pp. 19 and 22).

[32] NXP Semiconductors, *PCA9513A / PCA9514A (Datasheet)*, 2009. [Online]. Available: http://cache.nxp.com/documents/data%7B%5C_%7Dsheet/PCA9513A%7B%5C_%7DPCA9514A.pdf (Cited pp. 20, 49, and 58).

[33] S. de Jong, G. T. Aalbers, and J. Bouwmeester, “Improved command and data handling system for the Delfi-n3Xt nanosatellite”, in *International astronautical congress*, Glasgow, Scotland, 2008 (Cited pp. 20 and 49).

[34] ST3485, *ST3485 RS-422/RS-485 Transceiver Datasheet*, 2016. [Online]. Available: www.st.com/resource/en/datasheet/CD00003137.pdf (Cited p. 20).

[35] J. Axelson, *USB Complete: The Developer’s Guide*, 4th ed. Madison WI, USA: Lakeview Research LLC, 2009, isbn: 978-1-931448-08-6 (Cited pp. 20, 93, 94, 95, and 97).

[36] Maxim Integrated, *MAX3421E (Datasheet)*, 2007. [Online]. Available: https://datasheets.maximintegrated.com/en/ds/MAX3421E.pdf (Cited pp. 23 and 24).

[37] F. Dekking, C. Kraaikamp, H. Lopuhaä, and L. Meester, *A modern introduction to probability and statistics: Understanding why and how*. London, UK: Springer-Verlag, 2005 (Cited pp. 26, 27, and 28).

[38] M. Müller, R. Stephens, and R. McHugh, "Total Jitter Measurement at Low Probability Levels, Using the Optimized BERT Scan Method", Tech. Rep., 2007, p. 18. [Online]. Available: http://cp.literature.agilent.com/litweb/pdf/5989-2933EN.pdf (Cited pp. 27, 28, and 29).

[39] L. Barford, "Sequential Bayesian Bit Error Rate Measurement", *Ieee transactions on instrumentation and measurement*, vol. 53, no. 4, pp. 947–954, Aug. 2004, issn: 0018-9456. doi: 10.1109/TIM.2004.831129 (Cited p. 27).

[40] J. Guo, J. Bouwmeester, and E. Gill, "In-orbit results of Delfi-n3Xt: Lessons learned and move forward", *Acta astronautica*, vol. 121, pp. 39–50, 2016, issn: 00945765. doi: 10.1016/j.actaastro.2015.12.003. [Online]. Available: http://dx.doi.org/10.1016/j.actaastro.2015.12.003 (Cited p. 29).

[41] Texas Instruments, *MSP432P401R , MSP432P401M Mixed-Signal Microcontrollers (Datasheet)*, 2016. [Online]. Available: http://www.ti.com/lit/ds/symlink/msp432p401r.pdf (Cited pp. 31, 34, and 45).

[42] Keysight Technologies, *Keysight 33210A Waveform Generator*, 2014 (Cited pp. 31 and 85).

[43] M. Shah, A. Juneja, S. Bhattacharya, and A. G. Dean, "High frequency GaN device-enabled CubeSat EPS with real-time scheduling", *2012 ieee energy conversion congress and exposition, ecce 2012*, no. 1116850, pp. 2934–2941, 2012. doi: 10.1109/ECCE.2012.6342522 (Cited p. 32).

[44] A. E. Kalman and P. D, "How to Accommodate Additional Processors in the CubeSat Kit ™", in *Cubesat workshop developers' conference*, Huntingdon Beach, CA, 2007, pp. 1–22. [Online]. Available: http://mstl.atl.calpoly.edu/%7B~%7Dbklofas/Presentations/DevelopersWorkshop2007/Kalman%7B%5C_%7DAndrew.pdf (Cited p. 34).

[45] C. Brandon, "Use of Ada in a Student CubeSat Project Carl", *Ada user journal*, vol. 29, no. 3, pp. 213–216, 2008, issn: 13816551 (Cited p. 34).

[46] Texas Instruments, *MSP430F677xA , MSP430F676xA , MSP430F674xA Polyphase Metering SoCs (Datasheet)*, 2014. [Online]. Available: http://www.ti.com/lit/ds/symlink/msp430f6745a.pdf (Cited p. 34).

[47] ——, (Jul. 2015). Boosterpack pinout standard, [Online]. Available: http://www.ti.com/ww/en/launchpad/dl/boosterpack-pinout-v3.pdf (Cited pp. 34 and 35).

[48] PC/104 Embedded Consortium, *PC / 104 Specification*, 2008. [Online]. Available: www.pc104.org (Cited p. 35).

[49] Keysight Technologies, *Keysight technologies - 34401A Digital Multimeter*, 2016. [Online]. Available: http://www.keysight.com/en/pd-686884-pn-34405A/digital-multimeter-5-digit?cc=US%7B%5C&%7Dlc=eng (Cited p. 37).

