Design and Implementation of a USB-to-CAN Bridge for the GuRoo Project

by

Bartlomiej (Bartek) Bebel

Department of Information Technology and Electrical Engineering,

University of Queensland,

Australia

Submitted for degree of

Bachelor of Engineering (Honours) in the division of

Computer Systems Engineering

October 2001

Bartek Bebel
10/134 Station Road,
Indooroopilly, QLD, 4068

19 October 2001

Head, School of Information Technology
and Electrical Engineering,
The University of Queensland,
St. Lucia, QLD, 4072.

Dear Professor Kaplan,

In accordance with the requirements of the degree of Bachelor of Engineering in the division of Computer Systems Engineering, I present the following thesis entitled “Design and Implementation of a USB-to-CAN Bridge for the GuRoo Project”. This work has been performed under supervision of Dr. Gordon Wyeth.

I declare that the work submitted in this thesis has not been previously submitted for a degree at the University of Queensland or any other institution. To the best of my knowledge and belief, this thesis contains no material previously published or written by any other person, except where reference is made in the text.

Yours sincerely,

Bartek Bebel.

Abstract

This thesis details the design and implementation of a USB-to-CAN Bridge for use in the GuRoo, the walking robot project. GuRoo requires information to be sent from the main controlling unit, an iPAQ, to joint controller boards located in various parts of the robot.

The design presented in this thesis modifies one of the controller boards to act as a link between the iPAQ and the remaining controllers. iPAQ's Universal Serial Bus (USB) interface has been utilised to communicate via the bridging board to a Controller Area Network (CAN) linking the controller boards. Half duplex communication needs to be implemented to allow of passing commands from the iPAQ as well as transmitting status information from the controller boards.

The hardware design has been implemented and tests conducted to ensure it is working. At this stage the software is written, but not operational. It is expected that by the end of the semester the USB-to-CAN Bridge should be able to pass messages from and to the iPAQ.

The document concludes with suggestions for future work and improvements that can be made to the current product.

Acknowledgments

I would like to thank the following people for their involvement and/or support with this project:

Gordon Wyeth, my supervisor, for providing me with an opportunity to work on such an exciting project and his assistance throughout the year.

My girlfriend, Kylie Wust, and friends for supporting me in my journey. Thanks for helping me take my thoughts off the project and making me have some fun as well.

My colleagues in the robotics lab, for their company and assistance throughout the year. For those late Friday nights at the RE were we relaxed and had a few good laughs. Thanks guys.

Guys from the electronics workshop, for always being helpful and allowing me to use their equipment.

Table of Contents

Abstract ____ *iii*

Acknowledgments ____ *iv*

List of Figures ____ *vii*

List of Tables ____ *viii*

Chapter 1 – Introduction ____ *1*

1.1 Thesis Overview ____ **1**

1.2 The GuRoo Project ____ **2**

1.3 Scope of Work ____ **3**

1.4 Research Justification ____ **3**

1.5 Outline of Chapter Headings and Contents ____ **4**

Chapter 2 – Literature Review and Background Material ____ *6*

2.1 USB Overview ____ **6**

2.1.1 Introduction ____ 6

2.1.2 USB Topology and Transfer Types ____ 7

2.1.3 Electrical Specifications ____ 9

2.2 CAN Overview ____ **11**

2.3 What Has Been Done So Far ____ **13**

Chapter 3 – Specifications ____ *16*

Chapter 4 – Hardware Design ____ *18*

4.1 Main Controller Unit ____ **18**

4.2 Universal Serial Bus Interface ____ **22**

4.3 Controller Area Network Interface ____ **26**

Chapter 5 – Software Implementation ____ *28*

5.1 Introduction ____ **28**

5.2 Hardware Initialisation and Configuration ____ **29**

5.3 Main Loop ____ **31**

5.4 Interrupt Handlers **32**

5.5 CAN-to-USB and USB-to-CAN Message Conversion Code **34**

Chapter 6 – Product Evaluation ***35***

6.1 General Comments **35**

6.2 Meeting the Specifications **35**

6.2.1 Hardware Evaluation 35

6.2.2 Software Evaluation 36

6.4 USB Specification 1.1 Compliance **36**

6.5 Performance as an Engineer **37**

6.5.1 Strengths 37

6.5.2 Areas of Improvement 38

6.5.3 Areas of Skill Development 38

Chapter 7 – Conclusions and Pointers for Further Work ***40***

7.1 Conclusions **40**

7.2 Future Work **40**

References ***42***

Appendix A – Hardware Schematic and PCB Layout ***A1***

Appendix B – TMS320F243 Firmware ***B1***

Appendix C – Datasheets ***C1***

List of Figures

Figure 1.1 – GuRoo

Figure 2.1 – USB topology
Figure 2.2 – NRZI Encoding Scheme
Figure 2.3 – CAN Frame Specification
Figure 2.4 – iPC-I 165/cPCI
Figure 2.5 – USB-to-CAN

Figure 4.1 – TMS320F243 Block Diagram
Figure 4.2 – Controller and its supporting circuitry schematic
Figure 4.3 – JTAG and serial programmer connectors
Figure 4.4 – Reset controller schematic
Figure 4.5 – iPAQ to CAN network connection solutions
Figure 4.6 – USBN9603/4 packages
Figure 4.7 – USB interface controller and supporting circuitry
Figure 4.8 – CAN driver connections

Figure 5.1 – Software Progress Diagram
Figure 5.2 – Peripheral Interrupt Expansion Block Diagram
Figure 5.3 – Main USB servicing routine flowchart

List of Tables

Table 1.1 – GuRoo Project team

Table 2.1 – USB and RS-232 comparison

Chapter 1 – Introduction

1.1 Thesis Overview

The aim of this thesis is to build a Universal Serial Bus (USB) to Controller Area Network (CAN) Bridge to be used in a Humanoid Robot Design Project known as GuRoo. The bridge plays a vital role within the design enabling transfer of control from the main processing unit to individual joint controllers located throughout the robot.

The central control unit consists of Compaq's iPAQ pocket PC equipped with a USB port and running Windows CE. It needs to communicate with a number of controller boards networked together via CAN. This document describes a device that links the two communication standards allowing quick and efficient exchange of data.

Hardware selection, design process and construction of the device will be explained. Software development process is also described and the final code included in the accompanying appendices.

Hardware design of the USB-to-CAN Bridge is based around Texas Instruments' Digital Signal Processor (DSP) chip TMS320F243 used in joint controllers of the robot. The circuit board is located in the chest cavity of the robot with joint controller boards being placed close to the joints they control. This approach results in simplified wiring requirements and localised low-level control ensuing a very fast response in case of an emergency.

The processor used in USB-to-CAN Bridge carries a secondary task of power delivery control and over-current protection. This aspect of the design is not included in the document as this topic is covered by another thesis. That thesis is written by Nathaniel Brewer and can be obtained from the Department of Information Technology and Electrical Engineering, University of Queensland.

1.2 The GuRoo Project

The GuRoo project involves a team of 12 thesis students, each working on a particular aspect of robot's design. It is supervised by Gordon Wyeth, a member of the academic staff of the School of Information Technology and Electrical Engineering at the University of Queensland. Gordon has extensive knowledge about robotics and control,

Project sub-section	Students Involved
Mechanical Design	Damien Kee Mark Wagstaff Anthony Hunter
Main processor and internal communications	Shane Hosking Bartek Bebel
Joint Controllers	Jared Stirzaker Timothy Cartwright
Vision Hardware and Software	Andrew Blower (hardware) David Prasser (software)
Walking software and control loops	Andrew Smith Emanuel Zelniker
Power system	Nathaniel Brewer

Table 1.1 – GuRoo Project team

as well as his involvement in robot soccer for a long time. The GuRoo project is divided into six subsections as shown in Table 1.1.

In order to allow software development and hardware design to occur in parallel, a software simulation was developed. The simulation was used to develop walking software that will allow GuRoo (Figure 1.1) to stand upright and perform a sequence of joint rotations resulting in it moving forward.

Figure 1.1 - GuRoo

A lot of effort was made to make the work of the project as parallel as possible. The main objective was to develop each section independently (as much as possible) and then integrate all sections together. This approach should result in a relatively short development time and provide all of the team members with an aspect to work on.

1.3 Scope of Work

Research was conducted to establish if a similar product was available on the market and to establish whether other solutions have been attempted. Furthermore, detailed investigations of both USB and CAN specifications were conducted to isolate any potential problems and incompatibilities. The results are presented in Chapter 2.

The work performed in this thesis includes the design, construction, software development and testing of the device. Hardware schematics were drawn up, PCB laid out and fabricated. Component selection was conducted with the aim of minimising weight and space requirements of the design. Board assembly and extensive testing followed.

Software development included writing a C code for the main controller allowing it to communicate with the iPAQ. Plans for USB→CAN and CAN→USB message conversion code were also included. The objectives of firmware development were to make the message passing fast and code easy to follow.

1.4 Research Justification

The main drive for the research into this field was the need arising from GuRoo project. This design was specifically developed to meet this need; however, it does not have to be restricted only to this application. With some additional work, it could be easily changed into a commercial product. Majority of the work would require development of PC software to communicate with the device and manufacturing a separate PCB.

Currently the most popular option of connecting a PC to a CAN network is to purchase a CAN interface card that plugs into one of the extension ports of a desktop PC. This option does not allow for a lot of flexibility, as desktop PCs are bulky, heavy and difficult to move. Moving the card itself from one computer to another is not recommended and can be destructive to the card itself.

Being a USB device the USB-to-CAN Bridge developed for the GuRoo project enjoys all the benefits that come with USB devices. It can be easily transported due to its low weight and small dimensions. Ease of connection and an option of dynamic attachment and detachment allow it to be easily moved from one computer to another or even carried with a laptop to use in the field. Unlike RS-232 or parallel connections, USB is fast enough to control not only one, but possibly two or three CAN channels.

1.5 Outline of Chapter Headings and Contents

Chapter 2 Literature Review and Background Material
Contains information about previous work in this field and alternative solutions to the problem. It also includes background theory involved, as well as Universal Serial Bus (USB) and Controller Area Network (CAN) overviews.

Chapter 3 Specifications
Describes the design requirements placed on the device. Each requirement is in turn investigated and reasons behind it explained.

