A Conformance Test Framework for the DeviceNet Fieldbus

by
Soo Beng KHOH

A thesis submitted for the award of degree of

Doctor of Philosophy

International Manufacturing Centre
Department of Engineering
University of Warwick

October 1996

Abstract

The DeviceNet fieldbus technology is introduced and discussed. DeviceNet is an open standard fieldbus which uses the proven Controller Area Network technology. As an open standard fieldbus, the device conformance is extremely important to ensure smooth operation. The error management in DeviceNet protocol is highlighted and an error injection technique is devised to test the implementation under test for the correct error-recovery conformance. The designed Error Frame Generator prototype allows the error management and recovery of DeviceNet implementations to be conformance tested. The Error Frame Generator can also be used in other Controller Area Network based protocols.

In addition, an automated Conformance Test Engine framework has been defined for realising the conformance testing of DeviceNet implementations. Automated conformance test is used to achieve consistent and reliable test results, apart from the benefits in time and personnel savings. This involves the investigations and feasibility studies in adapting the ISO 9646 conformance test standards for use in DeviceNet fieldbus.

The Unique Input/Output sequences method is used for the generation of DeviceNet conformance tests. The Unique Input/Output method does not require a fully specified protocol specification and gives shorter test sequences, since only specific state information is needed. As conformance testing addresses only the protocol verification, it is foreseen that formal method validation of the DeviceNet protocol must be performed at some stage to validate the DeviceNet specification.

Acknowledgement

The author would like to thank his academic supervisors Dr. Ken Young and Mr. Richard McLaughlin of Warwick Manufacturing Group for their assistance, guidance, and the informative and stimulating discussions during the course of research. The author would also like to acknowledge the University of Warwick with whose grant this PhD research was carried out.

Secondly, Mr. Shawn Adams of Allen Bradley must be thanked for allowing me the opportunity to work at Allen Bradley, USA as a visiting researcher. Appreciation is also due to Mr. Ed Polce and Mr. Doug Stanton for their help on DeviceNet Conformance; Mr. Pat Maloney, Mr. Ron Holl, Mr. Dave Callaghan, Mr. Tom Kocjan and Mr. Dan Noonen on DeviceNet protocol comprehension; Mr. Dave Comeau on Visual C++ Software Design; Mr. James Gross on Systems Tests and many more who work at Allen-Bradley, Landerhaven site, USA. Thanks are due to Mr. Les Warrington and Mr. Adrian Vinsome for their help and guidance on Object- Oriented methodology and Visual C++/C++ development, Mr. Trevor Martin of Hitex UK who supplied the Hitop embedded software development environment and Phytec embedded controllers. Special thanks are also due to all the research fellows and technicians at International Manufacturing Centre and Advanced Technology Centre for their assistance and friendships.

Lastly, my deepest heartfelt thanks to many friends, who are too numerous to be individually named, and my family for their patience and support during the production of this thesis.

Soo Beng KHOH

October, 1996

Declaration

This thesis forms the original work by the author and has not been published or used in any other award submission. All published works have been acknowledged with references.

Disclaimer

The trademarks and companies mentioned in this thesis do not imply recommendation or endorsement by the author, nor that the products are necessarily the best available for the purpose. All trademarks remain the properties of the corresponding companies.

In loving memory of my beloved mother...

Contents

List of Figures

List of Tables

Glossary

CAL	CAN Application Layer
CAN	Controller Area Network
CID	Connection Identifier
CIM	Computer Integrated Manufacturing
CSMA/CD+NDBA	Carrier-Sense Multiple Access/Collision Detection + Non-Destructive Bitwise Arbitration
CTE	Conformance Test Engine
DDE	Dynamic Data Exchange
DLL	Dynamic Link Library
DUT	Device Under Test
ECU	Electronic Control Unit
EFG	Error Frame Generator
EMC	Electromagnetic Compatibility
EPR	Expected Packet Rate
FDT	Formal Description Technique
FSM	Finite State Machine
IEC	International Electrotechnical Committee
ISA	Instrument Society of America
ISO	International Organisation for Standardisation
IUT	Implementation Under Test
LLC	Logical Link Control
MAC	Medium Access Control
MACID	Media Access Control Identifier
MAU	Media Access Unit
MDI	Medium Dependent Interface
ODVA	Open DeviceNet Vendors Association
OLE	Object Linking and Embedding
OOA	Object Oriented Analysis
OOD	Object Oriented Design
OOP	Object Oriented Programming

PDU	Protocol Data Unit
PICS	Protocol Implementation Conformance Statement
PIXIT	Protocol Implementation eXtra Information for Testing
PLS	Physical Layer Signalling
PMA	Physical Medium Attachment
SDS	Smart Distributed System
SLIO	Serial Linked Input/Output
SOC	Statement of Compliance
SOF	Start-of-Frame
TTCN	The Tree and Tabular Combined Notation
UCMM	Unconnected Message Manager
UIO	Unique Input/Output Sequence Method

Chapter 1 - Introduction

With the reduction of product life-cycle from years to months, today's manufacturing systems must not only be efficient enough to produce high volume and quality products at minimal possible costs, they must also have sufficient flexibility for fast product change. The advent of higher power-to-cost ratio silicon has enabled microprocessors to be used extensively in the control and instrumentation of the manufacturing world. As such, computer-controlled systems are ideal for producing quality products competitively.

1.1 Evolution of Digital Control Systems

Digital computers were first introduced to improve the efficiency of measurement and control systems two decades ago. Initially these were stand-alone custom-designed control systems which were very bulky, expensive and unreliable. In 1968, a group of engineers from General Motors, USA conceived an idea to provide a flexible and cost effective solution to replace the hard-wired relay-logic in industrial control [1, 2]. They called their invention the Programmable Logic Controller (PLC). Since then the PLC has become the dominant control technology in the manufacturing and process industry.

The advancement in the semiconductor industry has had a dramatic impact on PLC technology. Intel introduced the first 8-bit 8008 microprocessor in 1972, development followed by the more powerful 8080 two years later [3]. To date, the growth of the semiconductor industry has been correctly predicted by Moore's Law[1]. In 1977, Allen-Bradley of the USA introduced the first industry standard PLC based on the newly introduced Intel 8080 microprocessor [2]. It was not until the 1980s that PLCs were used extensively in control and instrumentation in the manufacturing industry.

The advantage of using the PLC is that the sequence of operations or controls can be altered easily by programming the PLC using customised software called ladder-logic. The PLC (which consists of microprocessors) reads values from the sensors, evaluates and executes commands by sending new states to the actuators. Its modular design allows easy upgrade and expansion of the control system when the need arises. When there is a fault in a module, it can be easily replaced by another plug-in module, thus increasing the maintainability of the system, as well as production up-time.

This flexibility has enabled control system vendors to mass produce the PLC to be used in virtually all control applications. End-users can then alter the software to cater for their specific requirements. Control system vendors like Allen-Bradley, Festo, GEC, Hitachi, Omron, Mitsubishi, Selectron to name a few, have since shared the market for PLCs. This has brought PLCs within the budget of many potential users in the manufacturing industries, with higher performance systems introduced every year. For example, the PLC3 controller introduced by Allen Bradley in 1980 was 1,000 times more powerful than the company's first PLC [1].

[1] Gordon Moore is the co-founder of Intel and a chip designer. He observed that a new chip will be developed every 18 months and contain twice as much processing power as its predecessor. Combining the two observations, he predicted that chip processing power will increase exponentially and this became known as Moore's Law.

1.2 Complexity of Control Systems

As the activity of process control becomes more complex, more sensors and actuators are needed to perform the control sequences. More devices mean more wiring. In addition, the strive towards higher product quality in the competitive global market of the 90s has imposed new requirements on the manufacturing systems. The use of quality assurance technique such as the statistical process control (SPC) [4] requires manufacturing processes to be monitored closely. Furthermore, all instruments involved in the manufacturing processes must be in tip-top working condition to maximise production up-time. To achieve this, extensive preventive failure analysis and maintenance program is exercised to monitor and maintain the instrument's health. This resulted in the influx of the wiring harness as monitor lines are needed to obtain the health and status feedback from the devices. To meet the ever increasing demand of complex process control, the following observations are made.

- Flexibility

  The conventional point-to-point system was designed for transfer line facility in mass production environment, i.e. large batch size manufacturing. Most of the process/assembly stations were not expected to be reconfigured frequently as it is a very tedious and time-consuming job to set up and manage the wire spaghetti. Any addition of new device will involve the routing of new wires through the already complex cable conduit that is difficult to access. In other words, the time penalty for set-up and commissioning of these control system has become noticeable.

- Reliability

The reliability of the conventional point-to-point wiring is questionable. An average size PLC may consist of 8 I/O racks with 32 I/Os per rack. This means that there are about 200 wires that need to be labelled and managed properly. To get the wiring right first time is a challenging experience. The large number of wires may also lead to a decreased in system reliability, i.e. more wires, more chances of things going wrong. In addition, the use of junction boxes and connectors for wire management will contribute to the poor reliability of the system.

- Noise and Cross-talk interference

Cross-talk interference is another problem in wire-packed control systems. It is a situation where neighbouring wires induce electromagnetic noise to each other and caused stray signals and distortions. Therefore, additional care must be taken when routing the analogue-to-digital converter's signal wire to ensure that the accuracy of the analogue value is not affected. The use of shielded cable may be required in extremely noisy environments.

- Maintainability

The objective of a production system is to convert the raw materials into finished products. Machine breakdown or production downtime is a costly affair. Hence, the reliability and maintainability of the production system is of utmost importance. The speed with which a fault can be located and rectified, or prevented, plays a vital role. However, if a wire were to have bad electrical contact in the 200 wires conduit, identifying the fault may be a painstaking and time consuming job.

Ideally, the new generation of manufacturing control systems must be modular, easily reconfigurable, highly reliable and maintainable, fault tolerant and, most importantly, cost effective. They must be flexible in adapting themselves to fast product change. A solution is to inter-connect all the sensors and actuators with a single digital bus for realising the automation and controls. The multiplexing of these sensors and actuators signals via a single bus is achieved using the emerging fieldbus technology, i.e. a technology derived from the computer local area networks (LANs).

1.3 Fieldbus Technology

Fieldbus is defined as an all digital, high performance, multi-drop communication network for connecting process instrumentation to controllers [5]. In its simplest form, fieldbus is just a communication media which benefits the users with simple 'plug-and-play' installation, reduced wiring, improved diagnostic-ability, reconfiguration flexibility and ease of maintenance. Fieldbus is a computer network optimised for controls and instrumentation. Therefore, it features small data packets with deterministic bus arbitration scheme for guaranteed network response time.

Fieldbus system brings substantial costs savings. An Italian power utility predicts a 4% reduction in overall investment costs after allowing for 10-20% increase in new device costs [6]. In Du Pont's fieldbus installation, wiring cost savings of up to 20% were reported for its Brevard site facility in Asheville, North Carolina, USA [7]. The installation at BP Research and Engineering facility, Sunbury-on-Thames, UK, has also demonstrated savings in cost and installed cables, as well as enhanced process control system performance in an industrial situation [8].

The application of fieldbus is not restricted to process control and manufacturing industry. For example, Profibus (i.e. another instance of fieldbus) is used in the Warsaw underground, Poland for reporting the status of the subway equipment and data exchange between computers at the dispatch and control centre, and the Kabaty end station [9]. This is believed to be the first fieldbus installation in safety-critical railway signalling application [10]. Other areas of fieldbus application include building automation and marine application.

1.4 Fieldbus Standards

Fieldbus can be divided into open standard and proprietary. Proprietary fieldbus standard is developed and marketed by a single vendor with limited product range. Market trend has indicated that the proprietary fieldbusses are no longer welcome by the industry. The flexibility of sourcing devices from different vendors and plugging together to form a single field control and instrument system (i.e. an open fieldbus standard) has far greater benefits than relying on a single source. Many proprietary fieldbusses have since joined forces to reduce the many proprietary standards by forming the consortia for an open standard.

The Instrument Society of America (ISA) has collaborated with the International Electrotechnical Committee (IEC) to form the ISA/IEC SP-50 working committee in an attempt to define an international standard for the fieldbus. One of the objectives of the SP-50 working committee was to define an international fieldbus standard so as to eliminate the many *standards* available today.

In the meantime, two vendor-driven consortia fieldbusses, i.e. WorldFip[2] and Interoperable Systems Project[3] (ISP) had collaborated in October 1994 to produce the Fieldbus Foundation (FF) fieldbus standard. The Fieldbus Foundation has inherited many of the SP-50 specification which include the SP-50 physical layer (IEC 1158-2 standard) for intrinsically safe operation. Fieldbus Foundation (FF) hopes that when an international fieldbus standard is created, FF can be fully compatible with the SP-50 fieldbus standard and become the de-facto industry standard.

Within Europe, CENELEC[4] has approved WorldFip [11], Profibus [12] and P-Net [13] as the European fieldbus norm, EN 50170 [14, 15]. This may mean that Europe and America may well go in different directions, leaving us with no international fieldbus standard.

While the ISA/IEC SP-50 has neared its end of the international fieldbus standard specification, there have been another development of a much simpler, lower level fieldbus subset. This simple I/O function fieldbus is sometimes referred to as the **sensor bus**. Among them are Actuator Sensor Interface (ASI), Controller Area Network (CAN) [16] based networks, e.g. CAL [17], CANOpen [18], CAN Kingdom [19], DeviceNet [20] and Smart Distributed Systems (SDS) [21, 22, 23], Lonworks [24, 25] and Profibus-DP [26, 27]. The successful de-facto standard in this sector will depend on the cost-effectiveness, product availability, consortia of vendors, as well as the technical advantages.

[2] WorldFip is backed by Allen-Bradley, Honeywell and others.
[3] Interoperable Systems Project (ISP) is supported by Fisher-Rosemount, Siemens, Jonson Yokogawa etc.
[4] CENELEC - The European Committee for Electrotechnical Standardisation

1.5 Conformance to Fieldbus Standard

In an open fieldbus scenario, all the field instruments must conform to the rules laid out by the standard specification. Failure to adhere to the fieldbus standard protocol will bring catastrophic failure not only to the responsible culprit, but the entire fieldbus network or system. This will affect the end-users' confidence on the particular fieldbus standard. Conformance testing an open fieldbus implementation is the vital step to ensure that all the field device developers have uniform interpretation of the specification, and that their products conform to the protocol standard.

"A conformance testing is used to verify that the external behaviour of a given implementation of a protocol is equivalent to its formal specification" [28].

DeviceNet is an open standard fieldbus which uses the proven CAN technology. Since it is an open standard, the conformance testing is a must to ensure the smooth operation and success of this fieldbus standard. DeviceNet is owned by the Open DeviceNet Vendors' Association (ODVA), and consists of a consortium of more than 130 companies which include Allen-Bradley, Banner, Omron, Peperl & Fuch, Schrader-Bellows and SMC. ODVA has formed many Special Interest Group (SIG) to manage and govern the technical related issues of DeviceNet. Among the many ODVA SIGs, the Conformance SIG is of particular interest due to its responsibility on the conformance testing of DeviceNet implementation.

A significant number of conformance test suites have been developed by the Conformance SIG. Until now, the conformance test suites of DeviceNet still do not include the tests for error recovery and management. The error recovery sequences and state machines are defined in the protocol specification for product developers to follow. For example, when a fault is detected the device must indicate the detected fault by flashing its LEDs in a predefined fashion and move into a predefined "safe state". In addition, the device's internal object's attribute must register the detected fault. So far, the test for correct error recovery and management has been hampered by the lack of a suitable method of injecting errors.

Given the importance in conformance testing of the implementation of correct error recovery and handling, perhaps a special hardware can be researched and designed to carry out the job. This protocol emulator must be able to inject errors in a controlled manner. In other words, it must only induce error on the implementation under test so that the effect can be observed by another node, i.e. the test system.

1.6 Research Objectives

The primary objective of this thesis is to study, investigate and devise an appropriate compliance test tool for injecting error onto the bus. This allows the testing of DeviceNet open standard fieldbus implementations for correct error recovery.

The research started with the investigation of alternative uses of the Controller Area Network (CAN) technology in non-automotive sectors. It was envisaged that the CAN technology can be feasibly applied in the controls and instrumentation industry. The background study and investigation on the current controls and instrumentation trends were carried out. This led to the development of DeviceNet, i.e. a CAN derivative fieldbus. Progressively a knowledge gap had been identified in the area of conformance testing, in particular on the error injection for implementation fault testing.

The possibility of realising an automated approach for conformance testing had to be explored to achieve consistent and high-quality test results. Thus an error injection mechanism has been designed and included in the automated compliance test framework for realising the error testing.

The secondary objective includes the study and recommendation of the suitable object-oriented concept and design methodology for use in the prototype development. An interoperability test system for DeviceNet must also be investigated to test DeviceNet as a complete controls and instrumentation system, rather than as individually complied nodes.

CHAPTER 2

FIELDBUS CONTROL SYSTEMS

Fieldbus is a new technology of controls and instrumentation which uses the computer networks technology. These fieldbusses must be able to interface with the existing computer networks within a manufacturing organisation. A hierarchy of computer networks has been established, each responsible for the corresponding level needs. For instance, the computer networks at the lowest hierarchy normally consist of hard real-time controls, with short and bursty data packets for deterministic network response. On the other hand, higher level networks such as those used in the Management Information System (MIS) will place great emphasis on the data carrying capacity rather than deterministic network response time. Combining these various levels of computer networks, data from the shop floor can be fed directly into higher level systems to achieve the computer integrated manufacturing (CIM) systems within the organisation.

2.1 The Automated Manufacturing Hierarchy

In the paper published by C. McLean et al [29], the authors have proposed the five levels of control hierarchy based on analysis of a non-automated batch manufacturing system. The five levels were facility, shop, cell, work station and equipment. Computer Aided Manufacturing (CAM) was believed to be introduced sometime later as there is no evidence of it when McLean et al. published their findings in 1983. The automated manufacturing control hierarchy realised in the National Bureau of Standards' Automated Manufacturing Research Facility (currently known as the National Institute of Standards and Technology, NIST), which was planned to be operational in 1986 [29] is shown in Figure 2-1.

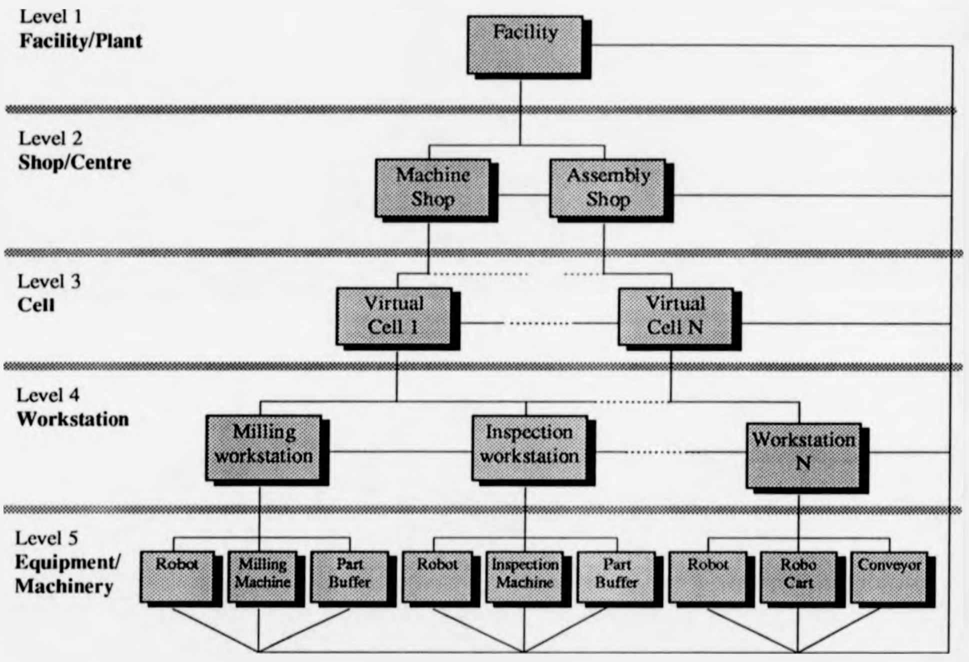


Figure 2-1 The automated manufacturing hierarchy of McLean et al.

Also known as the control pyramid [30], the hierarchy has to act as the foundation whenever automated manufacturing issues are discussed. So, the introduction of CIM has further caused the need to interconnect every machines or islands of machines in the manufacturing environment in order to facilitate better information flows within the CIM envelope. This has encouraged the development and utilisation of various communication networks between different hierarchies and within the same hierarchy.

According to the work done by Valenzano et al [31], the communication infrastructure to support CIM is composed of 3 types of communication networks as shown in Figure 2.2. The communication network within this context is referring to computer networks such as Local Area Network (LAN).

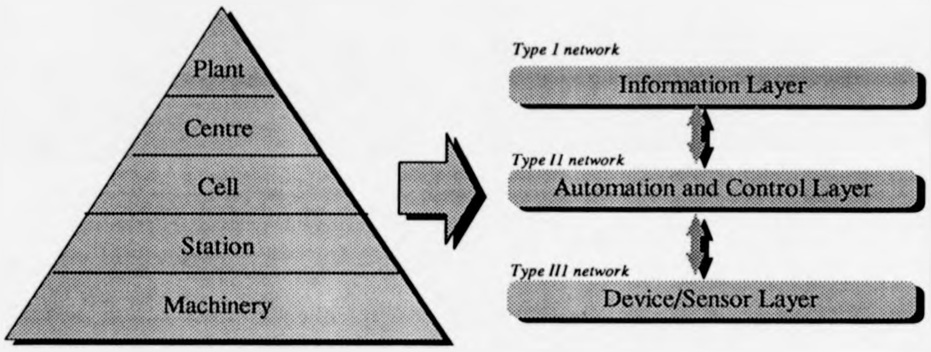


Figure 2-2 The automated manufacturing pyramid and its communication network hierarchy

At the higher level of the hierarchy, large data flow is required, with less emphasis on the real-time response. However, the situation is totally reversed at the bottom of the hierarchy where fewer data are exchanged rapidly with strict real-time constraints. These two extreme applications on both ends of the pyramid place greatly differing demands on networks requirements, and no single network technology today can be optimised for the full range of operations.

The currently recognised solution to the differing requirements is to use different levels of communication networks to cover these needs. This involves the use of many communication gateways to convert the different data formats from one level to another so that the data can be consumed by other communication networks.

However, there is a drastic change in the type-II and type-III networks. Years ago, these networks mainly consisted of vendor specific or proprietary solution. The majority of the communication networks were not part of the control system design, nor use LAN type communication. They were simply some extensions to be bolted on the controllers for communication to take place, e.g. RS-232, IEEE-488 communication standards. Emphasis was placed on the better design of the control system rather than integrating the communication protocol into the control systems. Because they were not computer networks, the OSI 7 layer model cannot be used to model the communication interface. Hence, communication between these devices were not elegantly carried out.

Today, many of these communication media such as RS-485, have been replaced with the open standards computer networks, most of which are based their design on the ISO/OSI 7 layer model for openness. The adoption of computer network allows the communication interface to be designed into the control and instrumentation systems, rather than the conventional 'bolted-on' approach. The availability of more powerful and advanced semiconductor technology, i.e. more powerful and cost effective processors, has enabled more complex and sophisticated communication to be implemented.

Suddenly, the communication requirements in the factory shop floor have opened another new market niche, i.e. communication networks. Digital busses were used to interconnect the machines and islands of automation together, These industrial communication networks are known as fieldbus. Elegantly designed gateways were introduced by the fieldbus vendors to transfer the data from one layer to another.

2.1.1 Information Layer

At the top of the hierarchy, information exchange through the network is in large blocks (in the region of several kilo bytes to megabytes). This level is incompatible with real-time critical applications due to the inverse relationship between real-time control and data packet size. Table 2.1 shows a comparison between various level of industrial computer networks in-terms of the network size, data volume and response time. Networks at the information level will therefore tend to have more data carrying capability but less real-time performance.

At this level, networks will be used for transferring data between departments in the CIM environment (Figure 2.2) within a company (Intranet), or even globally (Internet) using Ethernet with TCP/IP protocol. The packet size for Ethernet based networks varies between 64 bytes to 1,518 bytes at 10 to 50 Mbits/s typical. The applications that run at this level do not have any strict real-time constraints and any additional delays can be tolerated by the system. A typical application may be transferring Statistical Process Control (SPC) data from automation cells to the Management Information System

(MIS) database of the factory for quality assurance purposes. Other applications include email, file transfer, database query and update, etc.

2.1.2 Control Layer

One level down from the information layer is the control layer. At the control level, smaller quantities of data (as compared to information level) are being transferred between different cell controllers at high speed for exercising real-time control. Each data packet may be in the region of a hundred bytes. Most of the computer networks used in this layer are found on the shop floor to connect field instruments together. Hence, the term fieldbus is coined to refer to any computer network which is used for real-time control. For example, Profibus and Fip are currently the two top contender at this level.

2.1.3 Device Layer

Communication networks in the Device Layer are very similar to the control layer. Here, the communication network needs to satisfy hard real-time constraints as compared to the control layer. Since the communications are among devices such as sensors, actuators and the PLC, data flows are usually in terms of bits or in a few bytes. The network traffic mainly involves the cyclic exchanges of sensor values and actuator commands in a short and bursty fashion. The Protocol Data Unit (PDU) must be short in order to minmise the network latency time. In addition, the network must be capable of delivering high network efficiency in this frequent, periodic, small sized data application.

Network efficiency refers to the ratio between the amount of useful user data ferried across the network and the number of bits needed to perform the ferrying operation [32].

Since the network links mainly the sensors and actuators to the PLC, it is also known as Sensor Bus. Example of sensor bus include Actuator Sensor Interface (ASI), CANopen, DeviceNet and Smart Distributed System (SDS). Another reason for having a Sensor Bus is the economic factor, i.e. the cost of putting the network interface on a low-cost device must be minimal.

Table 2-1 The summary characteristics of different level computer networks in a manufacturing organisation

	Network size	Data volume	Response time
Information Layer	Large	Large	Slow
Control Layer	Moderate	Moderate	Moderate
Device Layer	Small	Small	Fast

2.1.4 Planning Horizon

With the use of more powerful hardware solely for communication, the planning horizon on each level will undoubtedly increase. Planning horizon is the amount of time any control system is allocated to handle its tasks [29]. The system does not need to know about any activities outside its planning horizons. Hence, the planning horizon acts as an information hiding mechanism so that the complex system can be decomposed and executed separately by less powerful sub-systems. The boundary of control in each sub-system is defined by its immediate higher level in the hierarchy.

The planning horizon of the 3 layer model may still remain the same, but be implemented differently. Instead of using the physical hardware to decompose the planning horizon, the planning horizon can be done in some clever software routine. In other words, instead of using 3 separate sets of PLCs to form a control hierarchy, the system can be implemented in a multi-processors PLC with 3 different independent sets of software routine (3 channels of software), each invisible to the other. The object-oriented technology in software design, which features the data abstraction and encapsulation, can be used in this instance to hide away unnecessary information, thus simplifying the control system design.

2.2 Overview of Fieldbus

The process control industry has been waiting for a technology which is capable of reducing the wiring harness, providing reconfiguration flexibility, reliability and, most importantly, cost effectiveness. These requirements have made fieldbus systems the key focal point for the development of better and more efficient digital control systems for the 21st. century.

With the application of fieldbus, the control systems in Figure 2-3 can now be replaced with a single wire serial data bus that interconnects all the sensors and actuators before linking back to the central controller. Figure 2-4 depicts a typical fieldbus control system. The availability of an on-board microcontroller allows the simple switch to have some 'intelligence'. This allows it to perform other functions such as self-diagnosis, self-calibration, error warning etc. In addition, extra parameters can be realised without the use of additional wire. The appearance of intelligent/smart sensors and actuators and control networks has marked a new era of control technology in factory automation. Figure 2-3 and Figure 2-4 show the comparison between the conventional point-to-point control system and the more simplified fieldbus control.

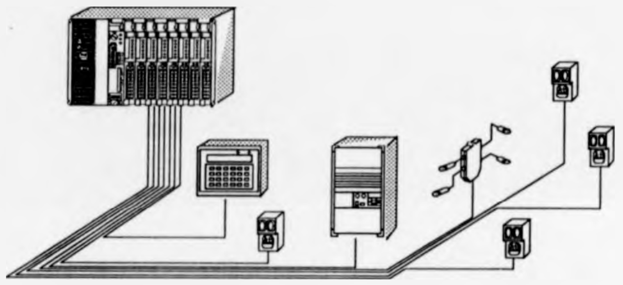


Figure 2-3 Conventional point-to-point PLC based control system

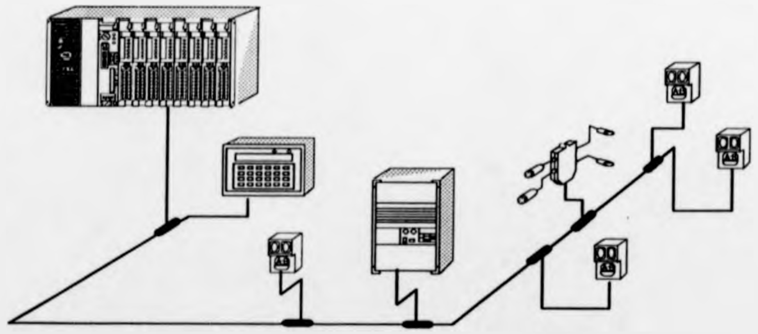


Figure 2-4 Distributed control using fieldbus network

2.2.1 Historical Background

The original idea of having a Fieldbus was simply to replace the ageing 4-20mA transmission signal designed some 25 years ago with a digital communication media. In the past, communication between smart field instruments was done mainly by proprietary digital busses and custom designed gateways. Most vendors developed proprietary digital busses for their own markets. This does not create a problem as the user company has large engineering resources to support the many gateways and signal converters. General Motors for instance, spend more than half of their automation budget on the implementation of custom interfaces between intelligent instruments. At the end of the 1980s, there were so many proprietary digital busses around that engineering staff of the user company could no longer cope with the specific proprietary solutions. In addition, many companies were involved in “rightsizing” exercises and laid-off many “redundant” employees. The industry has since realised that in order to utilise these smart instruments successfully, a new generation of communication standard is required. This is the same train of thought that GM had when it embarked on the Manufacturing Automation Protocol (MAP) and Technical and Office Protocol (TOP) project to minimise the interfacing problems.

In the simplest form, fieldbus networks replace complex wiring which benefits the users in terms of system installation cost, shorter commissioning time and improved diagnostics-ability. Fieldbusses interface with higher level computer networks through gateways to allow a continuous flow of data from the shop floor to higher hierarchy. This allows higher level systems access to data on lower levels, thus providing better integration within the organisation, i.e. a step closer to computer integrated manufacturing (CIM).

However, fieldbus is more than a digital communication network [33]. It is an open Field Control System (FCS) of the 21st. century which will change the scene in distributed control and field instrumentation. Although it took the industry almost a decade to realise the full potential and complexity of Fieldbus, this highly distributed process system will be an important issue in Automation Industry.

The availability of new generation high performance single-chip microcomputers or microcontrollers has started to change the scene of process control. For example, microcontrollers now have the capability and power to process complex PID control algorithms locally, instead of routing the raw data back to the central processors for processing. Fuelled by the smaller footprint of surface mounted technology and higher performance to cost ratio silicon, more advanced control routines can be feasibly and economically implemented on the field device itself. This has indirectly moved the control tasks from the central control unit to the field resident microcontrollers, hence the birth of distributed control systems. These microprocessor based instrumentation systems are collectively referred to as smart instruments. If implemented properly, these smart instruments will execute the control sequence in parallel. In 1989, Oxford University, Foxboro GB Ltd and ICI launched the Sensor Validation (SEVAC) project to develop smart sensors for fieldbus application [34].

2.2.2 Open Standard Fieldbus

In general Fieldbus networks can be divided into two categories, i.e. proprietary and open standard architecture. Proprietary fieldbus is developed and marketed by a single vendor with a range of products to offer. Since it's proprietary, the vendor has full authority and responsibility to optimise the performance and rectify the fault of its fieldbus system. This will either give an outstanding performance or otherwise. Being controlled by a single source means easier technical support access when things go wrong. For instance, one call to the vendor may solve the problem. However, when the system consists of several vendor's products, it may take a while for the vendors to work out whose device is at fault in an open architecture scenario.

On the other hand, open standard Fieldbus, which consists of numerous automation vendors, offers better product range and varieties. These different vendors' devices must be able to co-exist on the same network, thus interoperability and interchangeability is of paramount importance. For example, DeviceNet is an open standard fieldbus which consist of more than 130 vendors such as ABB Robotics, Allen Bradley, Hewlett-Packard, Hitachi, Omron, and Mitsubishi. More vendors means better product choice. In theory, the open standard fieldbus offers the users the 'pick and mix' solution. However, when things go wrong, it may be difficult to nail down which vendor's device is responsible for the fault. To prevent this problem from surfacing, all devices must undergo strict interoperability and conformance exercise before delivery to the end-users. Nevertheless, open standard tends to survive longer than a proprietary solution and become the de-facto standard.

The most important criteria in selecting a fieldbus network are not technical but political [35]. However, this does not mean that the technical constraints are unimportant. The technical constraints should always reflect the real user needs, which include the organisational and management constraints. For example, although fieldbus A is better than fieldbus B technically, fieldbus B may be chosen as the solution based on management grounds that it is more cost effective and readily available. In addition, there is no global solution as to which fieldbus to use. A successful implementation in one organisation may result in total disaster in another organisation. This is why there are so many proprietary and open fieldbus standards available to date.

There are a few general guidelines to consider when choosing the right network for the right level of operation. These include the network size, real-time capability and data volume requirements. Table 2.1 (Page 2-7) summarises the characteristics of each level's network in the hierarchy.

2.2.3 Intelligent instruments

The use of fieldbus has contributed to the development of intelligent instrument for industrial automation. In order for the fieldbus communication to take place, the field devices are now furnished with microprocessors and digital interfaces. With an onboard microprocessor, the instrument can now perform some 'thinking' process, hence the birth of intelligent devices. An intelligent device is more aware of its environment and surroundings through its sensors and clever software algorithm. The availability of low cost but more powerful silicon has also helped to make the dream of intelligent devices a reality. The development has enabled some of the intelligence or features previously resident in automation computers to be installed in these intelligent field devices.

2.2.4 Intrinsically Safe Fieldbus

The intrinsically safe version of fieldbus allows fieldbus technology to be used in the chemical and petro-chemical industries. This involves the use of special electronic line drivers to prevent ignition arising from electrical short-circuits or temperature rises. Essentially, they are the same as the non-intrinsically safe devices, except that special interface protection allows them to be used in hazardous conditions. A point must be made that intrinsically safe fieldbus is never certified for safety critical applications such as nuclear plant.

2.2.5 ISO/OSI 7-layer model[88]

As mentioned before, fieldbus is a low level industrial computer network optimised for real-time control. At the simplest form, it serves as a common communication medium which links the sensors, actuators and instruments in the manufacturing process control using a single cable bus. This scenario of networking sensors and actuators using fieldbus is analogous to networking computers in offices for information and resource sharing through a single wire bus. The difference is that conventional TCP/IP systems that use the Ethernet signalling as the backbone are optimised for file transfers. Data packet size of 1,500 bytes is typical. In addition to this, Ethernet signalling is non-deterministic and deteriorates exponentially when transmission bandwidth overshoots 33%. These higher level communication networks are not suitable for real-time control purpose. Since fieldbus is a form of computer network, it can be referenced using the ISO 7498 OSI model as shown in Figure 2.5.

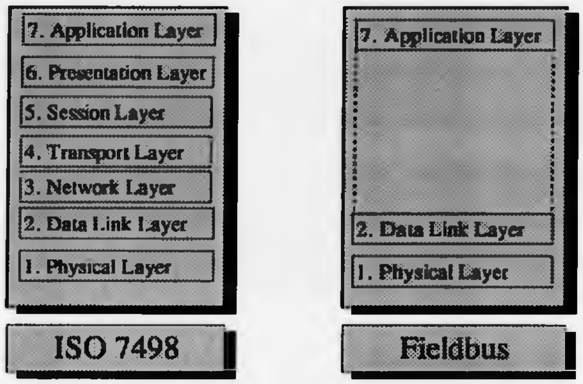


Figure 2-5 The ISO 7 layer model and reduced stack fieldbus model

As all fieldbusses use base-band transmission technique, they do not need to implement the full 7 layers of the OSI. Furthermore, more layers mean more time overheads involved, which is not desired in the real-time control entity. Therefore, all fieldbus protocols use the reduced stack 7 layer OSI model (i.e. the 3 layer model) for fast network response time as shown in Figure 2.5.

2.3 Discrete Control System

Fieldbus control system is discrete in nature where the system monitors the process under control at a discrete distant of time. The time distant maybe 500µs, 6ms, 30ms etc. depending on the criticality of the real-time system. In general, the discrete control system utilising fieldbus technology falls into two categories, i.e. event-triggered or time-triggered control systems.

2.3.1 Time-Triggered

Time-triggered system, also known as ***sampled-data*** system, involves the sampling of data and execution of control sequence at predefined time interval. This approach allows synchronisation and simultaneity in carrying out the control sequence. The fundamental mechanism in realising this method of control, i.e. time-triggered system, is through the use of polling. All the device's data on the fieldbus are polled and their states are updated at a discrete time interval. The advantage of this approach is that the network bus traffic is predefined. The bus loading for a time-triggered fieldbus is almost constant, regardless of devices' activities.

