University of Southern Queensland
Faculty of Health, Engineering & Sciences

Controller Area Network to Modbus Network Bridge to interface gas detection units with Building Management Systems

A dissertation submitted by

Matthew Quinton

in fulfilment of the requirements of

ENG4112 Research Project

towards the degree of

Bachelor of Electrical & Electronic Engineering

Submitted: October, 2015

Abstract

Building Management Systems (BMS') are computer systems designed to control systems inside buildings or other facilities. While BMS' are common, there is no one size fits all approach. Controller Area Network (CAN) is a communication protocol sometimes used within BMS'. Modbus is a very common industrial communications protocol. The two protocols are not directly compatible and need to be 'bridged' to communicate with each other.

Gas Detection Australia (GDA) design and manufacture gas detection equipment. They have a current and ongoing need to interface Modbus enabled equipment with CAN enabled equipment in client BMS'. This project is sponsored with the aim of producing a network bridge to translate between the two protocols. The specific Modbus variation implemented is Modbus ASCII master.

The design was based around the PIC 18F87K22 microprocessor. This was chosen to remain consistent with other GDA products. The communication interfaces were designed using integrated circuits that closely mimic the software development tools. This was a deliberate choice made to make software development simpler and to make it easier to translate source code to the finished product. A testing method was also created to allow the assessment of bridge performance.

Testing demonstrated proof of concept using the development board. Separate testing of RS-485 hardware suggests that the full hardware specification is valid. Stress tests were carried out and determined that the bridge could be expected to be capable of responding to four CAN messages per second. The testing was limited by issues relating to the inconsistent operation of the CAN interface.

University of Southern Queensland

Faculty of Health, Engineering & Sciences

ENG4111/2 *Research Project*

Limitations of Use

The Council of the University of Southern Queensland, its Faculty of Health, Engineering & Sciences, and the staff of the University of Southern Queensland, do not accept any responsibility for the truth, accuracy or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk, and not at the risk of the Council of the University of Southern Queensland, its Faculty of Health, Engineering & Sciences or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of the course pair entitled "Research Project" is to contribute to the overall education within the student's chosen degree program. This document, the associated hardware, software, drawings, and other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.

Dean

Faculty of Health, Engineering & Sciences

Certification of Dissertation

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.
I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

Matthew Quinton
0061024766

Acknowledgments

I would like to thank my academic supervisor, Mr Mark Phythian for his guidance and support. He often went above and beyond to help me achieve my project aims.

I would also like to thank my two professional supervisors from Gas Detection Australia, Mr James Boucher and Mr Chris Kelly. Without any of these people this project could not have taken place. I would especially like to recognise Mr Kelly, who was a continuous source of knowledge throughout the project. I would also like to thank Mr Trent Rutten from GDA for his assistance with hands on technical matters.

Finally I would like to thank my wife, Sam who was always there to listen and to help me through this project. Her patience and understanding were vital.

Matthew Quinton

Contents

List of Figures

List of Tables

Glossary of Terms

A - Ampere

AC - Alternating Current

ASCII - American Standard Code for Information Exchange

CAN - Controller Area Network

CR - Carriage Return (ASCII character)

DC - Direct Current

DFD - Data Flow Diagram

GDA - Gas Detection Australia Pty Ltd

GLCD - Graphical Liquid Crystal Display

IC - Integrated Circuit

IP - Ingress Protection Rating

LED - Light Emitting Diode

LF - Line Feed (ASCII Character)

OSI - Open Systems Interconnection

PCB - Printed Circuit Board

RMS - Root Mean Squared

SPI - Serial Peripheral Interface

UART - Universal Asynchronous Receiver/Transmitter

USB - Universal Serial Bus

V - Voltage/Volts

Messaging Rate - The time delay between CAN frames sent by the USB-CAN converter.

Chapter 1

Introduction

1.1 Motivation

GDA designs and manufactures electronic gas sensors and control units. Most of the sensors are industry standard 4-20 mA output and are hard wired back to control units. These control units are restricted by only having Modbus, relay switched Volt Free Contacts (VFC) and analogue current outputs to communicate with the customers' existing equipment. There are a growing number of customers requesting or already using Controlled Area Network (CAN) bus as part of their Building Management System (BMS), which GDA doesn't currently or plan to support as a direct hardware feature in their pre-existing products.

GDA wants to explore the development of a CAN bus to Modbus network bridge to allow customers to directly interface their BMS with GDA control units to simplify the installation and increase customer satisfaction. The current methods are to add a 4-20 mA current receiver and a CAN bus node with analogue inputs or a CAN bus node with digital inputs to receive the digital signals generated by the output relays on the control units. Depending upon the complexity of the customers' system and requirements this results in significant wiring, which could be replaced by a few wires used by the CAN bus and Modbus (Gas Detection Australia 2015).

1.2 Aim

GDA requires the development of a network bridge to interface their existing products with CAN enabled BMS'. The aim is to create a solution to this problem by developing a network bridge that is capable of interfacing with both CAN enabled BMS' and Modbus enabled gas detection controllers. The key goal is facilitating the delivery of information in a bidirectional manner between the two separate systems.

All variations of Modbus will be explored, however the chosen protocol for this bridge is Modbus ASCII.

1.3 Project Objectives

The objectives of this project are to design and develop a CAN bus to Modbus network bridge. The design should be of sufficient quality that it can be used as the base of a commercial product. It is intended that any prior research or products conducted or developed for this task will be considered but the final design will be original where appropriate. The network bridge must be able to reconcile messages sent from Building Management Systems that operate a CAN bus architecture with that of the Modbus architecture based gas detection control units.

The Modbus interface must be implemented upon the RS-485 serial standard to be consistent with existing products. Ideally, the network bridge will be configurable between Modbus RTU and Modbus ASCII. Due to the lack of a RS-485 enabled Modbus ASCII slave device to test on, testing will be performed using the Universal Asynchronous Receiver Transmitter (UART) as the physical layer.

The network bridge is intended to be a standalone product; however, it may be expanded to become a module that may be added to other GDA products. The network bridge should be developed in line with GDA procedures and methods. Final hardware components are to be based upon Printed Circuit Board technology but can be developed and tested using other methods. Software development is to take place using existing tools and methods available at GDA.

As well as developing the network bridge, testing procedures and documentation are also required to ensure correct functionality of the network bridge. Secondary objectives are to create testing procedures for the network bridge and also to develop documentation on the operation and maintenance of the network bridge.

1.4 Overview of the Dissertation

This dissertation is organized as follows:

Chapter 2 explores the literature and considers existing products as well as requirements of the sponsoring organisation, Gas Detection Australia.

Chapter 3 details the design methodology as well as the design of the testing parameters for the bridge.

Chapter 4 includes the final design of the hardware and software for the network bridge.

Chapter 5 examines and discusses the results obtained and the challenges faced in the design.

Chapter 6 concludes the dissertation and suggests further work in the area of CAN bus to Modbus network bridges.

Chapter 2

Relevant Literature and Design Constraints

2.1 Chapter Overview

The bridge is to be designed for use with Building Management Systems. This will require background information on these systems. This is to give an understanding on how the expected use affects the requirements of the bridge.

To be able to develop a device that translates two separate protocols requires an understanding of those protocols. This chapter covers some of the basic principles of each protocol and this gives enough understanding of what is required to reconcile them. The key differences will be highlighted so that the potential solutions can be developed in later chapters.

This is further supported by undertaking a review of the literature on this topic as well as a review of existing commercial products in this space.

2.2 Building Management Systems

Building Management Systems (BMS'), sometimes referred to as Building Automation Systems (BAS) are computer systems that control and monitor various systems within buildings or facilities. Typical systems that are controlled include heating, ventilation and air-conditioning (HVAC) electrical distribution, lighting, safety equipment, security systems, and many more (So 2001). The functionality and capabilities of BMS' are beyond the scope of this project. An understanding of where the network bridge might fit into a BMS is the primary concern. As part of the monitoring aspects of BMS', digital and analogue equipment feeds signals to the BMS. This might include thermostats, sensors, pressure transducers or numerous other devices (So 2001). The proposed network bridge is a 'device' that relays gas levels from the field to the BMS. The term device is loose because in this case it is not a simple sensor but will involve the sensor, controller and the network bridge. Logically, the network bridge is a gas sensor CAN node within the BMS.

There are many different communication interfaces used within BMS'. CAN is just one of the options available when choosing BMS architectures. There is no guarantee that equipment from one vendor will be compatible with equipment from another, even if they implement comparable technology (Butzin, Golatowski, Niedermeier, Vicari & Wuchner 2014). As a result, there are no standard requirements or acquisition techniques for BMS'. The network layout of the BMS, the protocol or protocols used, the individual pieces of equipment used, and the configuration of the network all contribute to the way a BMS requests and acquires information. Since all BMS differ it is not possible to satisfy all requirements with one rigid solution. The aim is to follow the CAN standard as closely as possible and to give as much flexibility for implementation in different CAN enabled BMS'. Flexibility would come from configurable parameters for the CAN interface such as data rates and message types. It may also have flow on affects to the implementation of the Modbus interface. If possible, examination of a real world BMS would be ideal for testing of the network bridge to confirm that it operates correctly.

2.3 Controller Area Network

Controller Area Network (CAN) was introduced in the mid-1980s by Robert Bosch. The protocol was designed for distributed automotive control systems. In 1991 Bosch published the 2.0 standard containing two parts; part A as the standard for the 11-bit identifier version and part B as the standard for the 29-bit identifier version. In 1993 CAN was standardised internationally by the International Organisation for Standards (ISO) as ISO 11898 (Ferreira & Fonseca 2011). There are two ISO components to a CAN implementation. ISO 11898-1 defines the data link layer and physical signalling and one of the parts labelled 11898-X, where X is 2-6 and defines the physical layer for different applications (ISO 2003*a*). ISO 11898-2 is the most popular of these physical layer extensions in use today. Even with the ISO standardisation in place for over twenty years, CAN is often referred to as CAN 2.0A or 2.0B in reference to the 1991 Bosch standards.

2.3.1 General CAN Features

As mentioned, CAN is a two layer OSI model. ISO 11898-1 is required for CAN implementations but the system developer has the choice from the 11898-X parts. ISO 11898-2 is the most common selection and is the assumed standard for this project. It is outside the scope of the project to implement more than one of the 11898-X parts. Most of the information found about CAN is derived direct from these standards (ISO 2003*a*, ISO 2003*b*).

CAN is a multi-master system with carrier sense multiple access (CSMA). Each CAN node on the bus can transmit and receive messages. Collisions are handled using arbitration of the identifier in the message. CAN is a message orientated system. Message contents are defined rather than nodes or their addresses. A node will examine the contents of a frame to determine if it is the desired destination for the frame - if it is not it will discard the message. Every message has an identifier that is unique on the network and this defines the content and priority of the message. CAN has numerous in-built fault handling features including a cyclic redundancy check (CRC).

The physical wiring is a differential pair and CAN defines transmission rates up to 1Mbit per second over a transmission length of 30 meters (ISO 2003*a*). Bus lengths of up to 5000 meters are possible at a baud of 10kbits per second. The trade-off for greater speed is bus run length. CAN has grown to be more than a control system bus for vehicles and

is now used in multiple industries including industrial communications, medicine, robotics and general embedded applications. CAN has benefited from large production volumes and sound performance to emerge in other industries (Ferreira & Fonseca 2011).

2.3.2 Standards

Exhaustive exploration of the standards is not required. CAN controller chips and compiler library functions handle the intricacies of the protocol and their application will be the focus of this project. An understanding of some aspects is important in order to properly link CAN to Modbus. ISO 11898-1 defines the data link layer and the physical signalling aspects of CAN. It forms the base upon which other standards are built. The basic concepts of CAN, the architecture, Logical Link Control (LLC) sub-layer and Medium Access Control (MAC) sub-layer are specified. ISO 11898-2 is one of the three extensions to ISO 11898-1 (ISO 2003*a*). It defines the high-speed (1Mbit/s) medium access unit and outlines the medium dependant interface (MDI) features. Electrical characteristics of the CAN nodes and bus are defined.

ISO 11898-1 is fundamental to any CAN implementation as it specifies the underlying principles of CAN. From their implementations can vary with requirements. The International Organisation for Standards (ISO) defines four other standards to complement 11898-1. They are labelled Part 3 through Part 6. It would be possible to use another standard for this project; however ISO11898-2 is recognised as the most popular standard by Ferreira & Fonseca (2011) and will be used for this project. This should allow maximum compatibility. Higher level protocols have been developed over the years to fill in the gap where more sophisticated messaging is required. There are several industry common protocols as well as academic based protocols in existence today. CANopen, DeviceNet, J1939, FTT-CAN and TT-CAN are some of the more common CAN based higher level protocols (Ferreira & Fonseca 2011, Corrigan 2008*b*).

2.3.3 Frame Format

There are two standard frame formats for CAN; one for each of the two addressing modes. Figure 2.1 is the 11-bit identifier frame and Figure 2.2 is the extended 29-bit identifier frame. The fields in each frame are considered below.

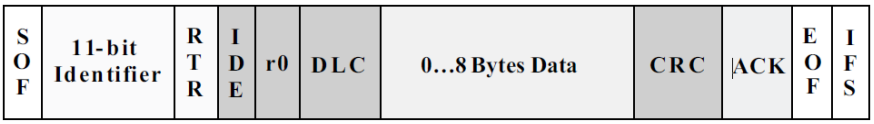


Figure 2.1: CAN Frame - 11bit Identifier (adapted from (Corrigan 2008*a*)).

The key difference between the two addressing modes is the increased identifier size.

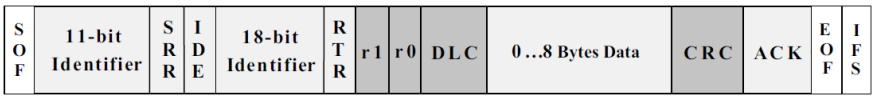


Figure 2.2: CAN Frame - 11bit Identifier (adapted from (Corrigan 2008*a*)).

Data Frames are the most common frame format and is of the form shown in Figure 2.1 and Figure 2.2.Remote frames are used when a node wants to request information from another node. The composition is similar to the data frame with two exceptions. The RTR bit in the arbitration field is marked as recessive to indicate a remote frame and there will be no data. An error frame is transmitted when a node detects an error in a message. The other nodes on the bus will respond by sending error frames of their own. There are safeguards in place to ensure that repeated transmissions of errors do not completely congest the bus. The overload frame is primarily used to provide a delay between messages in the event that a node is too busy. It has a similar format to the error frame.

An in depth discussion of the CAN protocol is beyond the scope of this project. Much of the implementation of CAN is handled by dedicated integrated circuits and software libraries. Appendix B contains further specifics about the CAN protocol.

2.4 Modbus

Modbus was created in 1979 by Modicon and was a propriety protocol designed for communication between Modicon Programmable Logic Controllers (PLCs) (de Sousa & Portugal 2011). Modbus was effectively the first ever industrial field bus protocol (Hui, Hao & Daogang 2013). Modicon openly published the protocol and it remains free to use, resulting in Modbus becoming the first de facto standard for industrial communications. Today it is maintained by Modbus-IDA, is still used widely in industry and continues to evolve (de Sousa & Portugal 2011).

2.4.1 General Modbus Features

Modbus is free and as such is one of the most widely used protocols across many industries. It is used with sensors and actuators that are network enabled. It is also used with more complex automation equipment such as RTUs, PLCs, human machine interfaces (HMIs) and supervisory control and data acquisition (SCADA) systems (de Sousa & Portugal 2011). Most of these implementations would be either on RS-485 or TCP networks. RS-232 Modbus implementations are limited to point-to-point connections, such as HMI to PLC (Thomas 2008).

Modbus is simple to implement and due to its wide use there is a lot of information freely available. There are also several different implementations, meaning you can pick the exact flavour that meets your needs. For example, Modbus RTU has smaller frames than Modbus ASCII so at the same baud is slightly faster (de Sousa & Portugal 2011). Modbus ASCII has deliberate delimiter, (colon character :) making frame reception an easier process than RTU.

Table 2.1: Modbus memory organisation

Name	**Memory range**	**Number of bits**	**Functionality**
Input registers	1(0)-65,536 (65,535)	16	Read
Holding registers	1(0)-65,536 (65,535)	16	Read and Write
Discrete inputs	1(0)-65,536 (65,535)	1	Read
Coils	1(0)-65,536 (65,535)	1	Read and Write

Modbus allows devices to read and write data to memory locations on another device. The device performing the reading and writing will have some processing ability (such as a PLC) but the device being written to and read does not need processing ability (such as a sensor), although it may have some. Modbus uses the client-server model. The device processing is the client and it makes requests to the server. The server never initiates an exchange. Clients can make multiple requests to multiple distinct servers one at a time. The specification does not allow for servers to have more than one client, but it is possible. Some devices acting as servers may have the ability to process simultaneous requests but it is generally done sequentially (de Sousa & Portugal 2011, Modbus Org 2006*c*).

The reading and writing functions are carried out in the memory of the server. The protocol defines the organisation of this memory but customisation is allowable (Modbus Org 2006*c*). Table 2.1 gives an overview of the specified organisation scheme. Note that the figures in the table are theoretical only. Many devices will have a memory range lower than specified here. Typically analogue inputs and outputs are handled through the registers and the digital inputs and outputs through the discrete inputs and coils.

To access the memory locations of table 2.1 requires specific querying from the Master to the Slave. This section gives an outline in brief according to the reference guide by Modicon, Inc. (1996). The are 24 separate function codes within the Modbus specification. Each function has a unique code that forms part of the Modbus data. For example, function '01' is 'Read Coil Status'. The function code used determines the format of the frame data. Reading is the key focus of the network bridge so that will be the focus here. The read functions (01 - 04) have very similar formats. The data component of the message begins with a slave address. The selected function code appears next. This is followed by the high and then low bytes of the address to be read. If the address fits in the low byte, the high will be zero. Likewise with the number of points to be read, which also has a high and low byte. They are the final two components respectively. An

example follows:

To read coils 5-10 (hexadecimal and begins at zero) from slave 11 the query is as follows:

Slave address:	0A
Function:	01
Address high:	00
Address low:	04
Number of points high:	00
Number of points low:	06

The response from the slave follows a similar format but is not the focus given the bridge is a master only. The query forms the data portion of the Modbus frames discussed next.

2.4.2 Frame Format

The implementations have similarities in their frame composition. They both include the function code and data fields.

Slave Address	Function Code	Data	CRC	
1 byte	1byte	0 up to 252 byte(s)	2 bytes	
			CRC Low	CRC Hi

Figure 2.3: Modbus RTU Frame (adapted from (Thomas 2008)).

Each implementation has an address and error checking field. The ASCII frame also has the explicit start bit (:) and end bits (carriage return and line feed). Figure 3 shows the format for an RTU frame and Figure 4 shows the format for an ASCII frame.

Start	Slave Address	Function Code	Data	LRC	End
1 char COLON	2 chars	2 chars	0 up to 252 byte(s)	2 chars	2 chars CR,LF

Figure 2.4: Modbus ASCII Frame (adapted from (Thomas 2008)).

As with CAN, there is a lot of detail involved with a Modbus protocol implementation. Integrated circuits, microprocessors and software libraries handle a lot of the details. As a result, exploration of the finer details is left to Appendix C.

2.4.3 RTU, ASCII and TCP

RTU and ASCII are asynchronous protocols that are implemented on serial channels, generally either EIA/TIA-232 (RS-232) or EIA/TIA-485 (RS-485). They are designed to be used on a physical bus with the slaves and master connected to the trunk. Both transfer a start bit as part of the frame. They differ in the way the frames are constructed and the way data is represented. RTU mode uses binary encoding and does not use delimiters (headers and trailers). ASCII encodes the data as ASCII characters and does use delimiters (Modbus Org 2006*b*). To compensate for the lack of delimiters while in RTU mode, timing limits between characters and frames are used. A no signal period of at least 3.5 character times between frames acts like the delimiter in RTU mode. In ASCII mode frames have a header and trailer. The : character is the delimiter in ASCII mode (de Sousa & Portugal 2011). Frame composition is considered later.

There are limitations on both serial variants of the protocol. Since the protocol was conceived in 1979 data types officially supported are limited to those available at the time (Modbus Org 2006*c*). It is possible to work around solutions to cater for modern formats such as using two 16 bit registers to store 32 bit values. There are also no security features inherent to the protocol.

Modbus TCP is also a direct byte representation protocol (de Sousa & Portugal 2011). It is not a serial protocol. The Modbus frames are sent through an existing TCP network and the connections are managed by the TCP protocol. There is a reserved port 502 for requests to establish Modbus TCP connections. Devices can be clients and servers simultaneously, send multiple requests simultaneously and do not have to wait for replies (Modbus Org 2006*a*). Modbus TCP frames only share the function code and data fields with the other two protocols, with the rest of the frame made up using the Modbus Application Protocol Header (MBAP) (de Sousa & Portugal 2011, Modbus Org 2006*a*). Theoretically the physical layer for Modbus TCP could be anything but is commonly Ethernet (Rinaldi 2010). From this point onwards, there is no reference to Modbus TCP. This section was added for completeness only.

2.5 Bridging the Protocols

Creating bridges or similar devices to link dissimilar protocols is not a new principle. Several papers have been written that explore the merging of CAN bus and Modbus (REF). There are also four companies that the writer is aware of that produce existing devices for this purpose.

2.5.1 Previous Works

A very brief paper by Guohuan, Hao & Wei (2009) identified the spread of fieldbus technology in many industries and applications as well as the incompatibility of the many existing standards and protocols. They briefly discuss the two protocols and then develop a design for the protocol conversion interface. The hardware design identifies five components inside a block diagram; MAX232 as the serial interface, a PIC18F458 as the microprocessor, two 6N137 optocouplers chips to isolate each bus driver from the processor, and an MCP2551 for the differential bus driving capabilities for CAN. The software design is only developed for the case of CAN as a master and Modbus as a slave. The brief description and a flowchart demonstrate the process at a high level of abstraction.

Wang, Zhang, Li & Ren (2013) studied the design of an adapter for Low-voltage distribution systems in Intelligent Electric Grids. Modbus is identified as the low level control network protocol of choice in this application. The major issue identified with the adapter is the dissimilar data sizes between the two protocols. Wang also refers to other studies that state issues with existing products. They claim that these devices do not properly take into account the data size problem. They may be able to transmit more than the 8 byte limit imposed by CAN but have issues with frame loss during communication. The articles referenced were not locatable to confirm the information, however, the frame size incompatibility is well known to the author. The rest of the paper outlines the selected hardware as well as the process required for developing the software.

The issue of conversion is addressed by using transparent transmission. When a Modbus frame is received and needs to be sent over CAN the adapter firsts completes the CRC check on the Modbus frame and then sends the whole remaining frame as the CAN data. The receiving CAN node must then parse the non-data fields of the Modbus frame. Wang claims this method reduces mapping errors when compared to stripping Modbus frames

down to just the data before being transmitted on the CAN bus.

The second major challenge addressed by software design is the issue of CAN data sizes. Wang applies techniques from the sub-packaging rule defined in Ethernet to deconstruct frames at the CAN transmitter and reconstruct at the CAN receiver. The final consideration is the delay in frame sending from the CAN interface. Wang noted that frames would sometimes be lost when transmitting at high speed. To address the issue a delay time between sending frames was calculated. This was based on the bit timing of the CAN bus based upon the selected baud rate along with the bus state, the priority of the sending node and the delay time of transmission. The authors claimed that the device was built, tested and met the requirements of the low-voltage distribution system.

The most recent study known to the author focused upon an adaptation layer for Modbus to CAN (Cena, Bertolotti, Hu & Valenzano 2014). The paper discussed how the traditional Modbus implementations' speed limitations are potential issues, the introduction of Modbus-TCP as a potential solution and the idea of layering Modbus on top of other protocols as another solution. The theme of this paper is different to the others in that it is not a converter but a variation of Modbus similar to that of Modbus-TCP. CAN was chosen as it is simple, fast and many microcontrollers have CAN controllers built in. The adaptation layer has significant differences to the proposed bridge in this project, however, the issue of large (greater than 8 bytes) Modbus data PDUs requiring fragmentation and reassembly for transmission on CAN is evident. The solution engaged to solve the fragmentation-reassembly issue was to change the composition of the CAN data frames. In the first transmission of a Modbus frame over CAN, 2 bits at the start of the data field are reserved for identifying the protocol and the length of the Modbus frame. Subsequent frames in the sequence gain back the two header bits for data since these fields are constant for any one Modbus message. The CAN CRC field is used in place of the Modbus CRC field which is parsed from the Modbus frame. While similarities are identifiable in this paper, the differences between a bridge or protocol converter and an adaptation layer are significant and much of the rest of Cena is largely inapplicable.

2.5.2 Lessons from Previous Works

There are common identifiable issues and themes within the above works. De-fragmenting and reconstitution of the Modbus PDUs for transmission over CAN is the major challenge.

There needs to be a method of reducing Modbus PDUs of up to 252 bytes into 8 bytes or less size for transmission. The solution posed in (Guohuan et al. 2009) offers no detail beyond that of a flowchart. Wang et al. (2013) propose sending the whole Modbus without the CRC field (as CAN has its own comparable field) and then parsing the unrequired fields at the CAN receiving end. This method is not suitable to this project as there is no guarantee of control over the other CAN nodes on the network. Similarly, the adaptation layer proposed by (Cena et al. 2014) require configuration of the receiving CAN nodes and given the different nature of the protocol bridge and protocol adaptation layers is not optimal for this project.

Wang's lesson regarding the mapping errors when simply stripping the data out of the Modbus frames is noted. The issue may be less pronounced on the buses the bridge will operate on. Testing may reveal if this is an issue.

There are two basic ways of achieving CAN transmissions using microcontrollers; using a microcontroller with embedded CAN controller and external CAN transceiver, or using a microcontroller with external CAN controller and external CAN transceiver. All papers reviewed suggest the former as the ideal path to take due to ease with implementations and hardware design. Using optocouplers to isolate the microcontroller from the transmission devices is also identified as desirable.

2.5.3 Existing Bridges and Available Products

Considering existing products is important. It allows appreciation for what has been achieved and may also help identify areas for improvement. It is highly unlikely that any of the products identified in this section would be suitable for the exact use intended for the network bridge. Ultimately, GDA would like a product developed internally. This allows for consistency in products and gives the opportunity to expand aspects of the network bridge into other products or applications if desirable.

Anybus Communicator from HMS

The Anybus Communicator is capable of interfacing CAN and Modbus RTU through a protocol conversion interface. This device has a configurable setup option for CAN

messaging that is configured through a graphical user interface (GUI). It supports both CAN standards 2.0A and 2.0B as well as serial communication standards RS-485 and RS-232. This product was designed to be used in fieldbus setups involving PLCs. It may require significant adaptation for it to function correctly in the BMS environment. It can only function as a Modbus slave. The obvious major drawback is the intended market for this device. It was designed for fieldbus use it may not interface as desired with BMSs expected for the network bridge (HMS 2015). In terms of general functionality the only significant improvement would be to add Modbus master capabilities.

CAN/Modbus Converters from ADF Web

ADF web offers several products in this category. There are Modbus TCP solutions that will not be considered and four different Modbus serial offerings. Electronically there are two variants; one is a Modbus master and the other is Modbus slave. The other two products are the same devices with extra galvanic isolation and a different enclosure. Each device has RS-232 and RS-485 connection and one CAN port. They support data rates on the CAN side up to the limit (1Mbps) and on the Modbus side up to the limit (115kbps) from the official standard. The devices are fully two directional messaging capable. Each device is only capable of being either a Modbus master or slave and not both (ADF web 2015). This may not be suitable for this task. Further, to use these devices requires the use of Windows based propriety software from ADF web called Compositor SW67011. The biggest improvement would be to have the configuration built into the device to remove the need for external software. Providing an interface to the device that allows on the fly configuration is consistent with practices at GDA. More flexibility with regards to Modbus configuration would also be desirable.

Modbus RTU to CAN converter from ICP DAS

This device is another protocol converter that supports CAN to Modbus RTU data transfer in both directions. It supports RS-485, RS-232 and RS-422 serial communication standards. One of its applications listed is building automation so it is likely useful in this task. It is configurable using proprietary software for the device. The device only supports Modbus RTU as a master and not as a slave. It does not support Modbus ASCII. The configuration requires software and cannot be done by interfacing with the device

directly (ICP DAS USA 2015). To be useful in this task, this device would ideally need support for Modbus ASCII and the ability to configure as a Modbus slave. Configuration would be better achieved with an interface on the device itself.

There are several products all of which have their own strengths and weaknesses. None of these products are suitable for this exact project, but they do provide some proof of concept for the objectives outlined. Further, each device has limitations and this provides opportunities for the network bridge.

2.6 Constraints

As a sponsored project the network bridge design must be carried out within the constraints of the host organisation and within their requirements. Below is a description of those constraints and requirements.

As a potential commercial product, the bridge must be financially viable. When selecting components for use in the bridge they must be approved by GDA from a cost perspective. All components in this project have successfully passed through this process. If a component to be specified is similar to one used on another existing GDA product, it may be used on the network bridge to help reduce stocking costs and make use of existing expertise within the organisation.

The development environment used in this project was chosen as it is the current environment employed by GDA. This includes the development board, microprocessor, programming language and compiler. Finally, GDA's general design guidelines were followed regarding component choice and software design methodology where possible.

There is also a design considerations to be stated. Modbus frames can theoretically contain data up to 508 bytes in length. In the specific application of the bridge within GDA, this is unlikely to be the case. Discussions with GDA engineers suggest it is reasonable to assume that data requirements will not exceed 16 bytes.

2.7 Chapter Summary

This chapter was about the fundamental concepts and considerations for the design of the network bridge. It contains some fundamental information about the CAN and Modbus protocols. It also explored previous work that had been carried out in this area including the investigation of several existing commercial products. Finally, constraint set out as a consequence of undertaking a sponsored project were considered.

Chapter 3

Design Methodology

3.1 Chapter Overview

The basic principle is to keep it as simple as possible whilst meeting the requirements previously outlined. Hardware design is more restrictive as it is not as simple to create basic design to build on. The final hardware design must be of commercial quality. Therefore the design of the hardware will be conducted with the aim of producing one finished product which can then be produced as a printed circuit board (PCB).

The hardware design methodology involves smaller increments leading to a complete design. Each logical section, such as the Modbus interface, is broken down and designed separately. Existing library functions will be used from the chosen compiler to prove concepts where possible. Only once all concepts are proven will more advanced techniques be use to refine the design.

Software design will be performed logically first, and then in code.

3.2 Equipment

The following is a run-down of the equipment used during this project. This list is not exhaustive but highlights the key tools used.

As previously mentioned the development environment used was consistent with GDA practices. The development board used was an EasyPIC Pro v7 from MikroElektronika. A PIC 18F87K22 microprocessor was used in development and in final design as discussed later. The EasyPIC Pro board supports UART over universal serial bus (USB) natively and this was used for serial communications. A 3.3V CANSpi click board was used in conjunction with the development board to provide the CAN interface. During testing, two 5V CANSpi modules were also used. To view activity on the buses, a UNIT UTD2102CEL 100MHz digital storage oscilloscope was used.

