Architecture for Grid-Enabled Instrumentation in Extreme Environments

Philip Taylor

Dissertation submitted for the Degree of Doctor of Philosophy

School of Earth, Ocean and Planetary Sciences,
Cardiff University, UK

March 2008

CARDIFF UNIVERSITY
PRIFYSGOL CAERDYDD

EPSRC
Engineering and Physical Sciences Research Council

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Abstract

Technological progress in recent decades has led to sensor networks and robotic explorers becoming principal tools for investigation of remote or "hostile" environments where it is difficult, if not impossible for humans to intervene. These situations include deep ocean and space environments where the devices can be subject to extreme pressures, temperatures and radiation levels. It is a costly enterprise to deploy an instrument in such settings and therefore reliable operation and ease of use are requisite features to build into the basic fabric of the machine.

This thesis describes the design and implementation of a modular machine system based on a peer-to-peer, decentralised network topology where the power supply and electronic hardware resources are distributed homogeneously throughout a network of nodes. Embedded within each node is a minimal, low-power single board computer on which a real-time operating system and MicroCANopen protocol stack are operating to realise a standard interface to the network. The network is based on a grid paradigm where nodes act as resource producers and consumers, sharing information so that the machine system as a whole can perform tasks. The resulting architecture supports "plug-and-play" flexibility, to allow users or system developers to reconfigure or expand its capabilities by adding/removing nodes at a later time.

An immediate application of this instrument is in-situ sampling of microbes in extreme aqueous habitats. The microbial sampler is targeted at providing improved sampling capabilities when performing physical, chemical and biological investigations in deep-ocean hydrothermal vent environments. At these depths the instrument is subject to immense pressures of many thousand pounds per square inch, where superheated, corrosive, mineral-loaded vent fluids mix with near-freezing seawater. In the longer term, it is anticipated that this flexible, open interface architecture on which the microbial sampler instrument is based will be applicable more generally to other sectors, including commercial and scientific markets.

In Memory of my sister

Emily Taylor

June 1973 - August 1975

Acknowledgements

The university has provided an inspiring environment in which to meet people from other research disciplines, form new friendships and share ideas, all of which have been of great benefit in catalysing the development of this project. I would like to thank fellow researchers in Cardiff and Oregon State Universities for their help, suggestions and support over the course of this project. I am especially indebted to Gwen Pettigrew for assistance in assembling much of the prototype electronic hardware and fabricating mechanical parts for the motor positioning systems; Oluseyi Odiende for developing firmware code to drive low-level hardware devices and suggesting CAN bus as a network solution for this grid-based instrument; Siarhei Smolau for sharing his knowledge and insights which led numerous, useful contributions to both hardware and firmware development; Daniel Wisdom of Oregon State University for his meticulous work cleaning and assembling the valves; John Daniels at Megatech of Oregon for doing an excellent final assembly of the printed circuit boards; Ian Bell of Maxon for his advice in specifying suitable motors for valve and pump assemblies and saving a substantial amount of money in the project budget; Andew Kalman of Pumpkin Inc for developing the fantastic Salvo RTOS and his ongoing technical support and advice which went beyond the call of duty.

I would like to express my gratitude to my industrial supervisor, Adam Schultz of Oregon State University for securing funding and some splendid opportunities to test and validate the instrument subsystems at sea. Also, to my academic supervisor, Xiangwei Wang at Manufacturing Engineering Centre, School of Engineering, Cardiff University for his friendship and assistance, especially during the period over which the bulk of this dissertation was written up.

Nor must I omit to express special thanks to my lecturers at the School of Electrical Engineering, Colchester Institute, UK especially Pat Thaiss, Roger Wheeler and Richard Matthews for instilling their knowledge, wisdom and pragmatism in many facets of electronics.

Finally, I like to thank my mum for encouraging my early interest in science and my wife; Sam for being such a fantastic mum for Molly and Emily, holding the household

together and a host of practical and essential tasks of life, too numerous to mention and putting up with my moody and anti-social behaviour during the write-up of this dissertation.

This PhD Studentship is sponsored by Engineering & Physical Sciences Research Council (EPSRC) and Earth-Ocean Systems Ltd. Research. Funding for the development of this project has been provided by Natural Environment Research Council (NERC) Marine and Freshwater Microbial Biodiversity (M&FMB), National Aeronautics and Space Administration (NASA) Ames Research Center and European Commission.

Phil Taylor

March, 2008

Contents

Figures

Tables

Listings

Chapter 1 Introduction

1.1 Reaching the Unreachable

Oceans cover over seventy percent of the Earth's surface, containing approximately 97 percent of the planet's water to support a diverse range ecosystems (NOAA 2008) and provide many varied habitats for marine life. In some places they extend down to depths of seven miles, making the ocean floor one of the most difficult and inaccessible frontiers to investigate. For thousands of years our observations have been limited to what could be seen from the sea-surface or the seashore because the immense pressures and total darkness were an insurmountable barrier to exploration of this region. Although over three hundred men and women have journeyed into space and twelve men have walked on the surface of the moon, only two people have descended and returned in a single dive to the deepest parts of the ocean, and they spent less than thirty minutes in a cloud of sediment on the ocean bottom (Baird 2005). And even though the waterless surfaces of Venus, Mars and our moon have been mapped in great detail, the National Oceanic and Atmospheric Administration estimates that well over 95 percent of the ocean floor still remains unexplored (NOAA 2008) and we have barely just begun to penetrate the sub-surface ocean and 'plumbing system' that lies beneath.

Technological advances in recent decades have led to unprecedented access of the ocean depths to reveal a myriad of strange and exotic life along with many other startling and unexpected discoveries, including confirmation of plate tectonics, deposits of methane ice, hydrothermal vent fields and microbes flourishing in sub-seafloor oceans and deep within rocks below. Many of these findings have overturned long established beliefs regarding the fundamental nature of the Earth, in particular our huge underestimation of the depth extent of the biosphere and the variety of energy sources that fuel the chemistry of life. It is now understood that microbes living in and beneath the ocean-floor makeup a substantial part of the Earth's biomass (D'Hondt 2002) where populations flourish several kilometers below the surface. These chemosynthetic organisms live in dark, increasingly hot, oxygen-depleted rocky cracks and caverns, habitats that are completely de-coupled from the sun, deriving their energy from the oxidation of hydrogen sulphide, methane or ammonia, rather than sunlight as photosynthetic plants do on the Earth's surface. Discovering communities of life

adapted to survive in such conditions has greatly strengthened the case for the possible existence of life on other planetary bodies within our solar system, especially Jupiter's moon Europa and, more recently, Saturn's tiny moon Enceladus where speculated analogous aquatic habitats exist.