2.3.2 Event-Triggered

On the other hand, ***event-triggered*** system only acts when there is a change of state within the device occurring. For example, if a switch is not triggered, there will not be any message on the bus. The notion of this event-triggered system allows lower bus loading when there is not much activity going on, except for the 'heart-beat' messages. However, when everything starts to happen at once, there is a risk of overloading the transmission bandwidth, or delaying of time-critical messages if not designed properly.

Fieldbus utilising the event-triggered approach normally has special priority allotted for the safety critical and time critical data. For example, when an alarm condition is encountered, a special high priority message may be used to ensure that the safety and time critical alarm message do get through irrespective of the network traffic condition. Nevertheless, there will always be some network traffic for the "heart-beat" message[2] when the system is idle. In real-life fieldbus implementation, a mixture of the event-triggered and time-triggered approach is used.

2.3.3 Time Consistency

Time consistency means that the set of values available on a given control device is identical to the samples of the states of the next lower devices at the same sampling instant [36]. For example, the DeviceNet scanner will issue the bit-strobe command to sample the data from all the devices (i.e. lower devices) on its scan list. If the system fulfils the time consistency criterion, then all the data values on the lower devices, e.g. photosensor input, flex input/output, pneumatics on/off states will be the same as those data values sampled by the scanner at the same sampling instant.

2.3.4 Space Consistency

Space consistency means that the value of the sampled data will be the same and identical throughout the network regardless of where the controller is situated, i.e. data consistency throughout the network [36]. The global acknowledgement of Controller Area Network (CAN) protocol, where all the

[2] "Heart-Beat" message is the message to tell the consumer of the data (client) that the producer of the data (server) is still alive. This will enable the proper countermeasure to be taken by the control system should the server become faulty and fail to provide the required parameters.

nodes receive, verify and acknowledge the same data fulfils the spatial consistency of fieldbus implementation. In other words, space consistency ensures that different copies of the data transmitted will be identical at the same sampling instant.

2.4 Conformance to the Open Standard Fieldbus

Conformance is very important in a multi-vendor, multi-product open standard scenario. The effect of not conforming to the open standard can lead to the loss of user confidence on the standard. For example, when IBM opened the PC architecture to the vendors, only those 100% compatible PCs survived the strong competition. Those vendors with 95% compatibility systems eventually lost their share of the market. The lesson learnt from the personal computer industry can be applied to the open standard fieldbus industry in general.

What happens if we have a 95% conformed device for an open standard fieldbus? At some point during its operation, the device will disrupt the network due to the incompatibility problem. This compatibility issue has resulted in the extensive use of the term interoperability. The interoperability of devices becomes the acid test for an open standard fieldbus. Every open standard fieldbus is trying to get the devices to work harmoniously on the same network, even if they come from different vendors.

Interoperability and interchangeability are the buzzwords C. Ajluni [37] used to describe the open standard fieldbus devices. It is also the intention of this thesis to look into the interoperability and interchangeability aspects of an open standard fieldbus, i.e. DeviceNet. This leads to the proposed design and development of a conformance test engine for the conformance testing of DeviceNet devices. The interoperability testing is discussed in Section 4.5.5.

2.5 Future of Fieldbus

The use of fieldbus is almost certain to increase in factory automation as well as building automation. Fieldbus will do to the control and instrumentation industry what the networked PCs did to the office automation. To date, however, there is no one fieldbus that caters for all of the manufacturing world's needs. The scenario always consists of various fieldbus standards working in concert to achieve this new era in production control. For example, Profibus, WorldFIP, Lonworks, CAN based network (e.g. DeviceNet, SDS) and others have different roles to play in the scene. Conversely, each company's choice of fieldbus is unlikely to be based on the technical aspects. Already different suppliers of automation control system hardware are pushing different fieldbusses, with the result that, even though the fieldbus may not be proprietary, the choice of fieldbus and supplier are not independent decisions.

One of the major reasons why fieldbuses have not been used more widely is the lack of availability of automation products with the communication capability. This will ease over the next year or two, but the fieldbus that is most likely to be widely adopted will be the one which has the widest range of compatible products in the market first and a large installed nodes.

In another development, the European Comission has initiated the Pre-Normative Requirements for Intelligent Actuation and Measurement (PRIAM) project to study the implication of intelligence in field devices [38]. PRIAM (an ESPRIT project) has also developed many prototype tools to assist effective application of the fieldbus technology in Europe. In order to review and enhance the findings of PRIAM project, the European Intelligent Actuation and Measurement Group (EIAMUG) was formed in 1994. The goal of EIAMUG is to shift the evolution of plant automation from current emphasis on control towards an integrated structure of control, maintenance and technical management function, or Control, Maintenance and technical Management (CMM) in short [39].

CHAPTER 3

THE DEVICENET FIELDBUS

DeviceNet is one of the low level open standard fieldbusses suitable for real-time control in industrial applications. It uses the proven Controller Area Network (CAN) technology as a backbone. CAN's high performance error detection mechanism and good electromagnetic immunity make DeviceNet feasible to be used safely in the noisy factory environment. The high volume of CAN silicon used in production cars make DeviceNet implementations cost effective in simple, low cost field devices.

3.1 Overview of CAN

CAN was originally designed as in-vehicle network by Bosch, Germany in 1986 for distributed real-time control [40]. Work on designing CAN protocol began in 1981 when the engineers at Bosch were faced with the task of establishing real-time communication between three electronic control units (ECUs), i.e. engine management unit, anti-lock brakes and automatic transmission control. Conventional universal asynchronous receiver/transceiver (UART) interface for point-to-point communication (one-to-one relationship) has immediately became unusable for multi-microcontrollers communication (many-to-many relationships). In addition, the use of electronics in vehicle is increasing each year. These electronics systems are getting more complex and sophisticated which resulted in the increase size of the wiring harness. Today's

Mondeo for example, uses a total of 1.5km of wiring [41]. It was apparent that CAN protocol needed to be developed not only to replace the conventional UART interface, but also to reduce the wiring harness so as to increase the manufacturability of modern vehicles.

The term Controller Area Network (CAN) was coined by Professor Lawrence of Fachhochschule Wolfenbüttel in Germany as the design neared its completion. After the protocol launch in 1986, the first CAN silicon was available from Intel in summer 1987. The first production car using the CAN technology was rolled out from production in 1991. CAN was used to inter-link the 5 electronic control units (ECUs) of this luxury car, Mercedes S Class at a baud rate of 500 kbit/s [40].

3.1.1 The ISO 11898 Standard

By 1993, CAN became the ISO 11898 [42] and ISO 11519-2 [43] standards for information exchange and real-time control in road vehicles. It has been widely used on many production cars including BMW [44], Jaguar [45] and Mercedes for high speed information exchange and real-time control between Electronic Control Units (ECUs) such as the engine management unit and Anti-lock braking System (ABS). Other innovative CAN automotive applications include the Mercedes-Benz's Electronic Stability Program (ESP - where driving on a frozen lake without losing control is possible) and Lucas's intelligent brake developed under PROgraMme for a European Traffic with Highest Efficiency and Unprecedented Safety (PROMETHEUS) project, as well as Arnold Schwarzenegger's High Mobility Modular Wheeled Vehicle (HUMMER) [41].

3.1.2 CAN Industrial Applications

Today, CAN technology is not only being used extensively in the automotive sectors, it can also be found in many industrial applications. Until 1995, over 10 million CAN chips were sold, of which the automotive industry only accounted for 3 million, while the other 6 million chips were used in non-automotive applications [46]. This reflected the popularity and wide acceptance of the technology in industry [47, 48, 58]. The use of Carrier-Sense Multiple Access/Collision Detection + Non-Destructive Bitwise Arbitration (CSMA/CD + NDBA) technique for guaranteed network latency time and a bit rate of up to 1Mbit/s makes CAN an ideal control network in a wide variety of real-time applications [49, 50]. Its high immunity against electromagnetic noise enable it to work in harsh electromagnetic environments [51, 52].

In addition, the high quantity of CAN chips available from Intel, Philips, Motorola, National Semiconductor, NEC and Siemens for automotive applications, helps lower the CAN implementation costs. The cost effectiveness of CAN technology allows network interface to be implemented directly onto low cost devices such as the proximity switches and photosensors in the automation field. With its many technical advantages, it is evident that the cost effective CAN technology can be used in industrial automation, with many other applications that are yet to be discovered [43, 53].

3.1.3 Higher Layer Protocols using CAN

In order to adapt the CAN technology efficiently in the automation world, new application layer protocols and standards need to be defined to realise a truly open networking standard. Many higher layer protocols based on CAN technology have been developed in recent years for industrial applications. Among them are CANOpen (CAN Application Layer profiles) [17, 18], CAN Kingdom [19], DeviceNet [53] and Smart Distributed System (SDS) [22, 23] open standard fieldbusses. There are also numerous proprietary CAN industrial implementations which include the marine navigational equipment (NAVICO), Philips medical equipment [54], aerospace electronics [55, 56], lift controls, building automation and access control [57].

3.2 DeviceNet - A Higher Layer Protocol

DeviceNet protocol was designed by Allen Bradley, USA for industrial automation. Since its launch in Spring 1994, DeviceNet has joined the many open standard fieldbusses available today for real-time control and communication in the process control automation. A year later, DeviceNet had become the property of the Open DeviceNet Vendors Association (ODVA). ODVA consists of a consortium of companies who developed DeviceNet products.

Even though CAN network can support numerous bus rates, DeviceNet standard chooses to support only 3 operating baud rates, i.e. 125 kbit/s, 250 kbit/s and 500kbit/s. According to a conducted survey [58], more than 66% of fieldbus applications require a bus length of less than 100 metres, and only 3% of the fieldbus applications require a length of more than 1000 metres. This statistic makes DeviceNet, which supports a maximum bus length of 500 metres (refer Table 3-1), an ideal candidate for networking sensors and actuators in automation islands.

Table 3-1 The DeviceNet Baud Rate and Bus length

Baud Rate	**Bus Length (Metres)**
500 kbit/s	100
250 kbit/s	250
125 kbit/s†	500

† All DeviceNet devices are default to operate at 125kbit/s.

3.2.1 The Network Topology

DeviceNet network uses linear bus network topology with 121Ω terminators (1% metal film, ¼ Watt resistor) at both ends of the trunk line. The bus network topology allows easy node connection and detachment from the trunk, without affecting other communicating nodes. The network can also be configured with various branches on the drop lines [59], thereby enhancing the flexibility of the network configuration. Each DeviceNet cable consists of 5 wires for CAN High and CAN Low signals, Power (+24V), Ground and Shielding. The power bus allows a maximum of 8 amperes of current to be drawn from any point of the thick trunk cable. This allows simple devices such as photosensors and proximity switches to be powered directly from the DeviceNet network. There are restrictions which govern the power consumption and cumulative drop length defined in the DeviceNet specification.

A DeviceNet network is capable of supporting 64 physical nodes, with up to 6 meters of drops via the tee-junction at the trunk line. The long drop cables offer the flexibility to route DeviceNet devices at difficult corners. Every DeviceNet device must have a unique Media Access Control Identifier (MACID) such that no two nodes will transmit the same data packet. The valid

MACIDs are from 0 to 63, with MACID 63 being default for non-configured device. The MACID can be freely assigned so long as they do not conflict with each other, i.e. two nodes being assigned the same MACID.

If 2 or more DeviceNet systems were to be connected on a single bus, the maximum number of nodes per system must not exceed 64 (i.e. only 64 different MACIDs are allowed on any network bus.). For instance, there may be 2 DeviceNet Master/Slave systems, one with 50 nodes, the other with 14 nodes, which utilise the same physical media. Even though connecting more than 64 physical nodes is possible, e.g. by connecting a passive network analyser node (without any assigned MACID), this must be prohibited for any operating network. The additional node may upset the impedance characteristic of the network, or even damage the bus driver of other nodes which are designed to sink and source a maximum of 64 nodes. A specially designed high impedance, low power bus driver may be used for network analyser equipment to minimise the disturbance on the network.

3.2.2 The DeviceNet Layered Architecture

The ISO 7 Layer OSI model has been used as a reference for DeviceNet protocol development. DeviceNet protocol is a superset of the existing ISO 11898 standard, which is CAN. To complete the DeviceNet protocol definition, an application layer optimised for real-time control and the Media Access Unit (MAU, i.e. cable) were defined on top of the ISO 11898. The Data Link and Physical Layer of ISO 11898 standard were adopted from the existing ISO 8802-2 and ISO 8802-3 Local Area Network (LAN) standards. As DeviceNet uses the ISO 11898 standard, it also adopted the ISO 8802-2 and 3 for the corresponding 2 layers of the OSI stack (ISO 7498). Figure 3-1 shows the relationships between ISO 7498, ISO 11898 and DeviceNet.

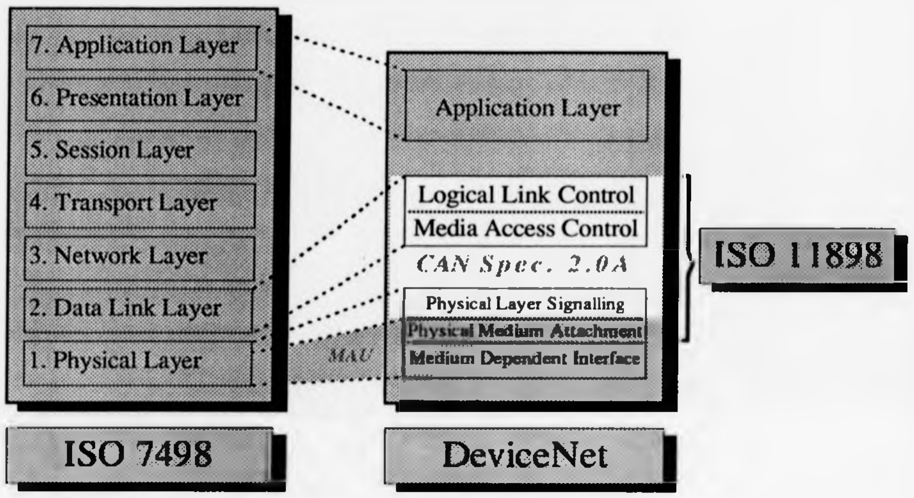


Figure 3-1 DeviceNet and its reference to the OSI 7 layer model

According to ISO 8802-2, the Data Link Layer is further divided into :-

- Logical Link Control (LLC), and
- Medium Access Control (MAC).

The LLC sublayer is responsible for functions associated with the acceptance filtering of the CAN frames, as well as the notification of bus overloading condition. The automatic retransmission of corrupted CAN frames is also the responsibility of the LLC.

MAC on the other hand will perform the role associated with:-

a) the construction of message frames

   i.e., by adding SOF (Start-of-Frame), Arbitration Field (11-bit Identifier and remote-transmission-request (RTR)), Control Field (2 reserved bits and DLC-Data Length Code), data field, CRC field, acknowledgement field and end-of-frame (EOF). It is responsible for transferring these bits on the wire, starting with SOF.

b) error detection and signalling facilities of CAN protocol.

   i.e. to perform the CRC checksum sequence and construct the appropriate Error Flags (Error Active/Passive)for error signalling, and validating the message frame against Bit/Stuff/CRC/Form/ACK Error.

c) acknowledge **all** CAN frames provided no error has been found in (b)

d) provide the interface to the Fault Confinement Entity to realise the various states in fault confinement such as Error Passive and Error Active states.

The ISO 8802-3 subdivides the Physical Layer of the ISO 7498 into :-

- Physical Signalling (PLS),
- Physical Medium Attachment (PMA), and
- Medium Dependent Interface (MDI)

The Physical Signalling defines the CAN bit stream encoding/decoding and bit time synchronisation. This includes the definition and programming of bit time partition, i.e. the SYN_SEG, PROP_SEG, PHASE_SEG_1, PHASE_SEG_2. Modern CAN controllers have combined the PROP_SEG and the PHASE_SEG1, and called it TSEG_1 [60, 61].

3.2.3 Medium Access Unit (MAU)

Medium Access Unit (MAU) refers to the part where the physical layer couples the node to the transmission medium. It consists of:-

- Physical Medium Attachment (PMA), and
- Medium Dependent Interface (MDI) (refer Figure 3-1).

In the context of DeviceNet, MDI simply refers to the DeviceNet open, sealed micro or mini style connectors. The PMA in this instance will be the Philips 82C250 bus driver, or any driver configuration that is capable of realising the CAN high speed physical signalling as shown in Figure 3-2.

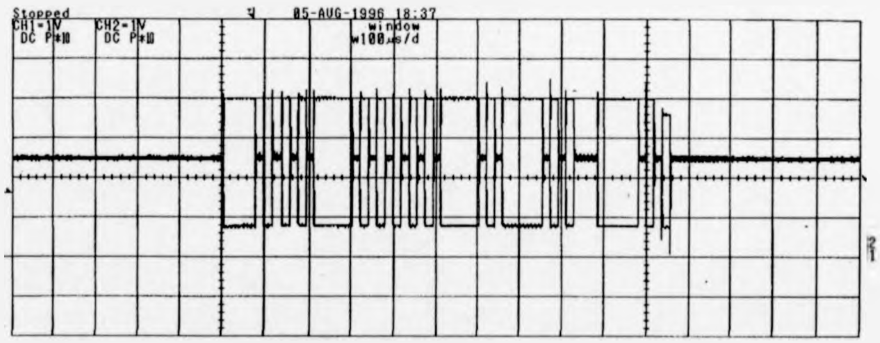


Figure 3-2 An Example of the ISO 11898 high speed physical signalling

3.3 DeviceNet Messaging Scheme

DeviceNet Messaging Scheme is responsible for all data transfers in the protocol. It partitions the 11-bit CAN identifiers into 4 main message groups, i.e. Message Group 1, Group 2, Group 3 and Group 4. These message groups have differing priority in bus arbitration, where Group 1 Messages command the highest priority. (In CAN, the lower the Identifier number, the higher the priority of the message.) Table 3-2 shows the breakdown of the message groups in the CAN identifier field. The 11-bit CAN Identifiers are referred to in later sections as the **Connection ID (CID)**.

Table 3-2 The DeviceNet Message Groups [62]

11-bit CAN Identifiers											Range in Hex	
10	9	8	7	6	5	4	3	2	1	0		
0	Group 1 Message ID				Source MAC ID						000-3FF	Message Group 1
1	0	MAC ID						Group 2 Message ID			400-5FF	Message Group 2
1	1	Group 3 Message ID			Source MAC ID						600-7BF	Message Group 3
1	1	1	1	1	Group 4 Message ID						7C0-7EF	Message Group 4
1	1	1	1	1	1	1	X	X	X	X	7F0-7FF	Invalid CAN Identifiers

Having defined the message groups within the CAN identifier field, DeviceNet protocol uses the CAN *data field* to define the protocol information in certain instances. This further expands the messaging capability of the DeviceNet protocol. Figure 3-3 shows the DeviceNet messaging scheme responsible for data transfers.

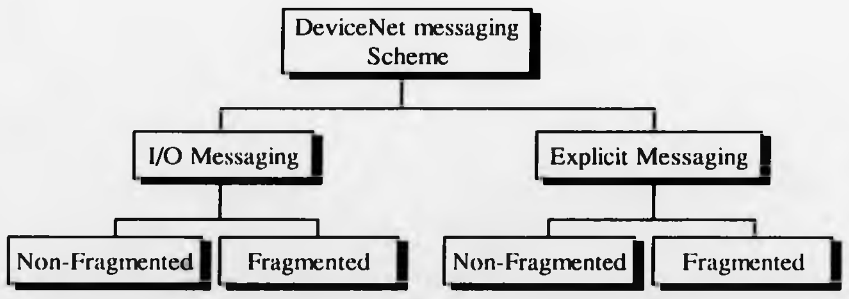


Figure 3-3 The summary of DeviceNet Messages

The I/O messages of DeviceNet utilise Message Group 1 for highest priority bus access and are suitable for real-time communication. Explicit Messages occupy Message Group 2 for point-to-point communication, and are more suited for application with fewer time critical messages. From Table 3-2, it can be seen that I/O messages (Group 1) have a higher priority than those explicit messages (Group 2). This corresponds with the CAN protocol where the lower the CAN identifier, the higher the priority for bus arbitration.

3.3.1 Explicit Messaging

All communications in DeviceNet starts with the Explicit Messaging. As mentioned earlier, the explicit message of DeviceNet uses the CAN data field to carry the protocol information. Explicit message divides the 8-byte CAN data field into:-

- Message Header, and
- Message Body.

An instance of the Explicit Message is *the Open Explicit Messaging Request/Response* messages as shown in Figure 3-4. The first column of the table shows the data byte offset in the CAN data field, while the remaining eight columns represent a single data byte. Analysing the *Open Explicit Messaging Connection Request* message reveals the first data byte (which is shaded) as the **Message Header** of the Explicit Message, while the remaining three bytes are referred to as the **Message Body**. It can be seen that the Open *Explicit Messaging Connection Request* is a four byte message, while the Open *Explicit Messaging Connection Response* message consists of six data bytes. Explicit Message with a **Message Body** of more than seven bytes will be transmitted under the **Fragmentation Protocol**, which will be discussed in Section 3.3.3.

Open Explicit Messaging Connection Request

	7	6	5	4	3	2	1	0
0	Frag (0)	XID	MACID					
1	R/R (0)	Service Code (4B)						
2	Reserved bits All "0"				Requested Message Body Format			
3	Group Select				Source Message ID			

Open Explicit Messaging Connection Response

	7	6	5	4	3	2	1	0
0	Frag (0)	XID	MACID					
1	R/R (1)	Service Code (4B)						
2	Reserved bits All "0"				Actual Message Body Format			
3	Destination Message ID				Source Message ID			
4	Connection Instance							
5	ID							

Figure 3-4 Open Explicit Messaging Connection Request/Response Message [62]

3.3.2 I/O Messaging

Explicit Messaging forms the fundamental communication mechanism of DeviceNet. Explicit Messaging is responsible for establishing the logical connections between nodes on the network. Having established the logical link between the producer and the consumer of the data, the devices can implicitly use the I/O messaging for very fast data exchange. Therefore, I/O messages are only useful to those nodes who have prior knowledge on the Protocol Data Unit (PDU)'s data. To a casual observer such as a bus analyser, the I/O messages will be meaningless, unless this casual observer is present during the establishment process of the logical connections. PDU using the I/O Messaging scheme does not have any protocol information encoded in the CAN data field, with the exception of the Fragmented I/O Message.

3.3.3 Fragmentation Protocol

It is possible to transmit information which consists of more than 8 data bytes per PDU in DeviceNet using the Fragmentation Protocol. Fragmentation protocol supports both the I/O Messaging and Explicit Messaging. The Fragmentation protocol adds a byte of overhead into the existing data format of both I/O and Explicit Messages. As a result, Fragmented I/O messages only transfer 7 bytes of data in one PDU, whilst the **Message Body** of Fragmented Explicit Message is reduced to 6 bytes per PDU.

Both I/O and Explicit Fragmentation Protocol Messages format are shown in Figure 3-5. The shaded areas are responsible for *Fragmentation Protocol*. Notice that the *I/O Fragmentation Protocol* uses a data byte less than the *Explicit Fragmentation Protocol*. The *Fragment Type* indicates whether the PDU is the first fragment (0), middle fragment (1) or last fragment (2). The first fragment must have the value 0 or 3F hex (111111 binary) in the *fragment count* field in order to be valid. Other successive PDUs that follow the first fragment must have the *middle fragment* indicated in the *fragment type* field. The fragment count will be incremented according to the following formulae.

I/O Fragmented Message Format

	7	6	5	4	3	2	1	0
0	Fragment Type		Fragment Count					
. . . . n†	I/O Message Fragment							

Explicit Fragmented Message Format

	7	6	5	4	3	2	1	0
0	Frag (1)	XID	MACID					
1	Fragment Type		Fragment Count					
. . . . . n†	Explicit Message Body Fragment							

† n ≤ 7

Figure 3-5 The Fragmentation Protocol Message Format [62]

$$\boxed{\text{Fragment Count} = (\text{Fragment Count} + 1) \bmod 64}$$

When the transmitter reaches its last fragmented PDU, it will indicate to the receiver by putting the value 2 in the *fragment type* field to indicate that it's the *last fragment*. In the Explicit Fragmented Messaging scenario, the receiver must acknowledge the reception of the each fragment by transmitting a value 3 in the *fragment type* field. The DeviceNet Specification, however, does not define whether the receiver needs to echo back the fragment count and data bytes to the transmitter, or just acknowledge with a value 3 in the *fragment type* field.

With fragmentation protocol, the maximum network efficiency[1] for I/O messages has dropped from 59% for the full 8 data byte PDU to 38%. Since the Message Body of Explicit Message has been reduced by a byte, the best network efficiency achievable is 44%, i.e. after a reduction of 7% efficiency. (Note : The network efficiency is calculated without taking into account the CAN protocol's stuff bits.)

In theory, there is no limit for the number of data bytes to be transmitted via the fragmentation protocol. By analysing the network efficiency, it is obvious that transferring data longer than 8 bytes is not efficient in DeviceNet. Most of the fragmentation data consist of the non time critical information such as the transmissions of Serial Number and Product type during initialisation phase of the network. The use of fragmentation messages is best avoided during run-time. If circumstances allow, it is preferable to keep the fragmented I/O messages to a minimum for hard real-time systems.

3.4 Application Layer Optimised for Control

The application layer of DeviceNet is optimised for real-time control. It can be divided into the following two categories:-

- client/server for low priority point-to-point messaging
- multicast for real-time I/O data

[1] Network Efficiency refers to the ratio of the amount of the user data and the number of bits needed to transmit such information (as defined by [23, Durante et al.])

3.4.1 Unconnected Message Manager (UCMM)

The Unconnected Message Manager or UCMM in short is responsible for establishing the logical connection between two nodes at initial stage. In DeviceNet, the data producer and data consumer must be linked logically using network binding tools such as the DeviceNet Manager Software. This network binding tool creates a logical link in which both devices, i.e. the producer and consumer of the data, communicate. Even though one can easily connect a device on the network, the device will not do anything until it has been configured, i.e. established connection. Therefore the term “plug-and-play” is perhaps not applicable to DeviceNet yet.

From section 3.3.1, we learnt that the Explicit Messaging is the first messaging scheme a device will use in DeviceNet. When a DeviceNet node is plugged into the network, logical link must be established prior to the consumption and production of data. In fact, the *Duplicate MACID Check Message* that a device transmits before going online is an explicit message. This message ensures a unique MACID is assigned before the new device is allowed to go on-line. The network binding tool will establish connection with the new device via the UCMM port using the *Open Explicit Messaging Connection Request Message*. The UCMM only processes 3 instances of DeviceNet network service message. They are:-

1. Open Explicit Messaging Connection service (Service Code 4B hex)
2. Close Explicit Messaging Connection service (Service Code 4C hex), and
3. Error Response service (Service Code, 14 hex) messages

The UCMM of a device can be analogous to a door-man of a hotel building. The person can either open the door to let you enter, close the door when you go out, or refuse to let you enter at all. If the connection instance is available and free, the UCMM will reply with *the Open Explicit Messaging Connection Response Message*, indicating a successful transaction with the appropriate parameters. All the succeeding transactions no longer involve the UCMM, but use the *Connection Object Class*, unless the following happen.

a) A *Close Explicit Messaging Connection Service* is requested, in which the UCMM closes the existing connection instance.
b) Another *Open Explicit Messaging Connection Service* is requested. This situation will cause the UCMM to send an *error response* if there is no available connection for the request (i.e. the device only has 1 connection instance), or reply with a *success response* if there is another free connection instance available (i.e. for device which supports multiple connection instances).

Since UCMM related messages do not utilise the Connection Object, they are not connection based messages, i.e. they do not occupy any available connection instance of the Connection Object Class.

3.4.2 The Connection Object Class

After successfully gaining access to the DeviceNet device via UCMM, the connection must be routed (by Message Router Object) to the Connection Object of the device for connection based messaging. The Connection Object Class allocates and manages the I/O and Explicit Messaging Connections of the device. It uses the services provided by the **Link Producer Object Class** and/or **Link Consumer Object Class** for transmission and reception of data. Again, the Connection Object can be analogous to a receptionist in a hotel. The receptionist will register the guest and issue the guest with a room key, before using the porter service to see the guest to his/her room. Once the guest knows the orientation of the hotel, it is possible for the guest to leave the hotel room and return, without the assistance of the receptionist. A "path" has been created in the mind map of the guest to allow him/her to shuttle between the hotel room and his/her destination.

In the context of DeviceNet, a unique connection instance will be created between the Connection Object Class of the new device and the tool. This instance will be assigned a unique Connection ID (CID) which will be used for subsequent transactions. The tool then configures the Connection Object Instance attributes by using the services provided by the Connection Object Class. It uses the *create service* of the Connection Object Class to instantiate a connection of your choice. For example, one can create an *I/O Messaging Connection* with attributes such as *change-of-state* **Transport Class** triggering (using transportclass_trigger attribute) connection with maximum PDU length of 4 bytes (using produced_connection_size attribute) and an EPR (expected_packet_rate) time-out of 200ms. When the instantiation process is completed, i.e. established connection, all subsequent transmissions associated with that connection instance will use the Group 1 I/O Messaging scheme with an assigned CID.

3.4.3 Application Object Class

Application Object Class of DeviceNet is responsible for providing the function of the device itself to the acquiring client end-point. The presence sensing of a photo-sensor (i.e. the ON/OFF state of the photo-sensor) is an instance of the Application Object Class.

Before the presence sensing data of the photo-sensor can be utilised, the logical connection between both the client and server end-points must be initialised via the Explicit Messaging connection. Once the Explicit Messaging connection with the Connection Object Class messaging is established, the other end-point of this logical link can require any attribute and service provided by this new node. If the service of the Application Object is needed, the Connection Object Class will utilise those services provided by the **Link Producer Object Class** and/or **Link Consumer Object Class**, i.e.

- create a connection instance
- delete a connection instance

for establishing the necessary connections.

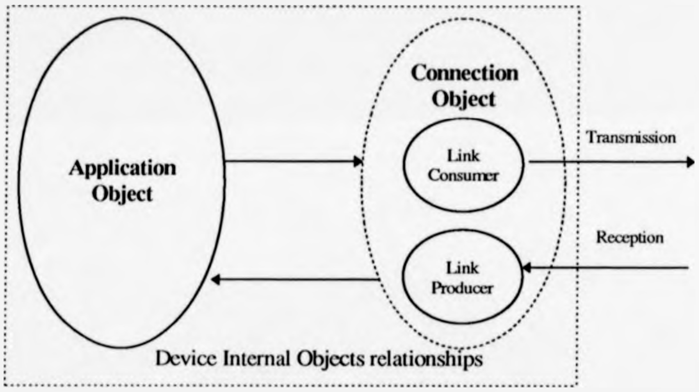
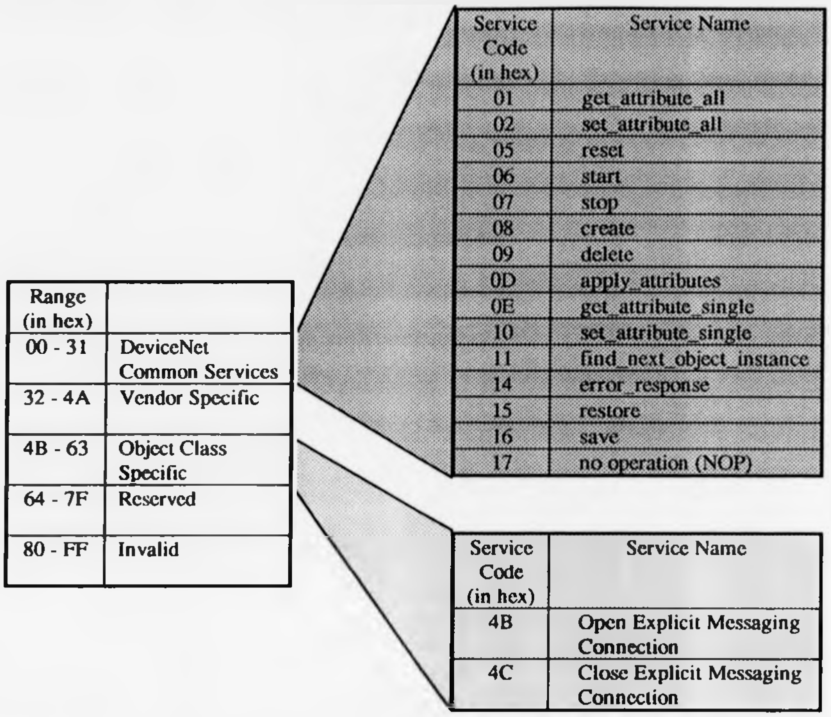


Figure 3-6 The interface between Application Object and Connection Object

For example, a connection instance between the Application Object of the photo-sensor and the PLC Scanner's object must be established before ON/OFF states of the photo-sensor can be consumed by the PLC Scanner. Figure 3-6 shows the internal object relationships of a DeviceNet device, e.g. photo-sensor, to enable the production and consumption of data (i.e. logical connection).

3.4.4 Network Services

DeviceNet protocol defines a standard set of network services that all DeviceNet compliant devices must implement. Figure 3-7 summarises all the DeviceNet network services available to-date. The figure also shows that vendor specific services are possible by implementing these services within the allocated area, i.e. service code 32-4A hex. All these services can only be realised using the Explicit Messaging Connection scheme. This allows many resources to be implemented, accessed and modified by the corresponding application. For instance, the user can modify the dark sensing of the photo-sensor into light-sensing, and save the modification in the device by initiating a set_attribute_single service with the appropriate parameter attributes.



Range (in hex)	
00 - 31	DeviceNet Common Services
32 - 4A	Vendor Specific
4B - 63	Object Class Specific
64 - 7F	Reserved
80 - FF	Invalid

Service Code (in hex)	Service Name
01	get_attribute_all
02	set_attribute_all
05	reset
06	start
07	stop
08	create
09	delete
0D	apply_attributes
0E	get_attribute_single
10	set_attribute_single
11	find_next_object_instance
14	error_response
15	restore
16	save
17	no operation (NOP)

Service Code (in hex)	Service Name
4B	Open Explicit Messaging Connection
4C	Close Explicit Messaging Connection

Figure 3-7 The DeviceNet Network services

3.5 DeviceNet Network Error Management

Much of DeviceNet error management is relying on the proven CAN technology. DeviceNet has included additional definition of error recovery and management to those already available in the CAN technology.

3.5.1 Error Detection in CAN

CAN does not have error correction facility. It only has the error detection mechanism and relying on the automatic retransmission of corrupted messages. Reference [63] provides a detailed analysis on the error detection mechanisms in the CAN protocol. There are 5 error detection mechanisms available in CAN. They are:-

3.5.1.1 Acknowledgement

All CAN nodes acknowledge when an error free message frame is received. The missing of acknowledgement is interpreted as an error by the transmitter.

3.5.1.2 Monitoring

Every transmitting node checks the bit stream detected on bus with its transmitted bits. This bit-by-bit checking of the transmitter guarantees safe detection of all global and local errors. A global error is the error agreed by all nodes on the system. An example of global error may be a voltage spike on the trunk line which was seen by all nodes. A local error is an error caused and/or detected by the local node only. For instance, when a node transmits a "1" on the bus, it expects to receive a "1" back. If it does not see a "1" on the bus, an error has occurred. An exception is that the transmitter allows its transmitted recessive bits ("1") to be overwritten by dominant bits ("0") at the arbitration field and the acknowledgement slot of CAN frame.

3.5.1.3 Frame Check

The CAN protocol contains fixed format bit fields for each message frame. All nodes check for the consistency of the frame format. Any mismatch on the frame format is interpreted as an error.

3.5.1.4 Cyclic Redundancy Check (CRC)

The CRC sequence in CAN is calculated by dividing the polynomial, f with the generator polynomial, gp,

$$\text{where } gp = x^{15} + x^{14} + x^{10} + x^{8} + x^{7} + x^{4} + x^{3} + 1$$

The coefficients of polynomial f, (calculated using modulo-2) consist of the destuffed bit stream of Start-of-Frame (SOF), Arbitration Field, Control Field and Data Field (if any). The remaining 15 lowest coefficients of the polynomial f are all set to 0. The remainder of the polynomial division (15-bit value) is then transmitted by the transmitter. At the receiver, the same process is repeated to obtain the remainder of the polynomial division. Any mismatch in the remainder checksum is interpreted as error.

3.5.1.5 Bit Stuffing

A stuff bit of the opposite polarity is inserted upon detection of 5 consecutive bits of the same polarity. The transmitting node will stuff the bit stream while the receiver will de-stuff appropriately. If a bit stream consists of more than 5 successive bits of the same polarity during the transmission of the CAN frame, an error is detected.

3.5.2 Fault Confinement in CAN

Table 3-3 shows the fault confinement of CAN protocol. If an error occurs during the transmission of a message, the Transmit Error Counter (TEC) on the transmitter will be incremented by a value 8. All other nodes (receivers) on the network will increment their Receive Error Counter (REC) by 1. This mechanism ensures that the faulty transmitter will enter the bus-off state before causing catastrophic failure to the operating network. Both the REC and TEC will be decrement by 1 every time a good message frame is received and acknowledged. This allows the nodes to recover from temporary error or short disturbances.

Table 3-3 The Transmit Error Counter (TEC)/Receive Error Counter (REC) values with respect to their corresponding states.