Chapter 4 Hardware Design
Contains information about hardware design and component selection. Various system configurations are examined and the most suitable configuration implemented. Each schematic subsection is individually explained and reasoning behind the chip selection provided.

Chapter 5 Software Implementation

Includes a review of the software written for USB-to-CAN Bridge. General description of the software and additional detailed investigations of important routines are presented. Hardware initialisation and interrupt handling routines have deemed to be essential to overall code understanding and therefore, have been well documented.

Chapter 6 Product Evaluation

Discusses the performance and specification compliance of the final product. Hardware and software are investigated separately and reflections about development of engineering skill included.

Chapter 7 Conclusions and Pointers for Further Work

Conclusions, followed by ideas for future work are presented.

Chapter 2 – Literature Review and Background Material

2.1 USB Overview

2.1.1 Introduction

USB is a successor of the well-known RS-232 serial communication protocol that has been around for many years and has now reached its limitations. Considerable improvements over RS-232 have been introduced and new features added. Some of these features include [1]:

- Dynamic attachment and detachment of peripherals,
- Ease of use (no user settings like interrupts or port addresses),
- Automatic configuration
- Power supplied via the USB cable (limited)
- Hot pluggable (plug and unplug with computer being powered up)

These changes were mainly motivated by the drive towards ease of use and small size, in line with the driving force of computer industry, resulting in every personal computer being equipped with a USB port. The trade-off on this, is higher complexity of software required to communicate via a USB bus. A quick comparison between the two interfaces is included in Table 2.1 (next page).

Many of the leading industry companies, including Compaq, Hewlett-Packard, Intel and Microsoft, developed universal Serial Bus in 1995. The major goals were to define an external expansion bus that makes adding peripherals simple and low cost. It was not supposed to represent an electronic analog of a mechanical system (like RS-232) [2]. It is fully supported by Microsoft's Windows 98 and later versions and is making its way into UNIX environments as well. USB system software hides hardware implementation details making the application software more portable and flexible.

	USB	**RS-232**
Transfer format	Asynchronous serial	Asynchronous serial
Maximum transfer rate	1.5Mb/s, 12Mb/s (480Mb/s in version 2.0)	20kb/s (115kb/s with some drivers)
Maximum connection length	5m (or up to 30m with 5 hubs)	15 – 31m
Maximum number of devices	127	2

Table 2.1 – USB and RS-232 comparison

2.1.2 USB Topology and Transfer Types

Universal serial bus incorporates a tiered star bus topology as seen in Figure 2.1. The host controller, also called Root Hub, initiates all data transfers. There can be only one host controller per USB network. Hubs are used to increase the number of connections, with each hub being the centre of a star. In Figure 2.1, the lines represent point-to-point wire connections between USB devices that are shown as nodes.

In order to cater for a wider range of peripherals, USB devices can operate at two speeds. Full-speed devices are able to communicate at 12Mb/s and low-speed devices at 1.5Mb/s. This ensures that low speed devices, like keyboards and mice do not waste USB bandwidth. The latest USB revision (rev. 2.0) provides for another transfer rate of 480Mb/s, however this thesis has been written with revision 1.1 in mind. Implementation of USB revision 1.1 was chosen for this design, due to its support by both the iPAQ and desktop PCs. USB revision 1.1 interface controllers were readily available and no

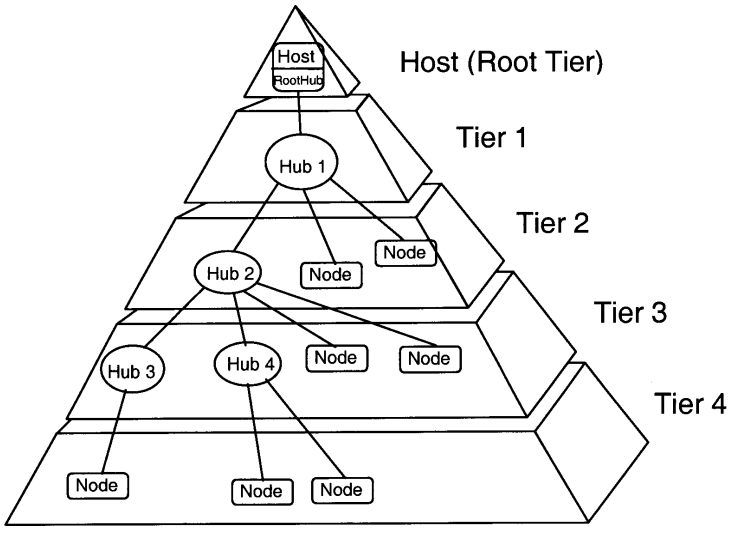

Figure 2.1 – USB topology [1]

significant benefits would have been obtained from USB revision 2.0 implementation. Provision for transfer speeds of 480 Mb/s would result in the product being over-engineered as 12 Mb/s provides enough bandwidth for this application.

The USB specification defines four data transmission formats (explained below) used on the bus [1], [3]. Generally, information is transferred as packets with each transfer type having its own packet structure and flow control.

Control Transfers – are intended to support configuration/command/status type communication flows between host software and a peripheral. Each USB device is required to support control transfers because they are used for device enumeration (see next paragraph). For full speed devices, the maximum packet size may be 8, 16, 32 or 64 bytes, whereas for low speed devices it is limited to 8 bytes. Handshaking is used to control dataflow and Cyclic Redundancy Check (CRC) for error detection.

Bulk Transfers – are used for transferring large amounts of data where time is not crucial. Only full speed devices can support bulk transfers with the maximum packet size being 8, 16, 32 or 64 bytes. USB does not allocate bandwidth for bulk transfers, instead bulk transfers will use up any unused bus bandwidth. Consequently, on a busy bus, bulk transfers may take some time to complete. Error detection is used to ensure a reliable data exchange.

Interrupt Transfers – are intended for use with small and limited-latency transfers. The maximum packet size for a full speed device is 64 bytes, or less, and low speed devices are limited to 8 bytes, or less. Handshaking is used to control dataflow and CRC to detect errors.

Isochronous Transfers – should be used when real-time delivery is critical. It provides fast data transfer, however the trade-off is that there is no provision for retransmission of corrupted data. Once an isochronous transfer mode is set up, USB permanently assigns a fraction of bus bandwidth to this connection for the duration of its existence. Only full speed devices can support isochronous transfers. Maximum data payload per frame is restricted to 1023 bytes per transaction per frame.

When a USB peripheral is plugged in, the host communicates with it to establish its resource requirements and identity the device driver it should use for communication.

This process is called device enumeration. During enumeration a unique address is assigned to each peripheral, which is then included in all messages sent to that device. The address consists of seven bits, therefore allowing up to 127 USB gadgets being connected to each host. It is worth mentioning that address 0000000_b is a special case and is assigned to all newly connected devices before they are enumerated and allocated a unique address. This means that since device addresses are allocated by the host, as opposed to being hard wired inside a device, there will never be a problem of two peripherals having the same address and a conflict occurring.

Hubs are used to physically allow simultaneous connection of 127 USB peripherals to one host. However, some restrictions apply. The maximum cable length is limited to five meters and hubs may be cascaded up to five levels deep, therefore extending combined cable length to 30 meters. The maximum cable length is limited by the delays within those cables, which have to be kept less than single frame duration. During normal operation, a hub works as a repeater passing USB signals either downstream (towards a device) or upstream (towards the host). Apart from passing signals, a hub is also responsible for power management and detection of new peripherals. According to USB specification revision 1.1 a hub must support both high and low speed devices.

2.1.3 Electrical Specifications

Universal Serial Bus provides a half-duplex serial asynchronous data transfer. Being a polled bus, all data transfers are initiated by the host controller. Information is encoded using differential signalling with Non Return to Zero Invert (NRZI) scheme. NRZI's advantage is the fact that it eliminates the need for an additional clock signal line. It encodes data by representing ones and zeroes using opposite and alternating high and low voltages. There is no return to zero or reference voltage between encoded bits. Meaning, logic high is represented by no change in signal levels, and logic low results in the signal levels being inverted as shown in Figure 2.2. Clock recovery circuitry uses signal transitions to reconstruct the clock. The problem of a long stream of consecutive "1s", resulting in no signal transitions, is solved using bit stuffing. Meaning that the data stream is examined and a "0" is inserted after every sixth consecutive one and then

encoded as NRZI. A receiver ensures that all zeroes that follow six ‘1’s are removed from the data stream during NRZI decoding.

A USB cable allows the transfer of signal as well as power. Two wires (D+ and D-) are used to transfer data in one direction at any one time (half-duplex) while the other two wires transfer power (5V). USB specifications state that a USB bus powered peripheral cannot consume more than 500mA of current at 5V. If more than 500mA is required, then the device needs to be self-powered. Some EMI shielding within cables is also provided to ensure good transmission.

When a USB device is connected, the operating system is responsible for its enumeration and device driver selection. USB descriptors are used to provide the operating system with the required data. Once connected, the host issues a number of descriptor requests and the information extracted from these descriptors is used to select a device driver.

Windows 98 (and later, including CE) is shipped with some generic USB drivers that can be used to communicate with USB peripherals. Another option is to use custom or vendor drivers. When choosing the controller chip it is usually very beneficial if a vendor driver comes with it. Finally, the most difficult and time-consuming solution is to write a custom device driver. This option is time-consuming, involves good knowledge of Windows’ Win32 Driver Model and advanced programming skills.

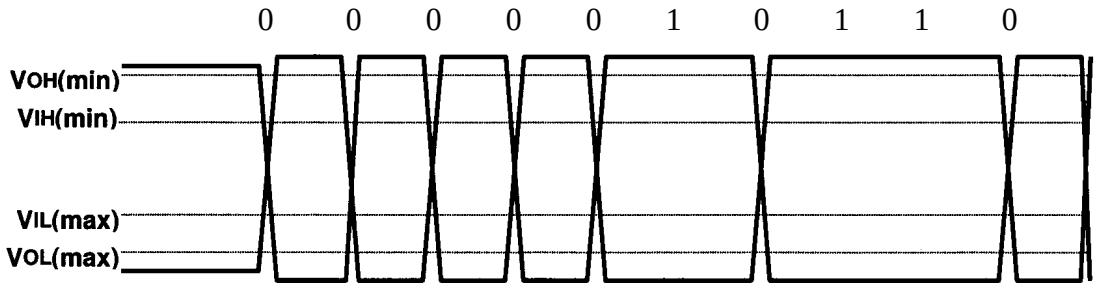

Figure 2.2 – NRZI Encoding Scheme [1]

2.2 CAN Overview

Controller Area Network (CAN) was developed by Robert Bosh GmbH. It is a serial communication protocol designed for real time distributed control using a bus topology. Sophisticated error checking and correction make it a very reliable and secure protocol. Data is passed in fixed format packets that may differ in length, allowing the maximum transfer rate of 1 Mbit/s.