A major limitation imposed on deep ocean exploration is the sheer cost of gaining access to the depths. Substantial support resources and infrastructure are required to deploy an instrument on the sea floor, meaning that any equipment failures during deployment will be expensive. The more inaccessible, remote and nastier the environment, the greater the price of being there; making ease of use and reliable operation requisite features to build into the basic fabric of the machine. Our explorations are pushing into ever more extreme environments and opening up new frontiers that present a new challenge to our determination, resourcefulness and abilities in science and engineering. Past speculations have often proved too narrow or just wildly inaccurate, indicating that present-day hypothesises as to what else remains to be discovered and its potential scientific or commercial value can only be validated by engaging in continued, and hopefully augmented, exploration efforts in these remote and hostile settings. There is little doubt that the development of sophisticated new tools will play an important role in future investigations to perhaps help answer long sought-after questions on the origins and evolution of life, on Earth and its possible existence on other planets.

1.2 Scope of Project

This dissertation outlines the interface standards, methodologies, and tools leveraged to develop and construct a decentralised machine system targeted specifically for use in extreme environmental conditions. These situations include: high pressure, very high or very low temperature, corrosive environments, the marine environment and even difficult industrial settings such as nuclear reactors or steel plants. Some example applications include: monitoring and telemetering temperature and pressure data sensors within a steel mill smelter or environmental monitoring sensors within a benthic observatory in the deep ocean to observe model factors governing climate change. Here the technology is specifically applied to the construction of a microbial fluid sampling apparatus capable of collecting water samples from in and around hydrothermal vent

fields several thousand meters below the sea surface. At these depths the instrument is subject to immense pressures of many thousand pounds per square inch, where superheated , corrosive, mineral-loaded vent fluids mix with near-freezing seawater.

Partitioning and redundancy methodologies are utilised build a machine system that is made up from a group of individual machines (nodes) that coordinate their activities to work together concurrently to accomplish a given task. The network architecture is based on a peer-to-peer grid paradigm. There is some disagreement on the definition of a grid (Foster 2002) and the field is undergoing change, however for this application the nodes act as resource providers and consumers, transmitting and receiving information (commands, sensor data, etc) between each other over a shared communication network. Additionally the nodes are not subject to centralised control and utilise standard, open, general-purpose protocols and interfaces to facilitate communication. This modular approach leads to much improved flexibility, as the instrument can quickly and easily be reconfigured to satisfy mission-specific requirements by simply adding/removing nodes to/from the network. Additionally, the open architecture encourages seamless integration of third-party nodes into the system framework as and when they become available.

Academic groups and research institutes, such as Ifremer, France are motivated to find strategies and technical solutions that will facilitate cross-compatibility and tool sharing procedures for manned, remote and autonomous vehicles and benthic observatories (MoMARETO EXOCET/D WP5, 2004). Further to this, MBARI, California are investigating methods of simplifying network buses within autonomous underwater vehicles (AUVs) to make the wiring more organised and easier to maintain in an effort to improve machine reliability and ease of use (Yin 2003). Consequently, a substantial part of this research effort attempts to address these ongoing initiatives to develop a standard framework for "smart" decentralised transducer networks. There is presently a huge demand for off-the-shelf network technology with "plug-and-play" support capability where nodes, such as input/output modules, sensors and actuators from different manufacturers are interchangeable with one another (Pfeiffer 2003). In general, instrument manufacturers and end-users are seeking simple interface solutions that will allow them to interconnect their instruments and reconfigure them. Instruments, such as fluorometers (Wetlabs 2008), microbial fluid samplers (Twose et

al. 2000; Behar et al. 2006; Taylor et al. 2006), methane sensors (Capsum 2005), pressure sensors (Seabird 2008), etc are constantly being developed for specialised research investigations. These designs are often insular solutions and not easily integrated into other system frameworks; for example as part of a permanent ocean floor observatory network (Arnaud 2004; Delany 2000) or into an ROV. These instruments exhibit limited flexibility and are not directly interoperable with equipment from other manufacturers with little opportunity to modify or expand their capabilities to accommodate any new mission specific requirements that arise.

To summarise, the features and subsequent new benefits to the end-user of this research project are listed below:

- **Decentralised Modular Topology** – The instrument may be configured with a variable number of nodes. For example, additional fluid sample bottles can be integrated into the framework subject to the requirements of the experiment or payload limitations of the ROV/HOV used for deployment.
- **"Plug-and-play" Functionality** - The end-user can remove or add new nodes to re-configure the instrument to meet specific mission requirements.
- **Open Architecture** – Compatibility facilitates integration of third party nodes such as fluorometers and methane detectors into the machine infrastructure and addition of "bridge" nodes to connect the instrument to the internet or host computer system.
- **Serviceable** - The system is relatively easy to service as repair of the machine is simply a matter of replacing faulty or damaged nodes.

1.3 Thesis Overview

The structure of this dissertation follows a top-down approach, working down from the highest levels of abstraction, so the reader can "drill-down" to reach the desired level of detail. Each chapter addresses a particular level of abstraction, beginning with scientific motivation, then specific application of the technology, followed by the frameworks and methodologies used to specify the architecture, the development of control firmware and support hardware to give a complete description of the machine system. The work is presented in the following chapters:

Chapter 2 Background – Provides some general scientific context and insight into the motivations for undertaking the development of a new microbial sampling instrument based on a decentralised machine architecture. The first section outlines the discovery and significance of microbes. This is followed by a brief history of technological developments in microbial sampling apparatus and a discussion on the latest challenges of sampling in extreme habitats. The chapter concludes, by describing the benefits of applying the methodologies and technology developed herein to construct a new instrument for sampling microbes in aqueous habitats.

Chapter 3 Communication System – The first section of this chapter evaluates possible buses on which to base the network and highlights the strengths and weaknesses of each type and concludes with a justification for choosing Controller Area Network (CAN) bus. The second section is concerned with selecting an appropriate higher layer protocol (HLP) to manage machine system behaviour. Existing CAN HLPs are investigated and their suitability for this application is discussed in detail.

Chapter 4 CAN and CANopen – Describes configuration of the CAN low-level hardware and which features of the CANopen protocol are utilised to support the network. This is followed with a detailed description of how node functionality is specified and implemented through the CANopen Object Dictionary (OD) and Process Data Objects (PDOs) to realise an optimised network solution for the machine system architecture.

Chapter 5 Firmware Architecture – This chapter explains in detail how a minimal CANopen implementation (MicroCANopen) and a custom designed hardware abstraction layer (HAL) are integrated into the real-time operating system framework (Salvo RTOS). Several methodologies are applied to fully specify, design and document the system in an effort to realise a maintainable firmware infrastructure. Top-level system decomposition and partitioning methodologies are used in all aspects of the design in an effort to "design-in" flexibility and robustness into the machine system architecture.

Chapter 6 Electronic Hardware Platform – Details the conceptualisation, design, construction and testing of the electronic hardware and systems. It concludes with a brief summary of its current state.

Chapter 7 Testing and Results – The first section of this chapter discusses the tests performed on the node sub-systems and is followed by a performance assessment of the instrument in sea-trials. The second section then progresses to describe the approach taken to integrate these nodes into a system network to realise a fully decentralised machine architecture and evaluates the results of the tests that followed.

