Edith Cowan University

Research Online

Theses: Doctorates and Masters

Theses

1998

Designing a higher layer protocol for small distributed microcontroller systems using the control area network protocol

Long G. Nguyen
Edith Cowan University

Follow this and additional works at: https://ro.ecu.edu.au/theses

 Part of the Engineering Commons

Recommended Citation

Nguyen, L. G. (1998). *Designing a higher layer protocol for small distributed microcontroller systems using the control area network protocol.* Edith Cowan University. Retrieved from https://ro.ecu.edu.au/theses/1603

This Thesis is posted at Research Online.
https://ro.ecu.edu.au/theses/1603

Edith Cowan University

Copyright Warning

You may print or download ONE copy of this document for the purpose of your own research or study.

The University does not authorize you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site.

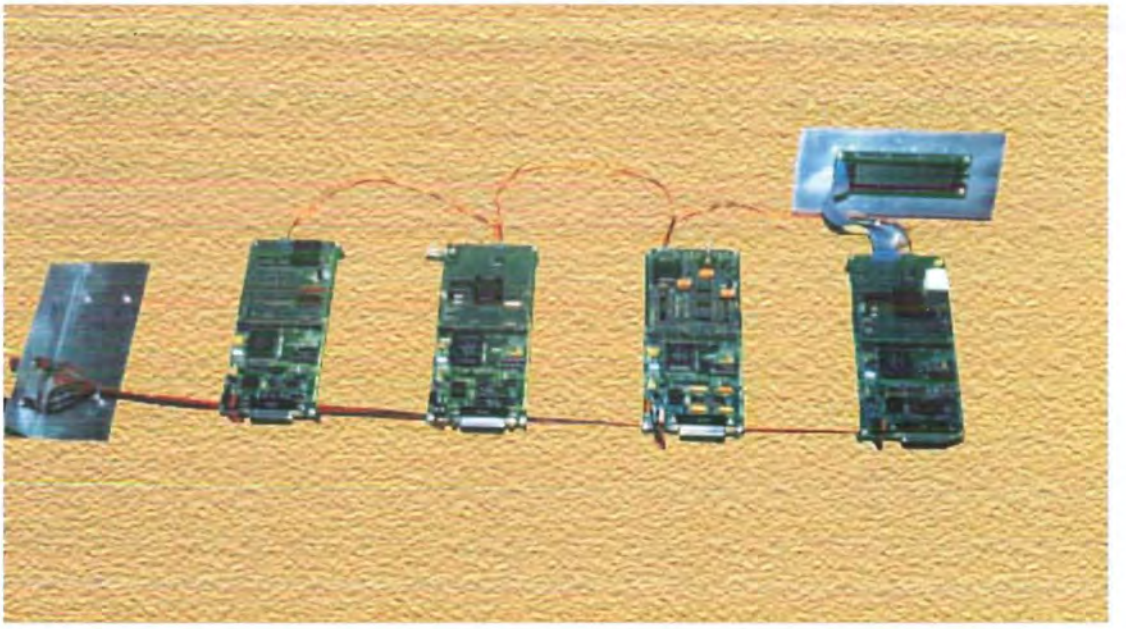
You are reminded of the following:

- Copyright owners are entitled to take legal action against persons who infringe their copyright.

- A reproduction of material that is protected by copyright may be a copyright infringement. Where the reproduction of such material is done without attribution of authorship, with false attribution of authorship or the authorship is treated in a derogatory manner, this may be a breach of the author's moral rights contained in Part IX of the Copyright Act 1968 (Cth).

- Courts have the power to impose a wide range of civil and criminal sanctions for infringement of copyright, infringement of moral rights and other offences under the Copyright Act 1968 (Cth). Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.

DESIGNING A HIGHER LAYER PROTOCOL FOR SMALL DISTRIBUTED MICROCONTROLLER SYSTEMS USING THE CONTROL AREA NETWORK PROTOCOL



EDITH COWAN UNIVERSITY
LIBRARY

Long Giang Nguyen
MEngSc (1993)

DESIGNING A HIGHER LAYER PROTOCOL FOR SMALL DISTRIBUTED MICROCONTROLLER SYSTEMS USING THE CONTROL AREA NETWORK PROTOCOL

L. G. NGUYEN

MEngSc

(1998)

USE OF THESIS

The Use of Thesis statement is not included in this version of the thesis.

DESIGNING A HIGHER LAYER PROTOCOL FOR SMALL DISTRIBUTED MICROCONTROLLER SYSTEMS USING THE CONTROL AREA NETWORK PROTOCOL

By

Long Giang Nguyen

BEng, Grad DipSc (Computer Studies)

A Thesis Submitted in Fulfilment of the Requirements for the Award of

Master of Engineering Science

at the School of Engineering and Mathematics.

Edith Cowan University,

Perth, Western Australia.

Date of Submission: 3rd September 1998

ABSTRACT

This thesis is concerned with designing a Higher Layer Protocol (HLP) for small distributed microcontroller systems using a well-established network protocol: the Controller Area Network (CAN) protocol which, currently, is widely used in the automation industries. Steps were taken to investigate three popular HLPs based on the CAN protocol: namely, Smart Distributed System (SDS), DeviceNet, and CAN Kingdom. Following the comparison of the three HLPs, the CAN Kingdom protocol was chosen for the task of designing the HLP in this project in order to satisfy the restrictions associated with small systems. Thus, the HLP (named the Small CAN Kingdom protocol) of this project was designed according to the principles of the CAN Kingdom protocol, which contains many advantages for open network solutions. This enables designers to enhance a system's performance relatively easily.

A complete hardware and software design of a small CAN-based system, utilising the Motorola MC68HC11 microcontrollers, the Intel 82527 CAN controller chips, and DS3695 (RS485 standard) transceivers, has been described. This small system can be used to demonstrate the performance of the Small CAN Kingdom protocol. The development of the system software has also taken into account the rules associated with this protocol.

DECLARATION

I certify that this thesis does not incorporate, without acknowledgment, any material previously submitted for a degree or diploma in any institution of higher education and that, to the best of my knowledge and belief, it does not contain any material previously published or written by another person except where due reference is made in the text.

Long Giang Nguyen

Date: 03/09/1998

ACKNOWLEDGEMENTS

I would like to take this opportunity to extend my gratitude to people, without whom this thesis could not be completed.

First of all, I would like to extend my sincere appreciation to Mr Barry Kauler for giving me the opportunity to participate in this research field, for his advice and support when I started my study and during my project progress.

Secondly, I would like to express my indebted gratitude to Mr Mike Wetton, my supervisor, for his countless support and guidance throughout the completion of this project. His sympathy, encouragement, enthusiasm, and patient assistance during the difficult periods were very much appreciated.

I sincerely thank Dr Binh Anson, Mr Ian Morris and Dr David McDougall who all provided me assistance with my written English expression.

I also thank A/Prof. Abdesselam Bouzerdoum, Dr Hon Cheung, Dr Daryoush Habibi, Dr Xiaoli Zhao and all the staff members of Engineering Department, Library, and International Office at Edith Cowan University for their assistance during my study.

Finally, I would like to extend my genuine and heartfelt gratitude to my sister, my parents, my grandparents, my fiancee, Mrs Huong and my friends for their love, encouragement, support and sacrifices. Without them, I would not have reached this far.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Overview

Demand for the use of "intelligent" devices to control manufacturing processes in automation industries has rapidly increased in the recent years. Accordingly, these devices are forced to transfer data between each other. A critical issue for the communication between such devices is the use of protocols.

The overall aim of this thesis is to design an application layer protocol (Higher Layer Protocol) for a small distributed microcontroller system based on the Controller Area Network (CAN) protocol, which is a widely-used network protocol in automation industries.

According to Cena and Valenzano (1995), one of the most essential requirements for industrial networks is that they must guarantee deterministically bounded response times; and hence, the protocols utilised in such networks have to satisfy this requirement.

There is a range of architecture and protocols which has been used in industrial networks to date, such as CAN, SP-50 FieldBus, MAP, Profibus, and FIP (Zuberi & Shin, 1995). Among these networks, CAN has gained widespread acceptance in industry due to its speed, low-cost network architecture, and especially, high reliability in noisy environments. According to Farsi and Ratcliff (1997), CAN has long been a market leader in the industrial fieldbus arena; and by the end of 1995, more than 6 million CAN nodes were installed (Table 1-1).

Table 1-1 Cumulative bus nodes sold up until the end of 1995

Bus Chips	Cumulative Quantity
P-Net	35,000
FIP	65,000
ASI	80,000
Profibus	500,000
Interbus S	1,000,000
LON	1,500,000
BITBUS	2,500,000
CAN	6,000,000

The CAN protocol provides users with many powerful features including multimaster functionality, and the ability to broadcast and multicast telegrams. The most important characteristic of the protocol is its priority-base arbitration which allows short response times for high priority messages (see Chapter 2).

However, to ensure inter-operability between CAN components, several Higher Layer Protocols (HLPs) have been developed to allow devices to communicate with each other in a standardised manner. These include Smart Distributed System (SDS), DeviceNet, CAN Kingdom, CANopen, and SAE J1939 (Korane, 1996; Farsi & Ratcliff, 1997). Chapter 3 of this thesis reviews the first three protocols in detail, especially, their current use in a broad range of industrial applications (Bladin, Bradley, Danioux, Gray, & Loaic, 1997). Each one of these protocols has taken a different approach for controlling systems, with respect to the application services provided to its users.

Although many HLPs have been developed for different kinds of CAN-based systems, there is still demand for a simpler HLP in order to simplify the control tasks of small distributed systems which utilise tiny 8-bit microcontrollers such as

the Motorola MC68HC11 or Intel 8051. The two main requirements of a HLP for small systems are as follows:

- To achieve a design methodology which is easy to understand, and
- To fit into a limited amount of on-chip memory available to small microcontrollers.

It is important that the requirements above are taken into consideration because in many small systems it may not be necessary to use a complex HLP, such as SDS or DeviceNet. Moreover, in some cases, it is not financially viable to increase the cost of circuit design by introducing external memory.

After comparing the main characteristics of SDS, DeviceNet and CAN Kingdom, the author of this thesis concluded that the CAN Kingdom protocol contains more advantages than the other two, with respect to the requirements of simplicity and ease of design for small systems. Additionally, the CAN Kingdom protocol allows devices that utilise other HLPs to be integrated into a CAN Kingdom system, with only minor changes to the control software (see Chapter 3).

Despite the above, full implementation of the CAN Kingdom protocol is a complex matter; and hence, it is the aim of this thesis to show that further simplifications can be made to the protocol in order to suit the requirements of small CAN-based distributed systems.

The HLP designed in this thesis, based on the CAN Kingdom protocol, is named the Small CAN Kingdom protocol. The programming codes written for this small protocol easily fit into 512 bytes EEPROM of MC68HC11 microcontrollers used in this project. The protocol also provides an open solution which enables later designers to enhance the application progress.

The main idea behind the Small CAN Kingdom protocol is that a master node in the system, the King, is responsible for the whole network configuration and governs

the communications between devices. The King, however, can be removed after setting up the network; and the system, therefore, can inherit the full potential of the CAN protocol such as multimaster, broadcasting or multicasting which are not efficiently utilised in other protocols (e.g. SDS or DeviceNet).

The CAN nodes in a Small CAN Kingdom system are called Cities. The Cities can be designed independently from each other without any concern for their inter-communication role in a particular network; responsibility for this lies with the King. The major consideration, in designing a City, is that it must be able to obey the King's instructions.

In order to develop the software and to demonstrate the performance of the Small CAN Kingdom protocol, a small distributed system is also designed in this project. The aim of this system is to illustrate the responsibility of the King, the communication between the King and Cities, and the communication between the Cities themselves.

1.2 Structure of the thesis

The remaining chapters of this thesis are concerned with details of the research which are summarised as follows:

Chapter 2 presents an overview of network topologies and industrial networks. The main features of the CAN protocol are also discussed in this chapter in order to show that the protocol is well-suited for industrial environments.

Chapter 3 introduces the importance of HLPs for CAN-based systems. The architecture of three popular HLPs are covered:

1. Smart Distributed System (SDS) from Honeywell,
2. DeviceNet from Allan Bradley, and
3. CAN Kingdom from Kvaser.

The advantages and disadvantages of these three HLPs are also discussed in order to select the HLP employed for the task of designing a simpler protocol in this thesis.

Chapter 4 details the design of the Small CAN Kingdom protocol, utilising the main ideas of CAN Kingdom, following the comparison of the three HLPs covered in Chapter 3. The methodology of designing a small distributed system is also introduced in this chapter. The small system consists of:

- A master node (the King) which is responsible for the network configurations, and
- Three Cities, each of which carries out particular tasks for its specific role in the system.

Chapter 5 reviews the two physical components used in the design of the two main parts of a CAN node: a microcontroller which controls the node's operations, and a CAN controller which manages the node's communication. The hardware and software development environments are also covered in this chapter in order to provide an efficient mechanism for the design of the Small CAN Kingdom system.

Chapter 6 presents the steps associated with the design of the hardware part of the system. All the interfaces between the components used in each CAN node are described. In addition, the chapter includes the steps of modifying the DS3695 (RS485 standard) transceiver chips in order to suit the requirements of the CAN bus. The DS3695 transceiver chips were used in this thesis due to time restrictions and the difficulty of obtaining standard CAN transceiver chips in Perth, Western Australia.

Chapter 7 is concerned with the design of the software which controls the small system in this project. The implementation of the software has taken into account the rules associated with the Small CAN Kingdom protocol. The designs of the software for the King, as well as the three Cities, are described in detail including

algorithms for each software module. Finally, efficient testing schemes, which were used to check the performance of the system, are covered in this chapter.

Chapter 8 is reserved for concluding remarks which were made following the completion of the research program. Some suggestions are then proposed concerning future developments and research.

Appendix A provides the description of the system development environments.

Appendix B contains the pin connection tables for each hardware interface design, and the diagrams showing the pin layout of these components. These tables and diagrams were used for wire-wrapping purpose in the hardware design.

Appendix C describes the steps associated with how system designers can use the King to set up the network.

Finally, **Appendix D** gives the complete program listing of the software designed in this project.

CHAPTER 2

NETWORK TOPOLOGIES AND THE CONTROLLER AREA NETWORK (CAN) PROTOCOL

This chapter provides an overview of network topologies and industrial networks, and their protocols, including the Controller Area Network (CAN) protocol. The prominent features of the CAN protocol are described in order to explain why it is suitable for hazardous industrial environments.

2.1 Network Topologies

2.1.1 Introduction to Computer Network

Hughes (1992, p. 3) states that "communications, whether among humans, animals, or computers, involve the transfer of information". With computers becoming widespread in society, the needs of communications have increased. In the home, for example, a data file is transferred from one personal computer to another or information can be accessed from a public database. In the office and educational institutions, communication channels are used to exchange e-mails or to share expensive peripherals such as laser printers or plotters. In the process industry, the transfer of information is necessary to coordinate the control of instrumentation associated with a plant. In the manufacturing industry, data is transmitted from one automated unit to another (Halsall, 1996, p. 3).

This exchange of information is called networking. Depending on the geographical distance of the communication, there are two types of network: Local Area Networks (LANs) and Wide Area Networks (WANs). The term LAN is used when computers are distributed around a single office or building. If the computers are located in different sites, then the term WANs is applicable (Halsall, 1996, p. 6).

In response to the growth in computer networks, the International Standards Organisation (ISO) has developed a reference model (Figure 2-1) for computer networking known as Open Systems Interconnect (OSI). According to this model, the entire communication subsystems are broken down into a number of layers, each of which performs a well-defined function. Henshall & Shall (1988) describe these layers as follows:

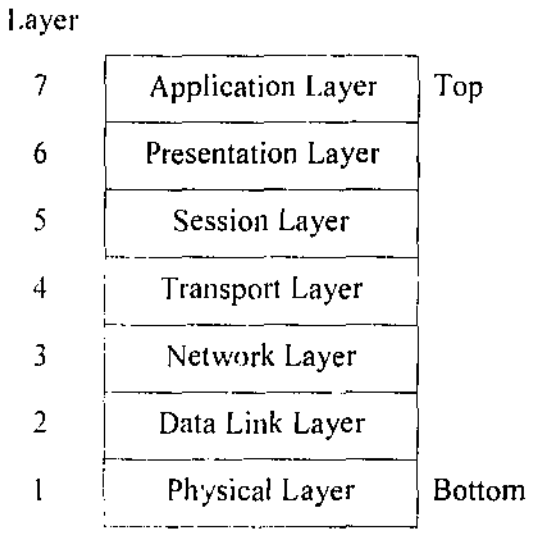


Figure 2-1 ISO/OSI Reference Model

The **Application Layer** contains a variety of protocols that are commonly needed. It provides the means for incompatible computers to communicate with each other.

The **Presentation Layer** is concerned with the syntax and semantics of the information transmitted. This may include character code translation, data conversion, or data compression and expansion.

The **Session Layer** focuses on providing the services used to organise and synchronise the dialogue that occurs between users, and to manage the data exchange.

The **Transport Layer** contains functions, which accept data from the session layer and split them up into smaller units, if required. It also determines what type of service to provide the session layer and users of the network.

The **Network Layer** is concerned with the task of controlling the operation of the subnet. It determines how packets are routed from source to destination.

The **Data Link Layer** provides reliable data transmission from one node to another. It is responsible for the error-free transmission of data frames.

The **Physical Layer** is responsible for transmitting raw bits over a communication channel.

2.1.2 LANs and Industrial Networks

According to Nunemacher (1990, p. 17), the generally accepted definition of a LAN is that two or more microcomputers are connected and communicate with one other through some physical media, such as twisted-pair or coaxial cable, in order to share data and peripheral devices. These microcomputers are usually located in the same limited geographic area.

In addition, Zuberi and Shin (1996) also state that a LAN protocol should fit the requirements of industrial automation. This is due to the fact that devices which exchange data are usually located in the same plant or the same factory.

2.1.2.1 Topologies of LANs

The topology of a LAN describes how the LAN is constructed. Nunemacher (1990, pp. 26-30) claims that there are three basic topologies:

1. Star
2. Ring
3. Linear Bus

Firstly, the **Star Network** has a central hub to which all the workstations, or nodes, and file server are attached via cable (Figure 2-2). The hub is the "heart" of the star, and all network traffic must pass through the hub. The advantage of this type of network is that it is easy to maintain and modify, since the only area of concentration is at the hub. However, because all nodes must be connected to the hub, large amounts of cable are required, and the potential for network failure increases. An example of the star topology is the Janet network (Crowcoft et al., 1993).

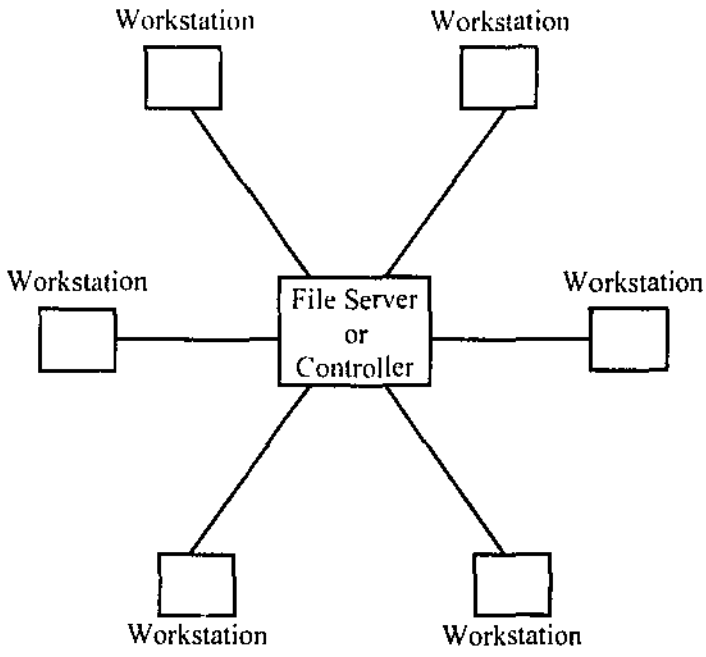


Figure 2-2 Star Network

Secondly, the **Ring Network** connects workstations on a single transmission, which forms a ring (Figure 2-3). Data travels around the ring in one direction and passes through each node. It is obvious that less cable is needed for this topology than the Star network. However, the entire network will fail if one node fails, and thus, it can be difficult to diagnose the fault. In order to maintain the system integrity, a bypass mechanism can be used to detect a faulty workstation. This topology is employed by the Token Ring network (Nunemacher, 1990, p. 103).

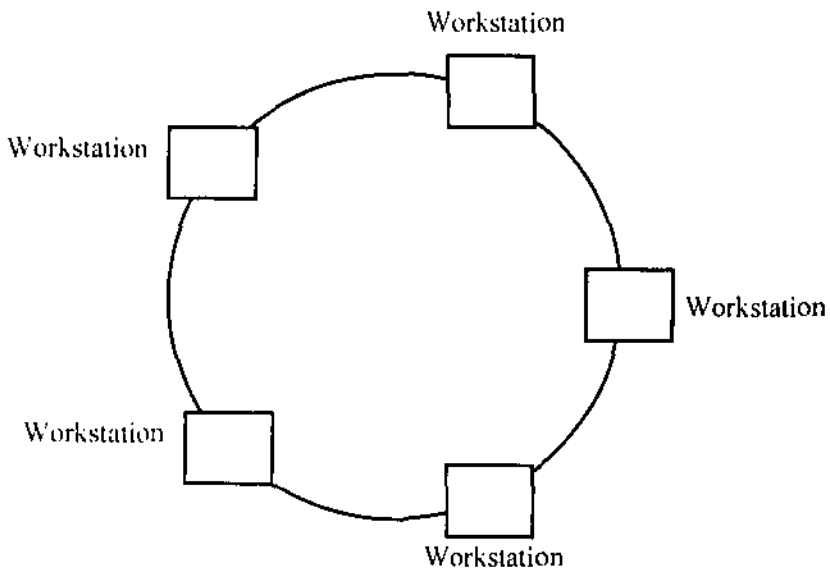


Figure 2-3 **Ring Network**

Finally, the **Linear Bus Network** consists of a number of nodes which are attached to a common cable or bus (Figure 2-4). The data travels on the bus in both directions and does not have to go through each node. The advantages of this topology are the short cable length and the simple wiring layout. Moreover, if one node goes down, it does not affect the whole network. This type of network is also easy to extend and to add nodes when required. However, the disadvantages of the bus topology are that the bus can be a bottleneck to the network when network traffic is very heavy, and that the fault diagnosis and isolation are difficult to maintain because all nodes in the bus can be the concentrators or hubs at one time.

A well-known example of network architecture which uses the Linear Bus Topology is Ethernet (Nunermacher, 1990, p. 83).

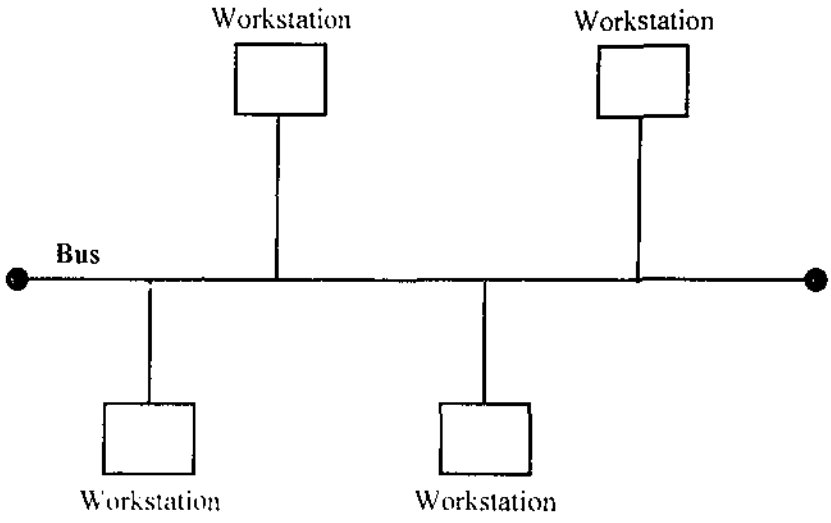


Figure 2-4 **Linear Bus Topology**

2.1.2.2 LANs in Industry

As mentioned previously, Local Area Network (LAN) architecture is suitable for industrial networks. Cena and Valenzano (1995), in discussing Ethernet, state that:

> The most popular and diffuse network in the office automation environment is without doubt Ethernet, based on the CSMA/CD approach. The reason for this is mainly due to the fact that the mechanism adopted to manage access to the shared transmissive medium is very simple, and allows components and communication boards to be used which are cheaper than those employed in other kinds of networks, such as for Token Ring.

CSMA/CD is an acronym for Carrier Sense Multiple Access with Collision Detection. The basics of CSMA/CD are that every node, which has messages to

transmit, listens to the traffic on the bus, and the node can access the bus when it is idle (free). If two or more nodes try to transmit messages at the same time, then a collision occurs. All nodes have to withdraw from the bus and wait for a random period of time before trying to access the bus again (Halsall, 1996, p. 280).

In addition, with respect to the requirements of industrial networks, Cena and Valenzano (1995) indicate that "though CSMA/CD is well-suited for office automation, it is not considered to be satisfactory for the automated factory environment". This is due to the fact that the industrial automation environment needs deterministic transfer time and synchronisation activities, while a random mechanism is used in the CSMA/CD to decide which station has the right to access the shared network. Moreover, as the network load increases, so does the number of collisions. This means the network throughput can be significantly reduced; hence, the required transfer time is not satisfied (Cena, Demartini, & Durante, 1996).

Consequently, this problem has stimulated technological inquiry and architecture has been proposed for industrial LANs such as Controller Area Network (CAN), SP-50 FieldBus, MAP, Profibus, FIP, and so on (Zuberi & Shin, 1995).

Of these network protocols, CAN has gained widespread acceptance in the industry because of its speed, low cost, real-time support, reliability in noisy environments, and priority-base arbitration (Zuberi & Shin, 1995).

The main features of the CAN protocol are described in the following section.

2.2 Controller Area Network (CAN) Protocol

2.2.1 Introduction

Controller Area Network (CAN) protocol (CAN Specification Version 2.0, 1991) is an advanced serial protocol, which was developed by Robert Bosch GmbH in the early 1980s. It was primarily used in the automotive industry, which is known to be both physically harsh and electronically noisy (Croft, 1996). Due to its versatility, CAN has recently been discovered to be suited to a broader class of applications in various automated factory environments (Cena &Valenzano, 1995) such as in pilot plant (Gollmer & Posten, 1994), in electrical wheelchairs (Van Woerden et al., 1994), in controlling mobile robots (Wargui, et al., 1996), and in manufacturing tin cans (Kirk, 1996).

The CAN protocol is based on the CSMA/CD access method, but it takes a much more systematic approach, which is known as Bus Arbitration mechanism (see section 2.2.3.1), to solve bus contention. This new method utilised by the CAN protocol guarantees that when a collision occurs, only the station transmitting the message with the highest priority is able to access the bus; and hence, the time critical requirement in industrial environments is satisfied (Cena &Valenzano, 1995).

It is noted that unlike many serial communication protocols, a CAN message contains no information related to the destination or the source addresses. Instead, messages are broadcast to all nodes in a CAN-based system; any number of nodes, therefore, can receive data simultaneously (Multicast Reception) (Ekiz, et al., 1996). However, each CAN message has a network-wide unique Identifier, which serves as the name of the message and the means to indicate its priority. This unique Identifier enables a CAN node to read only the messages which interest it (Cena &Valenzano, 1995).

Moreover, a CAN system employs Linear Bus topology; hence, any node can have access to the bus (Multimaster). This also makes the system easy to expand and to convert into different configurations (Croft, 1996).

A further advantage of the CAN protocol is its speed, according to Ekiz, et al. (1996), its baud rate can be up to a maximum of 1 Mbit/sec at 50m bus length.

The CAN Specification Version 2.0 (1991) provides a complete description of the CAN protocol. The specification consists of two parts: Part A and Part B. The main difference between the two parts is that Part A describes the original CAN protocol with 11-bit identifier messages ("Standard Format"); while Part B provides a larger address range for message identification of 29-bit ("Extended Format"), and also includes the Standard Format with some modifications. Therefore, the protocol specified in part B enables both types of messages to coexist within the same network.

2.2.2 Layer Architecture of CAN

To achieve design transparency and implementation flexibility, the ISO/OSI reference model and its layers' architecture are adopted for the specification of the CAN protocol.

In the CAN Version 2.0 Part A, the original CAN protocol is divided into three layers (Figure 2-5):

- The CAN Object layer
- The CAN Transfer layer
- The Physical layer

Application Layer
:
Object Layer - Message Filtering - Message and Status Handling
Transfer Layer - Fault Confinement - Error Detection - Error Signalling - Message Validation - Acknowledgment - Arbitration - Message Framing - Transfer Rate and Timing
Physical Layer - Signal Level and Bit Representation - Transmission Medium

Figure 2-5 **CAN's layers part A**

In part B, the layers of the CAN protocol are divided into two layers as shown in Figure 2-6:

- The Data Link layer consists of two sublayers:
  - The Logical Link Control (LLC) sublayer
  - The Medium Access Control (MAC) sublayer.

- The Physical layer

Application Layer
:
Data Link Layer
Logical Link Control (LLC)
- Acceptance Filtering
- Overload Notification
- Recover Management
Medium Access Control (MAC)
- Data Encapsulation/Decapsulation
- Frame Coding (Stuffing/Destuffing)
- Medium Access Management
- Error Detection
- Error Signalling
- Acknowledgment
- Serialisation/Desirialisation
Physical Layer
- Bit Encoding/Decoding
- Bit Timing
- Synchronisation
- Driver/Receiver Characteristic

Figure 2-6 **CAN's layers part B**

According to Figure 2-5 and Figure 2-6, the CAN Object layer and the CAN Transfer layer in Part A are included in the Data Link layer (ISO/OSI Layer 2) of Part B. Hence, the CAN protocol only contains the Data Link layer and Physical layer of the ISO/OSI model. However, the CAN specification provides a framework for data transmission which is the function of the Data Link layer. Consequently, upper and lower layers must be added to constitute an actual operational network (Bladin, et al., 1997).

2.2.3 CAN Basic Concept

The most important part of a CAN message is the Identifier field (see section 2.2.4), which is the outstanding feature of the CAN protocol. Tindell, Hansson and Wellings (1994) state that the Identifier serves two purposes:

1. Assigning a priority to the message, and
2. Enabling receivers to filter messages.

These tasks are done by the Bus Arbitration and the Message Filtering mechanisms specified in the CAN protocol, respectively.

2.2.3.1 Bus Arbitration

An essential feature of the CAN protocol is Bus Arbitration, which is a systematic approach of the CAN protocol with respect to message priorities and bus contention. Like any of the CSMA/CD protocols, a station in a CAN network starts to transmit its messages when the bus is idle (free). However, instead of withdrawing and waiting for a period of time when a collision occurs, the Bus Arbitration mechanism decides which station has the right to access the bus (Baba, Ekiz, Kutlu, & Powner, 1996).

This determination of station access is achieved through bit transmission. In CAN terminology, there are two bit levels: a "recessive" bit (usually, logic level '1') and

a "dominant" bit (usually, logic level '0'). If more than one station transmits data concurrently and one station transmits a dominant bit, then the bus maintains the dominant state regardless of the recessive bits (if any) transmitted from the other stations. The bus is only at the recessive state when all nodes in the system transmit recessive bits. In effect, the CAN bus acts like a large AND-gate, with each station being able to see the output of the gate (Tindell, et al., 1994).

Furthermore, when transmitting data, the Identifier is the first part of the message being transmitted onto the bus from the most-significant bit to the least-significant bit. According to the Bus Arbitration mechanism, if the station transmits a recessive bit (bit '1') and monitors the bus with dominant state (bit '0'), it stops transmitting since it knows that its transmitting message is not the highest priority on the bus. The station winning arbitration takes control of the bus, and the station losing arbitration becomes a receiver. This method enables the highest priority message to always be transmitted even though the bus load is heavy.

An example of the Bus Arbitration mechanism is given in Figure 2-7.

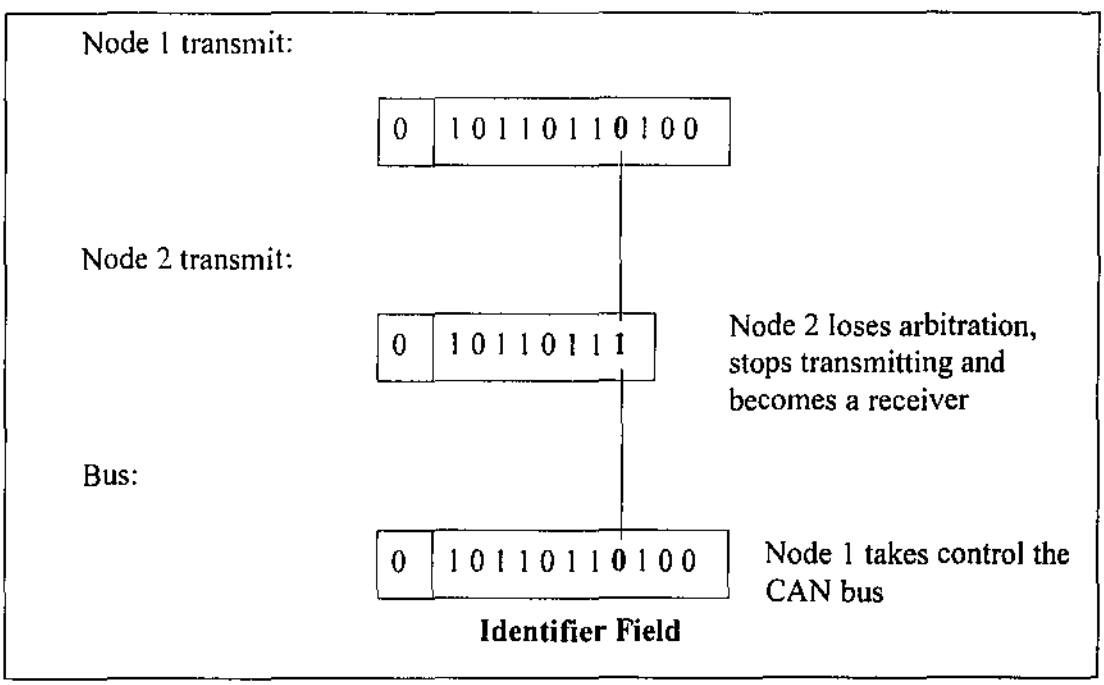


Figure 2-7 Example of Priority in CAN bus

2.2.3.2 Message Filtering

As mentioned earlier, CAN is a multicast protocol; thus, all stations can receive data simultaneously. Nevertheless, a station may be configured to accept particular messages through the Message Filtering mechanism. Typically, this is done with the aid of mask registers (Tindell, et al., 1994). Every bit of the mask registers must be programmable, which means they can be either enabled or disabled for message filtering. The length of a mask register can comprise the whole Identifier or just part of it.

It is important to note that any bit in the register can be set to "don't care", i.e. the CAN controller will not compare the message Identifier in the respective bit position. Consequently, a mask register can be used to select a particular message or a group of messages from the bus (CAN Specification Version 2.0, 1991, p. 56).

2.2.3.3 Error Handling

With respect to data consistency, Baba et al. (1996) state that "the CAN protocol implements powerful error detection mechanisms focusing on Cyclic Redundancy Check (CRC), bit stuffing, and both positive and negative acknowledgment". According to the CAN Specification Version 2.0 (1991, p. 59), these mechanisms include the following five error detection types:

1. A **Bit Error** is detected when a node transmits a dominant bit but monitors a recessive bit on the bus or vice versa.

2. A **Stuff Error** occurs when a CAN message contains six consecutive bits with the same bit level. This violates the Bit Stuffing Rule, which allows only five consecutive bits with the same polarity (CAN Specification Version 2.0, 1991, p. 58).

3. A **Cyclic Redundancy Check (CRC) Error** occurs if there is a mismatch value between the CRC field of a CAN message and the actual value calculated by the receiving node.

4. A **Form Error** is detected when a fixed form field of a message contains one or more illegal bits.

5. An **Acknowledgment Error** occurs if none of the nodes has received a transmitted message correctly.

With this five types of error detection scheme, the CAN protocol can detect almost every error in the system. The significant feature of this error detection scheme is that all stations in the network will be informed when an error occurs. This enables the transmitter to retransmit the message which has been corrupted. If repetitive errors are detected, the faulty station will remove itself from the bus (Baba, et al., 1996).

2.2.4 Data Transmission

In a CAN network, message transfer is manifested and controlled by four different frame types:

1. A **Data** frame carries data between nodes.

2. A **Remote** frame is used to request a certain message to be sent to the bus.

3. An **Error** frame informs all stations in the network that an error caused by the last message has been detected.

4. An **Overload** frame is sent by a station when it requires a delay to process data.

In addition, a CAN message can be in Standard or Extended formats, with both being covered by part B of the CAN specification. The following sections describe the format of the four frame types covered in Part B, and indicate the differences between the two parts.

2.2.4.1 Data frame and Remote frame

The main frame type in the CAN protocol is the Data frame, which is used to transmit data between stations in a CAN system. If a station wants to receive a particular message however, it can request the data by sending a Remote frame with the same Identifier as the respective Data frame.

The formats of Data frames (Figure 2-8) and Remote frames (Figure 2-9) are almost identical except for two significant differences:

1. There is no Data field in a Remote frame as it is used to request data.

2. The Remote Transmission Request (RTR) bit is recessive in Remote frames, while it is dominant in Data frames.

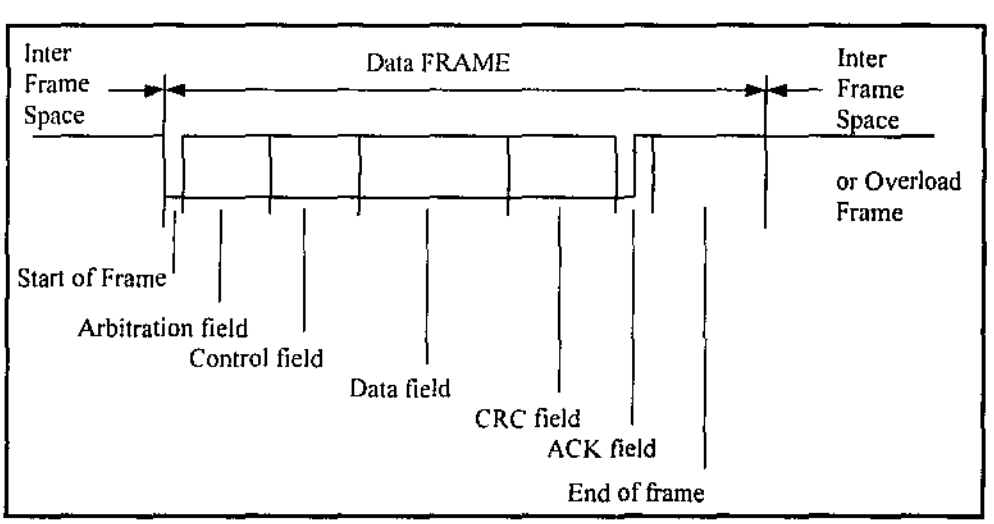


Figure 2-8 Data Frame

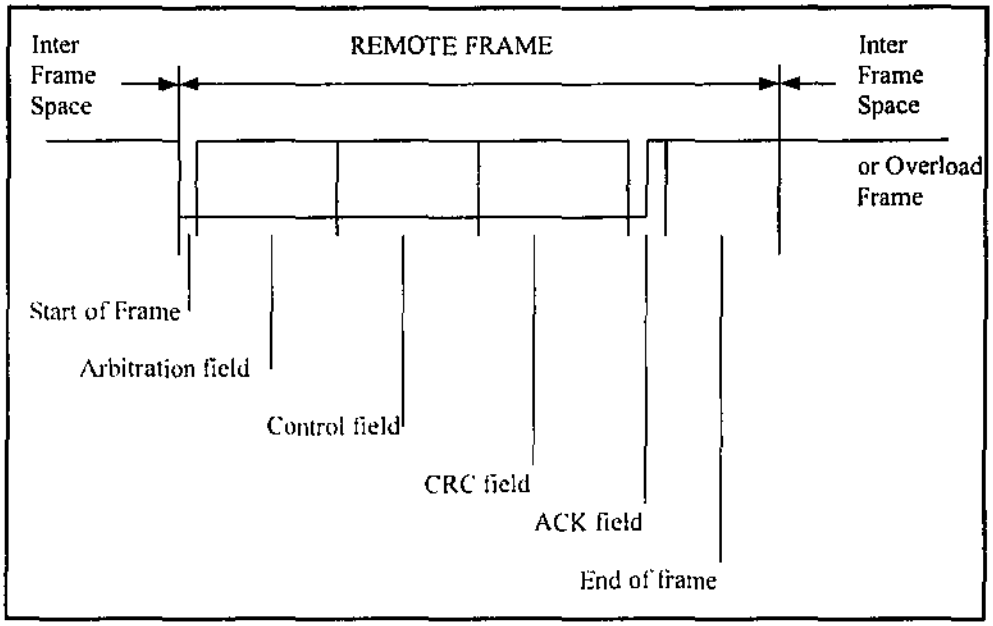


Figure 2-9 **Remote Frame**

2.2.4.1.1 Start of Frame (SOF)

Start of Frame is a single dominant bit, which marks the beginning of Data frames or Remote frames.

It is of interest to note that the Error frame and Overload frame do not have this bit because they are used to indicate special conditions of the CAN nodes. The descriptions of these two frames are covered in section 2.2.4.2 and 2.2.4.3, respectively.

2.2.4.1.2 Arbitration field and Control field

The **Arbitration** field, the "heart" of the CAN protocol, contains the message Identifier, which acts as the name of the message and the priority of the message (Baba, et al., 1996). The prominent difference between Standard and Extended formats is that a Standard message contains an 11-bit Identifier, while the Identifier of an Extended message has 29 bits.

It should be noted that according to CAN Specification Version 2.0 (1991, p. 44), the seven most significant bits of this field must not all be recessive.

The **Control** field of a CAN message informs the receiving stations of the number of data bytes (0-8 bytes) contained in the message. This is indicated by the 4-bit Data Length Code (DLC).

The formats of these two fields are slightly different between Standard and Extended messages as shown in Figure 2-10 and Figure 2-11.

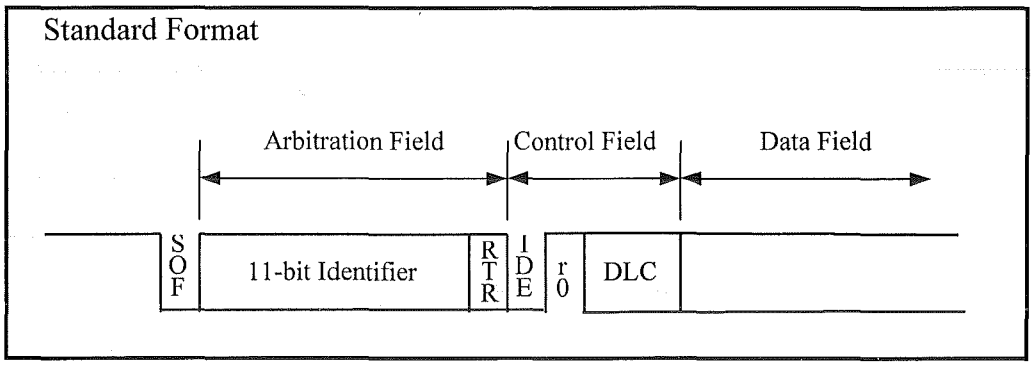


Figure 2-10 Arbitration Field - Standard

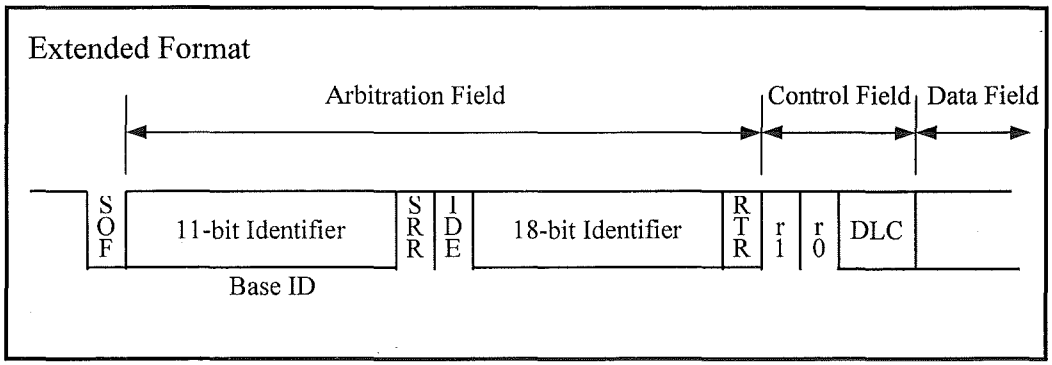


Figure 2-11 Arbitration Field – Extended Format

As seen in Figure 2-10 and 2-11, it is noted that:

Firstly, the **Remote Transmission Request** (RTR) bit is located at the end of the Arbitration field. This bit indicates whether the frame is Data frame or Remote frame. It is dominant in Data frame and recessive in Remote frame. With the Arbitration mechanism of the CAN protocol, a dominant bit will overwrite a recessive bit if the two bit levels are sent concurrently to the bus. Therefore, in the unlikely situation when a Data frame and a Remote frame, with the same Identifier, are transmitted at the same time, the Data frame wins arbitration due to the dominant bit following the Identifier. In this case, the node that has transmitted the Remote frame receives the desired data immediately.

Secondly, the **Substitute Remote Request** (SRR) bit is a recessive bit, which is transmitted in Extended frame. It is located in the same position of the RTR bit in Standard frame, and so substitutes the RTR bit in the Standard frame. When the Base ID of the Extended frame is the same as the Identifier of the Standard frame, and in the event of a collision between these two frames, the Standard frame has a higher priority according to the Bus Arbitration mechanism.

Finally, the **Identifier Extension** (IDE) bit indicates whether the message is Standard or Extended. This bit is located in the same position in either Standard or Extended frame. It is dominant in Standard frame and recessive in Extended frame. This construction of the IDE bit ensures that even in a collision between a Standard Remote frame and an Extended frame (Data or Remote), the Standard frame always wins arbitration.

Additionally, the r0 and r1 bits are reserved bits for future use. These bits are always recessive.

Note that the formats of Data and Remote frames in Part A of the CAN specification are the same as the Standard formats of those in Part B except the IDE bit was the reserved bit (bit r1) in Part A (CAN Specification Version 2.0, 1991, p. 12).

2.2.4.1.3 Data field

The **Data** field is transmitted within Data frames. It contains information that is exchanged between nodes in the CAN system.

The length of the Data field can be 0 to 8 bytes indicated by the Data Length Code (DLC). Each data byte contains 8 bits with its Most Significant Bit (MSB) being transmitted first (CAN Specification Version 2.0, 1991, p. 47).