The current CAN specification used is version 2.0, consisting of two parts:

- Standard CAN (Version 2.0A). Uses 11 bit identifiers.
- Extended CAN (Version 2.0B). Uses 29 bit identifiers.

The two types define different frame formats, with the main difference being the identifier length [4]. In addition, there are two different ISO standards for CAN, with ISO 11898 handling speeds of up to 1Mbit/s and ISO 11519 has the upper limit of 125 kbit/s [4].

Consequently, there are three types of CAN controller chip available on the market.

- type 2.0A,
- type 2.0B passive,
- type 2.0B active.

Controller type 2.0A can transmit only standard CAN messages and can be used only on standard CAN network. Type 2.0B passive controller can be used with both standard and extended CAN buses. However, it will not send and will ignore all extended mode messages. Its main advantage over type 2.0A is the fact that type 2.0B passive passes extended messages along, rather than causing an error as it happens in case of type 2.0A controller. The last type is 2.0B and it can both transmit and receive extended messages.

Unlike many other networks, CAN does not use station addressing, meaning that nodes do not have addresses and the messages are not passed from one network address to another. This scheme introduces a number of advantages, like system flexibility. Nodes can be easily added to the network without requiring any changes in software or checks for address conflicts. Message identifiers are used to describe a content of a message.

Each message is guaranteed to be accepted by either all or none of the nodes. When a node accepts a message it uses the message identifier to decide wether it is relevant to it or not. Consequently, any number of nodes can receive and act upon the same message.

The identifiers are also used to implement message prioritisation, with the lowest identifier numerical value being the highest priority. This is used when resolving bus collisions, ensuring the highest priority messages are sent first. Lower priority messages are automatically retransmitted during the next bus cycle.

CAN Version 2.0 message frame format is given in Figure 2.3. It consists of seven different bit fields [4]. A *Start of Frame field* (SOF) indicates the beginning of a message frame. *Arbitration Field* (12 bits) includes a message identifier and a Remote Transmission Request (RTR) used to indicate frame type. The *Control field* contains 6 control bits, two unused and the other four indicating data field length. *Data field* can consist of between zero and eight bytes and is followed by its CRC code in the *CRC field. Acknowledgement* and *End Of File* (EOF) fields followed by a three-bit intermission field (INT) complete each message. The main difference between CAN 2.0A and 2.0B frames is the size of their identifiers. In CAN 2.0A the identifier consists of 11 bits where in version 2.0B it has been extended to 29 bits.

Due to its use in control applications, CAN needs to achieve utmost safety of data transfer and provide measures of error detection. To achieve just that, a combination of cyclic redundancy checks with bit stuffing, bus monitoring and message frame checking was implemented. Consequently, the following was achieved:

- all global errors are detected,

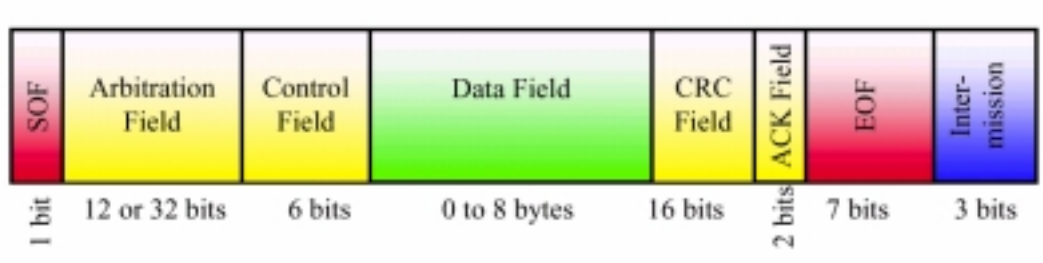

Figure 2.3 – CAN Frame Specification

- all local errors at transmitters are detected,
- up to 5 randomly distributed errors in a message are detected,
- burst errors of length less than 15 in a message are detected,
- errors of any odd number in a message are detected.

All of the above, when combined result in the undetected error probability of less than:

$$\text{Message Error Rate} * 4.7 * 10^{-11} \qquad [5].$$

Furthermore, the CAN specification ensures that when an error is detected by a node, no other node will act on the incorrect message. This is achieved by the node that has detected an error sending an error flag to alert all the other nodes on the bus. Additional measures are implemented to prevent a single node, which is having problems receiving messages, from continuously issuing error flags. Error handling is entirely done by the CAN interface chip and no microcontroller intervention is required.

2.3 What Has Been Done So Far

The most common way to connect to a CAN bus is an EISA or PCI card located inside a desktop PC. This solution is offered by a number of companies including a Germany based company specialising in CAN interface cards IXXAT Automation GmbH. Their iPC-I 165/cPCI card (Figure 2.4) is a relatively easy to use Plug & Play PCI interface card with a 16 bit on-board microcontroller running at 20 MHz. It supports up to 2 Mbytes of SRAM with up to 1 Mbytes of FLASH and 8 kbyte DPRAM (Dual Port RAM) [6]. Two separate CAN channels can be used for transmission with the microcontroller allowing filtering, pre-processing and storage of CAN messages (with timestamp) as well as real-time transmission.

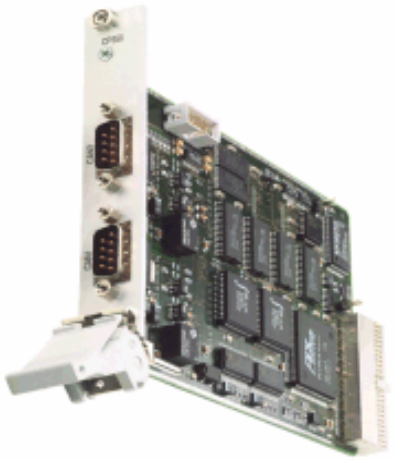

Figure 2.4 - iPC-I 165/cPCI [6]

Being an undisputed leader in their field [6], IXXAT offers a number of other more portable interface solutions allowing connection to: parallel port (called CANdy), PCMCIA bus (tinCAN) and USB (USB-to-CAN). When purchased the products come with manuals and Windows 98/2000 drivers. Drivers for new operating systems are available from company's website [6].

IXXAT's USB-to-CAN Intelligent CAN Interface (Figure 2.5) is based around Infineon SAB C161 microcontroller (16bit) running at 16 MHz. On board memory consists of up to 1 Mbyte SRAM (basic version 256 kbyte) and up to 512-kbyte FLASH memory. Two Philips SJA 1000 CAN controllers provide two separate CAN channels. USB-to-CAN keeps iPC-I 165/cPCI full functionality as well as being considerably less expensive. Low price, small size (165x85x32 mm), USB connection (USB revision 1.1) and high speed (up to 12 Mbit/s) make this product a very attractive alternative to a PCI CAN interface card.

Figure 2.5 – USB-to-CAN [6]

Moving away from commercial products towards academic research, Markus Traidl and Donal Heffernan from PEI Technologies have designed "A CAN Control Network Gateway to a PC Based on the USB Interface" [7].

The hardware is based on Intel's 80930Ax 16-bit microcontroller with an Intel 80251 microcontroller core. It runs at 12 MHz and includes an onboard USB interface module. The CAN side of the device is controlled by an Intel AN82527 controller. An RS-232 serial port is included to assist in the development and debugging processes, however it is not limited to that use.

The designers have developed a dedicated small real-time operating system kernel (USB-OS kernel) to allow structuring of the real-time software in the microcontroller. The kernel uses pre-emptive scheduling and supports interrupt handling for the USB

routines [7]. A custom Windows98/NT 5.0 device driver has been developed to access the USB-CAN Interface from the host operating system.

Preliminary performance results have been obtained using a not fully completed prototype device. During CAN→USB data transfer tests, an independent CAN node generated 8-byte messages 50 times per second and sent this traffic over the USB-CAN interface to the PC [7]. The average observed delay was 2 ms (minimum: 1.528ms, maximum: 2.522ms) [7], however if the PC operating system is busy the delays can easily exceed 5 ms. Delays encountered during USB→CAN transfers (8 data bytes + identifier 50 times per second) averaged to 1.073ms (minimum: 0.663ms, maximum: 1.574ms) [7].

Further tests were conducted to obtain an indication of the impact USB data transfer type has on transfer latency. These tests were inspired following the preliminary tests where the authors discovered that a considerable speed variation between CAN→USB and USB→CAN transfers is due to the use of different USB bus transfer types. CAN→USB communication used USB's interrupt mode, which places restrictions on the frequency and size of data transfers. On the other hand, USB→CAN used a bulk transfer mode allowing data to be passed immediately, provided the bus is idle. Consequently, CAN→USB transfers have been tested using bulk mode, with successful transmission rate of up to 3500 x 8 byte messages per second, corresponding to CAN data rate of about 400 kbits/s [7].

The authors concluded that the amount of work in the project exceeded their expectations, however they are planing to continue the research. They are determined to meet their ultimate goals of the device operating in real-time and with 1Mbit/s transfer rate.

Chapter 3 – Specifications

The final design was required to meet the following criteria:

- Provide connection between iPAQ and CAN network,
- Data transfer in both directions, at least half-duplex,
- Minimal data transfer delay through the device,
- System speed above 1Mbit/s,
- Microcontroller compliance preserved,
- The device was required to be small, lightweight, sturdy, and noise immune, to allow it to operate inside the robot.

In order to meet the above specifications the design was based on a fast and highly integrated microcontroller. The research included in Chapter 2 did not provide an "out of box" solution. IXXAT's PCI extension card is not appropriate, as the iPAQ does not have a PCI port. Their USB-to-CAN could prove to meet most of those specifications, however the team felt it was too bulky and that a better solution could have been developed.