[50] J. Bouwmeester, L. Rotthier, C. Schuurbiers, W. Wieling, G. V. D. Horn, F. Stelwagen, E. Timmer, and M. Tijssen, “Preliminary results of the Delfi-n3Xt mission”, in *The 4s symposium, porto petro*, 2014 (Cited p. 38).

[51] N. Cornejo, J. Bouwmeester, and G. Gaydadjiev, “Implementation of a Reliable Data Bus for the Delfi Nanosatellite Programme”, in *7th iaa symposium on small satellites for earth observation*, Berlin, Germany, 2009 (Cited pp. 49 and 60).

[52] NXP Semiconductors, *PCA9615 (Datasheet)*, 2016. [Online]. Available: http://www.nxp.com/documents/data%7B%5C_%7Dsheet/PCA9615.pdf (Cited pp. 50 and 58).

[53] M. Castro, R. Sebastián, F. Yeves, J. Peire, J. Urrutia, and J. Quesada, “Well-known serial buses for distributed control of backup power plants. RS-485 versus Controller Area Network (CAN) solutions”, *Iecon proceedings (industrial electronics conference)*, vol. 3, pp. 2381–2386, 2002. doi: 10.1109/IECON.2002.1185345 (Cited p. 51).

[54] Energia, *Energia*, 2016. [Online]. Available: http://energia.nu/ (visited on 12/02/2016) (Cited p. 52).

[55] (2016). Dwire, Delft University of Technology, [Online]. Available: https://github.com/DelfiSpace/DWire (Cited p. 53).

[56] W. Lawrenz, *CAN System Engineering*, 2nd ed. London: Springer-Verlag, 2013, isbn: 978-1-4471-5612-3. doi: 10.1007/978-1-4471-5613-0 (Cited pp. 61, 63, and 64).

[57] M. Di Natale, H. Zeng, P. Giusto, and A. Ghosal, *Understanding and Using the Controller Area Network Communication Protocol*. New York, NY: Springer New York, 2012, isbn: 978-1-4614-0313-5. doi: 10.1007/978-1-4614-0314-2 (Cited pp. 61, 62, and 63).

[58] K. Gingerich, “RS-485 Unit Load and Maximum Number of Bus Connections”, Texas Instruments, Tech. Rep., 2004. [Online]. Available: http://www.ti.com/lit/an/slla166/slla166.pdf (Cited p. 74).

[59] STMicroelectronics, *ST3485E Datasheet*, 2002. [Online]. Available: http://www.st.com/resource/en/datasheet/st3485eiy.pdf (Cited p. 74).

[60] Y. Wang and K. Song, “A new approach to realize UART”, in *Proceedings of 2011 international conference on electronic & mechanical engineering and information technology*, vol. 5, IEEE, Aug. 2011, pp. 2749–2752, isbn: 978-1-61284-087-1. doi: 10.1109/EMEIT.2011.6023602. [Online]. Available: http://ieeexplore.ieee.org/document/6023602/ (Cited p. 77).

[61] V. Viitanen. (Jun. 2010). Usb connectors. Image only, [Online]. Available: https://en.wikipedia.org/wiki/File:Usb_connectors.JPG (Cited p. 94).

[62] Maxim Integrated, *AN3598 - MAX3420E Programming Guide*, 2005. [Online]. Available: http://pdfserv.maximintegrated.com/en/an/AN3598.pdf (Cited p. 97).

[63] ——, *AN3785 - MAX3421E Programming Guide*, 2006. [Online]. Available: http://pdfserv.maximintegrated.com/en/an/AN3785.pdf (Cited p. 97).

[64] USB Implementers Forum. (Dec. 2016). Usb-if logo trademark license agreement and usage guidelines, [Online]. Available: http://www.usb.org/developers/logo_license/ (Cited p. 98).

[65] Microchip Technology Inc., *Pic24fj64gb004 datasheet*, 2010 (Cited p. 99).

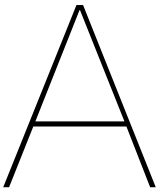
[66] NXP Semiconductors, *UTMI+ Specification*, 2004. [Online]. Available: http://www.nxp.com/assets/documents/data/en/brochures/UTMI-PLUS-SPECIFICATION.pdf (Cited p. 99).

[67] S. Lee, A. Hutputanasin, A. Toorian, W. Lan, R. Munakata, J. Carnaham, D. Pignatelli, and A. Mehrparvar, *Cubesat Design Specification*, San Luis Obispo, 2015. [Online]. Available: http://www.cubesat.org/s/cds%7B%5C_%7Drev13%7B%5C_%7Dfinal2.pdf (Cited p. 108).

[68] D. Gerhardt, M. Bisgaard, L. Alminde, R. Walker, M. A. Fernandez, A. L. Syrlinks, R. R. Keller, and J.-L. Issler, "GOMX-3: Mission Results from the Inaugural ESA In-Orbit Demonstration CubeSat", in *30th annual aiaa/usu conference on small satellites*, Logan, UT, 2016 (Cited p. 108).