Chapter 8 Conclusion – Evaluates what has been achieved of the machine and outlines possible recommendations for future development of the instrument.

Appendices (on CD ROM) – Containing bibliography, component data sheets, schematic diagrams, instrument power budget, a bill of materials (BOM), firmware reference manual containing a description of function libraries, functions prototypes and call graphs.

Chapter 2 Background

This chapter provides some general scientific context and insight into the motivations for undertaking this research and development project. It begins with a perspective on the discovery and significance of microbes, in particular their adaptation to colonise a diverse range of habitats, subsequent influence over the biosphere and implications for origins of life on Earth and other planets. The next section briefly outlines the history of technological developments in microbial sampling apparatus and discusses the latest challenges of sampling in extreme habitats. It concludes, by describing the benefits of applying the methodologies and technology developed herein to construct a new instrument for sampling microbes in extreme aqueous habitats.

2.1 Why Study Microbes?

2.1.1 Importance

As far as documented history goes back, mankind has sought to understand our place in the scheme of things. Many early attempts to create some kind of order, generally put forward views that humanity were in a unique position at the centre of the universe and held a special status above the rest of animal kingdom. These views were gradually supplanted by a more mature understanding of the natural world, as exponents of philosophy and scientific enquiry developed powerful new methodologies for rational analysis. Looking outwards, astronomy and cosmology established that our planet is in orbit around an average sized star on the outer rim of one of countless galaxies in an unimaginably huge, old and ever expanding universe. Looking closer to home, biology and palaeontology produced great bodies of physiological, morphological, fossil and genetic evidence supporting and demonstrating evolutionary theory - that cellular life began soon after the formation of the Earth and adapted and colonised the planet giving rise to all organisms inhabiting the biosphere.

Until relatively recently, the extent of the biosphere and the diversity of life it contains have been largely underestimated. This was partly because of technological limitations in molecular biology in gene manipulation and the availability of suitable technology for exploration of inaccessible environments. Advances during the last few decades in

both these fields have been instrumental in the development of an improved phylogenetic classification system that puts life into three domains: bacteria, archaea (similar in appearance to bacteria, however with notably different molecular machinery) and eukaryotes (organisms whose cells contain nucleus a and organelles). Prior to this, classical taxonomy approaches based on morphology and biochemical methods divided life into three great "kingdoms" of Plants, Animals and Fungi. In the new taxonomy, however, these kingdoms shrink to fit on one small, trifurcating branch on the phylogenetic tree, showing that life on Earth is overwhelmingly microbial. Each node with descendants represents the most recent common ancestor of the descendants and the edge lengths correspond to time.

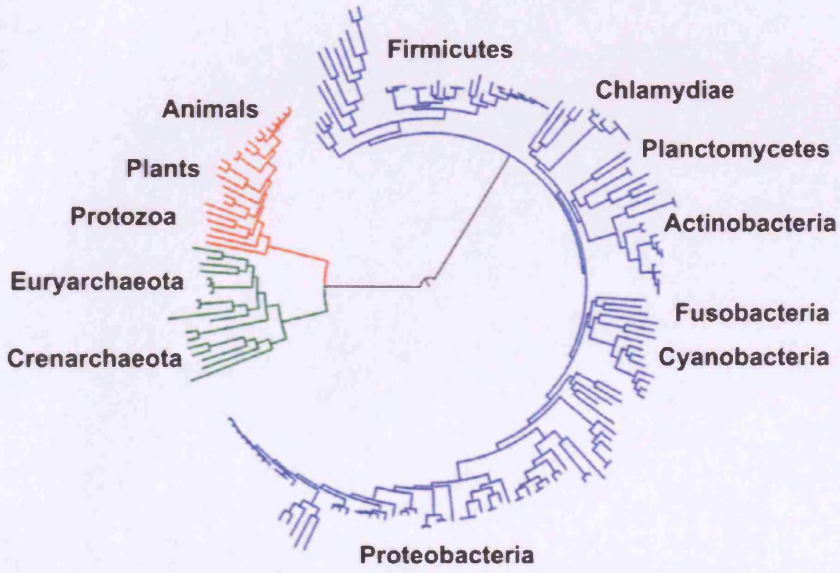


Figure 2.1 Phylogenetic tree showing the speculated common ancestry of all three domains of life. Bacteria are coloured blue, eukaryotes red, and archaea green (iTOL generated tree using methods described by Ciccarelli FD, *Science* 2006).

It is presently estimated that the Earth supports between three and thirty million species of organisms, of which, approximately 1.4 million have been described by science. Microbial life represents a huge untapped reservoir of biodiversity with enormous potential for biotechnological exploitation. It is anticipated that the novel biochemistry these tiny organisms, evolved in response to a wide range of environmental conditions, including extreme temperatures, high pressure, low oxygen and low nutrient environments will find uses in many diverse commercial applications. For example, the replacement of toxic catalysts used in industrial processes or within the pharmaceutical industry and the development clean and efficient new fuel cell technologies for energy storage. There are also strong applied interests in horizontal gene transfer and chemical

signalling systems at the molecular level, and the wide range of symbioses in which microbes are components.

2.1.2 Discovery and Early Research

The existence of microbes was suspected during the late Middle Ages (Avicenna, 1020), however it was the ingenuity of Robert Hooke and Antoni Van Leeuwenhoek in fabricating and using microscopes that allowed the first direct observations to be made. In 1665, Hooke presented the first published depiction of a micro-organism, the microfungus Mucor, which was soon followed by descriptions of microscopic protozoa and bacteria by Leeuwenhoek. Despite these encouraging initial findings, many years passed and it was not until the beginning of the industrial revolution that the necessary resources became available to allow significant progress to be made in the understanding of these tiny creatures. Leeuwenhoek had discovered organisms invisible to the naked eye, however further investigations by Lazzaro Spallanzani and Louis Pasteur would explain the link between microbes and processes such as grapes turning into wine, milk into cheese, or food turning bad.

Spallanzani researched the established belief about the spontaneous generation of cellular life in 1768. His experiment proved that microbes come from the air and that they could be killed through boiling. This work paved the way for further research in 1859 by Louis Pasteur. Pasteur expanded upon Spallanzani's work by exposing boiled (sterilised) nutrient broths to the air, some of which contained a filter to prevent all particles from passing through to the growth medium. These important experiments finally put an end to the long-held misconception that life could spontaneously arise from non-living matter (Aristotle) and also supported germ theory, which postulated that microbes are the cause of many diseases.

In 1876, Robert Koch validated this causal link between microbes and disease when he observed that the blood of cattle that were infected with anthrax always had large numbers of Bacillus anthracis. Koch also found that he could transmit anthrax from one animal to another by taking a small sample of blood from the infected animal and injecting it into a healthy one, causing the healthy animal to become sick. He also found that he could grow the bacteria in a nutrient broth, inject it into a healthy animal, and cause illness. These experiments became known as Koch's postulates and have become

a cornerstone of modern medicine and clinical microbiology, leading to such important innovations as antibiotics and hygienic practices.