2.2.4.1.4 Cyclic Redundancy Check field (CRC)

The **CRC** field contains the CRC sequence which consists of 15 bits and a recessive CRC Delimiter bit (Figure 2-12). The transmitter calculates special check bits for the bit sequence from the start of a frame until the end of the Data field. This CRC sequence is transmitted in the CRC field. The receivers, after receiving a frame, calculate the CRC sequence using the same formula and perform a comparison with the received sequence. If a mismatch is detected, a CRC error has occurred, and hence, an Error frame is generated. Consequently, the original message is repeated (CAN Specification Version 2.0, 1991, p. 47).

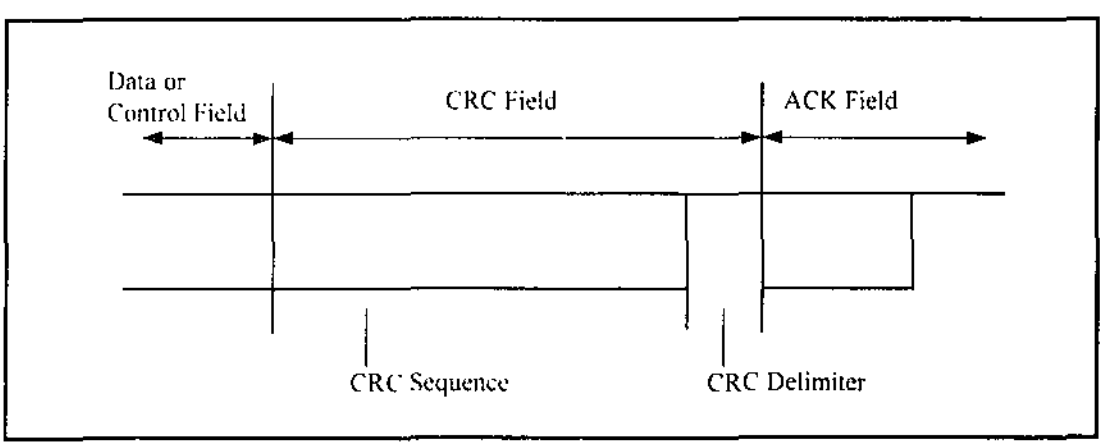


Figure 2-12 Cyclic Redundancy Check Field

2.2.4.1.5 Acknowledge field (ACK)

The **Acknowledge** field, containing an ACK Slot bit and a recessive ACK Delimiter bit (Figure 2-13), is used to indicate if a message has been received correctly. To achieve this, during the ACK Slot bit interval, the transmitter sends out a recessive bit. Then, any node that has received an error free frame acknowledges the correct reception of the frame by sending back a dominant bit. If during the ACK Slot bit interval the transmitter does not detect a dominant bit, this means none of other nodes have received the frame correctly, an ACK occurs and the original message has to be repeated (CAN Specification Version 2.0, 1991, p. 47).

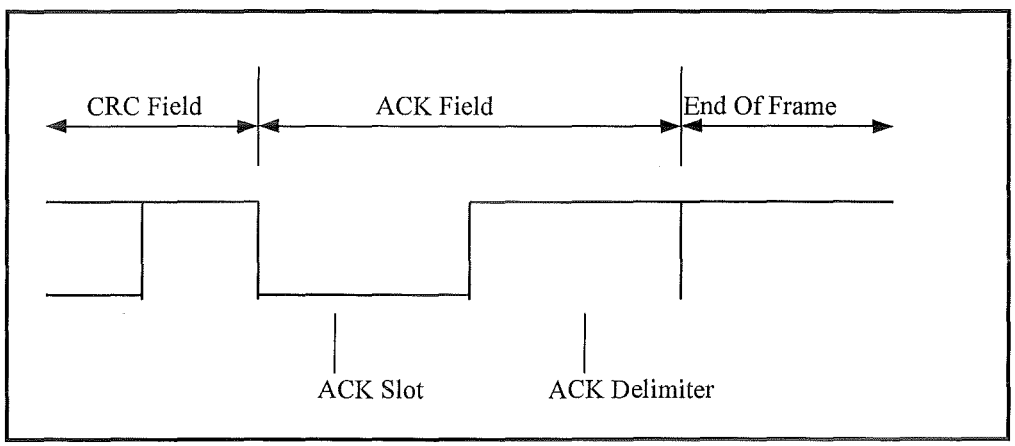


Figure 2-13 Acknowledge Field

2.2.4.1.6 End of Frame (EOF)

This field contains 7 recessive bits to indicate the end of a frame.

2.2.4.2 Error Frame

During transmitting its message, a station also monitors the bus. As a result, if an error occurs, the station will send an Error frame to inform the other nodes of this error. Moreover, if the repetitive errors are detected, it will withdraw itself from the bus according to the Fault Confinement Rules (CAN Specification Version 2.0, 1991, p. 61).

An Error frame consists of two fields: the Superposition of Error Flags that is contributed from different stations, and the Error Delimiter containing 8 recessive bits. The structure of an Error frame is shown in Figure 2-14.

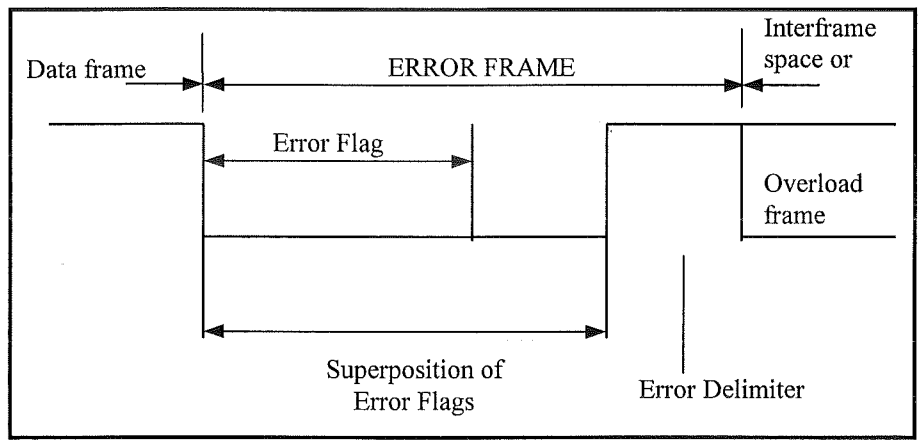


Figure 2-14 Error Frame

Depending on the Error flag being Active or Passive, then the Error frame is Active or Passive, respectively:

- An Active Error flag consists of six consecutive dominant bits.
- A Passive Error flag consists of six consecutive recessive bits unless it is overwritten by dominant bits from other nodes.

When detecting an error, a node will send an Active Error frame or Passive Error frame depending on its status as specified in the Fault Confinement Rules. Two further aspects regarding the error frame are highlighted as below.

Firstly, an Error flag construction violates the Bit Stuffing Rule of the CAN protocol, in which a sequence of a normal bit stream contains a maximum five consecutive bits with the same polarity. Therefore, a node will detect that there is an error if monitoring an abnormal bit stream on the bus, during the transmission or the reception of a message.

Secondly, the length of the Superposition of Error flags can be six bits up to a maximum of twelve; this depends on the number of bits sent by the different stations in detecting errors (CAN Specification Version 2.0, 1991, p. 51).

2.2.4.3 Overload frame

According to the CAN Specification Version 2.0 (1991, p. 51), there are three conditions which lead to the transmission of an overload frame from a node:

1. The internal conditions of a receiver which require a delay of the next data frame or remote frame.

2. Detection of a dominant bit at the first and the second bit of Intermission.

3. A node samples a dominant bit at the eighth bit (the last bit) of an Error Delimiter or Overload Delimiter.

Note that in the second condition, the term Intermission indicates the first three bits of the Interframe space, which is used to separate a Data or Remote frame from a preceding frame (CAN Specification Version 2.0, 1991, p. 53).

An Overload frame consists of two fields: 6 dominant bits of the Overload Flag and 8 recessive bits of the Overload Delimiter as shown in Figure 2-15.

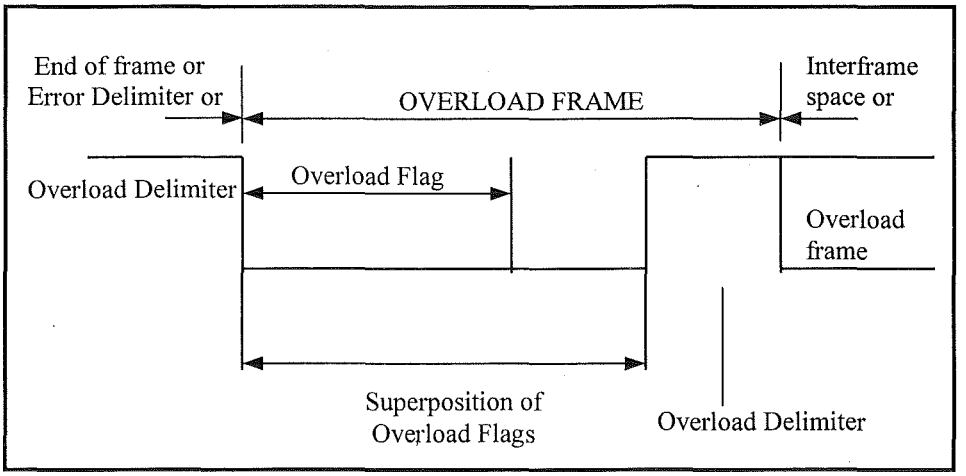


Figure 2-15 Overload Frame

As shown in Figure 2-15, an Overload frame has the same format as an Active Error frame. Therefore, in order to differentiate an Overload frame from an Error frame, the CAN protocol specifies that an Overload frame can only be generated during Intermission, while an Error frame is sent during the transmission of a message.

2.2.5 Implementation of CAN protocol

According to Croft (1996), the first working silicon CAN protocol controllers (CAN modules) became available in 1987. There are three recognised implementations of CAN protocol supported by silicon manufacturers which are the:

1. Basic CAN
2. Full CAN
3. CAN+

The significant differences between these CAN chips are in their functional implementations.

The Basic CAN chips only implement the basic functions of the protocol in hardware, for instance, the generation and the check of the bit stream. Other jobs are left for the host CPU such as Acceptance Filtering and whole message management. Therefore, these devices should only be used at low baud rates and low bus loads with a few different messages. Nevertheless, the advantage of Basic CAN is that the small size of the chips lead to its low cost.

The second implementation of the CAN protocol is the Full CAN controllers, which implement the whole CAN protocol in hardware including Acceptance Filtering and message management. Hence, these devices can alleviate the host CPU's load and can handle higher baud rates; consequently, higher bus loads can be performed. These chips, however, are more expensive than the Basic CAN devices.

The third version of silicon CAN controllers is CAN+, which combines the best features of the both previous device types.

CAN controllers also have the following characteristics:

Firstly, if the modules were specified prior to CAN Specification Version 2.0 Part B, they are only able to transmit and receive Standard CAN frames (11-bit identifier); and therefore, messages using 29-bit identifier will cause errors.

Secondly, with respect to CAN modules specified after the introduction of CAN Specification Version 2.0 Part B, there are two different types. Devices named "Part B Passive" can transmit and receive Standard frames, but tolerate Extended frames without generating errors. Devices named "Part B Active" can transmit and receive both Standard and Extended frames (Kvaser CAN Pages: The CAN protocol, n.d.).

Currently, a wide range of CAN controllers are available from different semiconductor companies such as 82527 from Intel (82527 serial communications controller architecture overview, 1996), SAB-81C90 and SAB-81C91 from Siemens (SAE 81C90/91 stand-alone Full-CAN controller data sheet, 1997). Moreover, microcontrollers with an integrated CAN module are also available such as C167CR from Siemens (C167CR 16-bit CMOS single-chip microcontroller data sheet, 1995) and 68HC12 from Motorola (Blandin, et al., 1997).

2.2.6 Advantages and Disadvantages of the CAN Protocol

2.2.6.1 Advantages

Since the introduction to the CAN protocol, it has been employed in a wide range of industrial networks. The main attractions are as follows:

- Cheap to implement
- Easy to install and uninstall a unit into or out of the system.

- A fast network: up to 1 Mbit per second.

- Reliable: CAN systems can work well in a hazardous industry environment.

- Well-suited for distributed control systems due to its prioritisation characteristic.

2.2.6.2 Disadvantages

The CAN protocol, however, does contain disadvantages relating to its functionality. These include:

- All nodes in the CAN system have to work at the same baud rate.

- Limitation of data byte in a message to maximum of 8 bytes.

- The multimaster concept may slow down the speed of the nodes in the system, especially for Basic CAN chips. This is because every node has to receive a frame, then uses the acceptance filtering method to decide whether to accept it or not.

- Every node may have to deal with error or overload conditions, caused by other nodes, while it is performing other jobs.

- In a heavy bus load condition, lower priority messages can be delayed indefinitely by higher priority messages due to the CAN Arbitration mechanism.

2.3 Conclusion

This chapter has provided an overview of computer networks and the ISO/OSI reference model for computer networking. The Local Area Networks (LANs) topologies, and the requirements of industrial networks were described. Some of the industrial network architecture available was mentioned, and the significant features of the Controller Area Network protocol were highlighted.

In summary, the CAN protocol provides a means to develop low cost, fast, and highly reliable networks for control systems. Notably, its real-time support characteristic has resulted in the protocol being accepted by a wide range of automation industries. The CAN's Arbitration concept allows different priorities for CAN messages. Thus, it is highly suitable for implementing complex distributed control systems in which nodes can be grouped together by message Identifiers; and hence, receive similar types of messages simultaneously and ignore messages for other nodes or group of nodes.

In addition, the ISO/OSI based model of the CAN protocol allows it to achieve design transparency and implementation flexibility. The error detecting methods used by CAN enable its system to detect almost every error on the bus. Faulty nodes withdraw themselves from the network operation without disturbing the other nodes.

The CAN protocol, however, only specifies how a small packet of data transfers on the network. It does not cover topics such as flow control, node addresses, establishing communication, and transportation of data greater than 8 bytes. These topics are all necessary to design actual network operations. Consequently, they are for designers to address in Higher Level Protocols (HLPs), which are covered in the following chapter.

CHAPTER 3

HIGHER LAYER PROTOCOLS (HLPS) FOR CONTROLLER AREA NETWORK (CAN)

This chapter describes the architecture of three popular CAN Higher Layer Protocols (HLPs) currently used in automation industries:

1. Smart Distributed System (SDS) from Honeywell,
2. DeviceNet from Allan Bradley, and
3. CAN Kingdom from Kvaser.

The advantages and disadvantages of these three protocols are discussed in order to choose a suitable HLP for the task of designing a simpler HLP in this project.

3.1 An Overview of Higher Layer Protocols for CAN

As discussed in Chapter 2, the CAN protocol is developed in accordance with the ISO/OSI model which contains seven layers. The implementation of a CAN system however, usually utilises three layers including:

- Application layer,
- Data Link layer, and
- Physical layer.

The main reason for this difference is noted by Cena, et al. (1996) who claim that "The network, transport, session and presentation layers have been omitted in order to get shorter response times for data transfer".

Figure 3-1 shows the ISO/OSI model of a CAN system in comparison with the ISO/OSI seven-layer model.

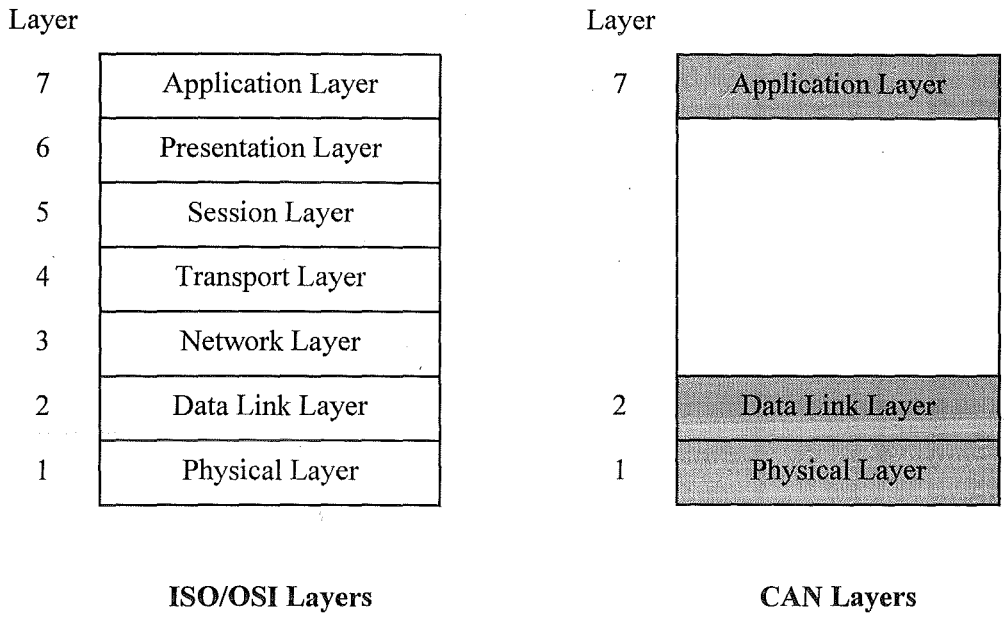


Figure 3-1 ISO/OSI Reference Model for CAN

However, while the CAN specification only deals with the Data Link Layer (see Chapter 2), the lower and the higher layers (Physical layer and Application layer) must be added to a CAN-based system in order to perform actual operations.

Although there are many publications relating to the use of the CAN protocol and the needs of HLPs for CAN systems, there is not much literature available solely for HLPs, except for the specifications themselves. This is because CAN is a new technology, and as Korane (1996) claims "one reason is simply that many designers are still learning how to use it".

The infancy of CAN and its associated technological requirements has seen various responses, by designers, to address perceived issues. However, there remains much

work to be done as many areas lack attention in sufficient detail. For example, all CAN-based systems need a HLP to ensure inter-operability between CAN components. Yet, in this regard, there are no worldwide standards for HLPs. As a result, many CAN user groups have developed HLPs that meet their own needs. Korane (1996) notes that up to 31 different CAN HLPs are currently in use; thus, designers may choose a proprietary protocol for their CAN system or, as is the case in this project, design their own according to one of the existing HLPs.

The three HLPs, Smart Distributed System, DeviceNet and CAN Kingdom, highlighted in this thesis, are all widely-used and popular among industry (Blandin, et al., 1997). Furthermore, they are all well-supported in terms of literature and physical materials available to enable designers to implement a system.

In order to understand the CAN's HLPs, it is desirable to have an overview of the OSI Application layer (Figure 3-2). Dickson and Lloyd (1992) state that "the OSI Application Layer standards define a range of system independent application services to support real 'users' or user programs". These services are built on the functions of the lower layers in order to support distributed systems.

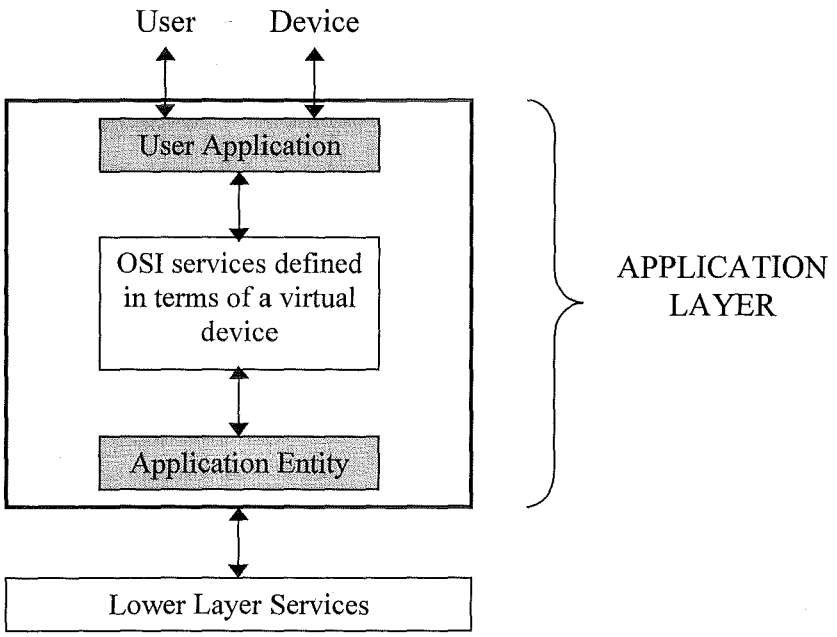


Figure 3-2 Model of the OSI Application Layer

As shown in Figure 3-2:

- The **User Application** is a local system independent part that interfaces to the user or devices.

- The **Application Entity** provides the standardised functionality of the Application layer to the user application.

Each of the three HLPs described in this thesis has a different design approach and provides different application layer services to its users. The main features of these protocols, according to Lennartsson and Fredriksson (1995), are:

- SDS, fundamentally, is point-to-point communication between a master (Host) and remote Input/ Output devices.

- DeviceNet is an open bus system where all modules have the same right to use the bus, and the use of the bus is only restricted by a few rules.

- CAN Kingdom specifies a set of protocol primitives which system designers can use to build a final protocol satisfying their system needs. The basic idea of CAN Kingdom is that a module, when connected to a system, has to wait for configuration instructions from the King (the Master node) before it can perform its work.

The following sections of this thesis describe and compare the aforementioned three HLPs in greater detail.

3.2 Smart Distributed System (SDS)

3.2.1 Introduction to SDS

Smart Distributed System, introduced by Honeywell's Micro Switch Division in 1994, includes a device-level control network based on the CAN protocol. The design of SDS meets the requirements of speed, reliability and flexibility for manufacturing automation applications and real-time control.

Basically, the SDS protocol specifies point-to-point communication between a host controller and remote Input/Output devices such as sensors, actuators, analog I/O devices, etc.

The development of the protocol has overcome some of the disadvantages of CAN such as data transfer no larger than 8 bytes, node addressing, and flow control.

3.2.2 SDS Basic Concept

An SDS system is developed based on the model as shown in Figure 3-3.

According to the Smart Distributed System Application Layer Protocol Specification Version 2.0 (1996), each node in the system contains three primary elements:

- Physical Component
- Logical Device
- Embedded Object

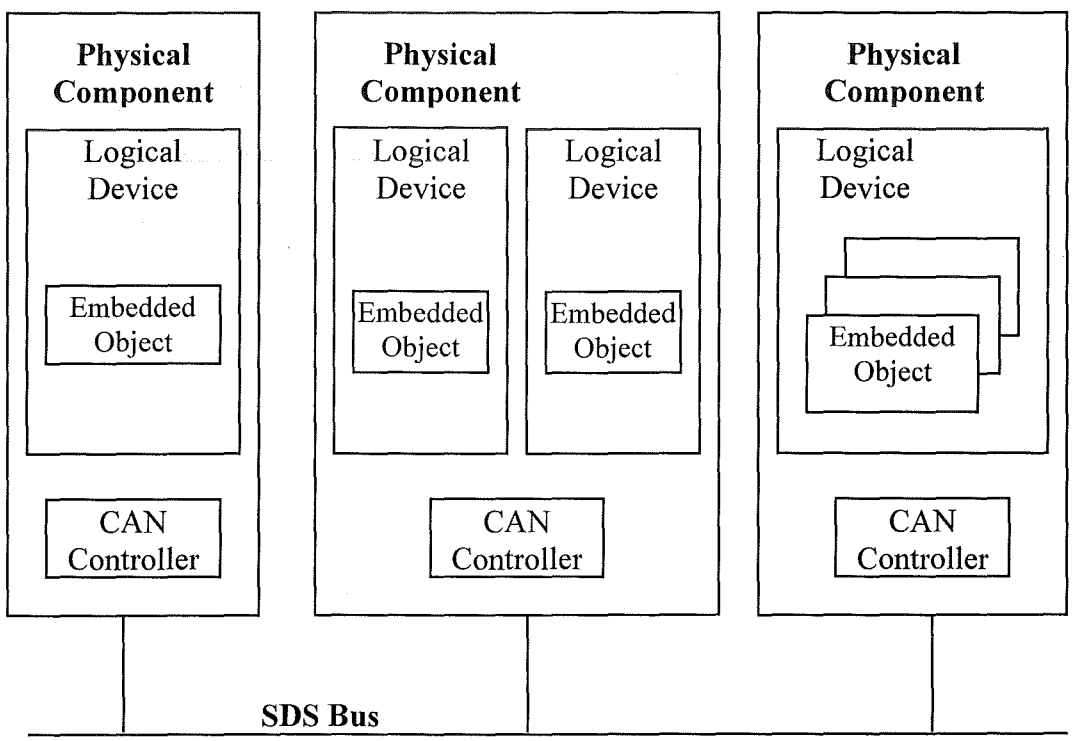


Figure 3-3 Smart Distributed System Model

The definitions of these elements are as follows:

- A **Physical Component** is a single physical package of hardware and software. It consists of 1 to a maximum of 126 Logical Devices. Typically, a Physical Component is an actual CAN node, which contains only one CAN controller circuit (chip).

- A **Logical Device** is an abstraction representing a separate entity within a Physical Component. It contains at least 1 and no more than 32 Embedded Objects. Each Logical Device has a unique address in an entire SDS system.

- The term **Embedded Object** is used to present an actual device such as switch, sensor, actuators, and so on. Each Embedded Object has a specific address within a Logical Device. The network address of an Embedded Object is the combination of its address, called Embedded Object Identification (EOID), and the Logical Device address, to which the Embedded Object belongs.

For example, to differentiate between the Embedded Object 2 of the Logical Device 3 and the Embedded Object 2 of the Logical Device 5, the combined addresses are shown below:

- Logical Address #3: EOID #2
- Logical Address #5: EOID #2

Additionally, in order to communicate with each other in the network, each SDS component (Physical, Logical or Embedded Object) has to be documented. The component's document contains attributes, actions, and events that are specific to the device (SDS Component Modelling Specification, 1995). These are defined as follows:

- **Attributes** provide information about the component such as vendor name, software version, data type and data structure.

- **Actions** contain the operations, which can be done by a device.

- **Events** are used to report the occurrence of an event for a device. For example, a state of a switch being ON or OFF.

Each attribute, action, and event has a unique Identification, respectively called Attribute ID, Action ID, and Event ID, in order to distinguish each document aspect from the other.

Note that these IDs are held by Honeywell Micro Switch. Accordingly, any device that uses the SDS protocol has to follow their standard.

Table 3-1 gives an example of an SDS component document.

Table 3-1 Example of an SDS Component Document

Attributes ID	Description	Primitive Tag				Default Value
		R/W	Type	Size	Cnt.h	
0	Network Data Descriptor	R	Uns	Byte	5	12h,00h,00h,00h,00h,00h
1	Baud Rate	R	Uns	Und	0	0
11	Serial Number	R	Uns	Long	0	N/A
12	Date Code	R	Char	Byte	3	N/A
55	Manufacturing Codes	R	Uns	Byte	0	N/A
61	Configuration Register	R/W	Uns	Byte	0	0

Action ID	Description	Parameter Type	Parameters	Data Type
0	No operation			
1	Change Address	Input	Addr, <Device ID>, <Partner ID, S-Num>	Uns 8, Uns 8, Uns 16, Uns 32
2	Self Test			
8	Enrol Logical Device	Output	S-Num, Partner ID	Uns 16,Uns 32
57	Password	Input	Password	Uns 8

Event ID	Description	Output Parameters	Output Data Type
0	Diagnostic Event Counter	Counter Value	Uns 8
3	End of Timer	Attribute ID, Data	Uns 8, Uns 16
Spec	COS_ON		
Spec	COS_OFF		

While an SDS system is in operation, the values of Attributes, Actions, and Events of a component can be read or changed. In doing so, the SDS Application Protocol provides several services which are described below:

- The **Read** service is used to read an attribute value of an Embedded Object (EO). For example, this service can be used to read the present value of a sensor.

- The **Write** service modifies or changes an attribute value of an EO. This may be used to set an actuator output to ON or OFF.

- The **Event** service reports the occurrence of an event in an EO. For instance, a Logical Device may report a self-test failure.

- The **Action** service is used to execute the operations specified for an EO. This may be used to initiate a self-test.

Moreover, if a device has only one Embedded Object such as a single binary input, simplified services are provided, by means of SDS short form messages (see section 3.2.3), to increase the system throughput as described below:

- The **Change Of State ON** (COS ON) service is used by a Logical Device to report a change of state to ON of its Embedded Object.

- The **Change Of State OFF** (COS OFF) service reports a Change of State to OFF of the object.

- The **Write ON State** service is used to write an ON state to a device.

- The **Write OFF State** service writes an OFF state to a device.

Furthermore, SDS Higher Layer Protocol also provides two special services to establish the connection and communication channels between devices as follows:

- The **Connection** service establishes a connection between two Logical Devices. It is used by one device to request a connection with another device. The two devices are then able to transfer data to each other following the successful connection establishment.

- The **Channel** service is used after a successful Connection service to provide communication channels for devices such as Multicast and bi-directional Peer-to-Peer channels (Smart Distributed System Application Layer Protocol Specification Version 2.0, 1996, p. 34).

Note that each SDS Application service contains parameters which include information related to the device addresses, Attribute ID, Action ID, Event ID, channel number, and so on. These parameters are responsible for the exchange of information between devices (Smart Distributed System Application Layer Protocol Specification Version 2.0, 1996).

3.2.3 SDS Application Protocol

This section describes how the SDS application protocol uses the CAN Data frames (see Chapter 2) to provide its services to user applications.

It is important to note that the CAN remote frames are not used in the SDS protocol. In addition, the protocol only utilises the CAN Standard format messages with 11-bit Identifiers specified in part A of the CAN Specification Version 2.0 (1991, p. 11).

In summary, a Standard CAN frame can be described as shown in Figure 3-4.

	Bit								
	7	6	5	4	3	2	1	0	
	ID-10	ID-9	ID-8	ID-7	ID-6	ID-5	ID-4	ID-3	CAN
	ID-2	ID-1	ID-0	RTR	DLC3	DLC2	DLC1	DLC0	Header
B	Data Byte 1								
Y	Data Byte 2								
T	Data Byte 3								CAN
E	Data Byte 4								Data
S	Data Byte 5								Field
	Data Byte 6								
	Data Byte 7								
	Data Byte 8								

Figure 3-4 Standard CAN frame format

According to the Smart Distributed System Smart Distributed System Application Layer Protocol Specification Version 2.0 (1996), the Identifier of the CAN header (Figure 3-4), which contains the Arbitration and Control fields, is divided into 3 subfields (Figure 3-5):

1. The Direction/Priority,
2. The Logical Address, and
3. The Service Type.

<table>
<tr><td colspan="8">Bit</td></tr>
<tr><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>Dir/Pri</td><td colspan="7">Logical Address {0...125}</td></tr>
<tr><td colspan="4">Service Type {0...7}</td><td>RTR</td><td colspan="3">Data Length Code</td></tr>
</table>

Figure 3-5 SDS header

The descriptions of these subfields are as follows:

- **The Direction/Priority** (Dir/Pri) subfield is presented by the most significant bit of the 11-bit CAN Identifier (ID-10). This bit determines the direction of the frame with respect to the content of the Logical Address subfield.

  - If Dir/Pri=1, the Logical Address subfield is the source address.
  - If Dir/Pri=0, the Logical Address subfield contains the destination address.

  Moreover, the Dir/Pri bit is used in Channel service to determine the priority of a message. The message with Dir/Pri=0 has higher priority than the message with Dir/Pri=1.

- The **Logical Address** subfield specifies the address of a Logical Device. The subfield starts from bit 3 through bit 9 of the CAN Identifier (ID-3 to ID-9). This allows a range of Logical Addresses from 0 to 125.

  Note: the address 126 and 127 cannot be used due to the restriction of the CAN specification, which means that the 7 most significant bits ID-10 to ID-4 must not all be recessive (CAN Specification Version 2.0, 1991, p. 11).

- The **Service Type** subfield is from bit 0 through bit 2 (ID-0 to ID-2). This indicates the type of the service specified for the message. The value of this subfield is 0 to 7 depending on the service carried by the message (see Table 3-2 and Table 3-3). The meaning of this field is different for Short Form and Long Form, which are two message formats used to transfer data in SDS protocol.

Table 3-2 Service Type value for Short Form messages

Service Value	Service Name
0	Change of State OFF
1	Change of State ON
2	Change of State OFF ACK
3	Change of State ON ACK
4	Write OFF State
5	Write ON State
6	Write OFF State ACK
7	Write ON State ACK

Note that the services with acronym ACK are used to inform the transmitters that the receivers have received the corresponding services successfully.

Table 3-3 Service Type value for Long Form messages

Service Value	Service Name
0	Channel
1	Connection
2	Not use
3	Not use
4	Write
5	Read
6	Action
7	Event

Also note that the SDS Application protocol does not make use of the Remote CAN frames. Therefore, the Remote Transmit Request (RTR) bit is always a dominant bit ('0').

As mentioned earlier, an SDS message can be one of the two formats: Short Form, and Long Form. The **Short Form** format is used for Change Of State ON (COS ON), Change Of State OFF (COS OFF), Write ON, and Write OFF services. The format of a Short Form frame is the same as shown in Figure 3-5. The value of the Data Length Code subfield in the Short Form format is always 0, which means there is no CAN data field in the message. Hence, this configuration increases the system throughput.

The **Long Form** format is used by other SDS services. The difference between Long Form and Short Form is that a message in the Long Form format contains at least two or more CAN data bytes. Long Form messages are used to access devices with more than one Embedded Object. Thus, some bits in the CAN data field are used to indicate the Embedded Object address (EOID).

Moreover, the Long Form format can be one of two forms: **Non-fragmented** and **Fragmented**. The Fragmented format can carry data greater than 8 bytes by dividing the data into fragments and transferring one fragment at a time.

Figure 3-6 and Figure 3-7 show the formats of the two types of Long Form frames.

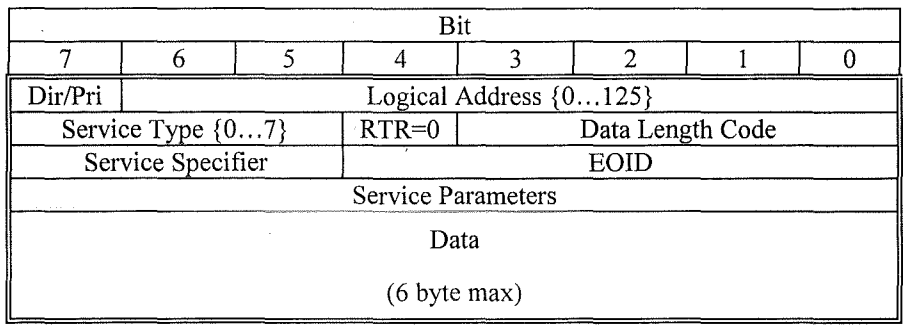


Figure 3-6 Non-fragmented Format

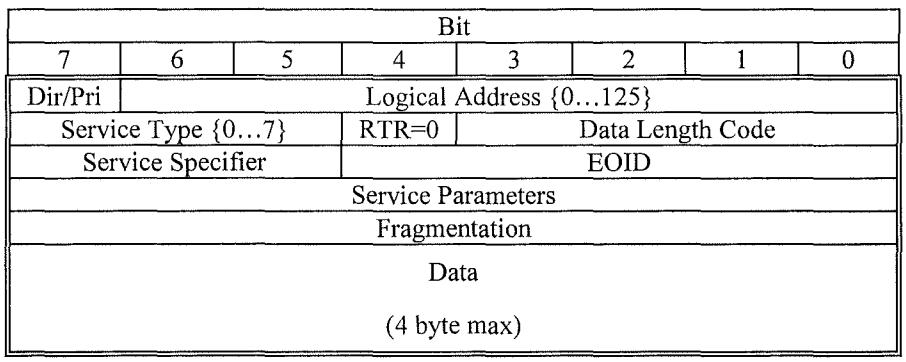


Figure 3-7 Fragmented Format

As shown in Figure 3-6 and Figure 3-7, some of the CAN data bytes are used to carry information related to the SDS messages (Smart Distributed System Application Layer Protocol Specification Version 2.0, 1996, p. 44). The following describes the information contained in each field of SDS Long Form messages:

- The **Service Specifier** field indicates whether the frame is fragmented or not. It is also used to request a specific service from a provider or to indicate if the requested service is successful.

- The **Embedded Object Identifier** (EOID) specifies the address of the Embedded Object. The value of this field can be 0 to 31.

- The **Service Parameter** field contains the parameters for the service carried by the message such as Attributed ID, Action ID, Event ID, etc.

- The **Fragmentation field** is used in Fragmented Long form format to indicate the fragment number and the total bytes of the fragmented message.

3.2.4 Advantages and Disadvantages of SDS

The SDS protocol, with its many advantages, provides a good solution for designing CAN-based systems. However, it also contains disadvantages that do not utilise the CAN protocol as efficiently as possible.

3.2.4.1 Advantages

- SDS overcomes the limitation of CAN protocol such as transferring data larger than 8 bytes, and node addressing.

- The design of SDS meets the requirements of flexibility, speed, and reliability of a real-time control system.

- SDS is a complete Higher Layer Protocol for a CAN network and is an ideal solution for controlling or monitoring I/O devices.

3.2.4.2 Disadvantages

- SDS only uses the CAN Standard format frame (11-bit Identifier)

- The formats of SDS frames cannot be changed. Hence, any CAN system that uses the SDS protocol has to follow exactly the SDS Standard.

- The priority of a CAN message depends on its Logical Device. In other words, the Logical Address decides the message priority.

- SDS does not use the Remote Frame specified in the CAN protocol.

- SDS uses a Master/Slave concept while the CAN protocol is based on the multi-master approach.

3.3 DeviceNet

3.3.1 Introduction to DeviceNet

Young (1995) writes that the purpose of the DeviceNet protocol is that it "simply defines the Application Layer that sits on top of CAN to specify how various devices communicate across the network". In doing so, it overcomes disadvantages of the CAN protocol such as node addressing and flow control.

DeviceNet is an open communication network mainly designed to connect factory devices such as push buttons, sensors, motor starters, and drives to control systems (Simonye, Alpena, & Witte, 1997). It provides a more versatile approach to module communication on the network where all nodes have the same right to access the bus.

In addition, DeviceNet uses the object-oriented method which provides efficient ways to model and design real-world objects (Hawryszkiewycz, 1994, p. 236). However, in DeviceNet, inheritance of the objects from one to another is not implemented (Moyne, Shah, McLaughlin, & Tang, 1997).

The references to the DeviceNet specification in this section are referred to by the Volume number and Section number of the specification. This is because the specification comes on a CD-ROM, and thus, the page number will be different for each updated version.

3.3.2 DeviceNet Basic Concept

According to the DeviceNet 2.0 Specification (Vol.1, Section 1-3), each node of a DeviceNet network is considered as a collection of **Objects**, each of which is an abstraction representing a particular component within a product. Similar kinds of Objects belongs to a **Class**. An **Object Instance** is the actual presentation of a certain Object within a Class. For example, an Instance of a Vehicle Class is a car or a truck. A cat is an Instance of an Animal Class.

Each Instance in a Class has the same set of **Attributes** specifying the characteristics of the Class, but the values of Attributes vary from one Object Instance to another.

An Object or a Class also provides a set of **Services** which are used to perform the tasks for the Object or the Class. It can be said that this feature of DeviceNet provides designers with greater flexibility than the Smart Distributed System (SDS). Instead of specifying fixed application services, DeviceNet allows its users to design different services that are more relevant to their Objects or Object Classes.

The DeviceNet hierarchical view of Classes and Objects is shown in Figure 3-8.

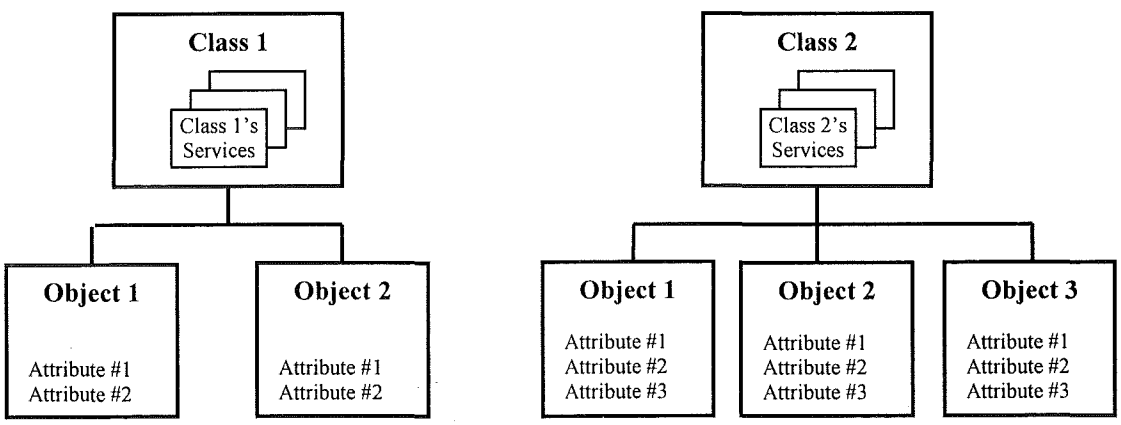


Figure 3-8 **DeviceNet hierarchical view of Classes and Objects**

DeviceNet also provides a set of *common services* to be used for devices that follow the DeviceNet standard. These services are fully described in Appendix G of the DeviceNet 2.0 Specification (1997, Vol. 1). For example:

- **Get_Attribute_All** service tells an Object or a Class to return all its Attributes.
- **Set_Attribute_All** service modifies the attribute contents of a Class or an Object.
- **Reset** service is used to reset a specified Class or Object.
- **Start** service places an Object into running mode.
- **Stop** service places an Object into stop or idle mode.

Furthermore, an Object or a Class has its distinct **Behavior** for particular events. In other words, the Behavior of an Object or a Class specifies how it responds to particular events.

Figure 3-9 illustrates the structure of a DeviceNet node.

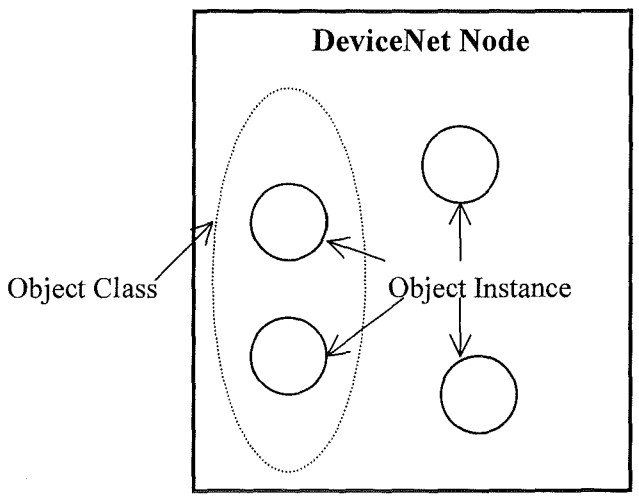


Figure 3-9 A DeviceNet node

A node in a DeviceNet network is identified by a unique Media Access Control Identifier (MAC ID). Similarly, a Class has a Class ID; an Object Instance has an Instance ID; an Attribute has an Attribute ID; and a Service has a Service Code.

In order to distinguish an Object within a network, a combined Identifier of MAC ID, Class ID, and Instance ID is used (e.g. MAC ID #4: Object Class #5: Instance #2).

The model of a DeviceNet system is shown in Figure 3-10.

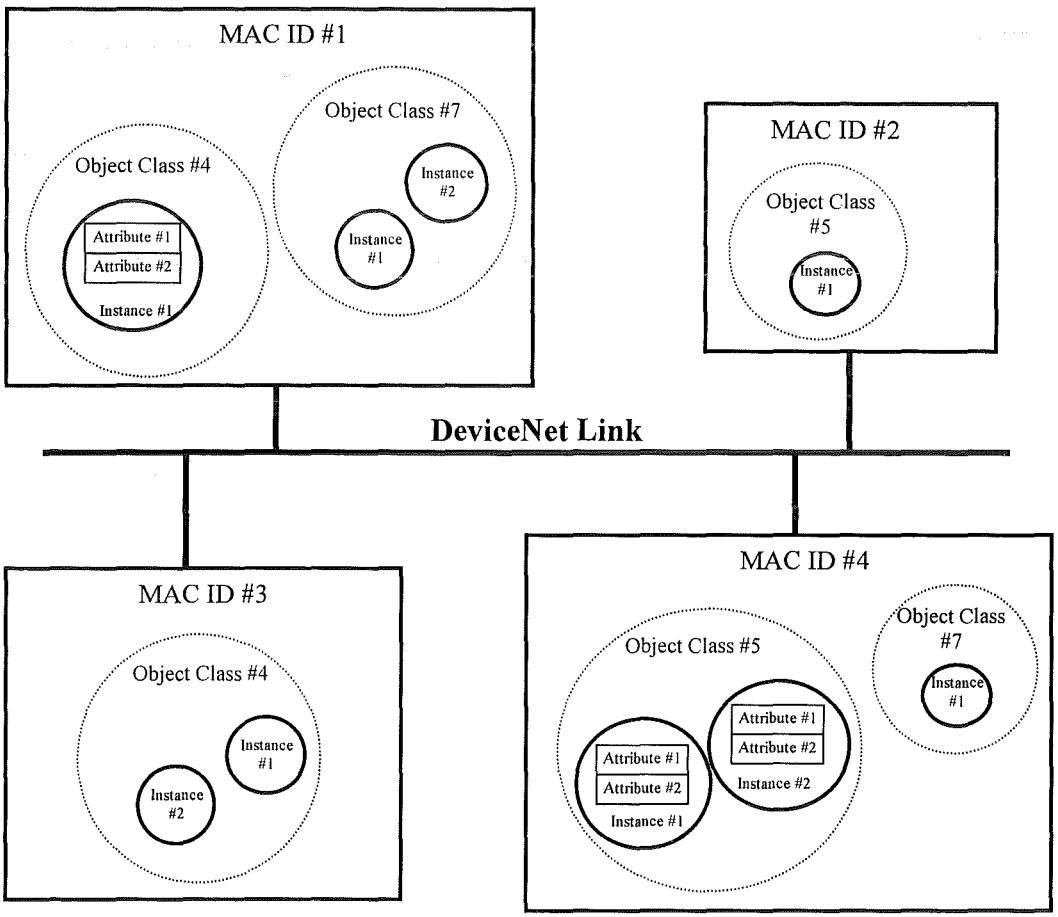


Figure 3-10 DeviceNet Model

With respect to the relationship between CAN and DeviceNet, a DeviceNet network utilises CAN's Data frames to exchange its messages. Data is exchanged in a DeviceNet network by means of Explicit messages and Input/Output (I/O) messages.

- **Explicit messages** are used to transfer generic, multi-purpose data such as requesting a connection with other devices, reporting errors, and so on (DeviceNet 2.0 Specification, Vol. 1, section 4-2).