[69] R. J. Hamann, J. Bouwmeester, and G. Brouwer, "Delfi-C3 Preliminary Mission Results", in *Proceedings of the 23rd annual aiaa/usu small satellite conference*, 2009, SSC09–IV–7 (Cited p. 108).

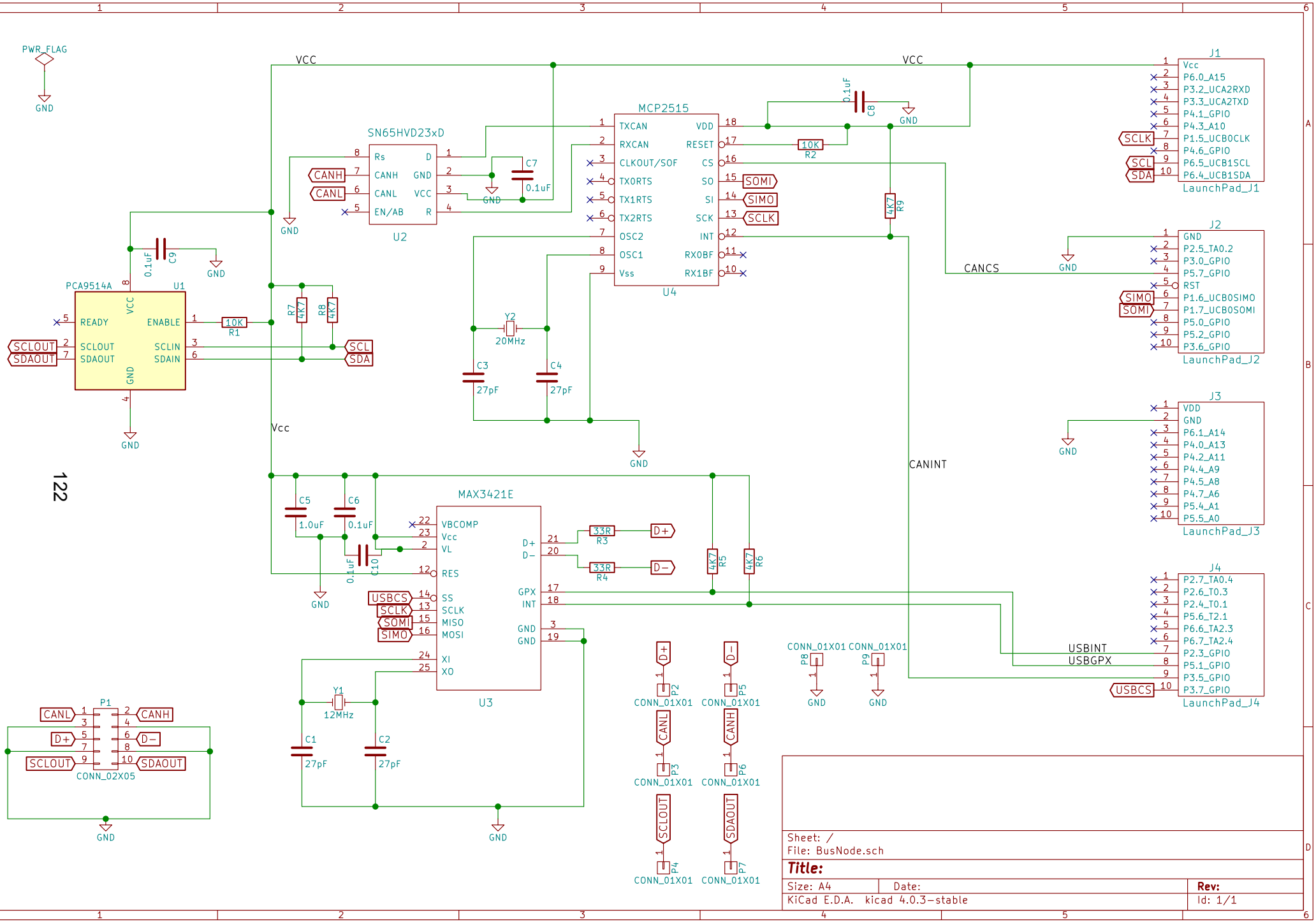
[70] GomSpace, *NanoMind Z7000 Datasheet*, 2016 (Cited p. 109).