Error-Active	$0 < TEC \leq 127$, or $0 < REC \leq 127$
Error-Passive	$128 < TEC \leq 255$, or $128 < REC \leq 255$
Bus-Off	$255 < TEC$, or $255 < REC$

If one of the many receivers on a network is faulty, the following situation will happen. Since the receiver is faulty, it will disagree with other receiving nodes in validating a CAN message frame. When the receiver interprets the valid CAN frame as an error, it will signal the error condition by transmitting an active error frame (assume all devices are in normal state prior to this). Because all other nodes on the network are interpreting the CAN message as a perfectly good message frame, the appearance of the active error frame (transmitted by the faulty receiver) violates the bit stuffing rule (a maximum of 5 consecutive bits of the same polarity are allowed) in the CAN protocol.

At this stage, all the healthy receivers will collectively transmit their active error flags; not to indicate that the message frame is corrupted, but to indicate the bit-stuffing rule violation. The faulty receiver which started the transmission of the active error frame will now detect more error flags on the bus due to the second wave of error frames transmitted by the healthy nodes.

In this situation, the faulty receiver that transmitted the active error frame, and greeted by more active error frames will have to increment its REC by a value of 8. All other receivers will increment their REC by 1. The transmitting node which is involved in this scenario will increment its TEC by 8.

Whenever a healthy CAN message is received and acknowledged, the TEC and REC will be decrement by a value 1. In the case of the faulty receiver, the TEC of the healthy transmitter and the REC of the faulty receiver will be incremented by 8. The REC and TEC counters' values will keep incrementing until an error passive state is reached. The faulty receiver node will then transmit the passive error frame instead of the active error frame. The passive error frame consists of 14 recessive bits, i.e. 6 recessive bits (passive error flag) + 8 recessive bits (error delimiter). The transmission of passive error frame could no longer influence the network bus (and all the healthy nodes) as the passive error frame can be overwritten by other nodes. The healthy transmitter will now receive valid acknowledgement from the rest of the healthy receivers and decrease its TEC counter, while the faulty receiver will keep incrementing its REC and go into the bus-off state.

3.5.3 Error Counters and Fault Confinement

Figure 3-8 shows the 8-bit Transmit Error Counter and its carry bit on the left. The counter is set to zero after the reset of a CAN controller. When an error is encountered, the TEC will increment its count by a value of 8. When the TEC.7 bit is equal to 1, the CAN controller will switch its fault confinement state into error passive. The counter will decrement its count by 1 when a valid message frame is received. If the value of TEC keeps increasing, the CAN controller will go into the bus-off state when the carry bit of the counter is set. The settings of any bit from TEC.0 to TEC.6 will force the CAN chip to operate in the error active state.

	Msb			8-bit Transmit Error Counter				lsb
Carry	TEC.7	TEC.6	TEC.5	TEC.4	TEC.3	TEC.2	TEC.1	TEC.0

Figure 3-8 The transmit error counter overflow mechanism

3.5.4 DeviceNet Bus-Off Counter

The bus-off counter of DeviceNet protocol specification counts the number of times the device's CAN chip went into the bus-off state. All CAN controllers have the ability to flag the embedded software with a bus-off interrupt when the internal CAN Transmit Error Counter (TEC) or Receive Error Counter (REC) overflowed at 255. The TEC and REC are build into every CAN controllers. They must not be confused with the DeviceNet Bus-off Counter. The 8-bit counter (DeviceNet) will not reset itself, or rollover, when the maximum count of 255 is reached. The counter can be initialised to 0 when the following occurrs:-

1. power-up
2. device initialisation
3. when a Set_Attribute_Single request specifying Bus-off counter attribute is received

3.5.5 DeviceNet Bus-Off Interrupt (BOI)

The BOI attribute in DeviceNet protocol defines how a device behaves when a CAN bus-off state is encountered. BOI can take two values, i.e. '0' and '1' as illustrated in Table 3-4.

Table 3-4 The DeviceNet BOI value and its corresponding meaning.

BOI Value	Required Action
'0'	The embedded software must hold the CAN chip in the bus-off (reset) state and ensure that the device enters the communication fault state.
'1'	The bus-off CAN chip is allowed to be reset and continue communication, after the predefined initialisation steps of the Network Access State machine are performed.

The BOI value '1' can be treated as a soft reset of the device. All the previously established connections can either be deleted or kept in the configured state depending on the preference of product developers. In either instance, the duplicate MACID detection process must be performed as defined in the Network Access State machine of DeviceNet.

3.6 DeviceNet Abstract Object Model

Even though object-oriented methodology is not a global solution for a problem, it is suitable for describing DeviceNet protocol implementation. The use of object oriented technique in decomposing a device's internal logic makes software design and development easier. The object-oriented approach encourages the reuse of the proven and tested codes.

Each DeviceNet device can be modelled using a collection of objects. These objects define the device's internal components interaction (e.g. memory, communication, logic etc.), the communication services available (e.g. I/O polled or bit-strobed messaging) and the device's behaviour in response to external stimuli. Generally, each DeviceNet device can be modelled using two abstract objects (Figure 3-9), i.e.

- the Application Object, and
- the Communication Object.

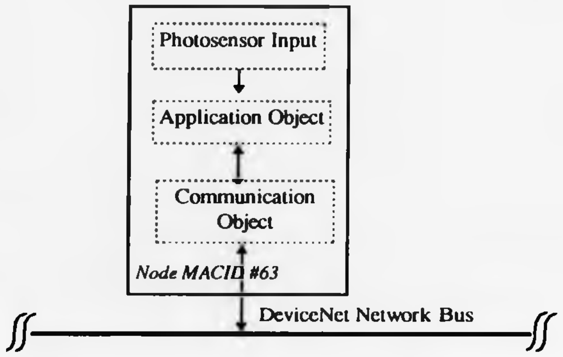


Figure 3-9 A DeviceNet Abstract Model

The application object is responsible for the device's dedicated function and must not be confused with the Application Layer of the OSI model and DeviceNet Specification. From the earlier example, the application object of a photo-sensor consists of the embedded software monitoring the logic state (ON/OFF) of the photo-transistor. On the other hand, the communication object will manage all the tasks related to DeviceNet communication, which consists of many more objects such as Connection Objects, Message Router Object etc.

Generally, the communication object includes all the services provided by the OSI model. It manages and provides the necessary services to realise the run-time data exchange among networked devices. For example, the logical link between the Application Object and Communication Object allows the

ON/OFF parameter of the photo-transistor to be visible and consumed by other networked devices. Deep in its engine room, the Communication Object will include the Physical Layer, Data Link Layer and the Application Layer of the 7 layer OSI model. The Communication Object is decomposed into many other objects, namely the Identity Object, Message Router Object, DeviceNet Object, Assembly Object, Connection Object, etc., in another level of complexity. This information hiding technique is one of the many features of object oriented design where complexities can be tailored to the level of concern.

3.7 Protocol Implementation

The DeviceNet protocol is implemented in the embedded software. Unlike Lonworks [64], Profibus and WorldFip fieldbus implementations, there is no dedicated protocol controller associated with the DeviceNet protocol implementation. In the above mentioned fieldbus implementations, special protocol chips which realise the higher layer of the OSI model such as the message formats, the network services and variables are hard-coded in the silicon. In DeviceNet implementation, all the protocol messaging scheme and data format are implemented in the embedded software of the device.

3.7.1 Software Implemented Protocol

Software implementation of the DeviceNet protocol benefits the developers by using off-the-shelf CAN controllers. The difference between a DeviceNet implementation and non-DeviceNet implementation lies only in the embedded controlling software. In theory, all CAN implementations can be transformed to be DeviceNet implementations by modifying the embedded software on the implementations.

The use of common chips to realise DeviceNet protocol offers various benefits. Firstly, it will be cheaper to use common parts which are readily available from more than 6 semiconductor manufacturers.

Secondly, the embedded software approach allows the developer the flexibility to support other CAN based fieldbusses such as CANOpen and SDS. This can be easily achieved by using different sets of embedded software for the corresponding protocol implementation. If conditions permit, all the higher layer protocols may even be implemented on the same device, with an automatic protocol selection facility to adapt this "multi-protocol" device into its corresponding environment. The idea of a "multi-protocol" device may prove a nightmare for the protocol conformance testers. The approach of developing various sets of protocol software for CANOpen, DeviceNet and SDS will be a more feasible solution today.

Thirdly, protocol implementation using software allows easy upgrades should the need arises. There are no expensive masks to be designed and manufactured. As most devices are equipped with flash memory, protocol version upgrades can easily be done by technical support staff by reprogramming the EEPROM with latest updates. This allows the older implementations to enjoy the same features as the latest developed device.

3.7.2 Dedicated Protocol Controller

On the contrary, on-chip solution adopted by Profibus, WorldFip and Lonworks open standards offers the device developers a quick interface for the corresponding protocols. The use of these protocol chips or dedicated protocol controllers simplifies the device design as all the PDUs and the 7 layer OSI services have been hardwired in the protocol controllers. This approach allows consistent implementation to be achieved quickly, but protocol upgrades may be a costly exercise as new masks are needed for the silicon.

Despite the easy standardisation of PDU format of this dedicated protocol controller approach, conformance testing is still needed to verify the correct operations defined by the protocol state machine. For example, when a device encounters a network error, the device must go into a "safe mode" and warn other devices on the network via a standard error message defined by the protocol. Whilst the standard error message format will be defined by the dedicated protocol chip, the triggering of this message is governed by the appropriate embedded software. In short, conformance testing of fieldbus devices using dedicated protocol controllers is still needed.

In space limited implementation such as a proximity switch, single chip implementation is preferred as there may be no room for another protocol chip to realise the necessary communication. It is also not economical to use this approach on low end, low cost devices. A simple low cost device requires simple network implementation, preferably a single chip microcontroller solution. This is why WorldFip and Profibus can also be used to interlink bigger and more complex equipment such as CNC machines, PLCs and drives. Single chip solutions may also help in qualifying design for the Certificate of Europe (CE)'s stringent Electromagnetic Compatibility (EMC) requirements.

3.8 Conformance to DeviceNet

From the previous section, the many issues involved in developing a DeviceNet device, which focus on the conformance of DeviceNet implementation were highlighted. Without conformance, interoperability of devices will not be possible. Without interoperability, the ideal of open standard fieldbus will not be achieved. Generally, the conformance to an open standard fieldbus can be divided into:-

1. hardware conformance
2. software conformance

3.9 DeviceNet Hardware Conformance Test

The Conformance Test Engine (CTE) framework in this project concerns the DeviceNet protocol messaging and its messaging schemes. The corresponding test equipment needed to carry out the hardware conformance test is not addressed. The hardware conformance test involves the use of high precision measuring equipment such as amp/volt meters, oscilloscopes and dummy loads (to emulate network nodes) to verify that the designed physical layer conforms to the DeviceNet specification.

3.9.1 Bit Timing Parameters

The utilisation of the approved physical layer driver chip no doubt will fulfil the electrical characteristics requirements. However, it is still not 100% certain that DeviceNet communication will take place successfully, especially at maximum bus length. The bit period of the CAN protocol can be programmed to accommodate different design criteria, e.g. maximum bus length and maximum oscillator tolerances. The utilisation of different bit timing parameters will cause communication failure even though the same physical layer design is used.

An easier solution is to ask the product developers what bit timing parameter values they have implemented on their devices. Nevertheless, the hardware conformance test must be set up to ensure that the product developers have correctly implemented the CAN bit timing parameters (i.e. TSEG1 and TSEG2 values) for the corresponding DeviceNet baud rates.

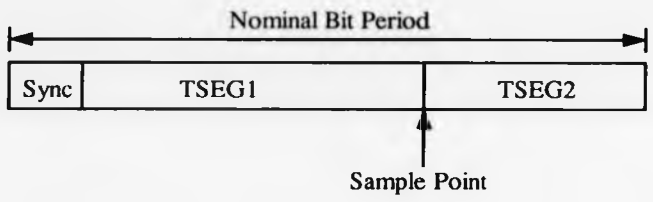


Figure 3-10 The CAN nominal bit period

Figure 3-10 shows a nominal bit time of a CAN frame that consist of the synchronisation segment (Sync), Time Segment 1 (TSEG1) and Time Segment 2 (TSEG2). Both TSEG1 and TSEG2 values are programmable via the Bit Timing Register 0 (BTR0) and Bit Timing Register 1 (BTR1) registers of the CAN controller. The Sync segment always consists of one system clock period, t_{SYS}., whereas TSEG1 and TSEG2 can have various t_{SYS}. In addition, TSEG1 must always be greater than TSEG2. The minimum value that the TSEG2 can have is 2 t_{SYS}, i.e. *information processing time*. This is the time needed for the CAN hardware to determine the sampled bit level (i.e. either "1" or "0"). The TSEG1 and TSEG2 values will determine the position of the sample point within the bit period.

Propagation Delay versus Oscillator Tolerance Optimisation

Greater TSEG1 value in combination with short TSEG2 (i.e. late sampling) will allow longer bus length to be utilised as the electrical signal has more time to propagate further, i.e. maximum propagation delay[2]. This configuration requires an accurate crystal oscillator to minimise the oscillator tolerances. Conversely, the sampling point must be set nearer to the middle of the bit period in order to compensate for the oscillator tolerance. As a consequence, shorter bus length is required. Propagation delay optimisation is used in DeviceNet. Table 3-5 shows a summary of BTR0 and BTR1 example values used in DeviceNet. BTR0 controls the baud rate prescaler so that the appropriate t_{SYS} can be obtained by scaling down the crystal oscillator's speed (usually 16Mhz or 20Mhz crystal) of the CAN controller. The Synchronisation Jump Width (SJW) that is used for lengthening and shortening the bit period (up to 3 t_{SYS}) is not used in DeviceNet.

Table 3-5 The BTR0 and BTR1 values used in DeviceNet

BTR0	SJW=0
	Prescaler =3 (÷ 4) @ 125 Kbit/s =1 (÷ 2) @ 250 Kbit/s =0 (÷ 1) @ 500 Kbit/s
BTR1	1 sample per bit
	TSEG1 (16 Mhz) : set to provide 13 t_{SYS} TSEG1 (20 Mhz) : set to provide 16 t_{SYS}
	TSEG2 (16 Mhz) : set to provide 2 t_{SYS} TSEG2 (20 Mhz) : set to provide 3 t_{SYS}

[2] The propagation delay involved in CAN protocol is the time needed by the electrical signal to go from one CAN controller to the other, and the time needed for its return.

3.9.2 Opto-isolation delay

Opto-isolation is required for devices that use high voltage supply such as the three-phase motor drive. The high voltage system must be isolated from the DeviceNet network with a 500 Volt isolation. The opto-isolator used in a DeviceNet implementation must fall within the 40ns maximum delay limit. The maximum combined delays, (i.e. both opto-isolator and the transceiver delays) are 120ns (80ns from transceiver delay) for the transmitter and 130ns (90ns from transceiver delay) for the receiver. Failure to comply with these requirements will introduce incompatible bit-timing which will cause a communication problem.

3.9.3 Physical Hardware

Physical hardware conformance starts from the basic and physical area which include the use of standard wire/cable, with corresponding colour coding and standard connectors. For instance, the CAN High signal is coded with WHITE wire, while the BLUE coloured wire indicates the CAN LOW signal. The wiring of the connectors and sockets must follow those stipulated in the DeviceNet specification. The area of interest may extend to the correctly implemented LEDs, e.g. RED for warning and GREEN for healthy operation.

3.9.4 CAN Controllers

Since the DeviceNet protocol is implemented using the embedded software in the microcontrollers, not all CAN silicon can be used to implement the DeviceNet fieldbus. The non-DeviceNet compliant CAN silicon can be categorised as below:-

a) the CAN chip which lacks the EPROM space needed to implement the DeviceNet Communication Objects.

b) the non-programmable, protocol only device, e.g. Serial Linked Input/Output (SLIO).

The architecture of CAN chips can be classified into two different categories, i.e.:-

- embedded (integrated CAN controllers), and
- peripheral (stand-alone CAN controllers)

with various message acceptance filtering capabilities. Table 3-6 shows the compatibility of currently available CAN controllers with DeviceNet implementation.

Embedded CAN controllers offer a tight coupling between the CAN communication section and the microcontrollers. These controllers feature register addressable CAN parameters and DMA transfer between CAN and the microcontroller core. Standalone CAN controllers are ideal for upgrading the existing embedded system for CAN communication. This can be analogous to the addition of an Ethernet communication card into an expansion slot of the Personal Computer for network communication in office LAN environment.

CAN controllers that allow some specific messages to be filtered (in addition to the mask) are referred to as **Full CAN controllers**, while CAN controllers that provide a mask to filter a range of identifiers are referred to as **Basic CAN controllers**. Both full and basic CAN controllers differ only in the message filtering capability of the silicon, and both are compliant with the CAN protocol specification.

Table 3-6 A summary of available CAN chips and their compatibility to DeviceNet

Manufacturer	Product	Type	Acceptance Filtering (Full/Basic CAN)	DeviceNet Compliant
Intel	8x196CA	Embedded	Full	✓
	82526/7	Peripheral	Full	✓
Motorola	68HC705 x4/16/32	Embedded	Basic	✓
National	COP 684/884 BC	Embedded	Basic	×
Semiconductor	MM 57C360/2	SLIO	Basic	×
NEC	μPD72005	Peripheral	Full	✓
Siemens	81C90/91	Peripheral	Full	✓
	81C515C	Embedded	Basic	✓
	81C806	Embedded	Full	✓
	81C815	Embedded	Basic	✓
	C167C	Embedded	Full	✓
Philips	82C150	SLIO	Basic	×
	82C200	Peripheral	Basic	✓
	8xC592/8	Embedded	Basic	✓

The choice of which CAN controller to use for DeviceNet implementation depends on the DeviceNet developers, provided that the one chosen fulfils the fundamental requirements of the DeviceNet Specification. In the system using basic CAN controller, the microcontroller will be interrupted more frequently as there is only an 8-bit mask to filter out the unwanted messages. If interrupt rate is critical in an embedded system design, then full CAN controller will be the better option as the microcontroller is interrupted only when the required message is received. This keeps the microcontroller from the tasks associated with network communication to a minimum.

Since the DeviceNet communication protocol relies heavily on the correctly implemented embedded software, it is important to thoroughly test the software for correct operation.

3.9.5 Physical Layer

All DeviceNet compliant devices must be able to work in a network of up to 64 nodes with maximum bus length. DeviceNet developers are free to design their own bus drivers on condition that the custom designed physical layer drivers must have equal or better specifications than that of the Philips 82C250. The Philips 82C250 bus driver complies with the ISO 11898 high speed physical layer signalling and comes in an 8 pin Small Outline Integrated Circuit (SOIC) package. Philips is improving the 82C250 to drive up to 110 nodes instead of the current 64 nodes [65]. Appendix A of the DeviceNet Volume I specification has defined the required electrical parameters for physical layer conformance.

The physical layer conformance test ensures that the electrical characteristics and parameters of the implementation fall within the limits defined in the DeviceNet specification. This area of concerns normally has been sorted out by developers in the early stage of the engineering design process, as redesigning the hardware can be a costly exercise.

Since the majority of product developers are using the same physical layer (i.e. the Philips 82C250 bus driver) chips in their products, the DeviceNet physical layer conformance test is not considered as critical as the protocol messaging test. The quality assurance of Philips semiconductor is relied upon to ensure that the electrical characteristics of the chips manufactured are all within the tolerance limits.

3.10 DeviceNet Software Conformance Test

Since DeviceNet is a software intensive implementation, considerable amount of effort should be placed in this area. The embedded software of the implementation will be tested only for protocol related function. Computer software which governs the internal operation of the device remains the responsibility of the product vendor. For example, the testing of the software which controls the ramp-up and ramp-down speed of a drive is beyond the scope of the conformance test. This leads to the proposed design of a conformance test framework for conformance testing the DeviceNet protocol software.

CHAPTER 4

THE CONFORMANCE TESTING CONCEPTS AND METHODOLOGIES

In order to achieve full interoperability of devices such as sensors and actuators in a fieldbus system, an exercise to ensure the correct implementation of the protocol is needed. This exercise is referred to as Conformance Testing. Conformance Testing is defined as a set of tests performed to check whether an Implementation Under Test (IUT) conforms to its formal specification [66]. It assures that the conforming products are implemented according to the formal specification, and they will interoperate and deliver the specified services.

Testing a new implementation may be time consuming and costly, but the consequences of not testing may be even dearer. For example, the destruction of the multi-million pounds Ariane-5 rocket and its invaluable cargo due to software failure emphasised the importance of software testing [67]. The software was adopted from European Space Agency (ESA) Ariane 5's predecessor and was thought to be proven and reliable until the catastrophic explosion of Ariane-5, 39 seconds after its launch on 4 June 1996. If the piece of software had been properly tested, the accident probably would not have taken place. However, totally eliminating software related accidents is not possible as today's software products are so complex and sophisticated, and they are constantly influenced by time and cost constraints. In addition, software engineering is a human activity and will always be prone to human error. Nevertheless, reasonable care must be taken when implementing safety critical software. This includes the use of methodologies and proof checkers (e.g. mathematical models and formal methods) to minimise the likelihood of human error.

4.1 The Black Box Testing Approach

Conformance Testing is classified as a black box approach if the external tester can only observe the outputs generated by the IUT upon the receptions of the appropriate inputs. This approach is used by the ISO 9646 for conformance testing. In an open standard, there are boundaries which govern the areas where a developer must conform to a standard set of protocols, and those areas which are proprietary to the developer. The information which falls in the proprietary areas will be classified and remain the responsibility of the product developers. Conformance testing should never be seen as a replacement of the quality assurance procedures utilised by vendors, as it governs only the externally visible public interface of an open standard. The internal operations of the implementation will not be of interest to the conformance testing. This black-box metaphor has also been used extensively in all object oriented systems.

4.2 Formal Tool for Protocol Representation

Formal tools in the form of language or notation must be used in order to describe the protocol accurately and precisely. This will help the comprehension of the protocol specification and avoid multiple interpretations of the standard protocol. A specification written in English language will be categorised as an *informal specification*, and will only be become a *formal specification* when the formal language or notation is used.

4.2.1 The Formal Description Techniques

The Formal Description Techniques (FDTs) are used to minimise the language ambiguities apparent in the protocol specification. Examples of available FDTs are SDL [68], ESTELLE [69], LOTOS [70] and TTCN (The Tree and Tabular Combined Notation). In the ISO 9646 definition, TTCN has been selected as the formal description language for the standard.

TTCN was designed to exactly express all attributes of an abstract test suite as specified in ISO 9646-2 specification. The TREE section of the notation is used in the dynamic behaviour descriptions to describe the occurrences of events according to the specification's state machine. The TABULAR section is used to simplify the representation of all static elements such as the PDU formats and verdicts associated with a particular test event.

In DeviceNet, the Abstract Syntax Notation One (ASN.1), i.e. another instance of formal notation, was used to formally specify information format and data types of the communication entities. ASN.1 (as defined in ISO 8824) is similar to the TABULAR section of the TTCN, which is useful in specifying the Application Layer of the OSI protocol. Neufeld [71] gives a good overview of the ASN.1 technique. Further information on the application of ASN.1 on DeviceNet protocol specification can be obtained from Appendix J of DeviceNet Specification Vol. I, Release 1.1.

As object-oriented technique is more widely used in the software industry, the object-oriented model has been used by Cena et al [72] to model the FIP. In another separate occasion, Juanole and Gallon [73] have proposed the Stochastic Timed Petri-Nets (STPN) model as the formal modelling of fieldbus protocol. Both the object-oriented model and STPN model have one thing in common, i.e. they consider the time factor. Hence, they can be used to model the fieldbus protocol as closely as possible.

Nevertheless, proven FDTs such as LOTOS, SDL and ESTELLE are still used for defining the lower layers of the protocol specification, whilst waiting for the new object-oriented model to mature.

Today, there are many commercial software tool-kits which support a wide range of FDTs. Some of these formal description software tool-kits are capable of compiling the formal language notation into run-time codes for test execution. This feature will help to automate the conformance test generation process, as well as automating the testing process of the implementation.

4.2.2 The Finite State Machine

The Formal Description Techniques (FDTs) discussed so far were based on the assumption that the communication protocol can be modelled using the finite state machine. In other words, the formal conformance test generation methodologies have been primarily based on the ***finite state machine*** (FSM) model [74] of the protocol specification.

"*A finite state machine, or finite automaton, is an abstract model describing the synchronous sequential machine and its spatial counter-part, the iterative network.*" ...Zvi Kohavi

A FSM is an automaton, with a finite number of states, that changes from one state to another when subjected to external stimuli. A *state* is the stable condition where the automaton pauses or rests. The automaton will generate an *observable output* (which may be *null* depending on conditions) when an external stimulus called *input* is applied. We denote a state transition of the FSM from initial state S_i to final state S_j, which is caused by an $input_k$ and generates an $output_l$ by the following relationship.

$$(S_i , S_j ; input_k / output_l)$$

In an IUT, there are three steps to verify whether the FSM is implemented correctly.

Step 1 Bring the IUT into state S_i (Test Preamble)

Step 2 Apply the $input_k$ and observe that the IUT generates the $output_l$

Step 3 Verify that the final state is S_j

A FSM is called *fully specified* if for every *permissible state* there exists a *permissible output* for every *permissible input set, I*. A FSM is called *partially specified* if some of the inputs are not allowed in some states. Most real life communication protocols are classified as *partially specified*. The FSM will be *minimal* if the protocol specification has only one *unique state* for a *permissible input set, I*.

A detailed discussion on the finite state modelling on OSI communication protocol can be obtained from reference [75]. The FSM used to model the Network Access state transitions of DeviceNet communication protocol is shown in Figure 4-1.

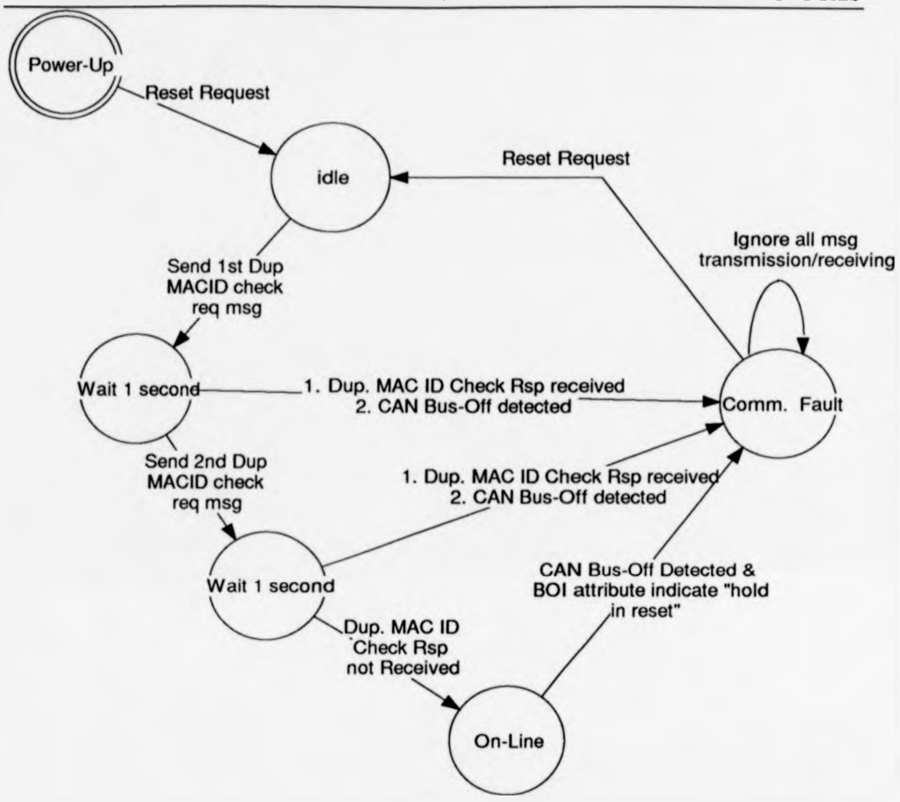

Power-Up
Reset Request
idle
Reset Request
Ignore all msg transmission/receiving
Send 1st Dup MACID check req msg
Wait 1 second
1. Dup. MAC ID Check Rsp received
2. CAN Bus-Off detected
Comm. Fault
Send 2nd Dup MACID check req msg
1. Dup. MAC ID Check Rsp received
2. CAN Bus-Off detected
Wait 1 second
CAN Bus-Off Detected & BOI attribute indicate "hold in reset"
Dup. MAC ID Check Rsp not Received
On-Line

Figure 4-1 The DeviceNet Network Access State Transitions

4.2.3 Formal Protocol Modelling

Formal Protocol Modelling involves the use of formal languages and the FSM to model the protocol specification. The formal protocol modelling exercise is to produce a formal protocol specification, and to detect any possible errors on the protocol itself.

Conformance exercise only conforms the IUT to the formal protocol specification. It does not check the correct behaviour of the protocol itself. For example, if the protocol defines that an implementation must produce an output HAPPY in state X when stimulated with input SUCCESS, all the IUT must produce the same output HAPPY when stimulated with SUCCESS in order to comply with the protocol; even though this is a faulty state transition on the protocol. In other words, the conformance exercise only ensures the correct implementation of a device to the protocol specification. It does not check the correct operation of the protocol itself.

4.3 Requirements of Conformance Testing

Reference [76] provides a good definition of conformance related issues in OSI protocols. There are three levels of conformance requirements in any OSI protocols, i.e., the *mandatory* requirements, the *conditional* requirements and the *optional* requirements. Mandatory requirements are those requirements that must be fulfilled whenever an implementation wishes to conform to a standard protocol, and must be observed at all times. The conditional requirements allow observations to be done only when the conditions defined in the specification apply. For instance, the implementation of a Presence Sensing Object in DeviceNet is a conditional requirement of the protocol which only governs presence sensing devices. On the other hand, the implementation of Identity Object is a mandatory requirement as this is needed in every DeviceNet implementation. Lastly, the optional requirements deal with those situations where an implementation is an optional requirement of the protocol. For instance, the Parameter Object of DeviceNet which provides the user with the configuration related data is an optional requirement. However, if the optional requirement is implemented, the device will be required to undergo the corresponding conformance testing process.

Each level of conformance requirement can be divided into two categories. They are:-

1. Static Conformance, and
2. Dynamic Conformance

Static Conformance are the fundamental requirements to allow an implementation to interconnect in a network. In the context of DeviceNet, it may govern the use of the right physical media to the use of correctly implemented message frames and PDUs. The dynamic conformance governs the predefined behaviour of an implementation and involves the protocol FSM. For example, before a DeviceNet device is allowed to go on-line, it must ensure that it has performed the duplicate MACID detection process to guarantee the uniqueness of its MACID. In short, dynamic conformance specifies the observable behaviours permissible by the protocol specification.

4.4 Types of Conformance Testing

The objective of conformance testing is to check whether an Implementation Under Test (IUT) conforms to its protocol specification. However, due to the practical limitations and economic factors, it is impossible to adopt an exhaustive approach for conformance testing. Therefore, five types of conformance testing (section 4.4.1~4.4.5) have been identified to indicate the level of conformance of an implementation. The basic interconnection test, capability test and behaviour test, with the exception of the conformance resolution test, must be standardised for a communication protocol.

4.4.1 Basic Interconnection Testing

The basic interconnection tests provide limited testing to ensure that there is sufficient conformance for the interconnection to take place. Any implementation which does not succeed in this test will have severe non-conformity problems and all subsequent tests will be terminated. The test acts as a preliminary filter before proceeding on to the more comprehensive and costly tests. In the context of DeviceNet, this may be a test for the Open Explicit Messaging Connection. If the implementation fails this test, then there is no point in proceeding as all subsequent tests have to depend on the Explicit Messaging Connections. Other examples of this test include the verification of :-

- the physical wire and connector of the DUT (Device Under Test) for correct colour coding and orientation,
- the testing of the appropriate voltage signals on the bus driver, and
- the correct flashing of the LED indicators.

4.4.2 Capability Testing

This test determines the presence of the implemented features declared in the Protocol Implementation Conformance Statement (PICS). (*The term Statement of Compliance (SOC) is used within the context of DeviceNet.*) It checks whether the appropriate features claimed in the PICS are consistently implemented in the IUT. The capability testing governs the area of the specification where the product developers have options for different implementations. For example, the MACID and baud rate settings of a DeviceNet device can be configured either using DIP switches or via the DeviceNet Object instance attributes. If the product developer decided to implement the DeviceNet Object, the developer must indicate this in the SOC document[1]. The DUT will then be tested for the proper implementation of the DeviceNet Object. The flexibility in implementing various device features as stated in the SOC for compliance allows the product developers to design innovative products for their market niche.

4.4.3 Behaviour Testing

Behaviour testing is used to determine the extent of the dynamic conformance requirements. As the number of possible combinations of states and events are infinite in a protocol, exhaustive testing must be avoided. Therefore, this test utilises the many conformance test generation methodologies, which are to be discussed in section 4.5. As an example, all DeviceNet devices must perform the duplicate MACID detection routine before going on-line to guarantee the uniqueness of their MACID. Therefore, all DeviceNet implementations must undergo the behaviour testing process.

4.4.4 Conformance Resolution Testing

This test is used to verify the uncertainties found in the previous three tests, so that a definite pass/fail result can be obtained. Even though previous tests failed to deliver the definite pass/fail answer, they help to narrow down the area where further tests are needed. Then, a conformance resolution test is performed to eliminate the uncertainty. As the situation of uncertainty will be different for each IUT, this test will not be standardised, i.e. no standardised abstract test case, and will be carried out on an ad hoc basis.

[1] The SOC document in DeviceNet takes the form of an electronic data file.

4.4.5 Interoperability Test

Interoperability test is used to determine whether two implementations will inter-operate. In the fieldbus protocol, interoperability refers to the ability of network devices to work together in a system. It can be further decomposed into three levels of interoperability, i.e.:-

i. basic interfacing,
ii. substitution with limited capability, and
iii. substitution with full integration [77].

The basic interfacing interoperability will guarantee the network communication for any compliant device. The latter two categories concern the function and data of the devices. Some fieldbus devices contain more parameters than others. For instance, the flow transducer from vendor A also gives the ambient temperature reading but this temperature parameter may be missing from devices available from vendor B. In fact this area almost crosses the fine boundary which separates the functionality of a device and its communication protocol conformance. While interchanging a category (iii) device with another same category device allows the full functions to be interchanged without restriction, interchanging devices from different categories can result in the unavailability of certain control data. In summary, interoperability concerns the communication protocol, while interchangability relates to the functionality and data of the device.

4.5 The Conformance Test Generation Methodologies

Conformance test generation methodologies are used to design and define the necessary conformance test suites for a protocol. The four most popular techniques used in generating conformance tests are as follows.

4.5.1 Transition Tour

The transition tour method was first suggested by Naito and Tsunoyama [78] for FSM-based sequential circuits. It is the most straight forward approach for conformance test generation. The state transitions defined by the FSM of the protocol specification are exercised at least once by applying the appropriate input sequence and observing the corresponding output. Sarikaya and Bochmann first applied this method in protocol conformance testing in 1982 [79]. The major disadvantage of this method is the omission of the state verification step which severely limits the fault detection capability of the technique.

4.5.2 Distinguishing Sequence (DS) Method

The DS-Method is a two phase approach. The test on the first phase checks that each state defined by the specification exists in the implementation. In the second phase of the test, the DS-method checks for correct output and transitions in the remaining transitions defined by the specification. This two phase approach is also used by Voung's UIOv method [80], the SW method [81] (Single Transition Checking using W set) and the partial W-method (Wp) [82].

A distinguishing sequence of an FSM is an *input sequence* which generates an *output sequence* that *uniquely identifies* the *state*, S_I of an implementation *prior* to the application of the *input sequence* (Figure 4-2).

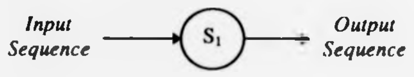


Figure 4-2 The DS-Method Testing

Since the DS-Method was originally developed for sequential circuit testing [75, 83] where all the states are fully specified, it suffers some drawbacks when applied to communication protocols that are typically partially specified. It will not be possible to generate every *input sequence* for every state due to the partially specified protocol. In addition, the number of tests to be generated for a FSM of *n* states will be n^{n+1}, which limits the use of this method.

4.5.3 W-Method

This method is also known as the Characterising Sequences Method and was introduced for FSM specifications that do not posses a distinguishing sequence. It is very similar to the DS-method where a characterising *set* of *input sequences* are applied to the *state* S_I and the *output sequences* are observed. Each characterising set of *state* S_I distinguishes *state* S_I from a group of states, hence uniquely identifying the *state* S_I after applying all the characterising sequences. This results in very long test suites for most modern protocols as compared with those developed using other methodologies.

In short, the Characterising Sequences Method is used as an alternative if the protocol specification does not have a distinguishing sequence. Another disadvantage of this method is that most real life protocols do not exhibit the distinguishing sequences due to partially specified specification. So, it will be difficult to find the distinguishing sequences in this situation.

4.5.4 Unique Input/Output (UIO) Method

The UIO method uses a unique sequence as an input for state, S_I such that its output sequence uniquely identifies the state, S_I. The UIO sequence is designed with the *knowledge* of what the *new state* will be. If an implementation under test does not return the expected output sequence, the test will be declared failed. The tester will not have any knowledge on the new state of the implementation when the failure occurred. This distinguishes the UIO method from the DS-method and W-method where the test designer knows the new state of the implementation, whether it returns the expected result or not.