  It is noted that the *common services* (as mentioned in the examples previously) are transferred within Explicit messages.

- **I/O messages** are used to exchange special-purpose data. These types of messages are used by I/O devices to transfer their information during the performance of their tasks.

  Note: DeviceNet does not define any protocol for the Data field of an I/O message (DeviceNet 2.0 Specification, Vol. 1, section 4-2). This enables users to design data formats that suit their needs. This feature of DeviceNet further reflects its flexibility in comparison with the SDS protocol.

The formats of these two types of DeviceNet messages are described in section 3.3.3.2.

3.3.3 DeviceNet Application Protocol

3.3.3.1 Use of CAN Identifier in the DeviceNet protocol

As referred to previously, each DeviceNet's node has a unique MAC ID which serves as its identifier in a network. In the DeviceNet protocol, the 11-bit Standard format Identifier of a CAN message is used to assign the MAC ID to a node, as well as the message identifier (Message ID).

A DeviceNet node can belong to one of four groups: Group 1, Group 2, Group 3, and Group 4. The formats of these Groups' messages ID and MAC ID are shown in Figure 3-11.

Identifier bits											Hex Range	Group
10	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**		
0	Message ID				Source MAC ID						000-3FF	1
1	0	MAC ID						Message ID			400-5FF	2
1	1	Message ID			Source MAC ID						600-7BF	3
1	1	1	1	1	Message ID						7C0-7EF	4
1	1	1	1	1	1	1	x	x	x	x	7F0-7FF	Not use

Figure 3-11 DeviceNet's use of the CAN Identifier Field

According to the CAN protocol, messages in Group 1 have the highest priority, and Group 4 messages have the lowest priority.

Note that the MAC ID fields in Group 1 and Group 3 contain the source nodes' addresses. In Group 2, the MAC ID field can be either the source or destination address (DeviceNet 2.0 Specification, Vol. 1, section 3-2).

When assigning a MAC ID to a node, or message ID to a message, the group priority should be considered. For example, in a car, the node controlling the air bag should be one of the highest priority modules; thus, its MAC ID should be assigned to Group 1. The node (or nodes) which carries out the tasks of engine management

could belong to Group 2. The node controlling the air conditioning could be one of the lowest priority; hence, its MAC ID could belong to Group 3.

Messages in Group 4 do not contain a MAC ID, hence, any node can utilise the Group 4 messages. These messages are solely used for system administration purposes such as recovering nodes which have gone off-line due to having the same network addresses with other nodes (DeviceNet 2.0 Specification, 1997, Vol. I, section 3.2.4).

In addition, the message priorities within a group are determined as follows:

- For Group 1 and Group 3, the message with the lower message ID has higher priority. When two or more messages try to access the bus simultaneously, the lowest message ID will gain the bus access. If the messages have the same message ID, then the message with the lowest MAC ID wins arbitration.

  For example, within Group 1, the device with MAC ID=20 and message ID=2 has higher priority than the device with MAC ID=5 and message ID=6.

- For Group 2, the lower MAC ID device has higher priority. When two or more messages try to access the bus at a same time, the lowest MAC ID node will gain the bus access.

  For example, within Group 2, the device with MAC ID=0 has higher priority than the device with MAC ID=1.

3.3.3.2 Use of CAN Data field in DeviceNet

Messages are exchanged in a DeviceNet network by means of Explicit and I/O messages.

I/O messages are left for users to define the information contained in the Data field of the messages (Figure 3-12).

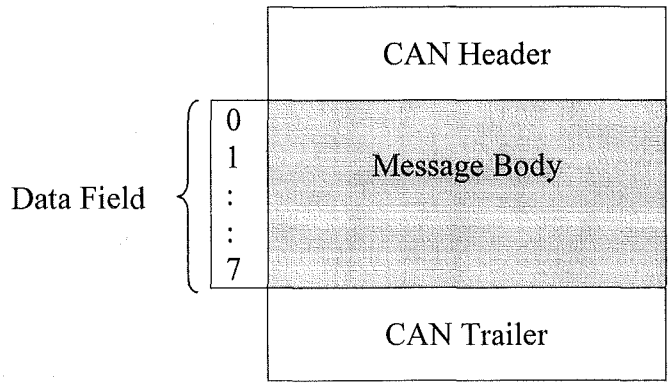


Figure 3-12 I/O message format

Explicit messages carry the *common service* information, which has been described in section 3.3.2. The format of an Explicit message is shown in Figure 3-13.

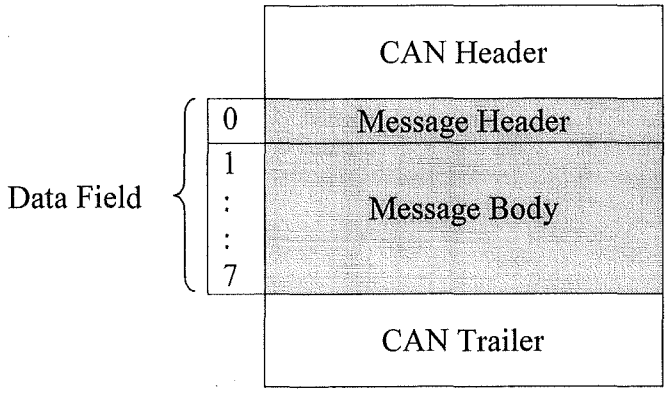


Figure 3-13 Explicit message format

The **Message Header** of an Explicit message contains information about:

- Whether the message is fragmented or not, and
- The MAC ID.

  Note that if the MAC ID in the Identifier field is the source ID, then the MAC ID carried in the Message Header is the destination ID, and vice versa.

The **Message Body** carries the service parameters of a specific service.

A typical DeviceNet message contains 0 to 8 bytes. However, data larger than 8 bytes can be transmitted as fragmentation, and is thus catered for in the protocol. The formats of fragment frames for I/O messages and Explicit messages are shown in Figure 3-14 and Figure 3-15, respectively.

The **Fragmentation protocol** field utilises the first data byte of the CAN data field. It contains information of the fragment type (i.e. the first, middle or last fragment), and the fragment number (DeviceNet 2.0 Specification, 1997, Vol. 1, section 4.4.1)

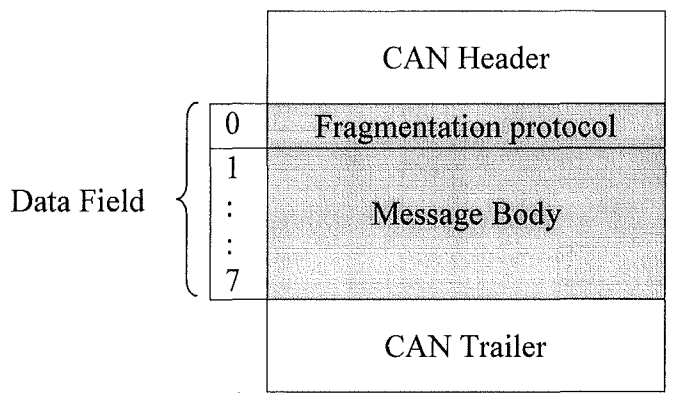


Figure 3-14 I/O message fragment format

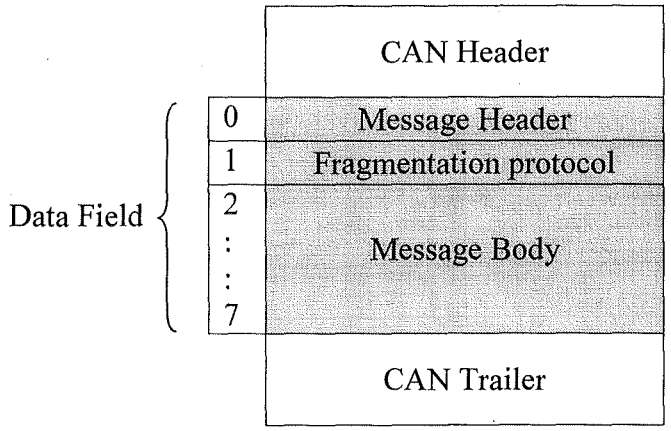


Figure 3-15 Explicit message fragment format

3.3.4 Advantages and disadvantages of DeviceNet

3.3.4.1 Advantages

The DeviceNet protocol is a complete Higher Layer protocol for CAN-based systems. It has the following advantages:

- Provides a solution for low cost networks.
- It is an open network for I/O devices.
- It has flexible data formats which utilise I/O messages.
- The design of DeviceNet meets the requirements for real-time control systems.

3.3.4.1 Disadvantages

The design of DeviceNet protocol has disadvantages as follows:

- It only utilises the Standard CAN format (11-bit Identifier)

- The use of the CAN Identifier is not efficient. No two nodes can have the same MAC ID. This violates the CAN specification where two or more nodes can have the same CAN ID to transmit or receive data.

- Amendment of the priority of a node or a message is an extremely complex task and time consuming.

- Only 64 nodes can exist on one DeviceNet link because the MAC ID range is 0 to 63.

  If a network needs to contain more than 64 nodes, however, it can be divided into subnets consisting of a maximum of 64 nodes. The subnets are connected together by *Network Routers* (DeviceNet 2.0 Specification, 1997, Vol. 1, section 1-6).

3.4 CAN Kingdom

3.4.1 Introduction to CAN Kingdom

This section introduces another approach for HLPs, called CAN Kingdom, which was developed by Kvaser AB (CAN Kingdom 3.01 Specification, 1996-1997).

Lennartsson and Fredriksson (1995), in discussing the basic idea of the CAN Kingdom protocol, state that "instead of specifying how modules should be finally designed, CAN Kingdom specifies how modules can be adjusted to actual system

needs". Whereas, in other HLPs such as Smart Distributed System (SDS) and DeviceNet, modules have to follow the standards exactly.

Problems arise with SDS, DeviceNet or other CAN HLPs when modules following different protocols are not able to work on the same network. This inconsistency is due to the conflicting interpretation of messages at the Application Layer Protocol. In addition, a module cannot know which other modules to communicate with in SDS and DeviceNet systems (Lenartsson and Fredriksson, 1995).

In order to overcome these disadvantages of other HLPs, instead of developing a complete new HLP, Kvaser AB introduced a new approach for the CAN Kingdom protocol. The protocol consists of a set of protocol primitives which designers can use to build their final HLP to suit their system needs.

In order to highlight this feature of CAN Kingdom, Korane (1996) quotes the president of Kvaser, Lars-Berno Fredriksson, who claims that: "The advantage of CAN Kingdom not being a protocol but a set of protocol primitives is that, especially for real-time systems, the system designer can choose the topology and bus access management best suited for the application". Lennartsson and Fredriksson (1995) also note that "In fact, DeviceNet and SDS modules can be integrated into CAN Kingdom systems (but not vice versa)".

The fundamental aspect of a CAN Kingdom system is the Network Manager, called the King, which is responsible for the whole network configuration and decides which nodes communicate with each other. Nevertheless, after setting up the network and deciding upon the communication between modules, the King can usually be removed from the system. Then, the system can inherit the full potential of the CAN protocol such as Multi-Master and Broadcasting (see Chapter 2). In some cases, the system can even be designed to work in other modes (Lennartsson and Fredriksson, 1995).

3.4.2 CAN Kingdom Basic Concept

3.4.2.1 CAN Kingdom model and terminologies

According to the CAN Kingdom 3.01 Specification (1996-1997), a CAN Kingdom system is described as being analogous to a country, a Kingdom, with a master node (called the King or the Capital), and CAN modules (called Cities). The exchanges of information between Cities, as well as between the King and Cities, are done by means of letters (or mails) via a postal system. Figure 3-16 shows the model of a CAN Kingdom system.

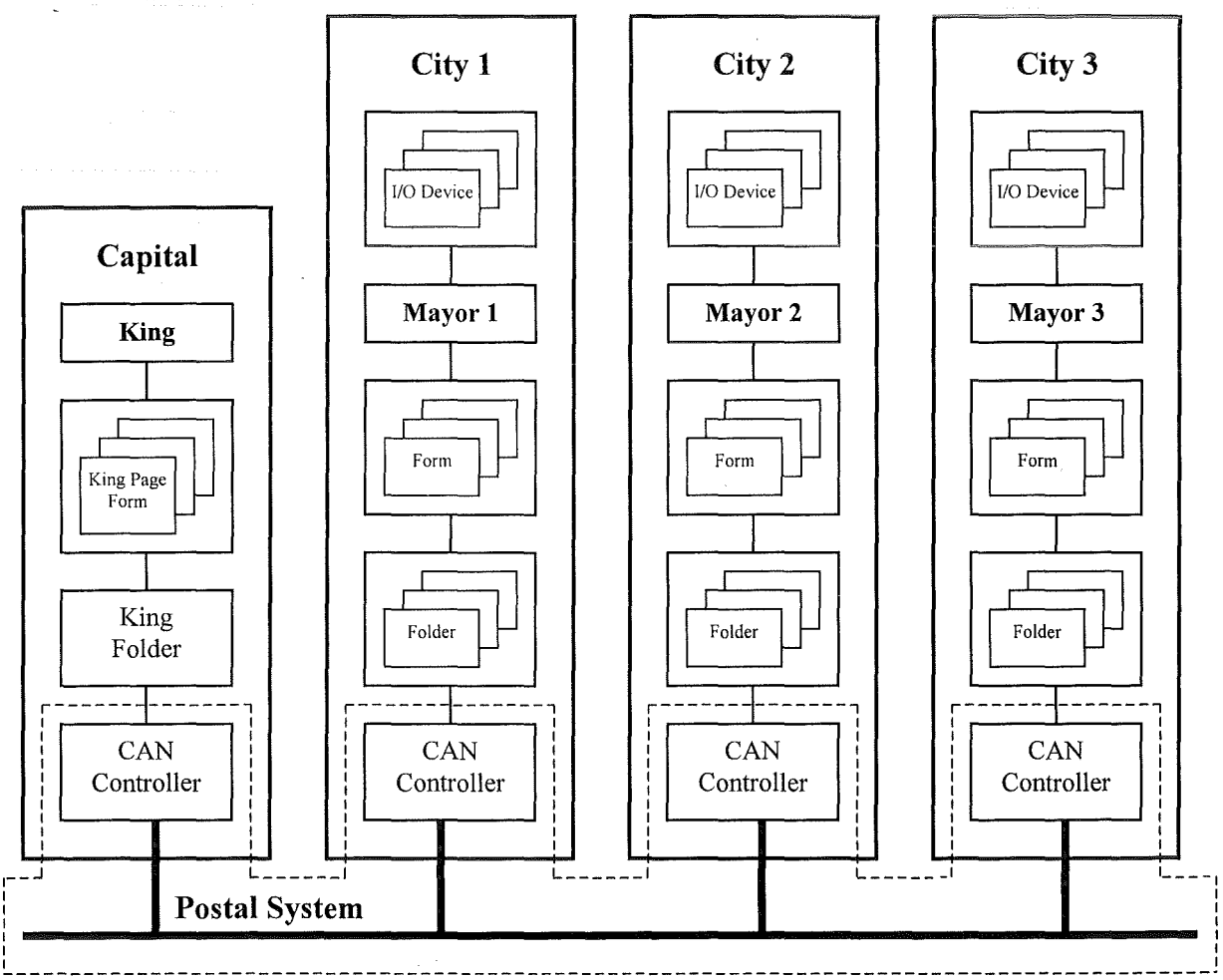


Figure 3-16 CAN Kingdom Model

The following list introduces terminologies defined in the CAN Kingdom specification:

- The **Capital** and the **King** are actually the Network Manager node and the software for controlling the network, respectively.

- A **City** and the **Mayor** of the City are the actual CAN node and the software to control this node.

- **Input / Output (I/O) Devices** are the devices to carry out the tasks of the CAN node such as sensors, actuators, switches, and so on.

- **Forms** are the tools in the CAN Kingdom protocol used to encode and decode CAN messages into meaningful information. A Form tells other Cities where on a page a certain piece of information should be placed or is expected, and in which format (BCD, Integer, etc.) the information is presented.

- **Folders** serve as letterboxes for incoming and outgoing Letters (CAN messages). Unlike normal letterboxes, each Folder contains one Letter at a time and has a Folder number from 0 to 255. A Folder has a Form (or Forms) associated with it so that the software knows how to encode and decode the data.

- The **Postal system** includes the CAN bus and the CAN protocol.

- The **Kingdom Founder** is the system designer, who decides how the system works.

- The **City Founder** is the module designer, who is responsible for the control mechanism that actuates the tasks of the CAN module (City).

- An **Envelope** is the CAN Identifier of CAN messages.
- A **Line** is one CAN data byte in the Data field of a CAN message.
- A **Page** is the CAN data field. As the CAN data field consisting of 0 to 8 bytes, each Page contains 0 to 8 Lines.
- **Letters** are the messages which are exchanged in the system. A letter consists of an Envelope and a Page. The Envelope serves as the destination address, to which the letter should go. The Page contains information of the Letter.

The Figure 3-17 shows the relationship between a CAN Data Frame and a CAN Kingdom Letter.

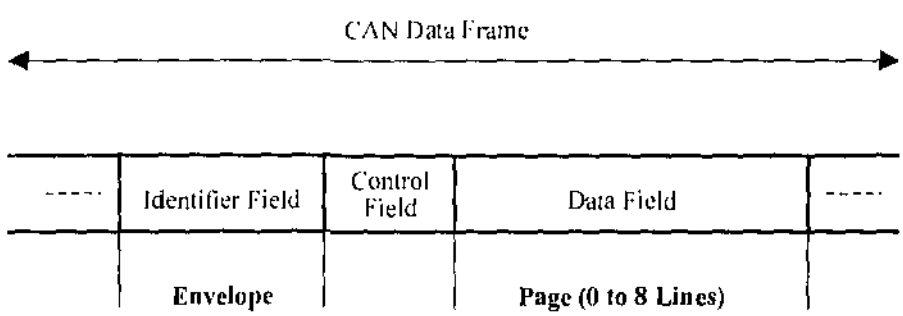


Figure 3-17 A CAN Kingdom Letter

3.4.2.2 Basic concept of CAN Kingdom

In the CAN Kingdom specification, the operation of a CAN Kingdom system can be divided into two phases:

- Set-up phase, and
- Run phase.

All the system configurations including data formats, bus management, CAN Identifier assignments, and so on, are executed during the Set-up phase. In the Run phase, the system operates according to the configurations established in the Set-up phase.

The basic idea of the CAN Kingdom is that at initialisation (the Set-up phase), all Cities have to wait for instructions from the King. The King is responsible for the configuration of the system. It owns all the Envelopes and assigns them to the Folders that keep messages to be transferred in the system. In other words, the King Founder usually decides upon the priorities of the messages and the communication between Cities during this phase.

It is clear that when designing a City, the City Founder (the Module Designer) does not need to be concerned about how his or her City will communicate in a particular system. Instead, the duty of the City is only to know how to receive and to follow the King's instructions.

Furthermore, because a City, when connecting to the system, cannot do anything before the King sanctions it, any City with a wrong baud rate setting can never destroy the system (Lennartsson and Fredriksson, 1995).

After setting up all necessary configurations for the system, the King tells the Cities when the Set-up phase has finished; then, the Cities can start to work as designed.

At the Run phase, the King can be removed; and thereafter, it does not get involved in the system operations.

The concept of Set-up phase and Run phase in the CAN Kingdom protocol can be demonstrated in Figure 3-18 and Figure 3-19.

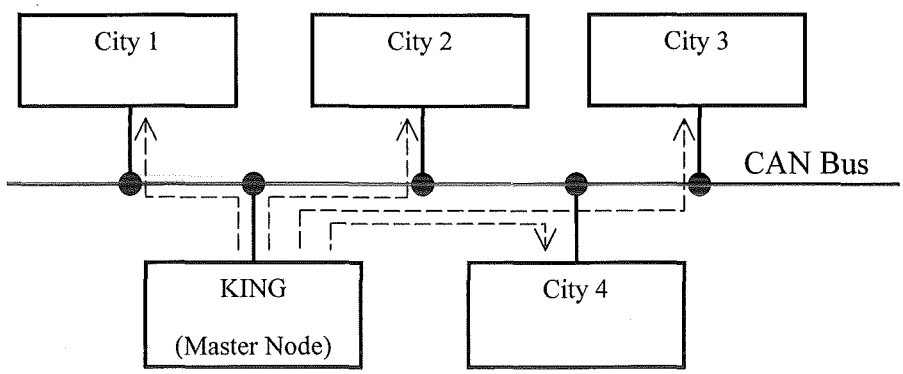


Figure 3-18 **Set-up phase**

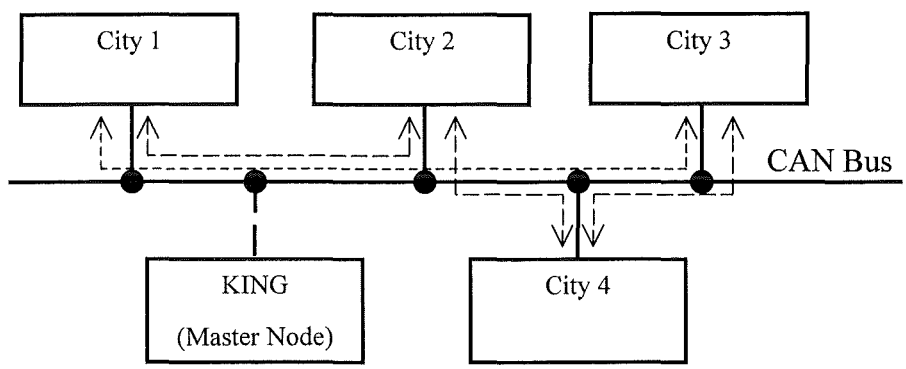


Figure 3-19 **Run phase**

3.4.3 Application Layer Protocol

The application services are provided in the CAN Kingdom protocol by means of Forms (see section 3.4.2). The Forms in the CAN Kingdom inform Cities where the expected data should be placed. Upon reception of a message (a Letter), a City picks up the Form associated with the receiving Folder in order to decode the data.

In the CAN Kingdom specification, there are several predefined Forms for King Pages as well as Forms to be used in Cities (CAN Kingdom 3.01 Specification, 1996-1997, p. 58). The CAN Kingdom Forms provide a flexible way which enables designers to construct suitable data formats for their system needs because the Form formats can easily be changed when required. The description of Forms used in Set-up phase and Run phase are detailed below.

During the Set-up phase, the King sends out its instructions to Cities via the King Pages which are constructed according to the King Page Forms. Upon reception of a King Page, the City (or Cities) uses a similar King Page Form to decode the Page into a meaningful instruction.

Note that a City has a unique address among the network so that the King can talk to an individual City. The King can also broadcast its messages to all Cities, or to a group of Cities if necessary. In the CAN Kingdom protocol, the highest priority Envelope (CAN Identifier) is reserved for identification of King Pages.

An example of a King Page Form is shown in Figure 3-19.

Document Name:	King Document		
Document List:	T1 Capital / 0 City		
Document Number:	0 Capital / 0 City		
Document Type:	Transmit (Capital)		
	Receive (City)		
Page Description.			
Page Number:	0		
Number of Lines:	8		
Data Description:	The King Page 0. Terminates the Set up phase. Order the Mayor to set its City into a specific working mode, eg in a Run or Freeze mode.		
Line Description			
Line 0: City or Group Address			
Line 1: 00000000	(Page 0)		
Line 2: rrrrrrAA	Action Mode		
		AA=00	Keep current Mode
		AA=01	Run
		AA=10	Freeze
		AA=11	Reset
		r=0	Reserved
Line 3: rrrrrrCC	Communication Mode		
		CC=00	Keep current Mode
		CC=01	Silent
		CC=10	Listen only
		CC=11	Communicate
		r=0	Reserved
Line 4: MMMMMMMM	City Mode		
		M=0	Keep the current Mode
		M≠0	Modes according to the City specification
Line 5: rrrrrrrr	r=0	Reserved	
Line 6: rrrrrrrr	r=0	Reserved	
Line 7: rrrrrrrr	r=0	Reserved	

Figure 3-20 Example of a King Page Form

The purpose of the King Page in Figure 3-20 is to inform the Cities that the Set-up phase has finished and the Cities can start to work in the mode specified by the page.

This King Page reveals:

- Line 0 of the page contains the City or Group address; thus, the City or the group of Cities with the address specified on the Line must follow the instruction presented by this King Page. Other Cities ignore this Page.

- Line 1 of the page contains the Page number (Page 0), which enables the receiving City to pick up the corresponding Form to decode the Page.

- The other Lines of the Page contain configuration information that the Cities have to follow when receiving this Page.

Importantly, during the Set-up phase the King sends many King Pages of information in order to establish each City's role in the Kingdom.

After the Set-up phase, the CAN Kingdom system can work as designed. This stage is called the Run phase. At this stage the King can be removed, as previously noted, from the system if no configuration changes are required.

In the Run phase, Cities communicate with each other as specified in the Set-up phase. They transmit and receive information to and from each other via Letters (CAN messages). Upon reception of Letters, a City uses Forms to decode the Letters to meaningful information. The data type of the message is also described in the Forms.

Note that if two or more Cities want to exchange information, they must have exactly matching forms in each City's itinerary.

Figure 3-21 gives an example of a Run phase Form for Cities which are typically used in temperature measurement applications.

Form List:	R0
Form No:	9
Document Name:	TEMPERATURE 1
	TEMPERATURE 2
Document No:	R0.2, R0.3
Document Type:	Receive
Page Description	
Number of Lines:	1
Data Description:	TEMPERATURE
	Range: 0-255^{o}C
Line Description	
Line 0	Temperature in Centigrade
	Resolution: 1^{o}C
	0 ^{o}C = 00000000
Data Format:	8-bit unsigned Integer

Figure 3-21 Example of a Form to be used in Run phase

The purpose of the Form in Figure 3-21 is to encode and decode the temperature in Centigrade. A message constructed by this form contains one Line (CAN data byte). The information carried by this line is temperature with the 1^{o}C resolution in 8-bit unsigned integer format. A transmitter sends temperature information according to this Form. Then, a receiver uses a similar Form to decode the incoming message.

In using the concept of Forms, City Founders (node designers) have the opportunity, and flexibility, to design their own Forms for their modules.

In order to inform the system designer (the King Founder) of details about the tasks of the Cities, the City Founders have to document all the Cities' Forms when designing their Cities.

Each City may contain a document for transmitting and a document for receiving information. A document consists of a set of Forms. Therefore, by studying the Cities' documents, the King Founder can decide which Cities are suitable for the system and the role of each City in the system.

Furthermore, unlike Smart Distributed System and DeviceNet which only utilise the CAN Standard format (11-bit Identifier messages), the CAN Kingdom protocol is specified for use of both Standard and Extended formats (29-bit Identifier messages) (Lennartsson and Fredriksson, 1995).

The CAN Kingdom protocol also specifies how to transfer data larger than a normal Page (8 bytes). In doing so, at least some bits of a Line in a Page are reserved for pagination, and the data usually is transferred in the same Envelope. Upon reception, the receiver can rearrange the data in the right order by looking at the Page numbers. Hence, if any Page has already been received, it can easily be omitted.

One example of transferring data larger than one Page is the transmitting of King Pages, where all King Pages are transmitted in the same Envelope (Envelope 0), and Line 1 of each King Page contains the Page number. When receiving the Page, the Cities know which Forms are to be used to decode the data by looking at the Page number (CAN Kingdom 3.01 Specification, 1996-1997, p.56).

3.4.4 Advantages and Disadvantages of CAN Kingdom

3.4.4.1 Advantages

The main purpose of the CAN Kingdom is to develop an open protocol with a set of protocol primitives. which the System Designers can use to construct a suitable protocol for their own needs. This approach has several advantages:

- The nodes following different Higher Layer Protocols for CAN can be integrated into a CAN Kingdom system with only minor adjustments in software.

- A node with a wrong baud rate setting in the CAN Kingdom system does not ruin the system because it has to obey the King's instructions when connecting to the network. and cannot do anything before the King gives permission.

- Although the King is responsible for establishing any communication in the Kingdom (CAN System), it can be removed from the network after providing all the necessary configurations and consistency checks for the system. Nevertheless, whenever required, the King can be reconnected and can send instructions to the system.

- Both Standard and Extended CAN frame formats can be used.

- The Remote Transmit Request (RTR) bit is utilised in a CAN Kingdom system.

3.4.4.2 Disadvantages

The disadvantages of the CAN Kingdom protocol are:

- CAN Kingdom is not a complete Higher Layer Protocol
- System Designers have to build up their own final protocol

However, the purpose of the CAN Kingdom protocol is a versatile system based on an open approach for Controller Area Network (CAN).

3.5 Conclusion

This chapter has presented an overview and discussed the importance of Higher Layer Protocols (HLPs) for CAN-based systems. The main features of three popular HLPs have been described.

As discussed, the Smart Distributed System provides an efficient protocol for communications between I/O devices and host controllers. The protocol meets the requirements of speed, reliability, and real-time support in control systems used in a wide range of automation industries. However, the protocol has disadvantages that reduce the potential use of the CAN protocol (e.g. only Standard format frames are utilised in the SDS protocol). Moreover, the SDS protocol is not flexible to provide users opportunities to enhance their systems.

DeviceNet is an open network where all nodes have the same right to access the bus. The protocol is also efficiently used to control I/O devices. The Object-Oriented approach of DeviceNet makes it more flexible than SDS. Instead of specifying fixed application services as SDS does, each object in DeviceNet can provide different services. Users also have more control when designing their

systems with the means of I/O messages. However, similar to SDS, DeviceNet uses only the Standard CAN format.

In addition, no two or more nodes can have the same Logical Addresses in an SDS system or MAC ID in a DeviceNet system. This violates the use of CAN Identifier where two or more modules can utilise the same CAN ID for the exchange of data. Furthermore, it is not easy to change a Logical Address or a MAC ID once it has been assigned. In other words, the priority of a node or a message cannot be changed without a complex redesign of the system.

The CAN Kingdom protocol provides more flexibility to its users. The features of the CAN protocol are used more effectively in a CAN Kingdom system. Designers are free to develop their own CAN modules. This means users can design their modules independently without being concerned about how the modules are going to work in a specific system.

The master node, the King, in a CAN Kingdom system decides the role of each module in the system. As a result, the priorities of messages or nodes can easily be changed even when the system is in run-time.

One of the great advantages of the CAN Kingdom protocol is that SDS, DeviceNet or other HLPs modules can be integrated in a CAN Kingdom system. In fact, CAN Kingdom system designers can select any suitable modules for their system.

The disadvantage of the CAN Kingdom protocol is that it is not a complete protocol; hence, designers have to construct their own final protocol.

CHAPTER 4

DESIGNING A HIGHER LAYER PROTOCOL FOR SMALL DISTRIBUTED MICROCONTROLLER SYSTEMS USING THE CONTROLLER AREA NETWORK PROTOCOL

This chapter is concerned with designing a Higher Layer Protocol (HLP) for small Distributed Microcontroller Systems using the Controller Area Network (CAN) protocol. This follows the choice of one of the three HLPs, which were discussed in Chapter 3, the CAN Kingdom protocol. A small CAN-based distributed system is then designed to implement and to test the protocol.

4.1 Choosing a Higher Layer Protocol

The main purpose of the design of the HLP in this project is:

- To achieve a simpler HLP for small CAN systems which have restrictions such as the limited amount of memory for control program, and the ease of system design.
- To ensure that the design methodology of the HLP is easy to understand.
- To provide opportunities for later designers to expand the application of this project's progress.

As discussed in Chapter 3, Smart Distributed System (SDS), DeviceNet, and CAN Kingdom protocols are all based on the CAN protocol. Each one of them is suitable

for particular applications in automation processing industries. The main features of these protocols are summarised as follows:

- SDS is based on point-to-point communication between a master (Host) and remote Input / Output (I/O) devices.

- DeviceNet is an open system in which all modules have the same right to access the bus.

- CAN Kingdom specifies a set of protocol primitives which system designers can use to build a final HLP to suit their needs. The principle of the CAN Kingdom protocol is that a master node in a system, the King, is responsible for the entire network configuration; yet, the King can be removed after the system is set up, and leave the Cities to perform the task of the system.

Nevertheless, the main disadvantages of the SDS and DeviceNet protocols are that designers have to follow the standards exactly, and hence, there are a few chances to modify the protocols to satisfy the requirements of a particular system. Moreover, these protocols utilise only the Standard format of the CAN protocol (see Chapter 3), and they are too complex to fit into the small amount of memory available in a small system. Furthermore, the designers, following SDS and DeviceNet standards, must be fully aware of how their nodes are going to work in a certain system. They also have to decide in advance the communication between the nodes.

The CAN Kingdom, on the other hand, is more flexible, as designers can easily select the services that are suitable for their system. In other words, the CAN Kingdom services can be chosen to fit into the small amount of memory if the restrictions of the system are the limitation of memory and the ease of design.

In addition, when designing a node, the designer does not need to be concerned about its communication in a typical CAN Kingdom system. This is the role of the King to decide which nodes communicate with each other. This means that module

designers can concentrate on the design of their modules to do particular tasks, without having to know how they will be used in a specific network.

For example, if a temperature measurement module is designed to measure the temperature in a coolant system, the main task of the designer is to design the module so that it can get the temperature information and store it into a Folder. When this module is connected to a CAN Kingdom network, the King will decide which other nodes receive its temperature information.

This feature of the CAN Kingdom protocol enables the nodes which follow the rules of other HLPs such as DeviceNet or Smart Distributed System to be integrated into CAN Kingdom systems (Lennartsson and Fredriksson, 1995).

Furthermore, the CAN Kingdom protocol is also specified to use both Standard and Extended CAN formats (CAN Kingdom 3.01 Specification, 1996-1997). Therefore, designers can utilise the latest technology of the CAN protocol.

Despite the advantages such as the above, the full implementation of the CAN Kingdom protocol is a complex matter; and hence, it is the aim of this thesis to show that further simplifications can be made to the protocol in order to suit the requirements of a small system.

It should be noted that the design of the Higher Layer Protocol in this project is based on the basic ideas of the CAN Kingdom protocol such as the responsibility of the King, and the use of Forms. Yet, the implementation of the protocol is different from the CAN Kingdom protocol such as the addressing method, and Form design.

Because the HLP in this project has been designed according to CAN Kingdom specification, it is named the **Small CAN Kingdom** protocol.

Note that the Small CAN Kingdom protocol uses the same terminologies as the CAN Kingdom protocol, such as the King, Cities, Forms, and so on (see Chapter 3).

4.2 Designing the Small CAN Kingdom protocol

4.2.1 Introduction

The main idea behind the CAN Kingdom protocol is that a node, when connected to a system, has to wait for instructions from the King. The King tells each node which other nodes it will communicate with. In doing so, the King sends a set of King's messages to a node or a group of nodes in order to set up the rules for the Kingdom.

To enable the King to send instructions to a particular City, each City has to have a unique address among the network. In the Small CAN Kingdom protocol described in this thesis, a simpler addressing method is used to access the Cities. Each City reserves one byte of memory to store an integer value in a range of 1 to 255. This integer value serves as a City's address in the Kingdom.

Note that the address 0 is reserved for the King to send broadcast orders. This means that all Cities in the network belong to a group with address 0.

Each City can be designed independently without any prior knowledge, on the designer's behalf, of the network to which it will eventually be connected. However, the Cities have to be able to receive and obey the King instructions. In other words, the node designers should follow some rules to enable their Cities to receive the King Pages when they are connected to a system. These rules are described in the following sections.

The model of a Small CAN Kingdom system can be visualised in the same way as a CAN Kingdom system and is shown in Figure 4-1.

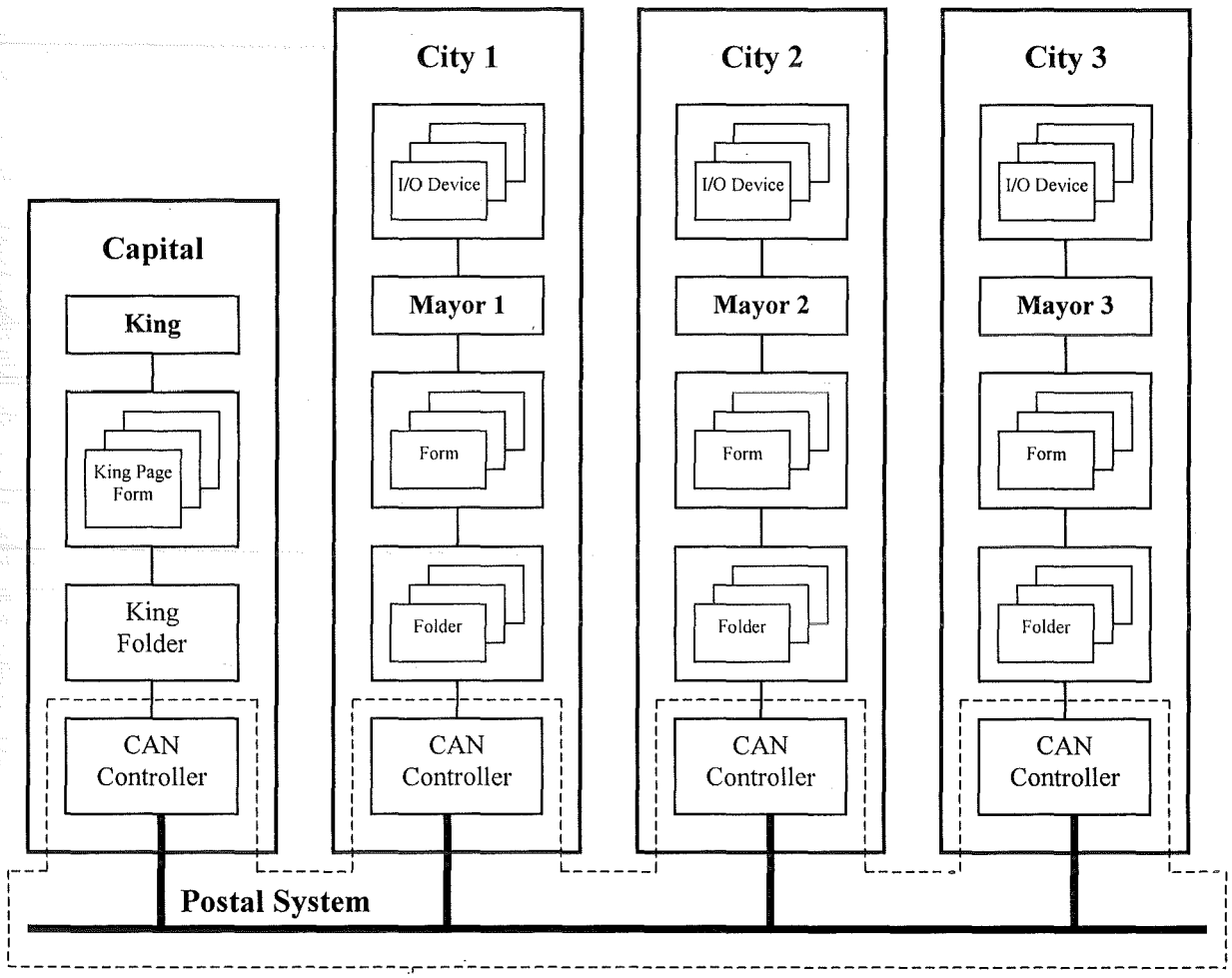


Figure 4-1 **Small CAN Kingdom Model**

Typically, a system designed according to the Small CAN Kingdom protocol consists of:

- A **Capital** with the **King** being responsible for the entire network configurations.

- A number of **Cities**, each of which has a unique network address so that the King can talk to each individual City. Each City carries out particular tasks specifying its role in the Kingdom.

The basic operations performed between the King and Cities and between the Cities themselves are described as follows:

The King sends instructions to the Cities via **King Pages**. The format of each King Page follows a specific **King Page Form**, which are used by a node to encode or decode the Page into a meaningful instruction. Before a King Page is sent out, it is stored into the **King Folder**, which acts as the bridge between the King and the postal system. It should be noted that the King Folder contains only one King Page at a time.

Each City has a **Mayor**, which is the software responsible for all the City's operations. A City also has a number of **Forms**, which are used to encode and decode incoming and outgoing messages. The City's **Folders** are the bridges between the Mayor and the postal system. They are used to store messages before being transmitted or read by the Mayor.

4.2.2 The King

The King is responsible for the entire network configuration. In fact, it is the responsibility of the system (network) designer. The system designer knows exactly how the system is intended to work, and how the Cities should communicate with each other in the system. His or her knowledge is then passed to the King and subsequently, the King sends instructions to the Cities to perform the configuration set-up.

The orders from the King are sent to the Cities by means of King Pages. Each King Page consists of two or more Lines (CAN data bytes). Each Line contains information meaningful to the Cities which receive the Page.

In all King Pages, the first Line of a Page contains the City's address, or the group address to which the City belongs. The second Line specifies the Page number indicating a specific King command. The other Lines, which may be included in a King Page, contain information required for the Page to execute a particular set-up.

Five King Pages of the Small CAN Kingdom protocol, which were designed and implemented in this thesis, are as follows:

1. **Page 0**: Terminate Set-up phase. The Cities are ordered to start to work.

2. **Page 1**: Assign an Envelope to a Folder.

3. **Page 2**: Change the City's address, or assign a City or a group to a new group

4. **Page 3**: Restore the original City's address or ungroup a group

5. **Page 4**: Baud rate setting

The descriptions of these King Pages are as follows:

- **Page 0** tells the Cities that the Set-up phase has finished. All the necessary configurations for the systems have been done, and hence, the Cities can start to work as designed.

- **Page 1** is responsible for assigning an Envelope (CAN ID) to a particular Folder. Folders are the parts of a City in which the City keeps its messages that are in the process of being, or having been, transmitted or received. This Page can be used to enable or disable a Folder. This means the King can enable or disable a City to transmit or receive a specific message.

- **Page 2**: The main purpose of this Page is to assign a City, or a group of Cities, into a new group. This enables the King to send orders to a group of Cities. This Page can also be used to change a City's address.

- **Page 3** is used to ungroup a group or restore the original City's address. When a group of Cities is ungrouped, the original address of each City is automatically restored. Consequently, if the King wants to talk to ungrouped Cities, the King has to send instructions to their original addresses.

- **Page 4** is used to set a new baud rate for the system. When connecting to the network, all Cities listen to the King's commands at a fixed baud rate. Then, new baud rates can be applied to the system by using this Page. It is noted that when the King changes the system baud rate, it has to change its own baud rate so that it can communicate with the Cities later on because all nodes in a CAN system have to operate at the same baud rate (see Chapter 2).

Note that the first two King Pages (Page 0 and Page 1) would be sufficient for a system which does not require complex set-up procedures. This is the minimum

requirement for a Small CAN Kingdom based system. In other words, these two Pages are mandatory for the system.

For example, when a system contains a few nodes, it may not be necessary to include King Page 2 and 3 (grouping and ungrouping instructions). However, the King can broadcast its instructions to all Cities by using the group address 0 because all Cities initially belong to group 0. In addition, if the system always works at a fixed baud rate, King Page 4 can also be omitted.

The information contained in each Line of a King Page Form is described as follows:

Page 0

Line 0: City or Group address

Line 1: Page number (Page 0)

Page 1

Line 0: City or Group address

Line 1: Page number (Page 1)

Line 2: Folder number

Line 3: Envelope value MSB (Most Significant Bit)

Line 4: Envelope value

Line 5: Envelope value

Line 6: Envelope value LSB (Least Significant Bit)

Line 7: Configuration

Note: Line 7 of Page 1 contains information to:

- Enable/Disable the Folder,
- Set the Letter in the Folder to Transmit/Receive, and
- Set a Letter to Standard/Extended format.

Page 2

Line 0: City or Group address

Line 1: Page number (Page 2)

Line 2: New City's or Group address

Page 3

Line 0: City or Group address

Line 1: Page number (Page 3)

Page 4

Line 0: City or Group address (should be the group address 0)

Line 1: Page number (Page 4)

Line 2: Baud rate value

Line 3: Baud rate value

Note that Lines 2 and 3 of Page 4 contain information for setting the system baud rate. This information is discussed in Chapter 7.

It should also be noted that only five King Pages have been described and implemented due to the time restrictions of this project and the limitation of on-chip memory available in the MC68HC11 microcontrollers. However, it is possible for later designers to add more King Pages for future expansion.

4.2.3 Cities

As mentioned previously, Cities have to wait for instructions from the King before they can start to work. Moreover, they should also be able to obey the King's instructions while they are performing their normal tasks. Therefore, the design of a City has the following requirements:

- A City must always be able to receive and obey the King's commands.

- The King's instructions should have the highest priority so that the Cities can receive and perform the instructions immediately while they are working.

Firstly, to achieve the requirements above, each City has a special Folder for the reception of King Pages and a set of King Page Forms associated with this Folder to decode the King Pages. These Forms are the same as the King Page Forms contained in the Capital. When a King Page arrives, by looking at the Page number, the City's Mayor can select the right Form to decode and perform the Page's task.

Secondly, each City reserves the highest priority Envelope (CAN ID) for the reception of King Pages. This allows the City to receive the Pages immediately, even though the bus load on the network is high.

In addition, and as discussed earlier, a City can be designed independently from another, and it does not need to be aware of how it is intended to work within a specific system. The King (or actually, the system designer) is responsible for that role. Therefore, each City has to be documented to inform the system designer what tasks it can do, and what information it transmits or needs to receive. This enables the designer to decide the communications between the Cities.

Each City may consist of three sets of documents as follows:

1. The **King Document** which contains information relating to the original address of the City and a set of King Page Forms. Note that the original address of a City should be left for the system designer to assign in order to avoid conflict between Cities.

2. The **Transmit Document** which informs the system designer what type of information the City can send out, where the information is located (Folder number), and which Forms to decode the information.

3. The **Receive Document** which tells the system designer what type of information the City needs to receive, where the information will be stored (Folder number), and which Forms are required to decode the information.

It is noted that when designing a system, the designer should ensure that each City can receive all the King Pages used in the system; although this is not mandatory, because the King can send orders to an individual City. However, if Page 4 (Baud rate setting) is used in a system, all the Cities must be able to perform it. Otherwise, the system could be out of action because all nodes in a CAN-based system have to work at the same baud rate.

Furthermore, if two or more Cities exchange information, they have to contain exactly matching Forms. For example, the documents housed in City 1 and City 2 inform the system designers that City 1 transmits temperature information from Folder 2, and City 2 receives temperature information in Folder 3. The Cities' documents also state that the Form for encoding data in Folder 2 of City 1 is the same as the Form to be used to decode data in Folder 3 of City 2. Therefore, during Set-up phase, the designer orders the King to assign the same Envelope to the two Folders so that the two Cities can communicate with each other.

For the sake of simplicity of the Small CAN Kingdom protocol, each City contains fifteen Folders with the Folder number in the range of 1 to 15. The first Folder

(Folder number 1) is reserved to receive King Pages. The rest of fourteen Folders can be used to store the City's Letters while it is working. However, the number of Folders can be increased, if required for future expansion.