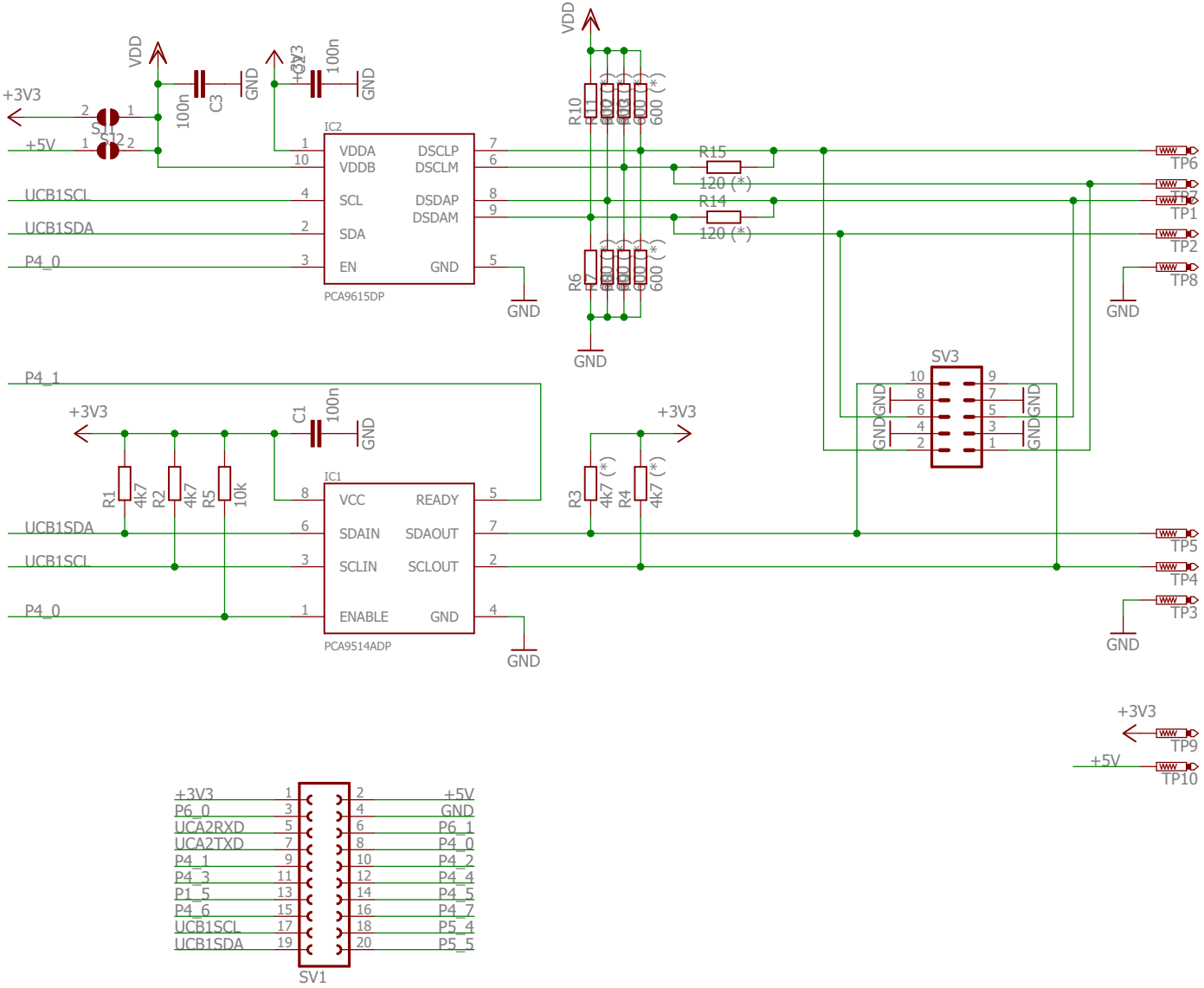


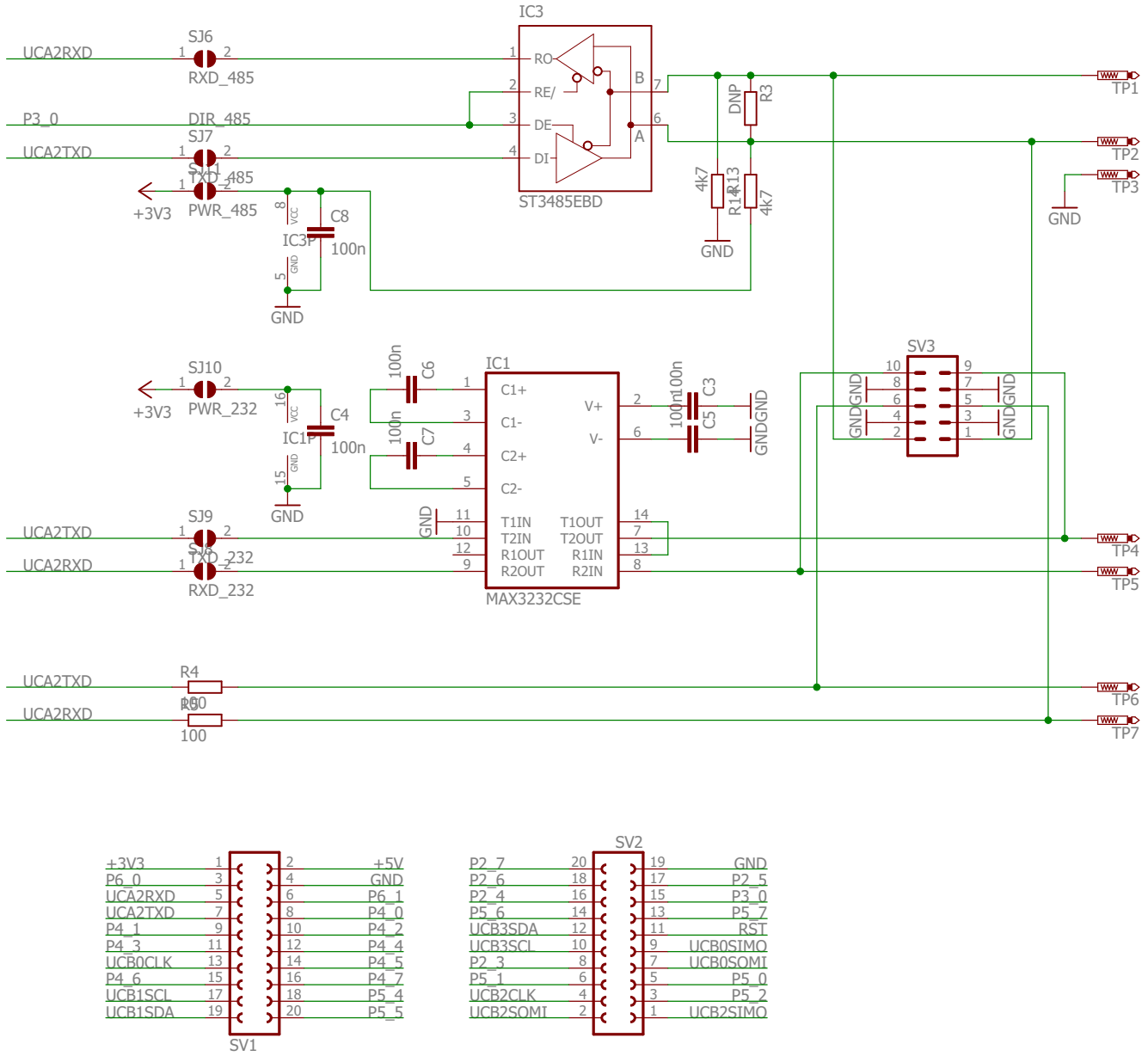
Electrical Diagrams

The following pages contain electrical diagrams of the daughterboards used in the bus tests:

- The diagram for the I^2C/CAN/USB main daughterboard: page 122.
- The diagram for the I^2C daughterboard (both generic and differential): page 123.
- The diagram for the RS485 daughterboard: page 124.


PWR_FLAG
GND
VCC
SN65HVD23xD
U2
CANH
CANL
EN/AB
Rs
D
GND
VCC
R
C7
0.1uF
MCP2515
U4
TXCAN
RXCAN
CLKOUT/SOF
TX0RTS
TX1RTS
TX2RTS
OSC2
OSC1
Vss
VDD
RESET
CS
SO
SI
SCK
INT
RX0BF
RX1BF
SOMI
SIMO
SCLK
10K
R2
C8
0.1uF
4K7
R9
CANCS
CANINT
J1
Vcc
P6.0_A15
P3.2_UCA2RXD
P3.3_UCA2TXD
P4.1_GPIO
P4.3_A10
P1.5_UCB0CLK
P4.6_GPIO
P6.5_UCB1SCL
P6.4_UCB1SDA
LaunchPad_J1
J2
GND
P2.5_TA0.2
P3.0_GPIO
P5.7_GPIO