One of the main requirements the device was required to meet was its ability to transfer data with minimum latency. Since the USB-to-CAN Bridge was to be used in the control environment, short delays become critical. The control loop team was consulted and the frequency of data transfers obtained. It has been agreed that velocity commands from the iPAQ to controller boards would need to be send every 2ms (500Hz). During the 2ms windows additional data would need to be send in the opposite direction (towards the iPAQ) as well. Rough estimates indicated that downstream (iPAQ to controllers) traffic would add up to around 250kbit/s and another ~250kbit/s of upstream transfers. Please note that these numbers do not include overheads introduced by framing bits and communication requirements. Consequently, the overall sum of downward and upward transfers comes under 1Mbit/s. This is very important, as the maximum CAN bandwidth is equal to 1Mbit/s.

Other requirements due to device's working environment relate to its weight and dimensions. The USB-to-CAN bridge board was to be located within robot's chest cavity, together with the iPAQ, vision board, power electronics, and four batteries. As a result, the device needed to be as small as possible and lightweight. The objective was to use small components (e.g. surface mount resistors and capacitors) and the selection of a microcontroller with most of the required hardware on board.

With the device's position close to the switch-mode power electronics, the batteries and vision hardware, it was important to have good noise immunity. Power electronics are in the immediate proximity to the USB-to-CAN Bridge and could induce communication errors. Preventative measures were required and included short tracks as well as appropriate shielding.

Finally, the design needed to be sturdy and securely attached to robot's frame preventing it from damages.

Chapter 4 – Hardware Design

4.1 Main Controller Unit

The immediate application of USB-to-CAN Bridge is GuRoo the Humanoid project. GuRoo required six controller boards to communicate with an iPAQ palmtop that runs high level walking software. This communication needed to be implemented via a CAN network. The bridge was used to interface between controllers' CAN network and a USB port on the iPAQ.

Controller boards were designed based on Texas Instruments' digital signal processor controller TMS320F243. In order to maintain compatibility with controller boards and allow co-operation during software development a decision was made to use the same controller chip in the bridge design. Running the code required for USB-to-CAN Bridge would take only a fraction of processor's calculation power, therefore to further utilise the controller's processing power the team decided to implement power distribution control on the same chip.

A detailed investigation of TMS320F243 revealed its suitability for the design. Its RAM, 544 words, was sufficient to run the software as well as temporarily buffer messages before passing them on. EEPROM, consisting of 8K words [11], provided more than enough room to store code as well as descriptors required by USB. On board Controller Area Network module greatly simplified connection to CAN network and the 16-bit Serial Peripheral Interface (SPI) provided connection for a USB node controller. A 5 MHz external crystal was stepped up using a Phase Locked Loop (PLL) to 20 MHz, to provide the internal clock. It gave the controller more than enough power to receive messages, convert them, and pass them on. Furthermore, a large number of general-purpose I/O pins, external interrupts, three power-down modes, 8 ADC channels and Serial Communications Interface (SCI), among other features, make this controller very versatile and suitable for a wide range of applications [11]. Figure 4.1 shows the block diagram of TMS320F243 providing a visual overview of its modules and how they are connected.

In order to keep all boards similar, one of the controller board designs was taken and stripped from unnecessary hardware to provide the basis for the USB-to-CAN bridge circuit. Extra hardware was added to provide USB connection, resulting in the circuit schematic included in Appendix A. Schematic of the controller and its supporting circuitry is included in Figure 4.2 for easier reference.

From Figure 4.2 it is clear that the controller support circuitry is minimal resulting in PCB space saving and therefore reduced cost. The main sections of the circuitry include

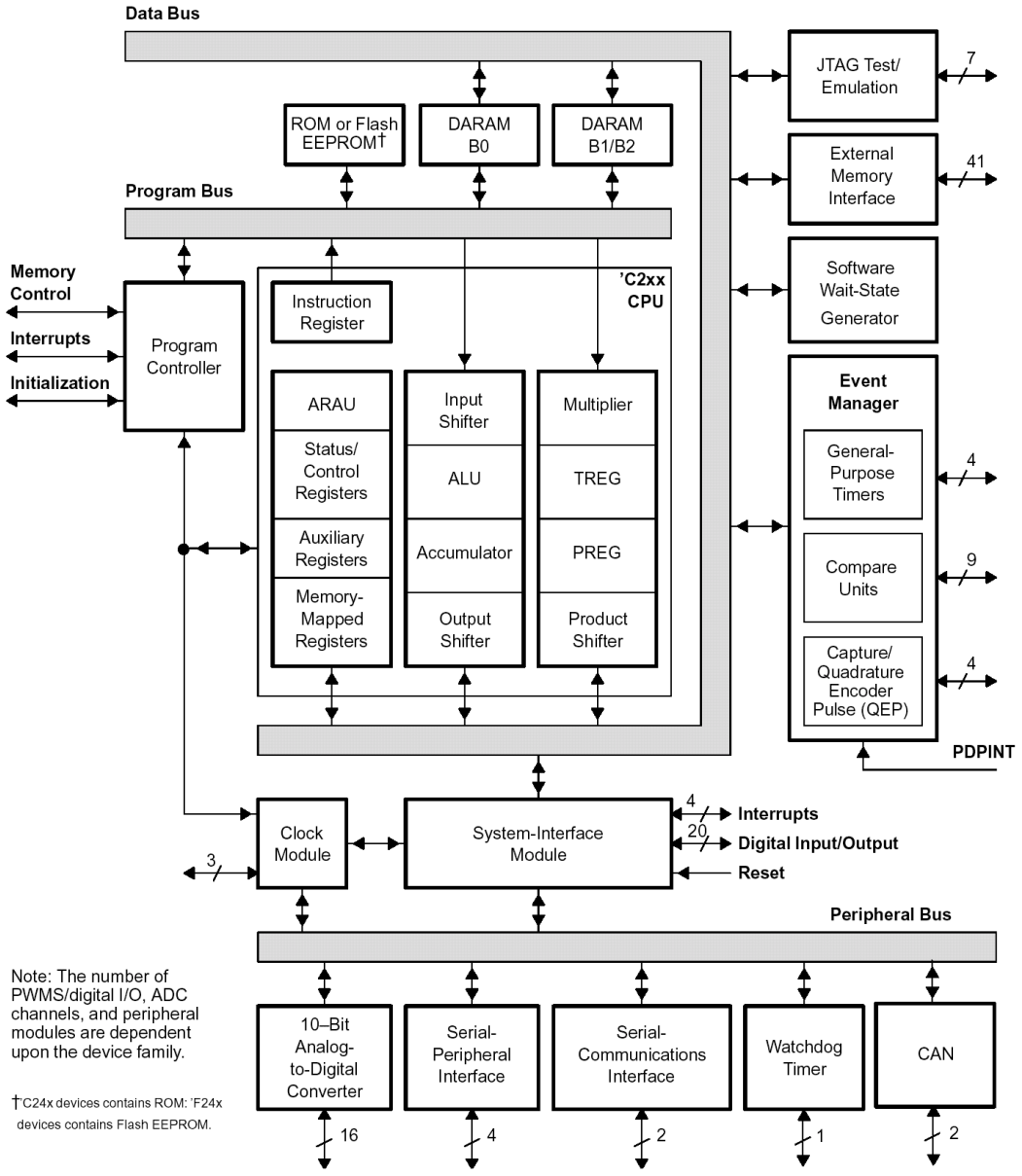

Figure 4.1 – TMS320F243 Block Diagram [11]

the 5 MHz oscillator crystal circuit providing the controller with its clock waveform. Internally, the 5 MHz sine wave was stepped up to 20 MHz as described previously. This 20 MHz signal is available on pin 116 of the controller – the CLKOUT pin. The decoupling capacitors were required in the immediate proximity to all power supply pins ensuring that any AC interference was removed. Due to the overall chip complexity, as well as the push to make microcontroller die as small as possible, different sections of the controller are powered by separate pins. This is a standard practice for complex microcontrollers consisting of various modules like ADCs, PWM channels and so on. Three LEDs and a switch have been included for testing purposes during code development and for diagnostics and error indication during normal operation. The LEDs are connected to general-purpose input/output pins of the controller. The switch is connected to an external interrupt input pin, causing a system interrupt when pressed. When the interrupt occurs, the code execution process is paused and the controller runs the code written to service this interrupt.

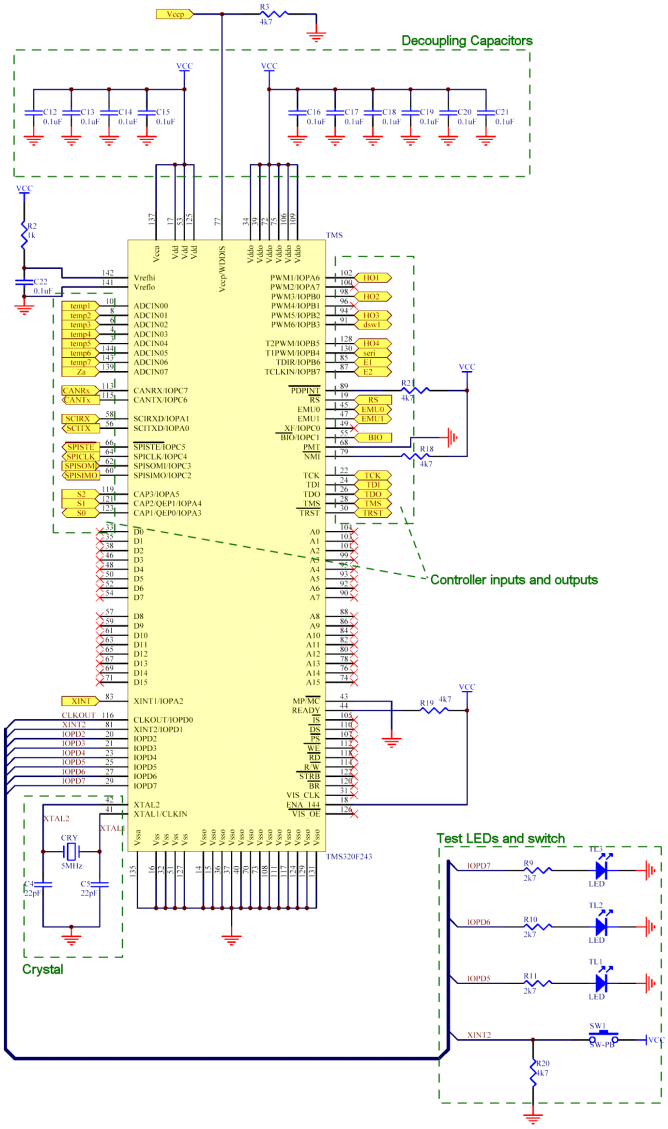

Figure 4.2 – Controller and its supporting circuitry schematic

During the software development process, the controller could have been programmed in two ways: via JTAG or a serial connection. The JTAG connector circuit is provided in Figure 4.3a. It consists of a 7x2 header providing

connection for the JTAG plug and feeds JTAG signals straight into the controller. In order to program through the JTAG, an additional hardware in the form of an extension card that plugs into a PC EISA slot needs to be purchased. The hardware can be purchased from Softronics Pty. Ltd. as a bundle including the extension card, cable with a connector, and software. The serial programmer is an easier alternative to JTAG programmer. The USB-to-CAN Bridge board includes a port, which provides connection for the serial programmer, Figure 4.3b. The actual programmer circuit was temporarily wired up on an additional breadboard, as it was required only during software development for device programming. The main difference between JTAG programming mode and the serial programmer was the fact that the serial programmer requires a boot loader to be loaded up in the microcontroller's memory. JTAG does not have this limitation.