During the network's run-time, each Folder can be used to store messages for a specific I/O device. When the device has something to be sent to another node (or nodes), the Mayor uses one of the Forms associated with this Folder to encode the data to form a Page, then stores the Page into the Folder. This Page and the Folder's Envelope, which is assigned by the King, are used to construct a Letter. The Letter can then be sent straight away by the City's Mayor, or by remote request from other Cities. A City sends remote requests by means of CAN Remote frames (see Chapter 2).

When a new Letter arrives, it is stored in a Folder. Then, the Mayor is notified (by an interrupt, for instance) and the reverse process is performed to control the I/O device. In other words, the Mayor picks up one of the Forms associated with the Folder in order to decode this Letter and then send the data to the I/O device.

Note that although one Folder is usually used to control one device, two or more similar devices can utilise the same Folder. For example, one single command can be invoked to set parallel switches to ON or OFF at the same time, and consequently, only one Folder is required for performing this task.

The operations of a Small CAN Kingdom system can be more fully understood by undertaking the design of an actual system as described in the following section.

4.3 Design a Small CAN Kingdom system

The aim of this section is to introduce the methodology for the design of a small distributed microcontroller system utilising the Small CAN Kingdom protocol. The actual hardware and software designs are covered in the later chapters.

4.3.1 Introduction

The purpose of the system is:

- To build a complete Higher Layer Protocol (HLP) for a small distributed system, and

- To make it easy to demonstrate the performance of the HLP in such a way as to illustrate the responsibility of the King, and the method of the communication between the King and Cities, as well as between Cities themselves.

The system is designed to include a Capital with the King controlling the network, and three Cities:

- The **King**, the master node, is responsible for the system configurations and deciding the communication paths within the system.

- **City 1** gets Analog / Digital (A/D) signals from an I/O device and sends to the CAN bus whenever the signal has been changed.

- **City 2** also gets A/D signals from an I/O device but only sends to the CAN bus when it receives a remote request (CAN Remote frames) from City 3.

- **City 3** is responsible for receiving information from both City 1 and City 2. The information is then displayed on a Liquid Crystal Display (LCD) along with the transmitting City's address. This City also sends a request to City 2 if the data is needed.

The reasons for choosing A/D signals for the City 1's and City 2's tasks are that A/D signals are used widely in devices such as temperature or speed measurement sensors. The LCD in City 3 provides a visual way to demonstrate the operations of the system.

4.3.2 System design

4.3.2.1 The King

The King is designed to receive set-up information from the system designer. The information is then encoded according to the King Page Forms and sent to Cities via the King Folder and the postal system.

Note that the King is also notified when a Page is successfully transmitted. This enables the King to configure itself if necessary. For example, if the King sends out Page 4 (Baud rate setting), it also needs to change its own working baud rate to that of the network.

The operation process of the King is shown in Figure 4-2.

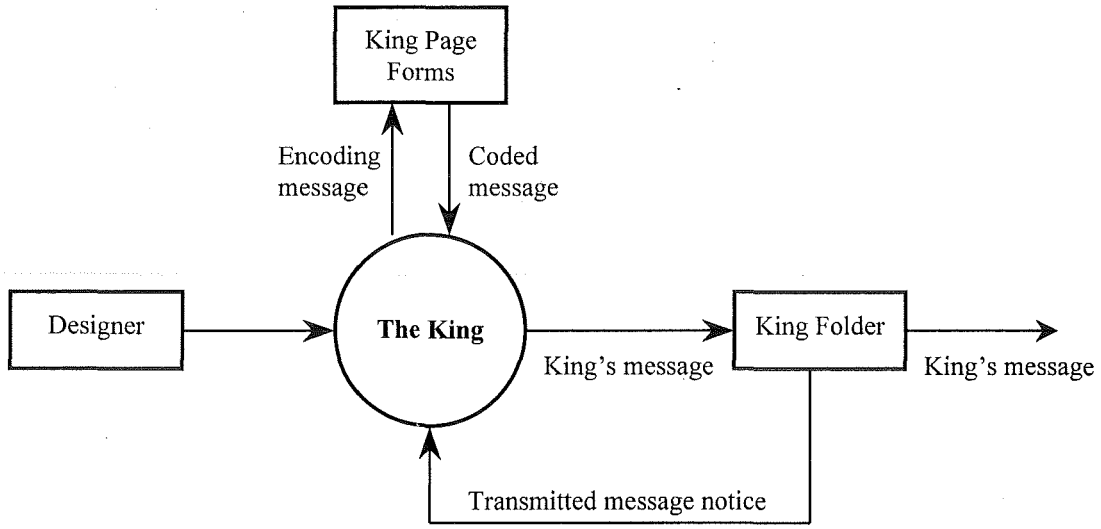


Figure 4-2 **The King Process**

4.3.2.2 City 1

The task of City 1, during run-time, is to get A/D signals from an I/O devices and send it to the CAN bus whenever the signal value has been changed.

The A/D signal is encoded to ASCII values by a Form, called the A/D Form in the City's Transmit Document. The data is stored in Folder 2 of the City. Then, the City's Mayor sends the data along with the City's address onto the bus.

The operation process of the City is shown in Figure 4-3.

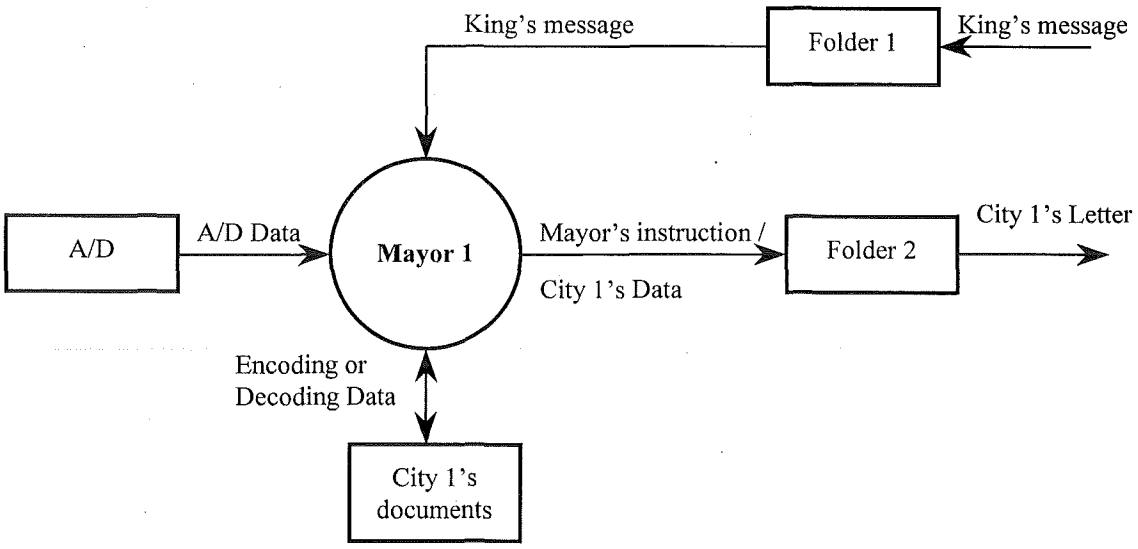


Figure 4-3 City 1's Operation Process

City 1 has two documents which inform the system designer about its task. The two documents are the King Document and the Transmit Document as shown in Figures 4-4 and 4-5.

Note that the City does not have the Receive Document as it does not receive any data.

The King Document

City's Address: 001
List of King Pages • King Page 0 • King Page 1 • King Page 2 • King Page 3 • King Page 4

Figure 4-4 The City 1's King Document

The King Document informs the system designer that the City's address is 001 and that the City can receive all five King Pages (from Page 0 to Page 4).

Transmit Document

This document contains the A/D Form associated with Folder 2. City 1's messages (Letters) are constructed according to this Form.

Form Type: Transmit Location: Folder 2 Remote Request: No Data Type: ASCII code (for all Lines)
Line Description Line 0: City's Address (Hundreds) Line 1: City's Address (Tens) Line 2: City's Address (Units) Line 3: ':' character Line 4: A/D Value (Hundreds) Line 5: A/D Value (Tens) Line 6: A/D Value (Units) Line 7: EOT (End of String character)

Figure 4-5 City 1's Transmit Document

The A/D Form carries the following information:

- It is used to encode the transmitting messages.
- The messages are located in Folder 2.
- The messages are transmitted to the CAN bus straight away without any request.
- Data types for all Lines in the messages are ASCII code. Note that it is possible to design different data types in each Line.
- Line 0 to Line 2 contain the City's address (001). This address can be changed by the King.
- Line 3 consists of the ':' character to separate the City's address and the A/D value.
- Line 4 to Line 6 carry the values of A/D signals.
- Line 7 is the End Of String character (04 Hex). This indicates the end of an ASCII string.

4.3.2.3 City 2

The task of City 2 is to get A/D signals from an I/O device and send it to the CAN bus when it receives a remote request.

The A/D signal is encoded to ASCII values by a Form, called the A/D Form in the City's Transmit Document. This Form is similar to the A/D Form in City 1, except the messages are sent only when requested. The data is stored in Folder 2 of the City. Then, the City's Mayor sends the data along with the City's address on to the bus in response to a remote request.

The operation process of the City is shown in Figure 4-6.

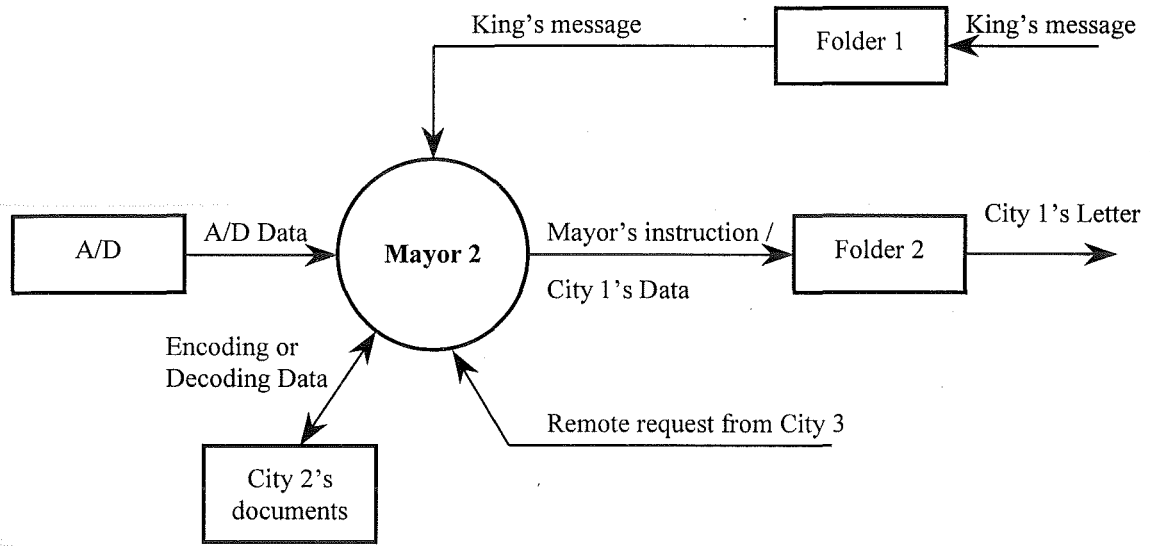


Figure 4-6 City 2's Operation Process

City 2 also contains two documents:

1. The King Document is the same as City 1's King Document as shown in Figure 4-4 except the City's address is 002.

2. The Transmit Document is shown in Figure 4-7.

City 2 does not have the Receive Document because it does not receive any data.

Form Type: Transmit Location: Folder 2 Remote Request: Yes Data Type: ASCII code (for all Lines)
Line Description Line 0: City's Address (Hundreds) Line 1: City's Address (Tens) Line 2: City's Address (Units) Line 3: ':' character Line 4: A/D Value (Hundreds) Line 5: A/D Value (Tens) Line 6: A/D Value (Units) Line 7: EOT (End of String character)

Figure 4-7 City 2's Transmit Document

Note that the description of this Form is the same as the Form for City 1 except the Remote Request is set to "Yes".

4.3.2.4 City 3

This City is designed to receive and display data in Liquid Crystal Display (LCD), typically, from City 1 and City 2.

Folder 2 and Folder 3 are used to receive a string of ASCII characters. The string length is no more than 8 bytes with the last byte being an EOT character.

The operation process of City 3 is shown in Figure 4-8.

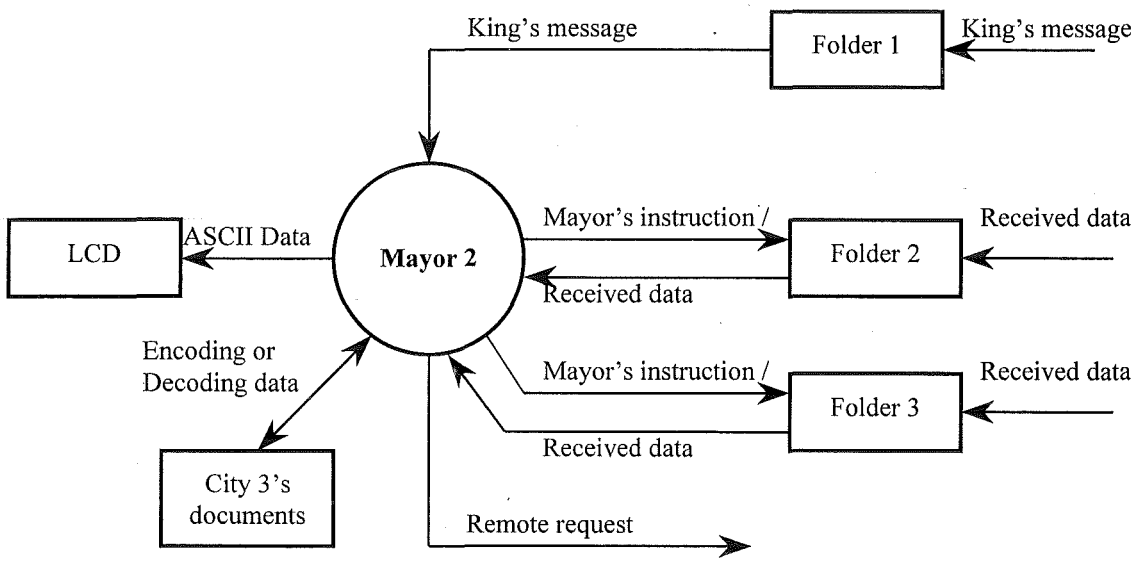


Figure 4-8 **City 3's Operation Process**

City 3 contains two documents:

1. The King Document is the same as City 1's and City 2's King Documents as shown in Figure 4-4. This means that City 3 can also receive all five King Pages. Note that the City's address is 003.

2. The Receive Document contains two Forms associated with Folder 2 and Folder 3 as shown in Figures 4-9 and 4-10.

Note that this City does not have a Transmit Document as it is used to receive data.

Form Type: Receive Location: Folder 2 Remote Request: No Data Type: ASCII code (for all Lines)
Line Description Maximum 8 Lines. Each Line is ASCII code. The last Line must be EOT.

Figure 4-9 **City 3's Receive Document** (Form for Folder 2)

The Form in Figure 4-9 records the following information:

- It is used to decode arriving messages.
- The messages are located in Folder 2.
- The Mayor does not generate Remote Request to receive the messages in this Folder.
- All Lines contain ASCII values at any range.
- The Folder 2 can receive a maximum of 8 bytes and the last byte must be EOT.

Form Type: Receive Location: Folder 3 Remote Request: Yes Data Type: ASCII code (for all Lines)
Line Description Maximum 8 Lines. Each Line is ASCII code. The last Line must be EOT.

Figure 4-10 **City 3's Receive Document** (Form for Folder 3)

The information contained in this Form is the same as for Folder 2, except:

- The location is Folder 3, and
- The Mayor sends remote request to receive messages for this Folder.

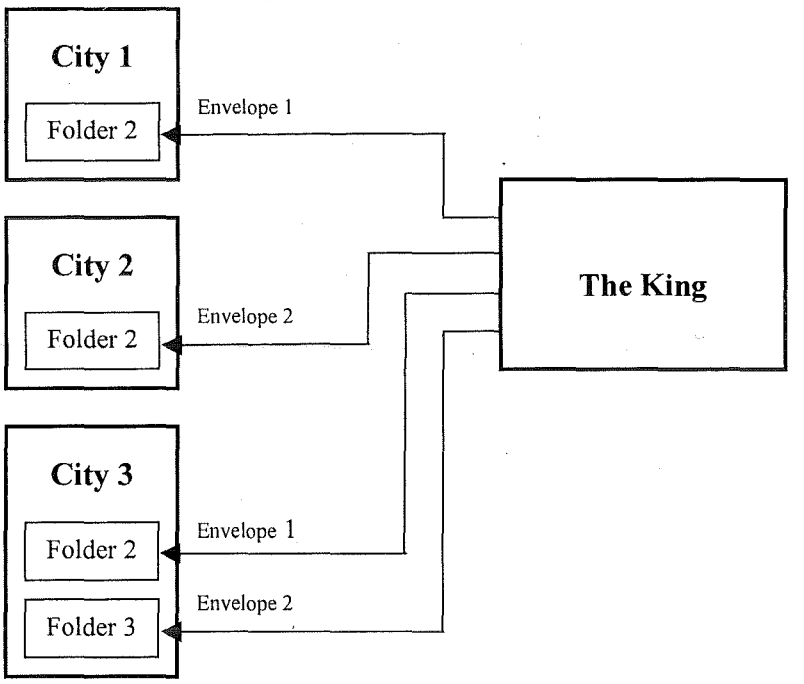


Figure 4-11 The Small CAN Kingdom system's Set-up phase

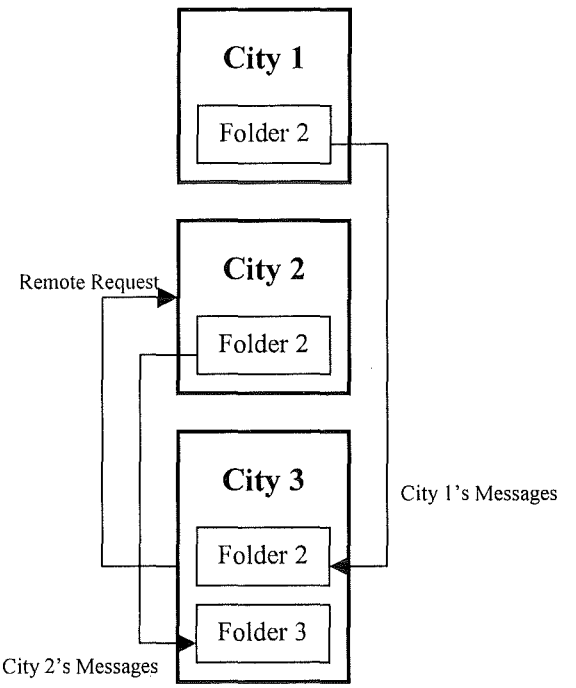


Figure 4-12 The Small CAN Kingdom system's Run phase

4.3.3.1 Set-up phase

During this phase, the King uses King Page 1 to assign the same Envelope (for example, Envelope 1) to City 1's Folder 2 and City 3's Folder 2, and to enable the two Folders.

The King also assigns a similar Envelope (for example, Envelope 2) to City 2's Folder 2 and City 3's Folder 3, and again enables the two Folders.

Optionally, the King may send other King Pages for further configuration such as to change the baud rate by King Page 4, or to change City address by King Page 2.

After completing the entire network configuration, the King sends King Page 0 to tell the Cities that the Set-up phase is finished and the Cities can begin their work.

It is noted that King Page 4 should be broadcast to all Cities by utilising the Group address 0. This enables all Cities to start their work simultaneously.

4.3.3.2 Run phase

In this phase, City 1 transmits its messages via its Folder 2 whenever it detects a changing value from the A/D device. City 3 receives the messages via its Folder 2. The Mayor 3 then uses the Form associated with this Folder to decode the messages and sends them to the LCD for display.

The Mayor 2 always updates its information and stores the information in Folder 2. The Mayor sends a CAN remote frames to City 2 in order to request the information. For example, a Remote frame can be generated by pressing a push button on City 3 (see Chapter 6). Consequently, the Mayor 2 sends out the message stored in Folder 2. Subsequently, City 3 receives this message via Folder 3 and displays the data in the LCD.

Examples of City 1's and City 2's messages displayed on City 3's LCD are shown in Figure 4-13 and 4-14, respectively.

```
* CAN SYSTEM DEMO *
CITY_001:128
```

Figure 4-13 City 1's message

The value 001 indicates the City 1's address. The value 128 is an example of an A/D value from City 1.

```
* CAN SYSTEM DEMO *
CITY_002:096
```

Figure 4-14 City 2's message

The value 002 indicates the City 2's address. The value 096 is an example of an A/D value from City 2.

As mentioned earlier, the King can be removed at the Run phase if no more system configuration is required. However, it can also send its instructions to the Cities during this phase. For example, the King can change the baud rate of the network, or change the Envelope for the messages, in order to change the message priority.

4.4 Conclusion

In conclusion, the CAN Higher Layer Protocol developed in this project, called the Small CAN Kingdom protocol, is based on the basic ideas of the CAN Kingdom protocol. The protocol provides an open approach which enables later designers to enhance their system's performance. For example, more King Pages can be added to the protocol easily to provide more services (see Chapter 7). However, some King Pages can be omitted if the system is restricted in hardware configuration such as the amount of memory, or if the system does not require complex configurations.

Furthermore, the simplicity of the Small CAN Kingdom protocol makes it easier to understand. The programming code of this protocol has been reduced to fit into the small amount of on-chip memory of a small CAN-based system (see Chapter 7).

The details of hardware and software designs for a Small CAN Kingdom system are provided in later chapters.

CHAPTER 5

INTRODUCTION TO MICROCONTROLLERS AND CAN CONTROLLER CHIPS

The aim of this chapter is to introduce the physical components used to design the two main parts of a Small CAN Kingdom system's node: The node manager (the Mayor or the King) which controls the node operations, and the CAN controller which is responsible for the node communication.

5.1 Microcontrollers

5.1.1 Overview

The responsibilities of the node managers are to control the operation of the nodes; thus, they should be "smart" enough to perform this role.

One of the most efficient ways to add "intelligence" to devices today is to use microcontrollers. Since their birth in the early 1970s, microcontrollers have become increasingly popular in domestic and industrial applications. Most machines and appliances in our daily lives contain microcontrollers, such as cordless and portable telephones, security systems, automobiles and gas pumps, automated teller machines, and much more. "In fact, electric lights are almost the only electrically powered devices that do not use microcontrollers, and even here things are changing with the welcome advent of power-saving and quick-starting intelligent ballast fluorescent lamps" (Khan, 1996). In addition, the architecture of microcontrollers ideally supports real-time applications which are the usual requirements in automation industries (Wetton, 1995, p. 24). Microcontrollers are embedded in various parts of manufacturing systems in order to control them.

A microcontroller contains many peripherals on a single chip such as Analog/Digital (A/D) converters, pulsewidth modulation channels, on-chip memory, and so on. This inclusion of I/O circuits and memory enables designers to develop various applications using single chip computers (microcontrollers). As a result, the costs of system designs are significantly reduced.

The main aim of designing the Small CAN Kingdom protocol in this thesis is to provide a CAN Higher Layer Protocol that can be utilised by small distributed systems. These systems consist of microcontrollers with hardware restrictions such as memory and the size of devices. Two such popular microcontrollers currently in use are the MC68HC11 from Motorola, and the 8051 from Intel.

Although any microcontroller could have been used to develop the system in this project, the MC68HC11 microcontroller appeared to be a suitable choice because it is one of the most versatile 8-bit microcontroller and is still used in a wide range of control applications (Chen, Rabb, & Taylor, 1996; Voskamp & Rosenstiel, 1996; Maskell & Grabau, 1998).

According to the HC11 M68HC11 E Series Technical Data (1993) manual, the MC68HC11 microcontroller contains a large number of features in a single chip which include:

- Single-chip or Expanded multiplex operation modes
- Eight channels 8-bit Analog-to-Digital (A/D) converter
- A fast Serial Peripheral Interface which allows the MCU to communicate with an IBM PC or with other microcontrollers utilising a master-slave connection
- Four 8-bit I/O ports with varying capabilities
- Multiplexed 16-bit address and data bus in Expanded mode

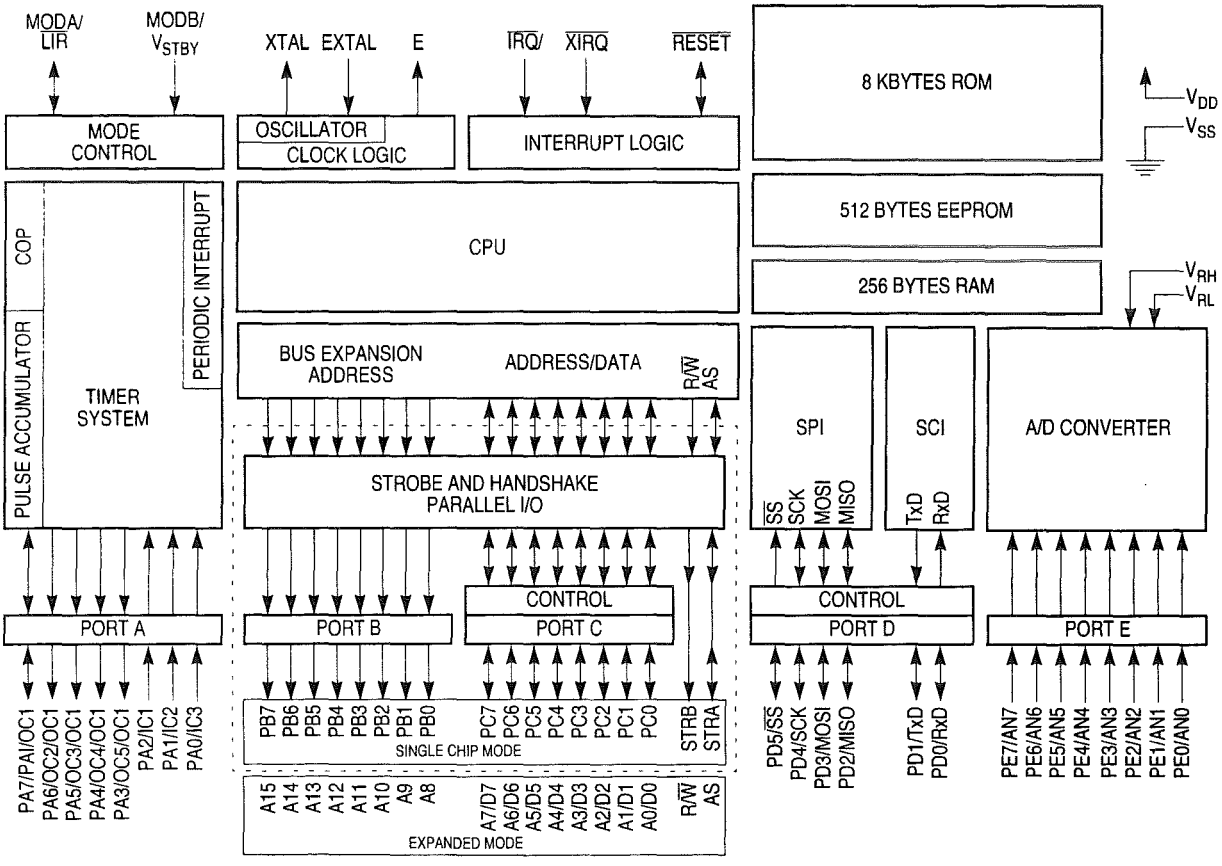
- Internal EEPROM and RAM
- Serial Communication Interface (SCI). A dedicated RS232C serial port with 8-bit data word transmission capability
- Multiple counter/timers with comparison functions and wave-form generation capability
- Sophisticated interrupt handling structure including a real-time interrupt mechanism
- 16-bit operations, including multiplication and division

The MC68HC11 is also supplied with a ROM-base monitor program, which contains various functions for application developments.

Another reason for choosing the MC68HC11 microcontroller is that Motorola, in Perth, Western Australia, provides excellent hardware evaluation equipment and software support for this series of products. Moreover, Edith Cowan University has several evaluation boards readily available, and also staff with expertise in HC11 technology. Furthermore, there are numerous documents related to this type of microcontroller. Hence, programmers do not have to get involved with designing the basic microcontroller units. In the case of this project, therefore, more effort can be allocated to the Higher Layer Protocol level of design.

5.1.2 MC68HC11 Block Diagram

According to Greenfield (1992, p. 61), a MC68HC11 microcontroller comes as a single Integrated Circuit (IC) packaged in either a 52-pin Plastic Leadless Chip Carrier (PLCC) or a 48-pin Dual In-line Package (DIP). The main features and the architecture of the microcontroller are illustrated in Figure 5-1.



CIRCUITRY ENCLOSED BY DOTTED LINE IS EQUIVALENT TO MC68HC24.

Figure 5-1 **MC68HC11 Block Diagram**

5.1.3 System development environment

To assist in designing and debugging a microcontroller system based on the MC68HC11, Motorola provides users with three types of evaluation boards which have the following charateristics:

- **M68HC11EVB** – Evaluation Board, provides a small, low-cost tool for debugging and evaluating MC68HC11 based systems. The main disadvantage of this board is that the users can only utilise Single-chip mode operation. External components are required to make use of the Expanded mode.

- **M68HC11EVBU** – Universal Evaluation Board, is a low-cost development tool. The board provides a wire-wrap area for custom interfacing. It supports both Single-chip and Expanded-multiplexed mode operations without additional circuitry.

- **M68HC11EVM** – Evaluation Module, provides users with more powerful and flexible tools for MC68HC11 based system development. The board simplifies user evaluation of prototype hardware/software products by providing timing and I/O circuitry. Single-chip and Expanded-multiplexed mode operations are supported. Pseudo ROM space is also provided on board.

In this project, the M68HC11EVBU evaluation boards were used in order to take advantage of Motorola's support to design the hardware part. Furthermore, because this type of evaluation board already supports Expanded mode operation, it is well suited for designing the interface between a HC11 MCU and a CAN controller chip (see Chapter 6). There are also a number of EVBU boards available at Edith Cowan University with software support development environment.

The following section describes the main features of the M68HC11EVBU evaluation board and its use in designing hardware interface.

5.1.3.1 Hardware design environment

The M68HC11EVBU evaluation board comes complete with good support tools for system development.

This type of evaluation board can be connected to a host personal computer (PC) via an RS232C terminal I/O port. Communication between the board and the host PC is controlled by a communication program such as Kermit or Procomm program for an IBM-PC, and Red Ryder program for Apple Macintosh (M68HC11EVBU Universal Evaluation Board User's Manual, 1992, p. 4/37). This connection, along with the BUFFALO monitor program, can be used to design the user interface between the King in the Small CAN Kingdom system and the system designer (see Chapter 7).

The BUFFALO, a ROM based monitor program inside the microcontroller, provides great support in assisting users to debug their programming codes easily. The program also contains a set of I/O and utility routines which can be used to develop applications. User codes can be assembled either by using the line assembler on the monitor program or by an assembler on the host PC, and then downloaded to the EVBU user RAM or EEPROM via Motorola S-records (M68HC11EVBU Universal Evaluation Board User's Manual,1992, p. 3/1).

In addition, a wire-wrap area is provided on the EVBU for MCU custom interfacing. With the wire-wrap hole pattern provided, most standard DIP or PLCC device wire-wrap sockets, strip sockets, headers, and connectors can be installed. Wire-wrap components can be installed on the top-side, and the wire wrapping can be performed on the bottom-side of the EVBU (M68HC11EVBU Universal Evaluation Board User's Manual,1992, p. 2/18). This area is used to design the external peripherals for a City in the Small CAN Kingdom system such as the CAN controller chip, LCD, and A/D devices (see Chapter 6).

5.1.3.2 Software design environment

In order to design complete applications, users should be able to write their source software with a suitable programming language. The source code is then compiled and translated to Motorola S-records. Finally, it is downloaded to the target microcontroller board for execution.

During the period of this research, the readily available programming languages for MC68HC11 were Assembly and C. In addition, one of the main requirements of developing the Small CAN Kingdom protocol is that the protocol should fit into the small amount of on-chip memory of the microcontroller. Hence, Assembly language was chosen because it was suitable for small control programs. A further reason for choosing the Assembly language is that the programming code could be easily debugged by the use of the BUFFALO monitor program.

In this project, an IBM PC was used to communicate with the M68HC11EVBU boards during both designing and setting up the system. Besides using the BUFFALO program, there were also other requirements for the development of the system as follows:

- A text editor to write the source programming codes
- An assembler to compile the program
- A program to create Motorola S-records
- A serial communication program

After considering the MC68HC11 features, the Motorola portable assembler (PASM) was the preferred language as it provided sufficient software design capabilities for a MC68HC11 based system. The programming codes were written by using an MS-DOS text editor (Edit program). The Ubuilds program was used to create S-records, and the MS-Kermit program was used for communication between the IBM PC and the evaluation boards.

In addition, there are four MS-DOS batch files created by the author of this thesis to provide a more efficient software environment:

1. K.BAT to invoke the MS-Kermit program
2. ASM.BAT to compile the programming codes
3. B.BAT to build the Motorola S-records
4. T.BAT to transmit an S-record file to the M68HC11EVBU board via a RS232C serial cable.

The description of these batch files is given in Appendix A.

The system development environment is shown in Figure 5-2.

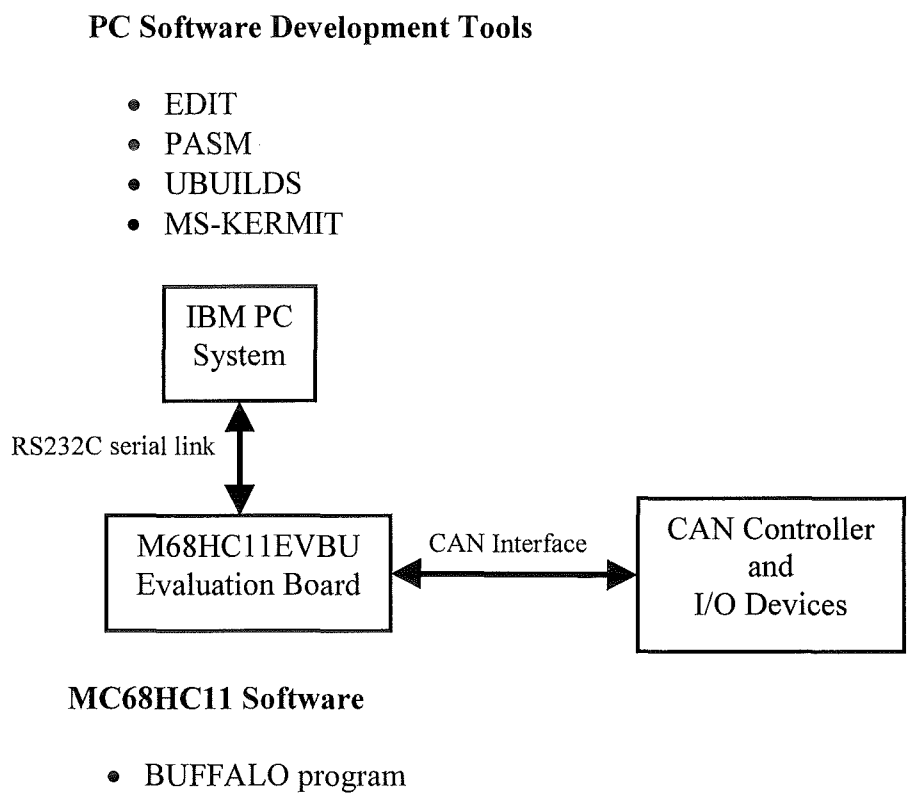


Figure 5-2 System development environment

5.2 CAN controllers

5.2.1 Choosing CAN controllers

As mentioned previously, CAN controller chips are used to manage communication between a CAN node and the CAN bus. In other words, the chips are the means for the Cities' managers (Mayors) to communicate with each other and with the King.

A CAN controller chip can be Basic CAN, Full CAN, CAN+, or can be "Part B Passive" or "Part B Active" chip (see Chapter 2). With respect to the aims of this project, the Full CAN controller chips were recommended to be used to reduce the workload for the 8-bit microcontrollers with a limited amount of on-chip memory.

Moreover, the Small CAN Kingdom protocol has been developed based on the CAN Kingdom protocol, which utilises both Stand and Extended CAN frame formats. Therefore, the appropriate CAN controller chips are "Part B Active" which support both of the message formats.

Considering the aforementioned discussions, the Intel 82527 CAN controller chips appeared to be a suitable choice. This type of chip was considered better than the similar 82C200 CAN controller from Philips with respect to real-time performance (Tindell, et al., 1994). In addition, the SAB-81C90 and SAB-81C91 from Siemens, support Standard frames but tolerate Extended frames without generating errors ("Part B Passive" chips). On the other hand, the Intel 82527 supports both Standard and Extended CAN frame formats ("Part B Active" chip).

5.2.2 Intel 82527 CAN controller

According to the 82527 serial communications controller architecture overview (1996), the Intel 82527 has a number of features that are ideally suited to be used to interface with a microcontroller to form a complete CAN node. The main features of the Intel 82527 chip are as follows:

- Support Standard and Extended CAN frames
- 15 Message Objects of 8-byte data length to store incoming and outgoing messages
- Flexible CPU Interface
  - 8-bit Multiplexed
  - 16-bit Multiplexed
  - 8-bit Synchronous Non-Multiplexed
  - 8-bit Asynchronous Non-Multiplexed
- Serial interface to CPU is available, when a parallel CPU interface is not required
- Flexible interrupt structure, including Message Object interrupts and the chip's status interrupts
- Two 8-bit bidirectional I/O ports can be used as general I/O ports. This is an advantage of the Intel 82527 chip when the microcontroller ports are not available.

The architecture of the chip can be demonstrated by its block diagram as shown in Figure 5-3.

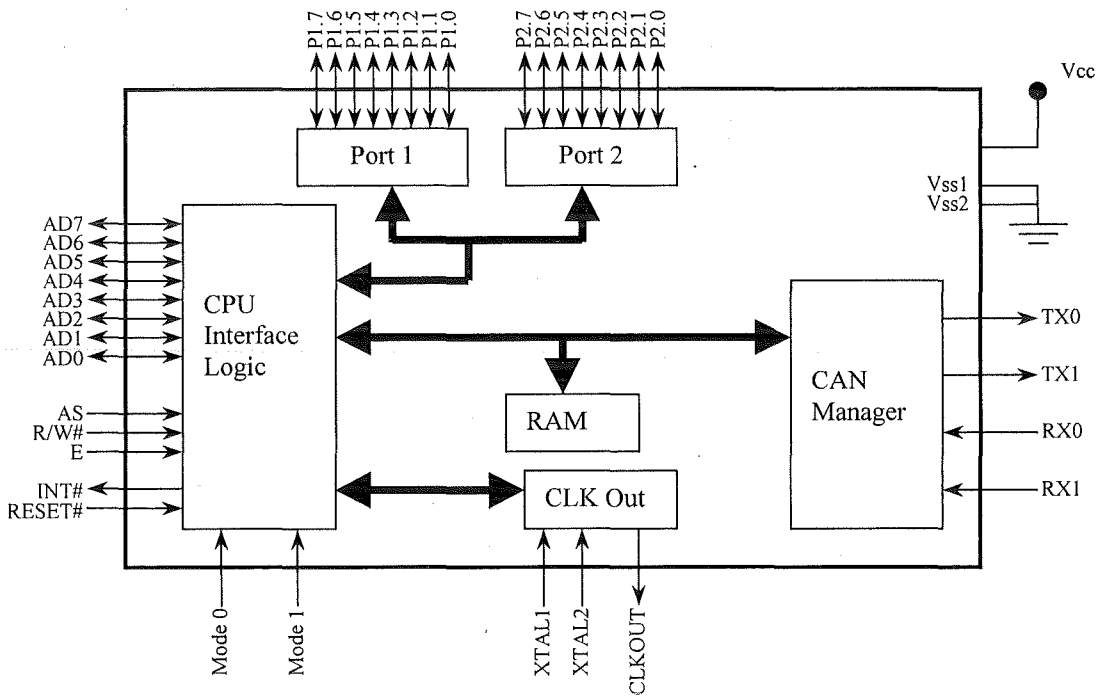


Figure 5-3 The 82527 block diagram

The descriptions of the diagram as follows:

- The **CPU Interface Logic** is responsible for the interface between the host CPU and the CAN controller. The interface mode depends on the value of Mode 0 and Mode 1 pins.
- The **CAN manager** controls the data stream between the RAM (parallel data) and the CAN bus.
- The **RAM** contains the set of registers which the CPU can be used to control the chip.
- **Clockout** is the on-chip clock generator consisting of an oscillator, clock divider register and a driver circuit.
- **Port 1** and **Port 2** can be used as general-purpose 8-bit I/O ports.

The CPU controls the CAN controller chip via a set of registers which can be mapped onto the CPU's memory map (see Chapter 6). The 82527 address map is shown in Figure 5-4. The details of the uses of these registers to control communication in a CAN based system are given in Chapter 7.

00H	Control Register
01H	Status Register
02H	CPU Interface Register
03H	Reserved
04-05H	High Speed Read Register
06-07H	Global Mask – Standard
08-0BH	Global Mask - Extended
0C-0FH	Message 15 Mask
10-1EH	Message 1
1FH	CLKOUT Register
20-2EH	Message 2
2FH	Bus Configuration Register
30-3EH	Message 3
3FH	Bit Timing Register 0
40-4EH	Message 4
4FH	Bit Timing Register 1
50-5EH	Message 5
5FH	Interrupt Register
60-6EH	Message 6
6FH	Reserved
70-7EH	Message 7
7FH	Reserved
80-8EH	Message 8
8FH	Reserved
80-9EH	Message 9
9FH	P1CONF
A0-AEH	Message 10
AFH	P2CONF
B0-BEH	Message 11
BFH	P1IN
C0-CEH	Message 12
CFH	P2IN
D0-DEH	Message 13
DFH	P1OUT
E0-EEH	Message 14
EFH	P2OUT
F0-FEH	Message 15
FFH	SPI Reset Address

Figure 5-4 Intel 82527 address map

5.3 Conclusion

This chapter introduced the physical components which are used to design the two main parts of a CAN node: the node manager (the Mayor or the King), and the CAN controller.

The discussion of selecting suitable devices for the project was presented. The MC68HC11 microcontrollers have been used as the host CPU to control the node operations. The Intel 82527 CAN controller chips manage the communication of the nodes.

In addition, the hardware and software environments for system development were also described in this chapter. The actual design is covered in the following chapters.

CHAPTER 6

HARDWARE DESIGN

This chapter describes the steps associated with designing the hardware part of the Small CAN Kingdom system which was introduced in Chapter 4. The pin connection tables between the chips, and the diagrams showing the pin layout of these components are shown in Appendix B. These tables and diagrams were used for wire-wrapping purpose in the hardware design.

6.1 Introduction

The aim of the system in this project is to build a platform which enables the Small CAN Kingdom protocol to be designed and tested. The system consists of:

- A master node (called the **King** or the **Capital**) which is responsible for network configurations,
- **City 1** transmits Analog/Digital (A/D) information,
- **City 2** transmits A/D information in response to a remote request (a CAN Remote frame) from City 3, and
- **City 3** displays the data from City 1 and City 2. It is also able to send Remote frames to City 2 when the data is needed.

Each node in the system contains an MC68HC11 microcontroller (MCU), an Intel 82527 CAN controller, and a Transceiver chip:

- The MC68HC11 controls the node's operations.
- The Intel 82527 performs the communication according to the CAN protocol.
- The Transceiver chip acts as the interface between the Intel 82527 CAN controller and the CAN bus.

In addition, each node includes peripherals to carry out its tasks:

- An IBM PC is connected to the Capital node via a RS232C serial link in order for system designers to enter King Pages (see Chapter 4).
- A/D devices are utilised in City 1 and City 2.
- A Liquid Crystal Display (LCD) and a remote request device are included in City 3.
- Indicators (LEDs) to indicate which phase (Set-up or Run phase) the City is in.

The block diagram of the whole system is shown in Figure 6-1.

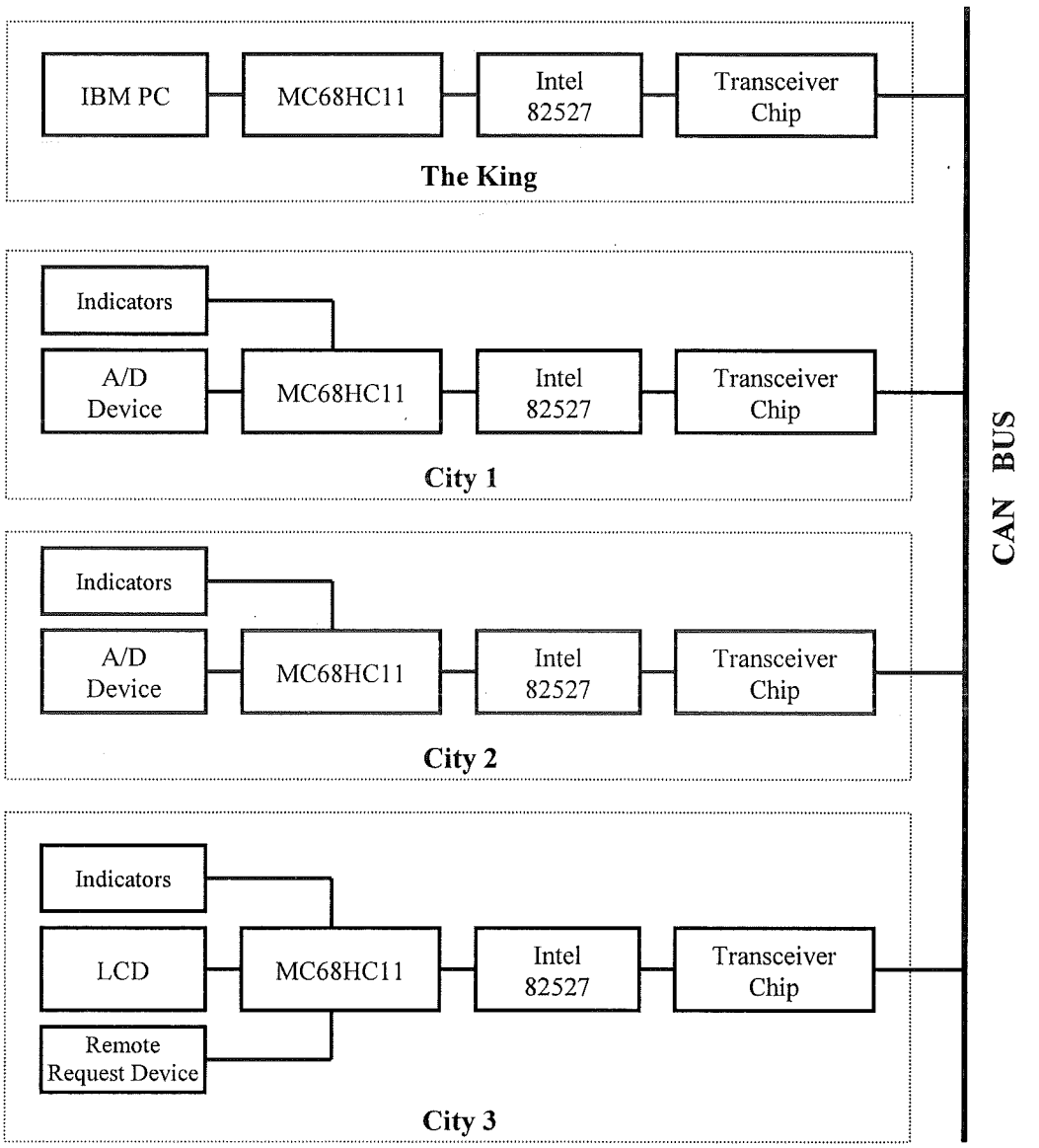


Figure 6-1 **The Small CAN Kingdom system block diagram**

6.2 Interfacing the Intel 82527 to an MC68HC11

The following highlights the important issues of the interface design between a Motorola MC68HC11 microcontroller and an Intel 82527 CAN controller chip (Figure 6-2). Four of these interfaces were later constructed, tested and used in this project.