This method, unlike the DS-method and W-method, does not require a fully specified protocol specification to realise it. Thus it results in a shorter test sequence, as only specific state information is needed. The limitation of this method is that it cannot tell which state the implementation is visiting when it fails the test. The UIO method was first introduced by Sabnani and Dahbura in a published paper [84]. A ***rural Chinese postman tour*** algorithm was later used by Aho et. al. [85] to optimise the UIO method so that more efficient conformance test sequences can be generated. The rural Chinese postman tour is an optimisation problem whereby all states in the state machine must be visited and traversed the minimum number of times.

4.5.5 Fault Coverage of test generation techniques

After reviewing the four most popular conformance test generation methods, the next step will be to find out which is the most efficient technique for DeviceNet. Sidhu and Leong did a Monte Carlo simulation [86] to estimate the fault coverage of these techniques [87]. They devised a method of generating random faulty specifications and ranked them into 10 different classes of faults. For example, a Class 1 fault is formed by modifying at random the output of a state in the specification state machine. A Class 2 fault will be the same as Class 1 except the modification is made on the tail state that follows the previous modified output state, and so on.

Both Sidhu and Leong also defined the two levels of conformance, i.e. strong conformance and weak conformance. Strong conformance requires the IUT to generate the same outputs for all the input sequences as defined in the specification. Conversely, weak conformance requires the IUT to have the same behaviour as that specified in the core edges of the protocol state machine. The DUT is allowed to have unspecified behaviour for the input sequences on those non-core edges. The edges in a given protocol state machine are referred to as core edges. The non-core edges consist of unspecified state input sequences, and is assumed that the protocol entity will produce a null output or ignore those input sequences.

Sidhu et al [87] then concluded that the three methods, i.e. DS, W and UIO methods are excellent for detecting faults in strong conformance testing. In addition, Dahbura and Sabnani also reported similar findings on UIO method in another published paper [89]. The Transition Tour method, however, could not detect all of the single faults presented for strong conformance testing as concluded by Sidhu et al. On the weak conformance testing, Transition Tour method again could not detect all of the single faults injected.

In summary, the four methods presented will detect any output error of the implementation, provided the implementation follows the FSM specification. However, transfer errors, i.e. errors in the next state reached by a transition, will not always be found. Therefore, the W-Method and Distinguishing Sequence method will be used to find the transfer errors provided that the number of states of the implementation remains within a certain boundary. The Transition Tour method mainly concentrates on the controllability issues while the rest of the methods address the observability issues. Controllability aims at bringing the IUT into the desired state where a test is to be conducted, i.e. test preamble. Observability aims at the identification of the IUT's current state after a state transition has taken place.

4.6 The Compliance Testing of OSI protocols

OSI protocols no doubt offer many advantages and benefits to the end users. Verifying the implementation for protocol conformance is a great challenge for both the compliance test designers and test engineers. Since the first meeting on Conformance Testing Methodology and Framework by ISO/TC97/SC16 working committee in October 1983 [90], there has been considerable research which focused on this area. The most significant contribution has been the definition of ISO/IEC 9646 multi-parts Conformance Testing Methodology and Framework in 1992, and with its Part 7 definition (ISO/IEC 9646-7) just being completed in 1995 [91]. ISO/IEC 9646 governs the conformance testing of any communication system that can be modelled using the Open Systems Interconnection (ISO 7498), which includes the open standard fieldbus.

The ISO 10303 multi-parts standards, in particular the ISO 10303-21 and ISO/DIS 10303-32, set the conformance test standards for industrial automation systems [92, 93]. As the 10303 multi-part standards concern the correct data representation and data exchange of the automation systems (e.g. PLCs), they are not used in this thesis. There is a fine line that separates the fieldbus as a communication systems, and a control and automation system. In fact a fieldbus system is an industrial automation system with built-in communication facility. The conformance testing of open standard fieldbus placed greater emphasis on the communication rather than the automation and control aspect of the system. In the context of DeviceNet, Volume I of the DeviceNet specification governs the communication section of the protocol, while Volume II concerns the data representation and exchange (e.g. device profiles and data byte scaling).

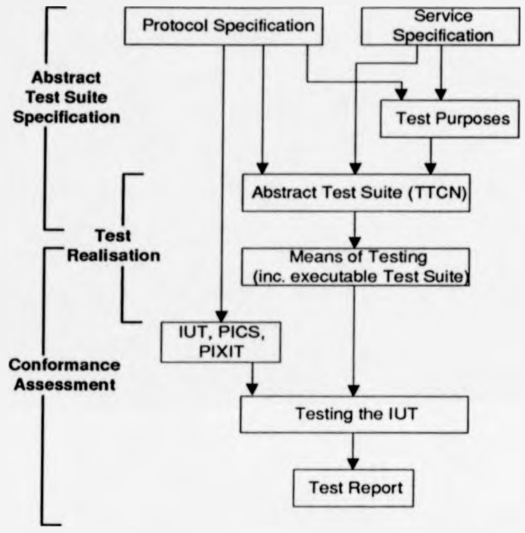
Hitherto, we have yet to have an international fieldbus standard, let alone the international standard for open standard fieldbus conformance. The international fieldbus standardisation process is progressing slowly, despite the effort by ISA/IEC and other standard organisations. Nevertheless, the work this thesis presents will be valid for conformance testing of any open standard fieldbus.

The open standard fieldbus, e.g. DeviceNet utilises 3 layers (Physical, Data-Link and Application Layer) of the OSI model, in comparison to a fully implemented 7 layer stacks of X.25 communication protocol. This well-known reduced protocol stack communication model (i.e. 3 layer model) enhances the real-time network performance by reducing the overheads of the communication software and network functions (i.e. either simplified or eliminated network functions). Since an open standard fieldbus can be modelled on the OSI model, much of the work defined in ISO/IEC 9646 can be adapted in this area of research. This statement is made based on the following factors:-

- Open Standard Fieldbus and the OSI Communication Protocol utilise the same Physical, Data Link and Application Layer of the 7 layer OSI Model (ISO 7498).
- The methodology defined by ISO/IEC 9646 for conformance focuses on single-layer testing, starting with the lowest layer and incrementing one layer at a time through the seven layers. This can be utilised for testing the open standard fieldbus implementations, i.e. the reduced protocol stack (3 layer model) communication model.

Therefore, it can be deduced that the conformance testing methodologies and techniques for testing the OSI protocols can be adapted for testing the open standard fieldbus protocols which uses the reduced stack OSI model. Figure 4-3 shows the standard conformance testing framework as perceived by ISO 9646 standard. It is also valid to say that all open fieldbus standards are designed using the ISO 7 layer model to achieve interoperability and openness of the fieldbus standards. Every conformance testing methodology available to date is a derivation and enhancement of this standard framework. Figure 4-3 shows the three stages in a conformance testing process, i.e. the abstract test suite specification, the test realisation and the conformance assessment processes. They will be individually discussed in the following sections. ISO 9646 uses the TTCN (The Tree and Tabular Combined

Notation) as the formal description techniques (FDTs) for the abstract test suite specification.



Legend

IUT	:- Implementation Under Test
PICS	:- Protocol Implementation Conformance Statement
PIXIT	:- Protocol Implementation Extra Information for Testing
TTCN	:- Tree and Tabular Combined Notation

Figure 4-3 The Standard Conformance Testing Methodology and Framework (ISO 9646)

4.7 Abstract Test Suite Specification

The complex conformance test architecture is impossible to comprehend without breaking it into many smaller modules. A conformance test suite can be decomposed into various *Test Groups* for different test objectives as depicted in Figure 4-4. At the lowest of the hierarchy are the *Test Events* which consist of the indivisible units such as the PDU transfers. The ordering and groupings of *Test Events* will form the *Test Case*. Each *Test Case* consists of a number of steps (i.e. Test Events) needed to carry out a specific conformance test on the IUT. A number of related Test Cases will collectively create a Test Group to fulfil certain test objective.

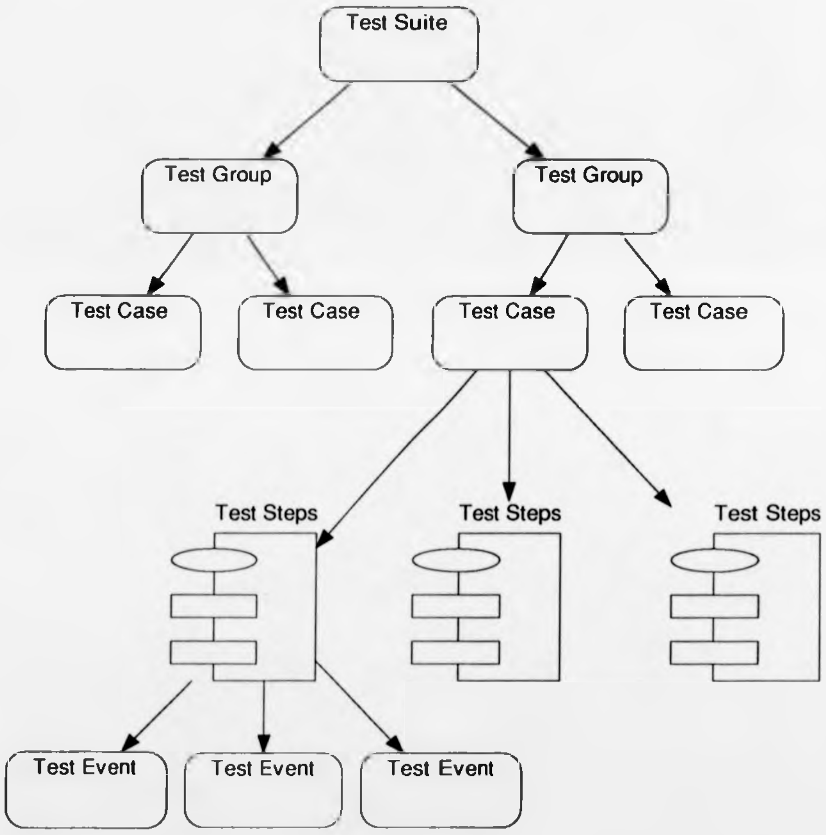


Figure 4-4 The Conformance Test Suite Structure [91]

The DeviceNet conformance suite can be divided into the following test groups:-

a) Network Access Test,

b) Transport Layer Test,

c) DeviceNet Messaging Test,

d) Device Profile Verification Test,

e) DeviceNet Master Test,

f) Group 2 Server Poll/Bit Strobe Test,

g) UCMM Connection Test, and

h) Object Class Test Groups [94].

Within a Test Group, there are many Test Cases. In the example of the Transport Layer Test Group, the Test Cases will include:-

a) Explicit Messaging Fragmented Production

b) Explicit Messaging Fragmented Consumption

c) I/O Connection Fragmented Production, and

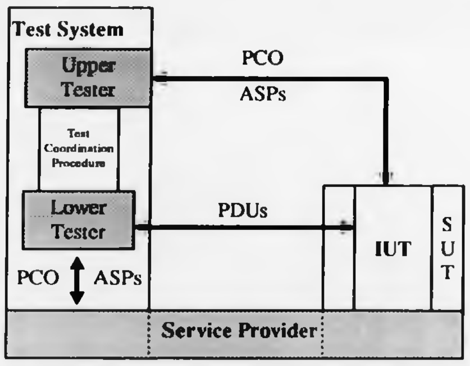
d) I/O Connection Fragmented Consumption Test Cases.

Each of the listed Test Cases will have its own test steps to carry out the testing. For instance, in order to carry out the Explicit Messaging Fragmented Production Test Case, the following steps will be needed:-

- Open an Explicit Messaging connection to the DUT and set the EPR to 0
- Send a Get_Attribute_Single request service to the Explicit Messaging Connection's Produced_Connection_Size attribute.
- Success Response expected with produced connection size
- If produced connection size ≤ 8 bytes, then proceed to Explicit Messaging Fragmented Consumption Test Case
- Send a Get_Attribute_Single request to the attribute with fragmentation
- Send a first fragment acknowledge with a Fragment_Count of 3 (which supposed to be 0), and expect no response from the DUT
- and so on

4.8 Conformance Test Realisation

The conformance test realisation is where the actual conformance testing process is physically carried out. A local single layer test method (ISO 9646) is shown in Figure 4-5. The test system (i.e. CTE in this instance) will initiate the necessary command and control to verify the IUT for conformance. In the reduced stack fieldbus protocol, the Service Provider is simply replaced with a media cable, as there is no public switching networks involved. An instance of the test method, i.e. CTE for DeviceNet will be discussed in-depth in Chapter 5.



Keys:-

ASP	Abstract Service Primitives
IUT	Implementation Under Test
PCO	Point of Control and Observation
SUT	System Under Test

Figure 4-5 The Local Single Layer Test Method

The setup (Figure 4-5) presupposes that only the Test system and the IUT are involved in the conformance testing process. Both the test system and the IUT are configured to have a one-to-one relationship.

4.9 Conformance Assessment Process

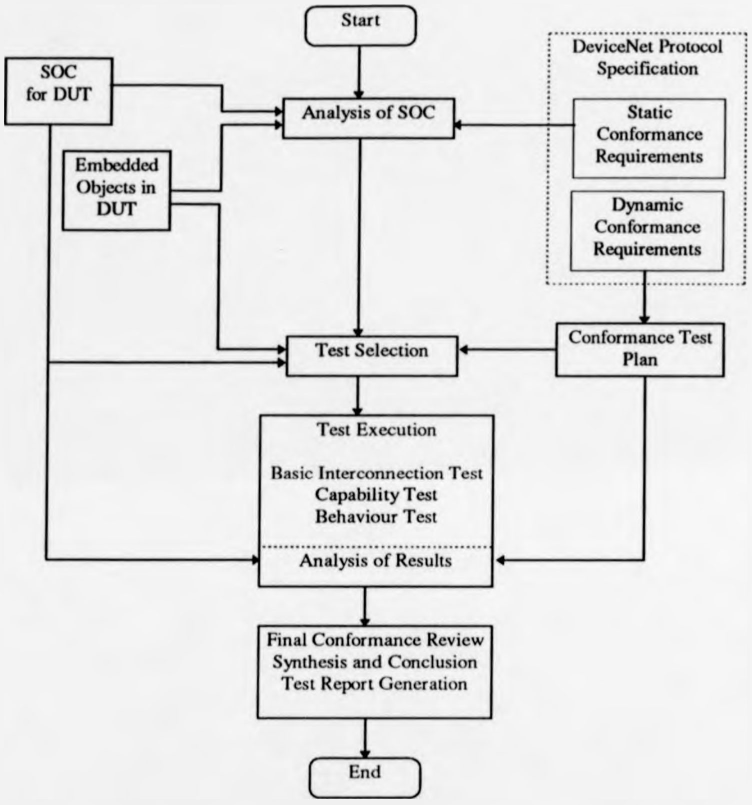


Figure 4-6 DeviceNet Conformance Assessment Procedure Outline (modelled from ISO 9646 and CCITT)

The conformance assessment procedure will act as a guideline on how to carry out the verification activities during test realisation: in other words, how to verify the IUT for conformance, having got the conformance test suites and conformance test setup. Figure 4-6 is a modified version of the conformance assessment procedure outlined in the ISO 9646 standard. It can be broadly divided into four stages:-

1. analysis of SOC
2. test suite selection
3. test execution
4. analysis and review

By examining the statement of conformance (SOC) document, the test engineer can then select the relevant tests for the IUT from the test suite database. The selected test suites are compiled. Then the compiled test suites are executed to verify the IUT against the SOC and the protocol specification. Lastly, the test results are collected and analysed. The analysis of test results can be done in real-time during test execution, or in the form of a report after the end of the test. Typically, both real-time analysis and post test execution analysis are used in real-life implementations. For instance, if an IUT does not respond to a request message, a real-time analysis can be done to report the failure on the IUT and terminate the conformance test.

4.10 Protocol Frames Definition

In the DeviceNet conformance testing scenario, protocol frames can be divided into three categories for easy identification:-

4.10.1 Valid frames

These are syntactically correct frames which are received at the correct or opportune time (i.e. the protocol entity expects to receive such a frame in its current state.).

4.10.2 Inopportune frames

These are syntactically correct frames but arrive at the wrong time (i.e. the protocol entity does not expect to receive such a frame in its current state). For instance, the correct syntax fragmented message with the wrong fragment count can be classified as an inopportune frame.

4.10.3 Invalid/Illegal frames

These are the frames which have incorrect syntax. CAN remote frame is a good example of an invalid/illegal frame as it is not allowed in the DeviceNet protocol.

The conformance test suite for detecting inopportune frames is the most difficult to design. This is because inopportune frames are perfectly healthy PDUs but appear at the wrong time or in the wrong sequence, depending on the state of the IUT. The consumption of Invalid/Illegal frame by the IUT at any point during the test will void the conformance to DeviceNet. Appendix C lists the Invalid DeviceNet Message Conformance Test Specification.

4.11 The Conformance Test Architecture for DeviceNet

The conformance testing processes must be designed, developed and certified prior to testing any implementation for conformance. There is a need to ensure that an executable test case meets its protocol specification before it is used to examine another system. The certification and verification process can go on forever. A conformance test suite needs to be tested before it can be used to test the implementation for conformance. At another level, the methodology used must be verified and tested before it can be used to develop the conformance test suite.

Generally, the standards are there for the masses to utilise. Since there will be only a few conformance test labs available for DeviceNet, the verification process of the conformance test lab may be less formal and simple. The verification process for the test lab may involve running the conformance test suite and obtaining an identical, repeatable test result. The production of repeatable and identical test results for the same conformance test suite at different test labs will ensure uniform conformance test lab equipment standards. However, this does not verify that the conformance test suite is free from any error.

The conformance testing and verification process at each stage is governed by the law of diminishing returns. There is no way to verify and test all the processes involved before they are utilised. Every level in the design assumes the correct implementation on which the design is based. For instance, tests to ensure that the personal computer used is 100% compatible with the PC architecture are regarded as unnecessary in the design of a DeviceNet conformance test system which uses a PC. In general, the conformance testing of a fieldbus protocol can be divided into the following vicious cycle as shown in Figure 4-7.

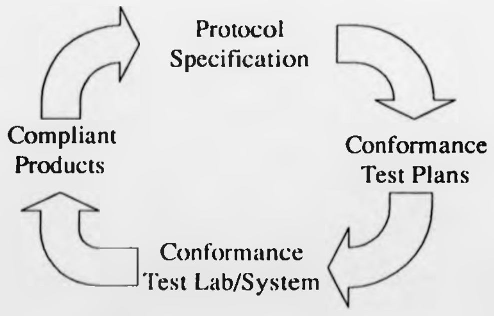


Figure 4-7 The Conformance Test Life Cycle

First, the protocol specification is defined. Then the formal method is used to derive the formal protocol specification from the defined specification. The formal protocol specification acts as the consistency and proof checker to reduce possible error and eliminate the ambiguity of natural language. Having verified the protocol, the next step is to develop the conformance test suites and systems so that the conformance testing process can be carried out. Here, formal design methodology such as DS, W and UIO methods are used to design the conformance test suites. The use of these methodologies guarantees a systematic and efficient approach in the test suite design. The test suites are then transferred into executable conformance test systems. At this stage, the conformance test system will need to be verified and tested prior to testing any IUT for conformance.

In DeviceNet, the conformance test software and the independent test labs must be verified before they are allowed to perform the product certification process. Even though the product developers are allowed to carry out self-compliance, the final IUT needs to be sent for conformance testing to an approved conformance test lab. In Profibus implementations, all Profibus compliant products must undergo conformance tests conducted by the so-called Test Centres. These independent Test Centres use the same conformance test hardware and software to verify the Profibus implementations for correct protocol realisation. Implementations which violate the standard are prohibited from bearing the Profibus certificate.

The conformance test cycle (Figure 4-7) will loop back to the protocol specification at the end of the vicious cycle. The reason is that errors may be found during the life cycle of the fieldbus protocol and amendment must be made to the protocol. In another instance, it may due to the new protocol updates to accommodate new features, e.g. highly distributed peer-to-peer communication in future.

4.11.1 Conformance Special Interest Group in ODVA

In the DeviceNet implementation, ODVA has taken the initiatives in forming a Special Interest Group (SIG) for compliance testing. This special interest group plays an active role in defining the appropriate conformance test plans to conform an implementation to the DeviceNet specification. ODVA is also following the footsteps of other open communication protocol to carry out the compliance testing process at independent test sites. For instance, the conformance certification program of Fieldbus Foundation has been developed by the Fraunhofer Institute of Germany, i.e. an independent test site.

4.11.2 Independent DeviceNet Test Lab

The use of independent conformance test sites will benefit the DeviceNet developers with the following:-

- ability to give impartial test results
- the use of standard plan, test setup and test cases ensure all DeviceNet implementations are equal and interoperable.
- single source to govern the protocol specification upgrades.

In addition to that, it is also recognised that the standard compliance test software is made available to DeviceNet developers so that they can pre-test their implementations during design and development stages. This may contribute to shorter product development lead time.

CHAPTER 5

THE DEVICENET CONFORMANCE TEST ENGINE FRAMEWORK

In order to successfully carry out the conformance testing process on a DeviceNet fieldbus implementation, an efficient and cost effective conformance test framework must be designed and developed. Figure 5.1 shows the proposed automated conformance test engine (CTE) framework set-up in an independent test laboratory. The testing processes are controlled by the host PC of the test system. All hardware components used are easily available and can be purchased off the shelf.

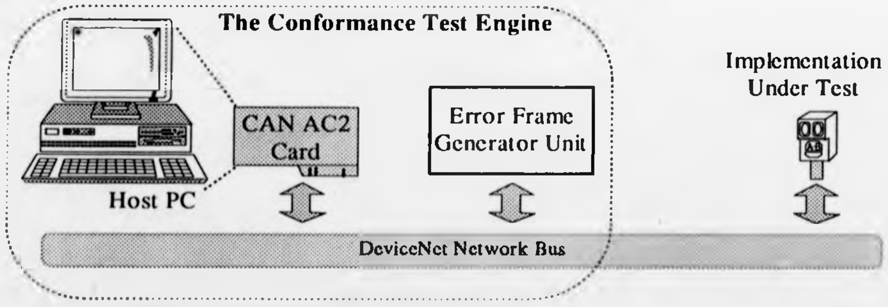

The Conformance Test Engine
Implementation Under Test
CAN AC2 Card
Error Frame Generator Unit
Host PC
DeviceNet Network Bus

Figure 5-1 The Conformance Test Engine Framework

5.1 Assumptions

The DeviceNet protocol specification is assumed to be a formal protocol and correct in its operations. No effort has been made to validate the protocol specification itself. The protocol validation, i.e. the process of checking the consistency and correct operations of the formal protocol specification is beyond the scope of the CTE. The conformance test framework only ensures the correct implementation of the Device Under Test (DUT[1]) to the specified protocol, i.e. protocol verification. Therefore any error involving the protocol specification will not be detected.

5.2 Automating the Conformance Testing process

Despite the use of conformance testing methodology, traditional testing, development and maintenance are tedious, time-consuming and error prone. The test results can be inconsistent due to variability in the expertise and practices of the test engineers. As the test systems get more complex and sophisticated, the number of tests in a test suite will grow. This growth may result in excessively long execution time for conformance and may not impress the product developers who send their DUT for conformance testing. Therefore, the process of conformance testing must be automated as much as technology allows. Automation is used in manufacturing industry to produce consistent quality products at competitive cost. Similarly, automated testing systems can be used to produce good quality and consistent test results [95, 96]. The conformance test engine (CTE) framework is designed to automate the conformance testing process.

[1] Device Under Test (DUT) of DeviceNet is similar in meaning to the term Implementation Under Test (IUT) defined in ISO 9646 standard

5.3 The criteria considered in CTE design

5.3.1 Economical factor

Even though cost is not a technical factor, often it is the most influential factor for a design. The conformance testing process must not be expensive. This means that conformance testing must be swiftly carried out and the conformance test rig and equipment should be based on commonly available platforms to save cost. The conformance testing process must be automated to reduce the time needed for testing.

5.3.2 Reliability and Repeatability

The CTE must not only be fast and accurate in carrying out the conformance testing process, it must also be able to provide accurate and reliable test results, i.e. the repeatability of all test results in all cases. In order to determine the robustness of the DUT, some tests may need to be run repetitively for days. The test results are then analysed using statistical methods to determine the consistency of the DUT. If an error is detected, the error is expected to replicate itself in all the test runs, i.e. the repeatability of test results. The CTE hardware must also be reliable enough to withstand many hours of non-stop operation. This is usually not a problem for most IBM PCs.

5.3.3 Upgrade-ability

Volume II of the DeviceNet specification governs all the device profiles. Device profile is there to ensure that interchangability of DeviceNet devices are possible with the definitions of parameters such as data scaling, data format and parameters resolution for a common device group. As the available devices increase in volume, more device profiles will be defined. This growing process requires a framework which can grow with the need. Therefore, the suggested solution will be to use a Windows database such as Access to store all the information. The utilisation of database allows easy upgrade and amendment when new device profiles are introduced. This database approach is not implemented in the current project.

5.3.4 Maintainability

Software maintenance is another key issue that need to be considered during system design. The CTE design encourage the use of Windows component software and object-oriented techniques. It is based on the vision that the CTE software requires minimal modification and update throughout its life cycle. Any protocol updates can be done by amending the test database.

5.4 CTE Hardware

In order to drive down the set-up cost, the DeviceNet conformance test framework must be designed and developed on a common platform which is proven and reliable. IBM PC architecture has been around since the early 1980s (IBM PC was launched in UK on 12th August 1981 with 16kb of RAM, mono monitor and optional cassette storage.) and has become a de-facto standard in the computer industry. These cost-effective and relatively powerful machines can be customised easily for the conformance testing of DeviceNet implementations. They have enough resources to offer for the purpose of conformance test application. In addition, there are many CAN PC interface cards (16-bit ISA bus format) available today.

The major components of the Conformance Test Engine (CTE) are :-

- a 486 based (or higher) IBM compatible computer
- a commercially available CAN card (Softing CAN-AC2), and
- a protocol emulator for error frame injection, i.e. the Error Frame Generator (EFG).

The CAN-AC2 card is chosen for this project due to its intelligent independent operating kernel architecture. The CAN-AC2 card itself is a single card computer which consists of a NEC V25 processor and two CAN controllers. It is plugged into the 16-bit ISA bus expansion slot of the PC.

The information exchange between the CAN card and the PC is executed via the dual-ported RAM on the CAN card and the upper memory block of the PC memory map (i.e. E0000-DFFFF Hex). This configuration allows fast data exchange between the two processors, i.e. NEC V25 and Intel 486 CPUs, and facilitates independent operations of both systems. In other words, both the CAN card and the PC can be viewed as two separate computer systems sharing data via the dual-ported RAM. The PC's processor will not interfere with the real-time embedded software of the CAN card and vice versa, thus increasing the real-time performance of the system.

5.5 CTE Software

The conformance testing software is the main module which resides on the personal computer and runs under DOS 6.22 and the Microsoft Windows 3.11 environment. A real-time kernel such as iRMX is not used, as the timing and task scheduling of the conformance testing process are not critical. The use of a dedicated operating system would make in-house testing more difficult to set up. The CTE should remain as simple as possible to enable product developers to carry out in-house testing themselves. It must utilise the commonly available resources such as DOS and Windows whenever possible. In addition, the resources must also be standardised so that the quality and consistency of the tests are not sacrificed.

The CTE software is developed using the Visual C++ object-oriented programming environment to take advantage of the Microsoft Foundation Class (MFC) library and Dynamic Link Libraries (DLLs). MFC is an object-oriented library for the Windows environment. For example, the mouse actions such as drag-and-drop and double click actions in Windows are all governed by the MFC library, and can be inherited from the foundation class.

One of the advantage of this is that when the need to upgrade to a later version of Windows arises, the source code can be compiled using the later versions of the MFC library. The utilisation of the MFC and DLL also allows forward compatibility of Windows programs to the latest version of Windows. For example, most Windows 3.11 programs can be run safely on Windows 95, with a few exceptions. The DLL is another feature of Windows programs where the program is dynamically linked during run time. This allows a smaller executable file size to be achieved. However, if a required library component is missing from the working directory, the program will stall with an execution error. For instance, if one accidentally deleted a *program.DLL* file, the program will not launch even though the *program.EXE* file is present. The purpose of DLL is to minimise the Windows executable code size.

5.5.1 Dual Purpose Conformance Test Software

The *conformance test software* can be categorised into two different modes of operations, i.e.-

- ***development test,*** and
- ***conformance test*** modes.

The development test mode will be used by product developers in-house to investigate their implementations during development. Normally, a reduced configuration CTE hardware is used by the product developers. Therefore, some functions on the conformance test software involving special hardware such as the EFG protocol emulator may be disabled. Any option not available to the developers will be automatically greyed out by the conformance test software, which is a standard feature of Windows software.

Once the product developers are satisfied with their implementations, the implementations will be sent to an independent test lab for conformance testing. The same piece of conformance test software is used to allow consistency in testing. All functions of the CTE will be enabled. The exercise of letting the product developers run the conformance test in-house aims to minimise the engineering iteration process involved in conforming an implementation to DeviceNet. This will result in a shorter development lead time and cost savings.

5.5.2 Modularity of CTE Design

The CTE framework has been designed for modularity using object-oriented techniques and component software. This is important as the development of the DeviceNet conformance test comes from the participating members in the ODVA conformance Special Interests Group (SIG). The conformance SIG has to liaise with other working groups such as the *drive profile* SIG to include the appropriate module for conformance testing. The modularity achieved through object-oriented design allows various modules to be added and combined together at different stages of CTE development. Furthermore, the design can be modified easily to utilise the component software in future. All the modules will then be compiled prior to run-time to achieve maximum efficiency and timing during conformance testing.

Component software can be analogous to Integrated Circuits (ICs) in the electronics world. With the use of the IC components, the design engineer can concentrate on the higher level of the design without getting into the lower level details. For example, the engineer can use the AND gate in an IC without having to design an AND gate from transistors and resistors. In software engineering context, an application can be divided into components such as Graphical User Interfaces (GUIs) for Man Machine Interface (MMI), databases for storing information and spreadsheets for calculations. Each component can be separately developed and combined together to form a complete package.

Conformance testing is a growing process, with endless introduction of new device profiles. Therefore, the database of conformance test suites needs to be updated frequently. If component software is used, one can use the Access database package to store all the conformance test suites and test steps for the CTE. If a test suite update is required, only the conformance test suites database needs to be updated. The rest of the framework remains intact.

In the instance of modularity, the EFG is an optional module of the CTE framework used at the independent test lab. The EFG unit is a CAN protocol emulator capable of emulating error conditions on a DeviceNet network. It is remotely controlled by the CTE software on the PC using the CErrFrGen object class. If the EFG hardware is available, the CErrFrGen class will notify the conformance test software and this feature is enabled. The CErrFrGen object class will then be available to the Windows environment, so that commands and controls to the EFG can be issued via the conformance test software. The EFG hardware will appear only as an encapsulated CErrFrGen object with its appropriate attributes. The conformance test software can be visualised as a collection of objects as shown in Figure 5.2. Again, the EFG object is another software component which contributes to the whole CTE software.

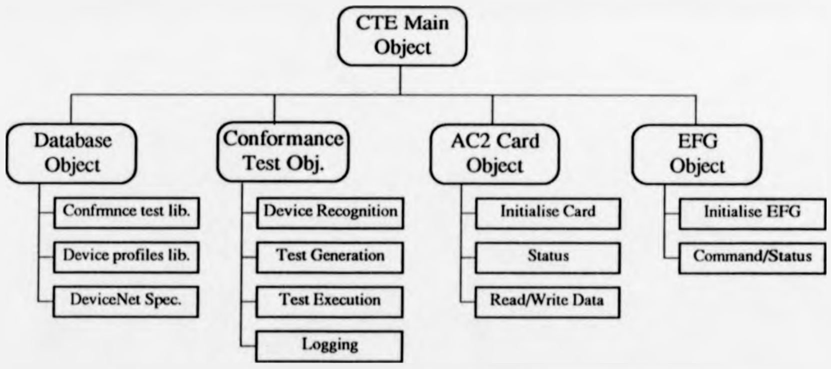


Figure 5-2 The object decomposition of CTE software

The design and development of EFG is unique in the area of DeviceNet research. With the help of an EFG unit, the DUT's behaviour during error conditions can now be investigated and tested. Chapter 6 provides detailed discussions on the EFG design.

5.6 Windows Based CTE Software Design

During the design phase of the project, considerable analysis has been done to decide whether the conformance testing framework should be developed under DOS or Windows platform. The study was finally completed with the decision to use Windows as the development platform based on the following reasons:-

5.6.1 Virtual Device Driver

Windows uses the Graphics Device Interface (GDI) which minimises the hardware dependency, i.e. SVGA drivers, printer drivers are totally encapsulated as an object. The conformance test software can make use of the already available software components and marshals the necessary controls and commands to achieve the conformance testing. For instance, the CAN-AC2 card is supplied with its own *ac2.dll* library. To utilise the functions provided by the CAN-AC2 card, the conformance test software calls the relevant object class. The *ac2.dll* file is included in the working directory.

5.6.2 Human Computer Interface (HCI)

Windows features a Graphics User Interface (GUI) which provides a more user-friendly human computer interface (HCI), thus improving usability. HCI is the study of the interaction between people, computers and tasks [97]. The consistent layout of Windows program and interface format allows easy interaction between the user and the computers. The learning curve involved in using another Windows software for a Windows user is very minimal. Windows' GUI interface has achieved many proven track records as the preferred user interface since its launch.

5.6.3 Flexibility and Expandability

Microsoft's Database Management System (DBMS) and Open Database Connectivity (ODBC) allow extensive database support and data-exchange between off-the shelf database packages such as Access to interface seamlessly into the CTE framework. The idea is to store the conformance tests in the database and to isolate the expertise. For example, the database section can be developed by computer scientists who are experts in database management, normalisation etc. When there is a change in the Specification or device profile, only the database section needs to be amended or updated. This can further be expanded into a knowledge-base in future.

5.6.4 Object Oriented Systems

The object-oriented nature of Windows environment encourages reuse of codes. For instance, the CTE software developer does not need to worry about the development of a relational database as it can be implemented using Microsoft Access.

5.6.5 De-facto Industry Standard

Windows has become the de-facto standard in the personal computer world. This popularity will contribute in the bug weeding process as there are more chances of discovering the software bugs, if any, due to the widespread usage. One may argue that DOS based programs will run faster than Windows. The advancement in the semiconductor has minimised Windows' shortcomings and given Windows programs performance similar to their DOS cousin. Windows has the advantage over DOS based program as a lot of the building blocks can be inherited from huge Windows library. Windows also comes in many different languages which is a very useful resource to have for today's global market.

5.6.6 Maintainability.

The Windows environment and programming is standard. There are no hidden extras that the software engineers need to know to maintain the codes. With object-oriented notation and a knowledge of Windows programming, any Windows programmer can maintain the codes.

5.7 Disadvantages of the Windows platform

5.7.1 Memory Use

Windows programs need larger physical memory (i.e. RAM) and a more powerful processor to run on. The average program size is also larger than an equivalent DOS based counterpart and this requires bigger secondary storage space. The availability of super-scalar Pentium Class processors and the plummeting cost of hardware have made Windows an attractive solution.

5.7.2 Timers

Another area of concern is that the hardware timer available on Windows based program may not be fine enough for controlling real-time application. The PC ROM BIOS initialises the Intel 8259 timer chip to produce a timer interrupt (08 Hex) every 54.925ms (i.e. "clock tick" resolution). Windows timer uses the timer logic built into the IBM PC's hardware and allows 32 timers to be active at any one time. The highest resolution of the interrupt or "clock tick" of Windows timer is 54.925ms. The 32 timers can be cascaded to achieve intervals shorter than 54.925ms.

The use of a PC CAN card with on-board embedded kernel software for independent running of the real-time application and the PC has made real-time control on PC possible. Besides, Personal Computers (PCs) have been used to run factory-floor control logic [98]. These PC architectures are called "soft" PLCs. These PCs can either use the Microsoft Windows designed for the desktops, or real-time operating kernels with Windows sitting on top of them. It has been shown that a well-designed Windows NT system will respond to a switch input from the field within a 20ms time window [99].

In another real-time application of controlling an X-Ray machine, a PC running iRMX for Windows was able to achieve a response time of less then 1 ms [100]. The CANalyser from Vector, which is based on the CAN-AC2 card and DOS operating system, is capable of achieving a 100µs time resolution when running on a 486-DX33 machine. The fastest time for the arrival of a Protocol Data Unit (PDU) will be 94µs (shortest PDU (47 bits) with zero length data field, exclusive stuff-bits) at 500kbit/s with 100% bus loading. This is more than adequate for DeviceNet conformance test application as the conformance testing does not utilise 100% of the transmission bandwidth (typically less than 10% bus loading).

5.7.3 Development Tool Wizard

A Windows based program has been the end-user's dream and the programmer's nightmare. There is no doubt that the Windows based software is very complex. However, help is at hand, as the wizard tools available in most program development environments are maturing. For example, the Visual C++ programming environment is capable of generating the necessary C++ program skeletons and comments when the corresponding questions within the Windows dialogue box are answered.

In short, most of the disadvantages of the Windows based approach for the DeviceNet conformance testing platform can be solved by using more powerful machines (486 or higher machines) and bigger memory and secondary storage, i.e. spend more money on the hardware platform. Until we can prove Moore's Law wrong, we will continue to have more powerful but cost effective processors. Hence, the processor and memory problems of Windows will soon be a problem of the past.