2.1.3 Origins, Evolution and Adaptation

Over the course of billions of years microbes adapted to colonise a diverse range of habitats on the planet, mediating a wide range of chemical transformations instrumental to driving global biogeochemical cycles. Over this immense time-span, single-celled plants and animals, bacteria, archaea and other microbes have exerted a powerful influence on the climate, changing composition of the ocean and atmosphere to transform the planet's surface to shape the Earth we see today. These tiny organisms pervade almost every environment on the Earth and are found in the air, on land, in fresh or salt-water environments and also coexist as symbionts within other organisms. For example, there are over a thousand different species of human gut flora residing inside the intestinal tract, contributing to gut immunity, synthesising vitamins such as folic acid, vitamin K, as well as fermenting complex undigestible carbohydrates, whilst at the same time inhibiting the growth of potentially pathogenic bacteria. Other species of microbes are decomposers, breaking down detritus into its constituent nutrients, some of which they utilise and some of which are released into the ecosystem to be recycled. Microbes are also found on plant roots carry out the role of nitrogen fixation, converting nitrogen gas into nitrogenous compounds to provide an easily assimilated form of nitrogen for many plants, which are incapable of fixing nitrogen themselves.

There are many species of microbes that not only survive, but flourish in extremes of pressure, pH and temperature and can even tolerate high radiation levels or high levels of toxicity; habitats that were once considered inhospitable to life as we knew it. Heat-loving microbes were discovered living in hot springs in Yellowstone National Park, Wyoming and later on the Galapagos Rift mid-ocean ridge 2.5 km beneath the ocean surface off the coast of Ecuador. This was the first of many such findings that subsequently uncovered hydrothermal vent fields harbouring microbes that thrive at around 100°C temperatures, in high acidity under hundreds of atmospheres pressure in Pacific, Atlantic and, more recently, Artic ocean ridges (Pedersen et al. 2004). Further to this, core samples from both ocean and continental drilling programs also revealed microbes living deep within rock several kilometers down in the Earth's crust.

There is a growing body of evidence supporting the hypothesis that the early chemistry of life began not in warm oceans, but on mineral surfaces near deep submarine vents in sulphurous, anaerobic, high temperature (near 100°C), high pressure conditions. This idea correlates closely with the discovery of hydrothermal vent ecologies associated with mid-ocean ridges and a recent extension of the iron-sulphur model (Wächtershäuser 2000). This model attempts to explain how polymerisation and complex chemistry arose from simple inorganic matter to form the first protocells inside "black smokers" on the ocean-floor (Martin and Russell 2002). Given that the Earth's surface was undergoing intense meteor bombardment, likely to have melted the crust and vaporised the ocean, the subsurface vents may have provided a relatively stable environment in which life could develop over four billion years ago (Nibet and Sleep 2001).

Moving to the other temperature extreme, there are microbes that inhabit sea ice, riddled with tiny channels and pores filled with high-salinity water, remaining biologically active at temperatures of several tens of degrees below zero (Deming 2007). Finding life in such environments has provided significant impetus to the search for life beyond Earth as the conditions required to sustain life are much wider than previously thought - temperatures ~50±70°C, a relatively small range of elements, liquid water, and an energy source. It is hypothesised that analogous conditions exist in the permafrost beneath the surface of Mars, the ice of Jupiter's moon Europa and almost certainly in other places in the universe.

Finding microbes in extraterrestrial habitats will undoubtedly provide fascinating new insights and perspectives of relevance to the origin and evolution life on Earth. It may be that during the formation of the solar system, where the process of accretion allowed cross coupling between habitats, that life on the inner planets shares a universal common ancestor. In this case, the phylogenetic tree would need to be extended again to accommodate the new relationships. Alternatively, examination of biomarkers and homochirality (molecular left or right-handedness) of extraterrestrial biochemistries could prove that life arose in parallel and completely independently on other planetary bodies in the solar system. Either way, the research will help to establish a more accurate estimate of life in the universe and, no doubt give rise a host of other questions for which we will be driven to seek yet more answers.

2.2 Technological Developments in Microbial Sampling

Early seafarers made some of the first explorations of the seafloor when they devised a number of ingenious methods for measuring, sampling, and viewing the ocean's depths. Viking sailors measured ocean depth and sampled sediments with sounding weights made from of a lead weight with a hollow bottom attached to a line. Once the weight reached the ocean bottom and collected a sample of the seabed, the line was hauled back onboard ship and measured in the distance between a sailor's outstretched arms (a unit that became known as a fathom).

2.2.1 Early Sampling Apparatus

Later, in the 1800's oceanographers made use of ships as platforms for exploration, mapping and sampling of the ocean. The researchers used wire-line soundings to determine depths and collect biological samples. The sounding weights, called Baillie sounding machines, were provided with a tube into which a sample of the seabed was forced when the weight hit the bottom of the ocean. The need to undertake more sophisticated microbial investigations in deep ocean habitats, catalysed the development of various types of device for collecting water samples during the second half of the 20th century.

The study of the ecology, diversity and function of microbes requires the use of molecular DNA sequencing and manipulation to determine the important members of the community and culture-dependent techniques to understand their physiology and functioning. It is therefore essential that any samples obtained are free from cross-contamination by microbes and microbial DNA from locations other than the site of sampling. In an effort to prevent contamination, the first samplers were based on a hydrowire design (e.g., Zobell 1963; Niskin 1962) that used sterile evacuated bottles, bellows-like polyethylene bags or rubber bulbs to contain the fluid sample. Later, much improved designs were constructed that were able to collect samples with reduced cross-contamination (Jannasch and Maddux 1967). These devices mechanically drew samples into sterile syringes away from the hydrowire, into a sterile dialysis bag that was removed from the sterile inlet prior to sampling.

2.2.2 Under Pressure

For sampling in deep ocean environments, particularly in hydrothermal vent fields where changes in hydrostatic pressure could potentially affect microbial viability and growth, numerous designs have been developed to acquire fluid samples without loss of compression. Examples include single sample versions with (Jannasch and Wirsen 1977) and without (Tabor et. al. 1981) sample inlet protection to reduce the potential for contamination. An improved sampling system was constructed that is able to take vent fluid samples and maintain both in-situ temperature and pressure of the sample collected (Malahoff et.al. 2002). Once aboard ship sub-samples can be transferred to multiple incubators without change in either pressure or temperature. Phillips et. al., 2003 also designed a sampler for the capture, temperature monitoring and in-situ incubation of hot smoker fluids under vent conditions.

2.2.3 Beneath Ice

There are presently various ongoing initiatives to develop sampling robots capable of exploring inaccessible environments such as sub-glacial lakes such as Lake Vostok in Antarctica, which lies beneath a 4-kilometer thick ice sheet. For example, NASA are motivated to investigate the nature and origin of the lake, obtain evidence of long-term climatic change associated with it and identify a site suitable for in-situ microbiological exploration. It is estimated that this habitat has remained isolated from the surface biosphere for tens of millions of years and potentially contains many unique species of microbial life. In such a difficult situation, reliable operation of the instrument is essential, but it is equally important that the device can aseptically penetrate the ice sheet without introducing contaminants into the pristine environment, when sampling the water column and sediments (Blake and Price 2002).