Windows 7 based personal computers (PC) were used to run the development environment and simulation software for Modbus and CAN. USB cables were used to connect the USB UART module of the development board to the PC. During initial testing, the two 5V CANSpi modules were used. Later on a USB-CAN Converter from ICP-DAS, Inc was used to simulate a CAN node. This was done so that testing was not performed on two devices created during the project. Using external devices that are rated as per the communication standards validates testing results. The converter translates CAN messages to USB so they may be interacted with via PC. 15cm male-male breadboarding wires were used as the CAN transmission lines. Short wires were used to avoid reflections and other physical effects on the line.

The software package used to complete software as MikroC Pro for PIC. Tha programming language used was 'C'. The terminal function of MikroC Pro was used for initial serial communications testing as well as troubleshooting. To simulate a Modbus slave, the freely available software program 'Modbus Slave' was used. This program is free for a trial period of 30 days. This program uses COM ports to access the UART over USB. The USB-CAN converter shipped with basic software that allows sending and receiving of CAN messages and was used in this project.

3.3 Hardware Design

Some of the traditional design choices are forgone here for the benefits of using tried and tested equipment already available at GDA. Technical functionality is the key requirement and should be observed first. Where possible, components used in existing GDA products may be used if they meet the specification and it provides benefit to GDA through reduced stocking costs or product familiarity. This is true regardless of whether the component selected is not the ideal selection based on other criteria. As a general rule, where two or more components meet all the requirements laid out, the cheapest component shall be selected.

The other major design consideration is alignment with the software development environment. The circuitry found on the EasyPic Pro and add-on boards has proven functionality. It makes sense to utilise these circuits as a template for two reasons; they are proven to work and take away much of the risk of trying new methods, and they make the transfer of software from the development environment to the final product much simpler. Minor changes to microprocessor pin configurations will be required[1].

Data-sheets for all devices mentioned by name in this section can be found in appendix G.

3.3.1 Microprocessor

The microprocessor is the central cog in the design. It affects the choice of all other components. Selection of the device was largely driven by GDA. The desire was to use a processor already deployed in other products and suitable for future commercial products. GDA favours the PIC range of processors from MikroElektronika. The device still needed to conform to the following requirements:

1. Ability to process CAN messages at data rates up to 1 Mbits per second and serial messages up to 19.2 kbits per second.
2. Sufficient memory to buffer communications data and serve other functionalities defined in ancillary circuitry later in the chapter.
3. Sufficient pins to service all functions of the bridge.

[1] This is due to the predefined pin configuration of the development board.

4. Be available to use on the intended development equipment and with the selected compiler and design environment.

Items one and two are critical in that they will affect the performance of the bridge's main functions. Item four is required so that the software design will integrate with the hardware design correctly. Selectable PIC processors for this task far exceed the requirements of item one. Below is a conservative calculation of the required memory space:

Let's assume that we need to buffer at any one time ten Modbus messages and ten CAN messages. The non-data portions of the CAN frames will be filtered by the hardware for us. We may need to buffer the entire Modbus frames. CAN frame requirements would be:

$$Mem_{can} = 8bytes \times 10 = 80bytes \tag{3.1}$$

This is fairly insignificant. Now we consider the modbus requirements. From figure 2.4, we see a the maximum size of a complete Modbus ASCII frame is 513 bytes. So:

$$Mem_{asc} = 513bytes \times 10 = 5,130bytes \tag{3.2}$$

We need at least 5.2 kbytes of memory for buffering alone using this conservative approach.

The microprocessor selected was a PIC 18F87K22 which operates up to 64MHz, supports up to 128k bytes program memory, has 80 pins and is a supported chip on the selected development board. The processor is to be operated at 5 volts.

3.3.2 CAN Interface

Using a PIC processor as the central processor for the network bridge gives two clear options for implementing the CAN interface; select a PIC processor with an in-built CAN module or select a PIC processor with an SPI module and utilise an external CAN-SPI integrated circuit to allow the processor and CAN transceiver to communicate. The process of selecting the CAN interface logically relies upon the selection of the processor.

In order to make the individual modules of the network bridge more portable, it was decided to use an external CAN device. The transceiver chip selected needed to support the full ISO standards (usually referred to as CAN 2.0A and 2.0B in datasheets). It also has to operate on the same supply voltage as the processor. In total, two CAN devices are required; a CAN-SPI conversion integrate circuit (IC) and a CAN transceiver IC.

The devices selected were the MCP2515 and MCP2551 from Microchip. The SPI module of the processor was used to communicate with the MCP2551 transceiver via the MCP2515 chip. Supporting circuitry was selected using the 3.3V CANSpi click board as a guide.

3.3.3 Serial Interface

To enable Modbus communication the bridge requires a serial interface. The physical layer chosen for this project was RS-485[2]. This implementation required a transceiver to convert signals from the processor to the correct levels for transmission on the serial bus.

The devices selected were DS485M from Texas Instruments. This is a product that had been used on a previous GDA product. Supporting circuitry was selected using the RS485 Click board from MikroElektronika as a guide.

3.3.4 Power Supply

There is no guarantee of the supply voltage from the Building Management System. GDA's experience suggests that a 24V AC, 24V DC or lower voltage supply is likely. Dealing with AC and assuming that the 24V AC supply occasionally spikes at 30V RMS. The equation for the max voltage the power circuitry needs to with stand is:

$$V_{max} = 30V * \sqrt{2} = 42.43V \tag{3.3}$$

The bridge's functional circuitry is mostly digital so the supply requires rectification if the source is AC. A bridge rectifier diode IC was selected to perform this task. The device

[2]UART is used for the software development stage and the final solution can be modified to make use of RS-485.

chosen was the DF04S, which accepts input bridge voltages up to 280V and an average rectified forward current of 1.5A.

The next stage of the power supply design in the switching regulator. A linear power supply was not considered. A 5V linear regulator would use too much power and generate excessive heat considering the supply voltage could be as high as 24V. The regulator IC chosen was the LM2671 from Texas Instruments. This device can tolerate input voltages up to 45V, can supply 5V in a fixed output configuration and operates at 96% efficiency.

The final power consideration was analogue supply. The touch panel feedback is an analogue signal connected to the processor. The analogue supply pin of the processor was connected. Low pass filtering components were added to reduce high frequency noise on the signals.

3.3.5 Isolation

Communication transmission lines from external sources could be the source of voltage or current spikes. It is therefore important to isolate the communication interfaces from the rest of the board circuitry for protection. In order to achieve this it was necessary to isolate power supplies for those interfaces as well.

Isolation of the CAN interface was achieved by placing optical isolation ICs between the MCP2515 CAN-SPI chip and the MCP2551 transceiver chip. Similarly, isolation of the Modbus interfaces was achieved by placing optical isolation ICs between the processor and the DS485M transceiver chip. The isolation ICs used were HCPL-2601S.

Optically isolated DC-DC regulators were used to isolate the power supplies for the communications on the board. They take the supply 5V and isolate it for CAN and Modbus to create separate power supplies. The isolated DC-DC converters ICs used were ROE-0505S.

3.3.6 Ancillary Circuitry

In the event of a power failure that is localised to the area containing the bridge, a battery backup system was added to allow the bridge to communicate the failure. The battery

was designed to take over the supply of power when the standard supply dropped below a certain level. A test point was added and connected to a processor pin for determination of this level. This allows the bridge to self diagnose a power fault back to the BMS so that it may be attended to. A battery test point was also added to provide battery level feedback.

Five LEDs were included in the design for use as indicators. A power LED is lit when the bridge power supply is active. There are two LEDs for Modbus communications; one for transmission and one for reception. They are intended to illuminate when the RS-485 transceiver is operating. Similarly, the CAN communication has one LED indicating bus activity[3]. The final LED is for error reporting. With the exception of the power LED, all LEDs are connected to the processor and their operation is to be configured in software.

The bridge design includes a Graphical Liquid Crystal Display (GLCD) and touch screen combination for use as a human-machine interface. The design used is adapted directly from the EasyPic Pro v7 development board but is arranged on different pins. To specify the limiting resistor for the supply voltage, the development board current was measured using a multi-meter. The value of the resistor used on the bridge was adjusted to reflect the 6V supply.

The operation of these devices are to be configured in software.

Numerous capacitors were used in the design of the bridge. Many were selected by following recommendations from chip manufactures for use with ICs or the processor such as decoupling capacitors. Some were also included due to the potential for long track lengths on the PCB and the high frequencies encountered. Much of this part of the design was carried out under guidance from GDA engineers rather than driven by research carried out by the writer.

[3]One LED is sufficient since CAN is a differential system where both lines are used for transmission and receival.

3.3.7 Printed Circtuit Board

PCB arrangement was focussed on having the functionalities of the board in separate areas. This was compounded by the use of isolation. Two layers were used; top and bottom. The processor was located centrally and its decoupling capacitors and crystal nearby. The power supply was placed at the bottom of the board. The CAN interface and its isolated power supply were located to one side of the board. Likewise with the Modbus circuitry to the other side. The communication headers were placed opposite each other for aesthetics. Keep-out sections were placed around each of the communications interfaces containing the isolated components. Thick tracks were run through the middle of the board for high current transfer. Three main ground planes were placed; a standard ground plane and separate isolated ground planes for each of the communication interfaces. Two smaller ground planes were also placed around each of the oscillators. The GLCD was placed on the top layer as it is intended to be embedded in the lid of the encapsulation. The LEDs are also located on the top layer. All headers and the contrast potentiometer for the GLCD were located on the bottom layer since it is the accessible side of the board post installation[4].

Ideally all tracks would have been placed manually but time restraints did not allow this. Some of the important track laying was carried out by hand but some use of Altium's auto-route feature were utilised.

[4]Assuming that the bridge is mounted as currently planned. GDA will be responsible for this.

3.4 Software Design

The key software design principle is modularity. Ideally, GDA will be able to use separate modules of this bridge in other projects. Where feasible, code was written in separate 'C' files and combined using a header file inside a project. The design of the software was approached on two levels; an overview of the whole bridge functionality illustrated using data flows and then by considering the protocol implementations.

Logically, the bridge is both a CAN node and a Modbus ASCII master. CAN message handling and Modbus message handling are established separately first. Interrupt sources were to be used to receive messages and forward the data out of the other interface.

To visualise the process and also to aid development, a data flow diagrams (DFD) was created to show the bridge operation. The figure can be found in chapter 4.

3.4.1 CAN Interface

MikroC Pro comes with a CANSpi library containing a suite of functions for the initialisation and operation of a CANSpi module attached to a PIC processor. The initial design idea was to use an SPI interrupt to start the processing of a CAN message using the library functions. CAN reception and transmission would be handled inside interrupt routines. During testing this was found not to be possible and will be discussed in the relevant chapter. A minor change was required. The interrupt routines were configured to set flags that were processed in the main loop. Again during testing this was proven ineffective for the CAN communications. Eventually the entire CAN functionality was moved to the main function of the project. All initialisation and message handling was processed there. For proof of concept, arbitrary configurations were used on the bridge. Testing of the CAN interface was undertaken using the USB-CAN converter.

3.4.2 Modbus Interface

Modbus functionality had to be added to the bridge from scratch. Some existing Modbus RTU code was available from GDA as a guide. During development, all serial communications were performed using the UART. The MikroC software includes a library for

RS-485 which is overlaid onto the UART. The bridge was designed using UART and the RS-485 implementation would only occur on the finished bridge (PCB).

The bridge is not a typical Modbus master. The logic for operation is not configured on the bridge itself. This would be performed on the BMS CAN node. The bridge needs only to communicate with a Modbus slave and report errors such as time-out and is not required to poll directly. Facilitation of the communications was divided into five key functions:

1. Serial Transmit
2. Modbus Transmit
3. Serial Receive
4. Modbus Receive
5. Longitudinal Redundancy Check (LRC)

The serial functions were developed first. To test their functionality the terminal window in MikroC was used to send and receive UART communications. Once the serial functions worked as desired, the Modbus functionality was created. Transmission was achieved first by creating a function that took the data and encased it in a Modbus frame. Then the Modbus receive functionality was created by coding a reversing of that process. Initially, both were tested using the terminal. The LRC was the last function created as it required working send and receive functions. Hand calculations were performed on messages sent to ensure that the LRC was accurate.

3.4.3 Ancillary Functions

This section is devoted to the functionality not strictly related to one protocol or the other. These are functions which support the translation between the two. CAN data is treated as numeric but Modbus ASCII encodes as ASCII. An ASCII to Hex and Hex to ASCII set of functions were created to allow the transition between the two encoding schemes. A simple example follows:

Consider the hexadecimal number 0x2D. To send this as Modbus ASCII the bridge needs

to convert each character into two characters, '2' and 'D'. '2' will be encoded as 0x32 and 'D' will be encoded as 0x44. These values are taken from the ASCII table.

Functions were created to automate this process as well as its inverse. In order to separate two digit hexadecimal numbers into a pair of single digit hexadecimal numbers, two functions were written.

The literature shows the incompatibility in data sizes between the two protocols. A method was required to successfully decompose and reconstitute Modbus frames with data lengths of greater than 8 bytes into two or more CAN frames. Due to reasons discussed in later sections, this was not implemented. The designed principle for this is discussed in the next chapter.

3.4.4 Main

The original design of the main function was to initialise the communications modules and provide GLCD functionality and LED indications. During the design it was noted that the interrupt implementation of CAN proposed would not work. Interrupt flag setting was implemented and this requires the response to them in main code. Code was implemented to check the status of flags and respond with appropriate communication operations. While the extended functionalities of the bridge such as the GLCD and LEDs were designed for in hardware, their implementation in software falls out of the scope of this project. Inclusion in hardware was given to allow this design to be built on in future.

3.5 Proposed Testing Methodology

The bridge's functionality needs to be verified. A testing procedure was developed with the aim of determining the capabilities of the bridge, its limitations and where it can be improved.

This section is an outline of the *intended* testing procedure.

To be satisfied that the CAN and Modbus functionalities work as desired they must be able to communicate with devices that conform with their standards. A series of tests were designed to assess the performance of the bridge. All CAN related testing will be conducted using the USB-CAN converter to simulate the BMS CAN node. Modbusslave will be used to simulate the Modbus ASCII slave. The first test is to verify the concept. The bridge should:

1. Receive a CAN frame
2. Extract the data
3. Put the data in a Modbus frame and send it
4. Receive a Modbus frame back and extract the new data
5. Put the new data into a CAN frame and send it back to the originating CAN node.

The device needs to successfully complete this test as the first stage. The parameters of each protocol such as speed, filters, flags etcetera can be selected arbitrarily. The test should be performed separately using the 11bit and the 29bit CAN identifiers to ensure compliance with both standards (2.0A and 2.0B).

After successful completion of the first test the bridge should use the same testing procedure to confirm that the bridge operates fully over the range of possible communication speeds. This means up to 1Mbit on the CAN side and 19.2kbit on the Modbus side. Each extreme from both protocols should be tested with each other, i.e. max CAN speed with minimum Modbus speed etc. These tests will confirm whether the bridge can function over the baud ranges given in the protocol standards.

The next suite of tests determine the performance of the bridge over that range. These are to be facilitated using stress tests on the bridge. An amount of frames will be sent

from the USB-CAN converter dependant on the BMS messaging rate. The faster the rate, the more messages that will be sent. A count of the number of frames that the BMS CAN nodes send and receive will be kept. The success ratio is the number of frames received divided by the number of frames sent. These results should be tabulated as percentages and graphed for analysis.

Testing should be performed at differing baud rates to determine how the bridge performs over the range of specified operation speeds. Four sets of parameters are to be used:

1. Maximum CAN baud and maximum Modbus baud
2. Minimum CAN baud and minimum Modbus baud;
3. Maximum CAN baud and minimum Modbus baud;
4. Minimum CAN baud and maximum Modbus baud

Performing the testing at the extremes will allow the performance of the bridge to be specifiable within the boundary of those results.

3.5.1 RS485 Confirmation

There was no RS-485 enabled Modbus ASCII slave device to test with during the project. The hardware design was adapted from the 3.3V RS-485 Click board from MikroElektronika (MikroeElektronika 2012*b*). In order to prove the hardware concept of the serial interface, testing was carried out using a RS-485 click board and an oscilloscope. The RS-485 was connected to a bread board and terminated using a 120 ohm resistor. A successful test would be achieved by the observation of an RS-485 master packet on the bus. A photograph of the testing set-up with the bread board, terminating resistor and oscilloscope can be found in appendix F.

3.5.2 Physical Cable Testing

In an attempt to rule out physical level issues, testing was carried out with the CAN interface and different transmission lines. The majority of testing was completed using fifteen centimetre male-male breadboarding wires. Tests were done using one meter and

ten meter lengths of twisted pair (as per the CAN physical specification). Measurements were taken with an oscilloscope. If transmission lines were the cause of CAN transmission errors, these tests would show evidence of abnormal signal levels. This may include overshoot, jitter, ringing etcetera.

3.6 Chapter Summary

This chapter covered all aspects of the methodology. This included the key equipment used, the hardware and software design approaches as well as the proposed testing methods.

Chapter 4

Design of the Network Bridge

4.1 Chapter Overview

This chapter details the hardware and software design elements of the network bridge. It is worth re-iterating that several hardware components included in the design do not have accompanying software elements. The bridge is to be expanded and developed further as a commercial product and this requires a complete hardware base. Figure 4.1 shows the overall logical design of the bridge. It is to operate as a CAN node and Modbus ASCII master simultaneously.

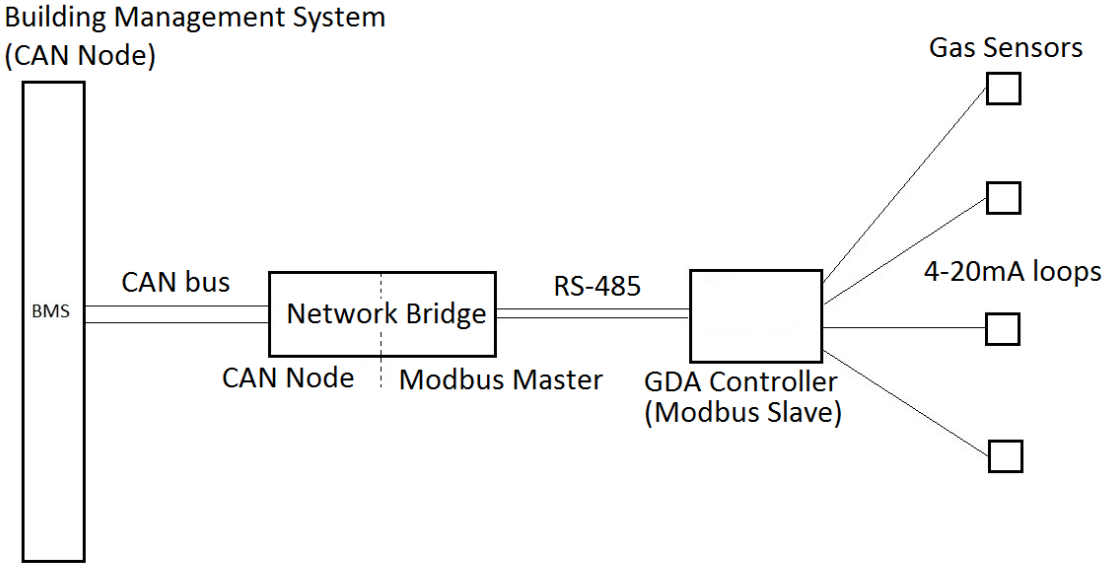


Figure 4.1: Network Bridge Logical Overview

4.2 Hardware Design

The hardware design process had two distinct phases; the schematic design and the PCB layout. Each functional subsection of the bridge is detailed using focussed versions of the schematic found in appendix E. This design will consider the major components of the design and will not focus on simpler features such as current limiting resistors or decoupling capacitors.

The final section will include the PCB layout of the bridge and justify the positioning of components on the board. All ICs included in this design have selected extracts from their data-sheets located in appendix G.

4.2.1 Microprocessor

The oscillator chosen is a 14.7456MHz crystal. The rationale for this selection was to use a crystal with a frequency wholly divisible by 9600. This allows the UART to generate the required bauds without error. Using crystals with non-divisible frequencies can cause issues that will be discussed in a later chapter. The timing of the CAN-SPI interface is set by the crystal connected to the MCP2515 IC and is not affected by this selection.

The programming header P2 is connected to SPI pins on the PIC to allow programming. The MCLR pin is arranged with resistors R2, R1 and capacitor C35 to give a time constant matched to the start-up time of the processor.

The processor data-sheet recommends 100nF capacitors for decoupling between the positive voltage pins and their associated grounded pins. Additional 10uF capacitors were also included due to the high power requirements of the optical isolation employed on the bridge. The value used was recommended by the GDA engineers.

4.2.2 CAN Interface

The MCP2515 IC is interfaced to the SPI1 pins of the processor. The reset and chip select (CS) pins are connected to pins on port B of the processor and must be configured in software. The oscillator used is a 10MHz crystal as recommended on the MCP2515 data-sheet. The two data pins, RX and TX are connected to the MCP2551 CAN transceiver

via the HCPI-2601S optical isolators. The isolated supply voltage and ground can also be observed. This will be discussed in a later section. The header P4 is for connection of the transmission lines. They are labelled CAN high and low and should only be connected as such. The CAN interface is shown in figure 4.2.

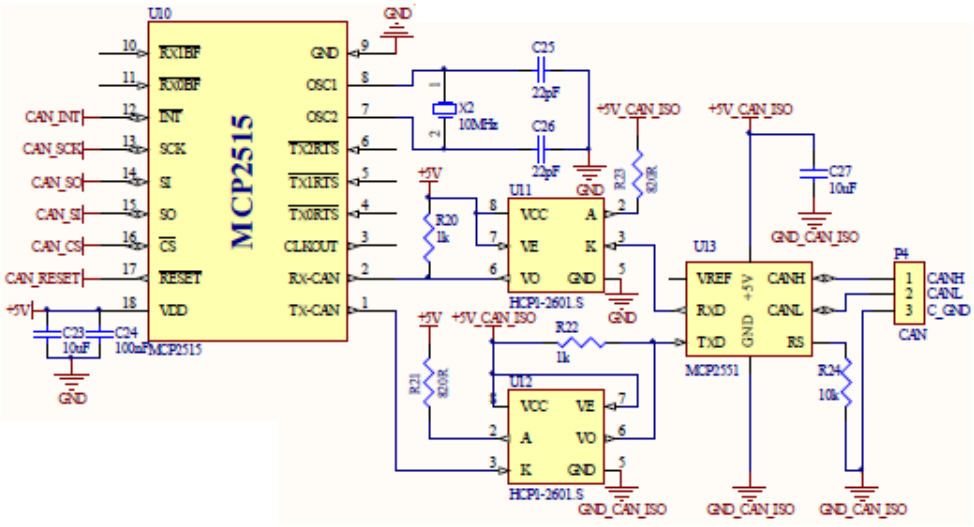


Figure 4.2: CAN Communications Hardware Interface

4.2.3 Serial Interface

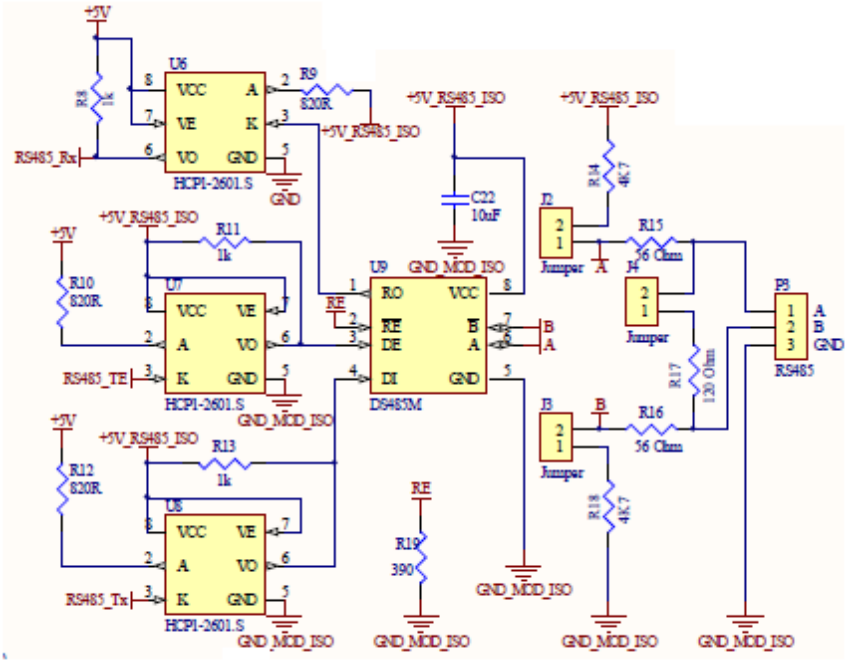


Figure 4.3: Serial Communications Hardware Interface

The RS-485 interface is shown in figure 4.3. There are 3 pins from the DS485M that

communicate with the processor through the HCPI-2601S optical isolation chips. These are the transmit enable pin and the two data pins (receive [RO] and transmit [DE]). The receive enable pin is held low (on) via R19, a 390 ohm resistor. The intent is to use this as a troubleshooting feature for transmission as the internal circuity of the DS485M will mean transmitted messages will echo on the reception input. The jumpers J2 and J3 can be used to stop the transmission lines floating, which can create noise on the bus[1]. Jumper J4 is used to enable a termination resistor when the bridge will be acting as the first or last node on the bus. The header P3 is the transmission line connection for the RS-485 bus.

4.2.4 Power Supply

Power to the bridge is supplied into P1. The bridge expects up to 24V that could be DC or AC. Rectification is provided by the DF04S bridge rectifier. The rectified voltage is passed to the LM2671 switching regulator. This device is configured to produce 6V. This will allow the expected 1V drop across the diodes D4-D6 to produce 5V. All components selected were specified using the data-sheet as a guide.

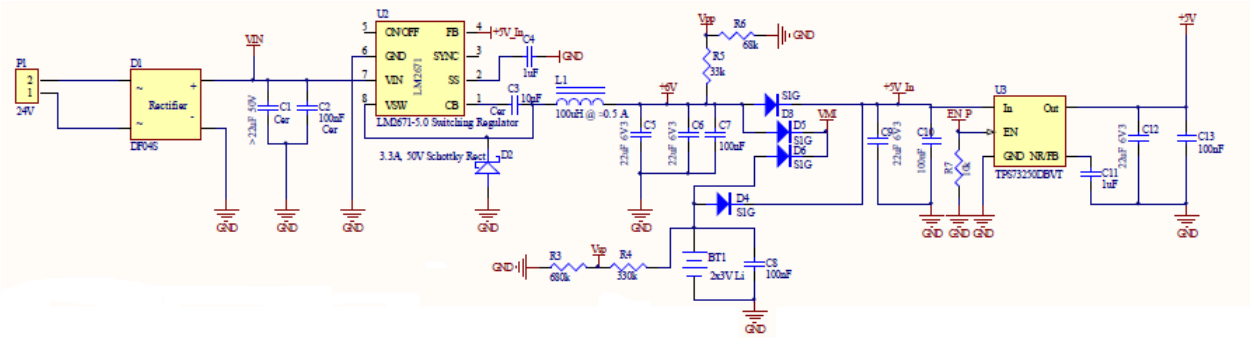


Figure 4.4: Power Supply Hardware

The processor is supplied from net VMI (the cathode end of D5 and D6). The communications interfaces are supplied from the output side of the voltage regulator U3. This device has a rated drop-out of 40mV at 250mA essentially making the output voltage the same as the input. The enable pin is connected to pin 27 on the processor. The rationale is that this gives control of the power supply to the non-processor components of the bridge to the processor. During a power outage, non-critical components can be disabled and the processor put in sleep mode to prolong the life of the battery. The advantage of

[1]The calculation for this resistor value in the datasheet recommends a lower value than used. The RS485 Click for MikroElectronica and a previous GDA device with a RS-485 interface use the value here.

this strategy is that the processor will not require a reset and allows a diagnostic message to be sent from the bridge to the BMS informing of a power outage. The diodes D3, D4, D5 and D6 are required to prevent current draw into the regulator U2 or charging of the battery.

The analogue supply is required for operation of the touch panel. The inductor L2 is for low pass filtering of the analogue supply.

The isolated DC-DC converters U4 and U5 are for the CAN and Modbus isolated power supplies.

4.2.5 Isolation

There are two isolation approaches present; the power isolation and the data isolation. The data isolation refers to the HCP1-2601S optical isolators between the CAN and RS-485 transceivers and the MCP2515 IC or processor respectively. These devices protect the data lines of the communications interfaces. The isolated supplies provide power to the communication circuitry on the outside edge side of the optical isolators.

4.2.6 Ancillary Circuitry

LED D11 is used to indicate power to the bridge. D7 is an indicator of serial reception. D8 is an indicator of serial transmission. D9 indicates when there is activity on the CAN bus. D10 is designed for use in fault indication. All LEDs bar D11 are software configured. The communication indicator diodes are to be powered when the relevant activity occurs on the buses. D10 is flexible and may be configured to meet user needs.

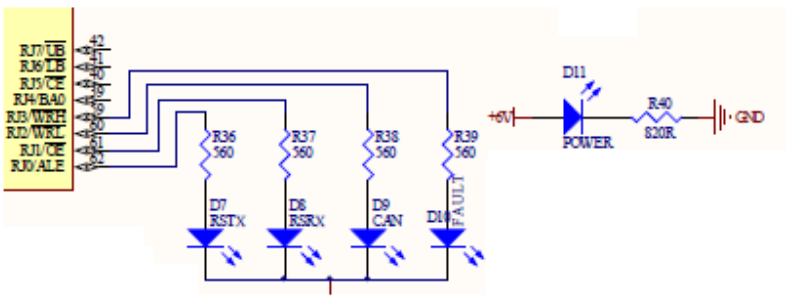


Figure 4.5: Indication LEDs

The GLCD and touch panel combination were used during design and testing. The implementation used for the bridge is a near replica of the circuitry found on the development board. Component R25 has been adjusted to account for the change in input voltage. The calculations are below:

A 1.4V drop was measured across the resistor. Its value was 20ohms so

$$I_{res} = 1.4V/20\Omega = 7mA \tag{4.1}$$

The voltage drop will change as a result of the great supply voltage (6V) on the bridge giving a drop of

$$V_{new} = 1.4 + 1 = 2.4V \tag{4.2}$$

Now we can see the new value of resistance required to maintain a current similar to the original

$$R_{new} = 2.4V/0.07A = 34.29\Omega \tag{4.3}$$

The next highest E12 series is selected at 39 Ohms. This is resistor R25.