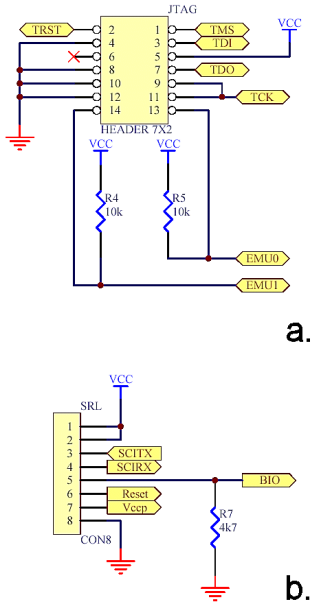

Figure 4.3 – JTAG and serial programmer connectors

The reset circuit in Figure 4.4 uses Maxim's MAX811 reset signal generator IC. The IC works by providing a reset signal, active low, during the controller's power up and power down stages. In addition, the chip monitors the controller's voltage rail (Vcc equal to 5V) producing reset signal whenever voltage drops below 4.85V. This device acts as a safety measure to ensure the controller is fully powered up before it starts executing code. If the controller was allowed code execution at voltages lower than 4.85V its operation would be unreliable and possible corruption of code could occur. As an additional feature, MAX811 allows manual reset via an onboard pushbutton. In USB-to-CAN Bridge, the reset signal generator feeds

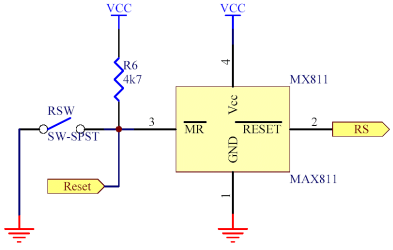

Figure 4.4 – Reset controller schematic

the reset signal to the main controller and USB interface controller.

4.2 Universal Serial Bus Interface

During the research phase of the project a number of iPAQ to CAN communication connections were investigated and the four connections shown in Figure 4.5 were selected as possible solutions for detailed investigation.

Connection in Figure 4.5a uses iPAQ's USB port to connect to a controller with both USB and CAN modules on board. This seemed the best solution requiring minimal amount of hardware and being the fastest one due to both interface modules being integrated into the same processor. The only drawback of this design was the fact that it was impossible to find a microcontroller with both modules on board.

Figure 4.5b takes advantage of iPAQ's Serial Peripheral Interface (SPI) as the connection to a CAN enabled microcontroller. This alternative showed a lot of potential as it was possible to be implemented with TMS320F243, however further investigations revealed some limitations. There was no evidence found of iPAQ's operating system (Windows CE) offering easy access to the SPI interface, and consequently a device driver would have to be written in order to use it. Furthermore, SPI requires the two devices it connects to be very close to one another, preferably on the same PCB. This requirement was possible to achieve, however it would cause

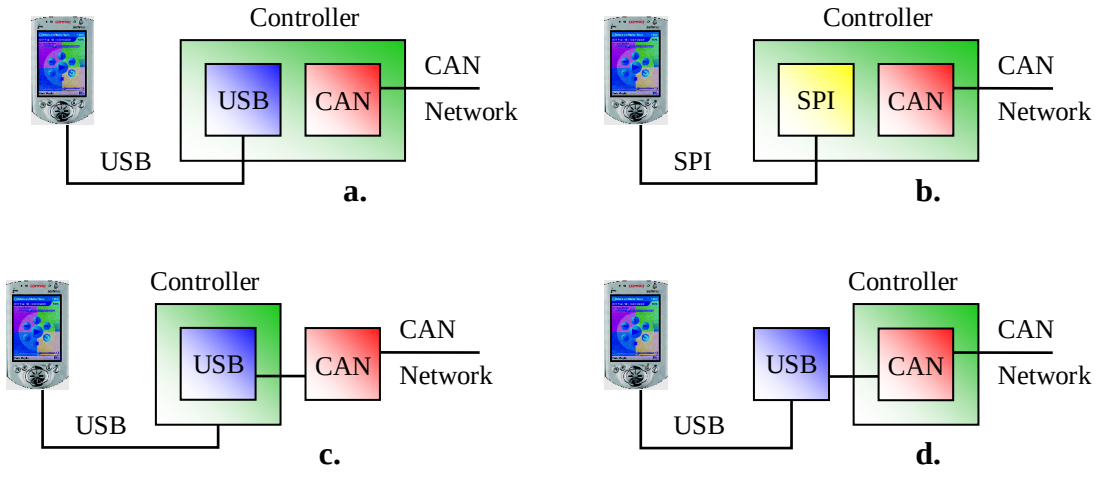

Figure 4.5 – iPAQ to CAN network connection solutions

problems because it would not be possible to include power electronics hardware on the same board. Lastly, it would work only with an iPAQ. It would not be achievable to use a PC (instead of the iPAQ) during software development and debugging stages. Programs would have to be written on a PC, compiled and then downloaded into the iPAQ so that they can be run on the CAN network – an inefficient and time consuming solution.

The third option involved using iPAQ's USB port to connect a USB enabled microcontroller that in turn connects to an external CAN transceiver, Figure 4.5c. This connection was also possible as there are microcontrollers with USB modules on board, however it would involve using a chip other than TMS320F243. This would contradict with team's goal of keeping the microcontrollers the same. Furthermore, when compared to solution in Figure 4.5d, one of the modules needs to be connected externally anyway and there is a significantly larger selection of USB interface chips as opposed to selection of CAN interface chips.

Finally, Figure 4.5d presents the chosen solution. A microcontroller, with onboard CAN module is connected to a USB interface chip that then connects to iPAQ's USB port. This solution allows the use of TMS320F243 as the microcontroller. Because the design utilises a USB interface it is possible to use it with any operating system supporting USB. The hardware is no longer limited to use with iPAQ only, and the distance between the two connected devices can be stretched up to 5 meters. This would, for example, allow the humanoid to perform practice walks while connected to a PC via a 5-meter USB cable – very flexible and versatile. Furthermore, inside the robot's chest the iPAQ can be placed a safe distance away from noise sources. The controller board that has power electronics switching large currents at voltages of around 40V emits a lot of Electro-Magnetic Interference (EMI) and iPAQ was not designed specifically for this kind of environment. By increasing distance between the two devices and introducing additional shielding a significant amount of EMI can be eliminated.

A number of USB interface microchips could have been used in the design. The above selection criteria have reduced this number by half, rejecting all the USB enabled

microcontrollers. Consequently, when it came to choosing a USB interface chip a number of additional requirements needed to be formulated:

1. Connection to one of TMS320F243's interfaces
2. Full compliance with USB specification version 1.0 and 1.1
3. Full speed operation (12 Mbit/s)
4. 5V operation with small current consumption

The three most appropriate chips included PDIUSBD11 and PDIUSBD12 from Philips Semiconductors, as well as USBN9603/4 from National Semiconductors. The only major difference between PDIUSBD11 and 12 is the interface they use. PDIUSBD11 uses I^2C serial interface and PDIUSBD12 uses an 8-bit parallel data bus. Further investigations revealed that PDIUSBN11's I^2C serial interface was not compatible with TMS320F243's SPI interface, therefore rendering it unusable. PDIUSBD12 with its parallel interface can be easily connected to the TMS320F243 microcontroller. It offers full speed operation and is USB specification 1.0 and 1.1 compliant. However, when compared to USBN9603/4, it lacks extensive documentation and support. Philips does provide some example code, but that is just about it. National Semiconductors supplies an extensive datasheet (60 pages) as well as free samples of USBN9603 interface chip. An evaluation board is also available for purchase from their website with its software package being available for download from the webpage as well. The package includes firmware code to be run on the evaluation board, code documentation, evaluation board schematics, programming guide, and a guide to USBN9603 interfacing. This package is available for free download and has the potential to significantly decrease the software development time for the project. Because of the above-mentioned reasons and USBN9603 fulfilling the requirements National Semiconductor's it was chosen for the design.

The Universal Serial Bus interface chip is manufactured by National Semiconductors in two versions: USBN9603 for self-powered devices and USBN9604 for bus-powered devices. The only difference between the two types is their behaviour after a reset. In USBN9604 assertion of the reset signal results in the clock generation circuitry being reset as well, where in USBN9603 the clock generation circuitry is not reset. This is particularly important for bus-powered applications where voltage can fall below

acceptable levels and the clock generation circuitry should be reset as well. Both chips come in two types of 28 pin packages: a standard size 28-pin SO package and miniature 28-pin CSP package. The size difference between the two packages is shown in Figure 4.6. Because USB-to-CAN Bridge has its own power source it is based on USBN9603 in SO package to provide better access to the pins during testing procedures.