RST
P1.6_UCB0SIMO
P1.7_UCB0SOMI
P5.0_GPIO
P5.2_GPIO
P3.6_GPIO
LaunchPad_J2
J3
VDD
GND
P6.1_A14
P4.0_A13
P4.2_A11
P4.4_A9
P4.5_A8
P4.7_A6
P5.4_A1
P5.5_A0
LaunchPad_J3
J4
P2.7_TA0.4
P2.6_TO.3
P2.4_TO.1
P5.6_T2.1
P6.6_TA2.3
P6.7_TA2.4
P2.3_GPIO
P5.1_GPIO
P3.5_GPIO
P3.7_GPIO
USBCS
LaunchPad_J4
PCA9514A
U1
READY
ENABLE
SCLOUT
SDAOUT
SCLIN
SDAIN
SCL
SDA
10K
R1
4K7
R7
4K7
R8
0.1uF
C9
Vcc
Y2
20MHz
C3
27pF
C4
27pF
MAX3421E
U3
VBCOMP
Vcc
VL
RES
SS
SCLK
MISO
MOSI
XI
XO
D+
D-
GPX
INT
GND
33R
R3
33R
R4
D+
D-
4K7
R5
4K7
R6
C5
1.0uF
C6
0.1uF
C10
0.1uF
USBCS
SCLK
SOMI
SIMO
Y1
12MHz
C1
27pF
C2
27pF
USBINT
USBGPX
CONN_01X01
P8
CONN_01X01
P9
P1
CANL
CANH
D+
D-
SCLOUT
SDAOUT
CONN_02X05
P2
P5
P3
P6
P4
P7
Sheet: /
File: BusNode.sch
Title:
Size: A4
Date:
Rev:
KiCad E.D.A. kicad 4.0.3-stable
Id: 1/1
122