4.2.7 Printed Circuit Board

The final PCB top and bottom layer layouts are found in appendix D. Appendix D also contains 3D visualisations of the PCB. Both the top and bottom layer have been utilised for ease of placement and aesthetics. Simple placement rules have been observed. The layout is broken up into functional elements; power supply, CAN interface, Modbus interface and the processor. The two communication interfaces are placed on either side of the board. Both of these sections are optically isolated, have their own ground planes and are bounded by keep-out sections. The power supply circuitry is located on the bottom of the board. Isolated power is kept to the edges of the board and away from non-isolated power. The processor is located centrally. Both crystal oscillators have ground planes surrounding them for shielding. The inductor L1 was placed safely away from the feedback of the regulator. Decoupling capacitors were placed near their respective ICs.

The processor was aligned so that the relevant pins for communications are easily accessible for the relevant communications interface. Communication headers are aligned for aesthetics. LEDs for the communications are located on the same side of the board as the interface. All circuitry that needs to be accessible by a technician is located on

the bottom layer. The intention is to place the PCB in an ingress protection (IP) rated enclosure with the GLCD and LEDs embedded in the lid and sealed. As such, only the bottom layer of the PCB will be normally accessible. The design of this enclosure is not within the scope of the project but was a design consideration.

4.3 Software Design

This section details the code written for the network bridge. All code referred to here can be located in appendix D. The modular approach to this software requires a header to tie the code together. The header file ProjectDevBoard contains the function prototypes for the project.

The first element of the design is the data flow diagram. The DFD shown in figure 4.6 is the ideal DFD in that it assumes that serial peripheral interface (SPI) interrupts function correctly. The tested bridge does not have this functionality and it is replaced with a flag check within a continuous loop. The other processes, data stores and flows are not affected by this change.

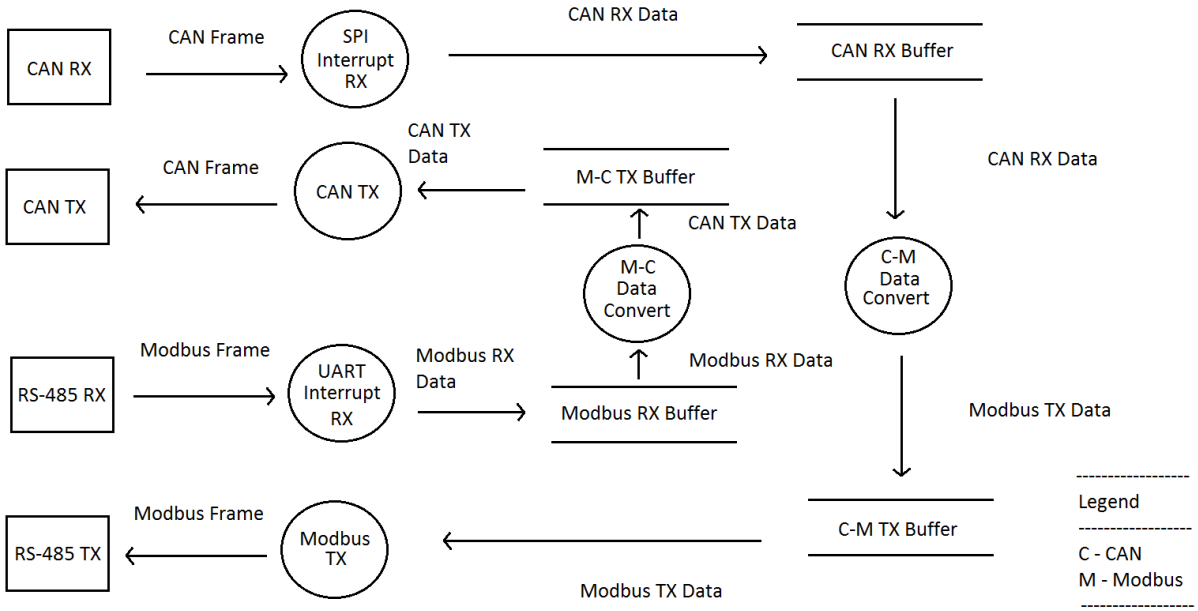


Figure 4.6: Bridge Data Flow Diagram

Modbus functionality is confined to the C file 'Modbus'. There are several variables within the global scope of the Modbus C file. The most important being the send and receive buffers and frame length storage. There are six modbus functions:

1. modbus_init - Initialisation of the Modbus interface.
2. TX_serial - Transmission of serial data.
3. TX_modbus - Creation of Modbus transmission frames.
4. RX_serial - Reception of serial data.
5. RX_modbus - Checking of LRC from serial receptions and data extraction.
6. LRC - Calculates the longitudinal redundancy check of a frame.

The initialisation function sets up the UART with the correct parameters for Modbus communications. This includes the baud rate, synchronisation (asynchronous for Modbus) and number of data bits[2].

The serial transmission function should only be called after the Modbus transmission function to ensure data integrity. As a result, the only function call is located inside the Modbus transmit function. It receives the size of the data to send as an argument. The send frame is put into the serial transmit register TXREG1, byte by byte until the full frame is sent.

The Modbus transmission function receives a buffer of data and the buffer length as arguments. An ASCII colon (hexadecimal 0x3A) is placed in position zero of the global send frame. The contents of the argument data frame are then copied to the global send frame. The LRC function is called and the result stored. Since the LRC will be a two byte result it has to be split into two ASCII characters. This is performed by the functions split_first and split_second. These two characters and the new line characters (ASCII CR and LF) are appended to the end of the global send frame. The function then calls TX_serial to send the Modbus frame.

[2]Note that parity checking was not implemented due to reasons discussed later. However, instead of the 7 data bits with a parity check required by the standard, 8 data bits with no parity was used since the 8th bit is always 0 due to the hexadecimal value of the ASCII characters used.

The first step in the serial reception function is to read the current byte off the UART. If a colon has previously been received the variable named Go will be in set mode (equal to one) and the function will extract the contents of the UART until the line feed character is received. If Go is not set (equal to zero) and a colon is received, Go will be set and the reception process can continue. Any other characters received in this circumstance will be ignored. A count of the number of bytes in the received Modbus frame is maintained for use in the RX_modbus function.

The Modbus receive function copies the serial frame into a holding array ignoring the colon character. The LRC of the received message is then checked and if correct, the data only portion of the frame is placed back in the reception array[3].

CAN functionality was entirely produced from the CANSpi library within the MikroC environment. All CANSpi functions were called directly from the main program, ProjectDevBoard. CAN flags for initialisation, sending and receiving are all cleared before being set using predefined bitwise arguments. The CANspi interface is then initialised and configured using the flag, mask and filter combinations. A brief description of the operation of masks and filters can be found in appendix B. The module is then set in operation mode where the two functions CANSPIWrite and CANSPIRead can be used to send and receive CAN frames respectively. The interrupt service routine for the UART reception is also found in the main program. It simply checks for the correct flag, calls the serial reception function, sets a flag for the main loop and clears the interrupt flag.

The other functions written for this project include the ancillary functions for hexadecimal to and from ASCII conversions. hex_ascii and ascii_hex makes use of the sequential ordering of ASCII characters for conversion between formats. The split_first and split_second functions makes use of simple bitwise operations to extract the first four or second four bits of a byte. This is used to take a two character byte and convert it into two, one character bytes as required when converting to a Modbus ASCII frame. The comb_digits functions performs the inverse of this. This is used when converting Modbus ASCII frames back to standard data to be sent on the CAN bus.

The last significant design component is the decomposition and reconstitution of multiple CAN frames. For the scope of this project it was previously deemed unnecessary since

[3]The size of array is passed as an argument and as such the calling function will know the size of the data frame.

Modbus data was unlikely to exceed 16 bytes. The design is included for completeness.

CAN data frames only have room for 8 bytes of data and Modbus ASCII could potentially contain 504 bytes of data[4]. The Modbus data is ASCII encoded. Eight bytes of data can only carry 4 bytes of information since a byte requires two characters. We only need more than one CAN frame if the data size is greater than 16 bytes. To account for this eventuality, the first data byte will carry information that determines the amount of data related to the current frame. If the data to be received is less than or equal to 14 bytes, the first byte is zero. If there is more data to follow the first byte is marked 0xFF. On every subsequent frame the process is repeated until all data is sent and the first byte of a CAN frame is zero.

4.4 Chapter Summary

This chapter detailed the hardware and software design of the bridge. A logical overview was presented. Hardware components were specified and justified. The software design was presented as data flows first, then as descriptions of code.

[4]While this is possible it is not anticipated in the intended application of the bridge. There may be a need to send more than one CAN frame worth of data.

Chapter 5

Results and Discussion

5.1 Results

The intention was to assess the bridge performance as outlined in the testing methodology. Due to time constraints and GDA ordering cycles, a finished PCB was not attainable during the project span, therefore, the tests were undertaken using a development board. As the design of the PCB is consistent with the hardware components used on the development board test platform[1], it is reasonable to assume that the PCB would perform similarly.

Given no RS-485 enabled Modbus ASCII devices were accessible for testing, UART over USB was used to give access to PC based simulation software as an alternative[2].

As a result of the testing set-up described, separate verification of the RS-485 functionality of the PCB was required.

5.1.1 Proof of bridge concept

As stated in the methodology, the first test is to verify the concept of the bridge by

1. Receiving a CAN frame
2. Extracting the data
3. Placing the data in a Modbus frame and sending it to the slave
4. Receiving the Modbus slaves response frame and extracting the new data
5. Placing the new data into a CAN frame and sending it back to the originating CAN node.

Figure 5.1 demonstrates the bridge concept by successfully completing all of the above steps. In the figure, the movement of data is easily traceable between the CAN and Modbus interfaces. Frames are being sent from the BMS simulating CAN node (right window, top view) to the bridge, which in turn sends the data (01 02 03) on to the Modbus slave. This is viewed as an 'RX' line on the Modbus serial traffic viewer (left

[1]The test platform refers to the development board and CANSpi click board.
[2]This is Modbusslave program.

window). The 'TX' in the same window is the slave response with data '00 00 08 00 00 01' in ASCII format. The bridge then extracts this data from the Modbus frame and sends it on the CAN bus to the BMS simulating CAN node (right window, bottom view). It is important to note that the data returned by the Modbus slave simulator is not a valid response to the transmitted frame. The simulator sends the same response to any transmitted message.

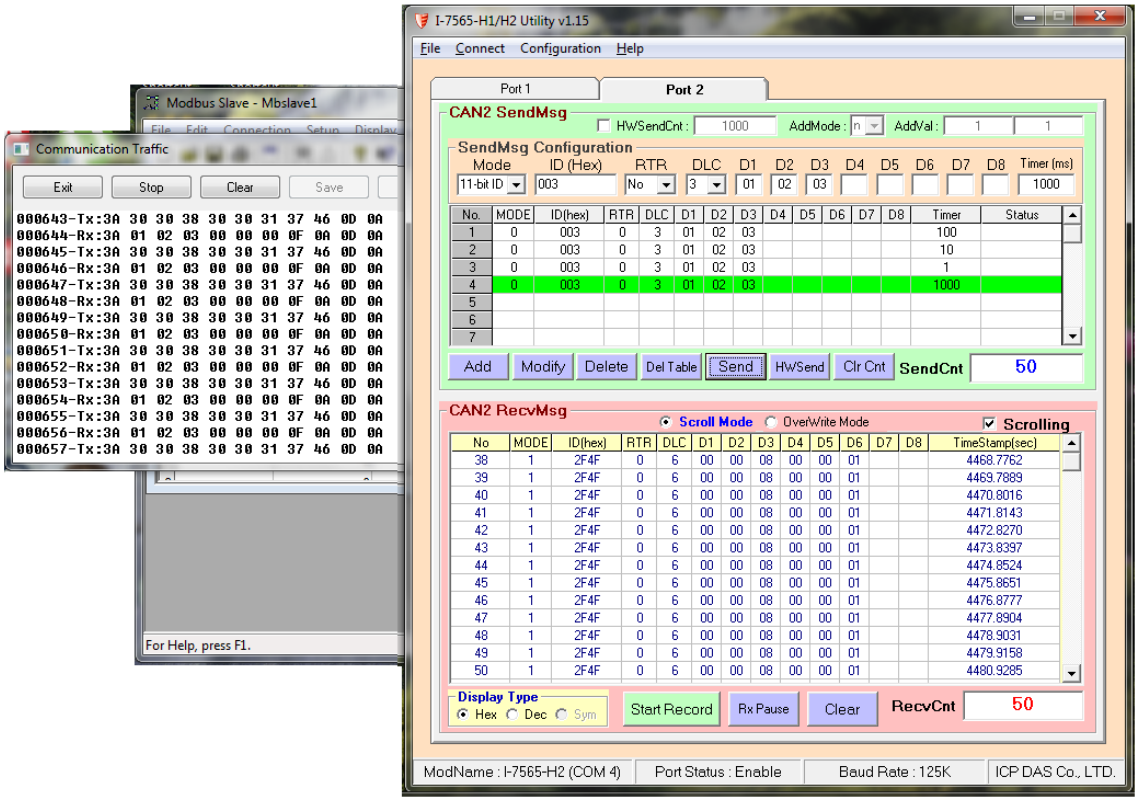


Figure 5.1: Proof of bridge concept - complete transmission

Table 5.1: Results for frame bridge frame loss - CAN baud 125kbps and Modbus baud 9.6kbps

CAN message rate (ms)	Frames Sent	Frames Received	Success %	Loss %
1000	100	100	100%	0%
750	100	100	100%	0%
500	100	100	100%	0%
250	200	200	100%	0%
200	400	379	95%	5%
150	400	285	71%	29%
125	400	238	60%	40%
100	400	191	48%	52%
50	800	191	24%	76%
25	800	96	12%	88%
10	1600	77	5%	95%

5.1.2 Bridge stress testing

Table 5.1 contains the results for stress testing of the bridge. It shows the percentage of successful transmission and frame loss at different CAN messaging rates. CAN messaging rates is the amount of time between CAN frame transmission from the BMS. In the testing, the BMS is simulated using the USB-CAN converter.

Figure 5.2 is a graphical representation of the tabulated data above. What is evident is that the bridge will successfully maintain 100% frame transmission when the CAN messaging rate is slower than approximately 220ms at CAN baud 125kbps and Modbus baud 9.6kbps. This leads to the conclusion that it takes the bridge about 220ms to process a complete message transaction. The loss is linear, and proportional to the CAN messaging rate. For example, when the messaging rate is 10ms, for every 77 frames we receive, over 1500 frames are lost. This is a roughly 1:20 ratio consistent with the linear loss conclusion.

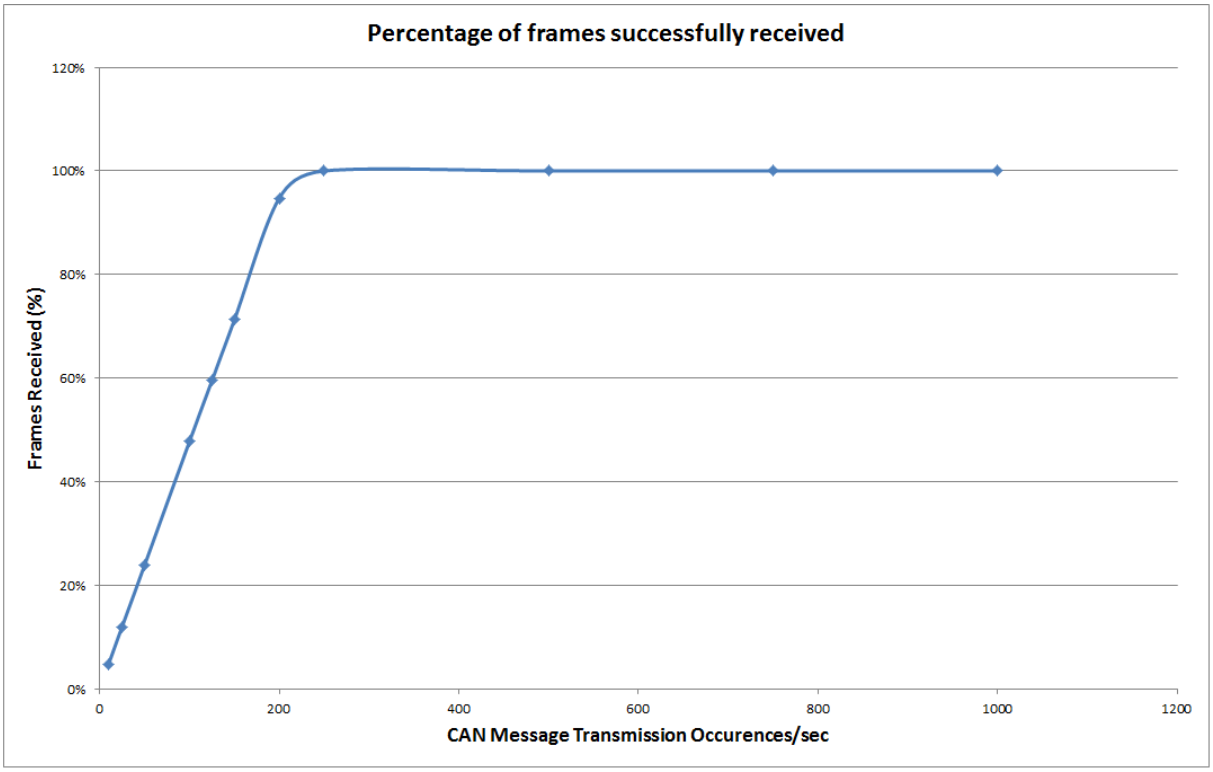


Figure 5.2: Percentage of frames received

5.1.3 Proof of RS-485 concept

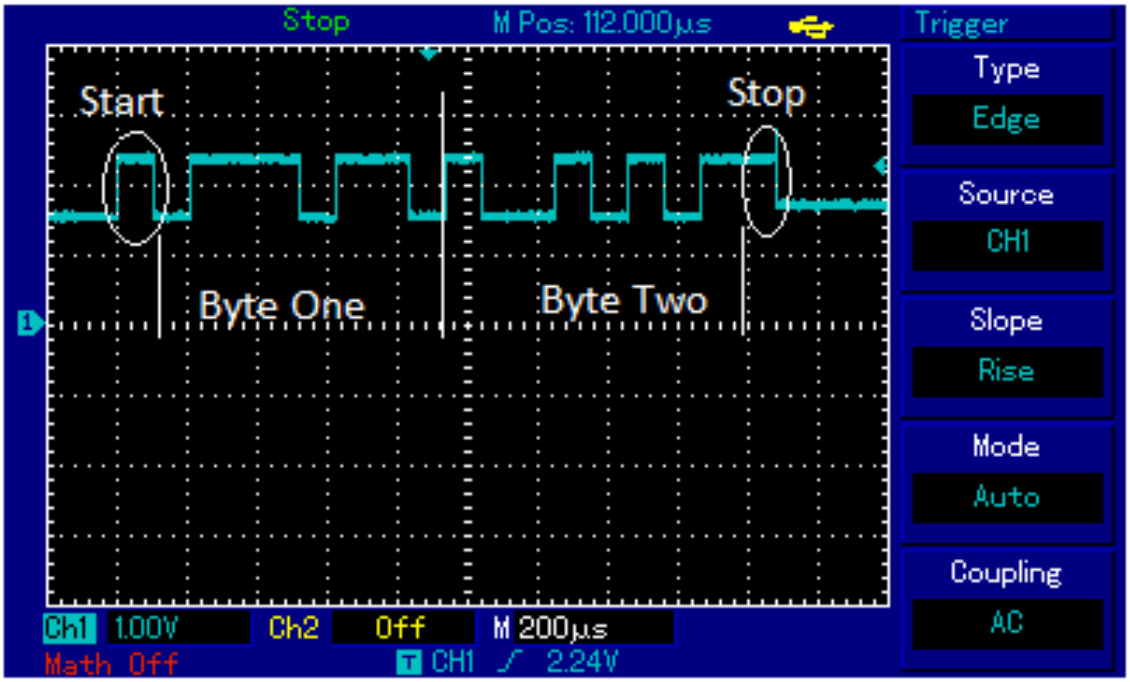


Figure 5.3: RS-485 master packet using 3.3V click board

As mentioned previously we needed to prove the concept of the RS-485 communication hardware of the bridge hardware design. Testing was carried out as per the methodology using a development board, RS-485 click board and an oscilloscope.

Figure 5.3 demonstrates this test. The start and stop bits are circled. This transmission has two eight bit bytes as marked. The least significant bit is sent first so bytes are read right to left. The value of byte one is 0x6D and the value of byte two is 0xA9. This is consistent with the transmitted message and successfully demonstrates RS-485 communications are possible with the designed hardware.

5.1.4 Physical Cable Testing

Figure 5.4 is the CAN bus as seen when the USB-CAN converter is reporting a transmission error. The transmission line is the breadboarding wire. This frame repeated periodically on the bus.

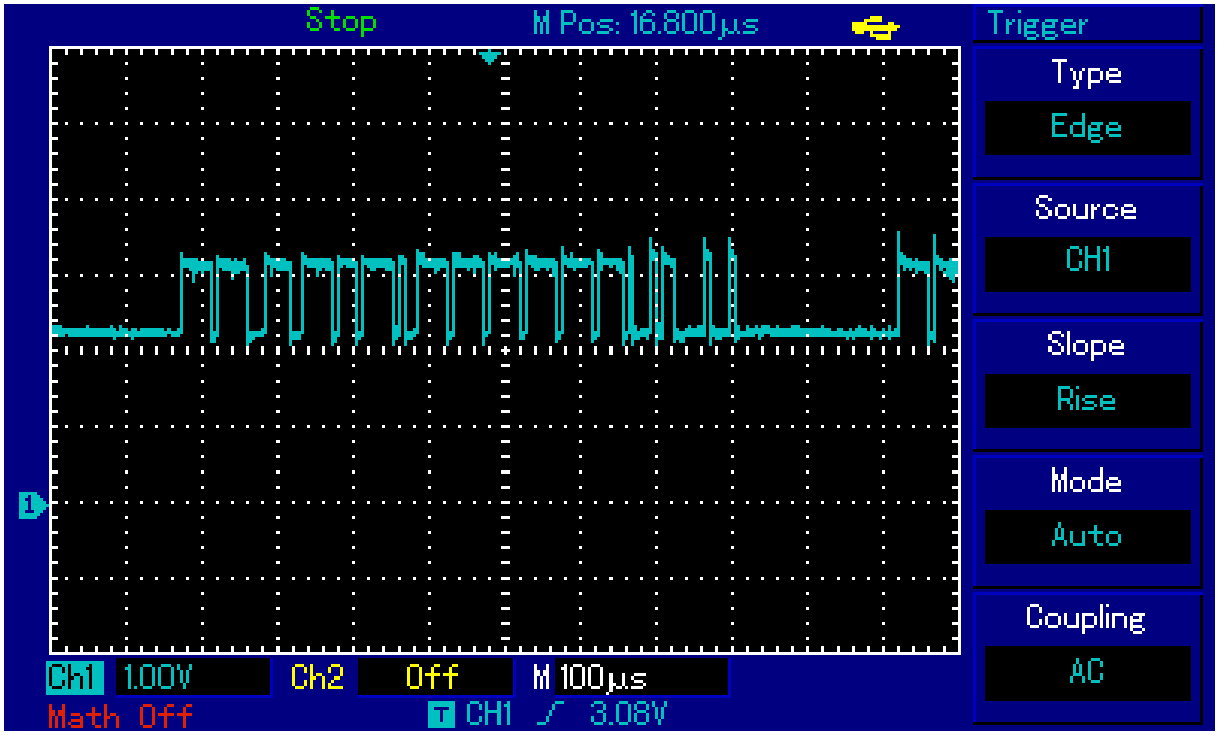


Figure 5.4: Short transmission line CAN traffic

Figure 5.5 is the CAN bus as seen when using the 1m twisted pair. It is the same frame as viewed in figure 5.4, however now we see significant overshoot on rising edges. Removal of the terminating resistor on the click board does not noticeably alter the image.

Figure 5.6 is the CAN bus as seen when using the 10m twisted pair. The overshoot is still prevalent and is joined by significant ringing on the frame. Again the terminating resistor had no meaningful effect.

To this point, transmission line tests had been performed using the development board that generates the errors on the USB-CAN converter. Figure 5.7 is taken from the working CAN bus on 10m twisted pair. The ringing and overshoot is present on this test as well. This is a frame from the USB-CAN converter.

Figure 5.8 shows the response from the bridge on the working bus using 10m transmission line. It is a much cleaner signal than the USB-CAN converter produces in any test.