In order to interface the Intel 82527 with the MC68HC11, the latter should be operating in the Expanded mode. In this mode, Port B pins of the MC68HC11 are used as the high order address output signals. Port C pins serve as the low order address and 8-bit data bus (multiplexed using the same pins). The MC68HC11 signals used to control the Intel 82527 CAN controller are via AS, R/W# and E pins. AS pin is used to control an external address latch. R/W# pin is used to indicate the direction of data transfer. E pin is used as timing reference when the MC68HC11 communicates with the Intel 82527 (HC11 M68HC11 E Series Technical Data, 1993).

Because the MC68HC11 is a non-Intel MCU, the Intel 82527 should be used in mode 2 (8-bit multiplexed, non-Intel). This CAN controller chip also has AS, R/W# and E pins that match the corresponding pins of the MC68HC11 (82527 Serial Communications Controller Architecture Overview, 1996).

Data transfer between the MC68HC11 and the Intel 82527 was made in 8-bit parallel form using the Intel 82527 pins AD0 to AD7, which were connected, respectively, to the MC68HC11's pins PC0 to PC7 of Port C.

The INT# pin of the Intel 82527 was used to generate an interrupt to the MCU when the CAN controller chip receives or successfully transmits a message. This pin was connected to pin IRQ# of the MC68HC11. This connection required an external pull-up resistor (10kΩ).

4.3.3 System operation

Initially, when the nodes are connected together on the CAN network, they all work at a fixed baud rate (for example, 125 kbit/sec). All the Cities receive the King commands via Folder 1 with the highest priority Envelope (Envelope 0).

The King is responsible for the network configuration. In doing so, the system designer studies the Documents of all the Cities. He or she should know that City 1 and City 2 transmit their messages with the data types in ASCII format. Each message of both Cities contains 8 bytes and the last byte is EOT character. This satisfies the requirements of the Forms in City 3. Hence, City 3 can receive messages from both City 1 and City 2.

Furthermore, the Form associated with City 3's Folder 3 informs that the Mayor of this City will send a remote request (Remote frame) in order to receive messages for Folder 3. Therefore, this Folder can be used to receive messages from City 2 as it only sends its messages when receiving a Remote Request.

As a result, the system designer can assign Folder 2 of City 3 to receive messages from City 1; and Folder 3 is used to receive messages from City 2. The assignments are carried out in the Set-up phase.

The Set-up phase and the Run phase operations are illustrated in Figures 4-11 and 4-12.

The RESET# pin of Intel 82527 can be tied to an MC68HC11 Port pin or a reset circuit. In this project, the MC68HC11's pin PA6 of Port A was used to reset the Intel 82527.

The CS# pin of the Intel 82527 enables the MCU to select the chip. This pin was connected to the MCU via an Address Decoder circuit.

In addition, pins XTAL1 and XTAL2 of the Intel 82527 were connected to a Quartz crystal circuit in order to provide the clock signals for the operations of the CAN controller chip. The chip's operations are controlled by two internal clocks:

- The system clock (SCLK), and
- The memory clock (MCLK).

The frequency of the Quartz crystal is called XTAL. The frequency of the SCLK can be equal to XTAL or XTAL/2 by programming. With the 8 MHz SCLK, the Intel 82527 can be interfaced to a 1Mbit/sec CAN bus.

The frequency of the MCLK can be equal to SCLK or SCLK/2 by programming. The maximum frequency of the MCLK is 8 MHz.

According to the Intel 82527 architecture overview, the chip was tested with XTAL setting to 8 MHz and 16 MHz. This project used 16MHz Quartz crystals due to their availability. Therefore, the Intel 82527 chip should be set: SCLK=XTAL/2 and MCLK=SCLK.

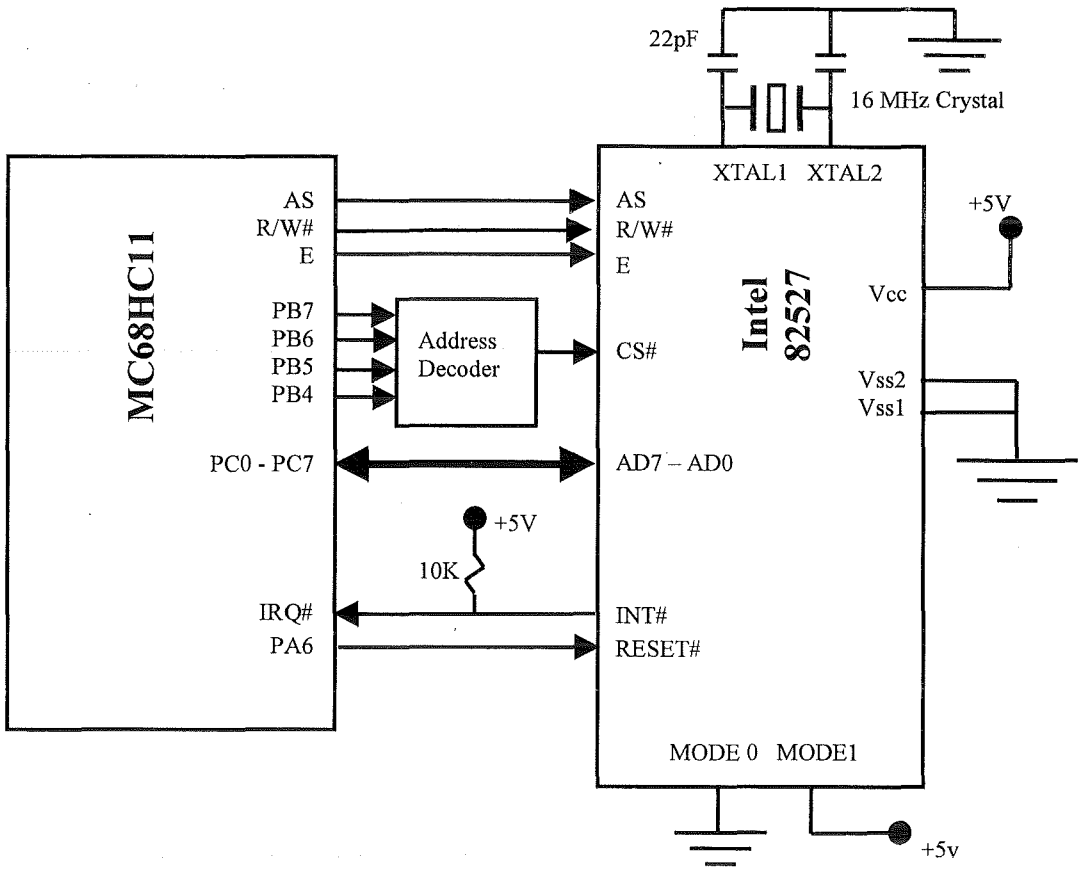


Figure 6-2 **MC68HC11 and Intel 82527 interface circuit diagram**

Designing the Address Decoder

The Address Decoder was used to map the Intel 82527 into the MC68HC11's memory map. It can be designed by using either a conventional 4-input AND-gate chip (74LS20) or an address decoder chip (74LS138). In this project, both circuit designs were used.

During the early stage of the project, the 4-input AND-gate chips were used for address decoding in City 1 and City 2 of the system. Therefore, in order to minimise the number of chips, the address range 7000H to 7FFFH of the MC68HC11 was used to map the internal Intel 82527's RAM into the MCU addressable space.

The MC68HC11 system memory map is shown in Figure 6-3.

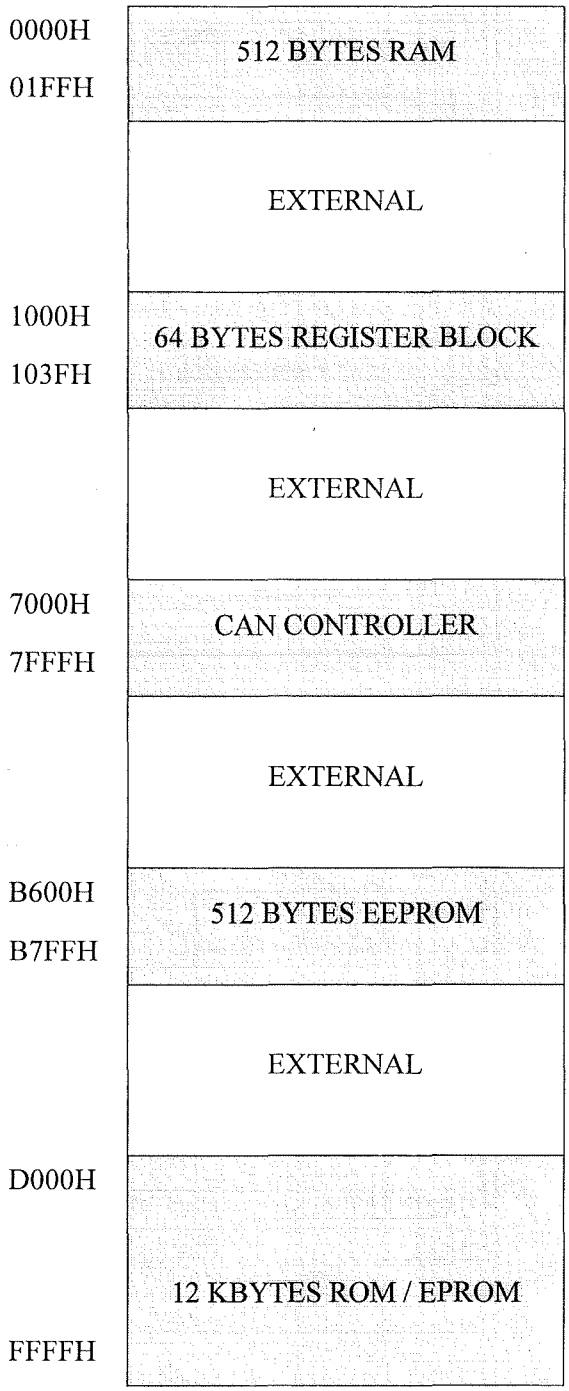


Figure 6-3 MC68HC11 memory map

The address decoder circuit design is as follows:

7			
A15	A14	A13	A12
0	1	1	1
w	x	y	z

The Chip Select (CS#) is active-low when the address bus has:

w'. x . y . z

The logic circuit is as shown in Figure 6-4:

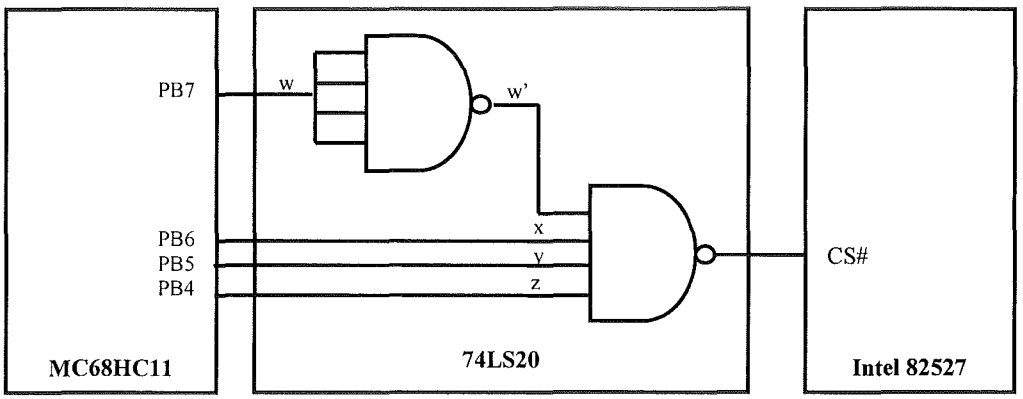


Figure 6-4 **Address Decoder circuit 1**

In order to add external memory to the MC68HC11 systems for subsequent applications, address decoder chips (74LS138) were used in the Capital and City 3 of the system. In the design of this interface circuit, pin Y3 of the 74LS138 chip was used to select the Intel 82527 so that the software controlling all CAN nodes could utilise the same address locations shown in Figure 6-3.

The alternative address decoder circuit design is shown in Figure 6-5.

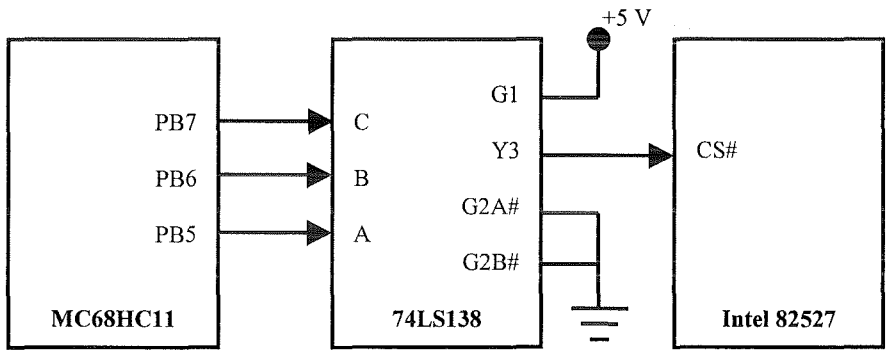


Figure 6-5 Address Decoder circuit 2

Note that when using the 74LS138 chip for address decoding, the address range for Intel 82527 in MC68HC11 is 6000H to 7FFFH.

6.3 Interfacing the Intel 82527 to a transceiver chip

6.3.1 CAN bus review and introduction to CAN transceiver chips

The CAN bus uses Non-Return to Zero (NRZ) with bit-stuffing. There are two different signalling states: dominant (logical 0) and recessive (logical 1). These correspond to the electrical levels utilised by the physical layer. The CAN modules are connected to the bus in a Wired-And fashion: if only one node is driving the bus to the dominant state, then the whole bus is in that state regardless of the number of nodes transmitting recessive bits (see Chapter 2).

A typical CAN bus consists of two wires, called CAN high (CAN_H) and CAN low (CAN_L). When the CAN bus is in dominant state, CAN_H is at a high voltage (5V) and CAN_L is at a low voltage (0V). When the CAN bus is in recessive state, both CAN_H and CAN_L are floating. This characteristic of the CAN bus means that whenever a CAN node transmits a dominant bit, the bus will be in dominant state (CAN_H is high voltage and CAN_L is low voltage).

The purpose of the Transceiver chip is to provide an interface between the Intel 82527 and the CAN bus. There are several Transceiver chips manufactured by companies such as Philips, Bosch, Siemens, Siliconix and Unitrode. The most common chip used in the automation industry is the 82C250 transceiver from Philips, which implements the physical layer for CAN-based systems.

However, due to the time restrictions and the difficulty of obtaining these chips in Perth, Western Australia, the conventional RS485 standard chips (typically, DS3695 chips) with some modifications were used and deemed to be satisfactory for this project.

The features of both PCA82C250 and DS3695 transceivers were reviewed and compared in order to modify the DS3695 circuitry to suit the requirements of the CAN bus.

6.3.2 PCA82C250 CAN transceiver

The pin connections of the PCA82C250 are shown in Figure 6-6:

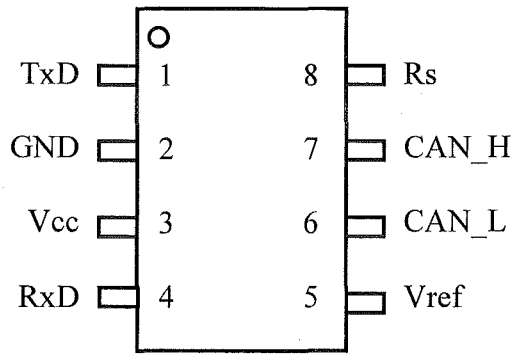


Figure 6-6 PCA82C250 CAN tranceiver

The description of the chip pins are in Table 6-1.

Table 6-1 PCA82C250 pin description

Symbol	Pin	Description
TxD	1	Transmit data
GND	2	Ground
Vcc	3	Supply voltage
RxD	4	Receive data
Vref	5	Reference voltage output
CAN_H	6	LOW level CAN voltage input / output
CAN_L	7	HIGH level CAN voltage input / output
Rs	8	slope resistor input

The truth table of the PCA82C250 is provided in Table 6-2.

Table 6-2 PCA82C250 truth table

TxD	CAN_H	CAN_L	Bus State	RxD
0	High	Low	dominant	0
1 (or floating)	floating	floating	recessive	1

6.3.3 DS3695 transceiver

The pin connections of the DS3695 are shown in Figure 6-7:

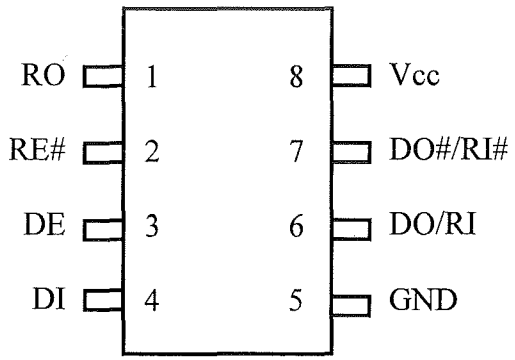


Figure 6-7 **DS3695 (RS485) tranceiver**

The description of the chip pins are in Table 6-3.

Table 6-3 DS3695 pin description

Symbol	Pin	Description
RO	1	Receive output
RE#	2	Receive enable
DE	3	Data enable
DI	4	Data input
GND	5	Ground
DO / RI	6	Data output / Receive input
DO# / RI#	7	Reversed Data ouput / Reversed Receive input
Vcc	8	Supply voltage

The truth table of the DS3695 is provided in Table 6-4.

Table 6-4 DS3695 truth table

Transmitting

Inputs			Outputs	
RE#	DE	DI	DO#	DO
X	1	1	0	1
X	1	0	1	0
X	0	X	High Z	High Z

Receiving

Inputs			Output
RE#	DE	RI - RI#	RO
0	0	≥ +0.2V	1
0	0	≤ - 0.2V	0
1	0	X	High Z

6.3.4 Modifying the DS3695

In this project, the Intel 82527 was configured to transmit data via TX0 and to receive data via RX0.

Comparing the truth tables of PCA82C250 and DS3695, it is noted that if the CAN bus is in recessive state (logical 1), then the CAN_H and CAN_L pins of the PCA82C250 chip are floating. This corresponds to High Z state of DO#/RI# and DO/RI pins of the DS3695.

Note that DS3695 transceivers transmit data via pin DI. According to the consideration above, if pin DI is at logical 1 (recessive), the DO# and DO pins should be in High Z state. From the DS3695 truth table, this corresponds to DI=X ("don't care" condition) and DE=0.

Also, in the DS3695 truth table, it can be seen that, if DI=0 (dominant state) and DE=1, then DO#/RI# = 1 and DO/RO = 0. This situation is the same as CAN_H and CAN_L pins of the PCA82C250 being in dominant state, respectively. Hence, DO#/RI# pin can be used as CAN_H, and DO/RI pin can be used as CAN_L.

From the discussions above, TX0 pin of the 82527 chip can be connected to DI pin of the DS3695. Moreover, the TX1 pin of the 82527 is always the reverse state of TX0 (if TX1 is enable). Thus, TX1 can be connected to DE. This ensures that when DI is at logic 0, DE is at logic 1, and vice versa.

It should also be noted that, the 82527 chip monitors the bus when sending messages. Hence, the RE# pin of the DS3695 should always be enabled. In other words, this pin should always be connected to ground.

During the reception of a message, the TX0 and TX1 pins are always in recessive state. This corresponds to TX0 = 1 and TX1 = 0. Thus, DE = 0.

The DS3695 receiving truth table shows that if the CAN bus is in dominant state (DO#/RI# = 1, DO/RI=0) or RI - RI# $\leq$ -0.2 V, then RO = 0 (dominant state).

However, if the CAN bus is in recessive state (DO#/RI# and DO/RI pins are in High Z state), the RO pin could be undefined. According to the CAN protocol, this pin should be at logic 1. Therefore, RO pin was connected to Vcc (logic 1) via a pull-up resistor (10kΩ).

The modified circuit diagram is shown in Figure 6-8.

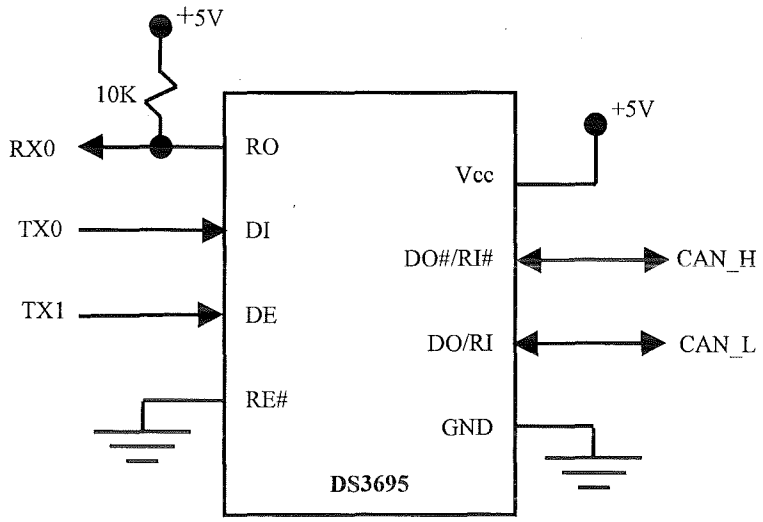


Figure 6-8 **Modified DS3695 circuit diagram**

The truth table for the modified circuit above is shown in Table 6-5.

Table 6-5 Truth table of the modified DS3695 transceiver

DI	DE	RE#	DO#/RI#	DO/RI	RO
1	0	0	High Z	High Z	1
0	1	0	1	1	0

6.3.5 Intel 82527-DS3695 interface circuit diagram

After being modified, the DS3695 chip satisfied the requirements of the CAN bus. The chip were interfaced with an Intel 82527 CAN controller chip according to the circuit diagram shown in Figure 6-9.

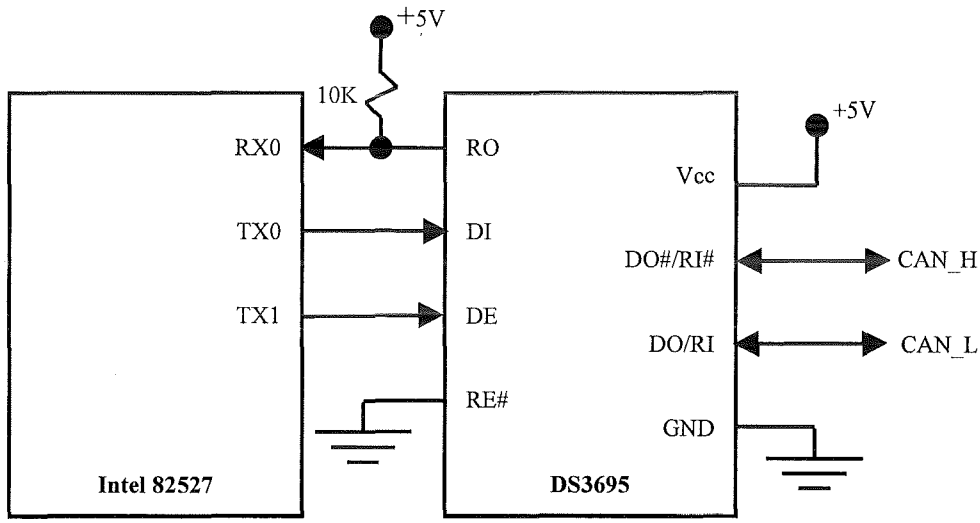


Figure 6-9 **Intel 82527-DS3695 interface circuit diagram**

6.4 Interface between the King and an IBM PC

As mentioned in Chapter 5, M68HC11EVBU evaluation boards were used to design the CAN nodes in this project. To enable system designers to communicate with the King during setting up the system, an IBM PC was connected to the King via RS232C terminal I/O port of the M68HC11EVBU. The connection has already been discussed in Chapter 5.

The system designers enter values for a King Page via a King Menu program which is designed in Chapter 7.

Figure 6-10 shows the connection diagram between the King and an IBM PC.

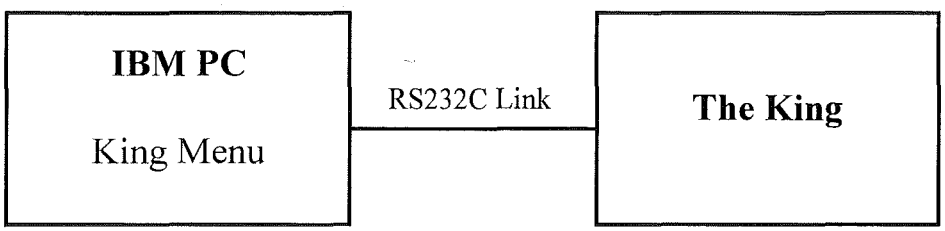


Figure 6-10 Interface between the King and an IBM PC

6.5 Interfacing A/D devices to MC68HC11

The Port E pins of the MC68HC11 can be used as inputs for A/D signals. A/D signals can be generated from any devices such as temperature or speed measurement sensors.

In this project, 10 KΩ rotary potentiometers were used to produce A/D signals. The signals were input to pin PE.4 of port E. The City's software application has been designed to receive and decode A/D signals (see Chapter 7).

It should be noted that VRL and VRH pins are used to provide reference voltage for the A/D converter circuitry. In the case of this project, VRL was connected to 0V and VRH was connected to 5V.

Figure 6-11 shows the circuit diagram of the interface.

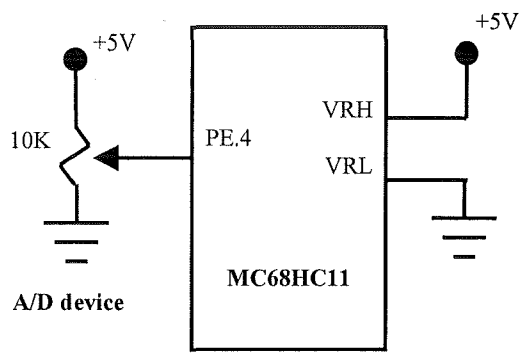


Figure 6-11 **A/D device – MC68HC11 interface**

6.6 Interfacing the LCD to Intel 82527

The Intel 82527 has two general purpose I/O ports: Port 1 and Port 2, which can be used to interface with external devices (see Chapter 5). Because MC68HC11 port pins can be used for various purposes, they may be reserved for future use. City 3 of this project used Port 1 and Port 2 of the Intel 82527 to control an LCD module.

In this project, the L2012 LCD was used. It is a low-power-consumption dot matrix LCD module with high contrast, wide viewing angle LCD panel with a CMOS LCD drive controller built-in. The controller has built-in character generator ROM/RAM, and display data RAM. All the display functions can be controlled by the LCD's instructions. The module can easily be interfaced with an MCU.

It should be noted that as the L2012 user manual was not readily available, the LCD design in this project referred to the Liquid Crystal Display module L4042 user manual (1988). All the pins and functions of these two types of LCD are the same except L2012 contains 20 characters on each line (the L4042 contains 40 characters on each line).

The control of LCD is maintained by three pins:

- E: the signal activates data write or read.
- R/W# is used to select Read or Write mode.
- RS: Register selection mode. This pin combined with the R/W# pin can instruct the LCD to perform certain functions (see the Liquid Crystal Display module L4042 user manual, 1988, p. 13-14).

Pins DB0 to DB7 of the LCD enable the MCU to output data a-byte-at-a-time to the LCD.

The two general I/O ports of the Intel 82527 (Port 1 and Port 2) can be used as additional port of the MCU:

- Port 1 was used to output data to the LCD.
- Pins P2.4 to P2.6 of Port 2 were used to control the LCD. They were connected to E, RS, R/W# of the LCD, respectively.

Pin VLC is used to adjust the contrast of the LCD.

Figure 6-12 shows the circuit diagram of the interface between the Intel 82527 and the LCD. The program to control the LCD is designed in Chapter 7.

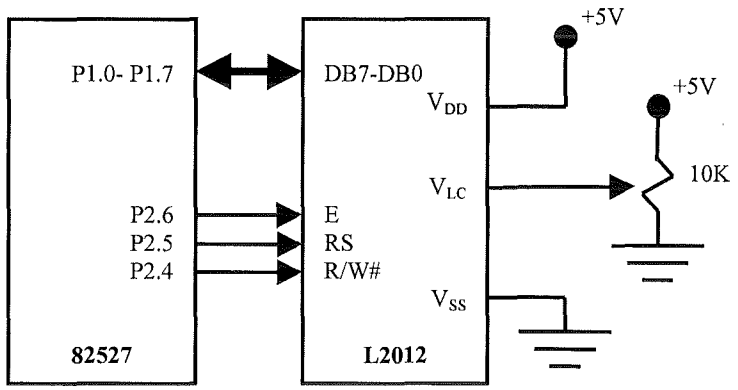


Figure 6-12 Intel 82527 – LCD circuit diagram

6.7 Designing a Remote Request device

The Remote Request device in City 3 was used to invoke the generation of CAN Remote frames to request data from City 2. Typically, the device was a push button connected to an input capture pin of the MC68HC11. Whenever the button is pressed, the logic level at the input capture pin is changed. The MCU detects the level change of the pin, then it generates a Remote frame. The software used to detect hardware level changes, and to send Remote frames is described in Chapter 7.

Pin PA0 of the MC68HC11's Port A was used for an input capture function. The circuit diagram is shown in Figure 6-13.

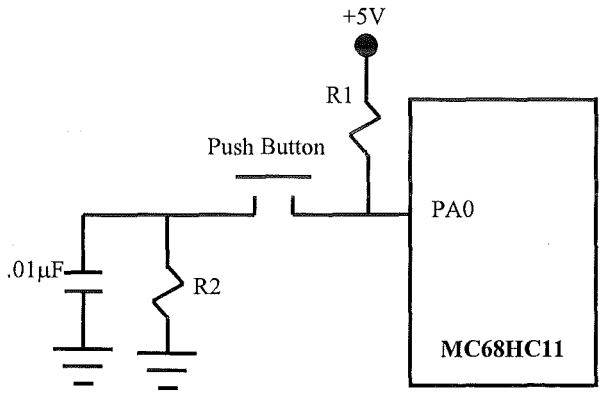


Figure 6-13 Remote Request Device circuit diagram

From Figure 6-13, the logic level of PA0 is normally 1. When the button is pressed, PA0 will be dragged down to less than or equal to 0.2 V (a logic 0).

The voltage at PA0, when the button is pressed, can be calculated by the following formula:

$$V_{PA.0} = (R2/(R1+R2))*5V$$

If the chosen value of R1 is 10kΩ and VPA.0 < 0.2 V, then R2 < 416Ω. The chosen value of R2 was 330Ω.

6.8 Designing Indicators

In this project, Light Emitting Diodes (LEDs) were used to indicate whether a City was at Set-up phase or Run phase.

Each City contained a red LED to indicate that it was at Set-up phase and a green LED to indicate that it was at Run phase. Accordingly, a dual-colour LED was used to reduce space.

A transistor was also needed to drive each LED. Any general-purpose transistor which allows a collector current of 20mA can be used (BC109 or BC549).

The Intel 82527's pins P2.0 and P2.1 of Port 2 were used to drive the red and green LEDs, respectively.

The circuit diagram is shown in Figure 14.

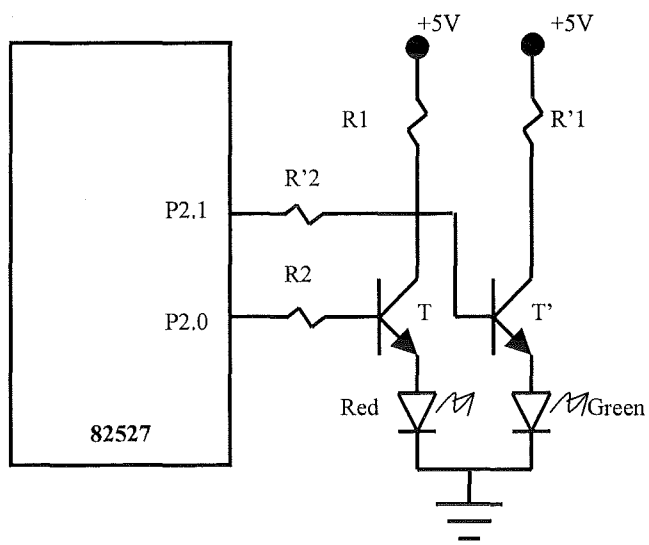


Figure 6-14 **Indicators circuit diagram**

From Figure 6-14, it is noted that because the two LEDs were the same, R'1=R1, and R'2=R2, the transistor types of T and T' should also be of the same type (BC549).

The current value going through R2 and R'2 can be chosen to be at 1mA. The voltage which drops in the LEDs is approximately 1V.

Therefore, R2=R'2= 4V/1mA=4kΩ. The chosen value of the resistors was 4.7KΩ.

The current value which goes through the LEDs can be 10-20mA.

Therefore, R1=R'1=4V/20mA=0.2Ω. The chosen value of the resistors was 330Ω.

6.9 Conclusion

This chapter described the design of the hardware part of the Small CAN Kingdom system which was introduced earlier in Chapter 4. The system consisted of four CAN nodes: The King and three Cities.

Each CAN node contained:

- AN MC68HC11 microcontroller being responsible for controlling the tasks of the node,
- An Intel 82527 CAN controller chip managing the communication protocol for the node,
- A modified DS3695 (RS485 standard) transceiver chip being used to provide an interface between the Intel 82527 and the CAN bus, and
- Peripherals to perform its tasks in the system.

The performance of the system is summarised as follows:

- The King is the network manager.
- City 1 and City 2 transmit A/D information to City 3.
- City 3 is responsible for displaying the values received from City 1 and City 2. It also generates Remote frames to City 2 in order to request information. The Remote frames are transmitted in a response to pressing a push button at the City 3.

In addition, each City contained a dual-colour LED to indicate its stage (Set-up phase or Run phase). Red indicated the Set-up phase and green indicated the Run phase of the City.

CHAPTER 7

SOFTWARE DESIGN

This chapter is concerned with the design of the software which controls the small system introduced in Chapter 4. The implementation of the software takes into account the rules associated with the Small CAN Kingdom protocol of this thesis. The complete programming codes are presented in Appendix D.

7.1 Introduction

The main tasks of the software are:

- To provide the means for users to design and download King Pages from a personal computer (PC) to the King. These King Pages are then sent to the Cities to perform the system configurations.

- To enable the Cities to receive and obey the King's instructions.

In addition, each City contains an application to execute its run-time role in the Kingdom. The role of each City is summarised as follows:

- City 1 transmits Analog/Digital (A/D) information,

- City 2 transmits A/D information after receiving a remote request (a CAN Remote frame) from City 3, and

- City 3 displays the data from City 1 and City 2. It is also able to send Remote frames to City 2 for requesting data.

The software modules required for each node are shown in the following diagrams (Figures 7-1,7-2, 7-3, 7-4).

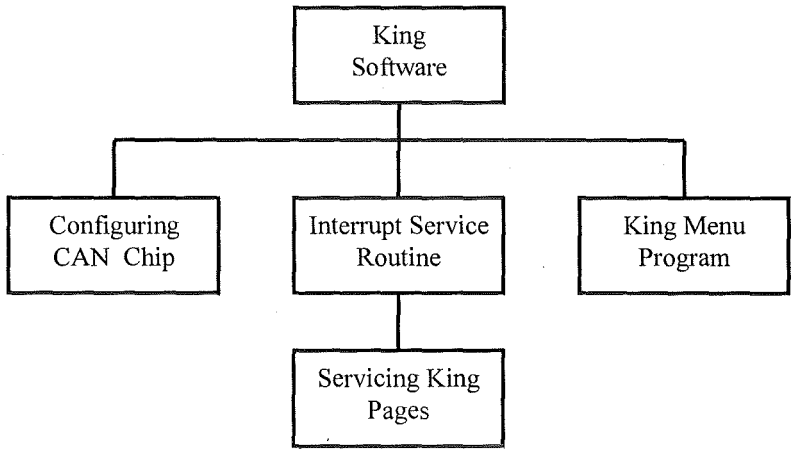


Figure 7-1 **Software modules for the King**

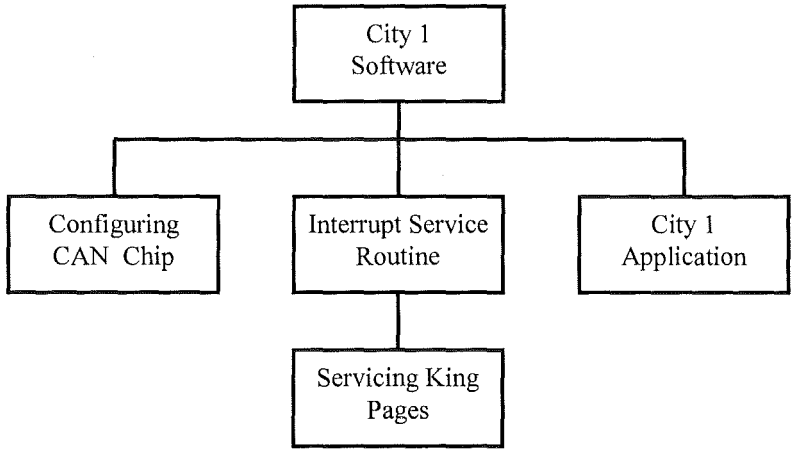


Figure 7-2 **Software modules for City 1**

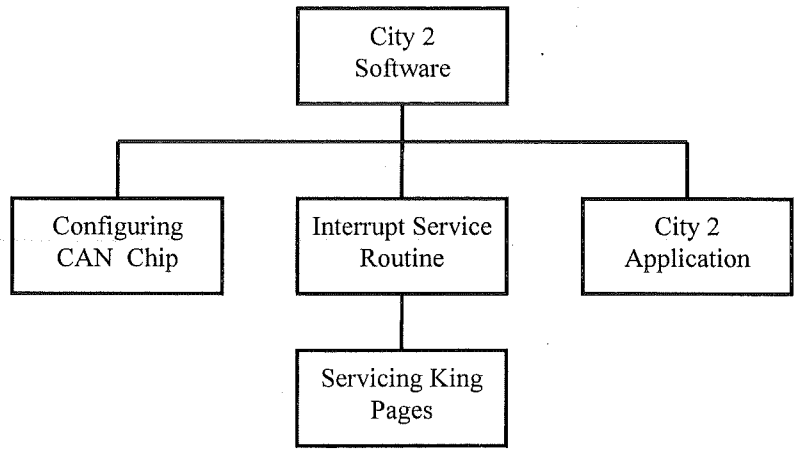


Figure 7-3 **Software modules for City 2**

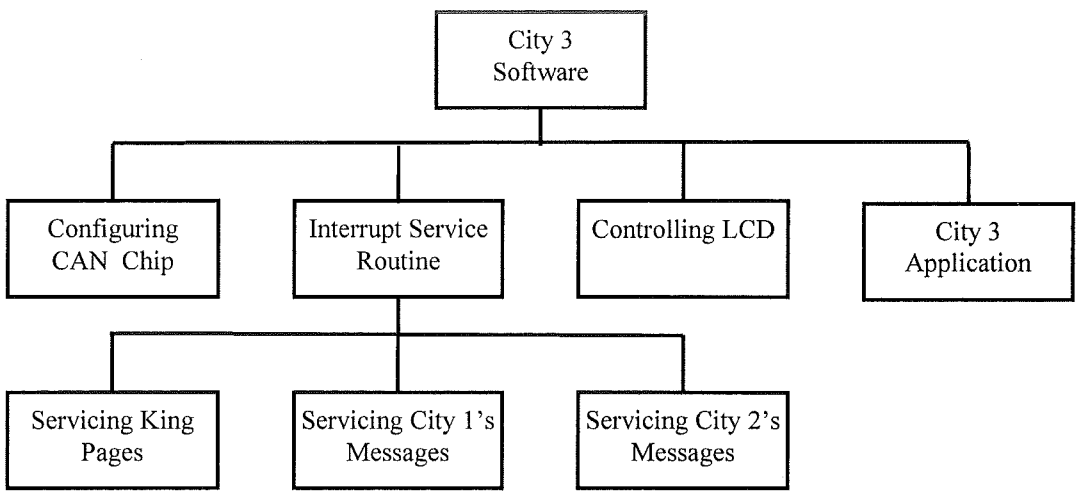


Figure 7-4 **Software modules for City 3**

The most important part of the software, in each CAN node (a City or the Capital), is the Interrupt Service Routine (ISR). This ISR can perform the following tasks:

- To decide the precise subroutine to service a King Page which arrives at a City,
- To enable the King to re-configure itself in accordance with the King's command which has been successfully transmitted, and
- To select the right Form to decode messages being transferred in the system.

The Configuring CAN chip modules are responsible for all the necessary set-up of the CAN controller chips (Intel 82527).

Each City (including the Capital) contains an application program to perform its specified role in the system:

- The Capital has a set of subroutines, including the King Menu program, which enables system designers to enter data for King Pages and to format the data according to King Page Forms.

- City 1 includes a set of subroutines which enables the City to get A/D data from the A/D device to construct Letters according to its A/D Form (see Chapter 4), and to send the Letters to the CAN bus.

- City 2 consists of similar subroutines to City 1, but it only sends its Letters after receiving a request (a Remote frame) from City 3.

- City 3 contains subroutines to display data from City 1 and City 2, and to send Remote frames to City 2 when City 2's data is needed.

The Controlling LCD module in City 3 contains a set of subroutines which are responsible for the operations of the LCD.