Figure 4.6 – USBN9603/4 packages [10]

USBN9603 is a very versatile full speed (12 Mbit/s) USB transceiver with three receive endpoints, three transmit endpoints, and one bidirectional endpoint. It is USB specification versions 1.0 and 1.1 compatible and can be operated at either 5V or 3.3V with a typical operating current of 30 mA [8]. It provides an 8-bit parallel interface as well as a MICROWIRE/PLUS™ serial interface for easy connection to any

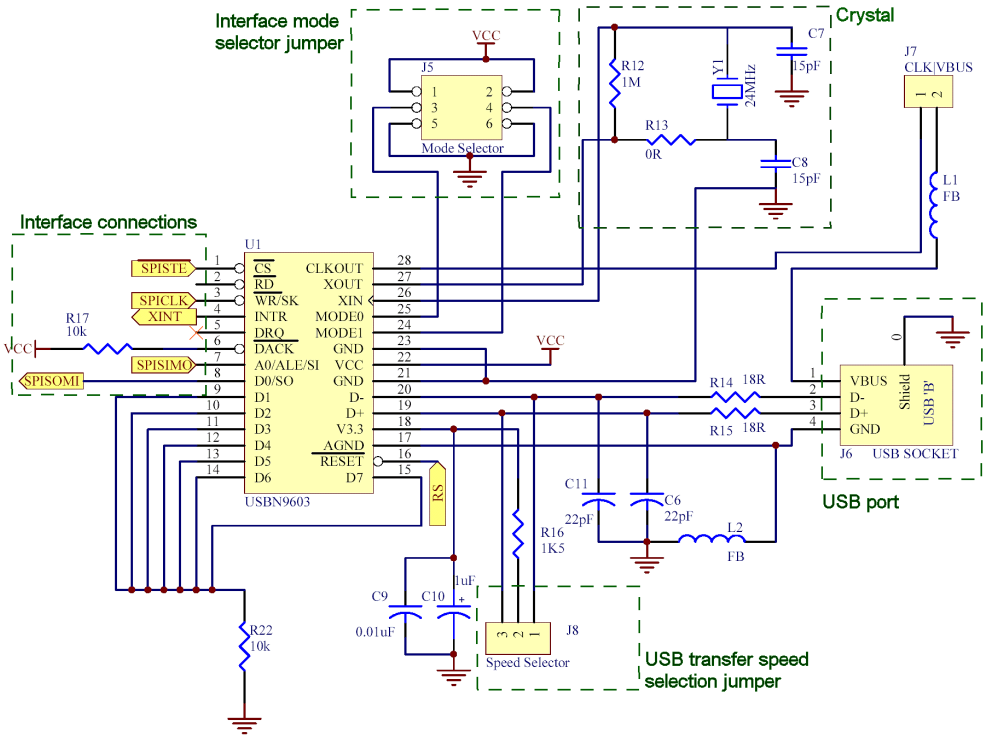

Figure 4.7 – USB interface controller and supporting circuitry

microcontroller. It has been investigated that MICROWIRE/PLUS™ serial interface is compatible with an SPI interface and the two chips can use it to communicate.

Figure 4.7 illustrates the schematic of USBN9603's connections. The interface mode selector jumper is used to choose the interface USB controller should use to communicate with the main controller. There are three modes available: Non-multiplexed parallel, Multiplexed parallel, and MICROWIRE (serial). It was decided to use the MICROWIRE interface as it requires fewer wires and is mode resistant to external interference. In order to select that mode, jumper pins 3 and 4 need to be connected to pins 5 and 2 respectively. The MICROWIRE interface itself consists of four wires, which on the schematic are shown as signals. Signals SPISIMO (SPI Slave In Master Out) and SPISOMI (SPI Slave Out Master In) are input and output lines respectively. Signal XINT is connected to the main controller's interrupt pin causing an interrupt whenever the USB controller requires servicing. The SPI, being a full duplex serial synchronous communication interface requires a clock signal provided via SPICLK signal line.

Other components include USB transfer speed selector jumper used to switch the transmission speed between full speed (12 Mbit/s) and slow speed (1.5 Mbit/s). The crystal used is a 24 MHz crystal connected as per circuit included in the datasheet. The 24 MHz clock is internally stepped up to 48 MHz to provide a reference for USB sampling frequency. Most of the capacitors included in the circuit, as well as the two Ferrite Beads (L1 and L2), are there to reduce noise and interference. Resistors R14 and R15 are included for the purpose of USB cable impedance matching.

4.3 Controller Area Network Interface

The main CAN module is located within the TMS320F243 microcontroller, but it requires an external driver chip between its CAN module and the physical bus. The chip selected to perform this function is PCA82C250 from Philips Semiconductors. Its role is to provide over-current protection of the CAN module and supply enough power to transmit data. A current limiting circuit was designed to handle a short circuit

situation by making the chip dissipate all of the power. Once the temperature rises above 160 °C, the current limiting circuit decreases the current resulting in lower power dissipation and prevention of driver destruction. The driver supports a maximum speed of 1 Mbit/s, which is in line with TMS320F243 specifications. PCA82C250's connections are shown in Figure 4.8.

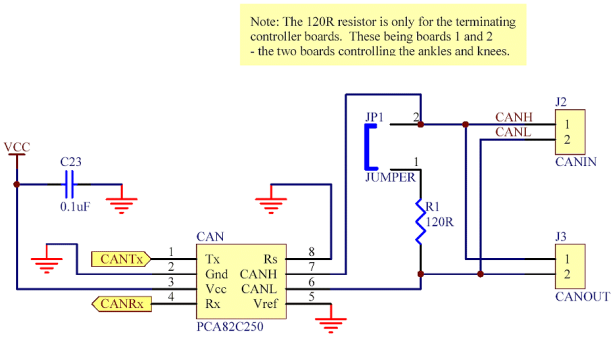

Figure 4.8 – CAN Driver Connections

Chapter 5 – Software Implementation

5.1 Introduction

The microcontroller software controlling the USB-to-CAN Bridge was written in C. C was chosen over assembler mainly to improve code readability and speed up code development. James Kennedy in his thesis [9] has questioned the speed of firmware produced by Texas Instruments C compilers, however taking into account microcontroller's high speed it should not cause a problem. Firmware consists of the main file called `Enum.c` and a number of header files linked to the main file.

Because of software complexity, a decision was made to write the code using a hierarchical structure. Consequently, at the lowest level there are simple routines that are responsible for single tasks. The higher-level routines use those primitive lowest level routines to perform more complicated tasks and so on until the highest-level routines that perform time consuming or complicated jobs. This hierarchical structure is straightforward, easily understood, and readily adapted and tested [10].

The USBN9603 related code developed for this project has been based on the firmware shipped with USBN9603/4 evaluation board. National Semiconductor provides the firmware (and its documentation) to reduce development time for the USBN9603 Universal Serial Bus Function Controller. The software is available from National Semiconductor Website [10], free of charge.

At this stage, the software is not completed, lacking USB → CAN and CAN → USB message conversion modules. The software provided in this thesis is responsible for enabling communication between USB-to-CAN Bridge and the iPAQ. The firmware enables the iPAQ to recognise and enumerate the USB-to-CAN Bridge upon its connection.

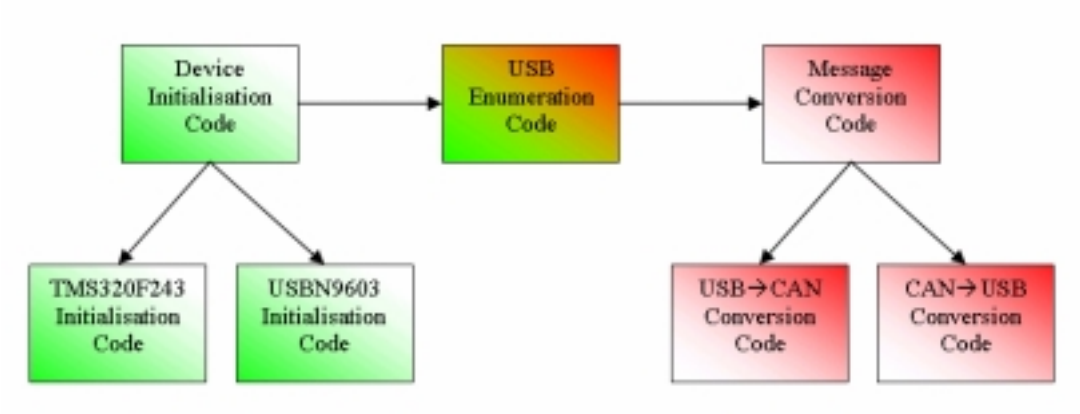

Figure 5.1 – Software Progress Diagram

Figure 5.1 represents a graphical representation of the software development progress. The green shaded boxes represent code that has been developed and tested, while the red shaded boxes mark software components that have not yet been developed. The box shaded both green and red represents the firmware that has been compiled, but remains untested.

5.2 Hardware Initialisation and Configuration

Both, the main microcontroller and the USB interface controller need to be initialised before they are ready to transfer data. The TMS320F243 is initialised within `init_F243()` routine, which also initialises its SPI interface by executing `init_SPI()` routine. Firstly, the routine disables all interrupts, to prevent them from occurring during the initialisation process, it sets up the watchdog module and System Control and Status Register (SCSR). Most of the microcontroller pins can work in one of two modes, usually either as a general purpose IO pin, or as an IO pin for one of the internal modules (ADC, SPI, SCI, and so on). The pin assignment is controlled by two registers: OCRA and OCRB (IO Control Register A/B) and those registers are configured to enable SPI interface, CAN module and external interrupt pins.

TMS320F243 interrupts can be divided into two groups: Hardware generated (including external and on-board peripheral interrupts) and software interrupts. The two groups add up to over 30 interrupts some of which that can be masked. Such a large

number of interrupts being enabled at the same time would have a devastating affect on the microcontroller's performance. The main code execution speed would be significantly decreased, as the microcontroller would spend a lot of time servicing various interrupts. Consequently, in order to make the data transfer as fast as possible, an effort was made to reduce the amount of interrupts used to a bare minimum. There is nothing that can be done about the nonmaskable interrupts, however most of the maskable ones can be turned off. At this stage, the only interrupts required are SPI interrupts and the external interrupt (XINT1) used by the USB interface chip when it needs servicing.

Apart from being able to mask interrupts, TMS320F243 also allows some degree of freedom when it comes to assigning interrupt priorities. Figure 5.2 shows the hardware generated interrupt hierarchy. For instance, interrupt XINT1 can be either Level 1 or Level 6 interrupt. Level 1 interrupts have the highest priority and only the most important interrupts should be allowed to operate as Level 1. In this case, XINT1 is configured as Level 1 interrupt because the USB interface chip should be serviced as quickly as possible to ensure minimal message transfer times. Whenever this interrupt occurs the main controller accesses USBN9603's internal registers to find out what needs to be done.