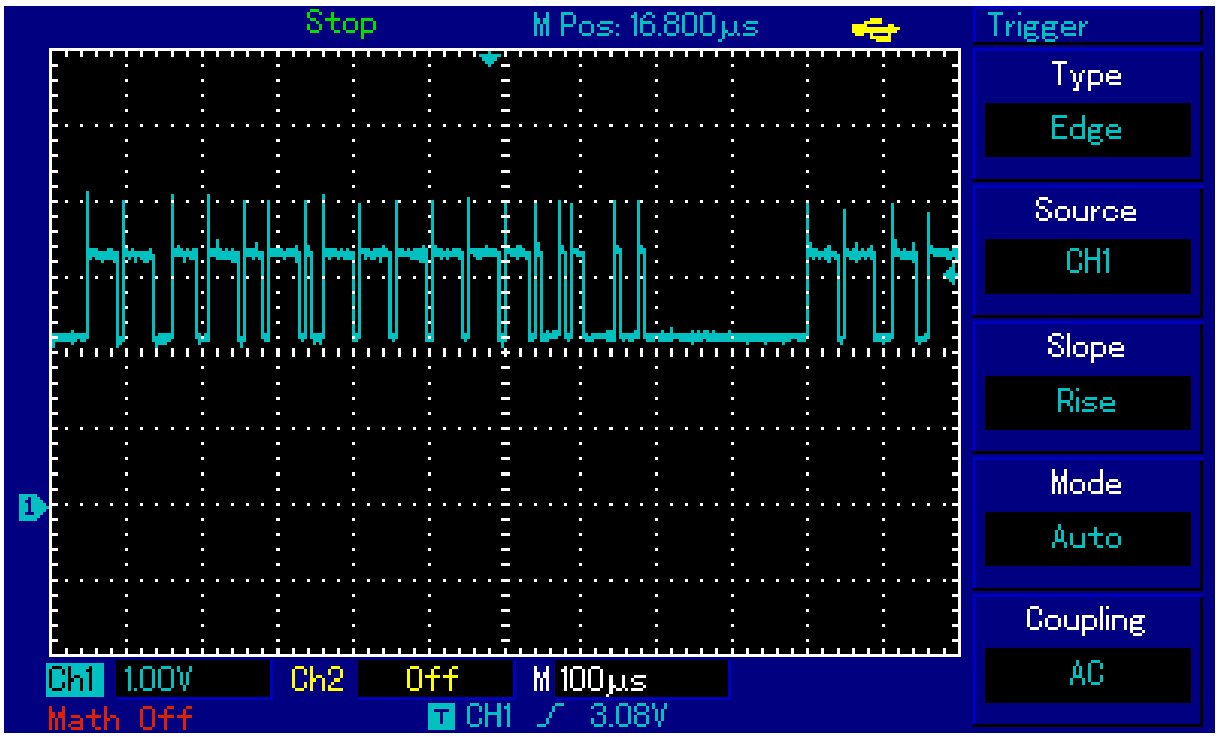


Figure 5.5: 1 meter twisted pair CAN traffic

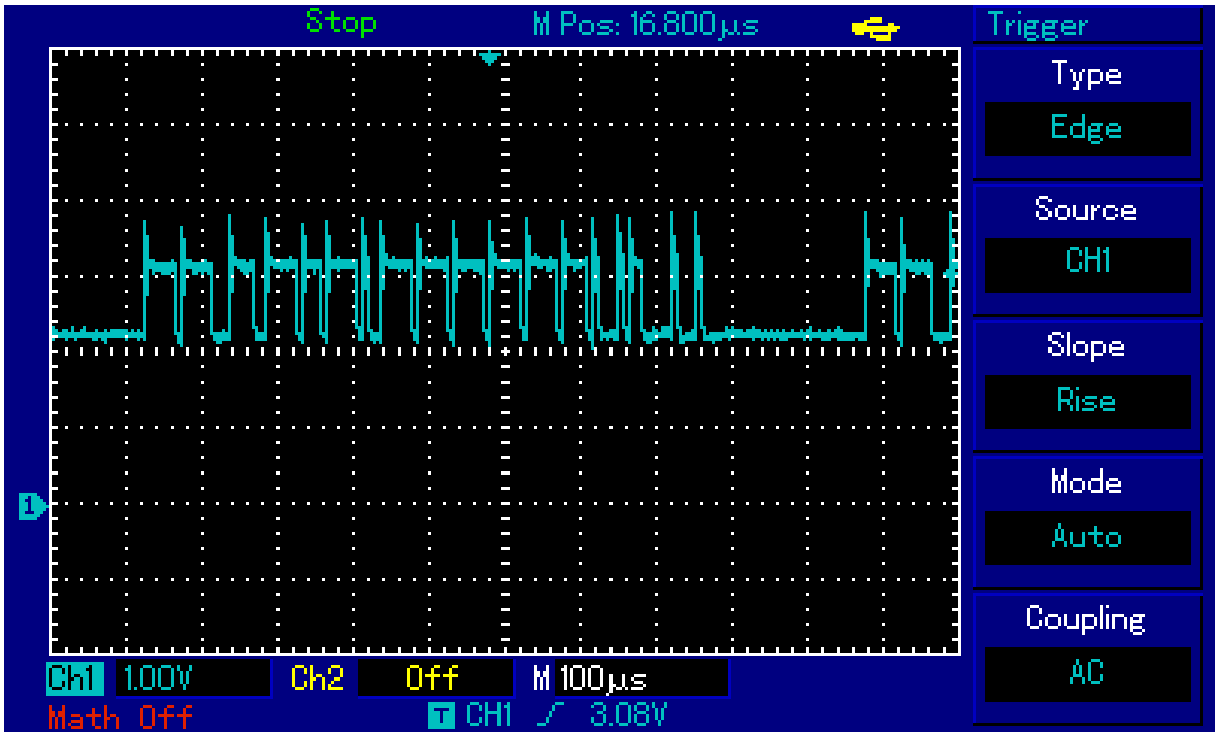


Figure 5.6: 10 meter twisted pair CAN traffic

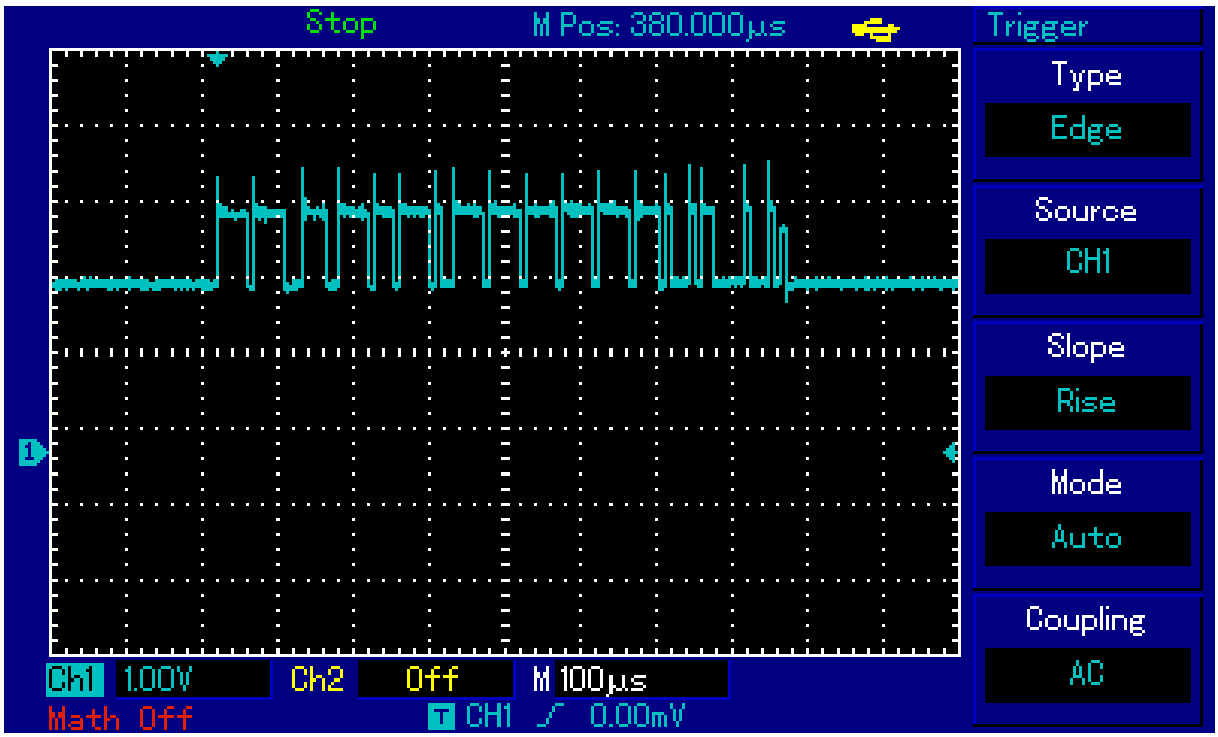


Figure 5.7: 10 meter twisted pair CAN traffic on working bus

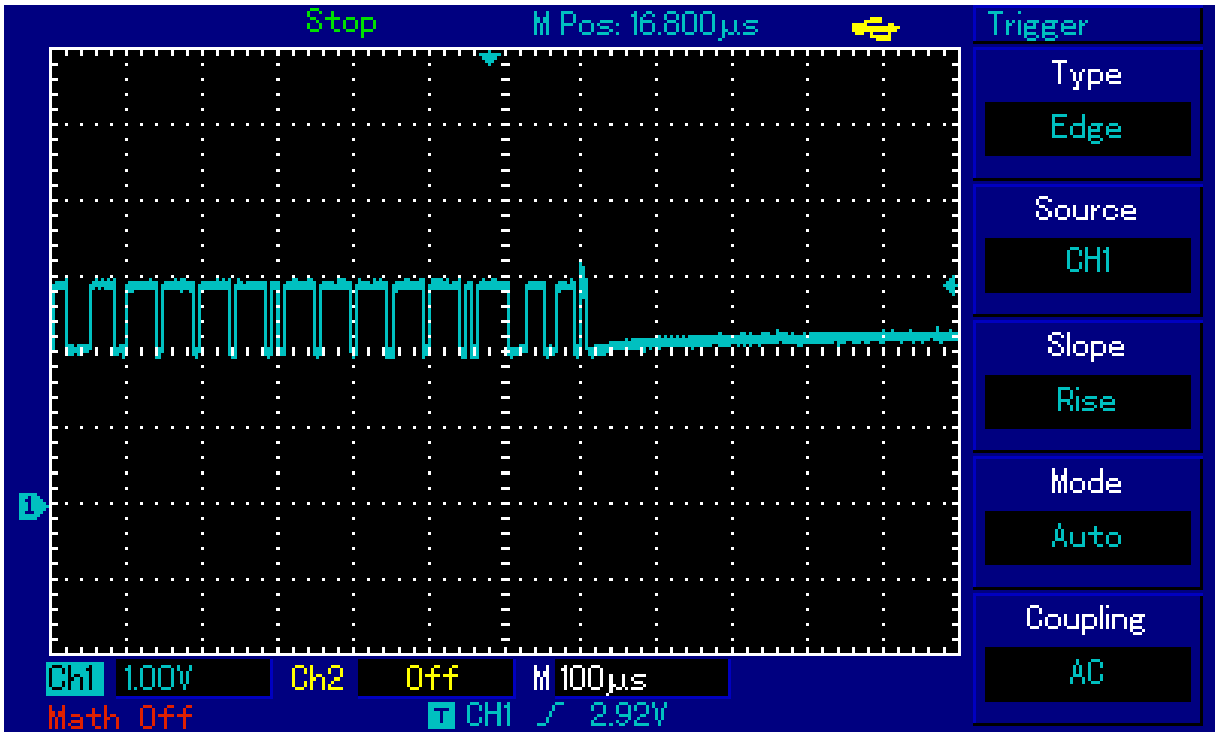


Figure 5.8: 10 meter twisted pair CAN response

5.2 Discussion

Initial testing of the bridge was unsuccessful. The cause of the issue was pinpointed to be either the bridge's CAN interface, the USB-CAN converter or a combination of both. These issues did not allow full completion of the testing as per the methodology. For the testing that was performed, results are discussed below. Following that is a discussion of the issues and limitations of the bridge, the methodology and the testing equipment.

5.2.1 Proof of Concept

The concept of the bridge has been demonstrated successfully. A full message was sent and received through both bridge interfaces. This proves that according to the methodology outlined, the hardware and software designs of the bridge can perform the basic functionality required.

5.2.2 Stress Tests

Initial testing showed malfunctions of the CAN interface, thus results for all stress testing parameters were not obtained. The discussion of the cause of these errors is found in the subsection labelled Issues and Limitations - CAN Interface.

The results that were attained showed that at CAN baud rate 125kbps and Modbus baud 9.6kbps, the bridge is able to complete, without loss, full CAN to Modbus and return message translation in around 220ms. This effectively means that the bridge can be polled by the BMS CAN node four times a second. This is far more than needed for a practical BMS requirement. If it were possible to increase the baud rates for the interfaces, it is likely we would see an increase in performance. This is unnecessary given the proven performance is adequate. There is no known reason why gas detection equipment would need to update its status at a faster rate.

There is a linear decrease in performance at faster messaging rates. The effect would be significant if messaging times were required in that region. For the application of the bridge that is not the case.

For all tests at all messaging rates, the number of CAN frames received back by the

BMS CAN node simulator was exactly equal to the number of Modbus frames sent by the bridge. This means that the frame loss must occur between the bridge's CAN frame reception and Modbus transmission. As it was not possible to alter the source code of the bridge during testing, a counter could not be placed at these points in software. If that were possible, it would identify where the loss had occurred and verify the assertion above.

5.2.3 Proof RS-485 Concept

The testing results for the RS-485 tests showed the expected signal on the bus that conforms to the RS-485 standard. This indicates that due to the consistency of design between the test platform and the PCB, the PCB can be expected to perform similarly and therefore conforms to the RS-485 standard.

5.2.4 Physical Cable Testing

The results of the physical cable testing showed that significant overshoot and ringing was present on the bus during tests with different parameters. These results were consistent across the working CAN bus and non-working CAN bus tests. This suggests that the transmission errors experienced were not caused by the transmission lines.

5.2.5 Discussion of issues and limitations - CAN Interface

The original design methodology for the CAN interface was to utilise the existing CANSpi library. It was intended that SPI interrupt would be used to signal that SPI data had been received. This would flag to the bridge that a CAN message was received. This approach was not suitable. The CANSpi library initialisation function sets the parameters of the SPI module. Under these conditions, SPI interrupts enabled within the main program do not function. As a result, the interrupt method for CAN reception was abandoned. Each pass through the main functions' infinite loop tests whether a CAN message has been received. This is the alternative solution.

Another issue encountered with using CANSpi library functions within interrupt routines is their re-entrancy. When there is a possibility of a function being called from the main program loop *and* an interrupt routine, the MikroC compiler throws a re-entrancy error. This is to stop the interrupt call of the function overwriting the stack or data from the original function call. This also occurred during testing of Modbus functions within interrupts. The issue is resolvable by setting flags in interrupt services routines that are checked in the main loop. This is the approach taken for the Modbus (UART) interrupts.

Two 5V CANSpi modules were used in early attempts to successfully initiate CAN communications. Use of these devices was not successful. The CANSpi click board was configured on the development board. One of the 5V CANSpi modules was configured on another identical development board. Previous research in chapter 2 had shown that 5V and 3.3V CAN transceivers can exist on the same bus. When CAN messages were not detected on either end, diagnostic checks were performed using an oscilloscope. The inspection found that SPI data was not transferring from the CAN-SPI as expected. The CAN-SPI chip was replaced on these boards and the testing was performed again. One of the two CANSpi 5V board still did not function. The other now produced a signal that resembled a CAN frame. The two different CANSpi modules still did not communicate with each other. At this point, the USB-CAN converter had become available and testing went forward on that platform only.

Tests performed with the USB-CAN converter and the development board fitted with the CANSpi click board were challenging. The most significant issue was the inability of the bridge to receive CAN messages. CAN frames were successfully sent from the bridge to the USB-CAN converter. To use the CANSpi library certain flags, masks and filters need

to be set. The data-sheet of the MCP2515 [3] and the help file for the CANSpi library in MikroC were difficult to reconcile. There is no consistency in naming. It takes educated guessing or trial and error to identify correct parameters for use. The masks and filters were ruled out as the cause. The issue was eventually identified to be with CAN flags. The library example provided with MikroC was used as a guide. In this example the CANSpi initialises with the 11 bit message type. The reception flags were configured for 29 bit reception only. This was difficult to identify due to the lack of consistency between the library help and component data-sheet. The change required to complete the test were alterations of the flag "_CANSPI_XTD_MSG" to "_CANSPI_STD_MSG" in two locations of the source code. This was a simple issue to resolve but time intensive to identify.

The above issues were found during early testing to demonstrate that the CANSpi library was viable. The major issue with the CAN interface was discovered when the proof of concept test was attempted. Inconsistencies were observed with the CAN interface. The USB-CAN intermittently reported a transmission error indicated via the 'err' LED flashing. The error did not occur with any pattern which made troubleshooting difficult. Occasionally, the USB-CAN converter would successfully send one or more CAN frames and then stop working, giving the error indication. Other times it would not send any successful CAN messages. An attempt was made to use the SPI to report the contents of the registers in the CAN-SPI IC in the hopes of diagnosing the error. The code listing can be found in appendix F. The initial conclusion was that the issue is related to differing configurations of the USB-CAN converter and the initialisation functions of the CANSPI library. There is insufficient information in either user manual to reconcile these issues easily or in a timely manner. It is not possible to rule out equipment failure given that the USB-CAN devices is a two port device and one of the two ports stopped working during testing.

To this point, the CAN interface had been very inconsistent and only functioned sporadically. The functionality was sufficient for proof of concept testing. Diagnostics were abandoned due to time limitations at this stage of the project.

During final testing and data gathering the bridge began to work flawlessly on the main development board. This was without making any change to source code that had previously not worked with any consistency. Uploading the same source code onto the secondary development board did not work. This meant any further configuration of the source code

[3]The CAN-SPI IC on the bridge is the same as the one found on the click board by design.

past this point was not possible. The only difference between the two development boards are the crystals connected to the processor. They should not effect the CAN interface given it has its own, separate oscillation source. The conclusion here is that the source code may have become corrupted. This would explain why the latest source code version does not function. Once rectified, it would be possible to carry out stress tests at the different baud rates. Full exploration of board capabilities would also be possible.

What was possible was tested using the working development board with the configuration parameters set in the source code. This is the CAN baud at 125kbps and Modbus baud at 9.6kbps as described in the stress testing above. The CAN message type is the 11 bit identifier version(CAN 2.0A).

Development of user written functions to access the CAN interface via SPI may address some or all of the above issues. It would give full control over the interface. This would also likely improve the performance of the bridge. The obvious downside is that a lot of work would be required to recreate the functionality offered by the CANSpi library and this is out of the scope of this project.

5.2.6 Discussion of issues and limitations - Modbusslave simulator

Equipment issues were experienced with the Modbusslave simulator. This program has a serial traffic viewer to allow the user to monitor outgoing and incoming serial messages in ASCII format. When doing initial testing with the development board some irregularities were observed. A screen capture of an example can be seen in figure 5.10. The data that was sent is as per figure 5.9.

```
while (1) {                          // Endless loop
    UART1_Write(0x3A);          // and send data via UART
    UART1_Write(0x30);          // and send data via UART
    UART1_Write(0x31);          // and send data via UART
    UART1_Write(0x32);          // and send data via UART
    UART1_Write(0x33);          // and send data via UART
    delay_ms(500);
}
```

Figure 5.9: Code exert of serial data sent

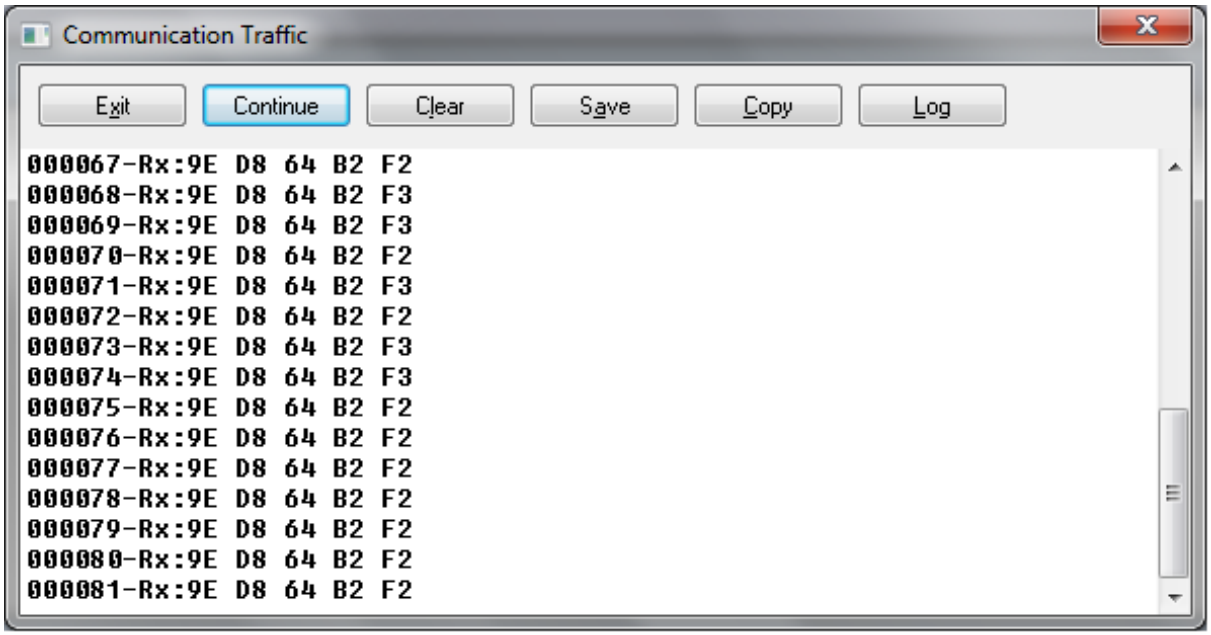


Figure 5.10: Serial synchronisation errors with 16MHz crystal

The data being sent from the bridge was not consistent with what was received by the Modbusslave traffic viewer. Reproducing the transmission and viewing the results on the MikroC terminal produced the expected outcome. The issue was identified as a timing error. The crystal on the development board was 16MHz. This led to small errors in

baud generation for the UART. This synchronisation error caused the Modbusslave to receive characters not sent and to not receive characters actually sent. Replacement of the 16MHz crystal with a 14.7456MHz crystal[4] removed the error. Serial traffic then appeared as expected. Interestingly, during compilation of results for this dissertation, the second development board with a 16MHz crystal could not reproduce this error. Whilst the change of crystal appeared to rectify the problem during the first encounter, it is not possible to rule out other factors as the cause. Using a perfect divisor crystal is still advisable when possible.

The resolution of the timing error led to further testing of the Modbusslave simulator. The simulator was configured as per an example on the developers website. The development board was then configured to send a test message that would extract some information off the simulator in a known fashion. The results of this test were not as expected. The simulator did receive and respond to the sent frame, but the response was not what was expected. It was also not a known error response. Further testing showed that regardless of the frame sent to the simulator, the same response is received (data component is 00 00 08 00 00 01). This can be seen in the results of the proof of concept. The version of the simulator used was a free trial. As such the software does not offer full functionality. For the purposes of testing the bridge it was adequate. It did prevent the testing of Modbus frames with data sizes over 6 bytes. Responses are always the same as described above, and as a result never contain enough (16bytes or more) data to test the decomposition and reconstitution of large data frames.

5.3 Chapter Summary

This chapter has covered the results from testing the bridge, including proof of concept, stress testing, RS-485 testing and the testing of physical cables. It also included a discussion of the significance of these results and detailed discussion of the issues and challenges encountered during the bridge design and testing.

[4]The same device found on the final bridge hardware. This discovery drove that selection.

Chapter 6

Conclusion and Future Work

6.1 Conclusions

The network bridge to translate between CAN and Modbus frames is of commercial interest to GDA. Research showed the concept was feasible. A PIC microprocessor based embedded system was designed to facilitate the protocol translation. The hardware (PCB) and software (C code) were outlined in detail.

Relevant testing demonstrated both the proof of concept and initial performance. The testing was limited but demonstrated the bridge will be able to respond fast enough for the given application. The bridge is able to respond to four CAN messages per second at CAN baud 125kbps and Modbus baud 9.6kbps. This is sufficient for the BMS application and meets the main project objective.

Many issues were encountered with the implementation of the CAN interface and they prevented further exploration of the bridge performance and capabilities. The cause of these issues were identified to be the source code for the CAN interface. This was a result of a lack of configuration control available to the user when utilising the inbuilt library functions of the MikroC development environment.

A limitation of the Modbus simulator was experienced. It did not respond with accurate or expected frames to legitimate requests. This issue did not affect the critical testing, but did prevent the testing of decomposition and reconstitution of Modbus frames with data sizes greater than sixteen bytes. The design of this process was still presented but not implemented.

As the bridge did not reach a complete finished state, documentation was not completed in full. Appendix I contains an outline of the intended content of these documents.

The main objectives of the project were met. The design was completed and proof of concept demonstrated. Full testing and exploration of the bridge performance and capabilities were not possible. The next section details relevant further work to rectify the issues and develop this topic further.

6.2 Recommendations and Future Work

The inconsistencies with the CAN interface are of most interest. It would be useful to do further testing with the SPI to communicate directly with the CAN-SPI chip to troubleshoot the CAN interface. If time was available, a full re-write of the CANSpi libraries from scratch would give more control to the developer and allow better diagnostics. Once complete, a full suite of stress testing could determine the full capabilities of the bridge.

To improve and expand the Modbus interface functionality a few simple changes could be made. Full support of all parity types could be implemented. This is a simple fix requiring the configuration of one bit per byte of the serial transmission. The decomposition and reconstitution of data sizes greater than 16 bytes could be implemented. This would require access to a more sophisticated Modbus slave simulator.

Modbus RTU functionality could be added to make the bridge more flexible. As seen with existing commercial products, they are generally static in their implementation. The implementation is relatively simple. The ASCII and Hex conversions are no longer required, and the frames do not use delimiters. This makes the frame composition simpler than Modbus ASCII. The added complexity comes in the sending of the frames. Interrupt driven timers are required to adhere to the timing laws of the Modbus RTU transmission scheme.

Finally, investigations into Modbus TCP could be carried out. This would require the addition of an Ethernet hardware interface to the bridge, adding greater flexibility to the device.

There are non-communications based work that would improve the bridge. The GLCD and touch panel could be configured to allow status displays and customisation of the bridge parameters such as baud rates. The software configuration for power failure and LED operation (including errors) could be implemented.

If performance improvements were required, one way to achieve this might be to further logically separate the CAN and Modbus interfaces. This would involve make the bridge a complete standalone Modbus master that interacts with the Modbus slaves as per normal Modbus operation. The bridge could maintain the key Modbus data in memory and simply release it when a CAN request came through.

References

ADF web (2015), 'Gateway/bridge: Can rom/to Modbus', http://www.adfweb.com/home/products/Can_modbus.asp?frompg=nav1_7. [Online; accessed April-2015].

ANSI (2014), Tia/eia Standard: Commercial Building Telecommunications Cabling Standard, Standard TIA/EIA-485, American National Standards Institute.

Blackman, J. & Monroe, S. (2013), Overview of 3.3v Can (controller area network) Transceivers, Applications report, Texas Instruments.

Butzin, B., Golatowski, F., Niedermeier, C., Vicari, N. & Wuchner, E. (2014), A model based development approach for building automation systems', *in* 'Emerging Technology and Factory Automation', Institute of Electical and Electronic Engineers, Barcelona.

CAN in Automation (2015*a*), 'Can: Physical layer', http://www.can-cia.org/index.php?id=systemdesign-can. [Online; accessed April-2015].

CAN in Automation (2015*b*), 'Controller area network', http://www.can-cia.org/index.php?id=systemdesign-can. [Online; accessed April-2015].

Cena, G., Bertolotti, I., Hu, T. & Valenzano, A. (2014), Design, verification, and performance of a modbus-can adaptation layer, *in* '10th IEEE Workshop on Factory Communication Systems', Toulouse.

Corrigan, S. (2008*a*), Controller area network physical layer requirements, Applications report, Texas Instruments, Texas.

Corrigan, S. (2008*b*), Introduction to the controller area network (can), Applications report, Texas Instruments, Texas.

de Sousa, M. & Portugal, P. (2011), Modbus, *in* B. Wilamowski & D. Irwin, eds, 'The Industrial Electronics Handbook: Industrial Communications Systems', CRC Press, chapter Modbus.

Fairchild Semiconductor Corporation (2006), 'Single-channel: 6n137, hcpl-2601, hcpl-2611 dual-channel: Hcpl-2630, hcpl-2631 high speed-10 mbit/s logic gate optocouplers', Datasheet.

Fairchild Semiconductor Corporation (2015), 'Df005s - df10s: Bridge rectifiers', Datasheet.

Ferreira, J. & Fonseca, J. (2011), Controller area network, *in* B. Wilamowski & D. Irwin, eds, 'The Industrial Electronics Handbook: Industrial Communications Systems', CRC Press, chapter Controller Area Network.

Gas Detection Australia (2015), 'Usq fourth year project bachelor of beng electrical/electronic engineering', Project Specification. Toowoomba.

Guohuan, L., Hao, Z. & Wei, Z. (2009), Research on designing method of can bus and modbus protocol conversion interface, *in* 'International Conference on Future BioMedical Information Engineering', Institute of Electrical and Electronic Engineers, Sanya.

HMS (2015), 'Anybus communicator: Can', `http://www.anybus.com/products/abccan.shtml`. [Online; accessed April-2015].

Hui, L., Hao, Z. & Daogang, P. (2013), Research and application of communication gateway of epa and modbus/tcp, *in* '5th International Conference on Computational Intelligence and Communication Networks', Mathura.

ICP DAS USA (2015), 'I-7530a-mr: Modbus rtu to can converter', `http://www.icpdas-usa.com/i_7530a_mr.html`. [Online; accessed Apirl-2015].

IEEE (2015), Ethernet, Standard IEEE802.3, Institute of Electrical and Electronic Engineers.

Institute of Engineers (1997), Towards sustainable engineering practice: engineering frameworks for sustainability, Technical report, Institute of Engineers, Canberra.

ISO (2003*a*), Road vehicles - controller area network (can) - part 1: Data link layer and physical signalling, Standard ISO 11989-1, International Organization for Standarization.

ISO (2003*b*), Road vehicles - controller area network (can) - part 2: High-speed medium access unit, Standard ISO 11989-2, International Organization for Standarization.

Maxim Integrated (2006), 'How far and how fast can you go with rs-485?', `https://www.maximintegrated.com/en/app-notes/index.mvp/id/3884`. Application Note.

Microchip (2010), 'Mcp2551: High-speed can transceiver', Datasheet.

Microchip (2011), 'Pic18f87k22 family: 64/80-pin, high-performance, 1-mbit enhanced flash mcus with 12-bit a/d and nanowatt xlp technology', Datasheet.

Microchip (2012), 'Mcp2515: Stand-alone can controller with spi interface', Datasheet.

MikroeElektronika (2012*a*), *CANSpi Click Manual v100*, MikroeElektronika.

MikroeElektronika (2012*b*), *RS485 Click Manual v100*, MikroeElektronika.

Modbus Org (2006*a*), Modbus messaging on tcp/ip implemenation guide, Technical report, Modbus Organisation, Inc.

Modbus Org (2006*b*), Modbus over serial line: Specification & implementation guide, Technical report, Modbus Organisation, Inc.

Modbus Org (2006*c*), Modbus protocol specification, Technical report, Modbus Organisation, Inc.

Modicon, Inc. (1996), Modbus protocol reference guide, Technical report, MODICON, Inc., North Andover, Massachusetts. Rev. J.

RECOM (2012), 'Roe0505s: Econoline dc/dc-converter', Datasheet.

Rinaldi, J. (2010), 'Modbus tcp vs. modbus rtu', `http://www.rtaautomation.com/modbus-tcp-vs-modbus-rtu`. [Online; accessed April-2015].

So, A. (2001), Building control and automation systems, *in* T. Samad, ed., 'Perspectives in Control Engineering Technologies, Applications and New Directions', Wiley-IEEE Press, chapter Building Control and Automation Systems.

Texas Instruments (1998), 'Lm2671 simple switcher power converter high efficiency 500ma step-down voltage regulator with features', Datasheet.

Thomas, G. (2008), 'Introduction to modbus serial and modbus tcp', *the EXTENSION: A Technical Supplement to Control Network* **9**(5), 1–4.

University of Southern Queensland (2012), 'Electronic circuits: Study book'. Toowoomba.

Wang, L., Zhang, T., Li, K. & Ren, B. (2013), Study of can-modbus communications adapter for low-voltage distribution system, *in* 'International Conference on Mechatronic Sciences, Electric Engineering and Computer', IEEE, Shenyang.

Appendix A

Project Specification

ENG 4111/2 (or ENG8002) Research Project

Project Specification

For:	**Matthew Quinton**
Topic:	CAN bus to MODBUS network bridge to interface gas detection units with Building Management Systems
Supervisors:	Mark Pythian James Boucher
Sponsorship:	Faculty of Health, Engineering & Sciences Gas Detection Australia Pty Ltd
Project Aim:	The aim is to investigate, design and build a network bridge to interface CAN bus enabled Building Management Systems (BMS) with MODBUS enabled gas sensors and control units.

Program:

1. Review the literature to investigate the relevant CAN bus and MODBUS standards and their implementations, how BMS function and any existing solutions for the network bridge.
2. Based upon the literature and sponsor requirements create network bridge specifications including consideration for hardware connections, standards, device intelligence and data integrity.
3. Develop a network bridge including hardware and software components for the specified hardware connection type.
4. Develop a testing procedure for the network bridge and verify correct functionality.
5. Produce all documentation including technical manual, operators manual, manufacturing procedure, programming guide and software manuals as necessary.

As time and resources permit:

1. Design and build/acquire/emulate a BMS system and test the network bridge in a quasi-situation or on-site with a customers BMS and evaluate performance.

2. Investigate and implement other MODBUS hardware configurations as a separate specification or additional board option

3. Investigate and implement other hardware connections options as a separate specification or additional board option

Agreed:

Student Name:	Matthew Quinton
Date:	13th April 2015
Supervisor Name:	Mark Phythian
Date:	13th April 2015

Appendix B

Controller Area Network

This appendix contains further technical information about the CAN protocol.

B.1 List of frame field descriptions

The following describe the fields of CAN frames as per ISO (2003*a*).

SOF Start of frame used to synchronise nodes after idle state (dominant bit).

Identifier Establishes message priority as per arbitration (11 or 29 bits).

RTR Remote transmission request dominant bit when information is required from another node. The identifier determines the specific node requested.

IDE Identifier extension this means that a standard CAN identifier is being transmitted with no extension (dominant single bit).

R0 Reserved bit unused at this stage.

DLC Data length code number of bytes being transmitted (4 bit).

Data Up to 64bits may be transmitted.

CRC Cyclic redundancy check 15 bits plus 1 bit delimiter contains checksum for error detection.

ACK Receiving nodes overwrite this bit with dominant to indicate error free message received. If the receiving node leaves as recessive the message is discarded and the sender repeats transmission.

EOF End of frame 7 bit field indicates end of frame. Also disables bit stuffing. Dominant means a stuffing error.

IFS Interframe space 7bit field contains time required to move a correctly received frame to its position in a message buffer by the controller /citecorr2.

The extended frame includes the extra 18 bit identifier as well as three other fields.

SRR Substitute remote request bit replaces RTR and is a placeholder.

IDE Identifier extension bit indicates that more identifier bits follow.

R1 Additional reserve bit .

B.2 Physical Connection

The physical medium is two wires and a separate ground connection. Unshielded 120 ohm twisted pair cabling is common. Using higher quality cable is recommended in harsher environments. To minimise the effect of reflections on the cables terminating resistors are required on both ends of the bus. Connectors used should also have the same characteristic impedance as the line and terminating resistors /citecorr1. There are several other considerations that need to be taken into account when designing CAN bus networks but are largely beyond the scope of this project. The aim is to add a node into an existing network and not to create or alter an existing or new network. The characteristics of the connections are of importance and should be considered when selecting components.

B.2.1 Topology

CAN is bus based. Replicated buses are not explicit in the standard but are possible and research has been conducted around star. Star networks require a coupler or some other central device to facilitate broadcasts to nodes. Faults have to be treated differently when in star mode and this is the basis for significant research beyond the scope of this project (Ferreira & Fonseca 2011).

B.2.2 Transmission and Signals

CAN is a differential signalling protocol. A high and a low signal is sent out on complementary connections. The message is dominant when the high is held high and the low is held low (ISO 2003*a*). Typically, the bus is passively biased to 2.5V. The high voltage is approximately 1V higher (3.5V) and the low voltage is approximately 1V lower (1.5V). This gives a typical voltage differential of 2V (Corrigan 2008*a*). 3.3V or 5V transceivers are common and can be interoperable with each other (Blackman & Monroe 2013). Figure B.1

demonstrates the dominant and recessive voltages levels for the CAN bus. Dominant bits are used in the arbitration process for multiple access.