It is possible that Lake Vostok will serve as a natural laboratory to aid in gaining valuable insights regarding the ocean posited to exist beneath the thick covering of ice on Jupiter's moon, Europa (e.g., Carsey et. al. 2000; French et. al. 2001). It is conjectured that any life in these environments would have evolved along unique lines to adapt and survive there. The study of such environments is pushing existing technology beyond its limits and NASA Jet Propulsion Laboratory is presently examining the feasibility of developing a "cryobot" vehicle for investigation of such habitats. The cryobot would penetrate the thick icy crust to release an autonomous "hydrobot" probe to explore

subterranean waterways in a search for signs of microbial life there. Exploration of this kind of remote and hostile environment presents a particularly difficult challenge for machine reliability, as there is no opportunity for direct operator control or human intervention should something go wrong.

2.3 A New Instrument for Sampling Microbes

2.3.1 Engineering Overview

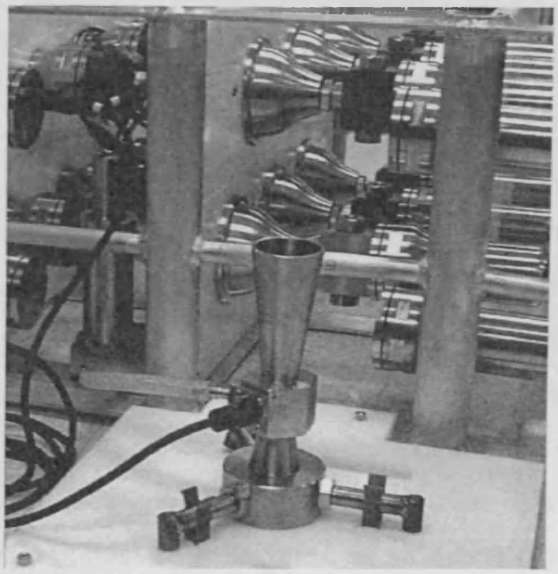


Figure 2.2 Robust, fault-tolerant fluid sampler system capable of gathering data in extreme marine environments.

The main objective of this engineering effort is to develop an appropriate network technology that can be immediately applied to the construction of an improved microbial fluid sampler instrument with several additional new features of benefit to investigations of extreme aqueous habitats. Part of this dissertation involved a comparison and evaluation to access the suitability of current state-of-the-art bus network technologies and protocols on which to base this distributed instrument system. The next phase of the development process was the design and construction of an embedded single board computer (SBC) hardware platform capable of fulfilling the specialised instrumentation needs for this application. These requirements include reliable, micro-power operation and supporting the communication system along with other dedicated hardware essential for measurement and control purposes. A commercial real-time operating system was leveraged and a bespoke hardware abstraction layer written to allow direct access and control of low-level hardware from the upper application layer. The hardware and firmware were developed concurrently from the beginning of the design process to achieve a high level of integration to yield the best possible performance characteristics in terms of power consumption and operational stability. Each node in the instrument is based on a pressure case containing its own internal power supply and dedicated SBC,

so that they can communicate with one another to realise the decentralised fluid sampler system shown above *[figure 2.2]*.

2.3.2 Application

Figure 2.3 Titanium fluid sampling bottle manufactured at MEC, Cardiff University, UK.

The instrument under development will provide improved in-situ physiochemical and microbiological sampling, as well as incubation capabilities to aid the scientific community with investigations into the phylogenetic relationships and physiological diversity between marine microbes. It is anticipated that the system will find wide applicability as a sealed-pressure fluid sampling apparatus, operable to a maximum depth of 6000m in the water column, to acquire samples within a temperature range of 0°C to 200°C. Fluids and aqueous suspensions of light sediments are collected by means of a sampling head connected to a set of 1000ml flow-through bottles *[figure 2.3]* and pump. The bottle manifold/assembly is housed within an aluminium cage designed to sit inside the basket of a manned submersible (HOV) or remotely operated vehicle (ROV). The sampling head can be positioned on a point of fluid outflow using the HOV or ROV robot arm to take one or more discrete fluid samples whilst measuring the various flow properties. A fluid flow-rate sensor is integrated into the head for measuring vertical effluent velocities over a range of 0.1 - 1000mm/s and temperatures to an accuracy of better than 0.5°C with 0.1°C resolution.

The internal environment of the sealed bottles is susceptible to pressure changes in response to changes in ambient or internal temperature and hydrostatic pressure, for example as the instrument returns to the surface. An additional pressure compensator node can maintain samples under isobaric conditions to the environment in which they were acquired to maximise the viability of any microbes present. The bottles then function as bioreactors allowing incubation of microbes and once returned to the laboratory; the instrument permits the introduction of new chemistry and extraction of samples under isobaric conditions for further biochemical analysis. Every effort is made to keep samples free of contamination by utilising chemically non-reactive and

biologically inert materials in the instrument construction. The high-pressure sample bottles, pressure cases and valve bodies are all fabricated from tough titanium alloy (90% Ti, 4% V, 6% Al) and the internal construction of the valves consist of a titanium ball valve and high performance polyaryletheretherketone (PEEK) seal rated to 250°C.

The standardised communication interface will facilitate connection of the instrument directly to an equivalent ROV bus or further integration into a permanent undersea observatory via a "bridge" node. In the first instance, the instrument may be operated interactively (via a serial data link) as a sampling tool from the ship on the surface. The operator has direct control to select any bottle (or combination of bottles) and the pump switched on to acquire a fluid sample. In this mode, the instrument simultaneously captures a continuous real-time stream of physical data representing the flow and temperature conditions of the sample fluids. Secondly, the instrument can be deployed using a winch in a pre-programmed mode, where it may be left to operate autonomously on the ocean floor for an extended period of time (days-to-months) to obtain time series samples or serve as a network-enabled node in a permanent ocean floor observatory system.

2.3.3 Design Advantages

2.3.3.1 Importance of Standardisation

A significant effort has been invested in adopting standard network interfaces and open protocols for inter-node communication. This approach has the advantage of not having to reinvent the wheel, but using existing solutions, set down in established standards, which have already been well considered and validated. Adhering to widely accepted standards provides a stable framework within which continuous development can take place and ensures longer-term prospects for the project. For example, toy Lego bricks made in 1963 still interlock with those recently manufactured in 2008 because all the bricks are precision-machined to the same high tolerance, to eliminate significant variations in thickness and even colour - one of the significant factors that makes Lego such a successful and enduring toy.

Another advantage is the availability of commercial diagnostic and development tools, as well as extensive documentation. These tools aid with design and testing of node

firmware, speeding-up the development phase and help to ensure correct conformance to standards during final integration of nodes into the system framework. Standardisation allows efficient working within the scope of the project, by repeatedly reusing the same solution throughout the design. It also creates the potential for future opportunities to collaborate with any industrial partners and academic establishments utilising the same technologies and standards. To this aim, extensive documentation on the machine system infrastructure is presented here to make it possible to maintain and adapt this technology for wider application *[see Appendices]*.