5.8 In-house testing with a Standard PC set-up

Conformance testing can take up as much as 30% of the total development budget of a software implementation [101]. In order to speed up the process of conformance testing and shorten the development lead time, product developers can use the same version of conformance test software during the course of product development. This will ensure that their implementations are compliant to the DeviceNet protocol specification prior to sending the implementation for conformance testing in the independent test lab. The conformance test software will not run if incompatible hardware is encountered, therefore consistency in conformance testing is achieved.

5.9 The Operation of CTE

Figure 5.3 shows that the statement of conformance (SOC) document in the form of an electronic data file is input into the CTE at the beginning of the test. This SOC data file was generated when the product developer filled in an electronic statement of conformance form. The DeviceNet SOC is a combination of the *protocol implementation conformance statement* (PICS) *and protocol implementation extra information for testing* (PIXIT) defined by ISO 9646.

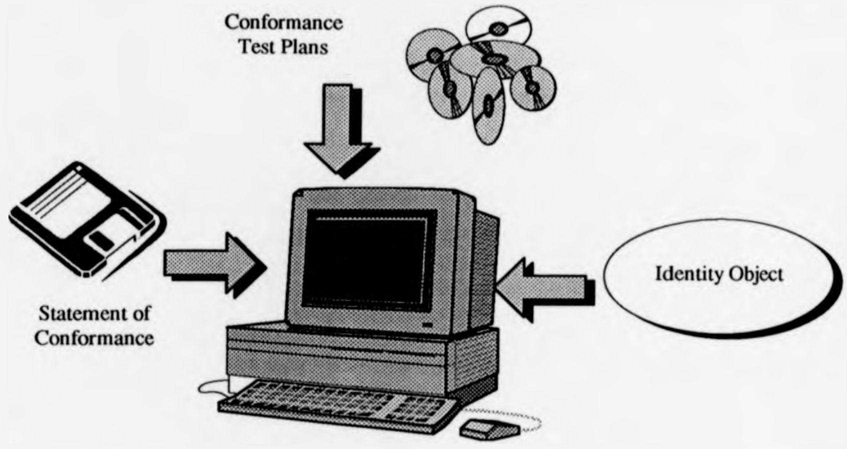


Figure 5-3 The CTE internal interactions

After obtaining information from the supplied SOC data file, the CTE then verifies the identity of the Device Under Test (DUT) by comparing the SOC information against the Identity Object of the implementation. This preliminary step will ensure that the SOC is referring to the correct DUT for subsequent testing. Once the identity of the implementation and its device profile is determined, the CTE will search its conformance test suite database for the relevant tests and compile them. The conformance testing will then begin.

It is proposed that the CTE software only contains the skeleton of the framework. The CTE software will only marshal the conformance testing operation. All the data and information obtained will be processed by separate modules. For example, the conformance test suites can be stored in carefully designed databases. Should the need to modify the conformance test suites arise, only the database needs data updates. The CTE software will remain the same.

Early investigations on DDE shows that data traffic on the CAN bus can be passed on to consuming client application. The client application in this instance can be just a pass/fail report analyser, or the more extensive knowledge based inference machine. Analysis tools such as LabView for Windows etc. have utilised the DDE hot links to their operations. It has been used extensively in the SCADA sector.

CHAPTER 6

THE ERROR FRAME GENERATOR DESIGN

The Error Frame Generation (EFG) module is designed to verify the error behaviour on DeviceNet implementation. Previously no conformance test suites were designed to test the correct error recovery sequence of a DeviceNet implementation. This was because there was no way of generating or injecting errors on the DeviceNet bus in a controlled manner. The EFG module developed in this project makes this possible.

The EFG Prototype Design

The Error Frame Generator (EFG) is a sub component of the CTE framework as depicted in Figure 6-1. The EFG is a protocol emulator capable of injecting CAN error flags onto the network bus to facilitate error testing on the DUT. The DeviceNet specification defines a sequence of states which a DeviceNet device must execute when error is detected on the network bus [62]. As such, all DeviceNet implementations must obey the error handling mechanisms defined in the specification.

All CAN controllers have the ability to indicate the detected errors to the Data Link Layer of the OSI stack, but they do not allow the user software to initiate transmission of the error flag. The error detection and signalling mechanisms are hard-coded in the MAC (Media Access Control) of the Data Link layer [42]. In order to verify the correct implementation of error management and fault confinement in DUT, a method of injecting errors into the DeviceNet signal bus is devised.

The objective of CAN Error Frame Generator is to generate the 'active error flag' on the DeviceNet bus. This simulates an error condition whereby the DUT must perform the stipulated error handling routine. The generation and injection of error frames using the EFG is the only way to observe and verify the DUT behaviour under error conditions. The EFG allows the DeviceNet conformance test suite to verify:-

- the correct implementation of error management in DUT
- the correct recovery procedure and sequence in DUT

Furthermore, the EFG can also be used to study the behaviour of a DeviceNet network when subjected to a random or periodic burst of errors.

6.2 The EFG Operation explained

The EFG is remotely controlled by the host PC using the following 2 distinct messages, i.e. EFG Request and Response Message as shown in Figure 6-1. Both messages use the Reserved DeviceNet messages, therefore they should not be consumed by the DUT. The consumption of these messages by DUT will void its compliance for DeviceNet. Both messages are used to 'ferry' the commands issued by the PC to the EFG, and the status from the EFG to the PC. The internal operation of the EFG can be controlled using the provided EFG Registers.

msb lsb

Data Byte	7	6	5	4	3	2	1	0
1	DUT							
2	ID Register							
3	Error Frame Register							
4	ID Count Register							
5	Status Register							
6	Configuration Register							

msb lsb

Data Byte	7		6	5	4	3	2	1	0
1	DUT								
2	ID Register								
3	Error Frame Register								
4	ID Count Register								
5	Status Register								

Figure 6-1 The EFG Request and Response Messages

6.2.1 EFG Request Message

The EFG Request Message (CAN Identifier 7C0 hex) consists of 6 data bytes. The 6 data bytes are further divided into 5 different registers, namely the DUT ID Register, Error Frame Register, ID Count Register, Status Register and Configuration Register. These registers will be individually illustrated in the following sections.

6.2.2 EFG Response Message

The EFG Response Message (CAN Identifier 7C1 hex) is a complement to the EFG Request Message. When the EFG unit receives the EFG Request Message from the host PC, it will echo back its internal states and the received data using the EFG Response Message. This allows the host PC to know the exact state and status of the EFG unit.

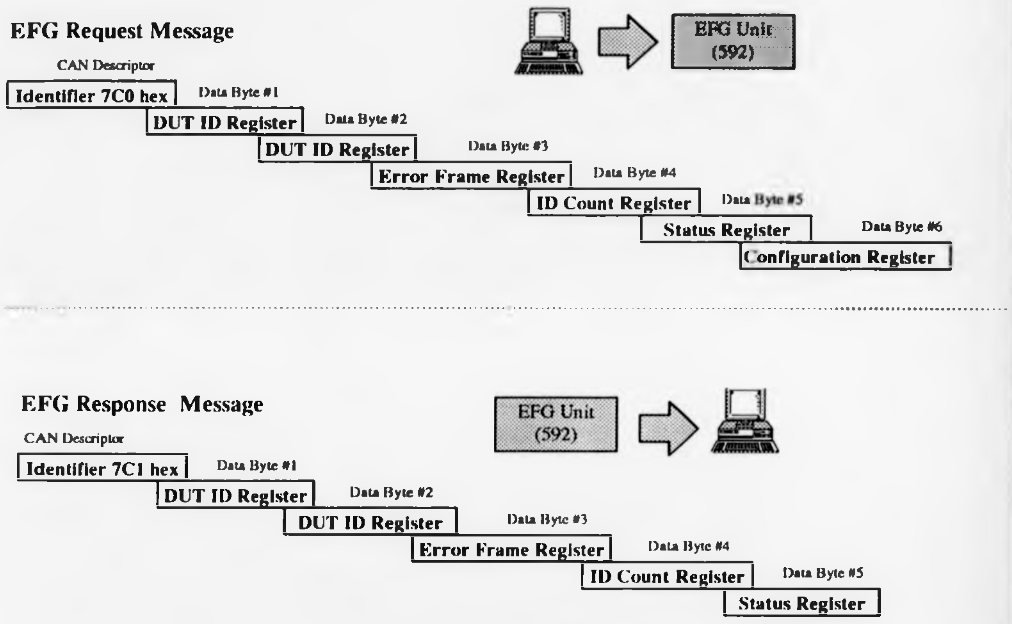


Figure 6-2 The operation of the EFG Request/Response Messages

Figure 6-2 illustrates how the two EFG Request/Response messages are used to transfer the command and status between the PC and the EFG module. CAN Identifier 7C0 hex, which is a reserved DeviceNet message, is used by the PC to send the appropriate command to the EFG unit. Only the EFG unit will consume the data packet as it is the reserved DeviceNet identifier. The hardware design and block diagram of the EFG module will be illustrated in Section 6.3.

6.2.3 DUT_ID Register

The DUT_ID register (i.e. the 1st and 2nd data byte of ERG Request/Response Message) contains the 11-bit CAN Identifier (ID10...ID0) and the data length (DLC3...DLC0) that is to be zapped with an Active Error Flag every time it appears on the CAN bus. The register (Table 6-1) is write-able during EFG Reset and it is Read-Only at all other times.

Table 6-1 The DUT_ID Register

Msb lsb

Data Byte	7	6	5	4	3	2	1	0
0	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3
1	ID2	ID1	ID0	R	DLC3	DLC2	DLC1	DLC0

6.2.4 Error Frame Register

The Error Frame Register (Table 6-2) is a read/write register which occupies the 3rd data byte of ERG Request/Response Message, i.e. data byte offset #2. During Reset, this register is write-able with values between 0 and 127. The EFG will cycle the generation of Active Error Flag depending on the corresponding bit enabled in the register. This register will echo back the number of Active Error Flags generated prior to the read status request command.

Table 6-2 The Error Frame Register

Data Byte	7	6	5	4	3	2	1	0
2	EF_R.7	EF_R.6	EF_R.5	EF_R.4	EF_R.3	EF_R.2	EF_R.1	EF_R.0

6.2.5 ID_Count

The ID_Count register (Table 6-3, i.e. the fourth data byte of the ERG Request/Response Message) displays how many instances of DUT ID has been monitored on the bus since the CTE became active.

Table 6-3 ID_Count Register

Data Byte	7	6	5	4	3	2	1	0
3	ID_C.7	ID_C.6	ID_C.5	ID_C.4	ID_C.3	ID_C.2	ID_C.1	ID_C.0

6.2.6 Status Register

The fifth data byte of the EFG Request/Response Message is the Status Register (Table 6-4) of the EFG unit. This register will indicate all the vital information on the EFG states and status.

Table 6-4 The Status Register and its functional description

Data Byte	7	6	5	4	3	2	1	0
4	EFG Error	Reserved	Reserved	Monitor DUT_ID	EF Counter	DUT_ID Counter	Ready	Reset

Reset (R[†]/W[††])	0-Normal mode operation. 1-Reset enabled.
Ready (R/W)	0- Disable EFG (Stop). 1- Enable EFG (Start).
DUT_ID Counter (R)	Displays the number of DUT_ID monitored on bus. 0- Disable the counter. The DUT_ID counter will reset to 00 hex. 1- Enable the counter. The counter will display how many. instances of the DUT_ID have been monitored on the bus.
EF Counter (R/W)	Displays the number of Active Error Flag generated (R). The number of Active Error Flag to be generated (W). 0- Disable the EF Counter. 1- Enable the EF Counter. The EFG unit will generate the Active Error Flag according to the EF Counter value.

	For example, if the EF Counter contains a value 5, only the first 5 instances of the DUT_ID will be zapped with an Active Error Flag. See also MODE in Configuration Register.
DUT_ID Status (R)	0- No. DUT_ID is loaded. 1- DUT_ID to be monitored is successfully loaded in the EFG. The DUT_ID to be monitored is echoed in the DUT_ID. field. This normally will cause the EFG Error bit set to '1'.
Reserved	
Reserved	
EFG Error (R)	0- EFG status OK. 1- EFG internal fault.

†R : = Read Only, ††W:=Writeable.

Note : The Error Frame Counter and Monitored DUT_ID Counter will default to zero whenever Reset mode is entered.

6.2.7 Configuration Register

The Configuration Register (Table 6-5) is the last data byte in the ERG Request Message. This register is only accessible when the Reset mode is enabled. The register is only applicable in the EFG Request Message.

Table 6-5 The Configuration Register and its functional description

Data byte	7	6	5	4	3	2	1	0
5	Reserved	Reserved	Reserved	Reserved	Reserved	EF Counter	DUT_ID Counter	Mode

Mode	0- EFG will generate an Active Error Flag whenever the DUT_ID is monitored on the bus. 1- EFG will generate the Active Error Flag according to the value loaded in the Error flags counter.
DUT_ID Counter	0- Disabled. The counter will not reset and remain at last count or maximum of 128. 1- Enabled The counter will roll-over to 0 when maximum count of 128 is reached.
EF Counter	0- Disabled. The counter will not reset and remain at last count or maximum of 128. 1- Enabled The counter will roll-over to 0 when maximum count of 128 is reached.

6.3 The EFG Hardware

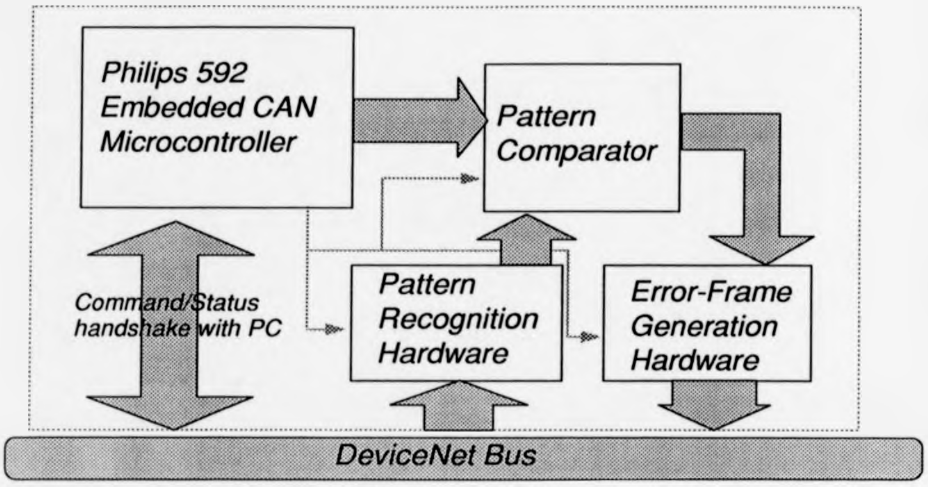

Philips 592 Embedded CAN Microcontroller
Pattern Comparator
Command/Status handshake with PC
Pattern Recognition Hardware
Error-Frame Generation Hardware
DeviceNet Bus

Figure 6-3 The block diagram of the Error Frame Generator Module Hardware

The hardware of the EFG module (Figure 6-3) consists of the following, i.e.:-

- Pattern Recognition Hardware,
- Pattern Comparator,
- Error Frame Generation (EFG) Hardware.

6.3.1 Pattern Recognition Hardware

The function of the pattern recognition hardware is to detect the start of frame (SOF) of the CAN frame, i.e. 11111110 binary. Once the start of frame is detected, the pattern recognition hardware will feed the CAN bit stream into the serial-to-parallel converter. This allows the serial bit stream to be converted into parallel word before feeding into the pattern comparator (74HCT688) input.

6.3.2 Pattern Comparator

The pattern comparator takes the role of "mask and match" filter in CAN protocol. It compares the CAN/DeviceNet identifier detected from the bus against the identifier stored in the register. The identifier stored in the register is downloaded from the 592 controller (via Port 4) before the detection and comparison sequence starts. It only compares the Identifier field of the CAN frame, i.e. 11-bit identifier. All subsequent bit streams following the Identifier field are ignored. If a match is found, the pattern comparator will send an enable signal to the error frame generator logic to transmit an active error flag.

A 16-bit word is used for the pattern comparator due to the following reasons:-

- CAN adopts the bit-stuffing coding technique with a stuff width of 5. The maximum stuff bits possible within the 11-bit identifier are two stuff bits.
- The pattern comparator only comes in an 8-bit package.

The 16-bit word typically consists of SOF (1-bit), 11-bit Identifier (11-bits), RTR (Remote Transmission Request - 1 bit), maximum of 3 stuff bits (3-bits) and 2 reserved bits.

Table 6-6 The bit word for pattern comparator

16-bit word to be compared by the pattern comparator																
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Min.	SOF	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0	RTR	R	R	DLC.3
Max.	SOF	ID10	ID9	ID8	ID7	S	ID6	ID5	ID4	ID3	ID2	S	ID1	ID0	RTR	R

Legend

DLC.3	:- The most significant 4-bit of the Data Length Code (DLC)
R	:- Reserved Bits (Standard CAN Frame's reserved bits)
RTR	:- Remote Transmission Request (default to "0")
S	:- Stuff Bits
SOF	:- Start of Frame

Table 6-6 shows the arrangement of the 16-bit word in the pattern comparator. The number of bits to be compared by the comparator varies between 13 bits (for the minimum packet header size) to 15 bits (for the maximum packet header size), illustrated by the shaded area on the table.

The original idea was to use a tri-state device to enable the appropriate number of bits to be compared. Since the bit-stream has been stuffed by the software in 592 controller, the number of bits to be compared can be controlled via an AND gate. A "1" at the 2 input AND gate will allow the data to flow through to the pattern comparator input. A "0" will cause the output of the 2 input AND gate to output a "0", thus a "0" is input to the pattern comparator's input.

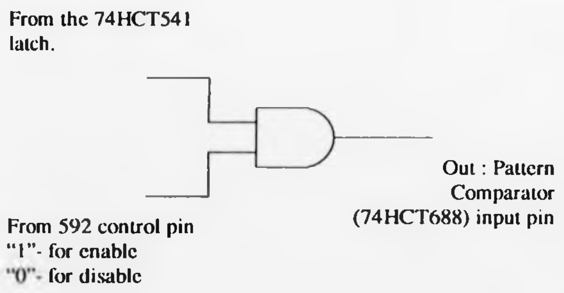


Figure 6-4 A simple data flow control using AND gate

By using this simple flow control technique, the software can determine how many bits are to be compared by the pattern comparator by switching the corresponding pin ON or OFF. If the software detects a maximum of 2 stuff bits, the pattern comparator can be configured to compare a total of 15 bits. If only 13-bits are needed for the comparator, i.e. the situation where no stuff bits are present, the software can be programmed to insert zeros for the non-concerned bits and disable the data flow to the pattern comparator by writing a "0" to the 2 input AND gate.

An improvement has been made to eliminate the AND gate and compare the first 16 bits of the CAN message bit-stream instead. The embedded software residing in the 592 controller is used to arrange the requested DUT identifier into the transmitted/received form of CAN bit-stream. This 16-bit CAN bit-stream is then downloaded into the pattern comparator. In the case of the shortest (unstuffed bit-stream) Protocol Data Unit PDU), the data length of the PDU must be known. The data length code (DLC, i.e. the most significant bit) is then included in the 16-bit CAN bit-stream for comparing.

6.3.3 Error Frame Generation Logic

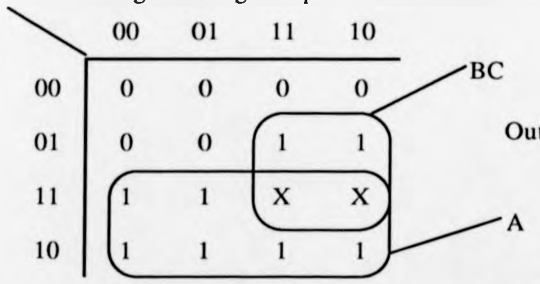
This module generates the active error flag on the DeviceNet bus. The generation of the active error flag simulates error conditions (local error) on the bus. This method of error injection only affects the DUT, thus allowing the DUT behaviour to be monitored by the host PC, i.e. another CAN node.

The active error flag is realised using a synchronous BCD counter. The counter is clocked using a global clock provided by the EXO-3 crystal oscillator. The clock frequency takes the same frequency as the CAN baud rate. For instance, if the CAN network is operating at 125kbit/s, the 125khz clock source is used to clock the counter. Table 6-7 shows the truth table of the counter with its corresponding output. The objective is to use the counter to generate 6 dominant and 8 recessive bits, i.e. an active error flag. The output function is then minimised using Karnaugh Map as shown in Figure 6-5.

Table 6-7 The Truth Table for Active Error Flag

Decimal	A	B	C	D	Output (F)
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	1
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	X
15	1	1	1	1	X

Using Karnaugh Map for minimisation



Output Function, Y = A + BC

Figure 6-5 The Karnaugh Map for Active Error Flag Minimisation

A digital logic is then designed to realise the Boolean expression, Y=A+BC. The output Y is fed into the transmit pin of the Philips 82C250 physical layer chip. This module effectively forms the error frame generator (EFG) logic.

6.4 EFG Hardware Design

The Error Frame Generator (EFG) prototype was wire-wrapped and implemented on a Euro-card. Wire-wrapping technique is used to allow more densely integrated circuit design and orientation. The following issues were highlighted during the course of EFG hardware design and development.

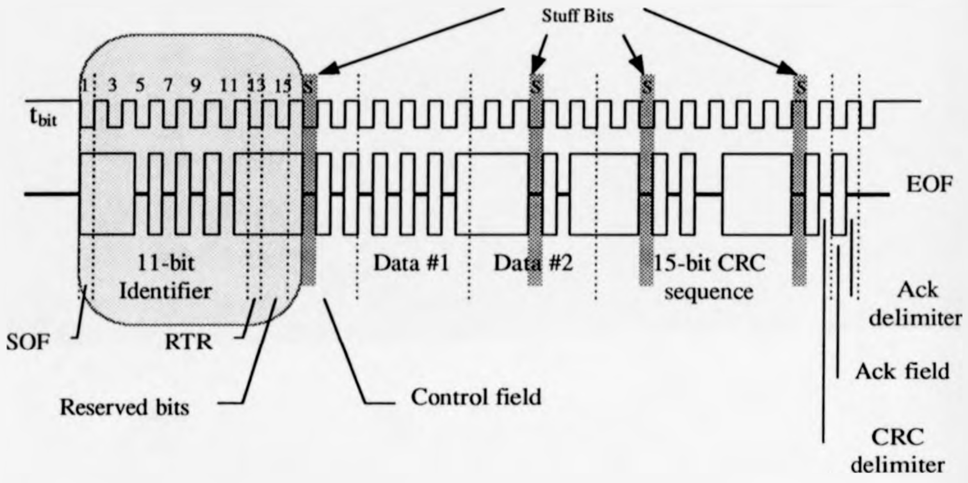
6.4.1 Clock Synchronisation

When the start of frame (SOF) of CAN frame is detected, all CAN controllers will synchronise their local clock at the same instant. Using the local crystal oscillator, each controller can then measure the exact bit period, t_{bit} for the correct sampling instant. Similarly, the EFG pattern recognition hardware must be able to synchronise itself to the SOF and maintain the correct bit period, t_{bit} until the end-of-frame. This precise timing for EFG unit is provided by the EXO-3 crystal oscillator with an 8-bit counter as the clock prescaler.

6.4.2 Hardware based CAN Message Screeners

From Section 6.3 of the EFG hardware design, it can be seen that a pattern recognition hardware is used to screen for the required CAN identifier, rather than the conventional way of using CAN controller. CAN controller is not used for detecting the CAN identifier in this instance due to the following reasons.

The CAN controller only flags an interrupt when a *complete* CAN frame has been received, i.e. an acknowledgement has been given by the receiving node. This could not be used for the purpose of error injection. The active error flag must be incident upon the transmitting CAN frame in order to forge the error condition. If the active error frame is not precisely incident on the CAN message frame (which is to be zapped into bus-off state), other nodes on the network will be affected by the EFG unit.



Legend

Ack	: acknowledgement	S	: stuff bits
CRC	: cyclic redundancy check	SOF	: start of frame
EOF	: end of frame	t_{bit}	: bit period, e.g. 8µs for 125kbit/s bus
RTR	: remote transmission request		

Figure 6-6 A CAN message frame with reference to the bit period clock, t_{bit}

Figure 6-6 shows a typical CAN message frame. The shaded area indicates the area of interests to the pattern recognition unit, i.e. the hardware based CAN message screener. The area consists of the arbitration field (the 11-bit identifier and 1 bit RTR), 2 reserved bits and the most significant bit of the control field of the CAN frame.

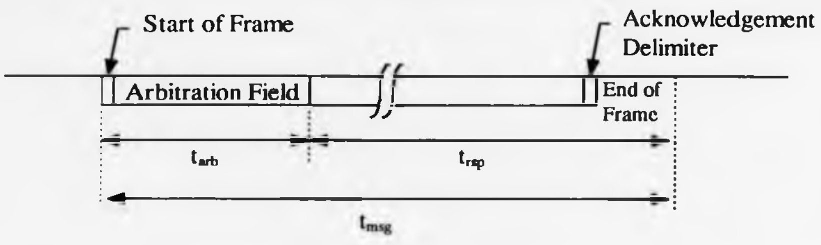


Figure 6-7 The definition of EFG response time

From Figure 6-7, the following relationships can be established, i.e.

$$t_{rsp} = t_{msg} - t_{arb}$$

where

t_{rsp}	:=	the time taken for EFG unit to recognise the required CAN identifier and transmit an active error frame
t_{msg}	:=	the CAN message frame validity period
t_{arb}	:=	the arbitration field validity period

For successful error injection, the error frame generation unit must fulfil the following criterion, i.e.:-

$$t_{rsp} < t_{msg} - t_{arb}$$

In other words, the transmission of the active error frame must be carried out before the temporal validity of the message frame has ended. The transmission of active error frame on the transmitting message forces the transmitter node to believe that it has caused an error. The transmitter will then initiate the error recovery sequences at once.

The shortest CAN frame consists of 47 bits (assuming zero length message with no stuff bits). First, the EFG unit must determine the CAN identifier by scanning the arbitration field. Once the correct CAN identifier is obtained, the EFG logic must be armed to transmit an active error frame, all within the 35 bit (i.e. 47 - 12 bit arbitration field) time. The designed EFG unit has successfully fulfilled these criteria.

6.4.3 EFG Response Time

The t_{bit} will be different for different DeviceNet baud rate. For example, if the DeviceNet network is operating at 125 kbit/s.

$$Period, t = \frac{1}{frequency, f}$$

$$t_{bit} = \frac{1}{125k}$$

$$= 8\ \mu s$$

Table 6-8 The relationships between baud rate and the bit period, t_{bit}

Baud Rate	125 kbit/s	250 kbit/s	500 kbit/s
Bit Period, t_{bit}	8 μs	4 μs	2 μs

At the fastest baud rate, i.e. 500 kbit/s, the turn-around response time, t_{rsp} of the EFG unit must be less than 2μs x 36 bits = 72 μs (the period from control field to the acknowledge field, assuming no stuff bits involved). If the EFG logic fails to deliver an active error flag within this response time, then the EFG unit will not be successful in injecting error on the target node.

The EFG unit designed in this project has successfully satisfied this crucial criteria. It has demonstrated its capability to zapp off the zero length CAN message frame successfully at 125kbit/s.

6.4.4 CTE Baud Rate Setting

In the EFG unit, the global clock is provided by the EXO-3 programmable crystal oscillator. The current design uses 3 DIP switches to manually set the EFG unit into one of the three different baud rates of DeviceNet. Even though it is possible to program the oscillator using the Philips 592 microcontroller, it is not adopted in the current design. The choice between manual and software controlled baud rate settings is only a matter of individual preference.

6.5 EFG Software Design

The software that controls the internal operations of the EFG unit has been designed and developed using the Hitop development environment. As can be seen from Figure 6-3, the heart of the EFG unit is the Philips 8xC592 microcontroller, i.e. a derivative of the Intel MCS-51 family. Therefore, Keil C compiler is used to compile the C source into the 8051 executables.

The functions of the EFG embedded software are:-

i). to receive and execute the commands from the host PC

ii).to setup and control the EFG logic for error generation at the appropriate moment.

iii).to insert the necessary stuff bits within the 16-bit word

Function (i) requires the handling of the CAN communication such as the screening of the EFG Request message and the transmission of EFG Response message. Function (ii) involves the setting and resetting of the appropriate port pins of the 592 controller for arming the EFG logic. In function (iii), the software will be responsible for inserting necessary stuff-bits into the DUT_ID register. This allows the 11-bit CAN identifier loaded in the DUT_ID register to have identical bit stream (inclusive stuffed bits) with those that would appear on the network bus. The EFG embedded software flowcharts are listed in Appendix B.

6.6 The EFG Unit Verification Process

After the wire-wrapped prototype and embedded software development had been completed, experiments were conducted to investigate the correct operation of the EFG logic (Figure 6-8). During this experiment, the Softing Analyser/Emulator was used to transmit messages on the network bus. The embedded software on the EFG module was executed via the Hitop monitor program. Once the Hitop monitor program was running, the EFG unit could be treated as an independent embedded system. Commands to the EFG unit were issued using the Softing Emulator. The Softing emulator has a window to show the network traffic. It can be seen that the EFG response message will appear every time an EFG request message is sent to the EFG module. This verifies that the EFG unit is active.

A DeviceNet flex I/O and a photo-electric sensor were attached to the bus as an experiment control. These two DeviceNet devices were in idle mode.

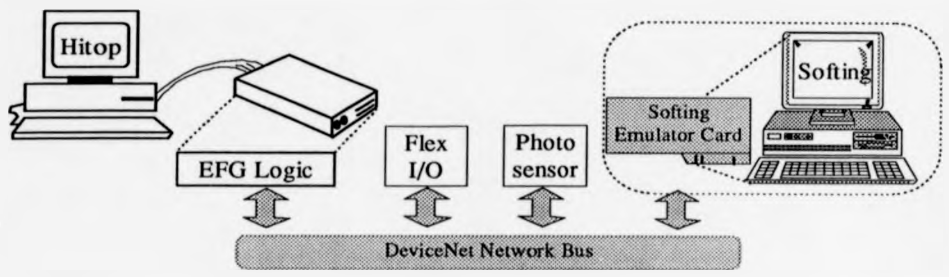


Figure 6-8 The EFG Verification Test Set-up

Experiment (I)

The EFG request message is a fixed data length message. In order to verify that the EFG unit only responds to the correct EFG request message, EFG request messages with different data length were sent. The EFG unit must not respond to any of the invalid data length EFG request messages.

Observation: EFG only response to Identifier 7C0 hex with data length code = 6.

Experiment (II)

An identifier (which is to be "zapped" by the EFG) is downloaded into the EFG via the EFG Request message (DUT_Identifier). The EFG's pattern comparator is enabled which arms the EFG unit for error injection. The same identifier is then downloaded into the Softing emulator for cyclic transmission.

Observation: The node (Softing emulator) that transmitted the DUT_identifier went into bus-off state immediately.

Experiment (III)

Experiment (II) is repeated but a single shot transmission is used rather than cyclic transmission.

Observation: Softing emulator gone into bus-off state upon the transmission of the single shot message.

The same result as Experiment (II) is obtained, i.e. the Softing emulator which transmits the cyclic identifier went into bus-off state upon the arming of the EFG unit. Experiment (II) and (III) were repeated with different DUT, DUT-1 and DUT+1. The summary of the test results can be seen in Table 6-9.

Table 6-9 A summary observation results of the experiment

		State of operations		
	Identifier (ID) Transmitted by Softing Emulator	Softing Emulator	EFG unit	Flex I/O & Photo-sensor
I	DUT_ID *(Cyclic)*	bus-off	error-active[††]	error-active[†]
II	DUT_ID - 1 *(Cyclic)*	normal	normal	normal
III	DUT_ID + 1 *(Cyclic)*	normal	normal	normal
IV	DUT_ID *(One-shot)*	bus-off	error-active[††]	error-active[†]
V	DUT_ID - 1 *(One-shot)*	normal	normal	normal
VI	DUT_ID + 1 *(One-shot)*	normal	normal	normal

[†]The error-active state is deduced based on the CAN protocol specification. The deduction is made because both the scanner and photo-sensor do not have any indicator to indicate the error-active state. The solid green communication status LEDs indicated that communication is healthy and unaffected.

[††]Since the Error Frame Register is not implemented in this experiment, the error-active state can only be assumed.

In all the above experiments, the EFG unit, Flex I/O and photo-eye remained active, even though the Softing emulator/analyser had gone into the bus-off state. From CAN protocol specification, it can be deduced that those nodes which remained active in the experiments were in error-active states.

Both the cyclic and one-shot transmission of the DUT_ID gave the same results. This is significant as it confirmed the fact that retransmission of CAN protocol during error existed. This is the reason why both cyclic and single shot gave the same results.

The transmission of DUT_ID, DUT_ID + 1 and DUT_ID - 1 proved that the EFG unit can efficiently distinguish the right message frame to be zapped into bus-off state. This verifies the internal operation of the pattern comparator, registers and EFG embedded software. Figure 6-9 shows the sequences involved in zapping a node into the bus-off state.

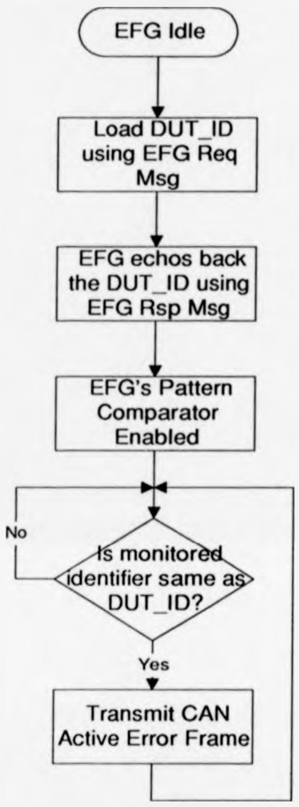

EFG Idle
Load DUT_ID using EFG Req Msg
EFG echos back the DUT_ID using EFG Rsp Msg
EFG's Pattern Comparator Enabled
No
Is monitored identifier same as DUT_ID?
Yes
Transmit CAN Active Error Frame

Figure 6-9 The error injection sequence

6.7 EFG Unit Constraints

Due to the lack of time and resources, not all the designed components of the EFG were implemented. The most vital part of the design, i.e. the ability to recognise a DUT_ID and zap the message frame with an active error frame is implemented successfully. The rest of the EFG features which allow fine-tuning of the EFG operations are not implemented. The followings are the constraints and abnormality found in the EFG design.

6.7.1 EFG Request/Response Message priority

The EFG unit is controlled by the host PC using the EFG Request/Response messages. Currently, both messages use the reserved DeviceNet identifiers, i.e. identifier 7C0 hex for EFG Request and 7C1 for EFG Response messages. The use of low priority messages for the ferrying of commands/controls may not be favourable in certain situations. This is especially true in the situation of a heavily loaded network where it may be impossible to get the command across.

For instance, the Softing emulator was configured to do cyclic transmission of Identifier 00 hex at 0ms interval. The cyclic transmission of such message prevented any EFG Request/Response message from getting to the EFG unit.

Between the EFG Request and Response message, it is justifiable to assign the EFG Request message to have a higher priority than the response. The argument behind this is that the EFG unit is an intelligent node which is self-sufficient to carry out the assigned tasks. The EFG Response message is used to verify the received command and data with the host PC. If the heavy network traffic prevents the EFG Response message from getting back to the host, the EFG unit will still be able to execute the host's previously issued command. The host, however, could not determine certain parameters such as the number of active error frames transmitted prior to the DUT bus-off state. In this situation, the host can either wait for the EFG Response message, or issue another EFG Request message to initiate another EFG Response message.

CHAPTER 7 - DISCUSSIONS

7.1 CAN Bit Timing Parameters Verification

The bit timing parameters of the CAN protocol will influence the correct operation of DeviceNet fieldbus communication. Instead of using high precision and expensive measuring equipment to verify the bit timing parameters implemented by the product developers, an alternative solution can be suggested consisting of two steps. The first step is to ask the product developer to find out which CAN controller has been used in the implementation, and what is the corresponding bit timing parameters. This gives a good indication as to whether the implementation uses compatible bit timing parameters. If the CAN controller used is new to DeviceNet, the ODVA Conformance SIG must appoint a lab to investigate the appropriate bit timing parameters for the new device. Once the optimum bit timing parameter is determined, the information will be available to all product developers for reference.

Once satisfied with the correct bit timing parameters, the implementation is then connected to a full size DeviceNet network with maximum length trunk and drop cables. The implementation is successively configured for operation in all the 3 DeviceNet baud rates, i.e. 125kbit/s, 250kbit/s and 500kbit/s. If communication is successfully established, it can be deduced that the bit timing parameters and opto-isolation delay are within the scope of the DeviceNet fieldbus specification. The DeviceNet network used in the interoperability test exercise can be used for this purpose.

7.2 Local Control for EFG Unit

The error frame generator(EFG) unit within the conformance test engine (CTE) framework is remotely controlled by the host PC to allow fully automated conformance testing to be performed. If the EFG is to be used as a standalone unit, the prototype can be modified to accept input from a keypad or other input devices, instead of using the EFG Request and Response message. As such, a portable EFG embedded system can be developed. The portable unit can be used to inject errors in any CAN system so that the system behaviour under error conditions can be studied.