After enabling the required interrupts `init_SPI()` routine is executed to initialise the Serial Peripheral Interface module. The routine firstly resets the SPI module and then sets it up from scratch. TMS320F243's SPI interface needs to be initialised so that it is compatible with USBN9603's interface. Therefore, since USBN9603 is an 8-bit device it transmits and receives only 8-bit packets of information. However, TMS320F243 is a 16-bit device and by default, it sends 16-bit of data in each packet. In order to make the two communicate effectively, the main controller has been forced to use only 8-bits per package.

The maximum SPI baud rate has been set to 2.86 Mbps as limited by the USB interface chip. USBN9603 datasheet specifies the maximum transfer rate of 3 Mbps. TMS320F243, being the master, is responsible for providing the USB interface controller with an SPI clock and its closest value to 3Mbps was calculated to be 2.86 Mbps. Compared to CAN's maximum speed of 1 Mbps this interface should not have any problems handling the traffic.

Once the main controller and its modules (especially SPI) are initialised, the USB interface chip can be accessed for initialisation as well. Firstly, a software reset is performed to ensure the chip is in its default state and it will not try to communicate with a USB host. USBN9603's interrupts are configured, the default receiver turned on and the chip itself enabled.

5.3 Main Loop

The main loop is a vital routine within the firmware, however it should be mentioned as well. The routine, `main()`is always run when the processor is reset or powered up and because of that the first thing it does is to initialise the hardware. TMS320F243's hardware is initialised when `init_F243()` is called and USBN9603 is set up using `init_usb()`.

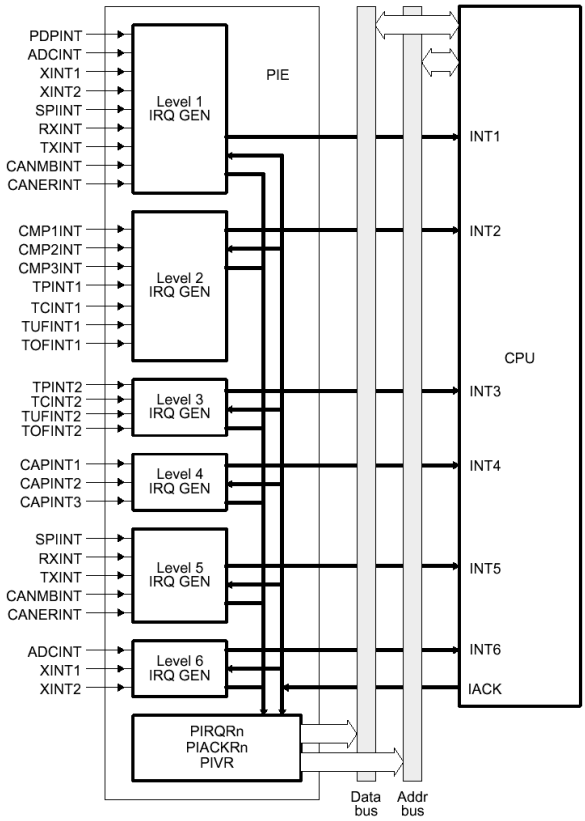

Figure 5.2 – Peripheral Interrupt Expansion Block Diagram [11]

At this stage of the firmware development, the routine simply progresses into time wasting mode, repetitively executing the

`while(1)` statement. However, when an interrupt occurs this execution is put on hold and the microcontroller jumps to appropriate interrupt service routine. Once the service routine has finished, the execution comes back to the `while(1)` statement and the microcontroller keeps executing it until the next interrupt. Later on, another more useful statement that, for example, monitors the board temperature and alerts the iPAQ if it gets too high can replace this line.

5.4 Interrupt Handlers

The main objective of this project was to provide timely exchange of information between CAN and USB and vice versa. In addition, in order to properly enumerate the device and then stay configured, the USB devices need to respond very quickly and at regular time intervals. These are the reasons why the USB-to-CAN Bridge firmware is interrupt driven. This means that whenever the device is addressed on the USB bus, the USB controller chip triggers an interrupt alarming the main controller as opposed to TMS320F243 periodically accessing the USB controller to find out if it needs for servicing. By using interrupts, the design ensures the main controller does not waste time unnecessarily accessing the USB interface chip.

A flowchart of the main USB interrupt handler is provided in Figure 5.3. This routine executes whenever TMS320F243 receives XINT1 interrupt, meaning the USB chip requires servicing. In other words, whenever anything important happens on the USB bus this routine is executed. USBN9603's Main Event Register (MAEV) stores the information of the reason behind throwing the interrupt. Therefore, it is accessed first to find out what has happened. Then, the routine checks for three cases: receive, transmit and NAK. Receive means that the host has transmitted requests to the device. RXEV register is read to find out which endpoint has been written to, and control passes to a routine that will transfer the data from USB chip to the main controller. The occurrence of a transmit event usually means that a previously queued transmission has been completed, or has encountered an error. Just like in the case of the receive event a register is accessed (TXEV) to find out which routine should be executed. Finally, the NAK event means that the USBN9603 generated a NAK (Negative Acknowledge)

handshake on the USB bus. NAK packet is sent when the device is busy or it has no data to send to its host. Hosts never send NAKs. USB-to-CAN Bridge sends NAK packets only from endpoint 0 (receive and transmit). Endpoints 1 and 2 operate as isochronous endpoints and they do not support handshaking and therefore cannot return NAK. This is justified by the idea that in isochronous transmission mode data needs to be transferred in real time and there is no time for retransferring. If the device misses isochronous data, there is no retransmission.

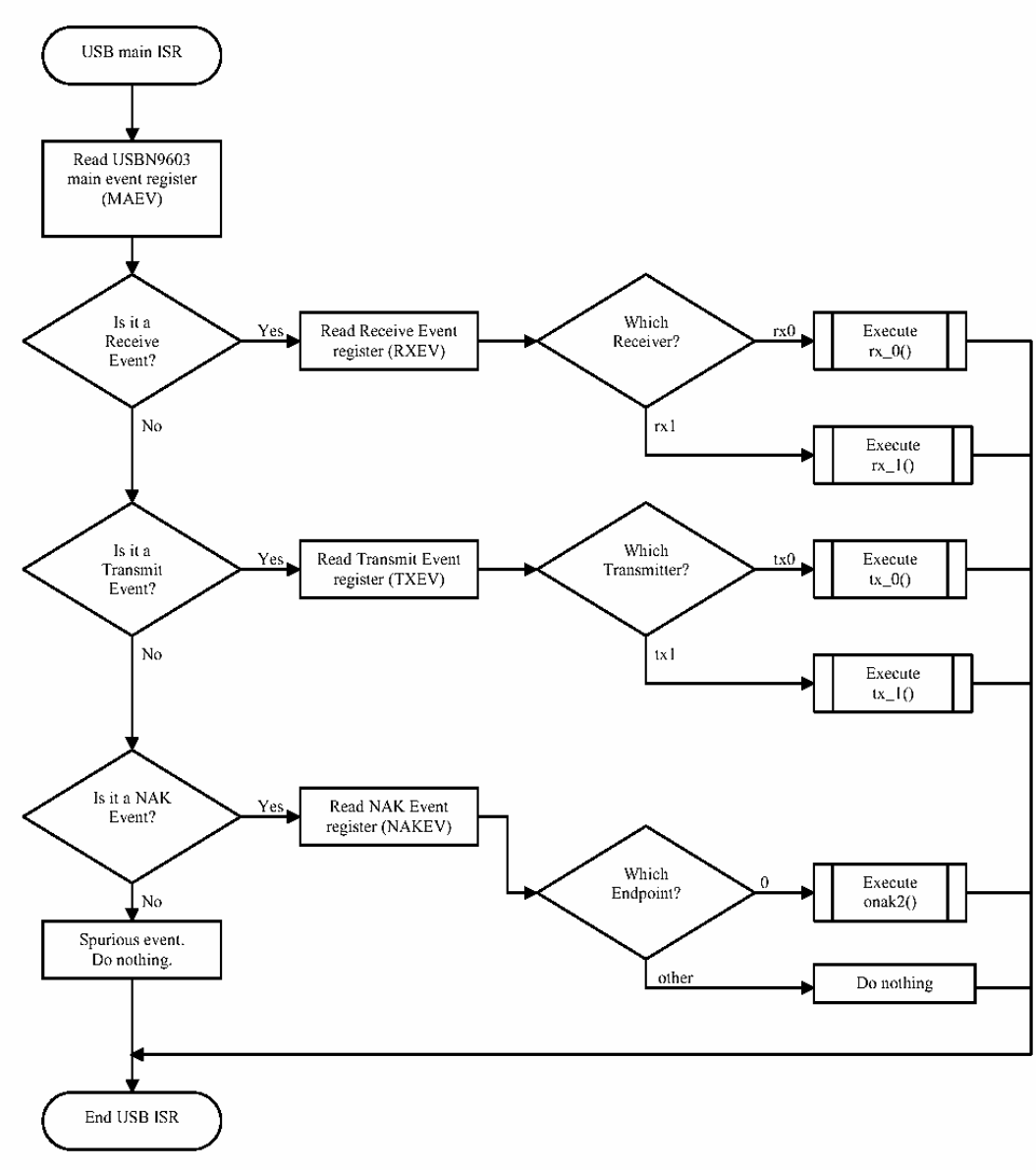

Figure 5.3 – Main USB servicing routine flowchart

USB servicing routine, `usb_isr()`, services one interrupt occurrence at a time. It is quite possible that more than one interrupt event may occur at the same time and the additional interrupts will be picked up once the interrupts are again enabled after `usb_isr()`, is completed. In addition, the order in which interrupt events are checked for is not accidental. NAK events are checked for last, to give then the lowest priority of the three types. They can be issued very often, which if they were checked for first could potentially result in the remaining two options being starved out.

5.5 CAN-to-USB and USB-to-CAN Message Conversion Code

This part of the code has not yet been developed, therefore only general concepts will be conveyed in this section. It is essential that this fragment of code is a fast as possible as it will be accessed very frequently and can significantly contribute to the overall delays.

The idea is to take data from a USB frame, extract CAN framing information and iPAQ commands from it, create a CAN frame and send it on the CAN network. At this stage, there is no information available as to how CAN framing information and iPAQ commands are to be placed for USB transfer.