2.3.3.2 Case for a Decentralised Architecture

Conventional centralised system networks rely on a master-slave model for data exchange between nodes, where a single node acts as the master controller of all slaves on the network *[figure 2.4]* polling them to detect their insertion or removal and

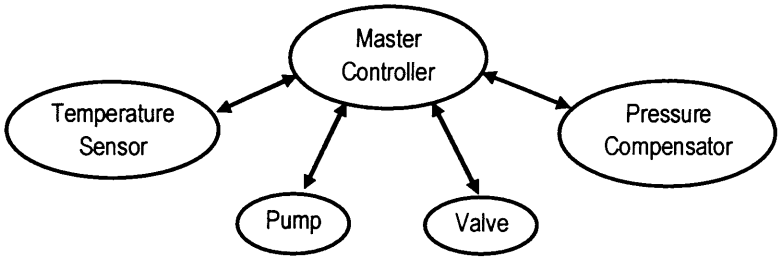


Figure 2.4 Centralised topology.

configuring them. In this scenario, slaves cannot typically communicate by themselves or with each other; only the master may initiate communication with slaves responding when requests are received from the master – although the network is distributed, it is not decentralised. In practice, this means each component has its own line of communication with the main computer, for example within a CTD where strategic placement of a given component is critical. This becomes particularly problematic when attempting to integrate new devices into the network at a later date. Often reorganisation of the devices onboard is required in order to add new components, and this often leads to convoluted cabling, as parts need to be disconnected, moved, and then reconnected. Additionally, because leaks are particularly prevalent around connectors, when parts are disconnected, moved, and reconnected, there is a possibility of making a bad connection between components.

An inherent weakness of the centralised machine (Baker, 2003) approach is that if the master controller node fails then the whole system fails. Failure can be catastrophic, meaning that the machine cannot continue to operate and fulfill its mission. At best, the master is a bottleneck when transfer of information to higher levels is required; at worst, it places the entire system network in jeopardy. In comparison, with a master-less or peer-to-peer communication model, nodes exchange information directly and efficiently without the overhead of routing messages via a master controller *[figure 2.5]*. The widespread availability of low-power, low-cost microprocessors and bus interface

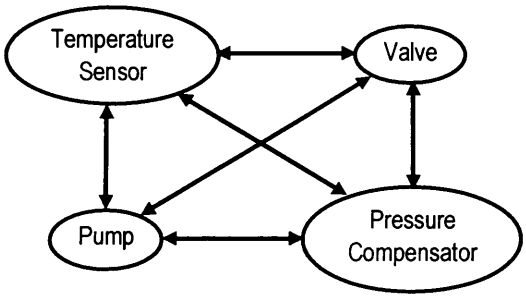


Figure 2.5 Decentralised topology.

controllers has made it possible to embed the necessary intelligence in node to allow this kind of de-centralised control.

Adding distributed intelligence to nodes also yields benefits to application designers and system integrators. The self-identifying capabilities of "smart" transducers and nodes can provide maintenance engineers with immediate feedback from device failures in the network when replacement parts are required. Failure and recovery schemes in distributed networks can be achieved gracefully; for example, if the system detects a faulty node, it can respond by possibly starting a redundant node to replace the lost functionality. The resulting decentralised network topology has several advantages over conventional centralised machine architecture, in that the system is modular, more easily maintained, extensible and more importantly, much more robust, as nodes have the capacity for independent and autonomous operation.

2.3.3.3 Benefits of Modular Design

A modular design containing discrete, partitioned functional units (or nodes) offers significant advantages over an integrated solution where all functions are implemented within one operational unit. Overall reliability is much improved for the simple reason

that it would require malfunction of all nodes within the network to cause total and catastrophic system failure. This scenario is highly improbable and a more realistic situation would be that only one or two node(s) develop faults. These could be intrinsic design faults, human errors during operation or unforeseen external environmental factors. Examples of fault occurrences in instruments whilst performing deep ocean fluid sampling, include failure of o-ring seals because of poor installation or an undetected manufacturing defect in a pressure case causing it to fail under pressure. A modular system there allows for the possibility of confinement of the fault to one subsystem, leaving the reminder to continue functioning and partially fulfill mission requirements or for another redundant system to substitute for the lost functionality.

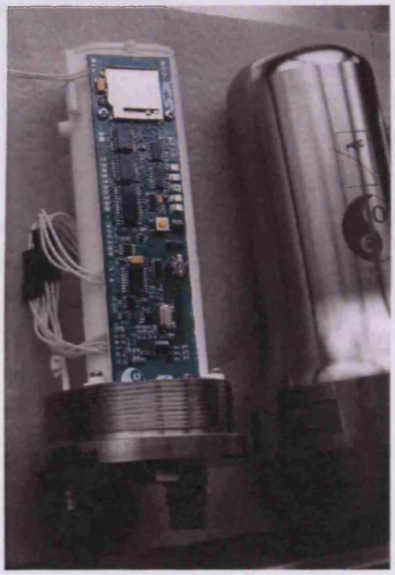


Figure2.6 Node electronics and pressure case housing manufactured at Cardiff University, UK.

Modularising the instrument and standardising the interface between nodes results in much improved flexibility as it opens up the opportunity to support "Plug-and-play" capability. *Figure 2.6* shows a practical implementation of a single node in the system which is based on a pressure case housing that contains its own dedicated single board computer (SBC), power source and communication interface. Users can link these nodes together in Lego-brick fashion to build their own custom-made instrument to satisfy mission-specific application requirements. This allows any number of sample bottles to be used, depending on frame dimension and undersea vehicle payload capacity; typical configuration for NASA, United States is two or three bottles; for NERC, United Kingdom up to six bottles. The simplest configurations could be a stand-alone, one node data-logger *[figure 2.7]* or perhaps a two-node temperature profiler connected to a host PC *[figure 2.8]*. A more sophisticated in-situ fluid sampling instrument would be based on systems containing from six *[figure 2.9]* up to approximately thirty nodes *[figure 2.11]*, depending on the capacity of the undersea vehicle used for deployment. Standard, open protocols and interfaces have been adopted to allow users and developers the opportunities of seamlessly integrating third-party tools into the system at a later date. These tools may include salinity meters,

fluorometers, methane detectors, spectrometers or other tools for geochemical and microbiological analysis of fluids.

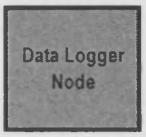


Figure 2.7 Standalone data-logger instrument.

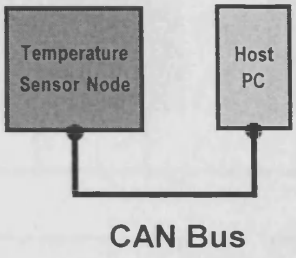


Figure 2.8 Simple two node temperature profiler.