7.3 Global Error Generator

The error frame generator (EFG) unit has demonstrated a method of injecting errors onto the DeviceNet bus. The errors generated are referred to as local errors in the CAN protocol. The local error is appropriate in the context of the conformance testing framework as this error condition only affects the IUT(implementation under test). This allows the IUT to be observed by other CAN nodes on the bus, i.e. the test system. The global error in the CAN protocol is the error agreed and detected by all CAN nodes. A global error generator is regarded as unnecessary as it will only be useful in the verification of the error detection mechanism in CAN protocol. The error detection facility in the CAN protocol has been mathematically validated and proven.

7.4 Test Preamble/Post-amble Optimisation

The Unique Input/Output (UIO) method is used for generating the conformance test suites for DeviceNet. The methodology ensures that a consistent conformance test suite is obtained. However, the rural Chinese postman optimisation problem still exists. This is the problem that involves the preamble and post-amble of the conformance test. For example, in order to test the Vendor ID attribute within the Identity Object, the test preamble which includes an open explicit messaging connection, and the setting of the EPR value to zero needs to be performed. After performing the test pre-amble, the test system is now ready for the actual test, i.e. to send a get_attribute_single service to the Identity Object specifying the Vendor ID attribute. The result of the test is then analysed based on the response received. The test post-amble concerns the necessary steps taken after the test to prepare the IUT for the next test. In this instance, in order not to repeat the test preamble again, all the attributes within the Identity Object are tested before the explicit messaging connection is closed. The idea is to repeat as little as possible the non value added test preamble and post-amble steps. There is no single solution to the rural Chinese postman tour problem. The optimisation of the above described problem is beyond the scope of this thesis.

7.5 Uniform Resource Interpreter

The conformance test is a growing system. Whenever a new device profile is introduced, the conformance test suite must be developed to accommodate the newly introduced profile. Soon, conformance testing will become the bottle-neck for open standard fieldbus advancement. Any modification and upgrade to the protocol specification will not be reflected immediately by the conformance test suites.

The recommended solution is to design the CTE software to mimic the uniform resource locator (URL) browser of the Internet. Conformance test suites are encoded with special tags in the database. The CTE software will read the corresponding test suite database and control tags according to the information contained in the statement of conformance(SOC) file. The conformance test suites will then be compiled and run. Whenever a new device profile is made available, only the test suite database needs updating. This allows the CTE framework to grow with the DeviceNet technology.

7.6 Future Systems

7.6.1 Interoperability Test

A conformance test only verifies that the product developers have correctly implemented the protocol specification, i.e. to test the implementation as a node. Although the implementation has undergone and passed the conformance test, it may not be absolutely certain that all devices will work harmoniously as a system. This can be illustrated simply with the following example.

In the DeviceNet master-slave configuration scenario, the master has to proxy for all its slave devices. Every DeviceNet connection has a connection time-out time (i.e. the expected packet rate(EPR) time-out) agreed by both devices involved in the connection during the dynamic connection establishment. A DeviceNet network may consist of Group 2 and Group 2 only devices. Group 2 only devices consist of slave devices which use the much simpler predefined-master-slave messaging connection sets of DeviceNet. If an open explicit messaging connection request message to a slave device is received, the master device will intercept the request and proxy to all its slave devices.

If a network manager tool is used to obtain some information from a slave node during run-time (Figure 7.1), the DeviceNet master will intercept the explicit message targeted for its slave and respond on the slave's behalf. From the network manager tool view-point, the communication between the slave device and the tool has been successfully established. The tool is not aware that the response message actually came from the master device. A typical EPR time-out value for explicit messaging connection is set.

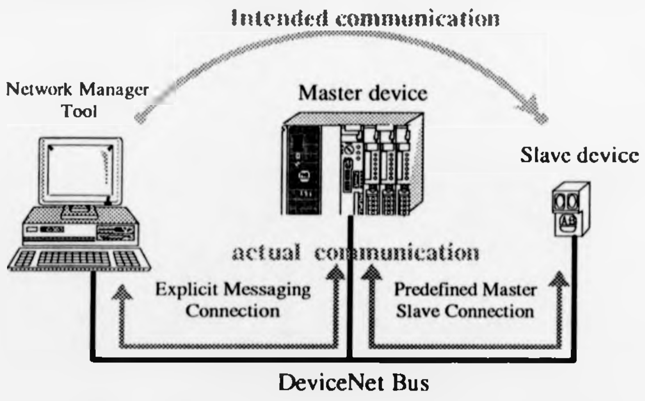


Figure 7-1 The communication between the network manager tool and a slave device owned by a master

If the information queried by the network manager tool is not available in the master device, the master device will have to open another connection to the slave device using the predefined master-slave connection set to get the information queried by the tool. As this involves another level of messaging connection, the timer for the explicit connection between the tool and the master will expire before the predefined master-slave connection's timer. As a result, the master could not provide the tool with the information requested. In this scenario, all devices involved are fully compliant with the protocol standard. The failure results because both the tool and the master device did not set their EPR time-out period appropriately. On the other hand, the master device did not realise that the tool was going to query for the information which is not available in the master device.

Therefore, an interoperability test is needed to find incompatibilities which may be due to incorrect configuration parameters, or the service semantics involved in interoperability. Furthermore, the interoperability test could provide valuable information on optimising the fieldbus system.

7.6.2 The Protocol Watch-dog

The EFG prototype has shown that a protocol emulator can be used to inject error into the DeviceNet bus and allow the error recovery management of the implementation to be studied. The EFG prototype can be further expanded into a strategic watch-dog tool that sits on the network and monitors the DeviceNet network. For example, the chattering of a node is not desired as it uses valuable network bandwidth. If the chattering node is transmitting a high priority message on a DeviceNet fieldbus (i.e. low CAN identifier number), lower priority messages will have difficulty in gaining bus access. This may jeopardise the correct operation of the overall system. In this situation, this network "watch dog" can be sprung into action by zapping the chattering node into the bus-off state.

Note that a chattering node is a healthy node which transmits healthy messages but at a much higher frequency than it was designed for. Therefore the fault-confinement of the CAN protocol does not apply in this instance. The supervisory node(watch-dog) can utilise the EFG unit to switch the abnormal node into bus-off. This provides an additional level of security to the operating network against failure caused by chattering nodes.

7.6.3 DeviceNet Fieldbus Emulator

An idea of developing a rapid prototyping machine to emulate the DeviceNet network as well as to perform the pre-runtime scheduling has been conceived. The principle behind this rapid prototyping machine is very similar to the in-circuit emulator used in embedded system development. The fieldbus emulator hardware set-up is identical to that used in the CTE framework design. The difference lies in the software that governs the hardware operations.

The fieldbus user can specify the system requirements and an environment is set-up to reflect the specified requirements as closely as possible. The DeviceNet emulator is then run which simulates the desired network conditions. This includes the mock-up PDU format and transfers. Alternatively, the emulator can be used to "play back" the actual network traffic captured earlier in a similar fieldbus installation. The end-user will then have an idea whether the planned network is capable of satisfying needs. If not, the various parameters on the network can be configured on the emulator before it is run again. For example, the user may want to know how much bandwidth can be gained by using the event-triggered transport class instead of the time-triggered option.

As the hardware set-up is identical to the CTE framework, this means that the conformance test lab can provide an additional pre-runtime scheduling service to the DeviceNet end-users.

7.6.4 DeviceNet Protocol Controller

It may be easier to put all the DeviceNet protocol messaging on a mask rather than to implement these messaging scheme in software. The only concern here will be the cost effectiveness of this approach. It is foreseen that if the conformance testing is efficient to weed out any irregularity of the implementation, the embedded software approach will definitely be better due to its flexibility. The product developers can develop products not only to conform to DeviceNet, but also to CANOpen, SDS, CAN Kingdom and any CAN system by altering the embedded software in the device. Furthermore, the use of the generic CAN controller will benefit the product vendors economically due to the mass utilisation of these controllers in the automotive industry. Any protocol upgrade can be done easily as there are no expensive masks to design.

7.6.5 Formal Methods for Protocol Specification Modelling

A formal method is needed to produce a formal DeviceNet protocol specification. By using formal methods to model the protocol specification, protocol errors can be found at a very early stage of the design. The conformance testing process addresses the issues of protocol verification, whereas a protocol validation process needs to be done using formal methods.

Conversely, formal methods suffer from the linguistic constraints where certain scenarios cannot be exactly modelled. Nevertheless, the results obtained will be better than those before its application as every methodology will disclose different kinds of problems. It is strongly suggested that a formal specification of DeviceNet, i.e. protocol validation should be done. Further delay in the formal specification process will make protocol fault rectification (if any) more complicated and costly. In addition, the formal specification may help to conform a fieldbus standard to an intrinsically safe fieldbus as mathematical proofs are obtained on the fieldbus behaviour.

7.6.6 Notation and Methodology Compilers

In the next decade, the demand for software engineers with programming language skills may decrease. There has been a considerable amount of research in the area of compiler design which converts formal design methodology models into run-time codes. For example, the ROSE tool, which is an object-oriented design methodology tool-kit, is capable of generating C++ run-time codes from object diagrams. It can also reverse engineer existing C++ run-time codes to create the object diagrams when required. Tools like these are on the increase. The maturing field of knowledge based and expert system will help to encapsulate the software experts' knowledge and include them in these compilers.

7.6.7 Knowledge-Based Fault Diagnostic Module

A knowledge-based inference machine could be incorporated into the CTE framework to diagnose the IUT's fault during conformance testing. Instead of giving a pass/fail verdict, the CTE would explain what has caused the IUT to fail the conformance test. A solution to the problem may then be recommended by the knowledge-base. This feature would be particularly useful to product developers during in-house testing of DeviceNet implementations.

7.6.8 Peer-to-peer and Highly Distributed Control

The ultimate aim of fieldbus control systems is the realisation of highly distributed control (HDC) systems. Fieldbus has provided the major structure needed for a highly distributed control system. With the use of an appropriate ladder logic compiler, the control and command can be residing on the field devices themselves. The argument behind this principle is very simple. A PLC program consists of rungs of control information with monitored inputs and outputs. Normally, there are only a few inputs to be monitored in each rung that trigger a specific output. If this information can be stripped down and loaded into the field device's memory, then the central controller such as a PLC is no longer needed for storing and executing the control sequences. This will realise the full automation hierarchy that has been the dream of the past decade.

Even though DeviceNet is designed with a client/server relationship in data production and consumption, the move to highly distributed peer-to-peer has many unforeseen boundaries. In theory, the producer/consumer model of DeviceNet will be valid for master/slave as well as peer-to-peer configuration. However, the peer-to-peer configuration may not benefit from the fast multi-cast bit-strobe command of the predefined master/slave connection set. All peer-to-peer communications take the form of request/response messages (Group 2 messages) which may be slower depending on the network bandwidth.

Chapter 8 - Conclusions

A method of injecting errors into the DeviceNet fieldbus to allow conformance testing on error recovery and management to be carried out has been developed. This prototype has been lab-tested to be workable. A conformance test engine, CTE has been devised to allow an automated conformance test approach. The automated CTE allows reliable and consistent conformance test results to be obtained without having to rely on the skills of conformance test engineers. In addition, an automated test also saves time and personnel. Furthermore, the designed conformance test framework and error frame generator can be used in conformance testing and error injection for all CAN based protocols.

The Unique Input/Output (UIO) method has been identified and used in generating the conformance test suites for the DeviceNet fieldbus. The use of Booch object-oriented design methodology in DeviceNet and CTE design is appropriate. The object-oriented approach complements the Windows component software integration. It is foreseen that formal method validation of the DeviceNet protocol must be performed at some stage to validate the DeviceNet specification.

An interoperability test must be performed to investigate the interactions between conformance devices on a full sized DeviceNet network. The interoperability test will ensure the correct inter-device communications.

It can also be used to determine the feasibility of a field application which relies on the combined efforts of various devices.

It is suggested that a knowledge-based inference engine should be developed and incorporated into the CTE framework for the diagnosis of non-conforming implementation in future.

References

[1] O.J. Struger, "*Controlling the Future*", Electrical Review, 16-29 April 1996, pp. 33-37.

[2] I.G. Warnock, "*Programmable Controllers - Operation and Application*", Prentice Hall, 1988, ISBN 0-13-730037-9.

[3] B. Gates, "*The Road Ahead*", Penguin Books Ltd, 1995, ISBN 0-670-85913-3.

[4] "*Guidelines to Statistical Process Control*", The Society of Motor Manufacturers and Traders(SMTT). London 1986.

[5] M. Santori & K.Zech, "*Fieldbus Brings Protocol to Process Control*", IEEE Spectrum, Vol. 33, No. 3, March 1996, pp. 60-64.

[6] "*Fieldbus - The Executive Guide*", Department of Trade and Industry Publication, 1993.

[7] D.M. Kelly, "*Digital Fieldbus Cluster Cuts Plant's Wiring Costs up to 20%*", InTech, April 1995, Vol.42, No.4, pp.62-64.

[8] B. Squires, "*Fieldbus Trials at BP*", The IEE Computing & Control Engineering Journal, Dec. 1995, pp. 254-258.

[9] S. Jasinski & M. Macicjewski, "*Profibus Goes Underground in Warsaw*", Assembly Automation, Vol.14 No. 1 1994.

[10] "*11 Km Under Warsaw*", ProfiNews, Issue 2, December 1994.

[11] "*WorldFip*", French National Fieldbus Standard Specification, NFC 46-602/603/604/605/606/607.

[12] "*Profibus Standard - Part 1, 2, 3(DP)*", German standard DIN 19245, April 1991.

[13] "*The P-NET Fieldbus for Process Automation*", The International P-NET User Organisation, 1995

[14] G. Wood, "*Fieldbus Status 1995*", The IEE Computing & Control Engineering Journal, Dec. 1995, pp. 251-253.

[15] InTech, "*Europe OKs Three Fieldbusses as Standards*", The Int. Journal for Measurement and Control, ISA Publication, July 1996, pp. 30.

[16] "*Bosch-CAN Specification Version* 2.0", Robert Bosch GmbH, Postfach 50, D-7000 Stuttgart 1, Sept. 1991.

[17] "*CAN Application Layer(CAL) for Industrial Applications - CiA/DS 201...CiA/DS 205, CiA/DS 207*", CiA, 1 May 1995, Germany.

[18] G. Gruhler & M.Rostan, "*Interoperable Automation : Components Using CANopen Profiles*", Proceedings of the 2nd International CAN Conference, London, 1995, pp.2.11-22.,

[19] L. B. Fredriksson, "*Controller Area Networks and The Protocol CAN for Machine Control Systems*", Mechatronics, Vol. 4, No. 2, 1994, pp 159-172.

[20] "*DeviceNet Communication Link Overview*", Allen Bradley Publication No. 1787-1.1 August 1994.

[21] M. Babb, "*New Sensors Have Intelligence, Will Communicate*", Control Engineering, February 1994.

[22] P.P Dierauer, "*SDS : A CAN Protocol for Plant Floor Control*", Proceedings of the COMETT CAN International Seminar, 10-11 Oct. 1994, Birmingham, UK.

[23] "*SDS-Application Layer Specification*", Honeywell Inc. 1994.

[24] L. Gould, "*While Waiting for Fieldbus and Sensor Bus, Reconsider Lonworks*", Managing Automation, December 1994.

[25] "*Interoperable Control Networks Using LonWorks Technology*", Seminar Notes, 7 June 1995, Motorola Aylesbury, UK.

[26] K. Bender, "*Profibus : The Fieldbus for Industrial Automation*" Prentice Hall, 1993, ISBN 0-13-012691-9.

[27] M. Volz, "*Profibus - Technical Description*", Profibus User Organisation, Germany, 1994.

[28] G.J. Holzmann, "*Design and Validation of Computer Protocols*", Prentice Hall, 1991, ISBN 0-13-539925-4.

[29] C. McLean, M. Mitchell, E.Barkmeyer, "*A Computer Architecture for Small-batch Manufacturing*", IEEE Spectrum, May 1993, pp. 59-64.

[30] I.G. Warnock, "*Programmable Controllers - Operation and Application*", Prentice Hall, 1988, ISBN 0-13-730037-9

[31] A. Valenzano, C.Demartini, L.Ciminera, "*MAP and TOP Communications - Standards and Applications*", Addison-Wesley, 1992, ISBN 0-201-41665-4

[32] G. Cena, L. Durante & A. Valenzano, "*Standard Fieldbus Networks for Industrial Applications*", Computer Standards & Interfaces 17, 1995, Elsevier Science, B.V., pp. 155-167.

[33] A. Chatha, "*Fieldbus:The Foundation for Field Control Systems*", Control Engineering, May 1994, pp. 77-80

[34] M. Henry, "*Sensor Validation and Fieldbus*", The IEE Computing & Control Engineering Journal, December 1995, pp. 263-239.

[35] V. C., Jones, "MAP/TOP Networking-A Foundation for Computer-Integrated Manufacturing", McGrawHill, 1988.

[36] J.D. Decogtignie, & P. Raja, "*Fulfilling Temporal Constraints in Fieldbus*", Proceedings of the IEEE 19th Annual International Conference on Industrial Electronics, Control and Instrumentation 1993, IEEE Cat 93CH3234-2.

[37] C. Ajluni, "*Interoperability: The Latest Buzzword in Sensors*" , Electronic Design, 5 Dec 1994, pp.59-62.

[38] G. Wood, "*Fieldbus Status 1995*", The IEE Computing & Control Engineering Journal, December 1995, Vol. 6, No. 6, pp.251-253.

[39] M.Machacek & F. Russo, "*The EIAMUG Project - Intelligent actuation and measurements as users need them*", The IEE Computing & Control Engineering Journal, December 1995, Vol. 6, No. 6, pp. 273-281.

[40] U. Kiencke, " *Controller Area Network - from concept to reality*", Proceedings of the 1st. International CAN Conference, Mainz, Germany pp. O.11 - O.20, 1994.

[41] E. Russell, "*Electronics on the Road*", Electronics World + Wireless World, May 1995, pp. 372-377.

[42] "*ISO 11898 - Road Vehicles- Interchange of Digital Information - Controller Area Network (CAN) for high-speed Communication*", First Edition, 1993-11-15, Amendment 1:1995, International Organisation for Standardisation, Switzerland.

[43] "*ISO 11519-2 Road vehicles -- Low-speed serial data communication -- Part 2: Low-speed controller area network (CAN)*", 1994, Amendment 1:1995, International Organisation for Standardisation, Switzerland.

[44] *"BMW 1993 Car Catalogue"*, BMW AG, Munich, Germany 1992.

[45] A.Croft, *"The XK-8 High Speed Powertrain Serial Communications System"*, IEE Colloquium on the Electrical System of the Jaguar XK-8, 18 Oct. 96, London, 1996, Digest No. 96/281, pp. 3.1 - 3.17.

[46] S. Josifovska, *"In the CAN"*, Electronics Weekly, 15 November 1995, pp.24.

[47] R.T. McLaughlin & S.B. Khoh, *"Autos Carry the CAN-Controller Area Network (CAN) for Industrial Applications"*, Assembly Automation, Vol. 14 No.1, MCB University Press, 1994, 0144-5154, pp. 17-19.

[48] M. Jaggi, *"CAN in Industrial Applications"*, Proceedings of the 1st. International CAN Conference, Germany, 1994, pp. 4.9-12.

[49] G. Cena and A. Valenzano, *"A Distributed Mechanism to Improve Fairness in CAN Networks"*, Proceedings of IEEE International Workshop on Factory Communication Systems, WFCS'95, Oct 1995, IEEE Cat. 95TH8141.

[50] K. Tindell and A. Burns, *"Guaranteeing Message Latencies on Control Area Network (CAN)"*, pp. 1.2-1.11, Proceedings of the 1st. International CAN Conference, Mainz, CiA Germany, 1994.

[51] N.J. Carter, C.R.Boyce and J.A.Philpot, *"Electromagnetic Compatibility (EMC) Comparison of Data Bus Media"*, IMechE 1989, Digest No. C391/033, pp. 75-78.

[52] R.T. McLaughlin, *"The Immunity to RF Interference of A CAN System"*, IEE Colloquium on The Integrity of Automotive Electronic Systems, IEE Digest No. 1993/063.

[53] D. Noonen, S.Siegel, P.Maloney, *"DeviceNet Application Protocol"*, Proceedings of the 1st. International CAN Conference, 1994, CiA Germany, pp. 10.11-18.

[54] E.J. Heins, "*A Real-time PC with iRMX for Windows controls Medical System Components directly via a CAN Network*", Proceedings of the 1st. International CAN Conference, CiA Germany, 1994, pp. 5.2-10.

[55] G.R. Peterson, "*A Low-cost Serial Control and Data Bus for Airframes*", Proceedings of the IEEE/AIAA 13th. Digital Avionics Systems Conference, 1994.

[56] M.Stock, "*CAN in an Airborne Flight Data Recording System*", Proceedings of the 3rd. International CAN Conference, CiA Germany, 1996, pp.5.18-24.

[57] P. Priller, "*A Distributed Access Control System Using CAN*", Proceedings of the 1st. International CAN Conference, CiA Germany 1994, pp. 12.20-21.

[58] Lawrenz, W., "*Worldwide Status of CAN - Present and Future*", Proceedings of the 2nd. International CAN Conference, CiA Germany, 1995, pp.0.12-25

[59] J. Hanneman, B. Lounsbury and S. Siegel, "*DeviceNet: Physical Layer, Media and Power Capabilities*", Proceedings of the 2nd. International CAN Conference, CiA Germany, 1995, pp. 6.2-6.11.

[60] P. Buehring, "*Bit Timing Parameters for CAN Networks*", Philips Application Note KIE 07/91ME, 20/03/91, Hamburg, Germany, 1991.

[61] S.B. Khoh, "*CAN Bus Timing Parameters Analysis*", Controller Network News, Issue No. 4, Spring/Summer 1995, University of Warwick.

[62] "*DeviceNet Specification - Volume I & II, Release 1.1*", Allen-Bradley Publication no. 1787-7.1 August 1994.

[63] "*Error Detection Capabilities of the CAN Protocol*", Robert Bosch GmbH, Stuttgart, October 1989.

[64] "*NEURON® Chip Distributed Communications and Control Processors - MC 143150, MC 143120*", Motorola Inc. 1994.

[65] A.v.D. Heuvel, "*Philips PCA82C250(T)/PCA82C250-N4 Tentative Device Specification*", Nederlandse Philips bedrijven B.V., 16 Nov 1993.

[66] R.J. Linn, M.Ümit Uyar, "*Conformance Testing Methodologies and Architectures for OSI Protocols*", IEEE Computer Society Press, 1994, Cat. No. EH0390-5.

[67] "*Computing*", VNU Publication, 25 July 1996, UK.

[68] CCITT Recommendation Z.100, "*Specification and Description Language (SDL)*",COM X-R15-E, 1987.

[69] "*ISO 9074 : Estelle : A Formal Description Techniqur Based on an Extended State Transition Model*", " , International Organisation for Standardisation, 1989.

[70] "*ISO 8807 : LOTOS-A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour*", " , International Organisation for Standardisation, 1989.

[71] G. Neufeld and S.Vuong, "*An Overview of ASN.1*", Computer Networks and ISDN Systems, Vol. 23, 1992, pp.393-415.

[72] G. Cena, Demartini, C., Durante., L., "*An Object Oriented Model for the FIP protocol*", IECON'94 - 2nd International Conference on Industrial Electronics, Control and Instrumentation, IEEE, 1994, pp. 1214-1219.

[73] G. Juanole and Gallon, L., "*Formal Modelling and Analysis of a Critical Time Communication Protocol*", Proceedings of IEEE International Workshop on Factory Communication Systems, WPCS - Switzerland, Oct. 1995, pp. 107-115.

[74] Z. Kohavi, "*Switching and Finite Automata Theory - 2nd Edition*", McGrawHill Publ., 1978, Reprint 1986, ISBN 0-07-099387-4.

[75] G.V. Bochmann, "*Finite State Description of Communication Protocols*", Linn, R.J., Ümit Uyar, M., "Conformance Testing Methodologies and Architectures for OSI Protocols", IEEE- CS Press, 1994, pp. 66-77.

[76] D. Rayner, "*OSI Conformance Testing*", Computer Networks and ISDN Systems, Vol. 14, 1987, pp. 79-98.

[77] Pierson, L.L., "*Broader Fieldbus Standards will Improve System Functionality*", Control Engineering, November 1994, pp. 58-59.

[78] Naito, S., and M. Tsunoyama, "*Fault Detection for Sequential Machines by Transition Tours*", Proceedings 11th IEEE Fault Tolerant Computing Symposium, IEEE Computer Society Press, 1981, pp. 238-243.

[79] Sarikaya, B., and G.V. Bochmann, "*Some Experience with Test Sequence Generation for Protocols*", Proceedings 2nd International Workshop on Protocol Specification, Testing and Verification", 1982, pp. 555-567.

[80] Vuong, S.T., W.L. Chan and M.L. Ito, "*The UIOv-method for Protocol Test Sequence Generation*", Proceeding of the 2nd. International Workshop on Protocol Test Systems, Berlin, Oct. 3-6, 1989.

[81] Sato, F., J. Munemori, T.Ideguchi and T.Mizuno, "*Test Sequence Generation Method Based On Finite Automata - Single Transition Checking Method Using W Set*", Trans. EIC Vol. J72-B-1, no.3, pp. 183-192, 1989.

[82] Fujiwara, S., G.V. Bochmann, F.Khendek, M. Amalou and A. Ghedamsi, "*Test Selection Based on Finite State Models*", IEEE Trans. Software Engineering, Vol. 17, No. 6, June 1991.

[83] Bhattacharyya, A., "*Checking Experiments in Sequential Machines*", John Wiley & Sons, 1989.

[84] K. Sabnani and Dahbura, A., "*A Protocol Test Generation Procedure*", Computer Networks and ISDN Systems, Vol.15, 1988, pp. 285-297.

[85] Aho, A.V., Dahbura, A.T., Lee, D. and Ümit Uyar, M., "*An Optimisation Technique for Protocol Conformance Test Generation Based on UIO Sequences and Rural Chinese Postman Tours*", IEEE Trans. Of Communication, Vol. 39, No.11, Nov. 1991, pp. 1604-1615.

[86] K. Binder, & D.W. Heermann, "*Monte-Carlo Simulation in Statistical Physics - An Introduction*", Springer-Verlag, 1988, ISBN 0-387-19107-0.

[87] Sidhu, D. and T.K. Leung, "*Fault Coverage of Protocol Test Methods*", Proceedings of INFOCOM, 1988, pp. 80-85.

[88] "*ISO/IEC 7498-1 : Information technology- Open Systems Interconnection - Basic Reference Model*", International Organisation for Standardisation, 1994.

[89] A.T. Dahbura, and Sabnani, K.K., "*An Experience in Estimating Fault Coverage of a Protocol Test*", Proceedings IEEE INFOCOM, IEEE Computer Society Press., 1988, pp.71-79.

[90] R.L. Robert, O. Monkewich, "*TTCN:The International Notation for Specifying Tests of Communication Systems*", Computer Networks and ISDN Systems, Vol.23, 1992, pp.417-438.

[91] "*ISO/IEC 9646 : Information Technology - Open Systems Interconnection - Conformance Testing Methodology and Framework*"- Part 1, 2, 4, 5, 6 - 1994, Part 3 - 1992, Part 5 - 1995, International Organisation for Standardisation.

[92] "*ISO 10303-31 Industrial Automation Systems and Integration - Product Data Representation and Exchange - Part 31:Conformance Testing Methodology and Framework - General Concepts*", 1994, International Organisation for Standardisation

[93] "*ISO/DIS 10303-32 Industrial Automation Systems and Integration - Product Data Representation and Exchange - Part 32:Conformance Testing Methodology and Framework - Requirements on Testing Laboratories and Clients*", International Organisation for Standardisation

[94] "*DeviceNet Protocol Conformance Test Specification*", Version 0.97-5/08/96, ODVA Conformance SIG Publication, 1996.

[95] R.Weber, K. Thelen, A. Srivastava & J. Krueger, "*Automated Validation Test Generation*", Proceedings of the 13th Digital Avionics Systems Conference, 1994, IEEE/AIAA, pp. 99-104.

[96] A. Avritzer, E.J. Weyuker, "*The Automatic Generation of Load Test Suites and the Assessment of the Resulting Software*", IEEE Transactions on Software Engineering, Vol. 21, No. 9, Sept. 1995, pp. 705-715, 927.

[97] Johnson, P., "*Human Computer Interaction - Psychology, Task Analysis and Software Engineering*", McGrawHill 1992, ISBN 0-07-707235-9

[98] M. Clarkson, "*Process Control's New Face*", Byte, Oct 1994, pp.111-118.

[99] InTech, "*(Micro)soft PLCs reign at IPC*", July 1996, pp. 17.

[100] E.J. Heins, "*A Real-time PC with iRMX for Windows Controls Medical System Components directly via a CAN network*", Proceedings of the 1st International CAN Conference 1994, CiA, pp.5.2-5.10

[101] Cohen, D.M., Dalal, S.R., Kajla, A., Patton, G.C., "*The Automatic Efficient Test Generator (AETG) System*", Proceedings of 5th International Symposium on Software Reliability Engineering", IEEE Cat. No. 94TH8017, ISBN/ISSN: 0-8186-6665-x, 1994

BIBLIOGRAPHY

[1] "*80C51-Based 8-Bit Microcontrollers - Data Handbook*", Philips Semiconductor, 1994.

[2] "*Fieldbus Devices - A Changing Future*", IEE Colloquium Digest No. 1994/236, IEE, 1994.

[3] "*The Electrical System of The Jaguar XK-8*", IEE Colloquium Digest No. 96/281, IEE, 1996.

[4] "*Using Rational ROSE/C++*", Rational Software Corp, May 1995.

[5] A.E.A Almaini., "*Electronic Logic Systems*", 3rd. Edition, ISBN 0-13-253519-X.

[6] A. Avritzer, and E.J. Weyuker, "*The Automatic Generation of Load Test Suites and the Accessment of the Resulting Software*", IEEE Transactions on Software Engineering, Vol.21, No.9, Sep. 1995. pp.705-715, pp. 927(corrections).

[7] S. Bennett, "*Real-time Computer Control : An Introduction*", Prentice-Hall, 1988, ISBN 0-13-762501-4.

[8] J. Berra, "*Delivering the True Promise of Fieldbus*", InTech, July 1996, pp. 37-41.

[9] G. Booch, "*Object Solutions - Managing the Object-Oriented Project*", 1996, Addison-Wesley, ISBN 0-8053-0594-7.

[10] G. Booch, "*Object-Oriented Analysis and Design with Applications*", 2nd. Edition, ISBN 0-8053-5340-2, Benjamin/Cummings Publish., 1994.

[11] B.B. Brey, "*The Intel Microprocessors 8086/8088, 80186, 80286, 80386 and 80486 - Architecture, Programming and Interfacing*", Maxwell-Macmillan, 1994, ISBN 0-02-946322-X.

[12] J.F. Brown, "*Embedded Systems Programming in C and Assembly*", Van Nostrand Reinhold, 1994, ISBN 0-442-01817-7.

[13] A. Caristi, "*IEEE-488 : General Purpose Instrumentation Bus Manual*", Academic Press. 1989, ISBN 0-12-159820-9

[14] S. Cavalieri, A.D. Stefano and O. Mirabella, "*Optimization of Acyclic Bandwidth Allocation Exploiting the Priority Mechanism in the Fieldbus Data Link Layer*", IEEE Transactions on Industrial Electronics, Vol. 40, No. 3, June 1993, pp. 297-306.

[15] D.M. Cohen, S.R. Dalal, A. Kajla & G.C. Patton, "*The Automatic Efficient Test Generator (AETG) System*", Proceedings of the 5th International Symposium on Software Reliability Engineering, IEEE Cat. No. 94TH 8017, ISBN/ISSN: 0 8186 6665-X, 1994, pp. 303-309.

[16] D. Crane, J.F. Leathrum and K.A. Liburdy, "*Test Data Visualization for Open Systems Standards*", Proceedings of the SPIE - The International Society for Optical Engineering, 1994 Vol. 2178, pp. 154-164.

[17] R.A. Day, "*How to Write and Publish A Scientific Paper*", 3rd. Edition, ISBN 0-521-36572-4, Cambridge University Press, 1989.

[18] J.D. Decotignie, & P. Raja, "*Fulfilling Temporal Constraints in Fieldbus*", Proceedings of the Annual International Conference on Industrial Electronics, Control and Instrumentation, IEEE, 1993, pp. 519-524.

[19] H. Deitel, and P. Deitel, "*C & C++ Multimedia Cyber Classroom*", Prentice-Hall, 1996, ISBN 0-132-31374-x.

[20] R. Dettmer, "*A Class Act - The Rise of Object-Oriented Technology*", IEE Review, November 1995, pp. 253-256.

[21] J. Dwyer, "*The Networking CAN CAN*", New Electronics, 24 Oct. 1995.

[22] M.A. Ellis, & B. Stroustrup, "*The Annotated C++ Reference Manual (ARM)*", Addison Wesley, 1990, ISBN 0-201-51459-1.

[23] J.V. Emden, "*Handbook of Writing for Engineers*", MacMillian, 1992, ISBN 0-333-46942-9.

[24] M. Ettl, U. Klehmet, "*A Performance Comparison Between the Fieldbus Protocol Standards PROFIBUS and FIP*", [M. Cosnard and R. Puigianer, "Decentralised and Distributed Systems (A-39)", Elsevier Science B.V., Holland, 1993 IFIP, pp. 245-258.

[25] J. Gerald, "*PC-Based Test:-Hardware and Software Innovations*", Evaluation Engineering, April 1991, pp. 12-19.

[26] J.S. Gerold, "*MES Links Production Planning to Plant-Floor Control*", Control Engineering, September 1994, pp. 119-121.

[27] G.A. Gibson, Y.C. Liu, "*Microcomputers for Engineers and Scientists*", 1987, Prentice-Hall, ISBN 0-13-586728-2.

[28] K.E. Gorlen, S. M. Orlow, P. S. Plexico, "*Data Abstraction and Object-Oriented Programming in C++*", John Wiley & Sons, 1990, ISBN 0-471-92346-x.

[29] M.P. Groover, "*Automation, Production Systems, and Computer Integrated Manufacturing*", Prentice-Hall, 1987, ISBN 0-13-054610-0.

[30] A. Habbadi, A.E.K. Sahraoui, "*Fieldbus for CIM Applications - A Case Study*", pp. 354-359.

[31] D.V. Hall, "*Microprocessors and Interfacing - Programming and Hardware*", 2nd. Edition, McGraw-Hill, 1992, ISBN 0-07-112636-8.

[32] F. Halsall, "*Data Communications, Computer Networks and Open Systems*", 4th. Edition, Addison-Wesley, 1996, ISBN 0-201-42293-X.

[33] J. Ham, "*Fieldbus : The Tools are Here*", InTech, March, 1995, pp. 36-38.

[34] M. Harrison, Thimbleby, H., "*Formal Methods in Human-Computer Interaction*", Cambridge University Press, 1990, ISBN 0-521-37202-x

[35] K. Hintz, & D. Tabak ,"*Microcontrollers - Architecture, Implementation and Programming*", McGrawHill, 1992, ISBN 0-07-112957-X.

[36] P. Horowitz, and T.C. Hayes, "*The Art of Electronics*", Cambridge University Press, 1989, ISBN 0-521-37095-7.

[37] P. Horowitz and T.C. Hayes, "*Student Manual for The Art of Electronics*" Cambridge University Press, 1989, ISBN 0-521-37709-9.

[38] J.P. Bowen, M.G. Hinchey, "*Ten Commandments of Formal Methods*", IEEE Computer, Vol. 28, No. 4, April 1995, pp. 56-63.

[39] I. Jacobson, Christerson,M. & Constantine, L.L., "*The OOSE Method: A use-case driven approach*", in Carmichael, A.R. (Editior) "Object Development Methods", BCS SIG Books, 1994.

[40] I. Jacobson, M. Christerson, P. Jonsson, G. Övergaard, "*Object-Oriented Software Engineering*", Addison-Wesley/ACM Press, 1993, ISBN 0-201-54435-0.

[41] Karg, R., "*Networking Intelligent Systems for Cost-Effective Automation*", Assembly Automation, Vol.13 No.3 1993, pp. 24-27.

[42] A. Kelly, & I. Pohl, "*A Book on C - Programming in C*", 3rd. 1995.

[43] D.J. Kruglinski, "*Inside Visual C++ - Version 1.5*" 2nd Edition, Microsoft Press, 1994, ISBN 1-55615-661-8.

[44] W. Lawrenz, B. Shirvani, "*Industrial Control with CAN-Autobus*", Proceedings of the COMETT CAN International Seminar, Birmingham, UK, 10-11 Oct. 1994.

[45] A. Leach, "*Profibus: The German Fieldbus Standard*", Assembly Automation, Vol. 14 No. 1, 1994, pp. 8-12. ISSN 01445-5154

[46] D. Mandrioli, S. Morasca and A. Morzenti, "*Generating Test Cases for Real-time Systems from Logic Specifications*", ACM Transactions on Computer Systems, Vol.13, No. 4, November 1995, pp. 365-398.

[47] M.M. Mano, "*Computer System Architecture*", Prentice-Hall, 1982, ISBN 0-13-166637-1.

[48] R.T. McLaughlin, "*CAN Controlling from Cars to X-Rays*", Networking May 1995, (IEE Review May 1995) pp. iv-vi.

[49] R.T. McLaughlin & S.B. Khoh, "*A SLIO (Serial Link Input/Output) CAN Implementation*", SAE Transactions 941661.