VDD
+3V3
100n
C3
GND
S1
+5V
IC2
VDDA
VDDB
SCL
SDA
EN
DSCLP
DSCLM
DSDAP
DSDAM
GND
PCA9615DP
UCB1SCL
UCB1SDA
P4_0
R10
R11
R6
R7
600 (*)
R15
R14
120 (*)
TP6
TP7
TP1
TP2
TP8
SV3
P4_1
C1
R1
R2
R5
4k7
10k
IC1
VCC
SDAIN
SCLIN
ENABLE
READY
SDAOUT
SCLOUT
PCA9514ADP
R3
R4
4k7 (*)
TP5
TP4
TP3
TP9
TP10
SV1
P6_0
UCA2RXD
UCA2TXD
P4_1
P4_3
P1_5
P4_6
UCB1SCL
UCB1SDA
P6_1
P4_2
P4_4
P4_5
P4_7
P5_4
P5_5


IC3
ST3485EBD
SJ6
RXD_485
SJ7
TXD_485
SJ11
PWR_485
DIR_485
UCA2RXD
UCA2TXD
P3_0
+3V3
GND
C8
100n
IC3P
DNP
R3
4k7
R13
R14
TP1
TP2
TP3
SV3
IC1
MAX3232CSE
C1+
C1-
C2+
C2-
V+
V-
T1IN
T2IN
R1OUT
R2OUT
T1OUT
T2OUT
R1IN
R2IN
C3
C4
C5
C6
C7
IC1P
SJ10
PWR_232
SJ9
TXD_232
RXD_232
TP4
TP5
R4
100
TP6
TP7

SV1
+3V3
P6_0
UCA2RXD
UCA2TXD
P4_1
P4_3
UCB0CLK
P4_6
UCB1SCL
UCB1SDA
+5V
GND
P6_1
P4_0
P4_2
P4_4
P4_5
P4_7
P5_4
P5_5
SV2
P2_7
P2_6
P2_4
P5_6
UCB3SDA
UCB3SCL
P2_3
P5_1
UCB2CLK
UCB2SOMI
GND
P2_5
P3_0
P5_7
RST
UCB0SIMO
UCB0SOMI
P5_0
P5_2
UCB2SIMO

B

Flowchart Conventions

The flowcharts presented and used within this thesis follow common, but not necessarily standardised symbols. These symbols are shown in Figure B.1, with short explanations below.

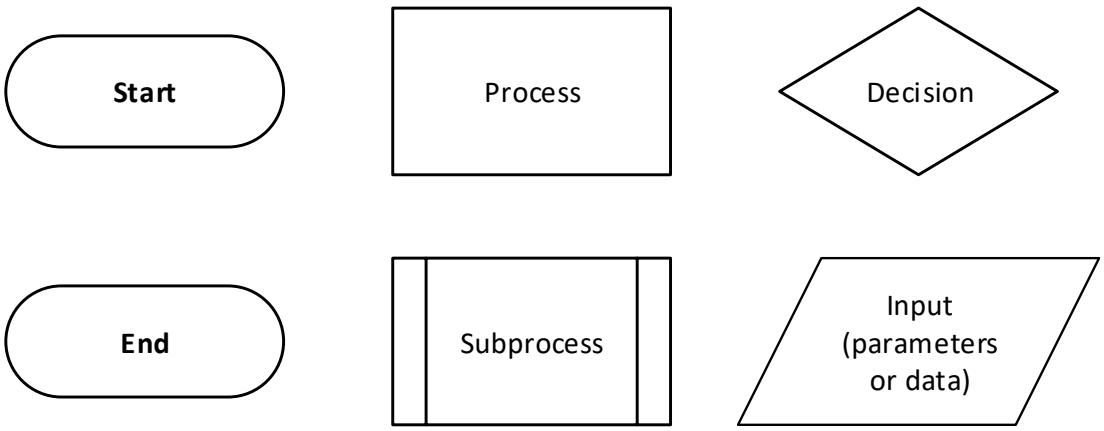


Figure B.1: Symbol definitions as used in the flowcharts included in this thesis paper

- **Start**: the start or entry point of a flowchart.
- **End**: the end point of a flowchart
- **Process**: a process, action or task performed. This usually takes specific inputs and results in outputs.
- **Subprocess**: similar to a standard process, but a subprocess refers to an external diagram detailing the actions. This is usually included in a secondary figure.
- **Decision**: similar to an *if*-statement available in most programming languages: a logical statement is checked, resulting in a choice between two or more options based on the statement.