Similarly, when transferring data from the controller board to iPAQ an opposite procedure needs to be followed. CAN messages need to be received, required data extracted formatted as USB messages and sent off.

Chapter 6 – Product Evaluation

6.1 General Comments

At the time of writing this thesis, the USB-to-CAN Bridge was not yet operational, mainly due to unfinished microcontroller software. The factors responsible for this disappointment, were the series of delays during the hardware development stage. The initial assumption that a lot of work could have been performed in parallel was not entirely correct. Consequently, a lot of time was spent working on controller boards' hardware before they could have been modified for the USB-to-CAN Bridge design. However, microcontroller software development will continue and the device should be at least semi operational at the end of the semester.

6.2 Meeting the Specifications

6.2.1 Hardware Evaluation

The hardware has been tested, however more tests are required before one can say it fully works. On the other hand, there is nothing to indicate any serious mistakes or faults.

Initially the signal on the CLKOUT pin of the TMS320F243 was displayed on a CRO to ensure the microcontroller has a working clock. A 20 MHz square wave was observed indicating the microcontroller is working. The microcontroller has been tested and programmed using both JTAG and serial port. Small test programs were run to test external interrupts and LEDs. The testing was successful as confirmed by correct output to the external LEDs.

USBN9603 could not be extensively tested due to the uncompleted firmware. CLKOUT pin has been investigated and expected waveform observed.

A four layer Printed Circuit Board (PCB) included three minor errors that have been corrected during component population process. The first one involved an incorrectly

designed switch footprint and was corrected by rotating the switch by 90º. The second error involved the serial communication port connector being changed without team notification. The change was justified by the desire to use already manufactured serial programming modules rather than designing new ones. Consequently, a prototype of a serial programming module had to be build to be used with the USB-to-CAN Bridge. In order to minimise delays involved in design and manufacturing of new PCBs, the programmer was implemented on a breadboard. Finally, an incorrect CAN driver footprint was used, resulting in additional work needed to place that component.

6.2.2 Software Evaluation

The software was written in C and currently consists of untested USB enumeration code. It is hoped that by the end of the semester the code will be completed and the USB-to-CAN Bridge working. As a result, this does not leave much time to test software performance and overall latency times.

6.4 USB Specification 1.1 Compliance

The device is fully USB specification 1.1 compliant apart from the USB connector used. According to USB specifications 1.0, 1.1 and 2.0 the upstream port of a device (in this case the one connected to iPAQ) has to use series-B receptacle. The USB-to-CAN Bridge uses a series-A receptacle that is reserved for use with downstream ports only. This specification requirement has been put in place to prevent users from connecting devices wrong way round. The USB-to-CAN Bridge does not comply with this requirement because of its connection requirements to the iPAQ. A standard USB cable cannot be used because the iPAQ has a non-standard USB connector. The USB port is implemented within a proprietary 12-pin connector (STRATAC 12P), which also provides connection to RS-232C interface and battery charging circuitry. Compaq (iPAQ manufacturer) only provides a STRATAC→USB ‘A’ type cable and in line with USB specifications a STRATAC→USB ‘B’ type cable needs to be used. This is due to the fact that iPAQ has been designed to mainly act as a USB device rather than a USB host.

Consequently, if full USB Specification compliance was to be preserved a custom made cable would need to be manufactured. The problem with this was that the STRATAC connector is not freely available. Foxconn, the only manufacturer of this connector has been approached for samples, but refused to provide any. Purchasing was also ruled out as the company was prepared to only deal in thousands of units or more.

6.5 Performance as an Engineer

At this point in time, it is important to stop and evaluate the performance as an engineer in the USB-to-CAN Bridge development process. Good engineering techniques are essential in any project development and can contribute to successful project completion.

6.5.1 Strengths

Majority of strengths have been utilised in the first part of the project. Due to extensive exposure to hardware development, component selection process was completed ahead of time. That followed by successful circuit design and implementation.

Previous experience in PCB CAD design using Protel 99 proved to be a significant advantage. A four layer PCB was designed and then manufactured. Performed testing did not find any obvious errors and no corrections had to be added.

The main idea throughout the project was to take a top-down approach to system design and problem solving. Meaning that big problems were broken up into smaller ones and so on until manageable size sub-problems were addressed. These smaller obstacles required further investigation and research, however once solved they added to the overall problem solution.

A comprehensive research stage was conducted to ensure most of potential problems and pitfalls are isolated during this project stage. Strong emphasis was put on the fact that the later in the project errors are discovered the more costly they are. This applies to both financial and time budgets.

Time was taken to assist team members with their problems, keeping in mind that this is a team project and if it is to be is a success, it requires team effort. Consequently, time was spent in other areas of the GuRoo project, in some extent resulting in the USB-to-CAN Bridge not being completed within the provided time frame.

6.5.2 Areas of Improvement

A major area of improvement involves software development. Limited knowledge of C programming language was the major problem. An effort was made to obtain sample code to be used as a reference, however at the end, time restrictions and frustration took over.

Insufficient time provisions for external problems have resulted in time budget blow out. Problems encountered with JTAG programming hardware were totally unaccounted for and resulted in the work being suspended for a couple of weeks. In the future, more effort needs to be taken to account for those events.

Improvements in the area of time management would also be beneficial. Throughout the project, a reasonable effort was made to ensure adequate progress is sustained. However, by implementing simple time management techniques the work could have been performed more effectively and with less stress.

6.5.3 Areas of Skill Development

Taking part in a team-oriented project allows learning more about team dynamics and intra-team communication. It was very helpful to know that if in doubt team members were available for consultation and assistance. Having 11 other team members working on the GuRoo has been a great motivation, especially with the knowledge that USB-to-CAN Bridge is an essential part of the project.

The large amount of research conducted into component selection has provided additional insights. The fact that a component can do its job is important, however the amount of company support in the form of engineering samples, evaluation boards, or example firmware is very important as well. At the end of the day, those additional benefits contribute to shorter development time and therefore lower product cost.

A major improvement in C programming skills has been observed and will continue to improve through further work in that field after thesis completion. It has been realised that those programming skills are essential to further professional development.

Chapter 7 – Conclusions and Pointers for Further Work

7.1 Conclusions

The final design was not completely operational at the time this thesis was written. However, it will be operational by the end of the semester. No major errors and omissions have been identified and individual sections of the hardware are working correctly.

Additional work needs to be completed before the device can be used in the environment it was intended for. Software needs to be completed and extensive testing conducted, including tests relating to the device's data passing delays.

When completed, the device will not be only limited to its current application. Through slight modifications, it can be manufactured on a separate PCB and used as a standalone device. Other projects within the school (e.g. SunShark) can utilise its potential to monitor CAN activity within their designs.

In general, the overall GuRoo project has been an ambitious endeavour and it is unfortunate that it was not completed on time. However, the team has worked successfully together assisting fellow members, and ensuring the continued project progress.

7.2 Future Work

- The main priority is to complete the firmware development and fully test the device operation. Once achieved, potential pitfalls and errors can be isolated and further work completed to eliminate them.

- Software testing could be performed to investigate exact delays through the device and compared with M. Traidl's and D. Heffernan's work in [7]. Once

investigated, the research into how to improve these characteristics could commence and firmware optimisations implemented.

- The device could be generalised and a Universal USB-to-CAN Bridge developed for use in any application. This would require reengineering the device based on a slower microcontroller to lower the overall cost of the device.

References

[1] Compaq Computer Corp., Intel Corp., Microsoft Corp., NEC Corp., *Universal Serial Bus Specification Revision 1.1*, 1998.

[2] M. Zerkus, J. Lusher, J. Ward, "USB Primer – Practical Design Guide", Circuit Cellar, http://www.circuitcellar.com (current: 16/6/01).

[3] J. Axelson, *USB Complete - Everything You Need to Develop Custom USB Peripherals*, Lakeview Research, Madison, 1999.

[4] S. Nilsson, "Controller Area Network – CAN Information", http://www.algonet.se/~staffann/developer/CAN.htm, (accessed: 11/10/2001)

[5] R. Bosch GmbH, "CAN Specification Version 2.0", http://www.bosch.de/k8/can/docu/can2spec.pdf, (accessed: 09/05.01).

[6] Ixxat Automation GmbH, "iPC-I 165/cPCI – Intelligent CAN interface for compact PCI bus systems", http://www.ixxat.de/english/produkte/canprod/interf/165cpci.shtml (accessed: 17/10/01)

[7] M. Traidl, D. Heffernan, *A CAN Control Network Gateway to a PC Based on the USB Interface*, Irish Signals and Systems Conference, 1998

[8] National Semiconductors, *USBN9603/USBN9604 Universal Serial Bus Full Speed Node Controller with Enhanced DMA Support Datasheet*, March 2001.

[9] J.M. Kennedy, *Design and Implementation of a Distributed Digital Control System in an Industrial Robot*, undergraduate thesis, University of Queensland, St Lucia, Department of Computer Science and Electrical Engineering, 1999.

[10] J. Lyle, “USBN9602 Firmware Description”, National Semiconductor, http://www.national.com/appinfo/usb/0,1808,00,00.html, (accessed: 10/05/01).

[11] Texas Instruments Inc., *TMS320F243, TMS320F241 DSP Controllers*, Literature No. SPRS064C, Texas Instruments Incorporated, 2000.

[12] F. Bormann, “Texas Instruments DSP – Laboratory, ‘Hands on TMS320F243/LF2407’”, University of Applied Sciences Zwickau (FH) (Germany), http://www.fh-zwickau.de/tutorial/dsp/, (accessed: 18/10/01).

Appendix A – Hardware Schematic and PCB Layout

A.2 - USB-to-CAN Bridge Schematic

A.3 - PCB Layout and Photograph

A.3.1 – Entire PCB Layout (including power electronics layout) Scale 1:1

A.3.2 – USB-to-CAN Bridge Layout Close Up (Power Electronic Circuitry Removed)

A.3.3 – Protel 99 3D board model (USB-to-CAN Bridge)

A.3.4 – Actual PCB Photograph

Appendix B – TMS320F243 Firmware

B.1 - Def9602.h [10]

B.2 - Regs243.h [12]

B.3 - Enum.h

B.4 - Enum.c

Appendix C – Datasheets

C.2 - TMS320F243 Datasheet

C.3 - USBN9603 Datasheet

C.5 - CAN interface datasheet

C.6 - MAX811 datasheet