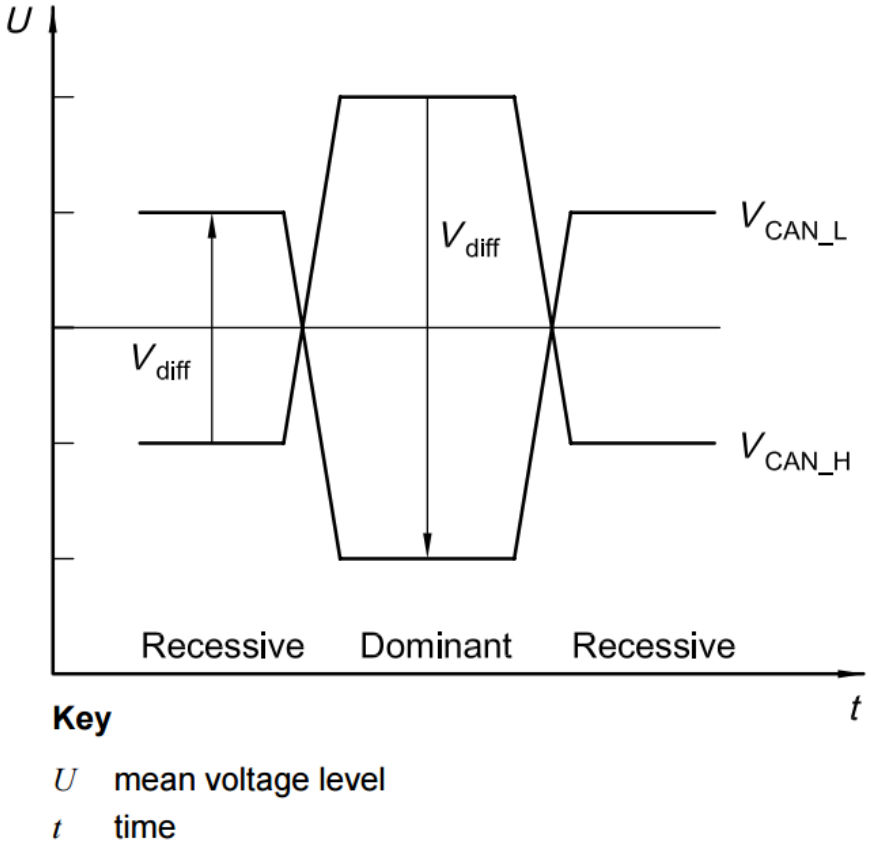


Figure B.1: CAN bit levels (adapted from (ISO 2003*a*)).

B.2.3 Encoding

CAN uses Non Return to Zero (NRZ) and as such only one timeslot is required per bit (CAN in Automation 2015*a*). Figure B.2. shows the NRZ encoding scheme as well as the Manchester encoding for point of difference. Manchester encoding uses the state change to signify a bit where NRZ uses a level. The key difference is that NRZ simply drives the bit value on the bus whilst Manchester schemes force a transition.

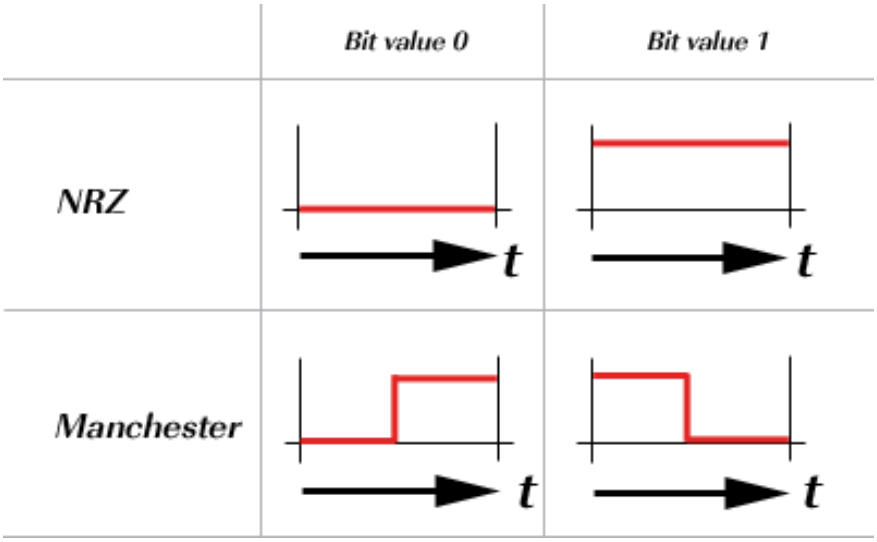


Figure B.2: NRZ and Manchester Encoding (adapted from (CAN in Automation 2015*a*)).

B.2.4 Synchronisation and Bit Stuffing

The NRZ encoding scheme has no inherent synchronisation so this is achieved using a start bit within the packets. Figure B.3 demonstrates the synchronisation issues with NRZ. If a large number of consecutive bits of the same value are transmitted it can be difficult to resynchronise due to the lack of a rising or falling edge. To overcome this issue CAN uses bit stuffing to break sequences of 6 or more consecutive bits of the same polarity. A bit of opposite polarity to the sequence is inserted after 5 consecutive bits. Bit stuffing introduces extra overhead to transmission. The effect can be estimated if the timing is an issuet (CAN in Automation 2015*a*).

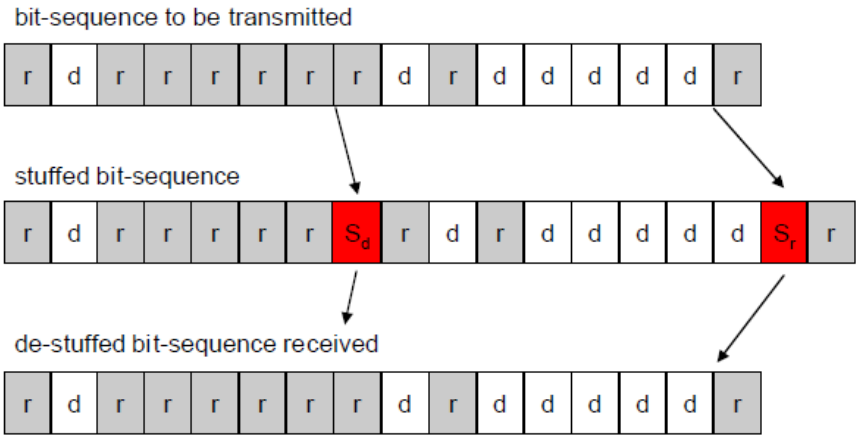


Figure B.3: CAN bit stuffing (adapted from (CAN in Automation 2015*a*)).

B.2.5 Errors

CAN has five error checking methods; three at message level and two at the bit level. When any of these errors is detected, an error frame is generated by the receiving node. The transmitting node will attempt to resend the frame until received correctly. The three message level checks are the CRC, the ACK and the form check. The CRC is a standard 15-bit checksum with a 1-bit delimiter. The ACK is comprised of two bits; the ACK bit and a delimiter. The form check is carried out by checking the SOF, EOF, ACK and CRC delimiters. During arbitration, conflicting bit values may be written to the bus and this is also true of the acknowledge slot. If this is detected for any other bit an error is generated. The second bit level error check is bit stuffing. If six consecutive bits of the same polarity are received then an error is generated.

B.3 Masks and Filters

The following contains some information about how the filters and masks work in CAN addressing (ISO 2003*a*).

Filters are a bitwise enabler of CAN IDs. In order for a message to make it through a filter, 1s must be in place of the corresponding ID bits. For example:

ID = 3 - convert to binary - 00000000011 (for an 11 bit address)
Filter - convert to binary - 00000000000 - would not allow this ID through since the 0s in the bit spacers of the ID filter
Filter - convert to binary - 00000000011 - would allow this ID through since the 1s enable the spaces in the filter

The masks take precedence over the filters. In other words, the mask enables the filter. For our previous example:

ID = 3 - convert to binary - 00000000011
Filter - convert to binary - 00000000000 - we saw before this would not allow ID 3 through, unless we configure a mask to override it
Mask - convert to binary - 00000000000 - here we are saying we don't care what the filter says, let everything in. The mask 0s disable the corresponding bits in the filter.
Likewise, a mask of 11111111111 will not let any message through.

This setup gives you quite a lot of flexibility in how you handle messages, and through which filters and buffers they pass.

Most of the abilities of these filters are not used within the scope of the bridge. They are important on more complicated CAN networks and would need to be considered for future commerical use of the bridge.

Appendix C

Modbus

C.1 Protocol Data Unit

The Protocol Data Units (PDU) is comprised of the function code and the data field of a Modbus frame. There is a large list of functions available for Modbus. The most commonly used are those associated with accessing memory addresses (de Sousa & Portugal 2011). At this stage, a complete discussion of the available function codes is not required.

C.2 Error Checking

Modbus has two built in error checking facilities; parity checking and frame checking. Parity checking is implemented by configuring network devices as either Even, Odd or No parity. If Even or Odd is chosen then one bit is adjusted so that the data frame has the same parity as the network configuration. This will detect bit errors where the number of errors is odd only. Frame checking is achieved using a Cyclic Redundancy Check (CRC) for Modbus RTU or Longitudinal Redundancy Check (LRC) for Modbus ASCII.

C.3 Physical Media

Some of the features of the protocol are not solely governed by the specification alone. This is due to the fact that Modbus does not define a physical layer. The physical media that carry it define some parameters. For this project the serial interface used is UART and RS-485.

RS-485 is a two wire balanced system with a third common wire. Four wire Modbus is possible to give full duplex capabilities to a network. This project is a two wire based network. A requirement of Modbus serial is to have shielded cables where one end of each cable is connected to protective ground (Modbus Org 2006*b*).

RS-485 allows multi-drop connections using a bus (trunk) with direct connections (daisy chain) or derivations (drop) cables. Derivation cables must be no more than 20 meters long (Modbus Organisation, Inc. 2006). 1200 meters is the maximum length for the trunk cable (Maxim Integrated 2006).

C.4 RS 485 Data Rates

The specification defines the two standard baud rates of 9600kbps and 19200kpbs of which the latter is the default (Modbus Organisation, Inc. 2006). The maximum baud listed in the specification is 115kpbs. It may be possible to operate Modbus on serial lines at speeds of 10Mbps (Maxim Integrated 2006), although is a trade-off between run length and speed and as such this speed will not be achievable at near maximum lengths. Using techniques such as pre-emphasis and receiver equalisation it may be possible to achieve data rates of 800kbps over 1200 meters and 50Mbps over lengths of hundreds of meters (Maxim Integrated 2006).

C.5 Encoding and Synchronisation

RS-485 is an asynchronous differential system that utilises a start bit/s and stop bit/s either side of the data. Timing between sender and receiver must be agreed upon ahead of time and synchronisation is carried out relative to the leading edge of the start bit (**?**).

Appendix D

Network Bridge Source Code - C

D.1 C Header

This listing contains the header file where function prototypes are defined.

Listing D.1: Project Header.

```
/*-----------------------------------------------------------------------------
-----------------------------------------------------------------------------

Gas Detection Australia
CANbus to Modbus Network Bridge
BENH Final Year Project

Matthew Quinton
0061024766

Project header
PIC: 18F87K22
V1.0D
-----------------------------------------------------------------------------*/
//-----------------------------------------------------------------------------
//-----------------------END OF HEADING----------------------------------------

/*-----------------------------------------------------------------------------
MODBUS FUNCTIONS
-----------------------------------------------------------------------------*/

void modbus_init();

void TX_serial(unsigned short numchar);

void TX_modbus(unsigned char* buff, unsigned char length);

void RX_Serial(unsigned char *ret);

void RX_modbus(unsigned char *ret, unsigned short length);

unsigned short LRC(unsigned char* buffer, unsigned short numchar);

/*-----------------------------------------------------------------------------
GENERAL FUNCTIONS
-----------------------------------------------------------------------------*/

unsigned short hex_ascii(unsigned short digit);

unsigned short ascii_hex(unsigned digit);

unsigned short split_first(unsigned short hex_dig);

unsigned short split_second(unsigned short hex_dig);

unsigned short comb_digits(unsigned short dig1, unsigned short dig2);
```

D.2 Modbus Functions

This listing contains the Modbus functions using the UART for serial communications.

Listing D.2: Modbus Functions.

```
/*-------------------------------------------------------------------------
-------------------------------------------------------------------------

Gas Detection Australia
CANbus to Modbus Network Bridge
BENH Final Year Project

Matthew Quinton
0061024766

Modbus ASCII Master Functions
PIC: 18F87K22
V1.0D
-------------------------------------------------------------------------*/
//-------------------------------------------------------------------------
//----------------------END OF HEADING-----------------------------------


/*-------------------------------------------------------------------------
Definitions and variables
-------------------------------------------------------------------------*/
#include "ProjectDevBoard.h"


unsigned short baud = 9600;             // Baud rate
unsigned short parity = 0;              // Parity(0 - even, 1 - odd, 2 - none)
unsigned short stop_bits = 1;           // Number of stop bits (1 - even/odd par
unsigned short slave;                   // Slave address for TX and RX

unsigned short MBReceived = 0;           //Modbus frame Recived flag
unsigned short SerialCount = 0;         //Number of bit recived through serial
unsigned short SerialBuffer[30];        //Buffer used to store partial modbus fr
unsigned short ModbusRecFrame[30];      // Send frame for Modbus
unsigned short ModbusSendFrame[30];     // Receive frame for Modbus
unsigned short ModbusFrameLength;       // Length of received frame

const unsigned short colon = 0x3A;      // Ascii COLON character
const unsigned short car_ret = 0x0D;    // Ascii carriage return character
const unsigned short lin_fd = 0x0A;     // Ascii line feed character
unsigned short val_msg;
//----------------------END OF SECTION-----------------------------------


/*-------------------------------------------------------------------------
```

```
FUNCTION: UART Initialisation

          Set up of serial comms.
/-------------------------------------------------------------------------*/
void modbus_init()
{
     UART1_Init(9600);              // Initialise UART1
     TXSTA.TX9 = 0;                 // 8 bit TX
     TXSTA.SYNC = 0;                // Asynchronous
     TXSTA.SENDB = 1;               // Send break character
     RCSTA.RX9 = 0;                 // 8 bit RX
     UART_Set_Active(&UART1_Read, 0, &UART1_Data_Ready, &UART1_Tx_Idle);
}
//---------------------END OF FUNCTION-------------------------------------



/*-------------------------------------------------------------------------
FUNCTION: Transmit serial

          Argument: numchar - number of characters to be transmitted
/-------------------------------------------------------------------------*/
void TX_serial(unsigned short numchar)
{
     unsigned short i;
     for (i = 0; i < numchar; i++)
     {
          while (PIR1.TX1IF == 0);      // Check for TX buffer space
          TXREG1 = ModbusSendFrame[i]; // Place next character in TX buffer
     }
     Delay_us(5);
}
//---------------------END OF FUNCTION-------------------------------------



/*-------------------------------------------------------------------------
FUNCTION: Modbus Transmit

          Argument: unsigned char* buff - message to TX
                    unsigned char length - length of message to TX
/-------------------------------------------------------------------------*/
void TX_modbus(unsigned char* buff, unsigned char length)
{
     unsigned char i;
     unsigned char new_length = length + 5;  // To allow for Start, LRC, end c
     unsigned char hex_LRC;                  // To store Hex value of LRC

     ModbusSendFrame[0] = colon;             // Insert start character :

     for (i = 1; i < length+1; i++)
     {
```

```
        ModbusSendFrame[i] = buff[i-1];
    }
    // Perform LRC and place in frame
    hex_LRC = LRC(buff, length);
    ModbusSendFrame[new_length-4] = split_first(hex_LRC);
    ModbusSendFrame[new_length-3] = split_second(hex_LRC);
    // Add carriage return and line feed to frame
    ModbusSendFrame[new_length-2] = car_ret;
    ModbusSendFrame[new_length-1] = lin_fd;
    // Transmit on serial line
    TX_serial(new_length);
}
//----------------------END OF FUNCTION-----------------------------------

/*-------------------------------------------------------------------------
FUNCTION: Serial Receive

        Argument: unsigned char* ret - buffer to RX
-------------------------------------------------------------------------*/
void RX_Serial(unsigned char *ret)
{
unsigned short c, k;
static unsigned short go = 0;

   int p;
   c = UART1_Read();
   if (RCSTA1 & 0x06)     // Check for frame error or overrun error
     {
     RCSTA.CREN = 0;
     c = RCREG;
     RCSTA.CREN = 1;      //if overrun or frame reset receiver
     }
   else
     {
     if (go)
       {
       SerialBuffer[SerialCount] = c;          //read bit
       SerialCount++;                          //inc counter
       if(c == lin_fd)                         //test if frame full
         {
         for(k = 0; k < SerialCount; k++)      //Swap buffer into modbus frame
         {
           ret[k+1] = SerialBuffer[k];         //copy byte
         }
         ret[0] = SerialCount;
         go = 0;
         MBReceived = 1;                       //Flag that modbus frame recived
         SerialCount = 0;                      //reset the serial bit counter
         }
       }
     else
       {
```

```
        if (c == colon)
          {
          SerialBuffer[SerialCount] = c;     //read bit
          SerialCount++;                     //inc counter
          go = 1;
          }
        }
      }
}
//----------------------END OF FUNCTION-----------------------------------------



/*-------------------------------------------------------------------------
FUNCTION: Modbus Receive

          Argument: unsigned char* ret - message to RX
                    unsigned short length - length of message to RX
-------------------------------------------------------------------------*/
void RX_modbus(unsigned char *ret, unsigned short length)
{
      unsigned short p;
      unsigned short g;
      unsigned short rx_lrc;
      unsigned char hold_arr[30];
      unsigned short new_len = length - 5;

      for (p = 0; p < new_len + 1; p++)   // Transfer to holding array
      {
         hold_arr[p] = ascii_hex(ret[p+1]);
      }
      // Extract LRC from frame
      rx_lrc = comb_digits(ascii_hex(ret[length - 3]), ascii_hex(ret[length - 2
      for (g = 0; g < new_len; g++)
      {
            ret[g] = hold_arr[g+1];      // Transfer data only back to frame ar
      }
      if (LRC(ret, new_len) != rx_lrc)  // Check for LRC
         ret[g] == 0xFF;                 // LRC error
}
//----------------------END OF FUNCTION-----------------------------------------


/*-------------------------------------------------------------------------
FUNCTION: Longitudinal Redunancy Check

          Argument: unsigned short numchar - number of characters in frame
                    unsigned char* buffer - frame to be LRC
-------------------------------------------------------------------------*/
unsigned short LRC(unsigned char* buffer, unsigned short numchar)
{
```

```
    unsigned short sum = 0;
    unsigned short i = 0;

    for (i = 0; i < numchar; i++)
    {
        sum = sum + buffer[i];                    // Sum frame
    }
    return (~(sum | 0b1111111100000000)) + 1;   // Perform 2s compliment
}
//---------------------END OF FUNCTION-----------------------------------
```

D.3 Ancillary Functions

This listing contains the ancillary functions used in the network bridge.

Listing D.3: Ancillary Functions.

```
/*-------------------------------------------------------------------------
-------------------------------------------------------------------------

Gas Detection Australia
CANbus to Modbus Network Bridge
BENH Final Year Project

Matthew Quinton
0061024766

Non communication functions
PIC: 18F87K22
V1.0D
-------------------------------------------------------------------------*/
//-------------------------------------------------------------------------
//----------------------END OF HEADING-----------------------------------

#include "ProjectDevBoard.h"

/*-------------------------------------------------------------------------
FUNCTION: Hex to Ascii

          Argument: hex char to convert
          Return: ascii value of character  (0 if error)
-------------------------------------------------------------------------*/
unsigned short hex_ascii(unsigned short digit)
{
    if (digit >= 0 && digit <=9)
       return digit + 0x30;
    else if (digit > 9 && digit < 16)
       return digit + 0x37;
    else
       return 0;
}

//----------------------END OF FUNCTION----------------------------------


/*-------------------------------------------------------------------------
FUNCTION: Ascii to hex

          Argument: ascii char to convert
          Return: hex value of character (0 if error)
-------------------------------------------------------------------------*/

unsigned short ascii_hex(unsigned digit)
```

```
{
    if (digit >= 0x30 && digit <=0x39)
       return digit - 0x30;
    else if (digit > 0x40 && digit < 0x47 )
       return digit - 0x37;
    else
       return 0;
}
//----------------------END OF FUNCTION-----------------------------------


/*-------------------------------------------------------------------------
FUNCTION: Split digits first

          Argument: hex number where first digit is to be extracted
          Return: extraced first digit
-------------------------------------------------------------------------*/

unsigned short split_first(unsigned short hex_dig)
{
  return hex_dig >> 4;
}
//----------------------END OF FUNCTION-----------------------------------

/*-------------------------------------------------------------------------
FUNCTION: Split digits second

          Argument: hex number where second digit is to be extracted
          Return: extraced second digit
-------------------------------------------------------------------------*/
unsigned short split_second(unsigned short hex_dig)
{
  return hex_dig & 0b00001111;
}
//----------------------END OF FUNCTION-----------------------------------

/*-------------------------------------------------------------------------
FUNCTION: Combine digits

          Argument: two digits to be combined
          Return: value of digits combined
-------------------------------------------------------------------------*/

unsigned short comb_digits(unsigned short dig1, unsigned short dig2)
{
 unsigned int a = 0;
 a = dig1 << 4;
 a = a | dig2;
  return a;
}
//----------------------END OF FUNCTION-----------------------------------
```

D.4 Main Function

This listing contains the main function. Note that the CAN functionality are not a separate listing and appear within this listing.

Listing D.4: Main Program.

```
/*-----------------------------------------------------------------------------
-----------------------------------------------------------------------------

Gas Detection Australia
CANbus to Modbus Network Bridge
BENH Final Year Project

Matthew Quinton
0061024766

Main Program
PIC: 18F87K22
V1.0D
-----------------------------------------------------------------------------*/
//-----------------------------------------------------------------------------
//-----------------------END OF HEADING----------------------------------------

#include "ProjectDevBoard.h"


unsigned char Can_Init_Flags, Can_Send_Flags, Can_Rcv_Flags; // can flags
unsigned char Rx_Data_Len;                                   // received data
char RxTx_Data[8];                                           // can rx/tx data
char Msg_Rcvd;                                               // reception flag
const long ID_1st = 12111, ID_2nd = 3;                       // node IDs
long Rx_ID;
unsigned short new_data;
unsigned char RX_M_data[30];
unsigned char RX_S_data[30];
unsigned short RX_S_Len;
unsigned char i;

// CANSPI module connections
sbit CanSpi_CS           at  RE0_bit;
sbit CanSpi_CS_Direction at  TRISE0_bit;
sbit CanSpi_Rst          at  RC0_bit;
sbit CanSpi_Rst_Direction at TRISC0_bit;
// End CANSPI module connections


// Uart interrupt service routine
void interrupt()
{
     if (PIR1.RC1IF)
```

```
    {
        RX_Serial(RX_S_data);
        PIR1.RC1IF = 0;
        new_data = 1;
    }

}

void main() {

  // Configure AN pins as digital I/O
  ANCON0 = 0;
  ANCON1 = 0;
  ANCON2 = 0;

  // Enable interrupts
  INTCON.GIE = 1;
  INTCON.PEIE = 1;
  PIE1.RC1IE = 1;

  // Clear flags
  Can_Init_Flags = 0;
  Can_Send_Flags = 0;
  Can_Rcv_Flags  = 0;

  // Confgire CAN flags
  Can_Send_Flags = _CANSPI_TX_PRIORITY_0 &
                   _CANSPI_TX_STD_FRAME &
                   _CANSPI_TX_NO_RTR_FRAME;

  Can_Init_Flags = _CANSPI_CONFIG_SAMPLE_THRICE &
                   _CANSPI_CONFIG_PHSEG2_PRG_ON &
                   _CANSPI_CONFIG_STD_MSG &
                   _CANSPI_CONFIG_DBL_BUFFER_ON &
                   _CANSPI_CONFIG_ALL_VALID_MSG;

  // Initialse communications
  // Serial
  modbus_init();
  Delay_ms(100);
  // SPI
  SPI1_Init();
  Delay_ms(100);

  // Clear Uart received flag
  new_data = 0;

  // CAN Initialisation
  CANSPIInitialize(1,5,3,3,1,Can_Init_Flags);    // Initialize external CANSPI
  CANSPISetOperationMode(_CANSPI_MODE_CONFIG,0xFF);
  CANSPISetMask(_CANSPI_MASK_B1,-1,_CANSPI_CONFIG_STD_MSG);
  CANSPISetMask(_CANSPI_MASK_B2,-1,_CANSPI_CONFIG_STD_MSG);
```

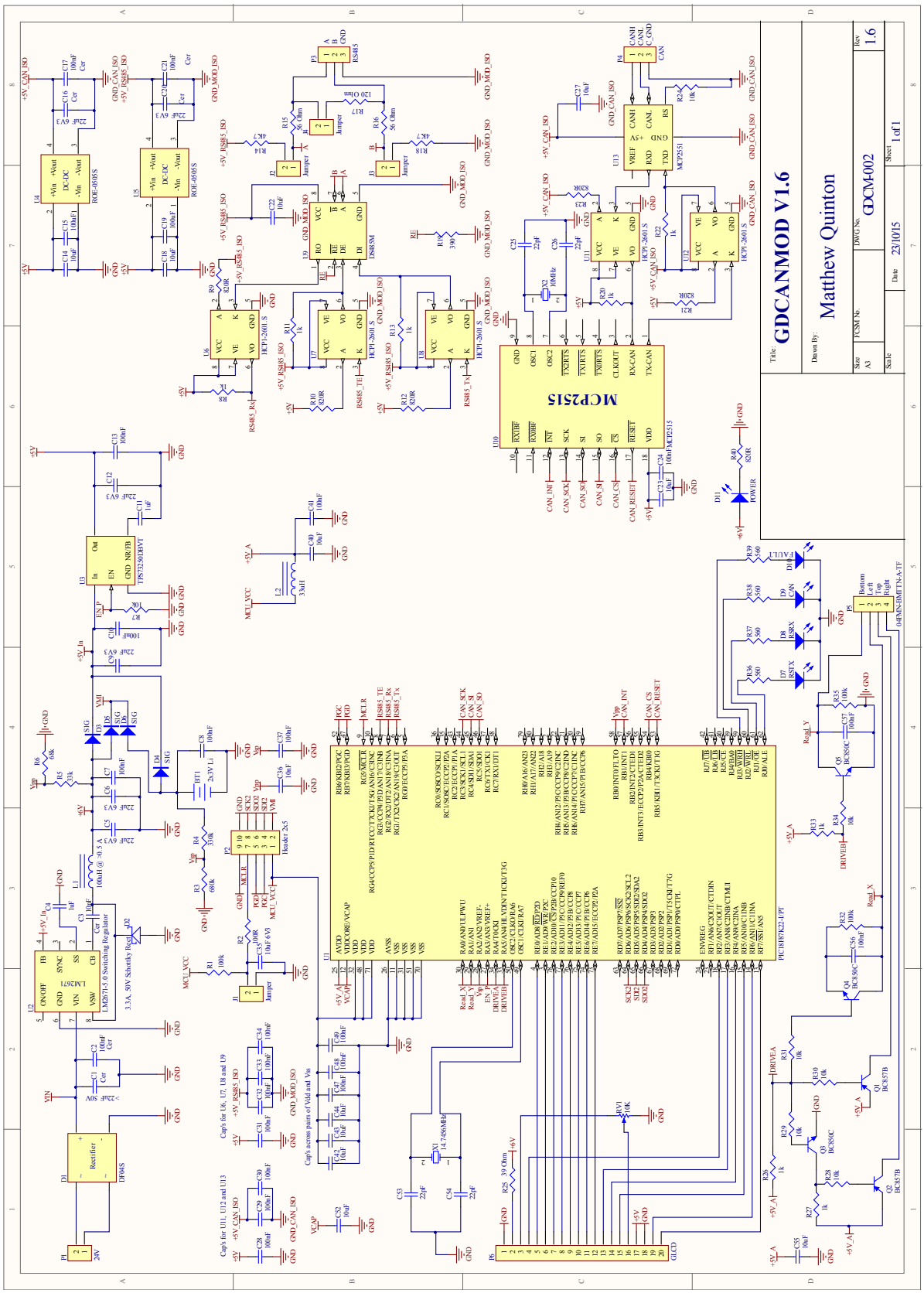
```
  CANSPISetFilter(_CANSPI_FILTER_B2_F4,ID_2nd,_CANSPI_CONFIG_STD_MSG);
  // set NORMAL mode after initialisation
  CANSPISetOperationMode(_CANSPI_MODE_NORMAL,0xFF);

  // Infinite loop to process messages to RX and TX
     while(1)
     {
      // Check for CAN received
      Msg_Rcvd = CANSPIRead(Rx_ID , RxTx_Data , Rx_Data_Len, Can_Rcv_Flags);
      if (Msg_Rcvd == 0xFF)
      {
        // If CAN received send on modbus
        TX_modbus(RxTx_Data,6);
        Msg_Rcvd = 0;
      }
      if (new_data)
      {
        // Process here if UART interrupt
        RX_S_Len = RX_S_data[0];
        RX_modbus(RX_S_data, RX_S_Len);
        // Write Modbus data to CAN
        CANSPIWrite(ID_1st, RX_S_data, RX_S_Len-5, Can_Send_Flags);
        new_data = 0;
      }
     }
}
```