[50] G. Meinert, "*Openness for Automation Networks*", InTech Oct. 1995, Vol. 42, No. 10, pp. 29-32.H. Zeltwanger, "An Inside Look at the Fundamentals of CAN", Control Engineering, January 1995, pp. 81-87.

[51] J. Mühlenkamp, "*Planning and Operation on Fieldbus Systems in Process Technology*", Ex Magazine, November 1992, pp. 34-40.

[52] A.J. O'Callaghan, and M.Leigh, "*Object Technology Transfer : Meeting The Training Needs of Software Development*", Alfred Waller, 1994, ISBN 1-872474-14-4.

[53] K. Ogata, "*Discrete-Time Control Systems*", Prentice-Hall, 1987, ISBN 0-13-216227-X

[54] C.H. Pappas, and W.H. Murray, III, "*The Visual C++ Handbook*", 2nd Edition, McGraw-Hill, 1995, ISBN 0-07-882125-8.

[55] A. Petzold, "*Programming Windows 3.1*", 3rd. Edition, Microsoft Press, 1992, ISBN 1-55615-395-3.

[56] E.M. Phillips, and D.S. Pugh, "*How to Get a Ph.D. : A Handbook for Students and Their Supervisors*", 1991, ISBN 0-335-15536-7.

[57] M.A. Plonus, "*Applied Electromagnetics*", McGraw-Hill, 1978, ISBN 0-07-Y66479-X.

[58] I. Pohl, "*C++ for C Programmers*", 2nd Edition, Benjamin/Cummings, 1994, ISBN 0-8053-3159-X.

[59] R.S. Raji, "*Smart Networks for Control*", IEEE Spectrum, Vol. 31, No. 6, June 1994, pp. 49-55.

[60] E. Rich & K. Knight, "*Artificial Intelligence*", 2nd. Edition, McGrawHill, 1991, ISBN 0-07-100894-2.

[61] M.G. Rodd, "*Engineering Real-time Systems*", The IEE Computing & Control Engineering Journal, October 1995, pp. 233-240.

[62] M.B. Rosson, "*Object-Oriented Analysis and Design with Applications-The Instructor's Guide*", Benjamin/Cummings, 1994, ISBN 0-8053-5341-0.

[63] K.E. Ruderstam, R.R. Newton, "*Surviving Your Dissertation*", Sage Publication, 1992, ISBN 0-8039-4563-9.

[64] M. Santori, K. Zech, "*Fieldbus Brings Protocol to Process Control*", IEEE Spectrum, Vol. 33, No. 3, March 1996, pp. 60-64.

[65] D.E.Y Sarna, & G.J. Febish, "*PC Magazine Windows Rapid Application Development*", Ziff-Davis Press, 1993, ISBN 1-56276-088-2.

[66] H. Schildt, "*Turbo C/C++ : The Complete Reference*", 2nd Edition, McGraw-Hill, 1992, ISBN 0-07-881776-5.

[67] A. Schimmele, "*The Fieldbus System*", Ex Magazine, December 1993, pp. 29-33.

[68] M. Schwartz, "*Information Transmission, Modulation, and Noise*", 4th. Edition, McGrawHill, 1990, ISBN 0-07-100931-0.

[69] S.H. Son, "*Advances in Real-Time Systems*", Prentice-Hall, 1995.

[70] A.D. Stefano, & O. Mirabella, "*Evaluating the Fieldbus Data Link Layer by a Petri Net-based Simulation*", The IEEE Transactions on Industrial Electronics, Vol. 38, No. 4 August 1991, pp. 288-297.

[71] H. S. Stone, "*Microcomputer Interfacing*", Addison-Wesley, 1983, ISBN 0-201-07403-6.

[72] J.W. Sullivan, Tyler, S.W., "*Intelligent User Interfaces*", ACM Press, 1991, ISBN 0-201-50305-0

[73] A.S. Tanenbaum, "*Computer Networks*", 3rd. Edition, Prentice-Hall, 1996, ISBN 0-13-349945-6.

[74] M. Thompson, "*The Thick and Thin of Car Cabling*", IEEE Spectrum, February 1996, pp. 42-45.

[75] R.J. Tocci, "*Microprocessors and Microcomputers - Hardware and Software*", 3rd Edition, Prentice-Hall, 1979, ISBN 0-13-581844-3.

[76] W. Tomasi, "*Advanced Electronic Communications Systems*", 3rd. Edition, Prentice-Hall, 1994.

[77] E. Turban, with L. E. Frenzel Jr., "*Expert Systems and Applied Artificial Intelligence*", ISBN 0-02-421665-8.

[78] F. Vasques, G. Juanole, "*Pre Run-Time Schedulability Analysis in Fieldbus Networks*", IECON'94, Proceedings of the 20th International Conference on Industrial Electronics, Control and Instrumentation, IEEE, 1994, pp. 1200-1204.

[79] J.F. Wakerly, "*Digital Design, Principles and Practices-2nd Edition*", Prentice-Hall International, ISBN 0-13-059973-5

[80] R. Weber, K. Thelen, A. Srivastava & J. Krueger, "*Automated Validation Test Generation*", Proceedings of the 13th. Digital Avionics Systems Conference, 1994, pp. 99-104.

[81] M.E. Woodward, "*Communication and Computer Networks : Modelling with Discrete-Time Queues*", ISBN 0-8186-5172-5.

[82] K.W. Young, R.T. McLaughlin & S.B. Khoh, "*Fieldbusses in Automated Manufacturing*", Internal Report, University of Warwick, 1995.

[83] K.W. Young, R.T. McLaughlin & S.B. Khoh, "*DeviceNet Interoperability and Compliance*", Proceedings of the 2nd International CAN conference, CiA Germany 1995, pp. 6.22-29.

[84] K.B. Zech, & O. Paul, "*Understanding Fieldbus*", Chemical Processing, March 1995.

ORIGINALS IN COLOUR

APPENDIX A

THE ERROR FRAME GENERATOR UNIT AND SCHEMATIC DIAGRAM

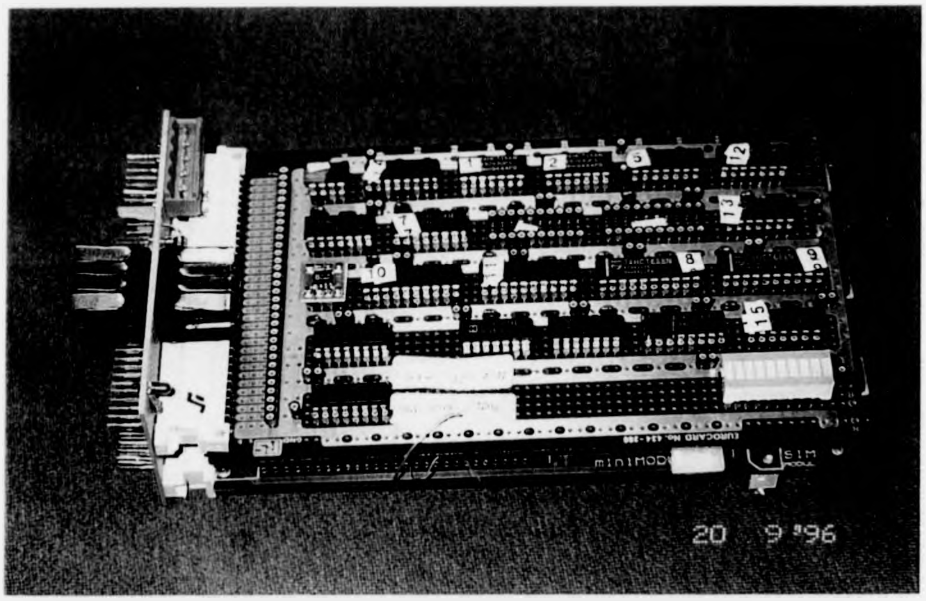


Figure A-1 The Error Frame Generator Unit

Figure A-1 shows the Error Frame Generator(EFG) prototype which consist of two Eurocards connected via a custom designed back-plane. On board voltage regulators (8~30V dc regulation) were used to allow the EFG to be powered from the DeviceNet bus. The bottom layer Eurocard is an off-the-shelf Philips 8xC592 microcontroller card, i.e. Phytec Mini-module 592. The upper layer Eurocard is the custom designed logic circuits for realising the CAN message screener and error frame generation logic. Figure A-2 and A-3 show the component side and the wire-wrapping side of the designed prototype.

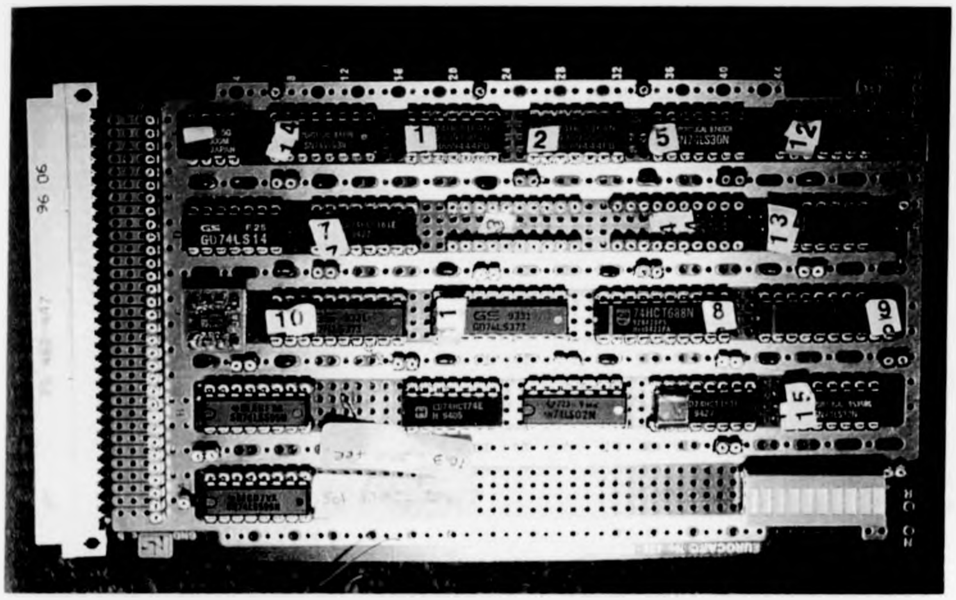


Figure A-2 The Component side of the designed prototype

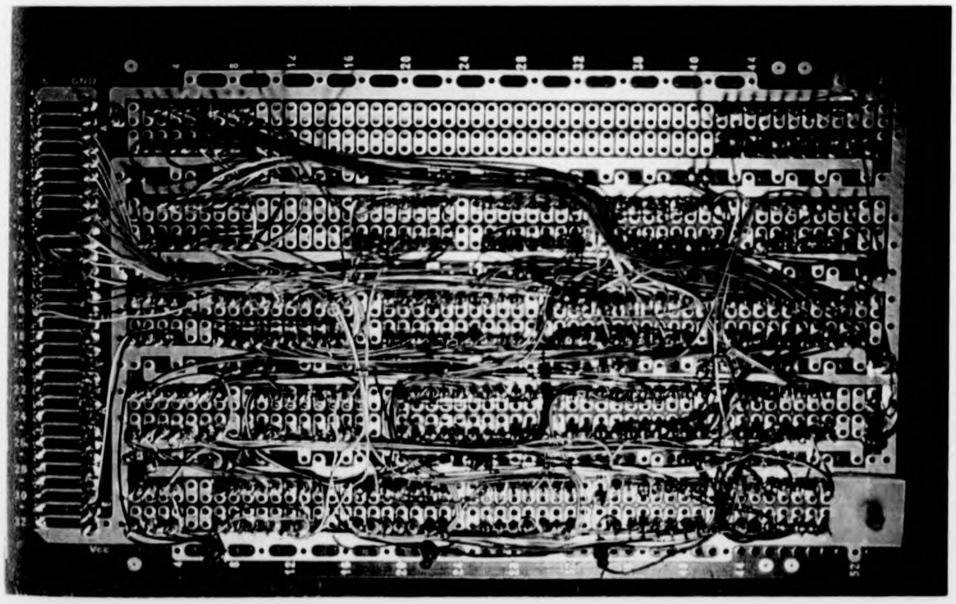
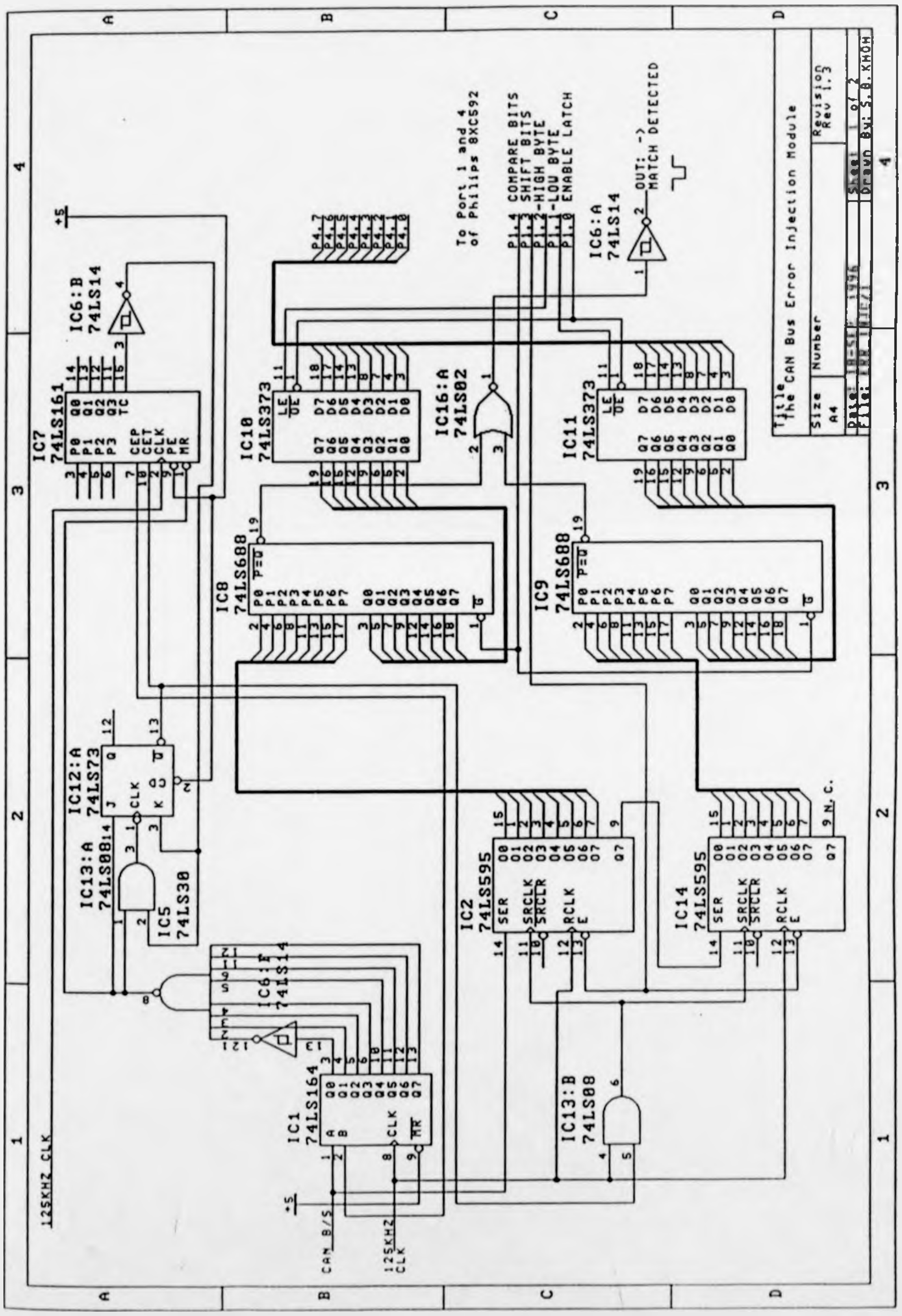
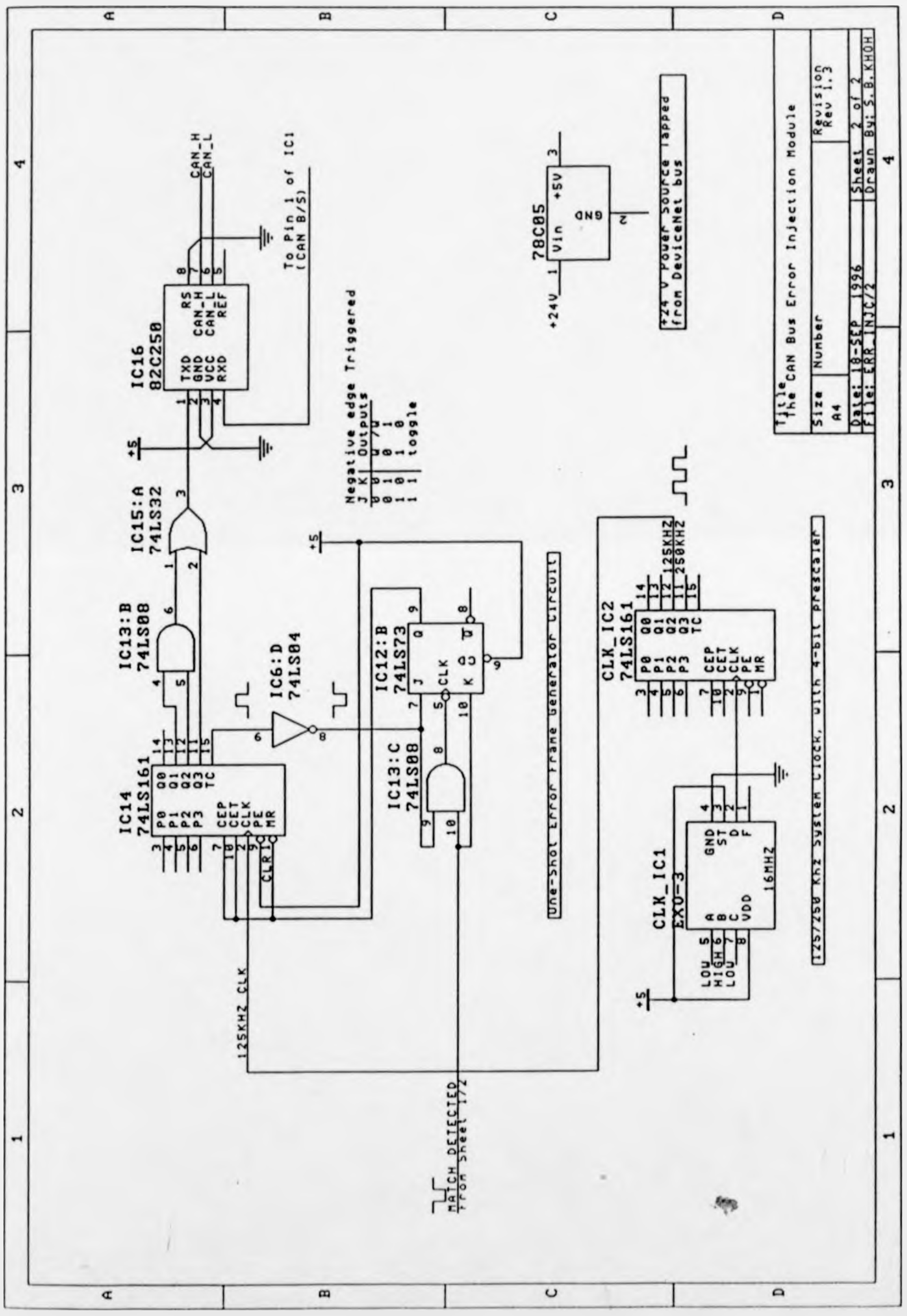
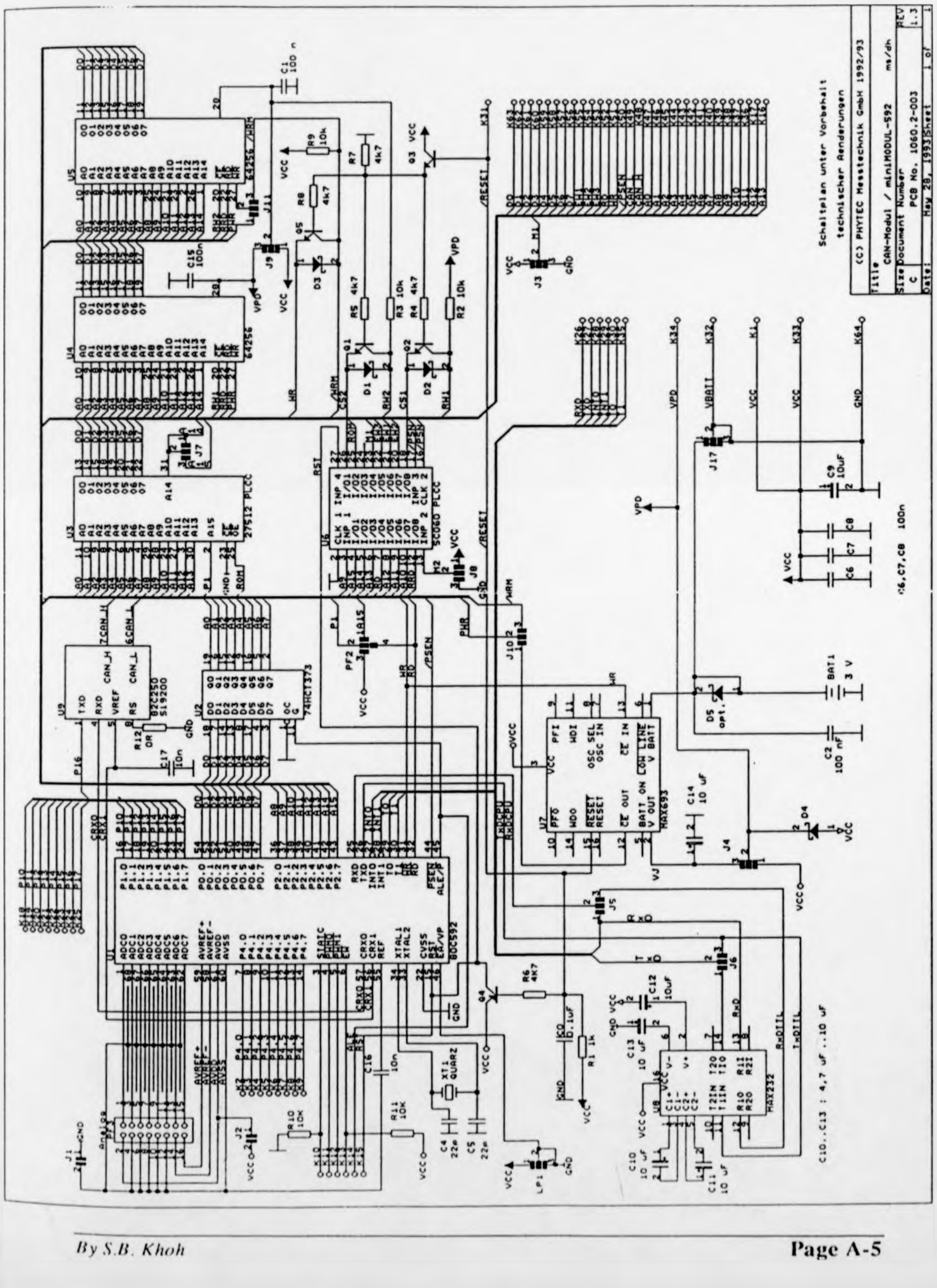


Figure A-3 The underside of the designed prototype



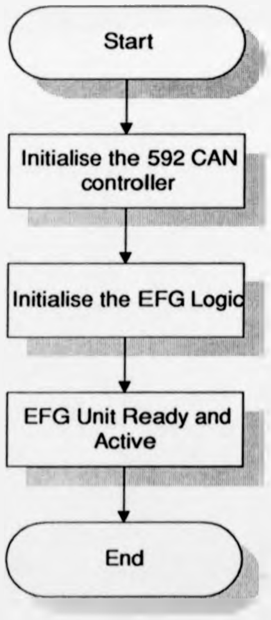
IC1 74LS164
IC2 74LS595
IC5 74LS30
IC6:A 74LS14
IC6:B 74LS14
IC6:F 74LS14
IC7 74LS161
IC8 74LS688
IC9 74LS688
IC10 74LS373
IC11 74LS373
IC12:A 74LS73
IC13:A 74LS08
IC13:B 74LS08
IC14 74LS595
IC16:A 74LS02
125KHZ CLK
CAN B/S
+5
To Port 1 and 4 of Philips 8XC592
P1.4 COMPARE BITS
P1.3 SHIFT BITS
P1.2 ~HIGH BYTE
P1.1 ~LOW BYTE
P1.0 ENABLE LATCH
OUT: -> MATCH DETECTED
N.C.
Title The CAN Bus Error Injection Module
Size A4
Number
Revision Rev 1.3
Date: 18-SEP-1996
Sheet 1 of 2
Drawn By: S.B.KHOH


IC16
82C250
TXD
GND
VCC
RXD
RS
CAN_H
CAN_L
REF
CAN_H
CAN_L
To Pin 1 of IC1
(CAN B/S)
78C05
Vin
+5V
GND
+24V
+24 V Power Source Tapped
from DeviceNet bus
IC15:A
74LS32
IC13:B
74LS08
IC6:D
74LS04
IC12:B
74LS73
IC13:C
74LS08
IC14
74LS161
CLR
Negative edge Triggered
J K Outputs
One-Shot Error Frame Generator Circuit
CLK_IC2
74LS161
125KHZ
250KHZ
CLK_IC1
EXO-3
16MHZ
125/250 KHz System Clock, with 4-bit prescaler
125KHZ CLK
MATCH DETECTED
From Sheet 1/2
Title
The CAN Bus Error Injection Module
Size
A4
Number
Revision
Rev 1.3
Date: 18-SEP-1996
Sheet 2 of 2
File: ERR_INJC/2
Drawn By: S.B.KHOH

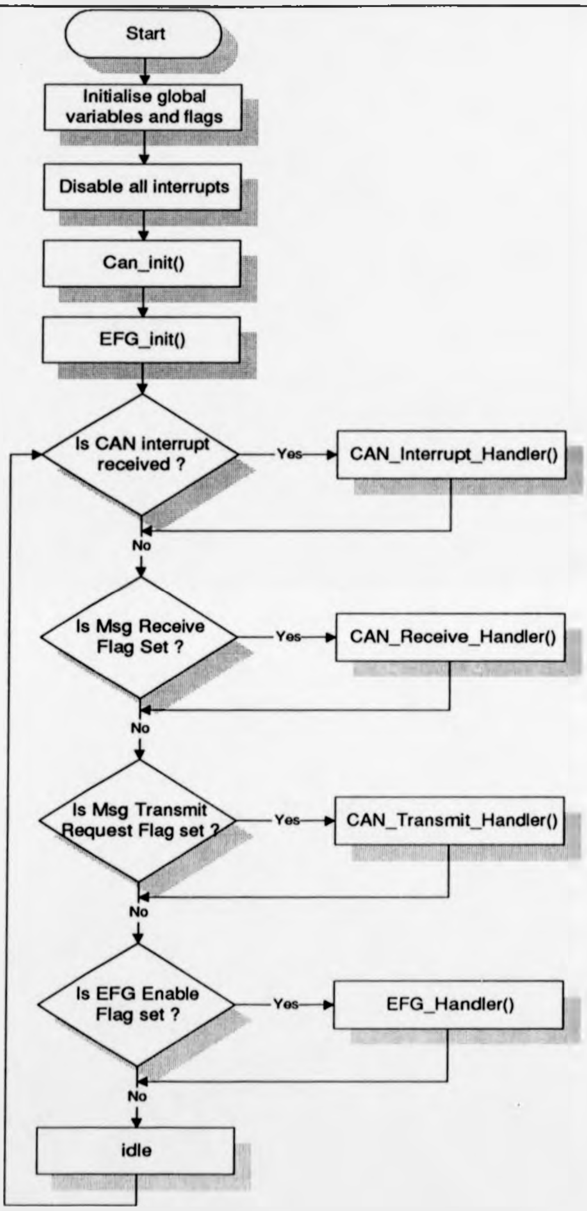

Schaltplan unter Vorbehalt
technischer Aenderungen
(C) PHYTEC Messtechnik GmbH 1992/93
Title
CAN-Modul / miniMODUL-592 ms/dh
Size Document Number REV
C PCB No. 1060.2-003 1.3
Date: May 28, 1993 Sheet 1 of 1
U1 80C592
U2 74HCT373
U3 27512 PLCC
U4 64256
U5 64256
U6 SCO6O PLCC
U7 MAX693
U8 MAX232
U9 82C250 SI9200
XT1 QUARZ
BAT1 3 V
C6,C7,C8 100n
C10..C13 : 4,7 uF ..10 uF

APPENDIX B

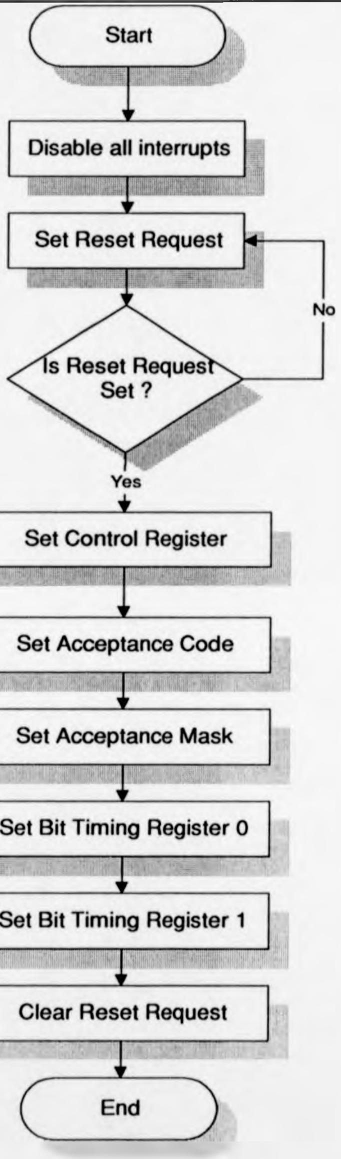
FLOWCHART FOR THE EFG EMBEDDED SOFTWARE



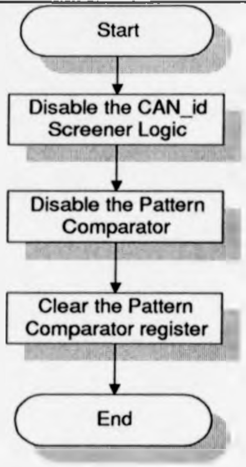
EFG Unit System Flowchart



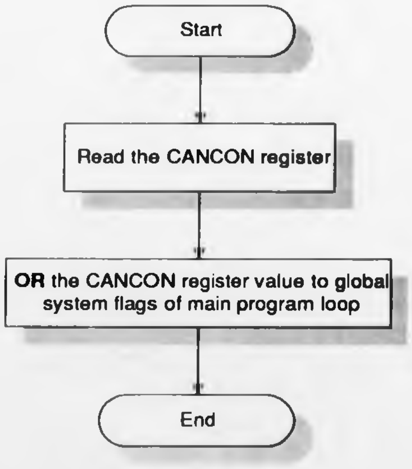
The Main Progam Software Flowchart



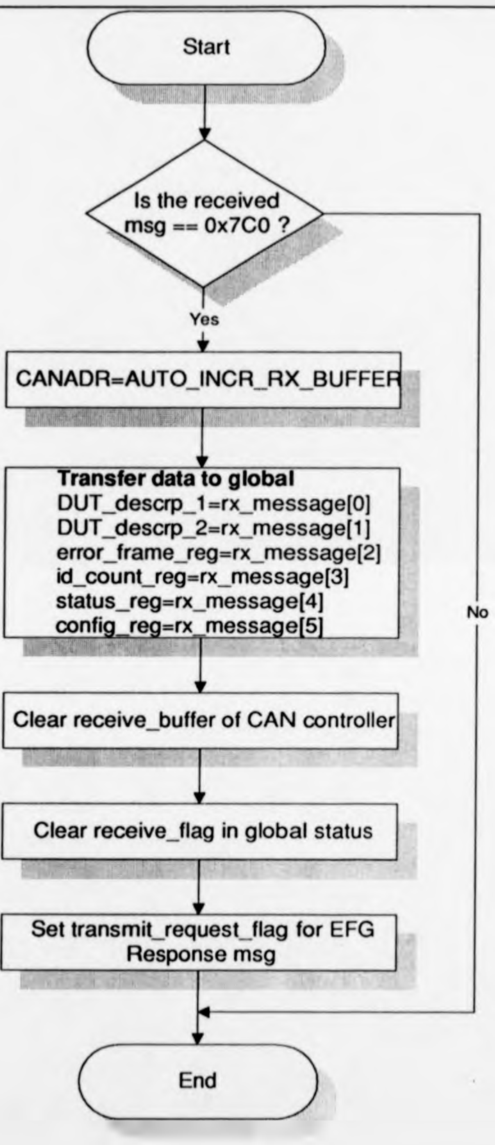
CAN_Init() subroutine
(The 592 CAN controller initialisation)


Start
Disable the CAN_id Screener Logic
Disable the Pattern Comparator
Clear the Pattern Comparator register
End

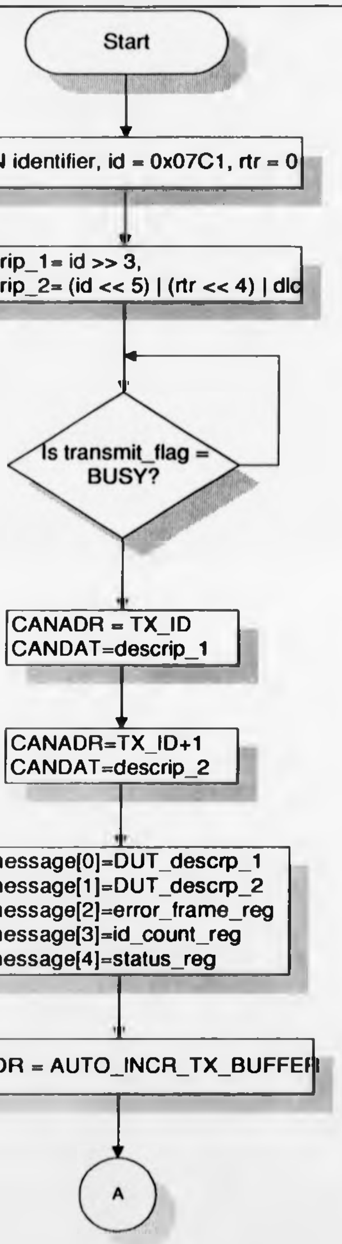
EFG Init() subroutine


Start
Read the CANCON register
OR the CANCON register value to global system flags of main program loop
End

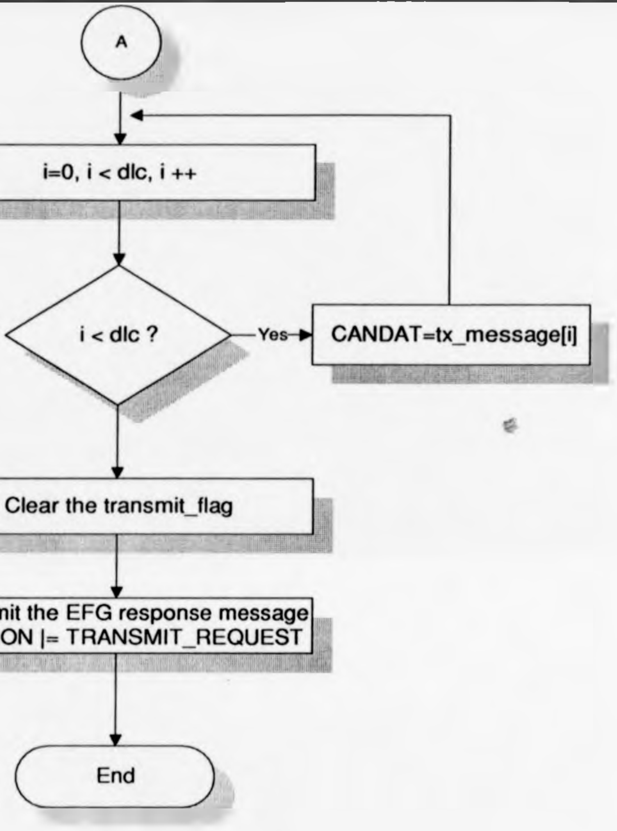
CAN_Interrupt_Handler() subroutine


Start
Is the received msg == 0x7C0 ?
Yes
No
CANADR=AUTO_INCR_RX_BUFFER
Transfer data to global
DUT_descrp_1=rx_message[0]
DUT_descrp_2=rx_message[1]
error_frame_reg=rx_message[2]
id_count_reg=rx_message[3]
status_reg=rx_message[4]
config_reg=rx_message[5]
Clear receive_buffer of CAN controller
Clear receive_flag in global status
Set transmit_request_flag for EFG Response msg
End

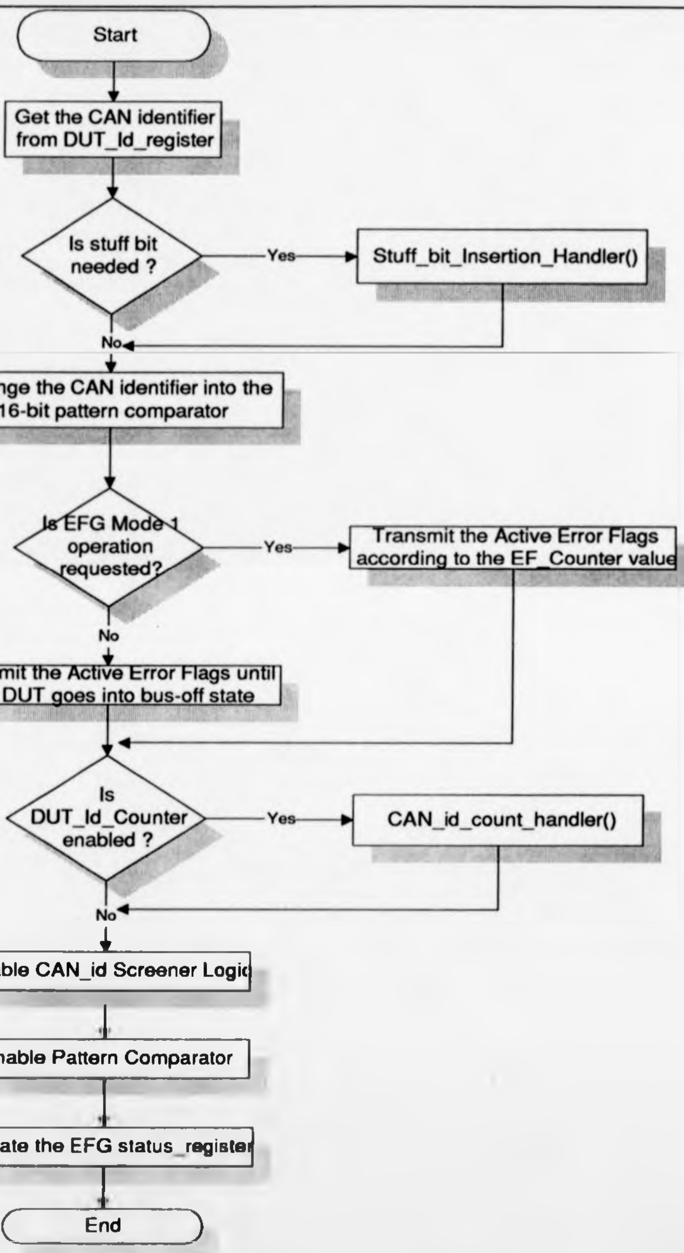
CAN_Receive_Handler() subroutine


Start
CAN identifier, id = 0x07C1, rtr = 0
descrip_1= id >> 3,
descrip_2= (id << 5) | (rtr << 4) | dlc
Is transmit_flag = BUSY?
CANADR = TX_ID
CANDAT=descrip_1
CANADR=TX_ID+1
CANDAT=descrip_2
tx_message[0]=DUT_descrp_1
tx_message[1]=DUT_descrp_2
tx_message[2]=error_frame_reg
tx_message[3]=id_count_reg
tx_message[4]=status_reg
CANADR = AUTO_INCR_TX_BUFFER
A

CAN_Transmit_Handler() subroutine (i)



CAN_Transmit_Handler() subroutine (ii)


Start
Get the CAN identifier from DUT_Id_register
Is stuff bit needed ?
Yes
Stuff_bit_Insertion_Handler()
No
Arrange the CAN identifier into the 16-bit pattern comparator
Is EFG Mode 1 operation requested?
Yes
Transmit the Active Error Flags according to the EF_Counter value
No
Transmit the Active Error Flags until the DUT goes into bus-off state
Is DUT_Id_Counter enabled ?
Yes
CAN_id_count_handler()
No
Enable CAN_id Screener Logic
Enable Pattern Comparator
Update the EFG status_register
End

EFG_Handler() subroutine

APPENDIX C

DEVICENET MESSAGE CONFORMANCE TEST SPECIFICATION

Introduction

The invalid DeviceNet Message Test is intended to verify that a DeviceNet device does not respond to messages other than the DeviceNet messages specified in the DeviceNet Specification Volume I, Release 1.2. Figure C-1 shows that DeviceNet protocol messages is a subset of CAN protocol messages. Therefore, it is important to ensure that DeviceNet devices:-

- only respond and/or react to those DeviceNet Messages specified in the DeviceNet Specification Vol. 1, Release 1.2.
- must not generate, respond and/or react to CAN messages outside the boundary of the DeviceNet specification, i.e. messages that are not-specified in the DeviceNet Specification.