7.2 System operation

The operation of each node in the system is described in the flow-charts in Figure 7-5, 7-6, 7-7, and 7-8:

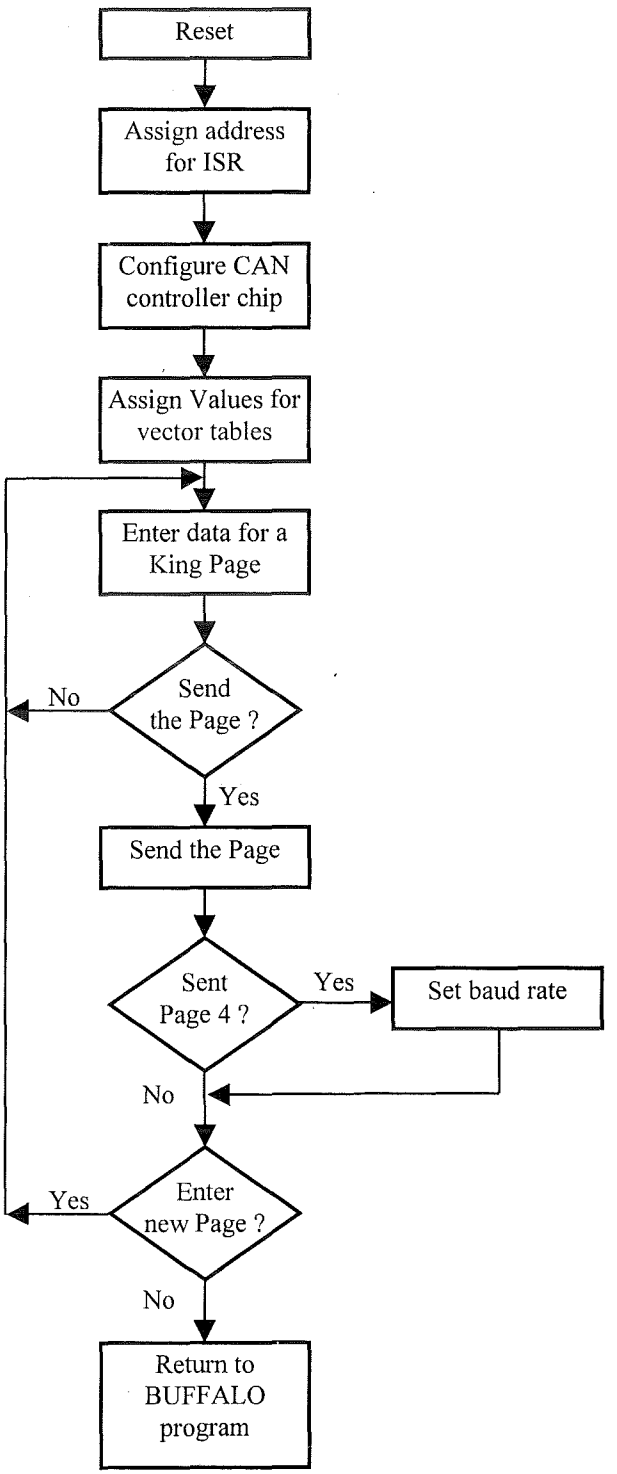


Figure 7-5 **King flow-chart**

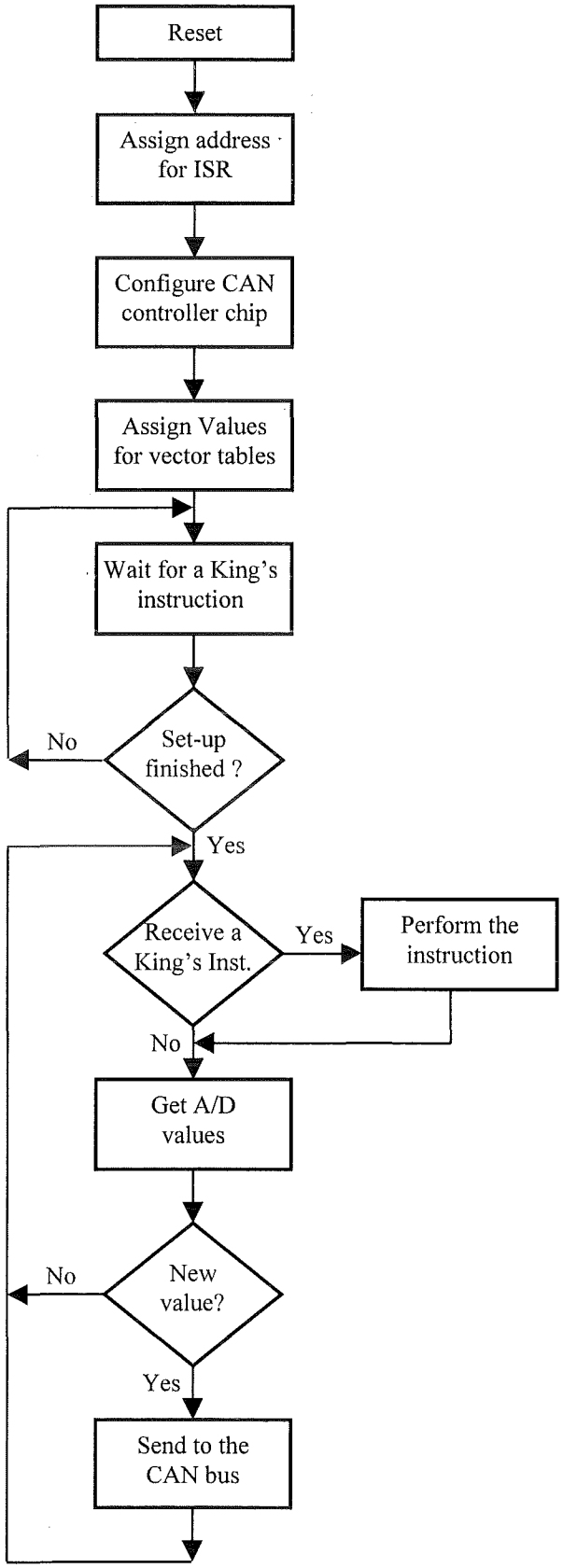


Figure 7-6 **City 1's flow-chart**

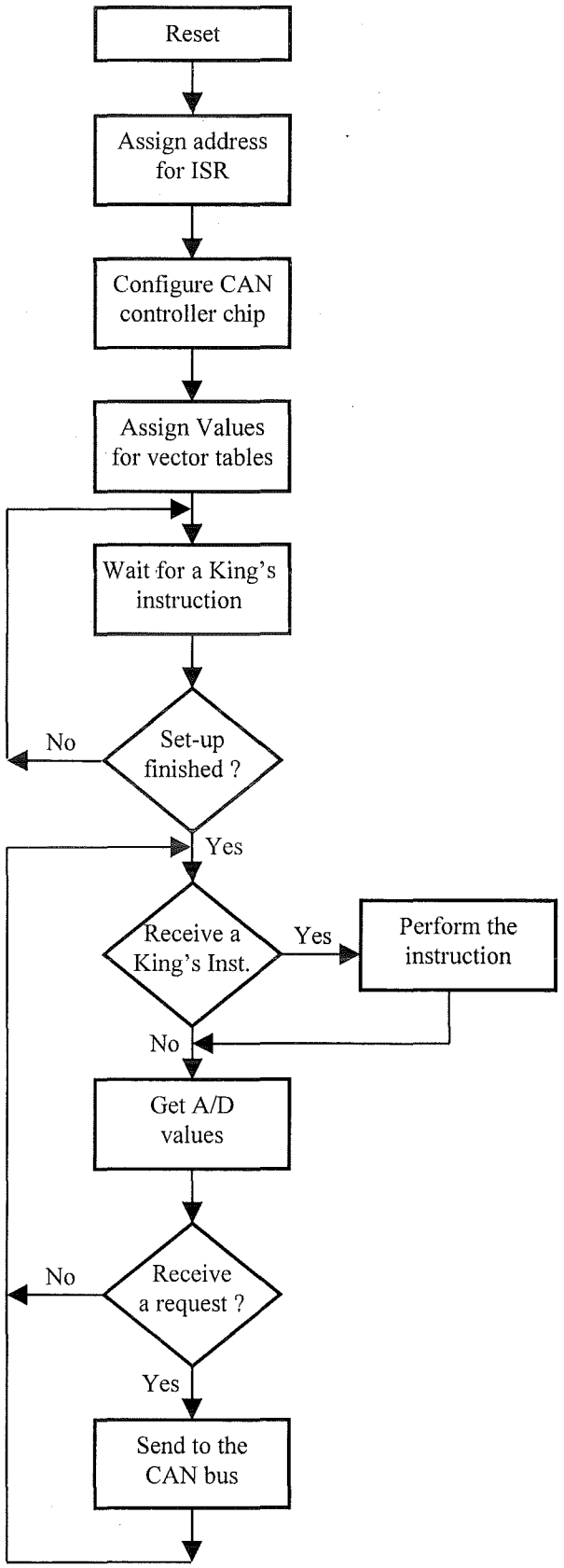


Figure 7-7 **City 2's flow-chart**

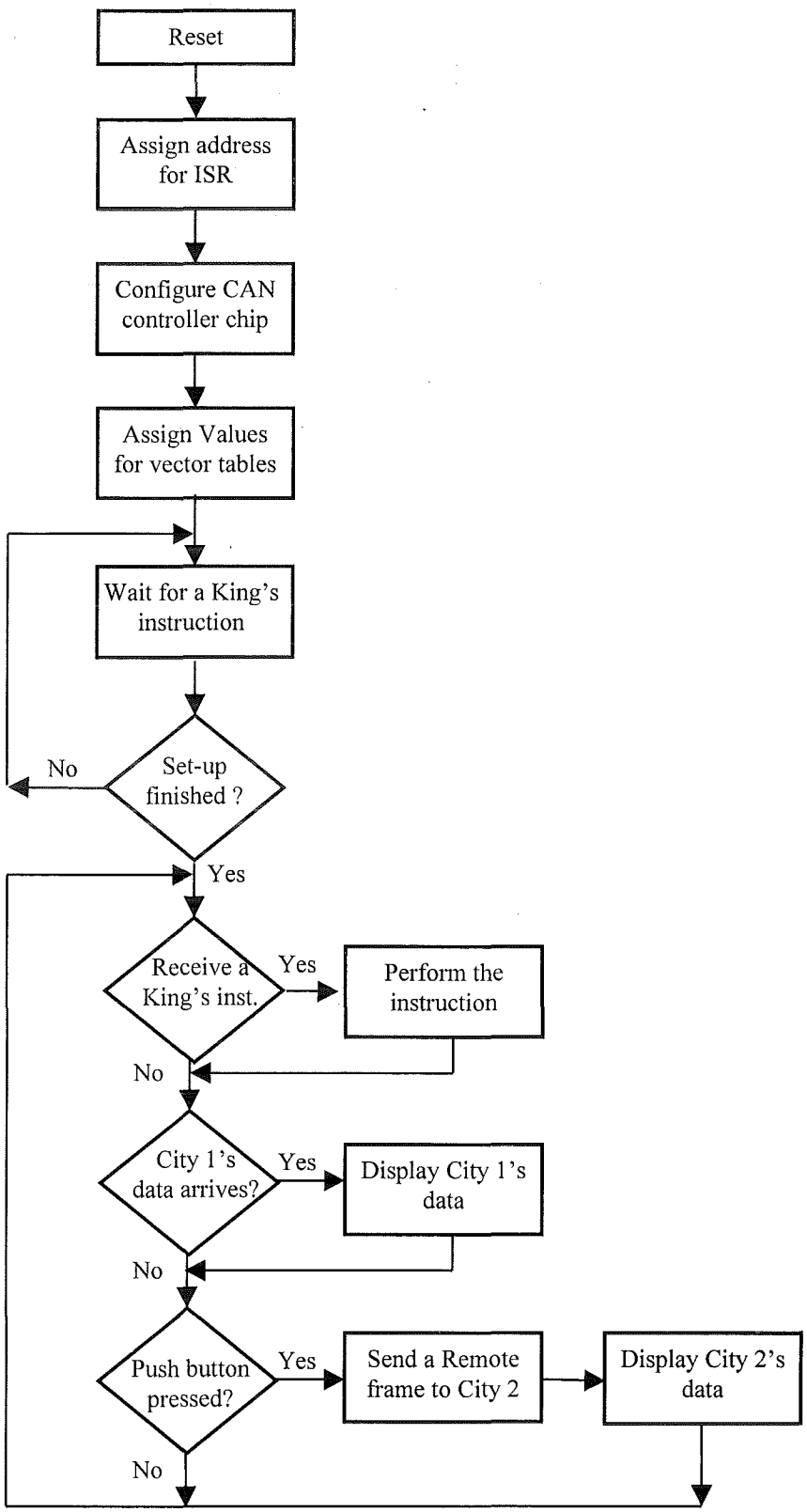


Figure 7-8 **City 3's flow-chart**

7.3 Software implementation

7.3.1 Configuring the CAN controller chips

The Configuring CAN chip modules consist of two subroutines:

- RESET_C (Resetting a CAN controller chip): the CAN controller chip is reset when its RESET# pin is driven to logic 0 for a minimum of 1ms. This is done by setting the Port A's pin PA6 of the MC68HC11 to 0 for 1ms and driving it back to 1.

- INIT_C (Initialising a CAN controller chip): this subroutine is responsible for the configuration of the CAN chip. The initial baud rate (125 kbit/s) and Mask Registers are set up during the operation of this subroutine.

7.3.1.1 Resetting a CAN controller chip

RESET_C subroutine algorithm

1. Set Port A's pin PA6 to 0
2. Delay 1ms
3. Set Port A's pin PA6 to 1

The delay time is achieved by a DELAY subroutine. The value of the index register IY specifies the multiples of 50 microseconds of the delay time.

DELAY subroutine algorithm

1. Load Accumulator B (Acc B) with 0
2. Increase the value of Acc B by 1
3. If the value of Acc B is not equal to 10 (the value for delaying 50 microsecond), then go back to step 2. Otherwise, continue to step 4.
4. Decrease the value of IY by 1
5. Go back to step 1 if IY is not zero. Otherwise, continue to step 6
6. Return from subroutine

7.3.1.2 Initialising a CAN controller chip

Programming issues

While RESET# pin of the Intel 82527 is held low, the Hardware Reset Status (RstSt) bit of the CPU Interface Register is 1 (i.e. no access to the CAN chip is possible). Therefore, the Initialising subroutine (INIT_C) should continually check this bit until it returns to 0.

The Change Configuration Enable (CCE) bit of the Control Register of the Intel 82527 should be set to 1 to enable the MCU to access the configuration registers. The Initialisation (Init) bit should also be set to 1 to prevent any incoming or outgoing messages during the initialisation.

The System Clock (SCLK) is set to XTAL/2 and the Memory Clock (MCLK) is equal to the SCLK (MCLK=SCLK).

TX0 and RX0 pins are used to transmit and to receive messages, respectively.

The initial baud rates for all CAN nodes are set to 125kbit/s. All Cities are set up at this fixed baud rate when they are connected to the system for the first time.

It is noted that the King has a special subroutine (called B_RATE) which is used to set up its baud rate. The King's baud rate values are stored into two bytes of memory (BTR0 and BTR1). The values from these two bytes are assigned by designers at the time the King is connected to the system, or by the King itself whenever it changes the system baud rate.

The system designers assign the values for BTR0 and BTR1 by the use of a modify memory command of the BUFFALO program (More details of how to use the King program are given in Appendix C).

All Message Objects of the Intel 82527 should be initially made invalid in order to prevent unused Message Objects being involved in the network communication.

Then, the user applications only validate the Message Objects which are used by the applications.

In this project, all bits of the mask registers are set to "must match". This means a particular Message Object can only transmit or receive one specified message. Nevertheless, this rule can be changed by a designer, as and when required in the future use.

In addition, all the Cities of this project use pins P1.0 and P1.1 of the Intel 82527's Port 1 to control the LEDs which are used for the indication purposes. City 3 uses both Port 1 and Port 2 for controlling the LCD module. To simplify the programming codes, all pins of these Ports are set to output. It should be noted that the unused pins of these Ports can be reconfigured later if required.

After setting up the necessary configurations, the CCE and Init bits must be reset to logic level 0 to prevent any accidental writing to the Intel 82527's configuration registers, and to enable this chip to communicate with the network.

The Message Object 1 of the Intel 82527 CAN controller in the Capital serves as the King Folder. It is assigned with Identifier 0 (in Standard format), which is the highest priority ID, to transmit King's messages. The Transmit Interrupt Enable (RXIE) bit of this Message Object should be enabled so that when the Intel 82527 successfully transmits a King message, it notifies the King by interrupting its host MCU.

The Message Object 1 of all Cities is also assigned with Identifier 0 (in Standard format) to receive King's messages. The Receive Interrupt Enable (RXIE) bit of this Message Object must be enabled so that when the Intel 82527 receives a King's message, it interrupts the MCU in order to service the message.

As a consequence of the aforementioned programming issues, the INIT_C subroutine was designed as follows:

INIT_C subroutine algorithm

1. Continually check bit RstSt until it is 0
2. Set CCE and Init bit to 1
3. Set SCLK=XTAL/2, MCLK=SCLK
4. Use TX0 and RX0 for transmitting and receiving messages. Set logic 0 is dominant and logic 1 is recessive.
5. Set baud rate (125 kbit/s initially)
6. Set Port 1 and Port 2 to output (for Cities only)
7. Reset Control Registers of all 15 Message Objects (including Message Object invalidation)
8. Set Mask Registers to "must match"
9. Set CCE and Init bit to 1
10. Enable Global Interrupt
11. Assign ID 0 (Standard format) to Message Object 1
12. Validate Message Object 1, enable Receive Interrupt for Cities, and enable Transmit Interrupt for the King.

Note that there is a difference in step 5 (Baud rate setting) for the King and Cities:

- Assign fixed baud rate (125 kbit/s) for all Cities
- The B_RATE subroutine is used to set up baud rate for the King.

B_RATE subroutine

Variables

- BTR0 (1 RAM byte) is used to store the value to set up Bit Timing Register 0 of the Intel 82527
- BTR1 (1 RAM byte) is used to store the value to set up Bit Timing Register 1 of the Intel 82527

B_RATE subroutine algorithm

1. Write the value in BTR0 to Bit Timing Registers 0
2. Write the value in BTR1 to Bit Timing Registers 1
3. Return from subroutine

7.3.2 Designing Interrupt Service Routine (ISR)

Programming issues

The mechanism for servicing incoming and outgoing messages in CAN nodes is the ISR (I_SERV). When the Intel 82527 chip in a City receives a message (a King's message or a message from other Cities), it generates an interrupt to its host microcontroller. The ISR determines the interrupt source and invokes the corresponding subroutine to service the interrupt.

A similar ISR is applied to the King after it successfully transmits a message. In other words, the Intel 82527 interrupts the host MCU (in the Capital) after it transmits a King's message. The ISR is responsible for calling a subroutine (B_SET subroutine) which enables the King to re-configure itself in accordance with its sending command.

The Interrupt Register of the Intel 82527 contains the value which specifies the interrupt source. By reading this value, the ISR can invoke the correct subroutine to service the Interrupt.

Table 7-1 shows the value of the Interrupt Register corresponding to each interrupt source.

Table 7-1 Values of Intel 82527's Interrupt Register

Interrupt Source	Register Value (Hex)
none	0
Status Register	1
Message Object 15	2
Message Object 1	3
Message Object 2	4
Message Object 3	5
Message Object 4	6
Message Object 5	7
Message Object 6	8
Message Object 7	9
Message Object 8	A
Message Object 9	B
Message Object 10	C
Message Object 11	D
Message Object 12	E
Message Object 13	F
Message Object 14	10

To enable the ISR to invoke the precise subroutine, each City includes an Interrupt service vector table (Table 7-2) which keeps the address of the subroutine servicing each interrupt source. Each location of the table contains a two-byte address for each routine. The table is located in the on-chip RAM from address 01E0H to address 01FFH. The City's application assigns the address of the service subroutine to its corresponding location in the table.

Table 7-2 Interrupt service vector table

Memory Location (Hex number)	**Subroutine Address**
01E0 01E1	Status Register
01E2 01E3	Message Object 15
01E4 01E5	Message Object 1
01E6 01E7	Message Object 2
01E8 01E9	Message Object 3
01EA 01EB	Message Object 4
01EC 01ED	Message Object 5
01EE 01EF	Message Object 6
01F0 01F1	Message Object 7
01F2 01F3	Message Object 8
01F4 01F5	Message Object 9
01F6 : 01FF	Reserved

I_PTR → 01E0 → Status Register → I_ADD

The design of the I_SERV Interrupt Service Routine (ISR) is as follows:

Variables

- I_PTR: pointer variable (2 bytes). I_PTR points to the location of the service subroutine in the vector table.
- I_ADD: pointer variable (2 bytes). I_ADD points to the actual address of the service subroutine for a corresponding Message Object.

If the value of the Interrupt Register of the Intel 82527 is named as Int_Reg, then I_PTR is calculated as follows:

I_PRT=(Int_Reg – 1) * 2 + The Base address of the vector table (01E0H) (*)

Note that the Base address of the vector table can be changed to map the table in another memory location.

The memory location pointed by I_PRT contains the address of the service subroutine for a corresponding Message Object.

I_SERV ISR algorithm

1. Load the Interrupt Register value
2. Calculate I_PRT by the formula (*)
3. Load the content of the memory location pointed by I_PTR into I_ADD
4. Call the subroutine specified by I_ADD
5. Reset the Interrupt Pending (IntPnd) bit of the corresponding Message Object.
6. Return from interrupt

7.3.3 Servicing King's messages

Programming issues

When a King Page arrives at a City or the King successfully transmits a King's command, the Intel 82527 interrupts its host MCU. The King's message (stored in the Message Object 1) is serviced by a subroutine called MSG_OB1.

Note that the address of the MSG_OB1 subroutine is assigned to the Message Object 1's location in the Interrupt service vector table.

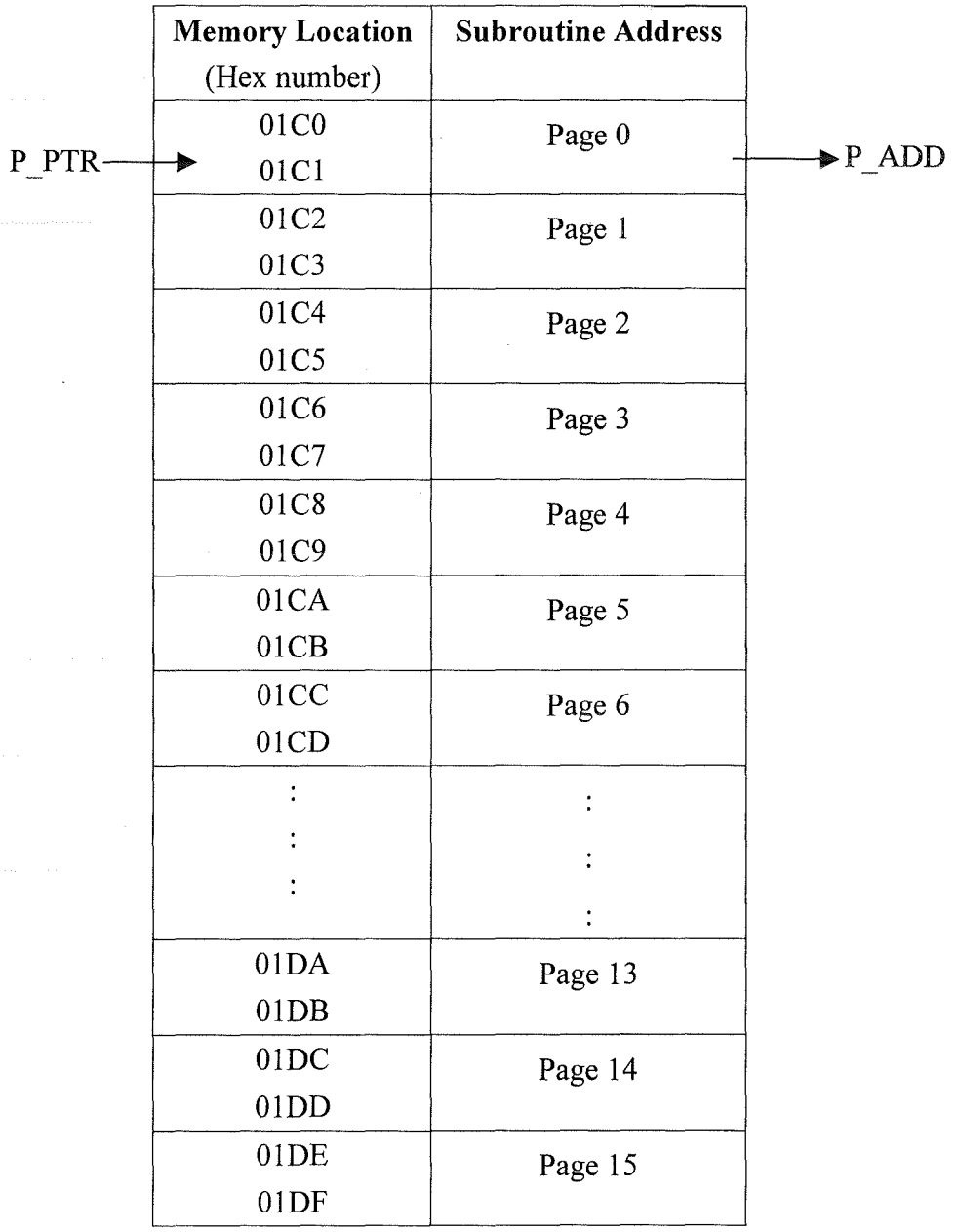
The MSG_OB1 subroutine has the following tasks:

- For a City to decide whether to accept the King Page or not, the MSG_OB1 subroutine compares the City's address contained in the Page (Line 0) with the City address. If Line 0 of the King Page contains the address which matches the City's address or contains the group address 0 (Broadcasting message), the King Page is accepted.

- If the City accepts a King Page, or after the King successfully transmits a King Page, the MSG_OB1 subroutine invokes a corresponding subroutine to service the Page according to its Page number (Line 1).

To enable the MSG_OB1 subroutine to call the correct subroutine to service the particular King Page. Each City (including the Capital) contains a King Page vector table (Table 7-3) located at address 01C0H to 01DFH. Each location consists of a two-byte subroutine address which is used to invoke the appropriate service routine for the Page. The City's or the Capital's application assigns the subroutine address to its corresponding location in the table.

Table 7-3 King Page vector table



Memory Location (Hex number)	**Subroutine Address**
01C0 01C1	Page 0
01C2 01C3	Page 1
01C4 01C5	Page 2
01C6 01C7	Page 3
01C8 01C9	Page 4
01CA 01CB	Page 5
01CC 01CD	Page 6
: : :	: : :
01DA 01DB	Page 13
01DC 01DD	Page 14
01DE 01DF	Page 15

Although only five King Pages were implemented in this project, later designers can have the opportunity to add up to 16 King Pages into the Small CAN Kingdom protocol to enhance the system's performance.

The design of the MSG_OB1 subroutine is as follows:

Variables

- C_ADD: the City's working address (1 byte)
- P_PTR: pointer variable (2 bytes). P_PTR points to the location of the service subroutine in the King Page vector table.
- P_ADD: pointer variable (2 bytes). P_ADD points to the actual address of the subroutine.

P_PTR is calculated according to the following formula:

P_PTR=(Value of Line 1 * 2)+The Base address of the vector table (01C0H) (**)

It is noted that the Base address of the vector table can be changed to map the table in another memory location.

The memory location pointed by P_PRT contains the address of the subroutine to service a corresponding King Page.

Note that each City has two memory locations for addressing purposes:

- The original City's address (ADDRESS) is stored in 1 byte EEPROM

- The C_ADD is initialised with the same value as the ADDRESS, but it can be changed by the King to assign group addresses or new City addresses.

MSG_OB1 subroutine algorithm

1. Reset the New Data (NewDat) bit in the Control Register 1 of the Message Object 1 (for Cities only)
2. Load Line 0 of the King Page (for Cities only)
3. If Line 0 contains the same value with the City's address or the group address 0, then continue to step 4. Otherwise, jump to step 8 (for Cities only)
4. Load Line 1 of the King Page
5. Calculate P_PTR according to the formula (**)
6. Load the content of the memory location pointed by P_PTR into P_ADD
7. Call the subroutine specified by P_ADD
8. Return from subroutine

It is noted that the steps from 1 to 3 enable a City to decide whether to accept the Page or not.

7.3.4 King software

Programming issues

The King's software controls the whole task of the King in the system. After successfully transmitting an instruction, the King is interrupted by the Intel 82527. The I_SERV ISR orders the MSG_OB1 subroutine to call a corresponding subroutine to service the interrupt.

If the King sends Page 0 to Page 3, it does nothing upon the interrupt. This is done by the subroutine NO_OP. The address of NO_OP subroutine is assigned to the locations for Page 0 to Page 3 in the King Page vector table.

If the King sends Page 4 (baud rate setting), it will configure itself to the new baud rate. This is done by the subroutine B_SET. The address of the B_SET subroutine is assigned to the Page 4 location in the King Page vector table.

In addition, communication between the King and system designers is controlled by the King Menu program (KING subroutine).

The following sections describe the design of each software module used by the King.

7.3.4.1 King's main program

King's main program algorithm

1. Reset CAN controller chip
2. Initialise the CAN controller chip
3. Enable the MC68HC11 interrupt
4. Assign address of the MSG_OB1 subroutine to address 01E4H in the Interrupt Service vector table

5. Assign address of the NO_OP subroutine to address 01C0H, 01C2H, 01C4H, and 01C6H in the King Page vector table
6. Assign address of the B_SET subroutine to address 01C8H in the King Page vector table
7. Call the KING subroutine (King Menu program) to enable designers to enter data
8. Set Data Length Code (DLC), Transmit, and Standard format for the King Folder (Message Object 1). Note that the DLC is calculated within the King Menu program
9. Send the King Page entered in the King Menu program
10. Wait for interrupt
11. Ask the designer whether to enter a new King Page or not. If not, continue to step 12. Otherwise, go back to step 7.
12. Return to the prompt of BUFFALO monitor program

7.3.4.2 NO_OP subroutine

NO_OP subroutine algorithm

1. No operation (NOP)
2. Return from subroutine

7.3.4.3 B_SET subroutine

B_SET subroutine algorithm

1. Write the value in Line 3 of King Page 4 to BTR0
2. Write the value in Line 4 of King Page 4 to BTR1
3. Set CCE bit in Control Register of Intel 82527 to logic 1
4. Call B_RATE to change the baud rate
5. Set CCE bit in Control Register of Intel 82527 to logic 0
6. Return from subroutine

7.3.4.4 King Menu program

The aim of this program is to enable system designers to design King Pages corresponding to the configuration instructions for the system. The design of each King Page is done by using an IBM PC. Next, the Page is downloaded to the Capital's microcontroller and sent to a City or a group of Cities. The software which enables the King Pages to be designed is referred to as the King Menu program.

7.3.4.4.1 Introduction

The King Menu program has two tasks:

1. Enabling system designers to enter King Pages
2. Formatting the entered data according to King Page Forms

System designers enter a King Page via the PC's keyboard, the data is echoed on the PC's monitor. The menu on the monitor would appear as shown in Figure 7-9.

```
*** KING PAGE***
LINE 0: 01
LINE 1: 01
LINE 2: 20
LINE 3: 0A
LINE 4: 40
LINE 5: 00
LINE 6: 00
LINE 7: 88
Send ? (Y/N): Y
New page? (Y/N): N
> (BUFFALO prompt)
```

Figure 7-9 King Menu

The King Page shown in Figure 7-9 is an example of a King Page 1. Each Line on the screen corresponds to a Line on King Page 1 Form.

Each Line consists of two Hexadecimal numbers which carry the information corresponding to a particular King Page as described earlier in Chapter 4.

After the user finishes entering data for a King Page, the King Menu program asks the user whether or not to send the Page. If a 'Y' character is entered, the King Menu program returns to the main program to send the Page. Otherwise, the Page is discarded and the prompt for entering a new Page is displayed on the monitor.

After finishing the system configurations, the user can return to the BUFFALO program by entering an 'N' character at the 'New Page? (Y/N):' prompt.

The King Menu program also provides the following facilities to assist users to correct the data being currently entered:

- If the current character being entered is not in the ranges of Hexadecimal characters, the current Line is discarded. The program repeats this Line to prompt the user to re-enter the data.

- After entering data for a Line, the user presses <Enter> to confirm the Line. The monitor will prompt the user to enter value for the next Line.

- Data for a Line can be changed by pressing <Ctrl+X> instead of <Enter>.

- If a King Page contains less than eight Lines, after pressing <Enter> to confirm the value of the last Line, the user can press <Ctrl+ESC> to ignore the rest of the Lines.

The format of each Line in a King Page is as follows:

Page 0

Line 0: CC (Hex number- indicate City or Group address)

Line 1: 00 (Page 0 – Finish Set-up phase)

Page 1

Line 0: CC

Line 1: 01 (Page 1 – Assign Envelop to Folder)

Line 2: FR

Line 3: AA (Arbitration 0) MSB

Line 4: AA (Arbitration 1)

Line 5: AA (Arbitration 2)

Line 6: AA (Arbitration 3) LSB

Line 7: mmrrdxrr (Binary number)

Note:

Line 2: F-Folder or Message Object number

R- Reserved, always being 0

Line 3, Line 4, Line 5, Line 6: contain the Arbitration values of the Arbitration field in each Message Object of the CAN controller chip (Intel 82527)

Line 7: mm- enable/disable the Folder

01- disable

10- enable

11- unchange

d- direction of the message contained in the Folder

0- Receive

1- Transmit

x- message format

0- Standard

1- Extended

Page 2

Line 0: CC

Line 1: 02 (Page 2 – Assign new City's or Group address)

Line 2: NN (new City's or Group address)

Page 3

Line 0: CC

Line 1: 03 (Page 3 – Ungroup/Restore the original City's address)

Page 4

Line 0: CC (should be 00 – broadcast message)

Line 1: 04 (Page 4- Baud rate setting)

Line 2: $s_{JW}s_{JW}$bbbbbb (Binary)

Line 3: $s_{Spl}t_{TSEG2}t_{TSEG2}t_{TSEG2}t_{TSEG1}t_{TSEG1}t_{TSEG1}t_{TSEG1}$ (Binary)

Note:

Line 2: $s_{JW}s_{JW}$ – (Re)Synchronisation jump width – Value: 0-3 (Decimal)

bbbbbb – Baud rate prescaler – value 0-63 (Decimal)

Line 3: s_{Spl} – Sampling mode

0- The CAN bus is sample once per bit time

1- The CAN bus is sample three times per bit time

t_{TSEG2} t_{TSEG2} t_{TSEG2} – Time segment 2 – Value: 1-7 (Decimal)

$t_{TSEG1}t_{TSEG1}t_{TSEG1}t_{TSEG1}$ – Time segment 2 – Value: 1-15 (Decimal)

7.3.4.4.2 Designing the King Menu program

The King Menu program uses four utility subroutines of the BUFFALO monitor program contained in the MC68HC11 on chip ROM as shown in Table 7-4:

Table 7-4 BUFFALO's utility subroutines

Address	Subroutine	Description
FFA0H	.UPCASE	Convert a lower case character in Accumulator A to upper case
FFC4H	.OUTCRL	Output ASCII carriage return follow by a line feed
FFC7H	.OUTSTR	Output a string of ASCII bytes pointed by address in the index register IX until encountering the EOT character (04H)
FFCD	.INCHAR	Input an ASCII character into Accumulator A and echo back

The King Menu program (called KING subroutine) also has the following subroutines:

- K_DISP controls the process of entering King Pages and calculates the number of Lines on the Pages.
- L_ENTER controls the value entering process for a Line.
- HEXBYTE gets two ASCII characters entered from the user, converts them to Hexadecimal numbers, and stores the numbers into TMP1 (1 byte).
- HEXBIN converts a character in Accumulator A into a Hexadecimal number.

King Menu program messages

The following messages are output to the PC screen to prompt the designers to enter values:

MSG0: 'New Page? (Y/N):'
MSG1: '***KING PAGE***'
MSG2: 'LINE 0:'
MSG3: 'LINE 1:'
MSG4: 'LINE 2:'
MSG5: 'LINE 3:'
MSG6: 'LINE 4:'
MSG7: 'LINE 5:'
MSG8: 'LINE 6:'
MSG9: 'LINE 7:'
MSG10: 'Send? (Y/N):'

Variables

- TMP1 (1 byte) stores the hexadecimal value of a Line
- PTR1 (2 bytes) points to the memory location of the Menu program messages
- PTR2 (2 bytes) points to the location to store the entered value (the Data field of the Message Object 1)
- NHEX (1 byte) indicates the value entered is not a Hexadecimal number if NHEX is not equal to 0
- STOP (1 byte): if this byte contains <Ctrl+ESC> character, the current Line is ignored
- COUNT (1 byte) indicates the number of Lines containing in a King Page. Note that the number of Lines is specified by the upper-half of COUNT, the lower-half is unused.

KING subroutine algorithm

1. Reset COUNT to calculate the number of Lines for a new Page
2. Output MSG1 to the PC screen
3. Load the memory address of MSG2 to PTR1
4. Load the address of the first byte in the Data field of Message Object 1 to PTR2
5. Call K_DISP subroutine
6. Output MSG10 to the PC screen
7. Receive character 'Y' or 'N' from users. If 'Y', then go to step 8 to return to the main program. If 'N', then go to step 1 to re-enter the Page
8. Return from subroutine

K_DISP subroutine algorithm

1. Call L_ENTER subroutine to enter values for a Line
2. If STOP contains <Ctrl+ESC> character, then go to step 8. Otherwise, continue to step 3.
3. Store the Line values to the address specified by PTR2
4. Increase COUNT by 10H.
5. Increase the value of PTR2 to point to the next location
6. Compare the content of PTR1 with the address of MSG9. If equal, then go to step 1. If not equal, increase PTR1 to point to the next Menu program's message address location.
7. Go to step 1
8. Return from subroutine

L_ENTER subroutine algorithm

1. Output the message to prompt users to enter values for a particular Line
2. Call HEXBYTE subroutine
3. If STOP contains <Ctrl+ESC> character, then go to step 7. Otherwise, continue to step 4.
4. If the entered value is not a Hexadecimal number, then go to step 1 to re-enter the value
5. If <Ctrl+X> is pressed, then ignore the value and go to step 1
6. If <Enter> is pressed, then go to step 7
7. Return from subroutine

HEXBYTE subroutine algorithm

1. Reset NHEX
2. Get a character from the user (by calling .INCHAR subroutine). It is noted that the character is stored in Accumulator A
3. Store the character to STOP
4. If STOP contains <Ctrl+ESC> character, then go to step 13. Otherwise, continue to step 5.
5. Call HEXBIN subroutine to convert the character to a Hexadecimal number
6. Go to step 12 if the value is not a Hexadecimal number
7. Store this number into the upper-half of TMP1
8. Get a character from the user (Call .INCHAR subroutine).
9. Call HEXBIN subroutine to convert the character to a Hexadecimal number
10. Go to step 12 if the value is not a Hexadecimal number
11. Store this number into the lower-half of TMP1
12. Increase NHEX
13. Return from subroutine

HEXBIN subroutine algorithm

1. Call .UPCASE subroutine
2. If the value in Accumulator A is in the ranges of '0' to '9' or 'A' to "F', then convert it to a Hexadecimal number
3. If the value is not in either the ranges mentioned in step 2, then load Acc A with FFH to indicate the value is not a Hexadecimal number
4. Return from subroutine

7.3.5 Designing the software to service King Pages in Cities

When a King Page arrives at a City (or Cities), the MSG_OB1 subroutine calls a corresponding subroutine to service the Page according to its Page number.

In this project, five subroutines were designed to service the five King Pages:

1. **PG_0** subroutine services King Page 0
2. **PG_1** subroutine services King Page 1
3. **PG_2** subroutine services King Page 2
4. **PG_3** subroutine services King Page 3
5. **PG_4** subroutine services King Page 4

The application program must assign these subroutines' addresses into their locations in the King Page vector table (Table 7-3)

7.3.5.1 PG_0 subroutine

Variables

- WORK (1 byte) indicates that the Set-up phase has finished
  - WORK = 0: Set-up phase
  - WORK = 1: Set-up phase has finished and the City can start to work

PC_0 subroutine algorithm

1. Assign WORK:=1
2. Return from subroutine

7.3.5.2 PG_1 subroutine

Variable

- OBJ_NO (2 bytes): the Base address which is the first memory address of the Folder number (Message Object number).

PG_1 subroutine algorithm

1. Determine the Folder number by reading the value of Line 2
2. Disable the Folder
3. Store Arbitration 0
4. Store Arbitration 1
5. Store Arbitration 2
6. Store Arbitration 3
7. Configure the Folder according to the value of Line 7
8. Return from subroutine

7.3.5.3 PG_2 subroutine

PG_2 subroutine algorithm

1. Store the value of Line 3 to C_ADD (the 1 RAM byte for City's addressing)
2. Return from subroutine

7.3.5.4 PG_3 subroutine

PG_3 subroutine algorithm

1. Get the original City's address in ADDRESS (the 1 EEPROM byte for City's addressing)
2. Store the original City's address to C_ADD (the 1 RAM byte for City's addressing)
3. Return from subroutine

7.3.5.5 PG_4 subroutine

PG_4 subroutine algorithm

1. Set CCE bit to 1 in Control Register to enable writing to Bit Timing Registers
2. Store the value of Line 2 to Bit Timing Register 0
3. Store the value of Line 3 to Bit Timing Register 1
4. Set CCE bit to 0 in Control Register to disable writing to Bit Timing Registers
5. Return from subroutine

7.3.6 Cities' software

The software in each City includes four parts:

1. Assigning address for the Interrupt Service Routine (ISR)

2. Initialisation being responsible for initialising CAN controller chip, assigning values for the two vector tables.

3. Set-up phase waiting for set-up instructions from the King

4. Run phase controlling the designed operations of the City

An important observation is the fact that the first three parts of the City application are almost the same for all three Cities with the fourth part different for each.

7.3.6.1 Assigning address for ISR

At the beginning of a City program, the command

```
JMP   I_SERV
```

is placed at address 00EEH of the Interrupt Vector table in the MC68HC11 on-chip memory.

When an IRQ interrupt occurs, the Program Counter points to the address of I_SERV ISR.

7.3.6.2 Initialisation

Algorithm for the Initialisation part

1. Reset CAN controller chip
2. Initialise the CAN controller chip
3. Initialise the LCD (for City 3 only)
4. Display the message '* CAN SYSTEM DEMO *' on LCD (for City 3 only)
5. Turn ON the Red LED to indicate Set-up phase
6. Assign the original address (ADDRESS) to C_ADD
7. Assign WORK to 0 to indicate Set-up phase
8. Assign address of the MSG_OB1 subroutine to address 01E4H in the Interrupt Service vector table
9. Assign addresses of the MSG_OB2 and MSG_OB3 subroutines to address 01E6H and 01E8H, respectively in the Interrupt Service vector table (for City 3 only)
10. Assign addresses of PG_0, PG_1, PG_2, PG_3, and PG_4 subroutines to address 01C0H, 01C2H, 01C4H, 01C6H, and 01C8H, respectively in the King Page vector table
11. Enable interrupt for MC68HC11

Note:

- City 3 contains a set of subroutines to control the LCD such as initialising LCD (INIT_L) and displaying messages in the LCD (DISP). These subroutines are described in Section 7.3.6.4.3

- City 3 receives messages from City 1 and City 2. Folder 2 and Folder 3 (Message Object 2 and Message Object 3) of City 3 are used to store messages from City 1 and City 2, respectively. MSG_OB2 subroutine is used to decode messages in Folder 2. MSG_OB3 subroutine is used to decode messages in Folder 3.

7.3.6.3 Set-up phase

Algorithm for Set-up phase in a City's main program

1. Wait for a King instruction
2. If WORK=0, then go back to step 1. Otherwise, continue to Run phase

7.3.6.4 Run phase

7.3.6.4.1 City 1

In Run phase, City 1 gets the A/D value from A/D device, formats the value according to the A/D form (see Chapter 4), and sends it to the CAN bus whenever the value has been changed.

Algorithm for Run phase in City 1's main program

1. Turn on the Green LED to indicate the Run phase
2. Set the Data Length Code of the Message Object 2 to 8 bytes
3. Get A/D value
4. If the value is changed, then continue to step 5. Otherwise, go back to step 3

5. Set the CPU update (CPUUpd) and New data (NewDat) bits in the Control Register 1 of Message Object 2 to logic 1
6. Get City 1's address.
7. Convert the address to ASCII characters, and store them into the first three bytes of the Data field (by calling ASCII subroutine)
8. Store the ':' character into the fourth byte of the Data field
9. Convert the A/D value to ASCII characters, and store them into the next three bytes of the Data field (by calling ASCII subroutine)
10. Store the End of String (EOT) character (04H) into the eighth byte of the Data field
11. Reset CPUUpd bit to 0
12. Transmit the message
13. Go back to step 3

ASCII subroutine

This subroutine is used first to convert two Hexadecimal numbers contained in Acc B into their equivalent Decimal numbers, then to convert the Decimal numbers into ASCII characters, and store the characters to the memory addresses specified by Index Register IY

ASCII subroutine algorithm

1. Convert the Hexadecimal numbers into Decimal numbers
2. Convert the Decimal numbers into ASCII characters
3. Store the characters into memory location specified by IY

7.3.6.4.2 City 2

In Run phase, City 2 gets the A/D value from A/D device, formats the value according to the A/D form (see Chapter 4), and sends it to the CAN bus when the City receives a request (Remote frame) from City 3.

Algorithm for Run phase in City 2's main program

1. Turn on the Green LED to indicate Run phase
2. Set the Data Length Code of the Message Object 2 to 8 bytes
3. Get A/D value
4. Set the CPU update (CPUUpd) and New data (NewDat) bits in the Control Register 1 of Message Object 2 to logic 1
5. Get the City 2's address.
6. Convert the address to ASCII characters, and store them into the first three bytes of the Data field (by calling ASCII subroutine)
7. Store the ':' character into the fourth byte of the Data field
8. Convert the A/D value to ASCII characters, and store them into the next three bytes of the Data field (by calling ASCII subroutine)
9. Store the End of String (EOT) character (04H) into the eighth byte of the Data field
10. Reset CPUUpd bit to 0
11. Delay 2msecs to allow the Intel 82527 CAN controller to complete the transmission of the message if there is a request from City 3
12. Go back to step 3

7.3.6.4.2 City 2

In Run phase, City 2 gets the A/D value from A/D device, formats the value according to the A/D form (see Chapter 4), and sends it to the CAN bus when the City receives a request (Remote frame) from City 3.

Algorithm for Run phase in City 2's main program

1. Turn on the Green LED to indicate Run phase
2. Set the Data Length Code of the Message Object 2 to 8 bytes
3. Get A/D value
4. Set the CPU update (CPUUpd) and New data (NewDat) bits in the Control Register 1 of Message Object 2 to logic 1
5. Get the City 2's address.
6. Convert the address to ASCII characters, and store them into the first three bytes of the Data field (by calling ASCII subroutine)
7. Store the ':' character into the fourth byte of the Data field
8. Convert the A/D value to ASCII characters, and store them into the next three bytes of the Data field (by calling ASCII subroutine)
9. Store the End of String (EOT) character (04H) into the eighth byte of the Data field
10. Reset CPUUpd bit to 0
11. Delay 2msecs to allow the Intel 82527 CAN controller to complete the transmission of the message if there is a request from City 3
12. Go back to step 3

7.3.6.4.3 City 3

In Run phase, City 3 checks the Remote Request device (the push button). If the push button is pressed, then the City sends a Remote frame to City 2 to request data. The ISR controls the data display of messages from City 1 and City 2.

Algorithm for Run phase in City 3's main program

1. Turn on the Green LED to indicate Run phase
2. Set Receive Interrupt Enable (RXIE) for Message Object 2 and Message Object 3
3. If the button is pressed (PA0=0), then send Remote frame to City 2
4. Delay 1 sec
5. Go back to step 3

It is noted that when a message arrives, the ISR invokes a corresponding subroutine to decode it:

- MSG_OB2 subroutine decodes the messages from City 1
- MSG_OB3 subroutine decodes the messages from City 2

MSG_OB2 subroutine algorithm

1. Determine the Base address for the Message Object 2 (7020H)
2. Call LCD subroutine to display data
3. Return from subroutine

MSG_OB3 subroutine algorithm

1. Determine the Base address for the Message Object 3 (7030H)
2. Call LCD subroutine to display data
3. Return from subroutine

LCD subroutine algorithm

1. Configure to display data on the second line of the LCD
2. Display the message in the corresponding Message Object
3. Return from subroutine

Subroutines to control the LCD

The LCD control subroutines were developed and thoroughly tested by Wetton (1995). The subroutines are listed as follows:

- INIT_L: Initialising the LCD
- DISP: Display a string in the LCD with the string's address specified by index register IX
- OUTPUT: output instructions from Acc A to the LCD
- SCREEN: Output data from Acc A to the LCD

In this project, these subroutines were utilised with minor changes in the software due to some differences in the hardware design (see Chapter 6).

INIT_L subroutine algorithm

1. Delay 16msecs
2. Output an initialising instruction to LCD
3. Delay a period of time required for each instruction
4. Go back to step 2 until the last instruction is fetched
5. Return from subroutine

DISP subroutine algorithm

1. Load a character at the address specified by IX to Acc A
2. If the character is EOT, then go to step 6. Otherwise, continue to step 3
3. Call SCREEN to display the character
4. Delay 50 micro secs
5. Go back to step 1
6. Return from subroutine

OUTPUT subroutine algorithm

1. Load an instruction to Port 1 of the Intel 82527
2. Set LCD's pins RS=0, R/W#=0, E=0
3. Set LCD's pins RS=0, R/W#=0, E=1
4. Set LCD's pins RS=0, R/W#=0, E=0
5. Return from subroutine

SCREEN subroutine algorithm

1. Load data to Port 1 of the Intel 82527
2. Set LCD's pins RS=1, R/W#=0, E=0
3. Set LCD's pins RS=1, R/W#=0, E=1
4. Set LCD's pins RS=1, R/W#=0, E=0
5. Return from subroutine

7.4 Testing

The following testing schemes were performed to check the system operations:

7.4.1 Set-up phase

After each City had been reset, all the Cities waited for instructions from the King.

Expected and actual result: The red LEDs in all of the Cities were turned ON to indicate that the Cities were waiting for the King's instructions.

The King sent King Page 1 messages to each City to determine communications between Cities. The values of Page 1 for each City were as follows:

To City 1:
Assigning ID 2 (Standard format) to Folder 2.