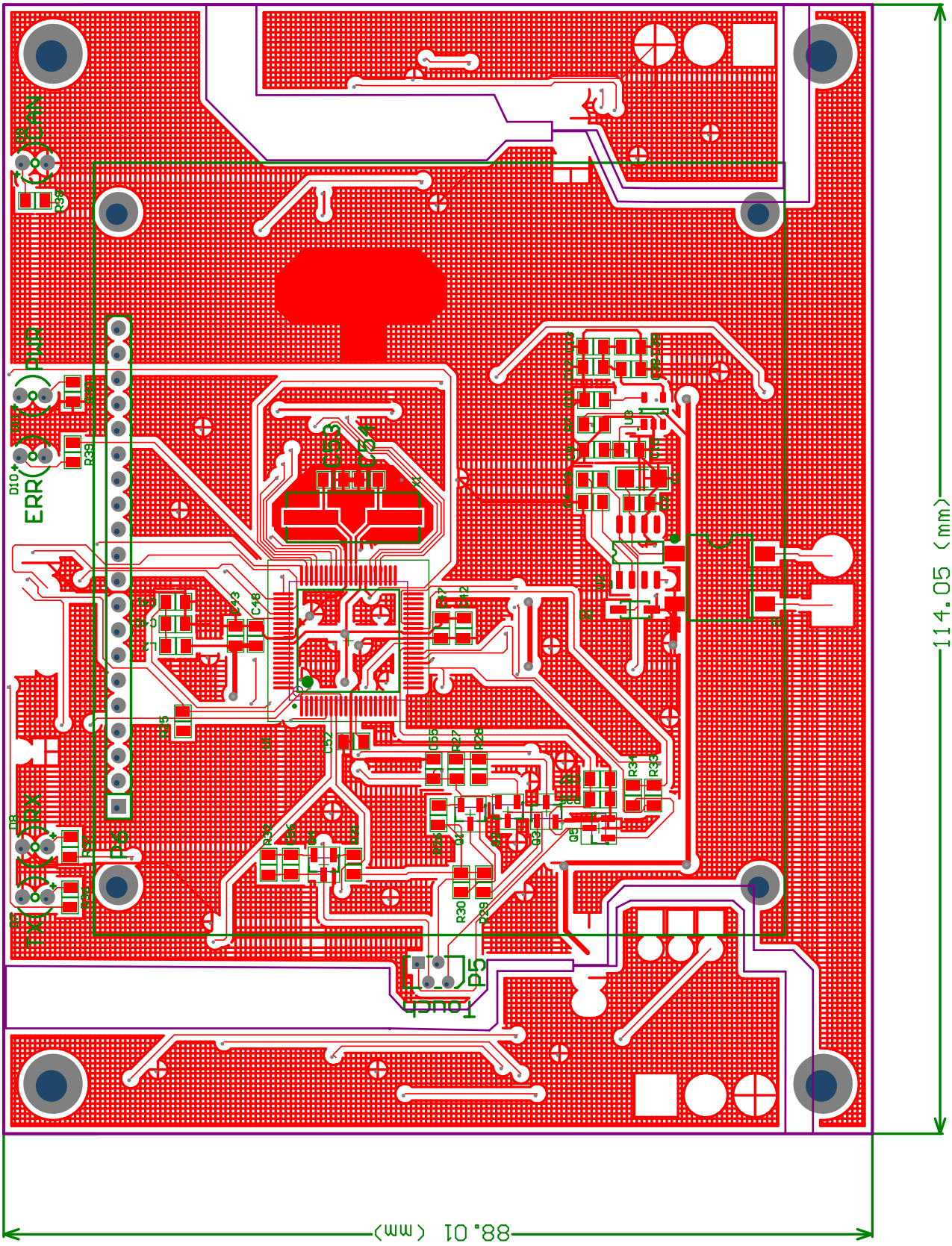
Appendix E

Hardware Design Images

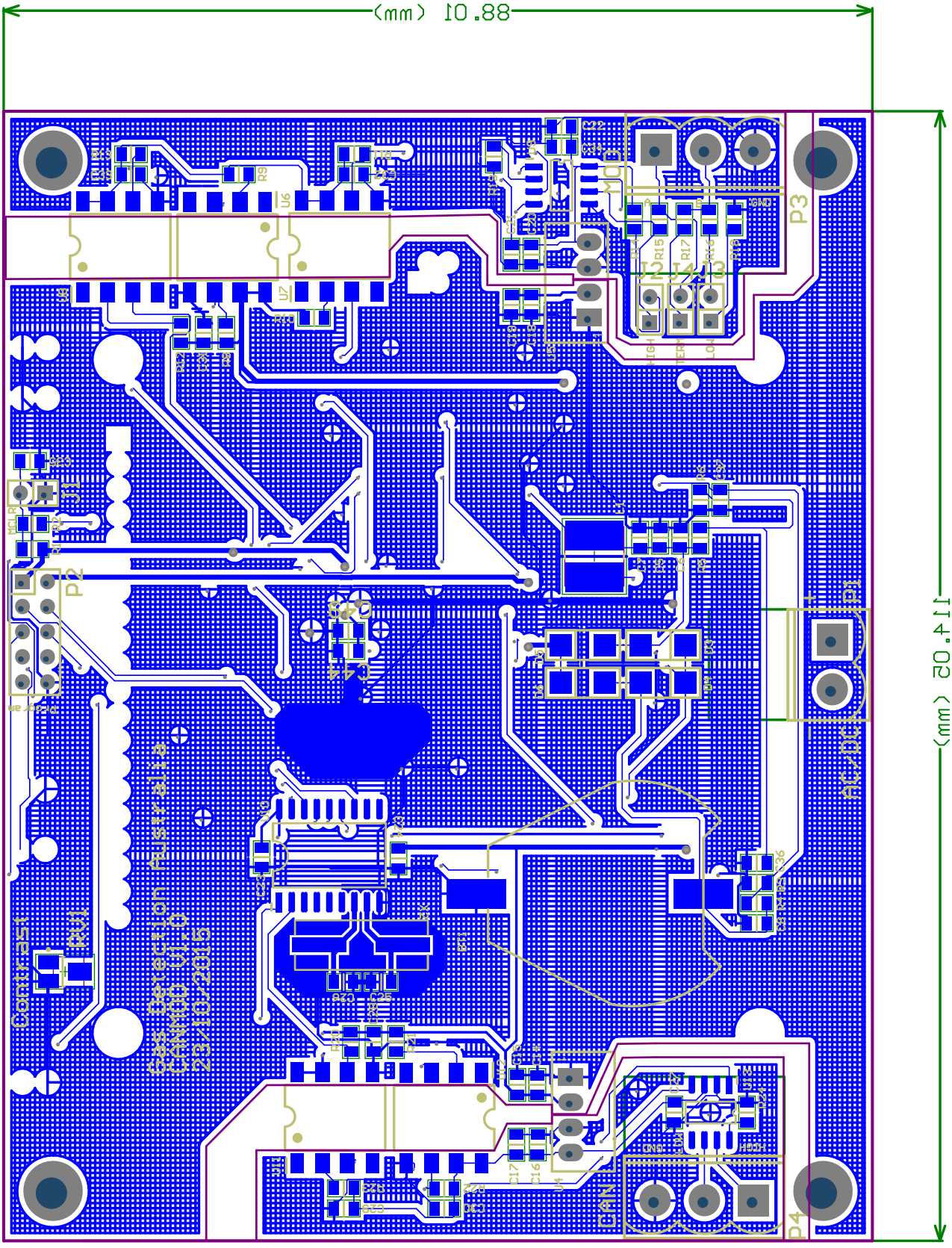
E.1 Schematic



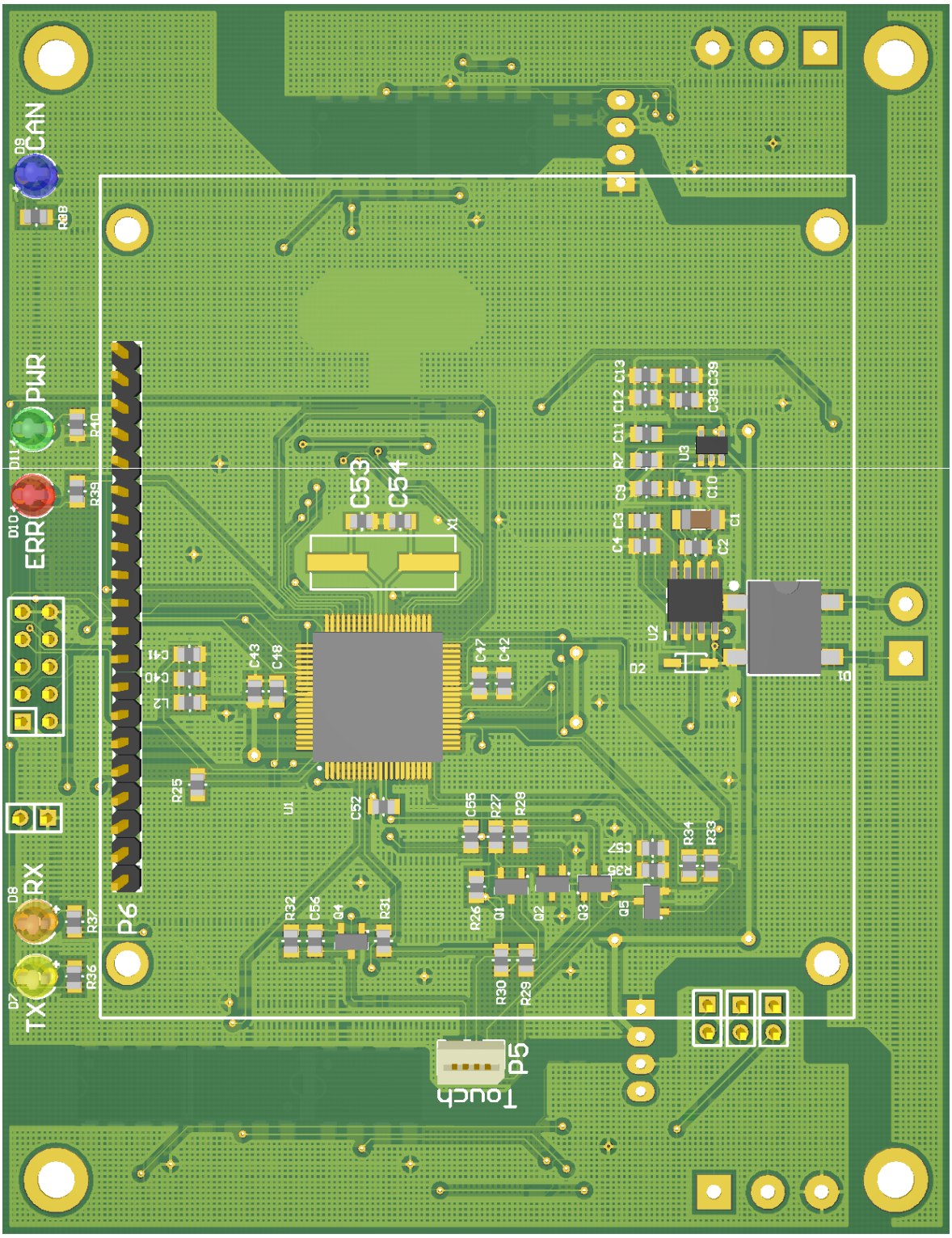
E.2 PCB Top Layer



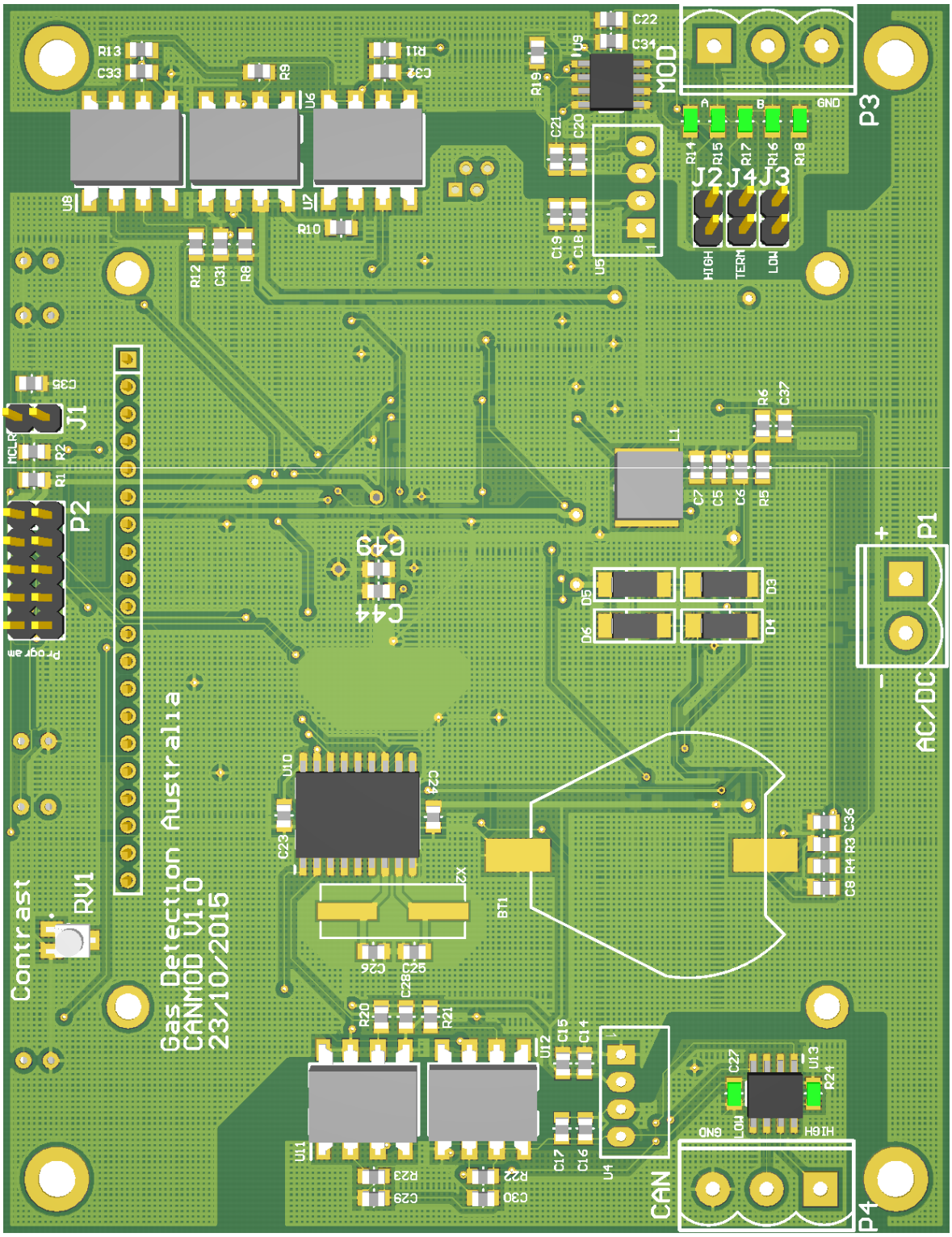
E.3 PCB Bottom Layer



E.4 PCB 3D Top Layer



E.5 PCB 3D Bottom Layer



Appendix F

Results and Discussion

Supporting Information

Put all the stuff here

F.1 Click Board Schematics

This is the schematics of the click boards used in testing. They can be compared to the hardware design of the bridge.

RS-485 Click Board (MikroeElektronika 2012*b*).

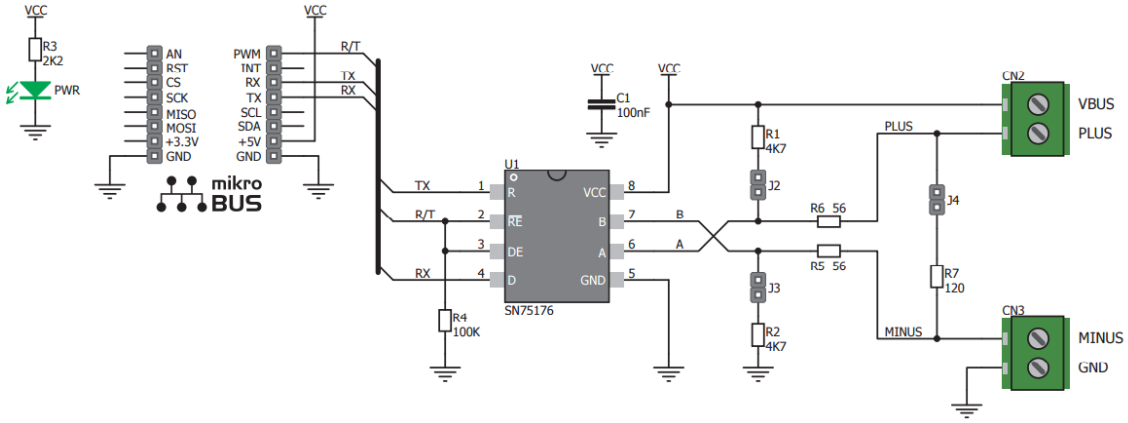


Figure F.1: RS-485 Click board schematic

CANSpi Click Board (MikroeElektronika 2012*a*).

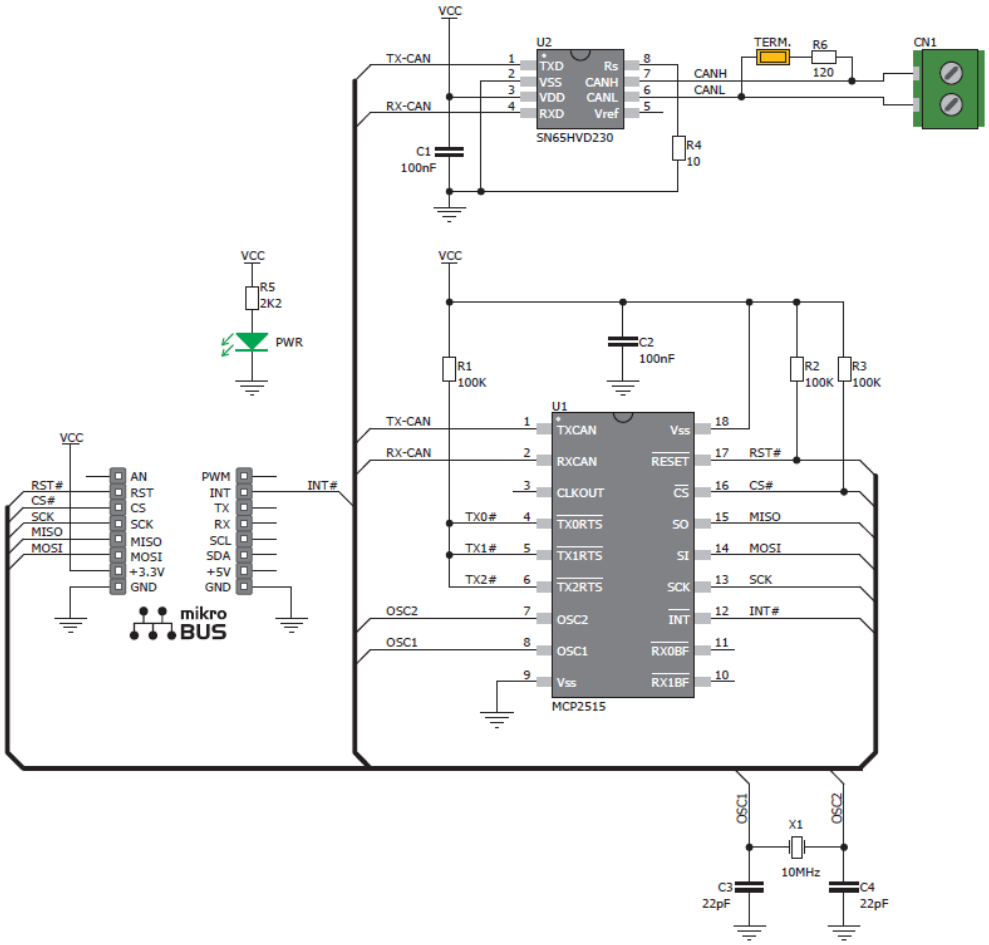


Figure F.2: CANSpi Click board schematic

F.2 Photos

RS-485 physical layer test image.

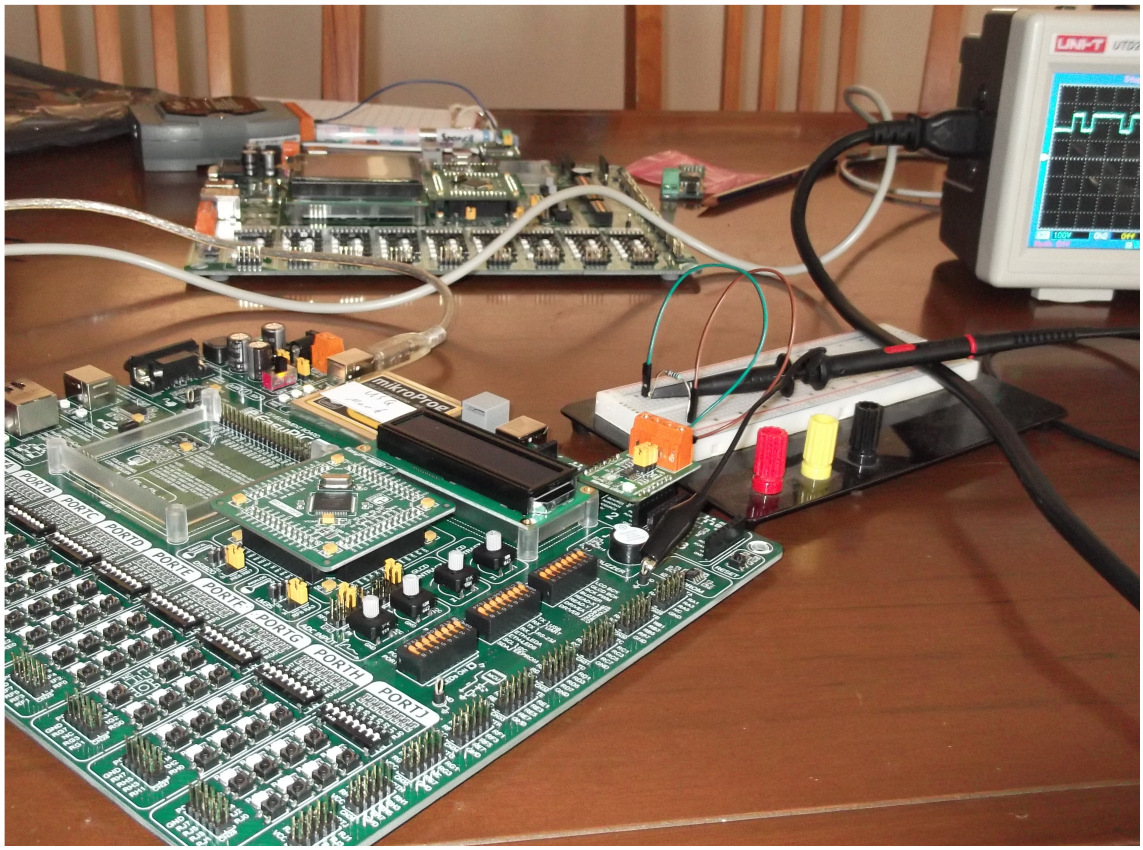


Figure F.3: RS-485 testing set-up

F.3 Code Listings

The following code was used in an attempt to diagnose the error with CAN transmission from the USB-CAN module to the development board with the bridge source code. It was not successful, however it would be ideal to further explore this as a debugging option.

Listing F.1: CANSPI Error Fetch.

```
// Code appended to ProjectDevBoard.c
// Intended to grab the error code from the CANSPI chip
// And display it on Port F LEDs of the development board

Msg_Rcvd = 0;
Delay_us(100);
CanSpi_CS = 0;
Delay_us(100);
SPI1_Write(0b10100000;
SPI1_Read(Msg_Rcvd);
LATF = Msg_Rcvd;
CanSpi_CS = 1;
Delay_us(200);
Msg_Rcvd = 0;
```

Appendix G

Datasheets

G.1 PIC18F87K22

MICROCHIP

PIC18F87K22 FAMILY

64/80-Pin, High-Performance, 1-Mbit Enhanced Flash MCUs with 12-Bit A/D and nanoWatt XLP Technology

Low-Power Features:

- Power-Managed modes:
  - Run: CPU on, peripherals on
  - Idle: CPU off, peripherals on
  - Sleep: CPU off, peripherals off
- Two-Speed Oscillator Start-up
- Fail-Safe Clock Monitor
- Power-Saving Peripheral Module Disable (PMD)
- Ultra Low-Power Wake-up
- Fast Wake-up, 1 μs Typical
- Low-Power WDT, 300 nA Typical
- Ultra Low 50 nA Input Leakage
- Run mode Currents Down to 5.5 μA, Typical
- Idle mode Currents Down to 1.7 μA Typical
- Sleep mode Currents Down to Very Low 20 nA, Typical
- RTCC Current Downs to Very Low 700 nA, Typical

Special Microcontroller Features:

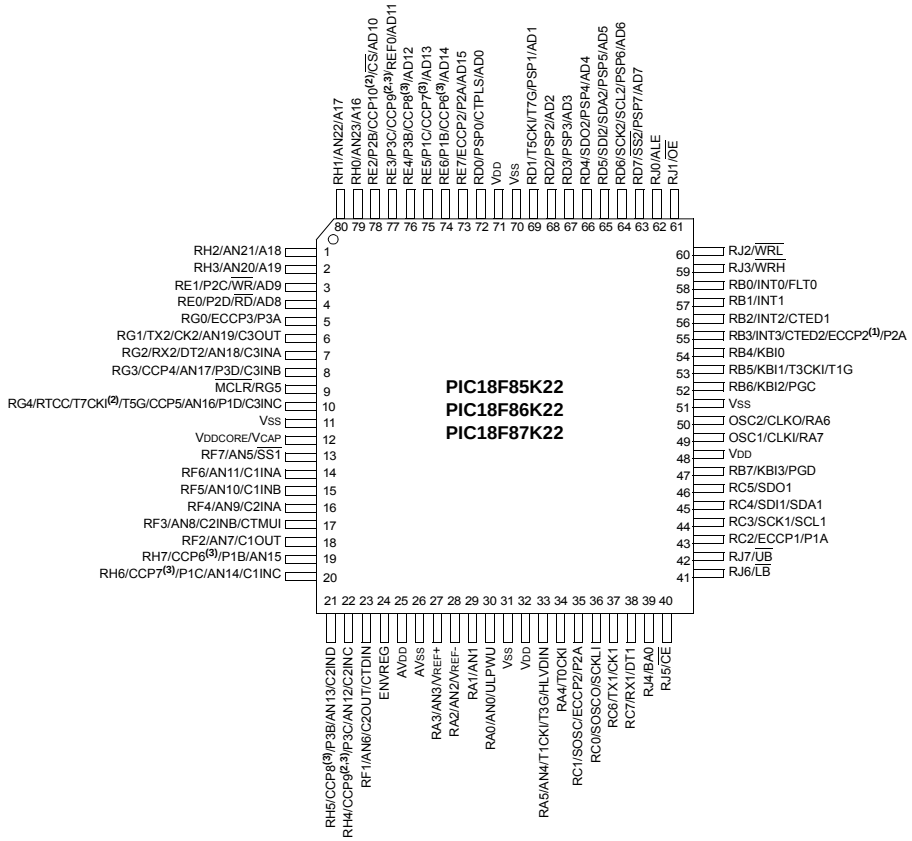
- Operating Voltage Range: 1.8V to 5.5V
- On-Chip 3.3V Regulator
- Operating Speed up to 64 MHz
- Up to 128 Kbytes On-Chip Flash Program Memory
- Data EEPROM of 1,024 Bytes
- 4K x 8 General Purpose Registers (SRAM)
- 10,000 Erase/Write Cycle Flash Program Memory, Minimum
- 1,000,000 Erase/write Cycle Data EEPROM Memory, Typical
- Flash Retention: 40 Years, Minimum
- Three Internal Oscillators: LF-INTRC (31 kHz), MF-INTOSC (500 kHz) and HF-INTOSC (16 MHz)
- Self-Programmable under Software Control
- Priority Levels for Interrupts
- 8 x 8 Single-Cycle Hardware Multiplier
- Extended Watchdog Timer (WDT):
  - Programmable period from 4 ms to 4,194s (about 70 minutes)
- In-Circuit Serial Programming™ (ICSP™) via Two Pins
- In-Circuit Debug via Two Pins
- Programmable:
  - BOR
  - LVD

Device	Program Memory		Data Memory		I/O	12-Bit A/D (ch)	CCP/ ECCP (PWM)	MSSP			EUSART	Comparators	Timers 8/16-Bit	External Bus	CTMU	RTCC
	Flash (bytes)	# Single-Word Instructions	SRAM (bytes)	EEPROM (bytes)					SPI	Master I^2C™						
PIC18F65K22	32K	16,383	2K	1K	53	16	5/3	2	Y	Y	2	3	4/4	N	Y	Y
PIC18F66K22	64K	32,768	4K	1K	53	16	7/3	2	Y	Y	2	3	6/5	N	Y	Y
PIC18F67K22	128K	65,536	4K	1K	53	16	7/3	2	Y	Y	2	3	6/5	N	Y	Y
PIC18F85K22	32K	16,383	2K	1K	69	24	5/3	2	Y	Y	2	3	4/4	Y	Y	Y
PIC18F86K22	64K	32,768	4K	1K	69	24	7/3	2	Y	Y	2	3	6/5	Y	Y	Y
PIC18F87K22	128K	65,536	4K	1K	69	24	7/3	2	Y	Y	2	3	6/5	Y	Y	Y

PIC18F87K22 FAMILY

Pin Diagrams – PIC18F8XK22

80-Pin TQFP



Note 1: The ECCP2 pin placement depends on the CCP2MX Configuration bit setting and whether the device is in Microcontroller or Extended Microcontroller mode.

2: Not available on the PIC18F65K22 and PIC18F85K22 devices.

3: The CC6, CCP7, CCP8 and CCP9 pin placement depends on the setting of the ECCPMX Configuration bit (CONFIG3H<1>).

2.0 GUIDELINES FOR GETTING STARTED WITH PIC18FXXKXX MICROCONTROLLERS

2.1 Basic Connection Requirements

Getting started with the PIC18F87K22 family family of 8-bit microcontrollers requires attention to a minimal set of device pin connections before proceeding with development.

The following pins must always be connected:

- All VDD and VSS pins (see **Section 2.2 "Power Supply Pins"**)
- All AVDD and AVSS pins, regardless of whether or not the analog device features are used (see **Section 2.2 "Power Supply Pins"**)
- $\overline{\text{MCLR}}$ pin (see **Section 2.3 "Master Clear (MCLR) Pin"**)
- ENVREG (if implemented) and VCAP/VDDCORE pins (see **Section 2.4 "Voltage Regulator Pins (ENVREG and VCAP/VDDCORE)"**)

These pins must also be connected if they are being used in the end application:

- PGC/PGD pins used for In-Circuit Serial Programming™ (ICSP™) and debugging purposes (see **Section 2.5 "ICSP Pins"**)
- OSCI and OSCO pins when an external oscillator source is used (see **Section 2.6 "External Oscillator Pins"**)

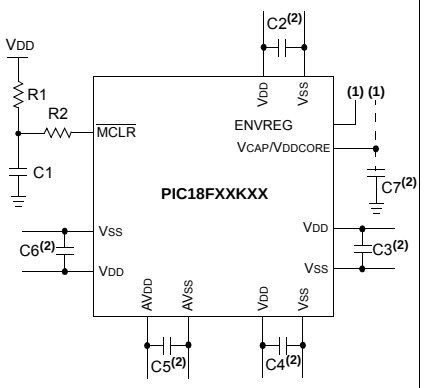
Additionally, the following pins may be required:

- VREF+/VREF- pins are used when external voltage reference for analog modules is implemented

Note: The AVDD and AVSS pins must always be connected, regardless of whether any of the analog modules are being used.

The minimum mandatory connections are shown in Figure 2-1.

FIGURE 2-1: RECOMMENDED MINIMUM CONNECTIONS



Key (all values are recommendations):

C1 through C6: 0.1 µF, 20V ceramic

R1: 10 kΩ

R2: 100Ω to 470Ω

Note 1: See **Section 2.4 "Voltage Regulator Pins (ENVREG and VCAP/VDDCORE)"** for explanation of ENVREG pin connections.

2: The example shown is for a PIC18F device with five VDD/VSS and AVDD/AVSS pairs. Other devices may have more or less pairs; adjust the number of decoupling capacitors appropriately.