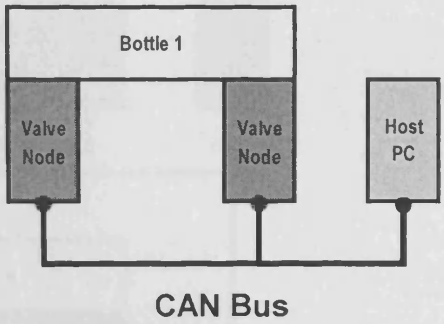


Figure 2.9 Prototype fluid sampling instrument.

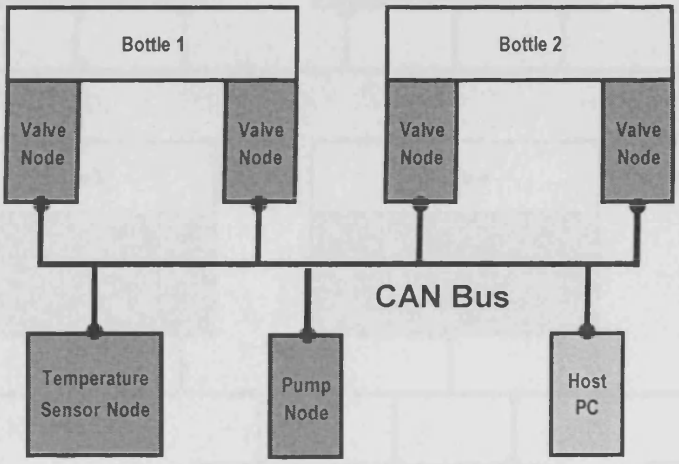


Figure 2.10 NASA fluid sampling instrument.

The examples above show how it is feasible to construct instruments with a variable number of fluid sample bottles, to reduce the instruments' physical size and satisfy payload restrictions of a particular the ROV used for deployment. Additionally, the modular approach provides a high degree of flexibility and allows the end-user the opportunity to modify the experiment. For example, the instrument can be adapted to meet mission-specific requirements by adding third party nodes such as fluorometers and methane detectors into the machine infrastructure and it is even feasible to build new configurations that were not anticipated by the original designer. The system is also relatively easy to service, as repair of the machine is simply a matter of replacing faulty or damaged nodes or use "bridge" nodes to connect the instrument to a host PC running diagnostic software. The ultimate goal is to achieve the simplicity of a Lego set where the functional units (bricks) can be linked together to achieve results quickly and easily without specialist knowledge or specialised tools.

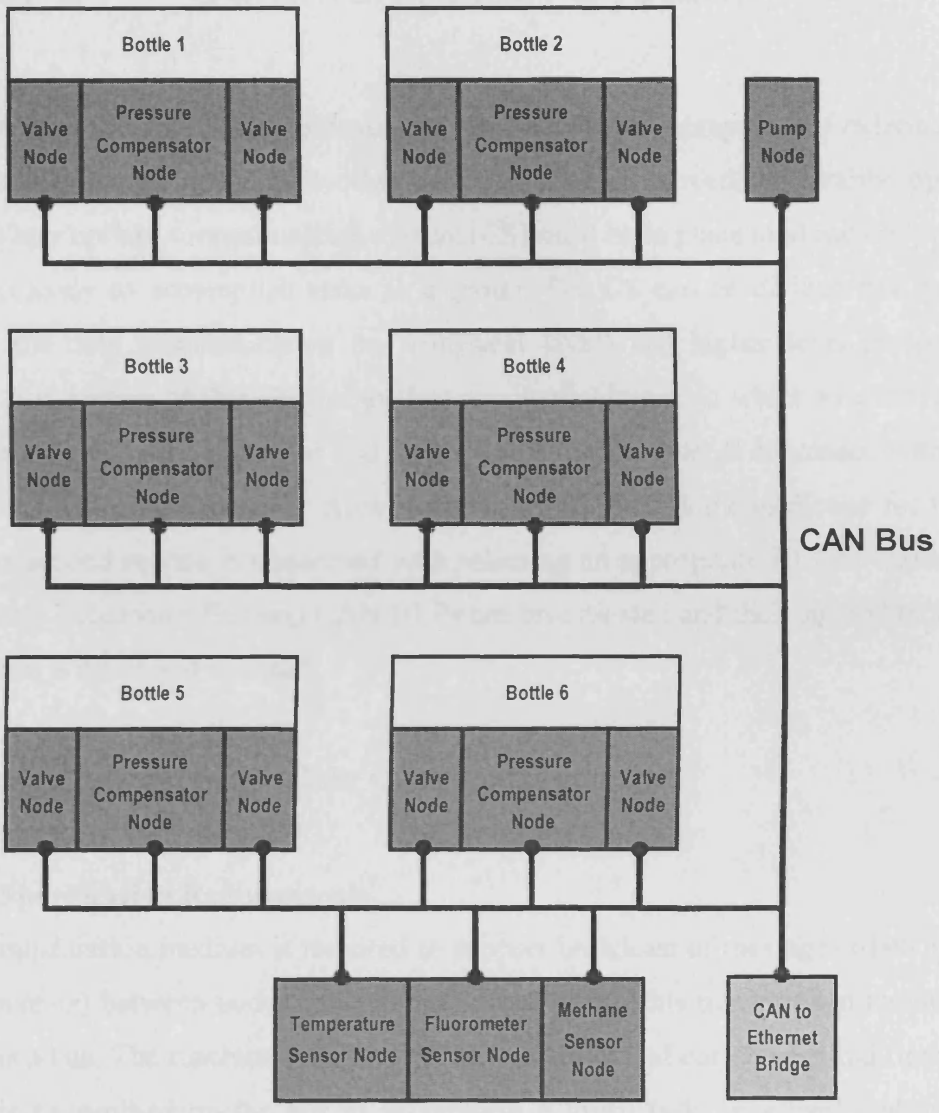


Figure 2.11 NERC fluid sampler with integrated third party sensors.

Chapter 3 Communication System

A key objective of this thesis was to develop a machine system composed of individual nodes that communicate with each other according to an agreed, preferably open standard. An appropriate communication system (CS) must be in place to allow nodes to operate collectively to accomplish tasks as a group. The CS can be divided into two subsystems: the data communication bus (physical layer) and higher level protocol (HLP). The first section of this chapter evaluates potential buses on which to construct the CS and summarises the strengths and weaknesses of each type. It concludes with a justification for selecting Controller Area Network (CAN) bus as the backbone for the network. The second section is concerned with selecting an appropriate HLP to manage machine system behaviour. Existing CAN HLPs are investigated and their suitability for this application is discussed in detail.