Page 1
Line 0: 01 (City 1)
Line 1: 01 (Page 1)
Line 2: 20 (Folder 2)
Line 3: 00 (Arbitration 0) MSB
Line 4: 40 (Arbitration 1)
Line 5: 00 (Arbitration 2)
Line 6: 00 (Arbitration 3) LSB
Line 7: 88 (Enable the Folder, Transmit, Standard)

To City 2:

Assigning ID 3 (Standard format) to Folder 2.

Page 1

Line 0: 02 (City 2)

Line 1: 01 (Page 1)

Line 2: 20 (Folder 2)

Line 3: 00 (Arbitration 0) MSB

Line 4: 60 (Arbitration 1)

Line 5: 00 (Arbitration 2)

Line 6: 00 (Arbitration 3) LSB

Line 7: 88 (Enable the Folder, Transmit, Standard)

To City 3:

Assigning ID 2 (Standard format) to Folder 2.

Page 1

Line 0: 03 (City 3)

Line 1: 01 (Page 1)

Line 2: 20 (Folder 2)

Line 3: 00 (Arbitration 0) MSB

Line 4: 40 (Arbitration 1)

Line 5: 00 (Arbitration 2)

Line 6: 00 (Arbitration 3) LSB

Line 7: 80 (Enable the Folder, Receive, Standard)

Assigning ID 3 (Standard format) to Folder 3.

Page 1

Line 0: 03 (City 3)

Line 1: 01 (Page 1)

Line 2: 30 (Folder 3)

Line 3: 00 (Arbitration 0) MSB

Line 4: 60 (Arbitration 1)

Line 5: 00 (Arbitration 2)

Line 6: 00 (Arbitration 3) LSB

Line 7: 80 (Enable the Folder, Receive, Standard)

After sending these Pages to the Cities, the King then sent King Page 0 to all Cities to inform them that the Set-up phase had been completed. The values of the Page were as follows:

Page 0

Line 0: 00 (Broadcast to all Cities)

Line 1: 00 (Page 0)

Expected and actual results:

- The red LEDs were turned OFF
- The green LEDs were turned ON to indicate that the Cities started to work
- From then, the Cities communicated with each other as designed

7.4.2 Run phase

7.4.2.1 Testing communication between City 1 and City 3

- Adjusting the A/D device in City 1

Expected and actual result: City 3 received the data and displayed it in the LCD. The LCD displayed the following information:

```
* CAN SYSTEM DEMO *
CITY_001:128
```

Note that the value 128 was the A/D value from City 1. The value was changed whenever the A/D device was manually adjusted.

7.4.2.2 Testing communication between City 2 and City 3

- Adjusting the A/D device in City 2
- Pressing the button on City 3

These two actions were executed several times to test the system fully.

Expected and actual result: City 3 received the data and displayed it in the LCD. The LCD displayed the following information:

```
* CAN SYSTEM DEMO *
CITY_002:068
```

Note that the value 68 was the A/D value from City 2. The value was changed whenever the A/D device was manually adjusted.

7.4.3 Additional testing

7.4.3.1 Changing message Identifiers

Whilst the system is working in the Run phase, the King can change the Identifier for Cities' Folders in order to change message priorities. This was done by sending King Page 1 messages to corresponding Cities.

7.4.3.1.1 Changing communication priority between City 1 and City 3

Because the messages in City 1 are sent immediately, whenever the A/D values are changed, Folder 2 of City 1 should be disabled before changing Folder 2 of City 3's message ID. Otherwise, if City 1 sends its messages and no City receives, the system could malfunction. The order of changing message ID steps was as follows:

- Disabling the Folder 2 of City 1
- Assigning new message ID for Folder 2 of City 3
- Assigning the same message ID for Folder 2 of City 1

According to the algorithm of PG_1 subroutine, a Folder can be disabled by sending the first three Lines of King Page 1. The following King messages were sent:

To City 1:
Disabling Folder 2

Page 1
Line 0: 01 (City 1)
Line 1: 01 (Page 1)
Line 2: 20 (Folder 2)

To City 3:

Assigning ID 10 (Extended format) to Folder 2.

Page 1

Line 0: 03 (City 3)

Line 1: 01 (Page 1)

Line 2: 20 (Folder 2)

Line 3: 00 (Arbitration 0) MSB

Line 4: 00 (Arbitration 1)

Line 5: 00 (Arbitration 2)

Line 6: 50 (Arbitration 3) LSB

Line 7: 84 (Enable the Folder, Receive, Extended)

To City 1:

Assigning ID 10 (Extended format) to Folder 2.

Page 1

Line 0: 01 (City 1)

Line 1: 01 (Page 1)

Line 2: 20 (Folder 2)

Line 3: 00 (Arbitration 0) MSB

Line 4: 00 (Arbitration 1)

Line 5: 00 (Arbitration 2)

Line 6: 50 (Arbitration 3) LSB

Line 7: 8C (Enable the Folder, Transmit, Extended)

Expected and actual result: The two Cities communicated with each other as specified. This was done by repeating the test in Section 7.4.2.1.

7.4.3.1.2 Changing communication priority between City 2 and City 3

City 2 sends its messages only when it receives Remote frames from City 3, hence, in this case, the order of changing the message IDs is not important and it is not necessary to disable City 2's Folder. The King sent the following messages to City 2 and City 3:

To City 2:

Assigning ID 9 (Extended format) to Folder 2.

Page 1

Line 0: 02 (City 2)

Line 1: 01 (Page 1)

Line 2: 20 (Folder 2)

Line 3: 00 (Arbitration 0) MSB

Line 4: 00 (Arbitration 1)

Line 5: 00 (Arbitration 2)

Line 6: 48 (Arbitration 3) LSB

Line 7: 8C (Enable the Folder, Transmit, Extended)

To City 3:

Assigning ID 9 (Extended format) to Folder 2.

Page 1

Line 0: 03 (City 3)

Line 1: 01 (Page 1)

Line 2: 30 (Folder 3)

Line 3: 00 (Arbitration 0) MSB

Line 4: 00 (Arbitration 1)

Line 5: 00 (Arbitration 2)

Line 6: 48 (Arbitration 3) LSB

Line 7: 84 (Enable the Folder, Receive, Extended)

Expected and actual result: The two Cities communicated with each other as designed. This was done by repeating the test in Section 7.4.2.2.

7.4.3.2 Changing Cities' addresses

The King sent King Page 2 to the Cities to assign a new address to a City. This test was done by the following steps:

- Assigning City's address 005 to City 1
- Assigning City's address 006 to City 2

The values of King Page 2, sent to each City, were as follows:

To City 1:

Page 2

Line 0: 01 (City 1)

Line 1: 02 (Page 2)

Line 3: 05 (new City or group address 005)

To City 2:

Page 2

Line 0: 02 (City 2)

Line 1: 02 (Page 2)

Line 3: 06 (new City or group address 006)

Expected and actual result: The Cities operated as specified. This was done by repeating the tests in Section 7.4.2

The display in the LCD of City 3 appeared as follows:

City 1:

* CAN SYSTEM DEMO * CITY_001:140

City 2:

* CAN SYSTEM DEMO * CITY_002:008

7.4.3.5 Changing baud rate

Initially, the system was working at 125kbit/s. In order to change the baud rate of the system, the King broadcasted King Page 4 to all Cities with a new baud rate setting.

For example, setting 100kbit/s to the system; the values of King Page 4 were as follows:

Page 4

Line 0: 00 (Broadcast message)

Line 1: 04 (Page 4)

Line 2: 43 (value for Bit Timing Register 0)

Line 3: 7A (value for Bit Timing Register 1)

Expected and actual result: The Cities operated as specified. This was checked by repeating the tests in previous sections.

Expected and actual result: The Cities operated as designed. This was done by repeating the tests in Section 7.4.2

The display in the LCD of City 3 appeared as follows:

City 1:

```
* CAN SYSTEM DEMO *
CITY_007:240
```

City 2:

```
* CAN SYSTEM DEMO *
CITY_007:102
```

7.4.3.4 Ungrouping a group or restoring the Cities' original addresses

The King sent King Page 3 to group 007 to ungroup the group. This test also demonstrated that the King could talk to a group through the group address.

The values of King Page 3 sent to group 007 were as follows:

Page 3
Line 0: 07 (group 007)
Line 1: 03 (Page 3)

Expected and actual result: The Cities operated as specified. This was done by repeating the testing in Section 7.4.2

The display in the LCD of City 3 appeared as follows:

City 1:

```
* CAN SYSTEM DEMO *
CITY 001:140
```

City 2:

```
* CAN SYSTEM DEMO *
CITY 002:008
```

7.4.3.5 Changing baud rate

Initially, the system was working at 125kbit/s. In order to change the baud rate of the system, the King broadcasted King Page 4 to all Cities with a new baud rate setting.

For example, setting 100kbit/s to the system; the values of King Page 4 were as follows:

Page 4

Line 0: 00 (Broadcast message)

Line 1: 04 (Page 4)

Line 2: 43 (value for Bit Timing Register 0)

Line 3: 7A (value for Bit Timing Register 1)

Expected and actual result: The Cities operated as specified. This was checked by repeating the tests in previous sections.

The values of the Bit Timing Register 0 and Bit Timing Register 1 were checked by examining their memory locations at the addresses 703FH and 704FH, respectively. This was done by a memory display command of the BUFFALO monitor program when the King Menu program returned to the monitor program. This examination is reliable due to the fact that all nodes in a CAN-based system always work at the same baud rate, and the values of the Bit Timing Registers are not changed when exiting the King Menu program.

This test was checked with different baud rates of: 50, 100, 200, 125, 250 kbit/s.

Baud rates of over 250 kbit/s, such as 500 kbit/s or 1Mbit/s, could not be performed by the system because the modified RS485 transceivers was not able to work at these baud rates.

Table 7-5 shows the values of Line 2 and line 3 of King Page 4 for the tested baud rates.

Table 7-5 Baud rate values

Baud Rate (kbit/s)	**Values for Line 2 and Line 3 of King Page 4**
250	Line 2: 41 Line 3: 67
200	Line 2: 43 Line 3: 34
125	Line 2: 43 Line 3: 67
100	Line 2: 41 Line 3: 7A
50	Line 2: 47 Line 3: 7A

7.4.3.6 Adding a new City to the system

This test illustrated that during the Run phase a new City could be connected to the system. This was done by removing a City (for example, City 1), then reconnecting it to the system. The test consisted of two stages:

Stage 1

This stage of the test demonstrated that when a City was set to a different baud rate from that of the system, it did not damage the system.

Assumption:

- The system baud rate at this stage was 200 kbit/s or any other baud rate different to 125 kbit s (initial baud rate)
- Folder 2 of City 3 was configured to receive a message with ID 10 (Extended format)

It should be noted that the address of City 1 after reset was 001.

Testing:

- Reconnecting City 1 and resetting it. The City waited for the Set-up instructions from the King.
- The King sent King Page 1 to assign ID 10 (Extended format) to the Folder 2 of City 1. The values of the King Page were as follows:

Page 1
Line 0: 01 (City 1)
Line 1: 01 (Page 1)
Line 2: 20 (Folder 2)
Line 3: 00 (Arbitration 0) MSB
Line 4: 00 (Arbitration 1)
Line 5: 00 (Arbitration 2)
Line 6: 50 (Arbitration 3) LSB
Line 7: 8C (Enable the Folder, Transmit, Extended)

- The King sent King Page 0 to City 1 to tell the City that the Set-up phase had finished. The values for the King Page were as follows:

Page 0
Line 0: 01 (City 1)
Line 1: 00 (Page 0)

Expected and actual result:

- The red LED on the City 1 was still ON. This meant the City had not received the King's instructions because it was listening to the King at a baud rate of 125 kbit/s.

- The rest of the system was in normal operation. This was tested by repeating the test at Section 7.4.2.2.

Stage 2

This stage was to test that a City could be connected to the system when the system baud rate was 125 kbit/s (initial baud rate). The following steps were taken:

- The King changed the system baud rate to 125 kbit/s by sending out King Page 4. The values of the Page were as follows:

Page 4
Line 0: 00
Line 1: 04
Line 2: 43
Line 3: 67

- Resetting City 1.

- Repeating the steps in the Testing section of Stage 1.

Expected and actual result: City 1 communicated with City 3 as specified. This was checked by repeating the test in Section 7.4.2.1.

7.4.3.7 Testing the role of the King

At the Run phase, and if there is no further requirement for the system configuration, the King can be removed from the network. It can be reconnected to the system at any time if required. The test was performed by the following steps:

- Removing the King from the network. This was done by resetting the King

Expected and actual result:
- The PC monitor returned to the BUFFALO prompt.
- The system was working normally without the King. This was checked by the test described in Section 7.4.2.

- Reconnecting the King to the network. This was done by running the King program (more details of how to use the King program are given in Appendix C).

- Performing the previous tests to check the operation of the King.

Expected and actual result: The King worked in the same way as it had done when it was originally connected to the network.

7.4.4 Testing the behaviour of the King Menu program

Table 7-6 shows the testing schemes were employed during the entering a King Page stage of the King Menu program.

Table 7-6 Testing King Menu program

Test	Expected and actual result
Entered a character which was not in the ranges of Hexadecimal numbers	Prompted the user to re-enter the data for the current Line
Pressed <Enter> to confirm a Line	Prompted the user to enter data for the next Line
Pressed <Ctrl+X> to re-enter value for a Line	Prompted the user to re-enter the data for the current Line
Entered a King Page containing 8 Lines	'Send? (Y/N):' prompt appeared
Used <Ctrl+ESC> to enter data for a King Page containing less than 8 Lines	'Send? (Y/N):' prompt appeared
Entered the character 'N' at 'Send? (Y/N):' prompt	Prompted the user to enter a new King Page
Entered the character 'Y' at 'Send? (Y/N):' prompt	'New Page? (Y/N):' prompt appeared
Entered the character 'Y' at 'Send? (Y/N):' prompt	Prompted the user to enter a new King Page
Entered the character 'N' at 'Send? (Y/N):' prompt	Return to the BUFFALO prompt appeared

7.5 Conclusion

This chapter described how the author of this thesis successfully designed the software that controlled the operation of a small system following the Small CAN Kingdom protocol. The software was used to demonstrate and test the protocol.

The Small CAN Kingdom protocol provides an open approach, which allows later designers to add more application services by means of additional King Pages in order to enhance the performance of future systems. The address of the subroutine used to decode a new King Page is simply assigned in the King Page vector table, thus the City's Mayor knows where to look for the subroutine.

Similarly, when a City implements a new Form (or Forms) for a particular Folder (Message Object), the address of the subroutine used to decode the Form (or Forms) is written into the Interrupt service vector table.

If a Folder is designed to receive several messages with different Form formats (e.g. the King Folder), a similar vector table to the King Page vector table should be used to specify the address of the subroutine used to decode a particular Form.

When a new City is designed, the City should reserve two bytes for addressing purposes: the original address (ADDRESS) can be stored in one byte non-volatile memory, and the working address (C_ADD) is stored in one byte volatile memory. Initially, the value of ADDRESS is assigned to C_ADD. The King then assigns a new City's address or a Group address to a City by changing the value of C_ADD.

Note that the value of ADDRESS should be left to the system designers to determine, in order to avoid conflict between the Cities.

Any new City following the Small CAN Kingdom protocol should be able to receive and obey at least King Page 0 and Page 1.

Typically, the software used to demonstrate the performance of the Small CAN Kingdom protocol described in this project was written in the M6800 assembly language. Each CAN node in the demonstration system contains an MC68HC11 and Intel 82527 CAN controller. The nodes with the same hardware parts can utilise similar implemented subroutines as shown in Table 7-7.

Table 7-7 Small CAN Kingdom protocol subroutine for a City

Address	**Name**	**Description**
B696H	PG_4	Subroutine to decode King Page 4
B6B3H	PG_3	Subroutine to decode King Page 3
B6BAH	PG_2	Subroutine to decode King Page 2
B6C1H	PG_1	Subroutine to decode King Page 1
B6FEH	PG_0	Subroutine to decode King Page 0
B674H	ADDRESS	1 byte EEPROM address for a City's original address
B705H	MSG_OB1	Subroutine to service Folder 1 (King's message Folder)
B730H	I_SERV	Interrupt Service Routine (ISR)
B770H	DELAY	Subroutine to delay a multiple of 50 micro secs. The multiple value is specified by index register IY
B77DH	RESET_C	Subroutine to reset CAN controller chips
B795H	INIT_C	Subroutine to initialise CAN controller chips

It should be noted that the memory locations of the vector tables (Table 7-2 and Table 7-3) as well as the memory locations used by the subroutines in Table 7-7 can be mapped anywhere in the addressable space available to the MCU. This only requires minor changes in the software.

The testing schemes have provided an efficient mechanism used to check the system's performance. All the tests were repeated many times and produced expected results.

CHAPTER 8

CONCLUSION

8.1 Summary

The Controller Area Network (CAN) protocol provides designers with a powerful mechanism to build complex distributed systems.

The Bus Arbitration concept employed by the protocol allows messages being transferred in the network to vie for contention of the bus in a predetermined way. Whenever a collision occurs, the highest priority message always gains access to the bus; the nodes which have attempted to transmit lower priority messages automatically become receivers.

A CAN network uses Linear Bus topology; thus, any node has the same right of access to the bus (Multimaster). Moreover, messages are broadcast to all nodes in a CAN-based system; therefore, any number of nodes can receive their expected data simultaneously (Multicast reception). This enables data to be exchanged in a short period of time.

The powerful error detection schemes of the CAN protocol provide the means for CAN-based systems to work in physically harsh and electrically noisy industrial environments.

In addition, CAN's OSI/ISO based model allows the protocol to achieve design transparency and implementation flexibility.

However, the CAN protocol only specifies the Data Link layer according to the ISO/OSI seven layer reference model (see Chapter 2). Upper and lower layers must be added to a CAN-based system in order to perform actual operations in a system.

With respect to the critical time requirements in the automation industries, CAN-based systems usually implement three layers including the:

- Application layer,
- Data Link layer, and
- Physical layer.

The Network, Transport, Session, and Presentation layers are thus omitted.

The main aim of this thesis is to deal with the Application layer of small distributed systems based on the CAN protocol. In other words, this thesis is concerned with the design of an Application Layer protocol (Higher Layer Protocol) for small systems using small microcontrollers such as the MC68HC11.

Steps were taken to investigate three popular CAN Higher Layer Protocols (HLPs), namely:

- Smart Distributed System (SDS),
- DeviceNet, and
- CAN Kingdom.

The study concluded that SDS provides an effective protocol for communications between I/O devices and the host controllers. The protocol, however, is too complex to suit the requirements of small systems. Additionally, the SDS application services are designed in fixed forms which means that they are extremely difficult to change to satisfy the needs of a particular system.

DeviceNet is an open network where all nodes have the same right of access to the bus. The Object-Oriented approach of DeviceNet makes it more flexible than SDS. Each object in the DeviceNet protocol can contain different services which specify the role of the object. Moreover, users can utilise the I/O messages of DeviceNet to provide special-purpose services for their devices (see Chapter 3).

However, SDS and DeviceNet protocols do not make use of many powerful features of the CAN protocol. They only employ the Standard message format (11-bit Identifier) specified in the CAN protocol. In addition, two or more nodes cannot have the same Logical address in an SDS system, or the same Media Access Control Identifier (MAC ID) in a DeviceNet system. This violates the optimum use of CAN Identifier which has been devised so that more than one node can utilise the same CAN ID. Therefore, the multicast functionality of CAN designed for the fast exchange of data cannot be applied in these protocols.

Furthermore, because the message priorities depend on the Logical Address or MAC ID of a device, they are also difficult to amend once already assigned. To do so requires a complex modification in the control software.

Another disadvantage of SDS and DeviceNet is that when in the process of designing a module, the designers must be aware of the other nodes which their module is going to communicate with. The designers also have to assign the module address in advance. In other words, the message priorities must be decided before the whole system design has been completed.

Conversely, the purpose of the CAN Kingdom protocol is to provide an open solution which enables "any module just to be hooked on the bus and then start working as a perfect teammate in the system" (Lennartsson, & Fredriksson, 1995). This idea is based on the fact that when modules following different protocols are connected on the same CAN bus, problems will arise due to the conflicting interpretation of messages at the application level. Therefore, instead of specifying a complete HLP, CAN Kingdom defines a set of protocol primitives which designers can use to build a final HLP to suit their needs.

The principle of the CAN Kingdom protocol is that a master node in the system, the King, owns all message Identifiers, then assigns them to messages transferred in the system during a set-up phase. Normally, the King does not get involved in the run-time of the system. Yet, it can send instructions to the Cities (CAN nodes) at any time while the system is working, should the configurations need to be changed. This approach enables the message priorities to be changed easily when required.

In addition, designers are free to construct the message formats for their devices. There is no restriction of the use of CAN data field in the CAN Kingdom protocol except for the construction of King Pages.

Furthermore, the CAN Kingdom protocol supports both Standard and Extended (29-bit Identifier) message formats specified in the CAN protocol. This enables users to employ the latest technology of the CAN protocol; and hence, the two types of messages can coexist within a CAN Kingdom system which it is not possible in SDS or DeviceNet systems.

In light of the advantages of the CAN Kingdom protocol for designing an open protocol, the author of this thesis decided to follow its basic idea to develop a HLP (the Small CAN Kingdom protocol) for small CAN-based distributed systems.

This small HLP has inherited the advantages of the CAN Kingdom protocol; and thus, it provides users with the freedom to design a final protocol that suits their system requirements. The data formats of messages can be designed without any restriction; they must however, be conveyed to the system designer by means of Forms (see Chapter 4). This enables the system designer to inform the King to assign appropriate modules, with matching Forms, for the exchange of information.

Besides the hereditary benefits of CAN Kingdom, the Small CAN Kingdom protocol also has its own advantages such as its simpler specification and small size of programming codes. Indeed, the software, which enables a City to receive and decode all the five King Pages, implemented in the protocol, requires only 362 bytes for the service subroutines, and 69 bytes for the variables and the two vector tables (see Chapter 7). To simplify matters further, the subroutines for the King Pages from Page 2 to Page 4 can be omitted if not required in particular systems.

Additionally, the implementation of the Small CAN Kingdom protocol allows users to add more services to enhance the system's performance by means of extra King Pages. The addresses of the subroutines to decode the new King Pages are simply written to their appropriate location in the King Page vector table (Table 7-3).

This thesis also describes how a network of MC68HC11 microcontrollers has been designed and successfully implemented. The software controlling the network has been based on the Small CAN Kingdom protocol. The system design includes:

- A Capital (the Master node) with the King being responsible for the system configurations.

- City 1 which sends Analog / Digital (A/D) converted signals along with its address to City 3 whenever the analog signal value changes,

- City 2 which also sends A/D signals and its address to City 3, but only if it receives requests from City 3 by means of CAN Remote frames, and

- City 3 which receives and displays the information from both City 1 and City 2 on a Liquid Crystal Display (LCD) module. This City has also designed to send Remote frames to City 2 when the data is needed.

The King is connected to an IBM PC so that the designer can construct a King Page, then the Page is downloaded to the King and sent to the network to perform the system configuration.

All Cities are able to receive and follow the instructions, contained in the five King Pages, associated with the Small CAN Kingdom protocol.

The LCD in City 3 provides a visual way to demonstrate the system operations. City 1 and City 2 send their information along with their addresses; thus, it is easy to observe the derivation of the displayed message. In addition, the system can demonstrate the ability of CAN data frames for requesting data.

Each City also contains a dual-colour Light Emitting Diode (LED) to indicate its states (Set-up or Run phase). Red indicates the Set-up phase; green indicates the Run phase.

Chapter 5 discussed reasons for choosing the MC68HC11 microcontroller and the Intel 82527 CAN controller. This resulted in the selection of suitable physical components for the hardware design in this project. The details of the hardware and software development environments were also described in this chapter.

In order to take advantage of Motorola's hardware support, the M68HC11EVBU evaluation boards were used to assist the hardware design of the system.

The software of the project was developed and debugged with the aid of the following programs:

- The ROM-based BUFFALO monitor program, which is included in the microcontroller chip's ROM, and was used to load and debug software.
- The Edit (MS-DOS line editor) program which was used to write the source programming codes.
- Motorola's portable assembler (PASM) used to compile the program.
- Motorola's Ubuilds program used to create Motorola S-records, and
- The MS-Kermit program employed to establish communication between an IBM PC and the M68HC11EVBU board's BUFFALO monitor program.

The complete hardware and software designs of the system were described in Chapter 6 and Chapter 7, respectively.

In Chapter 6, the interfaces between following devices were successfully designed:

- The MC68HC11 microcontroller and the Intel 82527 CAN controller chip (for all CAN nodes in the system).

- The MC68HC11 microcontroller and an IBM PC (for the King).

- The MC68HC11 microcontroller and A/D devices (for City 1 and City 2). In the case of this project, 10kΩ rotary potentiometers and a +5V supply were used to generate A/D signals.

- The Intel 82527 and a DS3695 (RS485 standard) transceiver chip (for all CAN nodes). The DS3695 chip was modified to suit the requirements of the CAN bus.

- An LCD module and the Intel 82527 (for City 3). The two general purpose I/O ports (Port 1 and Port 2) of the Intel 82527 were used to control the LCD.

- A push button and MC68HC11 (for City 3). This push button was used to invoke the generation of CAN Remote frames to City 2 to request data.

- A dual-colour LED and the Intel 82527 (for all Cities). The LED contained in each City was used to indicate the City's state.

In Chapter 7 described the design of all the software necessary for the King to construct and transmit its instructions and for the Cities to obey these instructions. The software required for each node to complete its specified tasks in the system was also been presented in detail.

- The King Pages were designed on an IBM PC, then downloaded to the King, and subsequently, sent to the Cities via the King's software. An Interrupt Service Routine (ISR), contained in the King, enabled it to reconfigure itself, if necessary, when a King Page had been successfully transmitted.

- When a King Pages arrived at a City, the Intel 82527 interrupted its host microcontroller. This forced the City's ISR to determine the interrupt source and to invoke an appropriate subroutine to service the King Page.

- During run-time, City 1's application software detected the change of the A/D signal, then ordered the Intel 82527 to send the amended information to City 3.

- City 2 always updated its information and sent to City 3 whenever it received a remote request.

- The software in City 3 captured the logical level change at the push button. Whenever, the push button was pressed, it sent a Remote frame to City 2 to request data. The City 3's ISR was also able to invoke the right subroutines to service the messages from City 1 and City 2.

The testing schemes in Chapter 7 provided sufficient checks for the system's performance. The following tests were accomplished:

- Testing the responsibilities of the King and the Cities in the Set-up phase. In this phase the King sent the configuration instructions to the Cities and decided upon the communications between the Cities (through the King Page 1). The Cities started to work when the King told them that the Set-up phase had been finished (through broadcasting the King Page 0 to all Cities).

- Testing the communications between Cities in the Run phase.

- Changing the message Identifiers to change the priority for the messages (through the King Page 1).

- Changing the Cities' addresses (through the King Page 2).

- Assigning a group address to Cities (through the King Page 2). This enabled the King to talk to a group of Cities at a time.

- Ungrouping a group of Cities or restoring the original Cities' addresses (through the King Page 3)

- Changing the system baud rate (through the King Page 4). This test was done successfully with different baud rates: 50, 100, 125, 200, 250 kbit/s. The baud rates above 250 kbit/s could not be performed because the modified DS3695 transceiver chips were not able to work at these baud rates.

- Adding a new City to the system. This test was done by removing a City from the system, then reconnecting it.

- Testing the role of the King. This test was done by removing the King from the system, then reconnecting it. This was to prove that the system operated normally without the King, and when the King was reconnected, it was able to send new instructions to the Cities.

In addition, another test was successfully performed to check the behaviour of the King Menu program (see Table 7-6). This program enabled the system designer to construct the King Pages and to correct any mistyping during entry of a King Page.

All the tests were repeated several times and the software was debugged whenever necessary in order to achieve accurate performance of the system.

8.2 Future trends and suggestions

The low cost of CAN components, high data integrity, and short reaction time of the CAN protocol together with its huge user base ensure that the future of the protocol is a bright one. However, the lack of a worldwide standard CAN Higher Layer Protocol has been a substantial issue. As a result, each manufacturer may either choose an existing apropos protocol or develop their own for their systems.

In this case, the open approach introduced in the CAN Kingdom protocol is likely to be an advantageous mechanism which can be used to produce a final suitable protocol for the requirements of different types of systems.

The Small CAN Kingdom protocol was developed based on the principle of CAN Kingdom, and hence, the users can customise the protocol to suit their needs. Furthermore, its simplicity and small size of programming codes are an ideal solution for small distributed systems.

Another CAN project concerned with the Intel 8051 microcontrollers is also currently in progress at Edith Cowan University, Perth, Western Australia. Such associated research in the CAN area also offers exciting prospects for the Engineering faculty of Edith Cowan University.

Although the software for this protocol was implemented and tested utilising an MC68HC11 microcontroller system, systems with different hardware components can be hooked on to a Small CAN Kingdom based system, as long as the implementation of their software supports the King's instructions.

The software implementation of the protocol enables later designers to enhance the system's performance by means of additional King Pages. For example, a new King Page to change the mask register so that a Message Object (or Folder) can transmit or receive a group of messages, or a King Page to set up the system clock for the network would be highly beneficial.

The Small CAN Kingdom protocol does not actually specify the exchange of data larger than 8 bytes. However, this can be done in a similar way to the transfer of King Pages.

It is noted that the memory location for the program, as well as the vector tables in each CAN node, can be mapped anywhere in the addressable space available to the microcontroller with only minor changes in the software (see Chapter 7).

The King Menu program, which enables the King Pages to be constructed, could be considered as visually unimpressive. The time restrictions of this project effectively serve to promote this recommendation a future agenda. The program could be written using a Graphic User Interface (GUI) environment (e.g. Windows) which would make it easier for users to design a King Page. In this case, a PC CAN board can be utilised to provide more powerful control facilities for the King.

A further observation is that the King Menu program could be designed to allow all King Pages to be created at an initial stage, and then sent sequentially to the Cities.

In addition, if the system's configuration is rarely changed, the Cities can store the King's instructions in non-volatile memory so that the system does not need to be reconfigured each time the system is power up.

In conclusion, this project has provided a solution for small distributed microcontroller systems to operate as a part of a powerful industrial network architecture: the Controller Area Network. It has also opened up an exciting research field in the automation industries.

REFERENCES

82527 Serial Communications Controller Architecture Overview. (1996, January). Intel Corporation.

Baba, M. D., Ekiz, H., Kutlu, A., & Powner, E. T. (1996). Toward adaptable distributed real-time computer systems. *Proceedings of the Third International Workshop on Real-Time Computing Systems and Applications* (pp. 170-175).

Benzekri, A., Bruel, J. -M., Fuertes, J. M., & Juanole, G. (1997). Controller area network: a formal case study. *Proceeding of 1997 IEEE International Workshop on Factory Communication Systems* (Vol. 1, pp. 365-372).

Blandin, J., Bradley, S., Danioux, R., Gray, P., & Loaic, G. (1997). A network architecture concept for deep ocean lander systems. *Technology Transfer from Reaseach to Industry, Seventh International on Electronic Engineering in Oceanography* (pp. 30-33).

Boyce, C. R. (1988, December). A four-station controller area network. *IEE Colloquium on Vehicle Networks for Multiplexing and Data Communication* (pp. 9/1-9/7). London.

C167CR 16-bit CMOS single-chip microcontroller data sheet (1995, June). Siemens.

CAN and DeviceNet (n.d.) [on-line]. Available WWW: http://www.industry.net/c/orgunpro/odva/over6 [1997, September 20].

CAN Kingdom 3.01 Specification. (1996-1997). Kvaser AB.

CAN Specification Version 2.0. (1991). Robert Bosch GmbH.

Cena, G., & Valenzano, A. (1995, October). A distributed mechanism to improve fairness in CAN networks. *Proceeding of 1995 IEEE WFCS '95 International Workshop on Factory Communication Systems* (pp. 3-11). Leysin, Switzerland.

Cena, G., Demartini, C., & Durante, L. (1996). Communication service and protocol specification using object oriented analysis. *Proceeding of the IEEE ISIE '96 International Symposium on Industrial Electronics* (Vol. 2, pp. 1043-1048).

Chen, J., Rabb, M., & Taylor, V. (1996, February). Bridge: A retargetable extensive profiling tool. *Proceedings of the Fourth International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication, MASCOTS '96* (pp. 44-50).

Croft, A. (1996). The XK8 high speed powertrain serial communication system. *IEE Colloquium on The Electrical System of the Jaguar XK8* (Digest no. 1996/281, pp. 3/1-3/17).

Cross, R., & Watson, T. (1994). *The Professional Analysis* (2nd ed.). Anderson Press, Australia.

Crowcoft, J., Hailes, S., Handley, M., Jena, A., Lewis, D., & Wakeman, I. (1993, April). Some multimedia traffic characterisation and measurement results. *Fourth IEE Conference on Telecommunication* (pp. 167-174).

DeviceNet 2.0 Specification. (Vol. 1, 1997, February). Allen-Bradley.

DeviceNet 2.0 Specification. (Vol. 2, 1997, February). Allen-Bradley.

Dickson, G., & Lloyd, A. (1992). *Open Systems Interconnection*. Prentice Hall, Australia.

Dodds, G., Beattie, W. C., & Schofield, R. D. (1989, September). *Microcontroller eurocard development. IEE Colloquium on Eurocard Computers – A Solution to Low Cost Control* (Digest no. 107, pp. 4/1 - 4/10).

DS3695/DS3695T/DS3696/DS3696T/DS3697/DS3698 Multipoint RS485/RS422 Tranceivers/Repeaters. (1996). National Semiconductor Corporation.

Ekiz, H., Kutlu, A., Powner, E. T., Li, G. -J., Hsu, D. F., Horiguchi, S., & Maggs, B. (1996, June). Design and implementation of a CAN/CAN bridge. *Proceedings of the second International Symposium on Parallel Architectures, Algorithms, and Networks* (pp. 507-513). Beijing, China.

Farsi, M. (1995). Application of a PLC as a cell controller using a communication network. *IEE Colloquium on Application of Advanced PLC (Programmable Logic Controller) Systems with Specific Experiences from Water Treatment* (Digest no. 1995/112, pp. 3/1-3/4).

Farsi, M., & Ratcliff, K. (1997). CANopen: the open communication solution. *Proceeding of the ISIE '97 IEEE International Symposium on Industrial Electronics* (Vol. 1, pp. 112-116).

Gollmer, K., & Posten, C. (1994, August). Flexible automation of a biotechnological pilot-plant with real-time network. *Proceedings of the Third IEEE Conference on Control Application* (Vol. 3, pp. 1941-1946). Glasgow, UK.

Greenfield, J. D. (1992). *The 68HC11 Microcontroller*. Florida: Saunders College Publishing.

Halsall, F. (1996). *Data Communications, Computer Networks and Open System* (4th ed.). Addision-Wesley Publishers Ltd, USA.

Hands, D.H. (1997). *A Design in Interfacing the MC68HC11 to the AMD AM29F010 flash memory chips*. Unpublished B.Eng. (Hons) thesis, Edith Cowan Universty, Perth, Western Australia.

Hawryszkiewycz, I. T. (1994). *Introduction to Systems Analysis and Design* (3rd ed.). Prentice Hall, Australia.

HC11 M68HC11 E Series Technical Data. (1993). Motorola Inc.

HC11 M68HC11 Reference Manual. (3rd rev. ed.) (1991). Motorola Inc.

HC11 MC68HC11E9 Programming Reference Guide. (1990). Motorola Inc.

Henshall, J., & Shaw, S. (1988). *OSI Explained: End-to-end computer communication standards*. Ellis Harwood Ltd., England.

Hughes, L. (1992). *Data Communication*. McGraw-Hill Inc., USA.

Introduction to the controller area network (CAN) protocol. (1993, September). Intel Corporation.

Khan, A. R. (1996, October). Workhorses of the electronic era. *IEEE Spectrum, 10* (33), 36-42.

Kirk, B. R. (1996). Improve machine performance using the controller area network-the automation of a high-speed can production line. *IEE Colloquium on Mechatronics in Automated Handling* (pp. 4/1-4/7).

Korance, K. J. (1996, September 12). Mobile machines get CAN in gear. *Machine Design, 68* (16), p. 50. Penton Publishing.

Kutlu, A., Ekiz, H., & Powner, E. T. (1996, June). Performance analysis of MAC protocols for wireless control area network. *Proceedings of the second International Symposium on Parallel Architectures, Algorithms, and Network* (pp. 494-499). Beijing, China.

Kvaser CAN Pages: The CAN protocol (n.d.) [on-line]. Available WWW: http://www.kvaser.se/can/protocol/index.htm [1997, September 20].

Lennartsson, K., & Fredriksson, L. -B. (1995). Fundamental parts in SDS, DeviceNet and CAN Kingdom, a brief comparison. *The Second Inernational CAN Conference*.

Liquid Crystal Display Module L4042 User Manual. (1st ed.) (1988, April). Seiko Instruments Inc., Japan.

M68HC11EVBU Universal Evaluation Board User's Manual. (2nd ed.) (1992, September). Motorola Inc.

Maskell, D. L., & Grabau P. J. (1998, May). A multidisciplinary cooperative problem-based learning approach to embedded systems design. *IEEE Transaction Eduction 41 (2)*, 101-103.

McLaughlin, R., Tang, K. H., Moyne, J., & Shah, J. (1997, October). DeviceNet conformance testing procedures and experiences. *Proceedings of the fourth International CAN Conference.*

McLaughlin, R., Tang, K. H., Moyne, J., & Shah, J. (1997, October). DeviceNet conformance testing procedures and experiences. *Proceedings of the Fourth International CAN Conference*. Berlin.

McLaughlin, R. T. (1993, March). The immunity to RF interference of a CAN system. *IEE Colloquium on Integrity of Automotive Electronic Systems* (pp. 4/1-4/8). London.

Moyne, J., Shah, J., McLaughlin, R. & Tang, K. H. (1997, October). DeviceNet modeling on DeviceNet network. *Proceedings of the fourth International CAN Conference.*

Moyne, J., Shah, J., McLaughlin, R., & Tang, K. H. (1997, October). DeviceNet modeling on DeviceNet network. *Proceedings of the Fourth International CAN Conference*. Berlin.

Mustafa, M. A. (1994). *Microcomputer Interfacing and Applications* (2nd ed.). London: Newnes.

Nunemacher, G. (1990). *LAN Primer: An Introduction to Local Area Networks.* M&T Publishing Inc., USA.

PCA82C250 CAN controller interface specification. (1997, October). Philips Semiconductors.

SAE 81C90/91 stand-alone Full-CAN controller data sheet (1997, January). Siemens.

SDS Component Modeling Specification. (1995, May). Micro Switch Division, Honeywell Inc., USA.

Serodio, C., Cunha, J. B., Cordeiro, M., Valente, A., Morais, Salgado, P., & Couto, C. (1997). MNet-DACS:Multi-level network data acquisition and control system. *Proceedings of the IEEE ISIE '97 International Symposium on Industrial Electronics* (Vol. 1, pp. 39-43).

Simonye, S., Alpena, L., & Witte, G. (1997). Applying DeviceNet in motor control centers at a cement plant. *IEEE/PCA XXXIX Conference Record of Cement Industry Technical Conference* (pp. 113-123).

Smart Distributed System Application Layer Protocol Specification Version 2.0. (1996, November). Micro Switch Division, Honeywell Inc., USA.

Tanenbaum, A. (1988). *Computer Network* (2nd ed.). Prentice-Hall Inc., Singapore.

Tindell, K.W., Hansson, H., & Wellings, A. J. (1994, December). Analysing real-time communication: controller area network (CAN). *Proceedings of Real-Time Systems Symposium* (pp. 259-263). San Juan, Puerto Rico.

Van Woerden, J. A., Nelisse, M. W., Perricos, C., Jackson, R. D., Davies, B., Hibberd, R. D., & Banerjee, D. (1994, October). M3S - a standard communication architecture for rehabilitation applications. *Computing & Control Engineering Journal* (Vol. 5 5, pp. 213-218). UK.

Voskamp, E., & Rosenstiel, W. (1996, March). Error detection in fault secure controllers using state encoding. *Proceedings of ED&TC '96 European Design and Test Conference* (pp. 200-204).

Wargui, M., Rachid, A., de Sario, M., Maione, B., Pugliese, P., & Savino, M. (1996). Application of controller area network to mobile robots. *Eighth Meditarranean Electrotechnical Conference MELECON '96* (Vol. 1, pp. 205-207).

Wellstead, P. E. (1994, November). Automotive control overview. *IEE Colloquium on Automotive Applications of Advanced Modelling and Control* (pp. 1/1-1/5). London.

Wetton, M. (1995). *A proposal for a development platform for microcontroller-based devices*. Unpublished master thesis, Edith Cowan Universty, Perth, Western Australia.

Young, K., McLaughlin, R., & Khoh, S. B. (1995, October). DeviceNet interoperability and compliance. *Proceedings of the second International CAN Conference.*

Young, K., McLaughlin, R., & Khoh, S. B. (1995, October). DeviceNet interoperability and compliance. *Proceedings of the Second International CAN Conference*. London.

Young, K. W. (1995). A fieldbus approach to robotic systems reconfiguration. *IEE Colloquium on Fast Reconfiguration of Robotic and Automation Resources* (pp. 5/1-5/4).

Zuberi, K. M., & Shin, K. G. (1995, April). Non-preemptive scheduling of messages on controller area network for real-time control applications. *Proceedings of Real-Time Technology and Application Symposium* (pp. 240-249).

Zuberi, K. M., & Shin, K. G. (1996). Real-time decentralized control with CAN. *Proceedings of 1996 IEEE EFTA '96 Conference on Emerging Technologies and Factory Automation* (Vol. 1, pp. 93-99).

APPENDIX A

This Appendix provides details of the software development environment used in this project. All the programs and the software for the Small CAN Kingdom system were stored on a floppy disk. This disk was also used as a boot-disk. The author found that the MS-DOS mode in Windows 95 was satisfactory to develop the system.

Besides the system boot files contained in the floppy disk, the disk consists of the following directories and files:

Kermit: the directory was used to store the MS-Kermit program.

Kingdom: the directory was used to store the software for the Small CAN Kingdom system. The descriptions of the files contained in this directory are covered in Appendix D.

Pasm: the directory was used to store the Motorola portable assembler (PASM) and the Ubuilds program.

Mytemp and **Tmp**: the two directories were used to store temporary files required during the operation of the PASM assemble.

Autoexec.bat: the batch file set up the necessary configuration for the environment. The content of this file is as follows:

```
@ECHO OFF
prompt $p$g

CD A:\Kingdom
PATH=A:\;A:\KERMIT;A:\PASM;A:\TMP;
set tmp=A:\mytmp
```

Asm.bat: this batch file was used to invoke the PASM program (Pasmhc11.exe) to compile a source programming code. The content of this file is as follows:

```
A:\PASM\pasmhc11 -dxs -l %1.lst %1.asm
```

B.bat: this batch file was used to invoke the Ubuilds program (Ubuilds.exe) to build a Motorola S-record. The content of this file is as follows:

```
A:\pasm\ubuilds %1.o
```

K.Bat: this batch file was used to invoke the MS-Kermit program (Mskermit.exe) to establish communication between an IBM PC and an EVBU board. The content of this file is as follows:

```
A:\kermit\mskermit
```

T.bat: this batch file was used to invoke the "Type" command of MS-DOS to download an S-record to an EVBU board. The content of this file is as follows:

```
type %1.mx > com2
```

It should be noted that the IBM PC serial port 2 was used in this project. If another port is used in later designs, the "com2" command must be changed to the appropriate port.

Edit.com: an MS-DOS text editor program was used to write the source programming codes.

The steps concerned with compiling and downloading a program (for example, King_pro.asm in the Kingdom directory) to the EVBU board are described as follows:

After booting the IBM PC by the floppy disk, the MS-DOS prompt is as follows:

```
A:\Kingdom>
```

The King_pro.asm program was written by using the Edit program. To compile the program, at the MS-DOS prompt, type:

```
A:\Kingdom>Asm King_pro
```

The King_pro.o (the object file) is generated by the PASM assembler. To create an S-record for the King_pro.o (King_pro.mx file), at the MS-DOS prompt, type:

```
A:\Kingdom>B King_pro
```

After the creation of the King_pro.mx (S-record file), the program can now be downloaded to an EVBU board with the use of the Kermit program and the MS-DOS "Type" command. The procedure of how to download an S-record to an EVBU board is described in the M68HC11 EVBU Universal Evaluation Board User's Manual (1992).

The Kermit program can be invoked by typing:

```
A:\Kingdom>K
```

To download the King_pro.mx to the EVBU board, at the MS-DOS prompt, type:

```
A:\Kingdom>T King_pro
```

APPENDIX B

This Appendix provides the pin connection tables which were used for wire-wrapping in the hardware design, and the diagrams which show the pin layout of the chips.

Table B-1 Interface between MC68HC11 and Intel 82527

MC68HC11 Pin Function	EVBU Pin Number	Intel AN82527 Pin Number	Intel AN82527 Pin Function	74LS138 Pin Number	74LS20 Pin Number
AS	4	5	AS	-	-
R/W#	6	7	R/W#	-	-
E	5	6	E	-	-
A15/PB7	35	-	-	3	9-10-12-13
A14/PB6	36	-	-	2	4
A13/PB5	37	-	-	1	2
A12/PB4	38	-	-	-	1
-	-	8	CS#	12	6
-	-	-	-	-	8-5
-	-	-	-	4-5-8→GND	7→GND
-	-	-	-	6-16→V_{CC}	14→V_{CC}
AD0/PC0	9	4	AD0		
AD1/PC1	10	3	AD1		
AD2/PC2	11	2	AD2		
AD3/PC3	12	43	AD3		
AD4/PC4	13	42	AD4		
AD5/PC5	14	41	AD5		
AD6/PC6	15	40	AD6		
AD7/PC7	16	39	AD7		
IRQ#	19	24	INT#		
PA6	28	29	RESET#		

Note that the 74LS20 chip and 74LS138 chip were not used on the same board.