PIC18F87K22 FAMILY

2.2 Power Supply Pins

2.2.1 DECOUPLING CAPACITORS

The use of decoupling capacitors on every pair of power supply pins, such as VDD, VSS, AVDD and AVSS, is required.

Consider the following criteria when using decoupling capacitors:

- **Value and type of capacitor:** A 0.1 μF (100 nF), 10-20V capacitor is recommended. The capacitor should be a low-ESR device, with a resonance frequency in the range of 200 MHz and higher. Ceramic capacitors are recommended.
- **Placement on the printed circuit board:** The decoupling capacitors should be placed as close to the pins as possible. It is recommended to place the capacitors on the same side of the board as the device. If space is constricted, the capacitor can be placed on another layer on the PCB using a via; however, ensure that the trace length from the pin to the capacitor is no greater than 0.25 inch (6 mm).
- **Handling high-frequency noise:** If the board is experiencing high-frequency noise (upward of tens of MHz), add a second ceramic type capacitor in parallel to the above described decoupling capacitor. The value of the second capacitor can be in the range of 0.01 μF to 0.001 μF. Place this second capacitor next to each primary decoupling capacitor. In high-speed circuit designs, consider implementing a decade pair of capacitances as close to the power and ground pins as possible (e.g., 0.1 μF in parallel with 0.001 μF).
- **Maximizing performance:** On the board layout from the power supply circuit, run the power and return traces to the decoupling capacitors first, and then to the device pins. This ensures that the decoupling capacitors are first in the power chain. Equally important is to keep the trace length between the capacitor and the power pins to a minimum, thereby reducing PCB trace inductance.

2.2.2 TANK CAPACITORS

On boards with power traces running longer than six inches in length, it is suggested to use a tank capacitor for integrated circuits, including microcontrollers, to supply a local power source. The value of the tank capacitor should be determined based on the trace resistance that connects the power supply source to the device, and the maximum current drawn by the device in the application. In other words, select the tank capacitor so that it meets the acceptable voltage sag at the device. Typical values range from 4.7 μF to 47 μF.

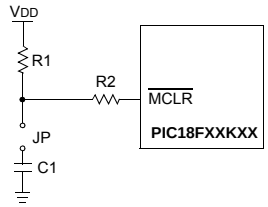
2.3 Master Clear ($\overline{\text{MCLR}}$) Pin

The $\overline{\text{MCLR}}$ pin provides two specific device functions: Device Reset, and Device Programming and Debugging. If programming and debugging are not required in the end application, a direct connection to VDD may be all that is required. The addition of other components, to help increase the application's resistance to spurious Resets from voltage sags, may be beneficial. A typical configuration is shown in Figure 2-1. Other circuit designs may be implemented, depending on the application's requirements.

During programming and debugging, the resistance and capacitance that can be added to the pin must be considered. Device programmers and debuggers drive the $\overline{\text{MCLR}}$ pin. Consequently, specific voltage levels (VIH and VIL) and fast signal transitions must not be adversely affected. Therefore, specific values of R1 and C1 will need to be adjusted based on the application and PCB requirements. For example, it is recommended that the capacitor, C1, be isolated from the $\overline{\text{MCLR}}$ pin during programming and debugging operations by using a jumper (Figure 2-2). The jumper is replaced for normal run-time operations.

Any components associated with the $\overline{\text{MCLR}}$ pin should be placed within 0.25 inch (6 mm) of the pin.

FIGURE 2-2: EXAMPLE OF $\overline{\text{MCLR}}$ PIN CONNECTIONS



Note 1: R1 ≤ 10 kΩ is recommended. A suggested starting value is 10 kΩ. Ensure that the $\overline{\text{MCLR}}$ pin VIH and VIL specifications are met.

2: R2 ≤ 470Ω will limit any current flowing into $\overline{\text{MCLR}}$ from the external capacitor, C, in the event of $\overline{\text{MCLR}}$ pin breakdown, due to Electrostatic Discharge (ESD) or Electrical Overstress (EOS). Ensure that the $\overline{\text{MCLR}}$ pin VIH and VIL specifications are met.

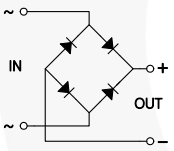
G.2 DF04S

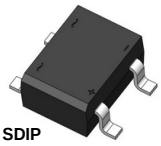
May 2015

DF005S - DF10S
Bridge Rectifiers

Features

- Maximum Surge Rating: I_{FSM} = 50 A
  I^2t = 10 A^2Sec
- Optimized V_F: Typical 0.94 V at 1.5 A, 25°C
- Glass Passivated Junctions
- Lead Free Compliant to EU RoHS 2002/95/EU Directives
- Green Molding Compound: IEC61249
- Qualified with IR Reflow and Wave Soldering
- UL Certified, UL #E258596





Description

With the ever-pressing need to improve power supply efficiency, improve surge rating, improve reliability, and reduce size, the DFxS family sets a standard in performance.

The design offers an surge rating of 50 A. This is important when improving reliability and increasing efficiency. High efficiency designs strive to reduce circuit resistance, which, unfortunately can result in increased inrush surge. As such high surge current ratings can be required to maintain or improve reliability.

The design also offers better efficiency by achieving a 1.5 A V_F of 1.1 V maximum at 25°C. This lower V_F also supports cooler and more efficient operation.

Finally, the DFxS achieves all this in a SDIP surface mount form factor, reducing board space and volumetric requirements vs. competitive devices.

Ordering Information

Part Number	Top Mark	Package	Packing Method
DF005S	DF005S	SDIP 4L	Tape and Reel
DF01S	DF01S	SDIP 4L	Tape and Reel
DF02S	DF02S	SDIP 4L	Tape and Reel
DF04S	DF04S	SDIP 4L	Tape and Reel
DF06S	DF06S	SDIP 4L	Tape and Reel
DF08S	DF08S	SDIP 4L	Tape and Reel
DF10S	DF10S	SDIP 4L	Tape and Reel

Absolute Maximum Ratings

Stresses exceeding the absolute maximum ratings may damage the device. The device may not function or be operable above the recommended operating conditions and stressing the parts to these levels is not recommended. In addition, extended exposure to stresses above the recommended operating conditions may affect device reliability. The absolute maximum ratings are stress ratings only. Values are at T_A = 25°C unless otherwise noted.

Symbol	Parameter	Value							Unit
		DF005S	DF01S	DF02S	DF04S	DF06S	DF08S	DF10S	
V_{RRM}	Maximum Repetitive Reverse Voltage	50	100	200	400	600	800	1000	V
V_{RMS}	Maximum RMS Bridge Input Voltage	35	70	140	280	420	560	700	V
V_{DC}	DC Reverse Voltage at Rated I_R	50	100	200	400	600	800	1000	V
$I_{F(AV)}$	Average Rectified Forward Current at T_A = 40°C	1.5							A
I_{FSM}	Non-Repetitive Peak Forward Surge Current 8.3 ms Single Half-Sine Wave	50							A
T_{STG}	Storage Temperature Range	-55 to +150							°C
T_J	Operating Junction Temperature	-55 to +150							°C

Thermal Characteristics

Values are at T_A = 25°C unless otherwise noted.

Symbol	Parameter		Value	Unit
P_D	Power Dissipation		3.1	W
$R_{\theta JA}$	Thermal Resistance, Junction-to-Ambient	Single-Die Measurement[1] (Maximum Land Pattern: 13 x 13 mm)	62	°C/W
		Multi-Die Measurement[2] (Maximum Land Pattern: 13 x 13 mm)	50	
		Multi-Die Measurement[2] (Minimum Land Pattern: 1.3 x 1.5 mm)	105	
Ψ_{JL}	Thermal Characterization Parameter, Junction to Lead	Single-Die Measurement[2] (Maximum and Minimum Land Pattern)	27	°C/W

Notes:

1. Device mounted on PCB with 0.5 inch × 0.5 inch (13 mm × 13 mm).
2. The thermal resistances ($R_{\theta JA}$ & Ψ_{JL}) are characterized with the device mounted on the following FR4 printed circuit boards, as shown in Figure 1 and Figure 2. PCB size: 76.2 x 114.3 mm.
   Heating effect from adjacent dice is considered and only two dices are powered at the same time.

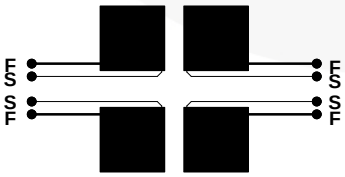


Figure 1. Maximum Pads of 2 oz Copper

Figure 2. Minimum Pads of 2 oz Copper

G.3 HCPL-2601.S

FAIRCHILD SEMICONDUCTOR®

July 2006

Single-Channel: 6N137, HCPL-2601, HCPL-2611 Dual-Channel: HCPL-2630, HCPL-2631 High Speed-10 MBit/s Logic Gate Optocouplers

Features

- Very high speed-10 MBit/s
- Superior CMR-10 kV/µs
- Double working voltage-480V
- Fan-out of 8 over -40°C to +85°C
- Logic gate output
- Strobable output
- Wired OR-open collector
- U.L. recognized (File # E90700)

Applications

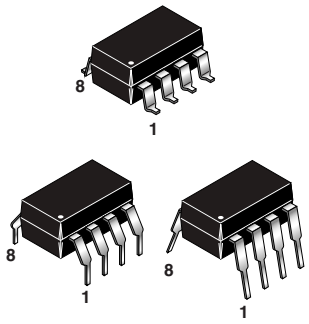
- Ground loop elimination
- LSTTL to TTL, LSTTL or 5-volt CMOS
- Line receiver, data transmission
- Data multiplexing
- Switching power supplies
- Pulse transformer replacement
- Computer-peripheral interface

Description

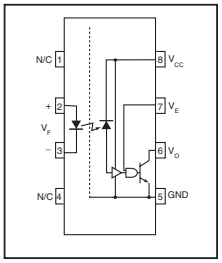
The 6N137, HCPL-2601/2611 single-channel and HCPL-2630/2631 dual-channel optocouplers consist of a 850 nm AlGaAS LED, optically coupled to a very high speed integrated photo-detector logic gate with a strobable output. This output features an open collector, thereby permitting wired OR outputs. The coupled parameters are guaranteed over the temperature range of -40°C to +85°C. A maximum input signal of 5 mA will provide a minimum output sink current of 13mA (fan out of 8).

An internal noise shield provides superior common mode rejection of typically 10kV/µs. The HCPL- 2601 and HCPL- 2631 has a minimum CMR of 5 kV/µs. The HCPL-2611 has a minimum CMR of 10 kV/µs.

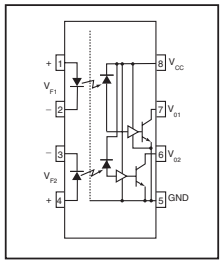
Package



Schematic



6N137
HCPL-2601
HCPL-2611



HCPL-2630
HCPL-2631

Truth Table (Positive Logic)

Input	Enable	Output
H	H	L
L	H	H
H	L	H
L	L	H
H	NC	L
L	NC	H

A 0.1µF bypass capacitor must be connected between pins 8 and 5. (See note 1)

Absolute Maximum Ratings (T_A = 25°C unless otherwise specified)

Parameter		Symbol	Value	Units
Storage Temperature		T_{STG}	-55 to +125	°C
Operating Temperature		T_{OPR}	-40 to +85	°C
Lead Solder Temperature		T_{SOL}	260 for 10 sec	°C
EMITTER				
DC/Average Forward	Single Channel	I_F	50	mA
Input Current	Dual Channel (Each Channel)		30	
Enable Input Voltage Not to exceed V_{CC} by more than 500 mV	Single Channel	V_E	5.5	V
Reverse Input Voltage	Each Channel	V_R	5.0	V
Power Dissipation	Single Channel	P_I	100	mW
	Dual Channel (Each Channel)		45	
DETECTOR				
Supply Voltage		V_{CC} (1 minute max)	7.0	V
Output Current	Single Channel	I_O	50	mA
	Dual Channel (Each Channel)		50	
Output Voltage	Each Channel	V_O	7.0	V
Collector Output	Single Channel	P_O	85	mW
Power Dissipation	Dual Channel (Each Channel)		60	

Recommended Operating Conditions

Parameter	Symbol	Min	Max	Units
Input Current, Low Level	I_{FL}	0	250	µA
Input Current, High Level	I_{FH}	*6.3	15	mA
Supply Voltage, Output	V_{CC}	4.5	5.5	V
Enable Voltage, Low Level	V_{EL}	0	0.8	V
Enable Voltage, High Level	V_{EH}	2.0	V_{CC}	V
Low Level Supply Current	T_A	-40	+85	°C
Fan Out (TTL load)	N		8	

*6.3mA is a guard banded value which allows for at least 20% CTR degradation. Initial input current threshold value is 5.0 mA or less.

Electrical Characteristics (T_A = 0 to 70°C Unless otherwise specified)
Individual Component Characteristics

Parameter		Test Conditions	Symbol	Min	Typ**	Max	Unit
EMITTER		(I_F = 10mA)	V_F			1.8	V
Input Forward Voltage		T_A = 25°C			1.4	1.75	
Input Reverse Breakdown Voltage		(I_R = 10μA)	B_{VR}	5.0			V
Input Capacitance		(V_F = 0, f = 1 MHz)	C_{IN}		60		pF
Input Diode Temperature Coefficient		(I_F = 10mA)	$\Delta V_F/\Delta T_A$		-1.4		mV/°C
DETECTOR							
High Level Supply Current	Single Channel	(V_{CC} = 5.5 V, I_F = 0 mA)	I_{CCH}		7	10	mA
	Dual Channel	(V_E = 0.5V)			10	15	
Low Level Supply Current	Single Channel	(V_{CC} = 5.5 V, I_F = 10 mA)	I_{CCL}		9	13	mA
	Dual Channel	(V_E = 0.5V)			14	21	
Low Level Enable Current		(V_{CC} = 5.5 V, V_E = 0.5V)	I_{EL}		-0.8	-1.6	mA
High Level Enable Current		(V_{CC} = 5.5 V, V_E = 2.0V)	I_{EH}		-0.6	-1.6	mA
High Level Enable Voltage		(V_{CC} = 5.5 V, I_F = 10 mA)	V_{EH}	2.0			V
Low Level Enable Voltage		(V_{CC} = 5.5 V, I_F = 10 mA)(Note 3)	V_{EL}			0.8	V

Switching Characteristics (T_A = -40°C to +85°C, V_{CC} = 5 V, I_F = 7.5 mA Unless otherwise specified)

AC Characteristics	Test Conditions	Symbol	Min	Typ**	Max	Unit
Propagation Delay Time to Output High Level	(Note 4) (T_A = 25°C)	T_{PLH}	20	45	75	ns
	(R_L = 350Ω, C_L = 15 pF) (Fig. 12)				100	
Propagation Delay Time to Output Low Level	(Note 5) (T_A = 25°C)	T_{PHL}	25	45	75	ns
	(R_L = 350Ω, C_L = 15 pF) (Fig. 12)				100	
Pulse Width Distortion	(R_L = 350Ω, C_L = 15 pF) (Fig. 12)	$\lvert T_{PHL} - T_{PLH} \rvert$		3	35	ns
Output Rise Time (10-90%)	(R_L = 350Ω, C_L = 15 pF) (Note 6) (Fig. 12)	t_r		50		ns
Output Rise Time (90-10%)	(R_L = 350Ω, C_L = 15 pF) (Note 7) (Fig. 12)	t_f		12		ns
Enable Propagation Delay Time to Output High Level	(I_F = 7.5 mA, V_{EH} = 3.5 V) (R_L = 350Ω, C_L = 15 pF) (Note 8) (Fig. 13)	t_{ELH}		20		ns
Enable Propagation Delay Time to Output Low Level	(I_F = 7.5 mA, V_{EH} = 3.5 V) (R_L = 350Ω, C_L = 15 pF) (Note 9) (Fig. 13)	t_{EHL}		20		ns
Common Mode Transient Immunity (at Output High Level)	(T_A = 25°C) $\lvert V_{CM} \rvert$ = 50V, (Peak) (I_F = 0 mA, V_{OH} (Min.) = 2.0V)	$\lvert CM_H \rvert$				V/μs
	6N137, HCPL-2630 (R_L = 350Ω) (Note 10)			10,000		
	HCPL-2601, HCPL-2631 (Fig. 14)		5000	10,000		
	HCPL-2611 $\lvert V_{CM} \rvert$ = 400V		10,000	15,000		
Common Mode Transient Immunity (at Output Low Level)	(R_L = 350Ω) (I_F = 7.5 mA, V_{OL} (Max.) = 0.8V	$\lvert CM_L \rvert$		10,000		V/μs
	6N137, HCPL-2630 $\lvert V_{CM} \rvert$ = 50V (Peak)					
	HCPL-2601, HCPL-2631 (T_A = 25°C)(Note 11)(Fig. 14)		5000	10,000		
	HCPL-2611(T_A = 25°C) $\lvert V_{CM} \rvert$ = 400V		10,000	15,000		

G.4 LM2671

LM2671 SIMPLE SWITCHER® Power Converter High Efficiency 500mA Step-Down Voltage Regulator with Features

Check for Samples: LM2671

FEATURES

- **Efficiency up to 96%**
- **Available in SOIC, 8-pin PDIP and WSON Packages**
- **Computer Design Software *LM267X Made Simple* (version 6.0)**
- **Simple and Easy to Design With**
- **Requires Only 5 external Components**
- **Uses Readily Available Standard Inductors**
- **3.3V, 5.0V, 12V, and Adjustable Output Versions**
- **Adjustable Version Output Voltage Range: 1.21V to 37V**
- **±1.5% Max Output Voltage Tolerance Over Line and Load Conditions**
- **Ensured 500mA Output Load Current**
- **0.25Ω DMOS Output Switch**
- **Wide Input Voltage Range: 8V to 40V**
- **260 kHz Fixed Frequency Internal Oscillator**
- **TTL Shutdown Capability, Low Power Standby Mode**
- **Soft-Start and Frequency Synchronization**
- **Thermal Shutdown and Current Limit Protection**

APPLICATIONS

- **Simple High Efficiency (>90%) Step-Down (Buck) Regulator**
- **Efficient Pre-Regulator for Linear Regulators**

DESCRIPTION

The LM2671 series of regulators are monolithic integrated circuits built with a LMDMOS process. These regulators provide all the active functions for a step-down (buck) switching regulator, capable of driving a 500mA load current with excellent line and load regulation. These devices are available in fixed output voltages of 3.3V, 5.0V, 12V, and an adjustable output version.

Requiring a minimum number of external components, these regulators are simple to use and include patented internal frequency compensation (Patent Nos. 5,382,918 and 5,514,947), fixed frequency oscillator, external shutdown, soft-start, and frequency synchronization.

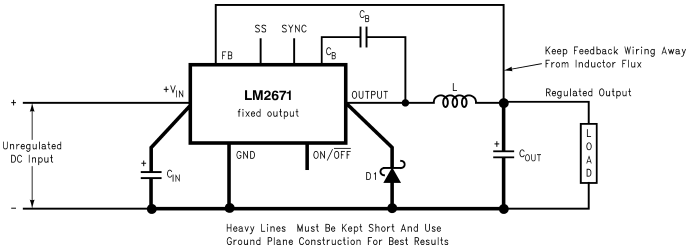
The LM2671 series operates at a switching frequency of 260 kHz, thus allowing smaller sized filter components than what would be needed with lower frequency switching regulators. Because of its very high efficiency (>90%), the copper traces on the printed circuit board are the only heat sinking needed.

A family of standard inductors for use with the LM2671 are available from several different manufacturers. This feature greatly simplifies the design of switch-mode power supplies using these advanced ICs. Also included in the datasheet are selector guides for diodes and capacitors designed to work in switch-mode power supplies.

Other features include a ensured ±1.5% tolerance on output voltage within specified input voltages and output load conditions, and ±10% on the oscillator frequency. External shutdown is included, featuring typically 50 µA stand-by current. The output switch includes current limiting, as well as thermal shutdown for full protection under fault conditions.

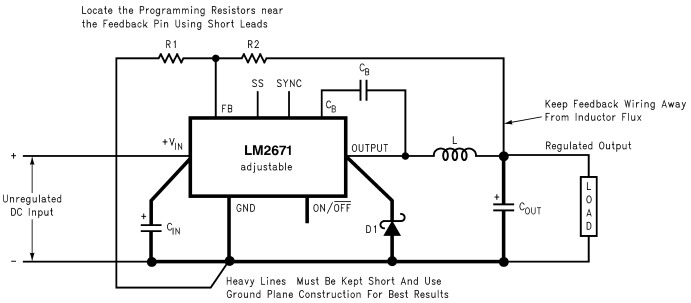
To simplify the LM2671 buck regulator design procedure, there exists computer design software, ***LM267X Made Simple*** (version 6.0).

TEST CIRCUIT AND LAYOUT GUIDELINES



C_{IN} - 22 µF, 50V Tantalum, Sprague “199D Series”
C_{OUT} - 47 µF, 25V Tantalum, Sprague “595D Series”
D1 - 3.3A, 50V Schottky Rectifier, IR 30WQ05F
L1 - 68 µH Sumida #RCR110D-680L
C_B - 0.01 µF, 50V Ceramic

Figure 22. Standard Test Circuits and Layout Guides
Fixed Output Voltage Versions



C_{IN} - 22 µF, 50V Tantalum, Sprague “199D Series”
C_{OUT} - 47 µF, 25V Tantalum, Sprague “595D Series”
D1 - 3.3A, 50V Schottky Rectifier, IR 30WQ05F
L1 - 68 µH Sumida #RCR110D-680L
R1 - 1.5 kΩ, 1%
C_B - 0.01 µF, 50V Ceramic
For a 5V output, select R2 to be 4.75 kΩ, 1%

$$V_{OUT} = V_{REF} \left(1 + \frac{R_2}{R_1}\right)$$

where V_{REF} = 1.21V

$$R_2 = R_1 \left(\frac{V_{OUT}}{V_{REF}} - 1\right)$$

Use a 1% resistor for best stability.

Figure 23. Standard Test Circuits and Layout Guides
Adjustable Output Voltage Versions

Application Hints

The LM2671 provides all of the active functions required for a step-down (buck) switching regulator. The internal power switch is a DMOS power MOSFET to provide power supply designs with high current capability, up to 0.5A, and highly efficient operation.

G.5 MCP2515

MCP2515

Stand-Alone CAN Controller with SPI Interface

Features:

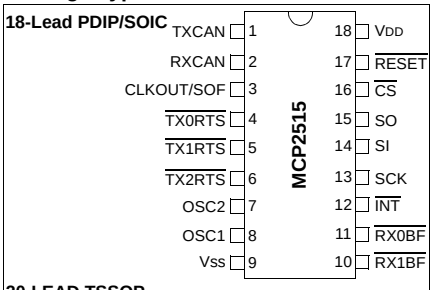
- Implements CAN V2.0B at 1 Mb/s:
  - 0 – 8 byte length in the data field
  - Standard and extended data and remote frames
- Receive Buffers, Masks and Filters:
  - Two receive buffers with prioritized message storage
  - Six 29-bit filters
  - Two 29-bit masks
- Data Byte Filtering on the First Two Data Bytes (applies to standard data frames)
- Three Transmit Buffers with Prioritization and Abort Features
- High-Speed SPI Interface (10 MHz):
  - SPI modes 0,0 and 1,1
- One-Shot mode Ensures Message Transmission is Attempted Only One Time
- Clock Out Pin with Programmable Prescaler:
  - Can be used as a clock source for other device(s)
- Start-of-Frame (SOF) Signal is Available for Monitoring the SOF Signal:
  - Can be used for time-slot-based protocols and/or bus diagnostics to detect early bus degradation
- Interrupt Output Pin with Selectable Enables
- Buffer Full Output Pins Configurable as:
  - Interrupt output for each receive buffer
  - General purpose output
- Request-to-Send (RTS) Input Pins Individually Configurable as:
  - Control pins to request transmission for each transmit buffer
  - General purpose inputs
- Low-Power CMOS Technology:
  - Operates from 2.7V – 5.5V
  - 5 mA active current (typical)
  - 1 µA standby current (typical) (Sleep mode)
- Temperature Ranges Supported:
  - Industrial (I): -40°C to +85°C
  - Extended (E): -40°C to +125°C

Description

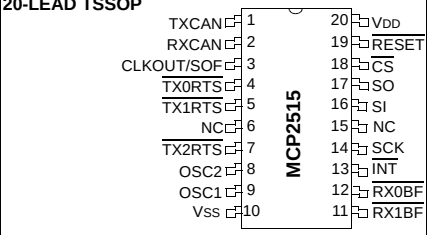
Microchip Technology's MCP2515 is a stand-alone Controller Area Network (CAN) controller that implements the CAN specification, version 2.0B. It is capable of transmitting and receiving both standard and extended data and remote frames. The MCP2515 has two acceptance masks and six acceptance filters that are used to filter out unwanted messages, thereby reducing the host MCU's overhead. The MCP2515 interfaces with microcontrollers (MCUs) via an industry standard Serial Peripheral Interface (SPI).

Package Types

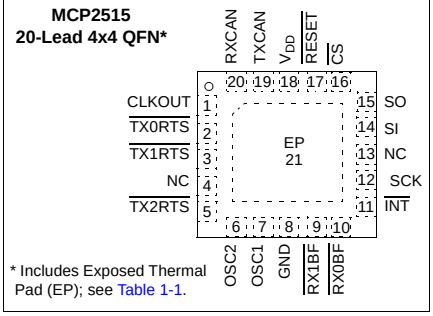
18-Lead PDIP/SOIC



20-LEAD TSSOP



MCP2515
20-Lead 4x4 QFN*



* Includes Exposed Thermal Pad (EP); see Table 1-1.

5.0 BIT TIMING

All nodes on a given CAN bus must have the same nominal bit rate. The CAN protocol uses Non Return to Zero (NRZ) coding, which does not encode a clock within the data stream. Therefore, the receive clock must be recovered by the receiving nodes and synchronized to the transmitter's clock.

As oscillators and transmission times may vary from node to node, the receiver must have some type of Phase Lock Loop (PLL) synchronized to data transmission edges to synchronize and maintain the receiver clock. Since the data is NRZ-coded, it is necessary to include bit-stuffing to ensure that an edge occurs at least every six bit times to maintain the Digital Phase Lock Loop (DPLL) synchronization.

The bit timing of the MCP2515 is implemented using a DPLL that is configured to synchronize to the incoming data, as well as provide the nominal timing for the transmitted data. The DPLL breaks each bit time into multiple segments made up of minimal periods of time, called the Time Quanta (TQ).

Bus timing functions executed within the bit time frame (such as synchronization to the local oscillator, network transmission delay compensation and sample point positioning) are defined by the programmable bit timing logic of the DPLL.

5.1 The CAN Bit TIme

All devices on the CAN bus must use the same bit rate. However, all devices are not required to have the same master oscillator clock frequency. For the different clock frequencies of the individual devices, the bit rate has to be adjusted by appropriately setting the Baud Rate Prescaler and number of time quanta in each segment.

The CAN bit time is made up of non-overlapping segments. Each of these segments are made up of integer units called Time Quanta (TQ), explained later in this data sheet. The Nominal Bit Rate (NBR) is defined in the CAN specification as the number of bits per second transmitted by an ideal transmitter with no resynchronization. It can be described with the equation:

EQUATION 5-1:

$$NBR = f_{bit} = \frac{1}{t_{bit}}$$

Nominal Bit Time

The Nominal Bit Time (NBT) (t_{bit}) is made up of non-overlapping segments (Figure 5-1). Therefore, the NBT is the summation of the following segments:

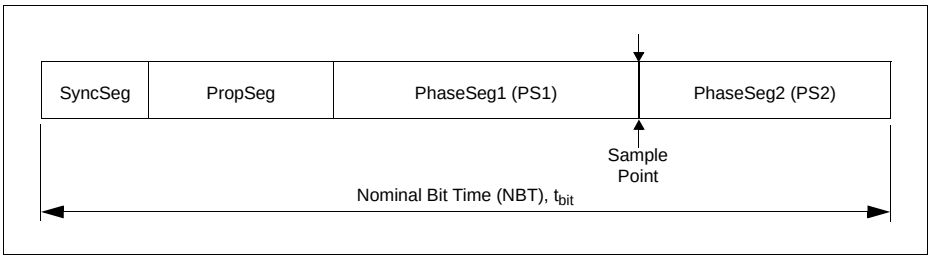
$$t_{bit} = t_{SyncSeg} + t_{PropSeg} + t_{PS1} + t_{PS2}$$

Associated with the NBT are the sample point, Synchronization Jump Width (SJW) and Information Processing Time (IPT), which are explained later.

SYNCHRONIZATION SEGMENT

The Synchronization Segment (SyncSeg) is the first segment in the NBT and is used to synchronize the nodes on the bus. Bit edges are expected to occur within the SyncSeg. This segment is fixed at 1 TQ.

FIGURE 5-1: CAN BIT TIME SEGMENTS



MCP2515

PROPAGATION SEGMENT

The Propagation Segment (PropSeg) exists to compensate for physical delays between nodes. The propagation delay is defined as twice the sum of the signal's propagation time on the bus line, including the delays associated with the bus driver. The PropSeg is programmable from 1-8 TQ.

PHASE SEGMENT 1 (PS1) AND PHASE SEGMENT 2 (PS2)

The two phase segments, PS1 and PS2, are used to compensate for edge phase errors on the bus. PS1 can be lengthened (or PS2 shortened) by resyncronization. PS1 is programmable from 1-8 TQ and PS2 is programmable from 2-8 TQ.

SAMPLE POINT

The sample point is the point in the bit time at which the logic level is read and interpreted. The sample point is located at the end of PS1. The exception to this rule is if the sample mode is configured to sample three times per bit. In this case, while the bit is still sampled at the end of PS1, two additional samples are taken at one-half TQ intervals prior to the end of PS1, with the value of the bit being determined by a majority decision.

INFORMATION PROCESSING TIME

The Information Processing Time (IPT) is the time required for the logic to determine the bit level of a sampled bit. The IPT begins at the sample point, is measured in TQ and is fixed at 2 TQ for the Microchip CAN module. Since PS2 also begins at the sample point and is the last segment in the bit time, it is required that the PS2 minimum is not less than the IPT.

Therefore:

$$PS2_{min} = IPT = 2TQ$$

SYNCHRONIZATION JUMP WIDTH

The Synchronization Jump Width (SJW) adjusts the bit clock as necessary by 1-4 TQ (as configured) to maintain synchronization with the transmitted message. Synchronization is covered in more detail later in this data sheet.