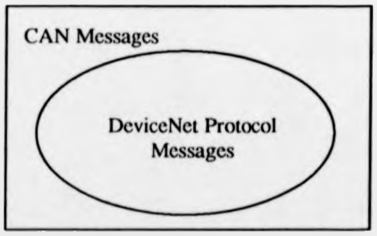


Figure C-1. The relationship between CAN and DeviceNet messages

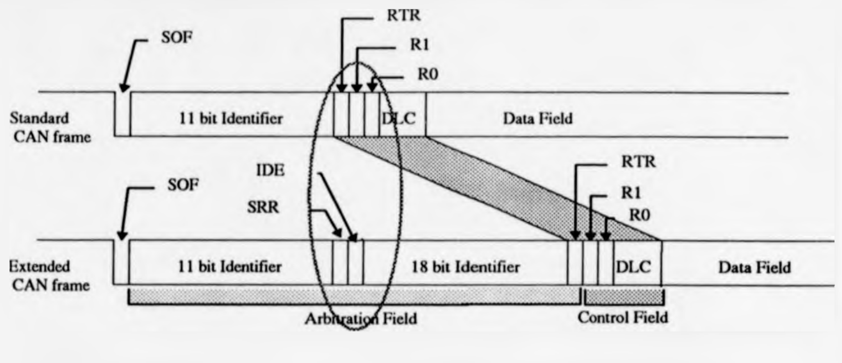
In general, the DeviceNet Message Test can be divided into the following categories.

1. Invalid Syntax DeviceNet Message Test
2. Unused CAN Remote Frame Test
3. Invalid Data-Length DeviceNet Message Test
4. Invalid Message Identifier DeviceNet Message Test
5. Invalid Explicit Message Header Test
6. Reserved/Unused DeviceNet Message Test

C.1 Invalid Syntax DeviceNet Message Test

This section defines the compliance for DeviceNet devices when presented with Valid CAN messages but Invalid Syntax DeviceNet Messages. DeviceNet message is a subset of CAN message as shown in Figure C-1. For example, CAN protocol supports the Remote Transmission Request (RTR) function but Remote Transmission Request is illegal in DeviceNet protocol. All DeviceNet devices, i.e. Client, Server, Peer, Group 2 Client, Group 2 Server, Group 2 Only Client, Group 2 Only Server must undergo this test.

DeviceNet utilises the 11-bit identifier Standard CAN Frame format (i.e. CAN 2.0A compliant device). The extended 29-bit identifier Extended CAN Frame format (i.e. CAN 2.0B compliant device) is prohibited in DeviceNet system. All CAN controllers supporting Extended CAN Frame format must be set to operate in the Standard CAN (CAN 2.0A) mode. The difference between Standard CAN Frame and the Extended CAN Frame lies in the packet header (Arbitration Field) of the CAN PDU as shown in Figure C-2. The Control field, Data field, CRC field, ACK and EOF frame format remain the same for both the Standard and Extended CAN frame.



SOF	: Start of Frame	IDE	: Identifier Extension
RTR	: Remote Transmit Request	DLC	: Data Length Code
SRR	: Substitute Remote Request	R0, R1	: Reserved Bits

Source : CAN Specification 2.0

Figure C-2 The Standard and Extended CAN Frames

Note

CAN 2.0A complaint controller will interpret the CAN 2.0B CAN frame as an error due to the different interpretations of CAN frame format. Note the circled area in Figure 2. Only CAN 2.0B and/or CAN 2.0B passive compliant devices allow CAN 2.0 A and CAN 2.0B messages to co-exist on the same bus.

CAN 2.0A device	Supports Standard CAN frame (11-bit identifier) format
CAN 2.0B device	Supports both Standard (11-bit Identifier) and Extended (29-bit Identifier) CAN frame format.
CAN 2.0B passive device	Supports Standard CAN frame (11-bit identifier) format, but will not interpret the Extended Frame (29-bit Identifier) as an error. The Extended CAN frame will be ignored by

the controller. The controller will acknowledge in the ACK bit of the Extended CAN frame.

C.2 Unused CAN Remote Frame Test

DeviceNet does not use the Remote Frame of CAN protocol. Therefore, all DeviceNet devices must not consume, react or respond to the Remote Frame.

NB: DeviceNet Specification Volume 1, Release 1.2 specifies that only the Standard CAN Frame (11-bit Identifier) is allowed. Implementations using CAN 2.0B compliant controllers must be default to run at CAN 2.0A mode. The compliance test plan assumes that DeviceNet developers have implemented their implementations in the Standard CAN Frame (11-bit Identifier) format.

Functional Description

This test verifies the response

1. when an Open Explicit Messaging Request service with Remote Transmission Request bit set.
2. when a CAN Remote Frame (Invalid DeviceNet message) is received.

Test Procedure

1)

- Send an Open Explicit Messaging (EM) request service. Use the following arguments when creating a connection via the UCMM : Message_Body_Format = 0 , Group_Select = a supported group, Message Id as appropriate for the chosen Group_Select.

- Request a Set_Attribute_Single service, Expected_Packet_Rate, value=0
- Request a Get_Attribute_Single service to Connection Object Class (05hex), Attribute ID = State (01 hex)

Pass : Expected response

Attribute Name (ID)	[Expected values, Responses]
State(01)	[USINT (3), Established]

- Request a Get_Attribute_Single service with RTR bit = 1 to Identity Object Class (01 hex), Attribute ID = Vendor (01 hex)

Pass : The device must **not** respond to and/or consume any of the Open Explicit Messaging request services.

2)

- Send a Standard CAN Remote Frame (i.e. RTR-Remote Transmission Request bit=1) message with CAN identifier = 0 hex, Data-Length = 0.
- The Remote Frame transmission is repeated 8 times, each with Data-Length=1, 2, 3, 4, 5, 6, 7 or 8 bytes
- The CAN identifier is incremented by one after Remote Frame with Data-Length = 8 bytes is transmitted.
- The test is repeated until identifier 0x7EF hex is reached.

Pass : The device must **not** respond to and/or consume any of the CAN Standard Remote Frame messages.

C.3 Invalid Data-Length DeviceNet Message

This section defines a fundamental test to verify that the DUT has implemented Data-Length Code check prior to data consumption. For example, the Open Explicit Messaging Connection Request message is always 4 bytes long. If Open Explicit Message Connection Request message is longer or shorter than 4 bytes, the DUT must not consume any of the message.

Functional Description

This test verifies the response when an Invalid Length Open Explicit Messaging Connection Request service is received.

Test Procedure

- Send an Open Explicit Messaging Connection Request service to DUT with the following arguments:
- Message_Body_Format = 0, Group_Select = a supported group, Message Id as appropriate for the chosen Group_Select
- Set the Data-Length Code (DLC) of the CAN protocol to 4 bytes.
- Request a Set_Attribute_Single service, Expected_Packet_Rate, value = zero
- Request a Get_Attribute_Single service, State

Pass : Expected response.

Attribute Name (ID)	[Expected values, Responses]
State(01)	[USINT (3), Established]

Close the Explicit Messaging Connection

- Send an Open Explicit Messaging Connection Request service to DUT with the following arguments:
- Message_Body_Format = 0, Group_Select = a supported group, Message Id as appropriate for the chosen Group_Select
- Set the Data-Length Code (DLC) of the CAN protocol to **5 bytes**.

Pass : The DUT must **not** respond to and/or consume any of the Open Explicit Messaging request services.

- Send an Open Explicit Messaging Connection Request service to DUT with the following arguments:
- Message_Body_Format = 0, Group_Select = a supported group, Message Id as appropriate for the chosen Group_Select
- Set the Data-Length Code (DLC) of the CAN protocol to **3 bytes**.

Pass : The DUT must **not** respond to and/or consume any of the Open Explicit Messaging request services.

C.4 Invalid Message Identifier DeviceNet Message

This section defines the test to verify that the DUT has implemented boundary check on the Message ID prior to data consumption. For example, the Open Explicit Messaging Connection Request message is always using Group 3 Message ID 6 for UCMM capable device. If Open Explicit Message Connection Request message is other than Message ID 6, the DUT must not consume any of the message.

Functional Description

This test verifies the response when

1. an Open Explicit Messaging Connection Request service with Message ID other than Group 3 Message ID 6 is received.
2. a Group 2 Only Open Explicit Messaging Connection Request service with Message ID other than Group 2 Message ID 6 is received.

Test Procedure

<u>Group 2 Device (UCMM Capable Device)</u>

1)

- Send an Open Explicit Messaging Connection Request service via Message Group 3 to DUT with the following arguments:

Message_Body_Format = 0, Group_Select = a supported group, Message Id as appropriate for the chosen Group_Select, Message ID in the CAN Identifier field = 6.

Pass : Open Explicit Connection Success Response received.

Wait for the connection to time-out.

- Send an Open Explicit Messaging Connection Request service via Message Group 3 to DUT with the following arguments:

Message_Body_Format = 0, Group_Select = a supported group, Message Id as appropriate for the chosen Group_Select, Message ID in the CAN Identifier field = 0.

- Wait for 500 miliseconds

Pass : The DUT must **ignore** the message.

The test is repeated with Message ID in the CAN Identifier field set to 1, 2, 3, 4, 5 and 7

- Wait for 500 miliseconds

Pass : The DUT must **ignore** the message.

Group 2 Only Device (UCMM Incapable Device)

- Send an Open Explicit Messaging Connection Request service via Message Group 2 to DUT with the following arguments:

Message_Body_Format = 0, Group_Select = 1 (Group 2), Message Id as appropriate for the chosen Group_Select, Message ID in the CAN Identifier field = 6 (Group 2 Only Unconnected Explicit Request Message).

Pass : Open Explicit Connection Success Response received.

Wait for the connection to time-out.

- Send an Open Explicit Messaging Connection Request service via Message Group 2 to DUT with the following arguments:

Message_Body_Format = 0, Group_Select = 1 (Group 2), Message Id as appropriate for the chosen Group_Select, Message ID in the CAN Identifier field = 0.

- Wait for 500 miliseconds

Pass : The DUT must **ignore** the message.

The test is repeated with Message ID in the CAN Identifier field set to 1, 2, 3, 4, 5 and 7

- Wait for 500 miliseconds

Pass : The DUT must **ignore** the message.

C.5 Invalid Explicit Message Header Test

This section defines the test for Invalid Explicit Message Header in DeviceNet protocol. The DeviceNet protocol information is defined in the data field of CAN PDU during explicit messaging. Therefore, it is important to check the correct interpretation and implementation of the DeviceNet Explicit Messaging mechanism. The Open Explicit Messaging Connection Request and Response format is shown in Figure 3.0.

Byte Offset	7	6	5	4	3	2	1	0
0	Frag [0]	XID	MAC ID (Destination)[1]					
1	R/R [0]	Service Code [4B]						
2	Reserved (All Bits = 0)				Requested Message Body Format			
3	Group Select				Source Message ID			

Open Explicit Messaging Connection Request
(Message Group 3, Message ID 6)

Byte Offset	7	6	5	4	3	2	1	0
0	Frag [0]	XID	MAC ID (Destination)					
1	R/R [1]	Service Code [4B]						
2	Reserved (All Bits = 0)				Actual Message Body Format			
3	Destination Message ID				Source Message ID			
4	Connection							
5	Instance ID							

Open Explicit Messaging Connection Success Response
(Message Group 3, Message ID 5)

Figure C-3 The Open Explicit Messaging Connection Request/Response Message

The Message Header of the Open Explicit Messaging Connection Request/Response messages is indicated as the shaded area in Figure C-3.

C.5.1 Invalid MAC ID in Message Header - Group 3 Message ID 6

The Open Explicit Messaging Connection Request service requests the establishment of a logical link between two DeviceNet nodes across which Explicit Messages will be transmitted. This test is only applicable to Group 2 devices (UCMM Capable Devices).

Functional Description

This test verifies the response of the DUT to an Invalid MAC ID in the Message Header of Open Explicit Messaging Connection Request Message.

[1] The Destination MAC ID is always specified in the Message Header associated with Open Explicit Messaging Connection Request/Response. pp 4-7 Vol. I

Test Procedure

- Open an Explicit Messaging (EM) connection using Group 3 Message ID 6. Use the following arguments when creating a connection via the UCMM: Message_Body_Format = 0, Group_Select = a supported group, Message ID as appropriate for the chosen Group_Select.

CAN Identifier MAC ID	**Message Header MAC ID**
Source	*Source*

Pass : The DUT must **ignore** the message.

- Open an Explicit Messaging (EM) connection using Group 3 Message ID 6. Use the following arguments when creating a connection via the UCMM: Message_Body_Format = 0, Group_Select = a supported group, Message ID as appropriate for the chosen Group_Select.

CAN Identifier MAC ID	**Message Header MAC ID**
Source	*Destination*

- Request a Set_Attribute_Single, Expected_Packet_Rate, value = zero(0).
- Request a Get_Attribute_Single, State (01)

Pass : Expected response

Attribute Name (ID)	[Expected values, Responses]
State(01)	[USINT (3), Established]

Close the Explicit Message Connection

- Open an Explicit Messaging (EM) connection using Group 3 Message ID 6. Use the following arguments when creating a connection via the UCMM: Message_Body_Format = 0, Group_Select = a supported group, Message ID as appropriate for the chosen Group_Select.

CAN Identifier MAC ID	**Message Header MAC ID**
Source	*Other than Source/Destination*

Pass : The DUT must **ignore** the message.

- Open an Explicit Messaging (EM) connection using Group 3 Message ID 6. Use the following arguments when creating a connection via the UCMM: Message_Body_Format = 0, Group_Select = a supported group, Message ID as appropriate for the chosen Group_Select.

CAN Identifier MAC ID	Message Header MAC ID
Destination	*Source*

Pass : The DUT must **ignore** the message.

- Open an Explicit Messaging (EM) connection using Group 3 Message ID 6. Use the following arguments when creating a connection via the UCMM: Message_Body_Format = 0, Group_Select = a supported group, Message ID as appropriate for the chosen Group_Select.

CAN Identifier MAC ID	Message Header MAC ID
Destination	*Destination*

Pass : The DUT must **ignore** the message.

- Open an Explicit Messaging (EM) connection using Group 3 Message ID 6. Use the following arguments when creating a connection via the UCMM: Message_Body_Format = 0, Group_Select = a supported group, Message ID as appropriate for the chosen Group_Select.

CAN Identifier MAC ID	Message Header MAC ID
Destination	*Other than Source/Destination*

Pass : The DUT must **ignore** the message.

- Open an Explicit Messaging (EM) connection using Group 3 Message ID 6. Use the following arguments when creating a connection via the UCMM:

Message_Body_Format = 0, Group_Select = a supported group, Message ID as appropriate for the chosen Group_Select.

CAN Identifier MAC ID	Message Header MAC ID
Other than Source/Destination	*Source*

Pass : The DUT must **ignore** the message.

- Open an Explicit Messaging (EM) connection using Group 3 Message ID 6. Use the following arguments when creating a connection via the UCMM: Message_Body_Format = 0, Group_Select = a supported group, Message ID as appropriate for the chosen Group_Select.

CAN Identifier MAC ID	Message Header MAC ID
Other than Source/Destination	*Destination*

Pass : The DUT must **ignore** the message.

- Open an Explicit Messaging (EM) connection using Group 3 Message ID 6. Use the following arguments when creating a connection via the UCMM: Message_Body_Format = 0, Group_Select = a supported group, Message ID as appropriate for the chosen Group_Select.

CAN Identifier MAC ID	Message Header MAC ID
Other than Source/Destination	*Other than Source/Destination*

Pass : The DUT must **ignore** the message.

C.5.2 Invalid MAC ID in Message Header - Message Group 2

When an Explicit Message is received, the MAC ID field within the Message Header is examined according to the following rules.

	If	Then
A	The **Destination MAC ID** is specified in the Connection ID (CAN Identifier Field)	The **Source MAC ID** is specified in the MAC ID portion of the Message Header
B	The **Source MAC ID** is specified in the Connection ID (CAN Identifier Field)	The **Destination MAC ID** is specified in the MAC ID portion of the Message Header.

If the Explicit Message fails to satisfy either A or B, then the message is discarded (Vol. I pp. 4-4). This rule only applies to Explicit Connection Messaging using Group 2 Messages.

CAN Identifier Bits											
10	9	8	7	6	5	4	3	2	1	0	
1	0	MAC ID						Group 2 Message ID			**Message Group 2**

Functional Description

This test verifies the response of the DUT to an Invalid MAC ID in the CAN Identifier Field and the Message Header of the Open Explicit Messaging Connection Request Message. The following table defines the Source and Destination MAC IDs for Test 1 through Test 6.

Test	Combination	**CAN Identifier's MAC ID**	**Message Header's MAC ID**
1	1	Source	Source
2	2	Source	Destination
3	3	Source	*Other than Source/Destination*
4	4	Destination	Source
5	5	Destination	Destination
6	6	Destination	*Other than Source/Destination*
7	7	*Other than Source /Destination*	Source
8	8	*Other than Source/Destination*	Destination
9	9	*Other than Source/Destination*	*Other than Source/Destination*

NB : *Other than Source/Destination* refer to the remaining 62 MAC IDs excluded Source and Destination MAC IDs.

Test Procedure

1)

- Open an Explicit Messaging (EM) connection. Use the following arguments when creating a connection via the UCMM:

Message_Body_Format = 0, Group_Select = 1 (Group 2), Message ID as appropriate for the chosen Group_Select.

Combination	**CAN Identifier MAC ID**	**Message Header MAC ID**
1	Source	Source

Pass : The DUT must **ignore** the message.

2)

- Open an Explicit Messaging (EM) connection. Use the following arguments when creating a connection via the UCMM:

Message_Body_Format = 0, Group_Select = 1 (Group 2), Message ID as appropriate for the chosen Group_Select.

Combination	**CAN Identifier MAC ID**	**Message Header MAC ID**
2	Source	Destination

- Request a Set_Attribute_Single, Expected_Packet_Rate, value = zero(0).
- Request a Get_Attribute_Single, State (01)

Pass : Expected response

Attribute Name (ID)	[Expected values, Responses]
State(01)	[USINT (3), Established]

Close Explicit Messaging Connection

3)

- Open an Explicit Messaging (EM) connection. Use the following arguments when creating a connection via the UCMM:

Message_Body_Format = 0, Group_Select = 1 (Group 2), Message ID as appropriate for the chosen Group_Select.

Combination	**CAN Identifier MAC ID**	**Message Header MAC ID**
3	Source	*Other than Source/Destination*

Pass : The DUT must **ignore** the message.

- Repeat the test with 61 other MAC IDs

Pass : The DUT must **ignore** the message.

4)

- Open an Explicit Messaging (EM) connection. Use the following arguments when creating a connection via the UCMM:

Message_Body_Format = 0, Group_Select = 1 (Group 2), Message ID as appropriate for the chosen Group_Select.

Combination	**CAN Identifier MAC ID**	**Message Header MAC ID**
4	Destination	Source

Pass : The DUT must **ignore** the message.

5)

- Open an Explicit Messaging (EM) connection. Use the following arguments when creating a connection via the UCMM:

Message_Body_Format = 0, Group_Select = 1 (Group 2), Message ID as appropriate for the chosen Group_Select.

Combination	**CAN Identifier MAC ID**	**Message Header MAC ID**
5	Destination	Destination

Pass : The DUT must **ignore** the message.

6)

- Open an Explicit Messaging (EM) connection. Use the following arguments when creating a connection via the UCMM:

Message_Body_Format = 0, Group_Select = 1 (Group 2), Message ID as appropriate for the chosen Group_Select.

Combination	**CAN Identifier MAC ID**	**Message Header MAC ID**
6	Destination	*Other than Source/Destination*

Pass : The DUT must **ignore** the message.

- Repeat the test with 61 other MAC IDs

Pass : The DUT must **ignore** the message.

7)

- Open an Explicit Messaging (EM) connection. Use the following arguments when creating a connection via the UCMM:

Message_Body_Format = 0, Group_Select = 1 (Group 2), Message ID as appropriate for the chosen Group_Select.

Combination	**CAN Identifier MAC ID**	**Message Header MAC ID**
7	*Other than Source/Destination*	Source

Pass : The DUT must **ignore** the message.

- Repeat the test with 61 other MAC IDs

Pass : The DUT must **ignore** the message.

8)

- Open an Explicit Messaging (EM) connection. Use the following arguments when creating a connection via the UCMM:

Message_Body_Format = 0, Group_Select = 1 (Group 2), Message ID as appropriate for the chosen Group_Select.

Combination	CAN Identifier MAC ID	Message Header MAC ID
8	*Other than Source/Destination*	Destination

Pass : The DUT must **ignore** the message.

- Repeat the test with 61 other MAC IDs

Pass : The DUT must **ignore** the message.

9)

- Open an Explicit Messaging (EM) connection. Use the following arguments when creating a connection via the UCMM:

Message_Body_Format = 0, Group_Select = 1 (Group 2), Message ID as appropriate for the chosen Group_Select.

Combination	CAN Identifier MAC ID	Message Header MAC ID
9	*Other than Source/Destination*	*Other than Source/Destination*

Pass : The DUT must **ignore** the message.

- Repeat the test with 61 other MAC IDs

Pass : The DUT must **ignore** the message.

C.5.3 Invalid Message_Body_Format in Open Explicit Messaging Connection

During the Open Explicit Messaging Connection request service, the Client will request one of the Message_Body_Formats of DeviceNet. The available Message_Body_Formats are:-

DeviceNet (8/8)	Class = 8 bit USINT	Instance ID = 8 bit USINT
DeviceNet (8/16)	Class = 8 bit USINT	Instance ID = 16 bit UINT
DeviceNet (16/16)	Class = 16 bit UINT	Instance ID =16 bit UINT
DeviceNet (16/8)	Class = 16 bit UINT	Instance ID = 8 bit USINT

The Server will respond with the appropriate Message_Body_Format that it supports in the Open Explicit Messaging Connection response service (Vol. I pp. 4-8). From now onwards, all the Explicit Messaging Connection must be done using the Message_Body_Format specified in the Open Explicit Messaging Connection response service.

Message_Body_Format	Class (bit integer)	Instance (bit integer)
0	8 USINT	8 USINT
1	8 USINT	16 UINT
2	16 UINT	16 UINT
3	16 UINT	8 USINT
4 - 0f hex	Reserved by DeviceNet	

Functional Description

This section verifies the response of the DUT when presented with an invalid message format in the Explicit Messaging Connection.

Test Procedure

- Open an Explicit Messaging (EM) connection. Use the following arguments when creating a connection via the UCMM:

  Message_Body_Format = 0, Group_Select = a supported group, Message ID as appropriate for the chosen Group_Select.

CAN Identifier MAC ID	Message Header MAC ID
Source	Destination

Use the Message_Body_Format value returned by the Server for subsequent transactions

- Request a Set_Attribute_Single, Expected_Packet_Rate, value = zero(0).

- Request a Get_Attribute_Single, State (01)

Pass : Expected response

Attribute Name (ID)	[Expected values, Responses]
State(01)	[USINT (3), Established]

If returned Message_Body_Format, value = 0, then use Message_Body_Format value = 1, 2 and 3 for subsequent test

- Request a Get_Attribute_Single service to Connection Object, Attribute = State (01)

Pass : Expected response

Get_Attribute_Single Request with Message_Body_Format, value=	Expected Get_Attribute_Single Response [Expected values, Responses]
1, 2, 3	[Invalid Attribute Value (0x09), Error_Response Value (94 09 FF)]

If returned Message_Body_Format, value = 1, then use Message_Body _Format value = 0, 2 and 3 for subsequent test

- Request a Get_Attribute_Single service to Connection Object, Attribute = State (01)

Pass : Expected response

Get_Attribute_Single Request with Message_Body_Format, value=	Expected Get_Attribute_Single Response [Expected values, Responses]
0, 2, 3	[Invalid Attribute Value (0x09), Error_Response Value (94 09 FF)]

If returned Message_Body_Format, value = 2, then use Message_Body_Format value = 0, 1 and 3 for subsequent test

- Request a Get_Attribute_Single service to Connection Object, Attribute = State (01)

Pass : Expected response

Get_Attribute_Single Request with Message_Body_Format, value=	Expected Get_Attribute_Single Response [Expected values, Responses]
0, 1, 3	[Invalid Attribute Value (0x09), Error_Response Value (94 09 FF)]

If returned Message_Body_Format, value = 3, then use Message_Body _Format value = 0, 1 and 2 for subsequent test

- Request a Get_Attribute_Single service to Connection Object, Attribute = State (01)

Pass : Expected response

Get_Attribute_Single Request with Message_Body_Format, value=	Expected Get_Attribute_Single Response [Expected values, Responses]
0, 1, 2	[Invalid Attribute Value (0x09), Error_Response Value (94 09 FF)]

C.5.4 Invalid Message Group_Select in Explicit Messaging

The Invalid Message Group_Select test verifies the proper implementation of the Message Group in Explicit Messaging. The test can be divided into two categories, ie. Group 2 device, and Group 2 Only device.

Functional Description

This section verifies the response of

1. the Group 2 (UCMM Capable) DUT when presented with an invalid Message Group_Select in the Explicit Messaging Connection.
2. the Group 2 Only (UCMM Incapable) DUT when presented with an invalid Message Group_Select in the Explicit Messaging Connection.

Test Procedure

1)

<u>Group 2 (UCMM Capable) Device</u>

- Open an Explicit Messaging (EM) connection. Use the following arguments when creating a connection via the UCMM (Message Group 3, Message ID 6):

  Message_Body_Format = 0, Group_Select = as shown in the following table, Message ID as appropriate for the chosen Group_Select.

Pass : Expected response

Message Group_Select	[Expected values, Responses]
0, 3	Success Response
1	Group Select_Resource_Error (94 02 01)
2, 4-f hex	Group_Select_Out_of_Range Error (94 20 01)

2)

<u>Group 2 Only (UCMM Incapable) Device</u>

- Open an Explicit Messaging (EM) connection. Use the following arguments when creating a connection via the Group 2 Only Unconnected Explicit Request Message (Message Group 2, Message ID 6):

  Message_Body_Format = 0, Group_Select = as shown in the following table, Message ID as appropriate for the chosen Group_Select.

Pass : Expected response

Message Group_Select	[Expected values, Responses]
0, 3	Group Select_Resource_Error (94 02 01)
1	Success Response
2, 4-f hex	Group_Select_Out_of_Range Error (94 20 01)

C.6 Reserved/Unused DeviceNet Message Test

The Reserved DeviceNet Message ensures that no DeviceNet devices consume, react/respond to the reserved or unused DeviceNet messages.

C.6.1 Reserved DeviceNet Message Group 4 Test

This section defines the test to ensure that no DeviceNet implementations utilise the reserved message group, ie. Message Group 4 in DeviceNet protocol.

Functional Description

This test verifies the response when a Group 4 Message is received.

CAN Identifier Bits											
10	9	8	7	6	5	4	3	2	1	0	
1	1	1	1	1	Group 4 Message ID						**Message Group 4**

Test Procedure

- Send an Open Explicit Messaging (EM) request service. Use the following arguments when creating a connection via the UCMM : Message_Body_Format = 0 , Group_Select = a supported group, Message Id as appropriate for the chosen Group_Select.
- Request a Set_Attribute_Single service, Expected_Packet_Rate, value=0

- Request a Get_Attribute_Single service to Connection Object Class (05hex), Attribute ID = State (01hex)

Pass : Expected response

Attribute Name (ID)	[Expected values, Responses]
State(01)	[USINT (3), Established]

Close the Explicit Messaging (EM) connection and wait for a Close Explicit Connection Success response.

- Send an Open Explicit Request Messaging Connection using the same arguments as previous test except changing the *Group 3 Message ID 6* to **Group 4 Message ID 6**.

Pass : The DUT must **not** respond to the message.

- Repeat the test by sending an Open Explicit Request Messaging Connection using the same arguments as previous test with a range of Identifiers from **Group 4 Message ID 1 to Group 4 Message ID 0x2f hex.**

Pass : The DUT must **not** respond to the message.

C.6.2 Unused DeviceNet Services Test (Other than Common Services)

The Unused DeviceNet Services Test has been defined in individual DeviceNet object tests. Refer to DeviceNet Conformance Test Specification - Test 500/002/01 for full details.

C.6.3 Unused DeviceNet Object Classes Test

The Unused DeviceNet Services Test has been defined in individual DeviceNet object tests. Refer to DeviceNet Conformance Test Specification - Test 500/002/01 for full details.

C.6.4 Reserved Message_Body_Format Test

The Message_Body_Format in the Explicit Messaging Connection holds the information on how the message body format to be used over the Explicit Messaging Connection. DeviceNet Specification Vol. I, Rel. 1.2 only defines 4 types of Message_Body_Format to be used over the Explicit Messaging Connection. Hence, it is important to ensure that all the Explicit Messaging Connections only utilise the 4 (four) types of Message_Body_Format as specified in the specification (pp. 4-8). This test ensures that no DeviceNet implementation utilises the Reserved Message_Body_Format.

Functional Description

This section defines the response of the DUT when an Open Explicit Messaging Connection Request service with Reserved Message_Body_Format is received.

Test Procedure

- Open an Explicit Messaging (EM) connection. Use the following arguments when creating a connection via the UCMM:
  Message_Body_Format = 0, Group_Select = a supported group, Message ID as appropriate for the chosen Group_Select.

CAN Identifier MAC ID	Message Header MAC ID
Source	Destination

Use the Message_Body_Format value returned by the Server for subsequent transactions

- Request a Set_Attribute_Single, Expected_Packet_Rate, value = zero(0).
- Request a Get_Attribute_Single, State (01)

Pass : Expected response

Attribute Name (ID)	[Expected values, Responses]
State(01)	[USINT (3), Established]

- Request a Get_Attribute_Single service to Connection Object, Attribute = State (01) with Message_Body_Format value = 4 hex.
- The test is repeated until Message_Body_Format value = 0f hex is reached.

Pass : Expected response

Get_Attribute_Single Request with Message_Body_Format, value=	Expected Get_Attribute_Single Response [Expected values, Responses]
4 - 0f hex	[Error Response]

During the establishment of Explicit Messaging Connection, the Client will request one of the Message_Body_Format of DeviceNet. The available Message_Body_Formats are:-

DeviceNet (8/8)	Class = 8 bit USINT	Instance ID = 8 bit USINT
DeviceNet (8/16)	Class = 8 bit USINT	Instance ID = 16 bit UINT
DeviceNet (16/16)	Class = 16 bit UINT	Instance ID =16 bit UINT
DeviceNet (16/8)	Class = 16 bit UINT	Instance ID = 8 bit USINT

The Server will respond with the appropriate Message_Body_Format that it supports in the Open Explicit Messaging Connection response service (Vol. I pp. 4-8). From now onwards, all the Explicit Messaging Connection must be done using the Message_Body_Format specified in the Open Explicit Messaging Connection response service.

C.6.5 Reserved Group_Select Test

The Group_Select field in the Explicit Message Header indicates the Message Group across which messages associated with this connection are to be exchanged. DeviceNet Vol. I, Rel. 1.2 defines only Message Group 1, Message Group 2, Message Group 3 as valid choices for the Group_Select field. Explicit Messages which requests Group_Select other than the above mentioned Groups are illegal.

Functional Description

This section defines the response of the DUT when:-

1). an Open Explicit Messaging Connection Request service with Group_Select=0 is received.

2). an Open Explicit Messaging Connection Request service with Group_Select=1 is received.

3). an Open Explicit Messaging Connection Request service with Group_Select=3 is received.

4). an Open Explicit Messaging Connection Request service with the reserved Group_Select is received.

Test Procedure

1)

- Open an Explicit Messaging (EM) connection. Use the following arguments when creating a connection via the UCMM:

  Message_Body_Format = 0, **Group_Select = 0**, Message ID as appropriate for the chosen Group_Select.

Combination	**CAN Identifier MAC ID**	**Message Header MAC ID**
1	Source	Destination

Pass : Open Explicit Messaging Connection Success Response or Error Response [94 02 01, Resource_Unavailable]

Close the Explicit Messaging Connection

2)

- Open an Explicit Messaging (EM) connection. Use the following arguments when creating a connection via the UCMM:

  Message_Body_Format = 0, **Group_Select = 1** (Group 2), Message ID as appropriate for the chosen Group_Select.

Combination	**CAN Identifier MAC ID**	**Message Header MAC ID**
1	Source	Destination

Pass : Open Explicit Messaging Connection Success Response or Error Response [94 02 01, Resource_Unavailable]

Close the Explicit Messaging Connection

NB: Group 2 Only device must return an Error Response [94 02 01, Resource_Unavailable]

3)

- Open an Explicit Messaging (EM) connection. Use the following arguments when creating a connection via the UCMM:

  Message_Body_Format = 0, **Group_Select = 3**, Message ID as appropriate for the chosen Group_Select.

Combination	**CAN Identifier MAC ID**	**Message Header MAC ID**
1	Source	Destination

Pass : Open Explicit Messaging Connection Success Response or Error Response [94 02 01, Resource_Unavailable]

Close the Explicit Messaging Connection

4)

- Open an Explicit Messaging (EM) connection. Use the following arguments when creating a connection via the UCMM:

  Message_Body_Format = 0, **Group_Select = 2**, Message ID as appropriate for the chosen Group_Select.

Combination	**CAN Identifier MAC ID**	**Message Header MAC ID**
1	Source	Destination

Pass : Expected Error response : [94 02 01, Reserved by DeviceNet]