- **Input**: this block describes some kind of (critical) input, such as parameters or data

C

Power Measurement Results

Table C.1: The measured power consumption of all TC buses in the various defined duty cycles

Bus	Idle Power (mW)			Once Per Second (mW)			Continuous (mW)		
	2 Nodes	5 Nodes	9 Nodes	2 Nodes	5 Nodes	9 Nodes	2 Nodes	5 Nodes	9 Nodes
I^2C	29.4	54.8	81.5	35.8	58.8	89.0	45.9	80.9	125.5
CAN	36.3	89.4	207.9	37.3	88.7	209.2	83.5	158.4	256.6
RS485	16.8	32.7	54.5	15.2	27.8	53.4	142.6	165.8	189.4
dI^2C	22.4	34.0	44.6	21.8	36.2	48.4	79.1	114.0	135.8
RS485 (no R)	7.6	25.1	47.2	6.4	19.3	44.8	21.1	45.7	70.7

Table C.2: Overview of PL bus power consumption

Bus	Idle Power (mW)	Active Power (mW)	Baud Rate (MHz)
CAN	41.91	91.476	1
RS485	16.2	181.5	1
SPI	1.7	4.6	1
USB	83.2	104.9	12

D

Reinventing Space Paper

As the Reinventing Space conference lacks a public index of the published papers and rather relies on direct linking, the paper submitted to the conference is included in this appendix starting on the next page.

The paper [20], titled *Design and Validation of an Innovative Data Bus Architecture for CubeSats*, and presented at the conference by the author of this thesis, describes the initial research into the data bus simulator and preliminary conclusions.

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.

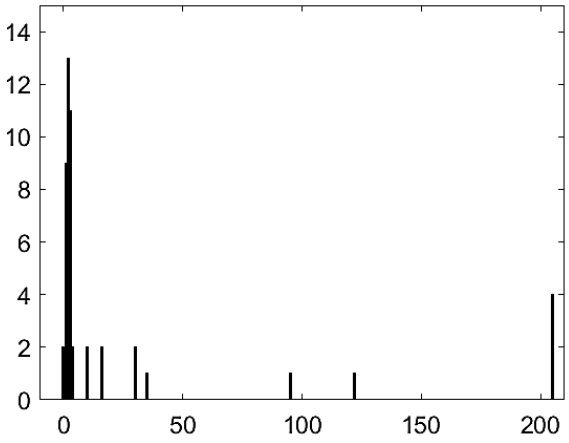


Figure 1. A histogram showing sizes of transactions in bytes during a normal housekeeping information polling cycle of Delfi-n3Xt.

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.

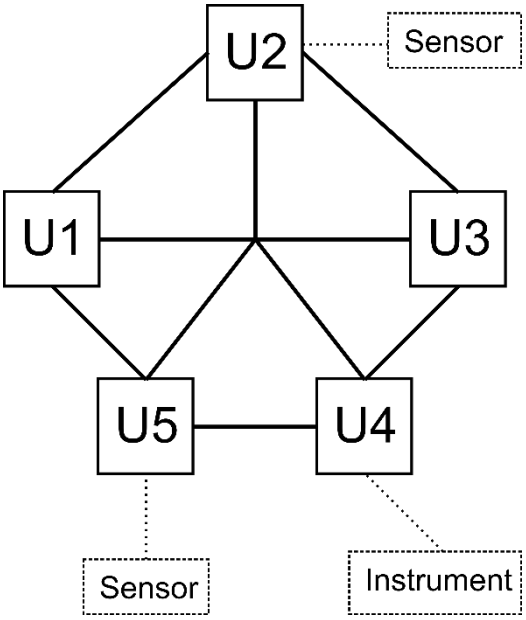


Figure 2. The TC bus reference case. Note that secondary buses connecting sensors and instruments (dashed) are not assumed to be part of the TC bus

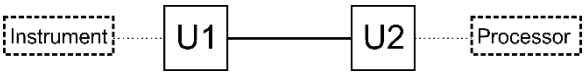


Figure 3. The PL bus reference case. As with the TC bus, the secondary nodes (dashed) are not part of the bus layout

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.

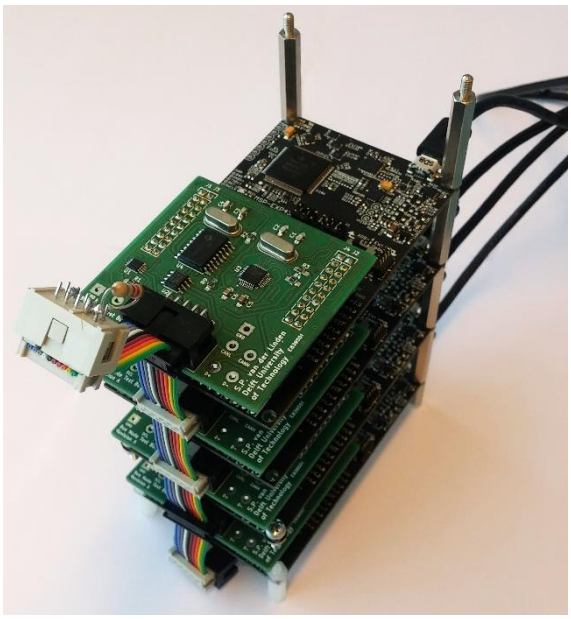


Figure 4. An example of the data bus testing setup showing four connected nodes over CAN, I^2C and USB (RS485 is on separate but similar boards).