Time Quantum

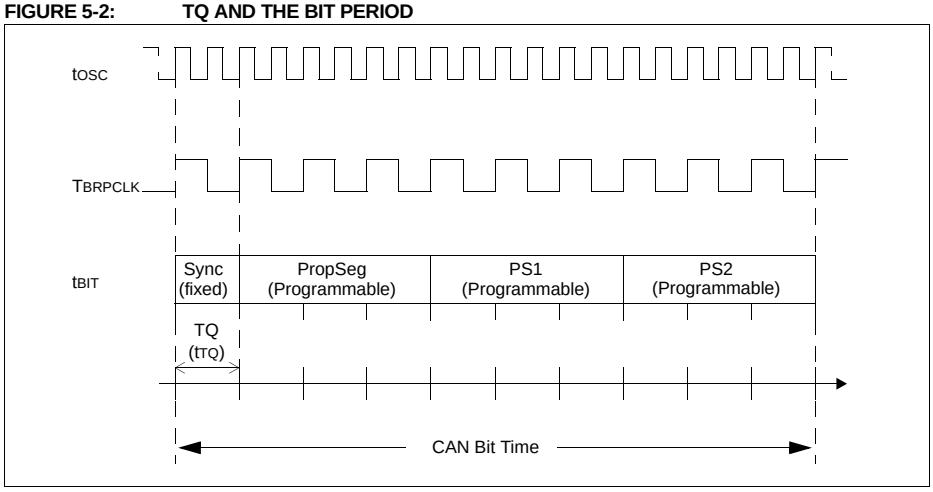
Each of the segments that make up a bit time are made up of integer units called Time Quanta (TQ). The length of each Time Quantum is based on the oscillator period (t_{OSC}). The base TQ equals twice the oscillator period. Figure 5-2 shows how the bit period is derived from T_{OSC} and TQ. The TQ length equals one TQ clock period (t_{BRPCLK}), which is programmable using a programmable prescaler, called the Baud Rate Prescaler (BRP). This is illustrated in the following equation:

EQUATION 5-2:

$$TQ = 2 \cdot BRP \cdot T_{OSC} = \frac{2 \cdot BRP}{F_{OSC}}$$

Where: BRP equals the configuration as shown in Register 5-1.

FIGURE 5-2: TQ AND THE BIT PERIOD



5.3 Programming Time Segments

Some requirements for programming of the time segments:

- PropSeg + PS1 >= PS2
- PropSeg + PS1 >= TDELAY
- PS2 > SJW

For example, assuming that a 125 kHz CAN baud rate with FOSC = 20 MHz is desired:

TOSC = 50 ns, choose BRP<5:0> = 04h, then TQ = 500 ns. To obtain 125 kHz, the bit time must be 16 TQ.

Typically, the sampling of the bit should take place at about 60-70% of the bit time, depending on the system parameters. Also, typically, the TDELAY is 1-2 TQ.

SyncSeg = 1 TQ and PropSeg = 2 TQ. So setting PS1 = 7 TQ would place the sample at 10 TQ after the transition. This would leave 6 TQ for PS2.

Since PS2 is 6, according to the rules, SJW could be a maximum of 4 TQ. However, a large SJW is typically only necessary when the clock generation of the different nodes is inaccurate or unstable, such as using ceramic resonators. So a SJW of 1 is usually enough.

5.4 Oscillator Tolerance

The bit timing requirements allow ceramic resonators to be used in applications with transmission rates of up to 125 kbit/sec as a rule of thumb. For the full bus speed range of the CAN protocol, a quartz oscillator is required. A maximum node-to-node oscillator variation of 1.7% is allowed.

5.5 Bit Timing Configuration Registers

The configuration registers (CNF1, CNF2, CNF3) control the bit timing for the CAN bus interface. These registers can only be modified when the MCP2515 is in Configuration mode (see **Section 10.0 “Modes of Operation”**).

5.5.1 CNF1

The BRP<5:0> bits control the Baud Rate Prescaler. These bits set the length of TQ relative to the OSC1 input frequency, with the minimum TQ length being 2 TOSC (when BRP<5:0> = ‘b000000’). The SJW<1:0> bits select the SJW in terms of number of TQs.

5.5.2 CNF2

The PRSEG<2:0> bits set the length (in TQ’s) of the propagation segment. The PHSEG1<2:0> bits set the length (in TQ’s) of PS1.

The SAM bit controls how many times the RXCAN pin is sampled. Setting this bit to a ‘1’ causes the bus to be sampled three times: twice at TQ/2 before the sample point and once at the normal sample point (which is at the end of PS1). The value of the bus is determined to be the majority sampled. If the SAM bit is set to a ‘0’, the RXCAN pin is sampled only once at the sample point.

The BTLMODE bit controls how the length of PS2 is determined. If this bit is set to a ‘1’, the length of PS2 is determined by the PHSEG2<2:0> bits of CNF3 (see **Section 5.5.3 “CNF3”**). If the BTLMODE bit is set to a ‘0’, the length of PS2 is greater than that of PS1 and the information processing time (which is fixed at 2 TQ for the MCP2515).

5.5.3 CNF3

The PHSEG2<2:0> bits set the length (in TQ’s) of PS2, if the CNF2.BTLMODE bit is set to a ‘1’. If the BTLMODE bit is set to a ‘0’, the PHSEG2<2:0> bits have no effect.

12.0 SPI INTERFACE

12.1 Overview

The MCP2515 is designed to interface directly with the Serial Peripheral Interface (SPI) port available on many microcontrollers and supports Mode 0,0 and Mode 1,1. Commands and data are sent to the device via the SI pin, with data being clocked in on the rising edge of SCK. Data is driven out by the MCP2515 (on the SO line) on the falling edge of SCK. The $\overline{CS}$ pin must be held low while any operation is performed. Table 12-1 shows the instruction bytes for all operations. Refer to Figure 12-10 and Figure 12-11 for detailed input and output timing diagrams for both Mode 0,0 and Mode 1,1 operation.

Note:	The MCP2515 expects the first byte after lowering $\overline{CS}$ to be the instruction/ command byte. This implies that $\overline{CS}$ must be raised and then lowered again to invoke another command.

12.2 RESET Instruction

The RESET instruction can be used to re-initialize the internal registers of the MCP2515 and set Configuration mode. This command provides the same functionality, via the SPI interface, as the $\overline{RESET}$ pin.

The RESET instruction is a single-byte instruction that requires selecting the device by pulling $\overline{CS}$ low, sending the instruction byte and then raising $\overline{CS}$. It is highly recommended that the Reset command be sent (or the $\overline{RESET}$ pin be lowered) as part of the power-on initialization sequence.

12.3 READ Instruction

The READ instruction is started by lowering the $\overline{CS}$ pin. The READ instruction is then sent to the MCP2515 followed by the 8-bit address (A7 through A0). Next, the data stored in the register at the selected address will be shifted out on the SO pin.

The internal Address Pointer is automatically incremented to the next address once each byte of data is shifted out. Therefore, it is possible to read the next consecutive register address by continuing to provide clock pulses. Any number of consecutive register locations can be read sequentially using this method. The read operation is terminated by raising the $\overline{CS}$ pin (Figure 12-2).

12.4 READ RX BUFFER Instruction

The READ RX BUFFER instruction (Figure 12-3) provides a means to quickly address a receive buffer for reading. This instruction reduces the SPI overhead by one byte, the address byte. The command byte actually has four possible values that determine the Address Pointer location. Once the command byte is sent, the controller clocks out the data at the address location the same as the READ instruction (i.e., sequential reads are possible). This instruction further reduces the SPI overhead by automatically clearing the associated receive flag (CANINTF.RXnIF) when $\overline{CS}$ is raised at the end of the command.

12.5 WRITE Instruction

The WRITE instruction is started by lowering the $\overline{CS}$ pin. The WRITE instruction is then sent to the MCP2515 followed by the address and at least one byte of data.

It is possible to write to sequential registers by continuing to clock in data bytes, as long as $\overline{CS}$ is held low. Data will actually be written to the register on the rising edge of the SCK line for the D0 bit. If the $\overline{CS}$ line is brought high before eight bits are loaded, the write will be aborted for that data byte and previous bytes in the command will have been written. Refer to the timing diagram in Figure 12-4 for a more detailed illustration of the byte write sequence.

12.6 LOAD TX BUFFER Instruction

The LOAD TX BUFFER instruction (Figure 12-5) eliminates the eight-bit address required by a normal Write command. The eight-bit instruction sets the Address Pointer to one of six addresses to quickly write to a transmit buffer that points to the “ID” or “data” address of any of the three transmit buffers.

12.7 REQUEST-TO-SEND (RTS) Instruction

The RTS command can be used to initiate message transmission for one or more of the transmit buffers.

The MCP2515 is selected by lowering the $\overline{CS}$ pin. The RTS command byte is then sent. Shown in Figure 12-6, the last 3 bits of this command indicate which transmit buffer(s) are enabled to send.

This command will set the TxBnCTRL.TXREQ bit for the respective buffer(s). Any or all of the last three bits can be set in a single command. If the RTS command is sent with nnn = 000, the command will be ignored.

12.8 READ STATUS Instruction

The READ STATUS instruction allows single instruction access to some of the often used status bits for message reception and transmission.

The MCP2515 is selected by lowering the $\overline{CS}$ pin and the Read Status command byte, shown in Figure 12-8, is sent to the MCP2515. Once the command byte is sent, the MCP2515 will return eight bits of data that contain the status.

If additional clocks are sent after the first eight bits are transmitted, the MCP2515 will continue to output the status bits as long as the $\overline{CS}$ pin is held low and clocks are provided on SCK.

MCP2515

Each status bit returned in this command may also be read by using the standard Read command with the appropriate register address.

12.9 RX STATUS Instruction

The RX STATUS instruction (Figure 12-9) is used to quickly determine which filter matched the message and message type (standard, extended, remote). After the command byte is sent, the controller will return 8 bits of data that contain the status data. If more clocks are sent after the eight bits are transmitted, the controller will continue to output the same status bits as long as the $\overline{CS}$ pin stays low and clocks are provided.

12.10 BIT MODIFY Instruction

The BIT MODIFY instruction provides a means for setting or clearing individual bits in specific status and control registers. This command is not available for all registers. See **Section 11.0 "Register Map"** to determine which registers allow the use of this command.

Note:	Executing the Bit Modify command on registers that are not bit-modifiable will force the mask to FFh. This will allow byte-writes to the registers, not bit modify.

The part is selected by lowering the $\overline{CS}$ pin and the Bit Modify command byte is then sent to the MCP2515. The command is followed by the address of the register, the mask byte and finally the data byte.

The mask byte determines which bits in the register will be allowed to change. A '1' in the mask byte will allow a bit in the register to change, while a '0' will not.

The data byte determines what value the modified bits in the register will be changed to. A '1' in the data byte will set the bit and a '0' will clear the bit, provided that the mask for that bit is set to a '1' (see Figure 12-7).

FIGURE 12-1: BIT MODIFY

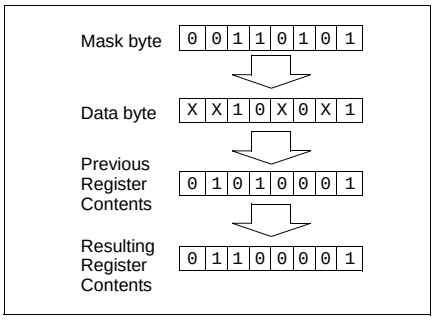


TABLE 12-1: SPI INSTRUCTION SET

Instruction Name	Instruction Format	Description
RESET	1100 0000	Resets internal registers to default state, set Configuration mode.
READ	0000 0011	Read data from register beginning at selected address.
READ RX BUFFER	1001 0nm0	When reading a receive buffer, reduces the overhead of a normal Read command by placing the Address Pointer at one of four locations, as indicated by 'n,m'. **Note:** The associated RX flag bit (CANINTF.RXnIF) will be cleared after bringing $\overline{CS}$ high.
WRITE	0000 0010	Write data to register beginning at selected address.
LOAD TX BUFFER	0100 0abc	When loading a transmit buffer, reduces the overhead of a normal Write command by placing the Address Pointer at one of six locations as indicated by 'a,b,c'.
RTS (Message Request-To-Send)	1000 0nnn	Instructs controller to begin message transmission sequence for any of the transmit buffers. 1000 0nnn Request-to-send for TXB2 — Request-to-send for TXB1 — Request-to-send for TXBO
READ STATUS	1010 0000	Quick polling command that reads several status bits for transmit and receive functions.
RX STATUS	1011 0000	Quick polling command that indicates filter match and message type (standard, extended and/or remote) of received message.
BIT MODIFY	0000 0101	Allows the user to set or clear individual bits in a particular register. **Note:** Not all registers can be bit-modified with this command. Executing this command on registers that are not bit-modifiable will force the mask to FFh. See the register map in **Section 11.0 "Register Map"** for a list of the registers that apply.

G.6 MCP2551

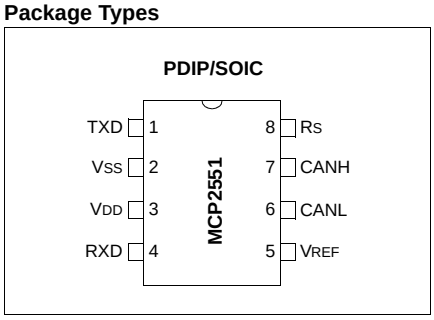
MCP2551

High-Speed CAN Transceiver

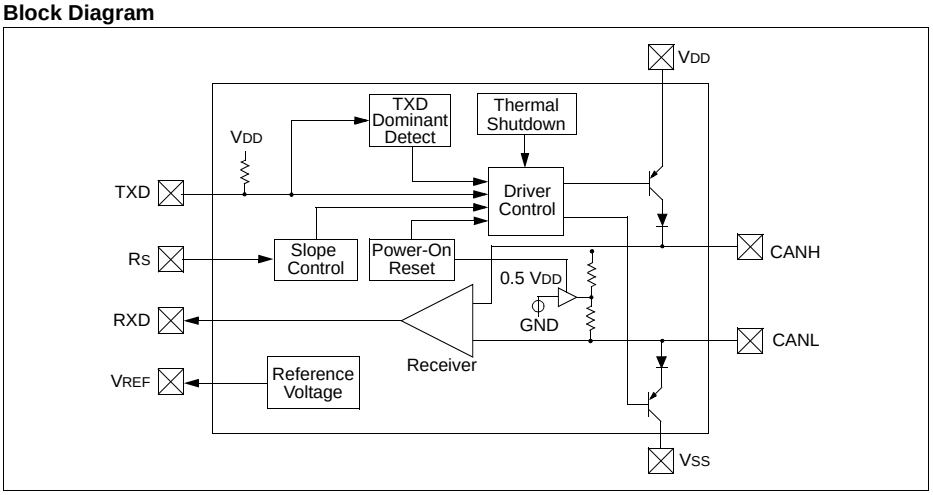
Features

- Supports 1 Mb/s operation
- Implements ISO-11898 standard physical layer requirements
- Suitable for 12V and 24V systems
- Externally-controlled slope for reduced RFI emissions
- Detection of ground fault (permanent Dominant) on TXD input
- Power-on Reset and voltage brown-out protection
- An unpowered node or brown-out event will not disturb the CAN bus
- Low current standby operation
- Protection against damage due to short-circuit conditions (positive or negative battery voltage)
- Protection against high-voltage transients
- Automatic thermal shutdown protection
- Up to 112 nodes can be connected
- High-noise immunity due to differential bus implementation
- Temperature ranges:
  - Industrial (I): -40°C to +85°C
  - Extended (E): -40°C to +125°C

Package Types



Block Diagram



G.7 ROE0505S

Features

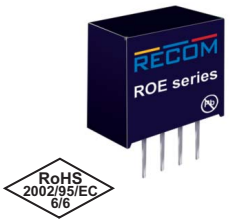
Unregulated Converters

- 1:1 Input Range
- Low Cost 1W Converter
- Efficiency to 75%
- -40°C to +85°C Operating Temperature Range

ECONOLINE
DC/DC-Converter

RECOM

1 Watt SIP4
Single Output



RoHS 2002/95/EC 6/6

ROE

Selection Guide

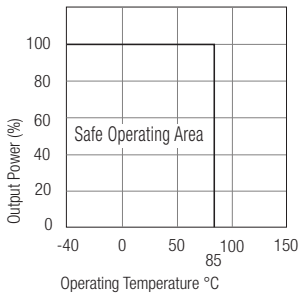
Part Number SMD	Input Voltage (VDC)	Output Voltage (VDC)	Output Current (mA)	Efficiency (typ.)
ROE-0505S	5	5	200	75%

Specifications (measured at T_A = 25°C, nominal input voltage, full load and after warm-up)

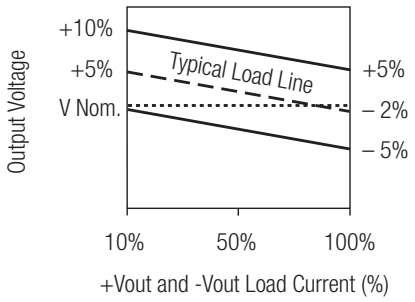
Input Voltage Range		±10% max.
Output Voltage Accuracy		-2% typ., ±5% max.
Line Voltage Regulation	(low line to high line at max. load)	1.2% typ.
	20% to 100% load (5V output)	10% max.
Output Ripple and Noise (20MHz BW limited)		52mVp-p typ. / 100mVp-p max.
Operating Frequency (Vin=nominal input)		50kHz min. / 82kHz typ. / 105kHz max.
Efficiency		75% typ. / 70% min.
Isolation Test Voltage (tested for 1 second)		1000 VDC min.
Isolation Capacitance		75pF max.
Isolation Resistance	(Viso=500V)	1GΩ min.
Short-Circuit Protection		1 sec.
Operating Temperature Range		-40°C to +85°C
Storage Temperature		-55°C to +125°C
Relative Humidity		95% RH
Package Weight		1.4g
MTBF (+25°C) *Detailed Information see Application Notes chapter "MTBF"*	using MIL-HDBK 217F	2400 x 10^3 hours
MTBF (+85°C)	using MIL-HDBK 217F	650 x 10^3 hours

Typical Characteristics

Derating-Graph
(Ambient Temperature)



Tolerance Envelope



Appendix H

Documentation and Manuals

H.1 Overview

There are limitations of producing commercial use documentation with an incomplete product. Given the software and hardware of the bridge are going to undergo significant change before these documents are complete, full documentation will not be produced. Below is an outline of what would be included in each of the required documents.

Technical Manual

Operators Manual

Software Manual

H.2 Technical Manual

This document will include the following information.

Mounting instructions - The bridge is to be placed inside a plastic container and sealed to obtain an IP rating. This enclosure will likely be intended for a wall mount so will include screw holes. The manual should include instructions.

Power supply - Include connections diagrams and suitable supply voltages. For the bridge this is currently 12V - 24V AC or DC.

Communications connections - Include the connection instructions for both communications interfaces. Will demonstrate the correct wiring for each bus as well as the use of the ground connections.

Any other information regarding installation or hardware diagnostics. This might include information about test points.

H.3 Operators Manual

This document will focus on how to use the bridge. It will be a simple document since the bridge is intended to be configured and left alone.

Bridge Setup - Section will include information about the default configuration of the bridge. Will refer you to the software manual if you wish to make configuration changes to the bridge.

CAN Setup - Will give instructions on how to configure the BMS CAN node for successful interaction with the bridge - how to access the Modbus slave from the CAN node.

Diagnostics - This section will give a troubleshooting guide to the bridge. Will include information about the fault (ERR) LED and the meaning of different frequency flashes.

Software changes - refer software manual

H.4 Software Manual

This document will outline how to make software configuration changes to the bridge. The types of changes the user can expect include:

CAN baud rate
CAN addressing mode (STD or XTD)
CAN flags, filters and masks
Modbus baud rate
Modbus parity (none, even, odd)
Modbus mode (ASCII or RTU)
Error reporting to BMS CAN node
Power outage response
GLCD display information

The menu operations via the GLCD and touch panel were not included in the scope of the project so implementation of these features are not possible.

Appendix I

Preliminary Report Documentation

I.1 Overview

This section is a selected extract from the preliminary report containing non-technical aspects of the project. This includes consideration for sustainability and the environment, a risk assessment, ethical considerations, the initial design concept and methodology, resource requirements and a proposed time-line.

These were originally submissions as part of the preliminary report submitted in June 2015.

4. Consequential Effects

4.1 Sustainability and Environment

The Institute of Engineers (1997) published a document with guidelines for engineers when dealing with issues of sustainability. The key considerations for this project are considered below. Wherever possible, these factors will be considered in decisions regarding this project.

4.1.1 E-waste and finite resources

E-waste and finite resource use are issues that confront any involved in the manufacturing of electronic devices. Silicon is the major resource for integrated circuits and fortunately, the Earth is abundant with this resource. However, the processes involved in the manufacture have negative environmental effects. These include the use of cleaning agents, solders (especially lead based), plastics in packaging, and disposal of waste products (ELE1502 Electronic Circuits: Study Book 2012). These issues present a challenge to this project. Ultimately, there is little that can be done on this project as these practices are well ingrained in the industry and there is less flexibility in a sponsored project. When the opportunity to take a sustainable option arises throughout the project, it is important that it is taken.

4.1.2Eradication of poverty

It is important to consider whether engineering projects actively contribute to the increase or decrease of poverty. For there to be true sustainability there needs to be equality for all genders, ages and races regarding living standards and employment participation (Institute of Engineers 1997). This project has little consequence on these issues directly but is part of larger systems that may have had significant impact historically.

Electronics in general contribute to this issue as a lot of manufacturing is done in countries with lower labour costs (ELE1502 Electronic Circuits: Study Book 2012). This raises the issue of exploitation of the works in these countries. The counter argument is that it may not be feasible to produce electronic products at the same prices if labour costs are higher.

Building management systems and their autonomous nature may have contributed to the loss of employment for some demographics; however this is a complex issue involving many factors and the discussion is well beyond the scope of this project. The network bridge itself performs a function that a human could not.

It is important to recognise the effect such systems have on the population wherever they are used but is not something considered further here.

4.2 Risk Assessment

The network bridge is a small, low power electronic device. It presents little innate safety risk due to these traits but is not immune to hazards. Table 2 considers risks appropriate to this type of device during design and production and Table 3 considers risk during the installing, operation and maintenance of the network bridge (ENG4111 Research project part 1: Project reference book 2011).

Table 2: Project risk assessment – Safety during development

Hazard	Likelihood	Exposure	Consequence	Control
Electrical shock – low voltage	Slight	Frequently	Critical damage to equipment	Ensure powering of circuits only occurs after rigorous checking for short circuits
Soldering iron	Slight	Frequently	Minor burns	Clear working space – when soldering taking place it should be the only active task

Table 3: Project risk assessment - Operation

Hazard	Likelihood	Exposure	Consequence	Control
Network bridge failure	Significant	Continuous	BMS receives incorrect information – could be severe dependant on environment	The network bridge must have in-built features that see it fail safe – when failure occurs it should be immediately reported to the BMS
Electric shock	Extremely slight	Occasionally	Electric shock – possible injury or death	Installations to be performed by trained operators – due care taken when connecting power to device as required for any electronic installation

The other type of risk to be managed is project completion risk. That is, what obstacles might be present during the project and what effect might they have upon a successful completion. These risks are considered in Table 4.

Table 4: Project risk assessment - Project Completion

Hazard	Likelihood	Exposure	Consequence	Control
Components take longer to receive than planned	Significant	Occasional	Latter project stages are delayed or not able to be completed	Complete these activities at the earliest possible stage to give a buffer to unplanned delays
Network bridge or individual aspects do not work	Significant	Continuous	Project may not be completed during time allotted	Planning of all aspects so that the flow on effects of current work is known. Identify extra time in which to allocate to the project if needed (ie weekends)

4.3 Ethical

The main ethical issue in consideration for this project is that of intellectual property. Since it is a sponsored project, the work and any outcomes are property of GDA. At various stages of the project there will be requirements to interact with third parties. As such, consideration must be paid to what information can be shared during these interactions. Further to this, it is important to abide by Engineers Australia's Code of Ethics throughout the project.

Initial design concept

At this stage design is in its infancy but Figure 5 demonstrates the intended logical functionality of the network bridge. There is clear logical distinction between the CAN and Modbus nodes even though they will both be housed in the same physical device. This figure is not meant to be a complete design, but is a useful diagram for visualising how the project will be approached. This approach is discussed further in the methodology below.

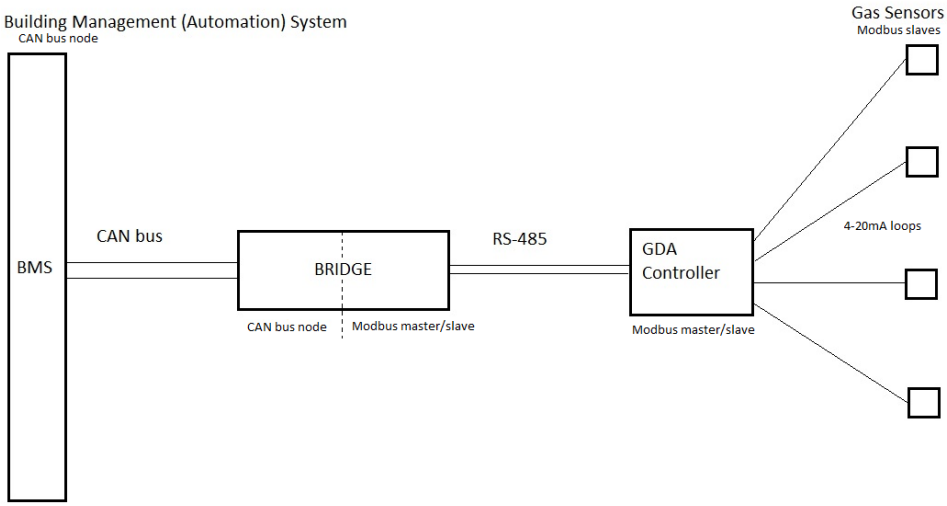


Figure 5: Initial design concept

5. Methodology

5.1 The method

The network bridge has two major components; hardware and software. While they are interlinked, it is convenient to develop them separately and then combine them after proving concepts individually. The other key methodology for the project is the testing procedures for the hardware, software and the completed network bridge. Conceptually, there are three separate components to the bridge; the CAN node, the Modbus node and the intelligence that connects them. Figure 5 demonstrates this logical separation.

Hardware development will involve designing circuitry to provide the physical interface used to implement the protocols. It will be necessary to use a microprocessor to implement the protocols and to provide the logical interface between them. The microprocessor selection is the first step in the hardware design. It will influence what other hardware can and will be used. There will likely be sundry electronics required to operate the microprocessor in the manner desired. After the selection of a microprocessor is the selection of hardware components for the CAN bus and Modbus interfaces. It is likely the Modbus interface will be directly implementable from the microprocessor with few extra components due to many devices supporting serial communications. The CAN bus interface will require further consideration as it is not supported completely in most microprocessors. Even though many parts may have an integrated CAN controller, which is not the only circuitry required. As a result, full circuitry to enable a CAN interface is required and will be specified.

The software development procedure is based around the conceptual view of the network bridge. Each of the two protocol interfaces will be developed first. It does not matter in which order this is done. Where possible, existing code libraries within the compiler for development of CAN, Modbus and serial communications will be utilised to prove concepts. Once each interface has been implemented using existing code libraries, they will be rewritten using techniques consistent with GDA practices. Once the CAN bus interface and Modbus interface are developed, the next state is to connect the two logically. This will involve the development of a real time operating system with buffers to handle messages from each interface. It will be necessary to calculate and then verify data timing and data throughput requirements to determine how best information transfer is handled. This may have an effect on the hardware specifications and this will be adjusted if necessary.

GDA has preferences to certain manufacturers and suppliers due to existing development platforms and business relationships. As a result, the selection of hardware and development tools are more limited than might otherwise be the case. A PIC branded microprocessor will be chosen for the task. The compiler and development software to be used is MicroC Pro for PIC version 6.60. The EasyPIC Pro v7 development board will be used for software development prior to the completion of hardware design and testing.

Testing for the correct operation is required for the network bridge. As discussed, testing of each protocol interface will be conducted individually at first. This will involve setting up another node, or simulating a node, using that protocol and connecting it to the network bridge. Known data and instructions can then be passed to and from the network bridge to verify that it can correctly transmit using each protocol. To test the full functionality of the network bridge a combination of

the above testing procedure will be required. Simulated or real nodes for both CAN and Modbus will be required. Placing the network bridge in between will allow testing of data transmission and requests between the two nodes.

5.2 Resources

This project is sponsored by GDA and as such all resourcing is done through this channel. Below is a list of requirements for the project:

- Microprocessors for both development and final products
- Printed circuit board (PCB) and electronic components
- PIC development board
- C language compiler for PIC
- Testing equipment (devices to simulate Modbus or CAN bus nodes, oscilloscopes etc.)
- Electronics manufacturing and prototyping equipment (soldering etc.)

All of the above will be acquired through GDA. We they already have possession of such equipment and resources it will be used. Any extra requirements will need to be justified to and requested through GDA's purchasing channels. Vendors will be likely chosen from existing suppliers or by those approved by GDA. All costing decisions are made by GDA and justification may need to be made if desirable products are more expensive than alternatives. It is acceptable to locate other sources for resources if required but they must be approved by GDA before purchasing will occur.

The most critical resources to this project are the software development tools and the PCBs. GDA has possession of multiple development tools suitable for the task which should mitigate risk associated with this resource. The greater risk is with PCB production. As the manufacturing of any designed PCBs will likely take place internationally, there may significant timing issues with their acquisition. To help reduce the risks of this, hardware design will be a priority early in the project development to allow as much time as possible. If it is becomes clear that procurement of PCBs will not happen within acceptable time frames it may be possible to use breadboards or prototyping boards to complete the hardware construction.

5.3 Timelines – Gantt chart

The following Gantt chart demonstrates the expected timing of project stages and their effect on milestones for the project.

Task	Label	Predecessors	Expected Duration (weeks)	Date (weeks apart - starting June 22, 2015) 22/06	29/06	6/07	13/07	20/07	27/07	3/08	10/08	17/08	24/08	31/08	7/09	14/09	21/09	28/09	5/10	12/10	19/10	26/10
Hardware design	A		1																			
Hardware prototyping	B	A	2																			
Hardware prototyping - testing	C	B	1																			
Hardware PCB layout	D	C	1																			
Hardware PCB delivery	E	D	3																			
Hardware PCB testing	F	E	1																			
Software design - Modbus	G		2																			
Software design - CAN	H		2																			
Software design - Bridge	I	G, H	4																			
Software testing - Modbus	J	G	1																			
Software testing - CAN	K	H	1																			
Software testing - Bridge	L	J, K	4																			
Network bridge hardware/software combination	M	F, L	3																			
Network bridge hardware/software combination - testing	N	M	2																			
Documentation	O	N	2																			
Dissertation - incremental	P		15																			
Dissertation - completion	Q	N, O, P	2																			

This chart demonstrates that many aspects of the project can be worked upon independently and must be done concurrently to allow for completion of the project within the defined timelines. It is important that stages that rely on earlier tasks be completed as identified or within reasonable variations.