Table B-2 Interface between Intel 82527 and DS3695

Intel AN82527 Pin Number	Intel AN82527 Pin Function	DS3695 Pin Number	DS3695 Pin Function
1	VCC		
20	VSS2		
23	VSS1		
44	Mode0		
30	Mode1		
18	XTAL1		
19	XTAL2		
26	TX0	4	DI
25	TX1	3	DE
22	RX0	1	RO
		8	VCC
		5	GND
		6	DO/RI
		7	DO#/RI#

Table B-3 Interface between Intel 82527 and L2012

Intel AN82527 Pin Function	Intel AN82527 Pin Number	L2012 Pin Number	L2012 Pin Function
P1.0	38	7	DB0
P1.1	37	8	DB0
P1.2	36	9	DB0
P1.3	35	10	DB0
P1.4	34	11	DB0
P1.5	33	12	DB0
P1.6	32	13	DB0
P1.7	31	14	DB0
P2.6	11	6	E
P2.5	12	4	RS
P2.4	13	5	R/W#
		1	VSS
		2	VDD
		3	VLC

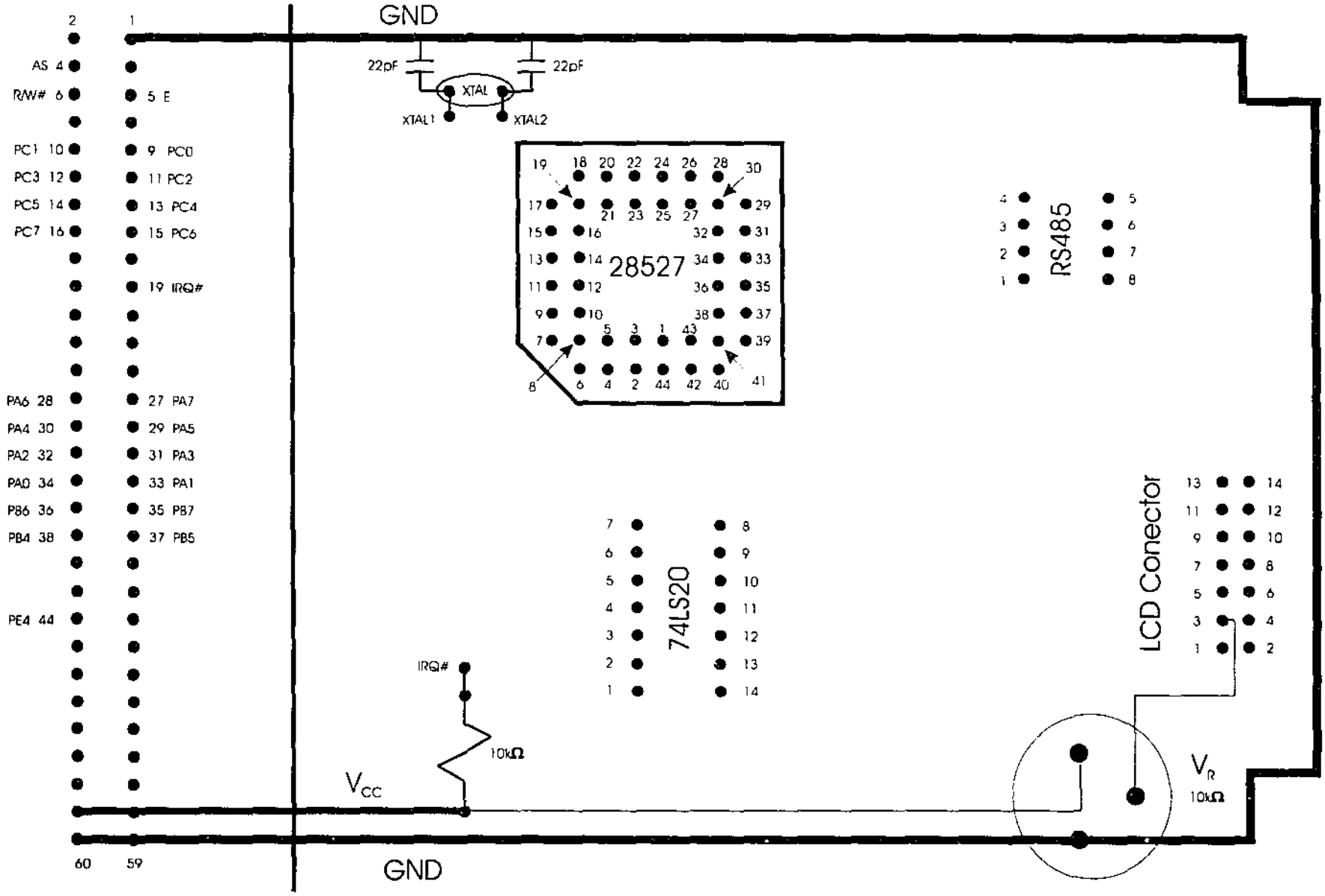


Figure B-1 Pin Layout 1

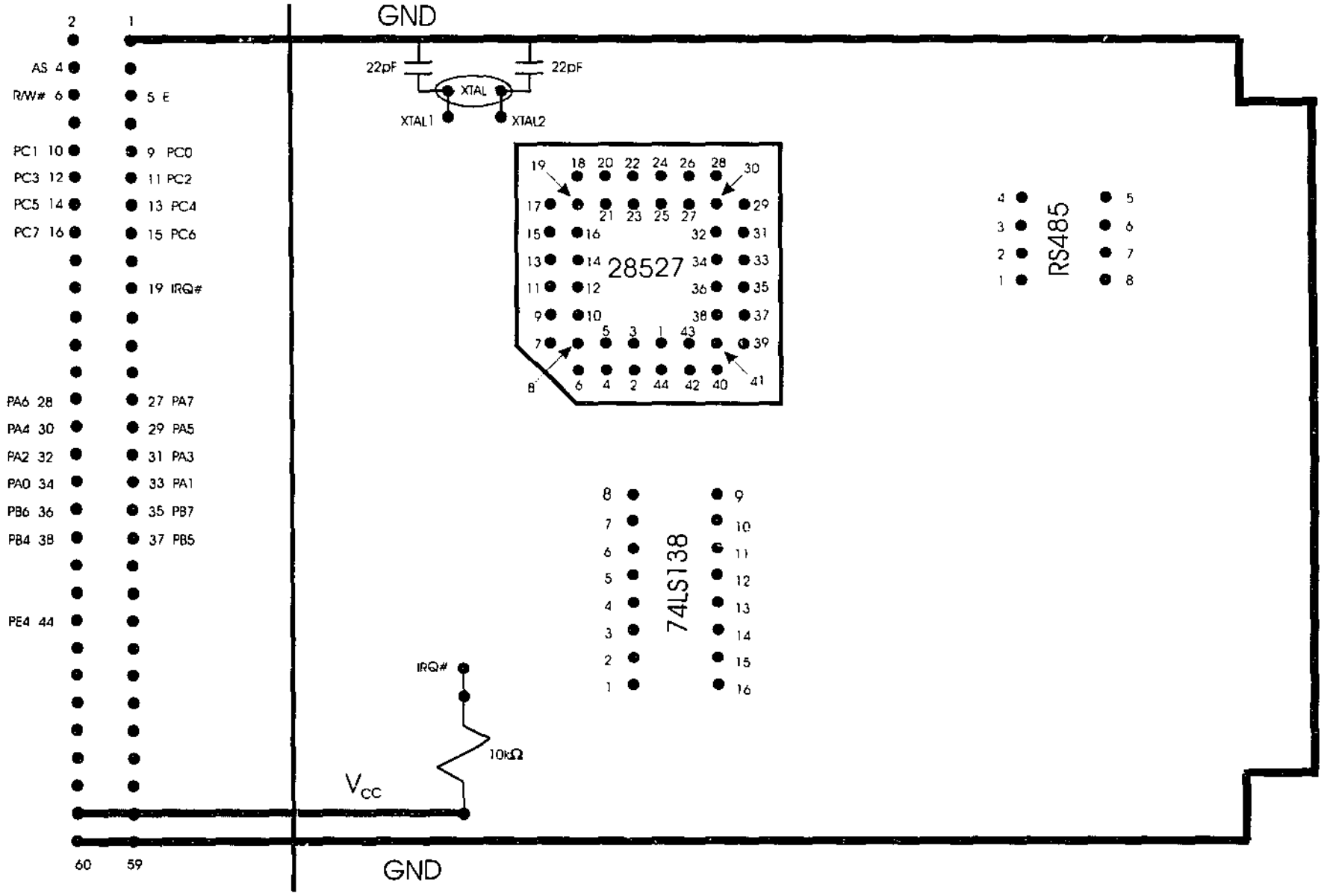


Figure B-2 Pin Layout 2

APPENDIX C

This Appendix describes how to use the King in configuring the network.

The King program starts at the address B600. At power-up, the designer needs to set up the baud rate for the King by using the memory modify command of the BUFFALO monitor program (MM command) in order to assign the baud rate values to BTR0 and BTR1 (see Chapter 7).

Note that BTR0 and BTR1 are located at the address 0100H and 0101H, respectively. These variables need to be assigned values only at power-up. When the King changes the system baud rate, it will write the new baud rate values to these memory locations by itself.

For example, to assign the initial baud rate to the King (125 kbit/s), at the BUFFALO prompt, type:

```
> mm 100
0100 43

>mm 101
0100 67
```

Note: 43 and 67 are the hexadecimal numbers and are the values for the baud rate 125 kbit/s (see Table 7-5).

The King program is invoked by the "go" command of the monitor program, at the BUFFALO prompt, type:

```
>g b600
```

The menu of the King Menu program will appear and enable the designer to enter values for King Pages and send them to Cities as described previously in Chapter 7.

The King Menu program also provides facilities to users to correct data during entering a King Page. The facilities can be summarised as follows:

- If the current character being entered is not in the ranges of Hexadecimal characters, the current Line is discarded. The program repeats this Line to prompt the user to re-enter the data.

- After entering data for a Line, the user presses <Enter> to confirm the Line. The monitor will prompt the user to enter values for the next Line.

- Data for a Line can be changed by pressing <Ctrl+X> instead of <Enter>.

- For a King Page less than eight Lines, after pressing <Enter> to confirm the value of the last Line, the user presses <Ctrl+ESC> to ignore the remaining Lines.

APPENDIX D

This Appendix provides the complete program listing of the software designed in this project.

It is noted that due to the restriction of memory, the author had to utilise all the memory space available for user applications of the EVBU boards.

The software for the whole system is stored in the Kingdom directory (see Appendix A), and it includes the following files:

- **King_pro.asm** contains the software to control the tasks of the King.
- **City_rou.asm** contains all the subroutines necessary for a City to follow the Small CAN Kingdom protocol (see Table 7-7).
- **AD1_City.asm** is the software to control City 1
- **AD2_City.asm** is the software to control City 2
- **AD_rou.asm** contains the ASCII subroutine (see section 7.3.6.4)
- **LCD_City.asm** is the software to control City 3
- **LCD_rou.asm** consists of a set of subroutines to control the LCD module

The contents of the above files are listed as follows:

```
*****************************************************************************
*    This program is to enter values for a King Page.                       *
*    File          : KING_PRO.ASM                                           *
*    Author        : Long Giang Nguyen                                      *
*    ID            : 0959416                                                *
*    Date          : 21-5-1998                                              *
*    Last modified: 12-6-1998                                               *
*****************************************************************************

*** EQU ***
MONITOR EQU     $E0BF           ; Start of BUFFALO program
.INCHAR EQU     $FFCD
.UPCASE EQU     $FFA0
.OUTCRL EQU     $FFC4
.OUTSTR EQU     $FFC7
CTRLX   EQU     $18             ; Change line's value
CTRLESC EQU     $1B             ; Exit entering King Page
ENTER   EQU     $0D             ; Confirm value
EOT     EQU     $04

PORTA   EQU     $1000
BREG    EQU     $7000
KVAL    EQU     BREG+$17        ; First Data byte of Msg Obj 1

        ORG     $00
MSG0    FCC     'New page?(Y/N):'
        FCB     EOT

*** Assign address for ISR
        ORG     $00EE
        JMP     I_SERV

*** Variables for the program ***

        ORG     $100

BTR0    RMB     1               ; Bit Timing Register 0
BTR1    RMB     1               ; Bit Timing Register 1

TMP1    RMB     1               ; HEX Byte
PTR1    RMB     2               ; Pointer to message
PTR2    RMB     2               ; Pointer to value
NHEX    RMB     1               ; Indicate not HEX
STOP    RMB     1
COUNT   RMB     1               ; The number of byte of the King Page

M_OBJ   RMB     1
I_PTR   RMB     2               ; Pointer to ISR
I_ADD   RMB     2               ; Address of ISR
P_PTR   RMB     2               ; Pointer to King Page
P_ADD   RMB     2               ; Address of King Page
OBJ_NO  RMB     2               ; Msg Obj No
```

```
MSG1    FCC     '*** KING PAGE ***'
        FCB     EOT

MSG2    FCC     'LINE 0: '
        FCB     EOT
MSG3    FCC     'LINE 1: '
        FCB     EOT
MSG4    FCC     'LINE 2: '
        FCB     EOT
MSG5    FCC     'LINE 3: '
        FCB     EOT
MSG6    FCC     'LINE 4: '
        FCB     EOT
MSG7    FCC     'LINE 5: '
        FCB     EOT
MSG8    FCC     'LINE 6: '
        FCB     EOT
MSG9    FCC     'LINE 7: '
        FCB     EOT
MSG10   FCC     'Send?(Y/N):'
        FCB     EOT

** I_SERV: Interrupt service routine for receiving a msg

I_SERV  EQU     *

SERV    LDAB    BREG+$5F        ; Determine the source of INT
        DECB
        LSLB
        ADDB    #$E0            ; Calculate the service address
        LDAA    #$01
        STD     I_PTR
        LDX     I_PTR
        LDD     0,X
        STD     I_ADD
        LDX     I_ADD
        JSR     0,X

        LDAA    BREG+$5F        ; Determine the source of INT
        CMPA    #$1             ; to allocate the base address
        BEQ     YYY             ; of Msg Obj
        CMPA    #$2
        BEQ     MSG15

MSG_N   SUBA    #$2
        LSLA
        LSLA
        LSLA
        LSLA
        STAA    M_OBJ
        BRA     XXX

MSG15   LDAA    #$F0
        STAA    M_OBJ

XXX     LDX     #BREG
        LDAB    M_OBJ
        ABX
        LDAA    #%11111101      ; Reset IntPnd
        STAA    0,X
YYY     RTI
```

```
*** Start program in EEPROM ***

*** MAIN PROGRAM ***
        ORG $B600

START   LDS     #$47
        JSR     RESET
        JSR     INIT_C

        CLI                     ; Enable Int

        LDD     #MSG_OB1        ; Assign values for the vector table
        STD     $01E4           ; Service for Msg Obj 1
        LDD     #NO_OP          ; Ignore the page 0 to page 3
        STD     $01C0
        STD     $01C2
        STD     $01C4
        STD     $01C6

        LDD     #B_SET          ; Assign the service for King Page 4
        STD     $01C8


R_ENTER LDAA    #%10111111      ; Invalid Msg Obj 1
        STAA    BREG+$10

        JSR     KING            ; Enter values for a King Page

        LDAA    #$08            ; Set the DLC, Transmit, Standard
        ORAA    COUNT
        STAA    BREG+$16

        LDAA    #%10111111      ; Valid Msg Obj 1
        STAA    BREG+$10

        LDAA    #%11101110      ; Send the page
        STAA    BREG+$11

        WAI                     ; Wait for interrupt

        JSR     .OUTCRL
        LDX     #MSG0
        JSR     .OUTSTR
        JSR     .INCHAR
        JSR     .UPCASE
        CMPA    #'Y'
        BEQ     R_ENTER

        JMP     MONITOR         ; Return to BUFFALO program
```

```
**** SUBROUTINES *******
*** NO_OP: This subroutine allows the King to ignore the King's Messages
* that do not need to be processed
NO_OP   EQU     *
        NOP
        RTS

*** MSG_OB1: Subroutine to decode Msg Obj 1  (King's orders)
MSG_OB1 EQU     *
        LDAB    BREG+$18
        LSLB
        ADDB    #$C0
        LDAA    #$01
        STD     P_PTR
        LDX     P_PTR
        LDD     0,X
        STD     P_ADD
        LDX     P_ADD
        JSR     0,X
        RTS



*** B_RATE: Assign Baud Rate for the King
B_RATE  EQU     *
        LDAA    BTR0            ; Set Bit Timing Registers
        STAA    BREG+$3F
        LDAA    BTR1
        STAA    BREG+$4F
        RTS

*** B_SET: This subroutine allows the King to change its baud rate
* when it changes the system baud rate
B_SET   EQU     *
        LDAA    BREG+$19
        STAA    BTR0
        LDAA    BREG+$1A
        STAA    BTR1

        LDAA    BREG+$00        ; Get value of Control Register
        ORAA    #$40            ; Set CCE bit
        STAA    BREG+$00

        JSR     B_RATE          ; Set Baud rate for the King

        LDAA    BREG+$00        ; Reset CCE bit
        ANDA    #$BF
        STAA    BREG+$00


        RTS
```

```
*** RESET: Reset CAN chip
RESET   LDAA    PORTA           ; Set PA6 to 0
        ANDA    #$BF
        STAA    PORTA

        LDY     #20             ; Delay 1ms.
        JSR     DELAY

        LDAA    PORTA           ; Set PA6 to 1
        ORAA    #$40
        STAA    PORTA
        RTS


*** INIT_C : Initialise the 82527
INIT_C  EQU     *
BEGIN   LDAA    BREG+$02        ; Load CPU Interface Register
        ANDA    #$80            ; Check if RstST bit 0
        BNE     BEGIN

        LDAA    #$41            ; Set CCE and Init bits
        STAA    BREG+$00        ; Store Control Register

        LDAA    #$40            ; Set CPU Interface Register
        STAA    BREG+$02        ; SCLK=XTAL/2, MCLK=SCLK, Disable CLKOUT

        LDAA    #$40            ; Set Bus Config Reg
        STAA    BREG+$2F        ; By pass Input comparator, use TX0 and RX0

        JSR     B_RATE          ; Set Baud Rate

        LDX     #BREG+$10       ; Reset all message control registers,
        LDAA    #$55            ; invalid all the Msg Objects
        LDAB    #$10
LOOP1   STAA    0,X
        STAA    1,X
        ABX
        CPX     #BREG+$F0+$10
        BNE     LOOP1

        LDAA    #$FF
        STAA    BREG+$06        ; Set Global Mask to Must Match
        STAA    BREG+$07        ; (Standard and Extended)
        STAA    BREG+$08
        STAA    BREG+$09
        STAA    BREG+$0A
        STAA    BREG+$0B

        LDAA    #$02            ; Reset CCE and Init bits,
        STAA    BREG+$00        ; Enable Global Interrupt

        LDAA    #$00            ; Assign ID=0 to Msg Obj 1
        STAA    BREG+$12
        STAA    BREG+$13
        STAA    BREG+$14
        STAA    BREG+$15

        LDAA    #%11101111      ; Set TXIE
        STAA    BREG+$10

        RTS
```

```
** DELAY: The multiple of 50 micro sec is specified by IY

DELAY   LDAB    #$00
XX      INCB
        CMPB    #10             ; 50 micro sec per loop
        BNE     XX
        DEY
        BNE     DELAY
        NOP
        RTS



*** KING subroutine: Display and enter values for a King Page
KING    EQU     *
        LDAA    #$00
        STAA    COUNT           ; Data Length Code = 0
        JSR     .OUTCRL
        LDX     #MSG1
        JSR     .OUTSTR
        LDD     #MSG2
        STD     PTR1
        LDD     #KVAL
        STD     PTR2
        JSR     K_DISP

        JSR     .OUTCRL
        LDX     #MSG10          ; Confirm the page
        JSR     .OUTSTR
        JSR     .INCHAR
        JSR     .UPCASE
        CMPA    #'Y'
        BEQ     EXIT            ; Start entering if not confirmed
        BRA     KING

EXIT    RTS
```

```
*** K_DISP: Display a King page to enter values

K_DISP  JSR     L_ENTER
        LDAA    STOP            ; Exit entering values
        CMPA    #CTRLESC
        BEQ     END_K
        LDAA    TMP1
        LDX     PTR2
        STAA    0,X
        LDAA    COUNT           ; Increase Data Length Code
        ADDA    #$10
        STAA    COUNT

        LDD     PTR2            ; Increase pointer
        ADDD    #1
        STD     PTR2

        LDD     PTR1            ; Increase pointer to the next message
        CPD     #MSG9
        BEQ     END_K
        ADDD    #$09
        STD     PTR1
        BRA     K_DISP

END_K   RTS



*** L_ENTER: Enter value for a line, the value is stored in TMP1

L_ENTER PSHX
        PSHA
LL      JSR     .OUTCRL
        LDX     PTR1
        JSR     .OUTSTR
        JSR     HEXBYTE
        LDAA    STOP
        CMPA    #CTRLESC
        BEQ     END_L

        LDAA    NHEX
        BNE     LL

        JSR     .INCHAR
        CMPA    #CTRLX
        BEQ     LL
        CMPA    #ENTER
        BEQ     END_L

END_L   PULA
        PULX
        RTS
```

```
**** HEXBIN: Convert Hex in acc A to Bin

HEXBIN  JSR     .UPCASE
        CMPA    #'0'
        BLT     HEXNOT
        CMPA    #'9'
        BLE     HEXNMB
        CMPA    #'A'
        BLT     HEXNOT
        CMPA    #'F'
        BGT     HEXNOT

        ADDA    #$9
HEXNMB  ANDA    #$0F
        BRA     HEXRTS
HEXNOT  LDAA    #$FF            ; Indicate not HEX
HEXRTS  RTS


*** HEXBYTE: Convert 2 Hex number into 1 byte

HEXBYTE PSHB
        PSHA
        LDAA    #$00            ; Clear Not HEX
        STAA    NHEX

        JSR     .INCHAR         ; Get upper-haft byte
        STAA    STOP
        CMPA    #CTRLESC
        BEQ     MM
        JSR     HEXBIN
        CMPA    #$FF
        BEQ     NN

        LDAB    #4
SHIFT   ASLA
        DECB
        BGT     SHIFT
        STAA    TMP1

        JSR     .INCHAR         ; Get lower-haft byte
        JSR     HEXBIN
        CMPA    #$FF
        BEQ     NN
        ORAA    TMP1
        STAA    TMP1
        BRA     MM

NN      INC     NHEX            ; Indicate not HEX

MM      PULA
        PULB
        RTS

        END
```

```
*******************************************************************************
*   This program is a set of subroutines for all CAN nodes (Except the King)*
*   File name     : CITY_ROU.ASM                                              *
*   Author        : Long Giang Nguyen                                         *
*   ID            : 0959416                                                   *
*   Date          : 16-5-1998                                                 *
*   Last modified: 8-6-1998                                                   *
*******************************************************************************

*** EQUATE ***

PORTA   EQU     $1000
BREG    EQU     $7000

*** Variables ***

        ORG     $01B0

WORK    RMB     1               ; WORK=0  Set-up, WORK=1 finish Set-up
C_ADD   RMB     1               ; City's address
M_OBJ   RMB     1
I_PTR   RMB     2               ; Pointer to ISR
I_ADD   RMB     2               ; Address of ISR
P_PTR   RMB     2               ; Pointer to King Page
P_ADD   RMB     2               ; Address of King Page
OBJ_NO  RMB     2               ; Msg Obj No
TEMP    RMB     1


*** Subroutines ***
        ORG     $B696

*** Subroutine for King Page 4
PG_4    EQU     *
        LDAA    BREG+$00        ; Get value of Control Register
        ORAA    #$40            ; Set CCE bit
        STAA    BREG+$00

        LDAA    BREG+$19        ; Set Bit Timing Reg 0
        STAA    BREG+$3F
        LDAA    BREG+$1A        ; Set Bit Timing Reg 1
        STAA    BREG+$4F

        LDAA    BREG+$00        ; Reset CCE bit
        ANDA    #$BF
        STAA    BREG+$00
        RTS

*** Subroutine for King Page 3
PG_3    EQU     *
        LDAA    ADDRESS
        STAA    C_ADD
        RTS
```

```
*** Subroutine for King Page 2
PG_2    EQU     *
        LDAA    BREG+$19        ; Get Group address
        STAA    C_ADD
        RTS

*** Subroutine for King Page 1
PG_1    EQU     *
        LDD     #BREG            ; Get the Msg Obj No
        ADDB    BREG+$19
        STD     OBJ_NO
        LDX     OBJ_NO

        LDAA    #%01111111      ; Invalid the Msg Obj
        STAA    0,X

        LDAA    BREG+$1A
        STAA    2,X             ; Store Arbitration 0
        LDAA    BREG+$1B
        STAA    3,X             ; Store Arbitration 1
        LDAA    BREG+$1C
        STAA    4,X             ; Store Arbitration 2
        LDAA    BREG+$1D
        STAA    5,X             ; Store Arbitration 3

        LDAA    BREG+$1E        ; Set up the Msg Config Reg
        ANDA    #$0C
        STAA    TEMP
        LDAA    6,X
        ANDA    #$F0
        ORAA    TEMP
        STAA    6,X

        LDAA    BREG+$1E        ; Enable/Disable MsgVal
        ORAA    #$3F
        STAA    0,X

        RTS

*** Subroutine for King Page 0
PG_0    EQU     *
        LDAA    #$1             ; Indicate the Set up phase
        STAA    WORK            ; has finished.
        RTS

*** City's address
ADDRESS RMB     1
```

```
*** Subroutine to decode Msg Obj 1 : King's orders

MSG1     EQU     *
         LDAA    #%11111101      ; Reset NewDat
         STAA    BREG+$11

         LDAB    BREG+$17
         CMPB    #$00            ; Broadcast Msg
         BEQ     PROCESS
         CMPB    C_ADD
         BEQ     PROCESS
         RTS

PROCESS  LDAB    BREG+$18
         LSLB
         ADDB    #$C0
         LDAA    #$01
         STD     P_PTR
         LDX     P_PTR
         LDD     0,X
         STD     P_ADD
         LDX     P_ADD
         JSR     0,X
         RTS
```

```
** I_SERV: Interrupt service routine to receive a msg

I_SERV  EQU     *

SERV    LDAB    BREG+$5F        ; Determine the source of INT
        DECB
        LSLB
        ADDB    #$E0            ; Calculate the service address
        LDAA    #$01
        STD     I_PTR
        LDX     I_PTR
        LDD     0,X
        STD     I_ADD
        LDX     I_ADD
        JSR     0,X

        LDAA    BREG+$5F        ; Determine the source of INT
        CMPA    #$1             ; to allocate the base address
        BEQ     YYY             ; of Msg Obj
        CMPA    #$2
        BEQ     MSG15

MSG_N   SUBA    #$2
        LSLA
        LSLA
        LSLA
        LSLA
        STAA    M_OBJ
        BRA     XXX

MSG15   LDAA    #$F0
        STAA    M_OBJ

XXX     LDX     #BREG
        LDAB    M_OBJ
        ABX
        LDAA    #%11111101      ; Reset IntPnd
        STAA    0,X
YYY     RTI


** DELAY: The multiple of 50 micro sec is specified by IY

DELAY   LDAB    #$00
XX      INCB
        CMPB    #10             ; 50 micro sec per loop
        BNE     XX
        DEY
        BNE     DELAY
        NOP
        RTS
```

```
*** RESET: Reset CAN chip
RESET   LDAA    PORTA           ; Set PA6 to 0
        ANDA    #$BF
        STAA    PORTA

        LDY     #20             ; Delay 1ms
        JSR     DELAY

        LDAA    PORTA           ; Set PA6 to 1
        ORAA    #$40
        STAA    PORTA
        RTS
```

```
*** INIT_C : Initialise the 82527
INIT_C  EQU     *
BEGIN   LDAA    BREG+$02        ; Load CPU Interface Register
        ANDA    #$80            ; Check if RstST bit 0
        BNE     BEGIN

        LDAA    #$41            ; Set CCE and Init bits
        STAA    BREG+$00        ; Store Control Register

        LDAA    #$40            ; Set CPU Interface Register
        STAA    BREG+$02        ; SCLK=XTAL/2, MCLK=SCLK, Disable CLKOUT

        LDAA    #$40            ; Set Bus Config Reg
        STAA    BREG+$2F        ; By pass Input comparator, use TX0 and RX0

        LDAA    #$43            ; Set Bit Timing Registers
        STAA    BREG+$3F        ; Define the CAN bus frequency 125 kBits/sec
        LDAA    #$67
        STAA    BREG+$4F

        LDAA    #$FF
        STAA    BREG+$9F        ; Set Port1 for Output
        STAA    BREG+$AF        ; Set Port2 for Output

        LDX     #BREG+$10       ; Reset all message control registers,
        LDAA    #$55            ; invalid all the Msg Objects
        LDAB    #$10
LOOP1   STAA    0,X
        STAA    1,X
        ABX
        CPX     #BREG+$F0+$10
        BNE     LOOP1

        LDAA    #$FF
        STAA    BREG+$06        ; Set Global Mask to Must Match
        STAA    BREG+$07        ; (Standard and Extended)
        STAA    BREG+$08
        STAA    BREG+$09
        STAA    BREG+$0A
        STAA    BREG+$0B

        LDAA    #$02            ; Reset CCE and Init bits,
        STAA    BREG+$00        ; Enable Global Interrupt

        LDAA    #$00            ; Set ID=0 to receive King Page
        STAA    BREG+$12        ; for Msg Obj 1
        STAA    BREG+$13
        STAA    BREG+$14
        STAA    BREG+$15

        LDAA    #$00            ; Set Msg Obj 1 to Receive, Standard
        STAA    BREG+$16

        LDAA    #%10111011      ; Valid the Msg Obj 1, Set RXIE
        STAA    BREG+$10

        RTS


        END
```

```
*******************************************************************************
*   This program is to get A/D values from PortE.4 and convert it to ASSII,*
* then send to the CAN bus.                                                   *
*   File_Name     : AD1_CITY.ASM                                              *
*   Author        : Long Giang Nguyen                                         *
*   ID            : 0959416                                                   *
*   Date          : 16-5-1998                                                 *
*   Last modified: 10-6-1998                                                  *
*******************************************************************************
*** EQUATE ***

BREG    EQU     $7000           ; CAN registers
RESET_C EQU     $B77D           ; Reset CAN chip
INIT_C  EQU     $B795           ; Initialise CAN chip
DELAY   EQU     $B770

I_SERV  EQU     $B730           ; ISR
MSG_OB1 EQU     $B705           ; Subroutine for Msg Obj 1
PG_0    EQU     $B6FE           ; Subroutine for King Page 0
PG_1    EQU     $B6C1           ; Subroutine for King Page 1
PG_2    EQU     $B6BA           ; Subroutine for King Page 2
PG_3    EQU     $B6B3           ; Subroutine for King Page 3
PG_4    EQU     $B696           ; Subroutine for King Page 4

ADDRESS EQU     $B704           ; EEPROM address
C_ADD   EQU     $01B1           ; RAM address
WORK    EQU     $01B0           ; Indicate whether the set up phase finished

PCTR    EQU     BREG+$EF        ; Use p2.0 p2.1 for indication (LED)

ADCTL   EQU     $1030           ; A/D Control
ADR1    EQU     $1031           ; A/D Result Register 1
EOT     EQU     $04             ; End of String
BASE2   EQU     BREG+$20        ; Base address for Msg Obj 2

ASCII   EQU     $B60B           ; Convert Hex to ASCII

*** Variable ***
        ORG     $00
BCDNO   RMB     6               ; BCD Number buffer
TEMP    RMB     1

        ORG     $00EE
        JMP     I_SERV          ; Assign address for ISR
```

```
*** Main program ***
        ORG     $100

MAIN    EQU     *

        JSR     RESET_C         ; Reset CAN controller
        JSR     INIT_C          ; Initialise CAN controller

        LDAA    #$01            ; Red LED
        STAA    PCTR            ; Indicate Set-up phase

        LDAA    ADDRESS
        STAA    C_ADD

        LDAA    #$00
        STAA    WORK

        LDD     #MSG_OB1        ; Assign values for the vector table
        STD     $01E4           ; Service for Msg Obj 1
        LDD     #PG_0
        STD     $01C0           ; Service for King Page 0
        LDD     #PG_1
        STD     $01C2           ; Service for King Page 1
        LDD     #PG_2
        STD     $01C4           ; Service for King Page 2
        LDD     #PG_3
        STD     $01C6           ; Service for King Page 3
        LDD     #PG_4
        STD     $01C8           ; Service for King Page 4

        CLI
WAIT    WAI
        LDAA    WORK
        CMPA    #$1
        BEQ     START
        BRA     WAIT
```

```
*** Get A/D value from PE4, convert it into ASSCII
*** then send it to CAN bus
START   EQU     *

        LDAA    #$02            ; Green LED
        STAA    PCTR            ; Indicate Run-phase

        LDAA    #$88            ; DLC=8, Dir = Transmit
        STAA    BASE2+$6

AD      LDAA    #$04            ; A/D on channel AN4
        STAA    ADCTL           ; Start conversion

CHECK   LDAA    ADCTL
        ANDA    #$80            ; The conversion finishes ?
        BEQ     CHECK

        LDAB    ADR1            ; Get the A/D value
        CMPB    TEMP            ; Compare with the previous value
        BEQ     DLY             ; if equal, then don't send
        STAB    TEMP            ; Store new value

        LDAA    #%11111010      ; CPUUpd:=1, NewDat:=1
        STAA    BASE2+$1

        LDY     #BASE2+$7       ; Update value
        LDAB    C_ADD           ; Get City's address
        JSR     ASCII

        LDAB    #':'
        STAB    3,Y

        LDY     #BASE2+$B
        LDAB    ADR1            ; Get the A/D value
        JSR     ASCII

        LDAB    #EOT
        STAB    $3,Y

        LDAA    #%11110111      ; CPUUpd:=0
        STAA    BASE2+$1

        LDAA    #$66            ; Transmit value containing
        STAA    BASE2+$1        ; in Msg Obj 2

DLY     LDY     #40             ; Delay 2 msecs
        JSR     DELAY           ; assume min bus rate = 1kbit/s
        BRA     AD


        END
```

```
*******************************************************************************
*   This program is to get A/D values from PortE.4 and convert it to ASSII *
* and send to the CAN bus if requested.                                       *
*   File_Name     : AD2_CITY.ASM                                              *
*   Author        : Long Giang Nguyen                                         *
*   ID            : 0959416                                                   *
*   Date          : 8-6-1998                                                  *
*   Last modified: 10-6-1998                                                  *
*******************************************************************************
*** EQUATE ***

BREG     EQU     $7000          ; CAN registers
RESET_C  EQU     $B77D          ; Reset CAN chip
INIT_C   EQU     $B795          ; Initialise CAN chip
DELAY    EQU     $B770

I_SERV   EQU     $B730          ; ISR
MSG_OB1  EQU     $B705          ; Subroutine for Msg Obj 1
PG_0     EQU     $B6FE          ; Subroutine for King Page 0
PG_1     EQU     $B6C1          ; Subroutine for King Page 1
PG_2     EQU     $B6BA          ; Subroutine for King Page 2
PG_3     EQU     $B6B3          ; Subroutine for King Page 3
PG_4     EQU     $B696          ; Subroutine for King Page 4

ADDRESS  EQU     $B704          ; EEPROM address
C_ADD    EQU     $01B1          ; RAM address
WORK     EQU     $01B0          ; Indicate whether the set up phase finished

PCTR     EQU     BREG+$EF       ; Use p2.0 p2.1 for indication (LED)

ADCTL    EQU     $1030          ; A/D Control
ADR1     EQU     $1031          ; A/D Result Register 1
EOT      EQU     $04            ; End of String
BASE2    EQU     BREG+$20       ; Base address for Msg Obj 2
ASCII    EQU     $B60B          ; Subroutine convert Hex to ASCII


*** Variable ***
         ORG     $00
BCDNO    RMB     6              ; BCD Number buffer

         ORG     $00EE
         JMP     I_SERV         ; Assign address for ISR
```

```
*** Main program ***
        ORG     $100

MAIN    EQU     *

        JSR     RESET_C         ; Reset CAN controller
        JSR     INIT_C          ; Initialise CAN controller

        LDAA    #$01            ; Red LED
        STAA    PCTR            ; Indicate Set-up phase

        LDAA    ADDRESS
        STAA    C_ADD

        LDAA    #$00
        STAA    WORK

        LDD     #MSG_OB1        ; Assign values for the vector table
        STD     $01E4           ; Service for Msg Obj 1
        LDD     #PG_0
        STD     $01C0           ; Service for King Page 0
        LDD     #PG_1
        STD     $01C2           ; Service for King Page 1
        LDD     #PG_2
        STD     $01C4           ; Service for King Page 2
        LDD     #PG_3
        STD     $01C6           ; Service for King Page 3
        LDD     #PG_4
        STD     $01C8           ; Service for King Page 4

        CLI
WAIT    WAI
        LDAA    WORK
        CMPA    #$1
        BEQ     START
        BRA     WAIT
```

```
*** Get A/D value from PE4, convert it into ASSCII
*** then send it to CAN bus if requested

START   EQU     *

        LDAA    #$02            ; Green LED
        STAA    PCTR            ; Indicate Run-phase

        LDAA    #$88            ; DLC=8, Dir = Transmit
        STAA    BASE2+$6

AD      LDAA    #$04            ; A/D on channel AN4
        STAA    ADCTL           ; Start conversion

CHECK   LDAA    ADCTL
        ANDA    #$80            ; The conversion finishes ?
        BEQ     CHECK

        LDAA    #%11111010      ; CPUUpd:=1, NewDat:=1
        STAA    BASE2+$1

        LDY     #BASE2+$7       ; Update value
        LDAB    C_ADD           ; Get City's address
        JSR     ASCII

        LDAB    #':'
        STAB    3,Y

        LDY     #BASE2+$B
        LDAB    ADR1            ; Get the A/D value
        JSR     ASCII

        LDAB    #EOT
        STAB    $3,Y

        LDAA    #%11110111      ; CPUUpd:=0
        STAA    BASE2+$1

        LDY     #2000           ; Delay 10 msecs
        JSR     DELAY

        BRA     AD

        END
```

```
*****************************************************************************
*    This program contain a subroutine that converts a Hexadecimal byte    *
* to ASCII code                                                             *
*    File_Name    : AD_ROU.ASM                                              *
*    Author       : Long Giang Nguyen                                       *
*    ID           : 0959416                                                 *
*    Date         : 8-6-1998                                                *
*    Last modified: 10-6-1998                                               *
*****************************************************************************

** Equate **

OPTION  EQU     $1039
BCDNO   EQU     $00
AD_PRO  EQU     $100

** Start program **
** This codes enable the program to start at EEPROM **
        ORG     $B600

        LDS     #$47
        LDAA    #$93            ; Enable A/D converter
        STAA    OPTION
        JMP     AD_PRO          ; Jump to main program

** ASCII: Convert 1 Hex data byte in ACC B to ASCII,
** store it into memory specified by IY
ASCII   EQU     *
        LDAA    #$00
        LDX     #100            ; Divide by 100
        IDIV
        STX     BCDNO           ; Store Hundreds
        LDX     #10             ; Divide by 10
        IDIV
        STX     BCDNO+2         ; Store Tens
        STD     BCDNO+4         ; Store Units

        LDAA    BCDNO+1         ; Hundreds
        ORAA    #$30
        STAA    0,Y

        LDAA    BCDNO+3         ; Tens
        ORAA    #$30
        STAA    1,Y

        LDAA    BCDNO+5         ; Units
        ORAA    #$30
        STAA    2,Y

        RTS

        END
```

```
*******************************************************************************
*   This program is to get a ASSCII values from Msg Obj 2 of the CAN chip    *
* and display on LCD.                                                         *
*   File_Name     : LCD_CITY.ASM                                              *
*   Author        : Long Giang Nguyen                                         *
*   ID            : 0959416                                                   *
*   Date          : 16-5-1998                                                 *
*   Last modified: 10-6-1998                                                  *
*******************************************************************************
*** EQUATE ***

BREG     EQU      $7000           ; CAN registers
RESET_C  EQU      $B77D           ; Reset CAN chip
INIT_C   EQU      $B795           ; Initialise CAN chip
DELAY    EQU      $B770           ; Delay subroutine

I_SERV   EQU      $B730           ; ISR
MSG_OB1  EQU      $B705           ; Subroutine for Msg Obj 1
PG_0     EQU      $B6FE           ; Subroutine for King Page 0
PG_1     EQU      $B6C1           ; Subroutine for King Page 1
PG_2     EQU      $B6BA           ; Subroutine for King Page 2
PG_3     EQU      $B6B3           ; Subroutine for King Page 3
PG_4     EQU      $B696           ; Subroutine for King Page 4

ADDRESS  EQU      $B704           ; EEPROM address
C_ADD    EQU      $01B1           ; RAM address
WORK     EQU      $01B0           ; Indicate whether the set up phase finished
BASE2    EQU      BREG+$20        ; Base address for Msg Obj 2
BASE3    EQU      BREG+$30        ; Base address for Msg Obj 3

PCTR     EQU      BREG+$EF        ; Use p2.0 and p2.1 for indicator (LED)
PORTA    EQU      $1000           ; Use port A.0 to send remote frame

INIT_L   EQU      $B616           ; Initialise the LCD
DISP     EQU      $B634           ; Display a string on LCD
OUTPUT   EQU      $B64A           ; Output an instruction to LCD
SCREEN   EQU      $B65D           ; Display 1 character in Acc A on LCD
LINE2    EQU      $A8             ; Configure to display on line 2 of LCD
EOT      EQU      $04             ; End of String

*** Assign ISR ***
         ORG      $00EE
         JMP      I_SERV          ; Assign address for ISR

*** Messages ***
         ORG      $00

BASE     RMB      2               ; Base address of the Msg Obj
MSG1     FCC      '* CAN SYSTEM DEMO *'
         FCB      EOT
MSG2     FCC      'CITY_'
         FCB      EOT
```

```
*** Main program ***

        ORG     $100

MAIN    JSR     RESET_C         ; Reset CAN controller
        JSR     INIT_C          ; Initialise CAN controller

        JSR     INIT_L          ; Initialise LCD
        LDX     #MSG1
        JSR     DISP

        LDAA    #$01            ; Red LED
        STAA    PCTR            ; Indicate Set-up phase

        LDAA    ADDRESS
        STAA    C_ADD

        LDAA    #$00
        STAA    WORK

        LDD     #MSG_OB1        ; Assign values for the vector table
        STD     $01E4           ; Service for Msg Obj 1
        LDD     #MSG_OB2
        STD     $01E6           ; Service for Msg Obj 2
        LDD     #MSG_OB3
        STD     $01E8           ; Service for Msg Obj 3

        LDD     #PG_0
        STD     $01C0           ; Service for King Page 0
        LDD     #PG_1
        STD     $01C2           ; Service for King Page 1
        LDD     #PG_2
        STD     $01C4           ; Service for King Page 2
        LDD     #PG_3
        STD     $01C6           ; Service for King Page 3
        LDD     #PG_4
        STD     $01C8           ; Service for King Page 4

        CLI
WAIT    WAI
        LDAA    WORK
        CMPA    #$1
        BEQ     START
        BRA     WAIT
```

```
START   EQU     *

        LDAA    #$02            ; Green LED
        STAA    PCTR            ; Indicate Run phase

        LDAA    #%11111011      ; Set RXIE for
        STAA    BASE2+$0        ; Msg Obj 2
        STAA    BASE3+$0        ; Msg Obj 3

CHECK   LDAA    PORTA           ; Check if remote request (button pressed)
        ANDA    #$01            ; PA.0=0 ?
        BNE     CHECK
        LDAA    #%11101111      ; Send remote frame
        STAA    BASE3+$01
        LDY     #20000          ; Delay 1 sec
        JSR     DELAY
        BRA     CHECK

*** Subroutine to decode Msg Obj 2
MSG_OB2 EQU     *
        LDD     #BASE2
        STD     BASE
        JSR     LCD
        RTS

*** Subroutine to decode Msg Obj 3
MSG_OB3 EQU     *
        LDD     #BASE3
        STD     BASE
        JSR     LCD
        RTS


*** Subroutine to display data to LCD
LCD     EQU     *
        LDAA    #LINE2          ; Configure to display on line 2
        JSR     OUTPUT
        LDY     #$0001
        JSR     DELAY

        LDX     #MSG2
        JSR     DISP

        LDX     BASE
        LDAA    #%11111101      ; Reset NewDat
        STAA    1,X
        LDX     BASE
        LDAB    #$7             ; First data byte of the Msg Obj
        ABX
        JSR     DISP            ; Display received msg
        RTS

        END
```

```
*****************************************************************************
*   This program is a set of subroutine to control the LCD                  *
*   File name     : LCD_ROU                                                 *
*   Author        : Long Giang Nguyen                                       *
*   ID            : 0959416                                                 *
*   Date          : 16-5-1998                                               *
*   Last modified: 10-6-1998                                                *
*****************************************************************************
*** EQUATE ***

BREG    EQU     $7000           ; CAN chip's memory location
DELAY   EQU     $B770           ; Delay subroutine
LCD_PRO EQU     $0100           ; Main program

PDATA   EQU     BREG+$DF        ; Use port1 for output data to LCD
PCTR    EQU     BREG+$EF        ; Use port2  bit 4-6 for control
FUNC    EQU     $38             ; Function set to 8 bits, 2 lines
EOT     EQU     $04             ; End Of String Characters

********************
        ORG     $B600

        LDS     #$47
        JMP     LCD_PRO

TIME    FCB     $60             ; Delay time used during initialisation
        FCB     $02
        FCB     $02
        FCB     $02
        FCB     $02
        FCB     $40
        FCB     $02
        FCB     $02

INSTR   FCB     FUNC            ; Instruction used to initialise LCD
        FCB     FUNC
        FCB     FUNC
        FCB     FUNC
        FCB     $08             ; Display OFF
        FCB     $01             ; Display CLEAR
        FCB     $0C             ; Display ON
        FCB     $06             ; Entry Mode: inc display, no shift
```

```
*** Subroutines ***

** INIT: Initialise LCD

INIT    LDY     #$140           ; Delay 16 msec
        JSR     DELAY

        LDX     #TIME

LOOP    CPX     #TIME+8         ; Last Instr has been output
        BEQ     BACK

        LDAA    08,X            ; Fetch next Instr
        JSR     OUTPUT

        LDY     0,X             ; Fetch next time value
        JSR     DELAY

        INX
        BRA     LOOP

BACK    RTS

** DISP: Display a string at the address specified by Reg IX

DISP    PSHA

DD      LDAA    0,X             ; Load Char
        CMPA    #EOT            ; End of String ?
        BEQ     EXIT
        JSR     SCREEN

        LDY     #$01            ; Delay 50 micro sec
        JSR     DELAY

        INX
        BRA     DD

EXIT    PULA
        RTS

** OUTPUT: Output an instruction an ACC A to LCD

OUTPUT  STAA    PDATA           ; Instruction for LCD

        LDAA    #$02            ; RS=0, R/W#=0, E=0
        STAA    PCTR

        LDAA    #$42            ; RS=0, R/W#=0, E=1
        STAA    PCTR

        LDAA    #$02            ; RS=0, R/W#=0, E=0
        STAA    PCTR

        RTS
```

```
** SCREEN: Output data on ACC A to LCD

SCREEN  STAA    PDATA           ; Instruction for LCD

        LDAA    #$22            ; RS=1, R/W#=0, E=0
        STAA    PCTR

        LDAA    #$62            ; RS=1, R/W#=0, E=1
        STAA    PCTR

        LDAA    #$22            ; RS=1, R/W#=0, E=0
        STAA    PCTR

        RTS

        END
```