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 I²C 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]
I²C	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

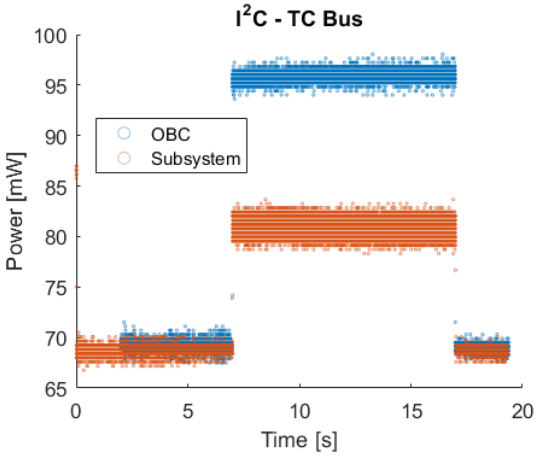


Figure 5. The raw measured power consumption of both the OBC and the generic subsystem when using I²C, including MSP432 and other supporting systems.

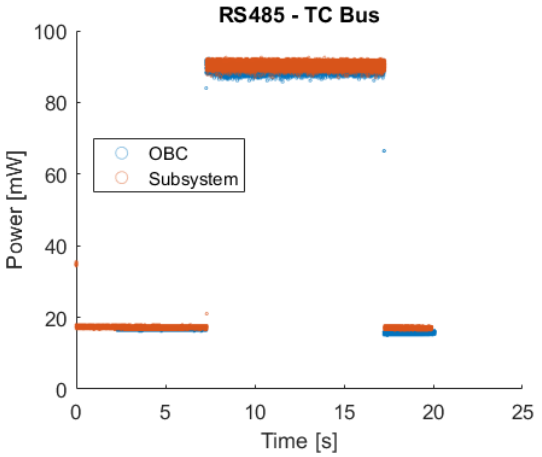


Figure 6. The power consumption of both the OBC and the generic subsystem when using RS485. The bus termination resistors are 2 x 120 Ω.

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

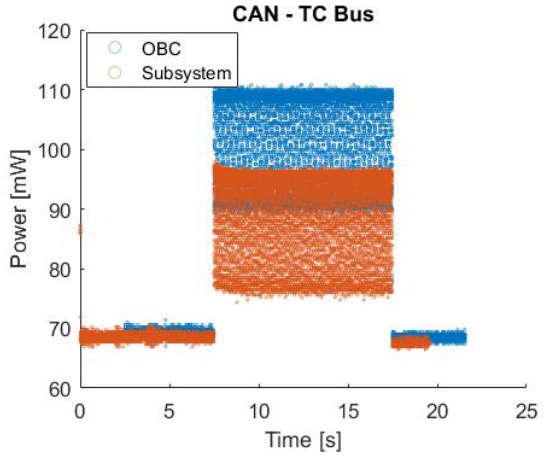


Figure 7. The power consumption of both the OBC and generic subsystem when using CAN. The bus termination resistors are 2 x 120 Ω.

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.

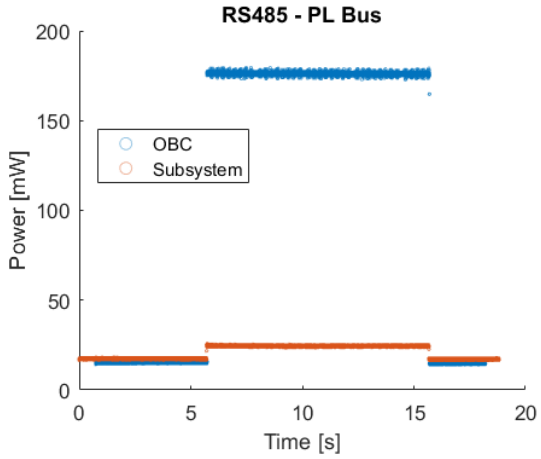


Figure 8. The power consumption of the OBC and generic subsystem when performing a large data transfer from the OBC to the subsystem over RS485.

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.

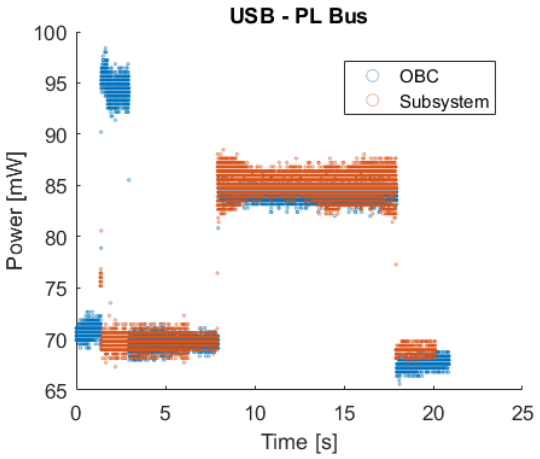


Figure 9. The power consumption of the OBC and generic subsystem when performing a large data transfer from the OBC to the subsystem over USB. Note the extra (peak) power required for enumeration, required once to establish the connection.

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).