3.1 Selecting an Appropriate Data Communication Bus

3.1.1 Bus Specification Requirements

A physical transmission medium is required to support broadcast of messages (data and control commands) between nodes. In computer architecture, this transmission medium is a known as a bus. The machine system depends on the logical correctness and timing of commands transmitted on the bus to accomplish a given task. In a hard real-time system, the completion of an operation after its deadline is considered useless and may even lead to a complete system failure, whereas a soft real-time system can tolerate such lateness. For example, collecting a water sample would require two valve nodes on a flow through bottle to open and a pump node to switch on. A sensor node would then measure fluid flow rate and temperature and after a period for a few minutes the pump would be switched off and the valves closed. This task can tolerate jitter in the order of many milliseconds. On the other hand, tight control loops require low jitter and deterministic latency i.e. hard real-time performance. For example, an active pressure compensator control sub-system made up of a pressure sensor and piston actuator connected to each other over CAN bus to form a closed control loop. It is proposed that the sensor and actuator are housed in within the same node as a self contained unit. This

relaxes hard real-time specification requirements, significantly reduces bus traffic and widens options for leveraging a COTS HLP.

The bus network must satisfy the specification requirements outlined below:

- True peer-to-peer network.
- Support real-time performance.
- Perform reliably in an electrically noisy environment. This system architecture is intended for use in mission and safety critical applications. It must be able to tolerate electromagnetic interference (EMI) noisy supply lines and even hard radiation.
- Perform reliably in mechanically harsh environments. These situations include: high pressure, vacuum, very high/low temperature, operations in corrosive environments, in the marine environment, in difficult industrial settings such as nuclear reactors and steel plants. Also the system should be capable of operating in aerospace environments where it can be exposed to high acceleration, sustained vibration conditions, dust and dirt.
- Micro-power consumption. In the near future it is anticipated that this system will be integrated into permanent seafloor observatories where it can remain dormant for periods of many months waiting on an external trigger event to wake it up. Also the HLP must be efficient in terms of clock cycles required for each byte transfer.
- Fault tolerance. The malfunction of one node should be isolated to the failure of only that node (fault confinement). The rest of the system should ideally be unaffected. In the event of large sections of the system failing then any data collected by the remaining functional part of the system should be recoverable once the system resumes operation.
- Ease of repair. Should a node fail and require replacement, it should simply be a matter of unplugging the node and replacing it with another with little or no system reconfiguration.
- Physically compact, lightweight with minimum cabling and low pin count. Reliability and cost issues with marine connectors.

3.1.2 Brief overview of buses

Buses fall broadly into two main categories as shown in the class diagram below. Parallel buses transport data words striped across multiple wires, whereas serial buses transport data in bit-serial form. At high data rates (above 400MHz) parallel buses become susceptible to signal skew and cross talk. Serial buses are inherently immune to this problem and can consequently be operated at higher data rates in daisy-chain or hub topologies (e.g. USB). Multi-drop connections do not work quite as well at high data rates because of reflections occurring on the bus. The addition of extra power and control connections, differential drivers, and data connections in each direction means that serial buses often have a few more conductors than the I²C (2-wire) serial or Dallas 1-wire buses.

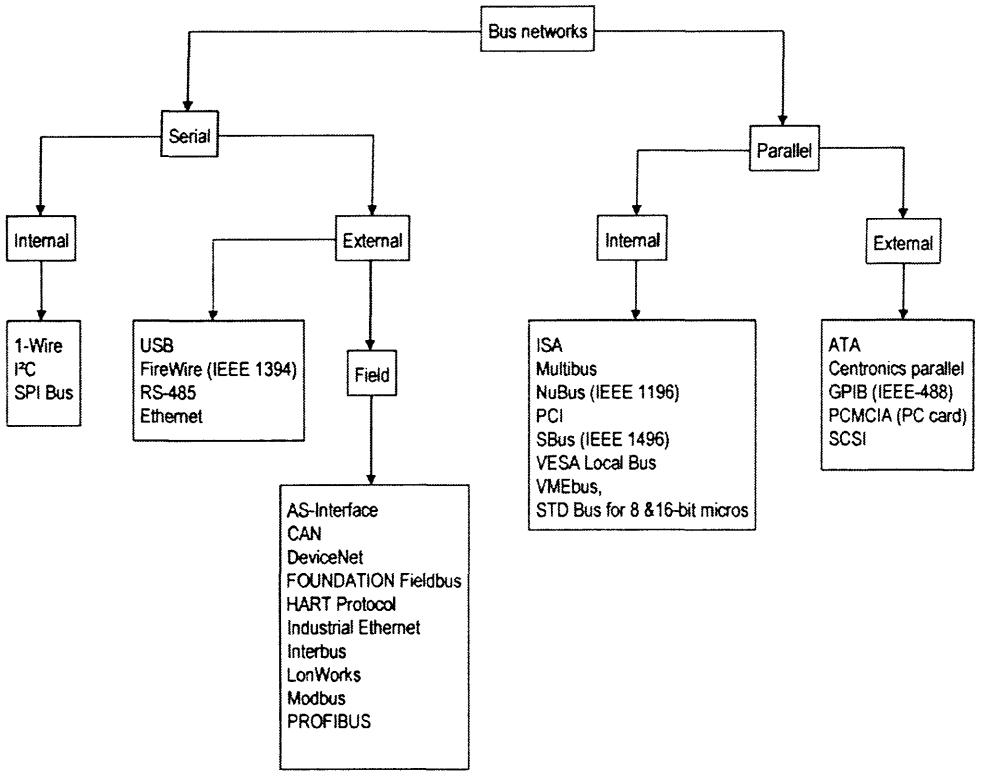


Figure 3.1 Bus categories

Computers utilise both internal and external buses for moving data around. An internal bus connects all the internal peripherals of a computer with the motherboard or components, such as the microprocessor and memory on the motherboard itself. An external bus connects peripherals such as mouse or a printer to a computer and hence to the motherboard inside. The buses considered for this application are now described in further detail in the pages that follow.

3.1.3 RS-485

RS-485 (now EIA-485) was developed to overcome the transmission limitations of single-ended, point-to-point RS-232 communication. At higher data rates, or over long cable runs in electrically noisy environments, single-ended methods were often inadequate. Differential balanced data transmission yields higher signal integrity because it effectively cancels ground shifts and induced noise signals that can appear as common mode voltages on a network.

3.1.3.1 RS-485 Physical Layer

RS-485 satisfies the requirements for a true multi-point communications network, and the standard specifies up to 32 drivers and 32 receivers on a single (2-wire) bus. With the introduction of repeaters and high-impedance drivers/receivers this can be extended to hundreds of nodes on a network. In contrast RS-422 is multi-drop, that is, it does not allow multiple drivers but only multiple receivers. RS-485 therefore enables the configuration of inexpensive local networks and multi-drop communications links. It offers high data transmission speeds (35 Mb/s up to 10m and 100 Kb/s at 1200m).

3.1.3.2 RS-485 Protocol

RS-485 only specifies electrical characteristics of the driver and the receiver. It does not specify or recommend any data protocol. Data-link layer must be implemented by the HLP designer, increasing firmware size, complexity and the development time required. Examples of HLPs that are implemented on RS-485 include SCSI, as well as the following fieldbus standards: ARCNET (Attached Resource Computer Network), Modbus, Profibus (Process Field Bus), European Installation Bus (EIB) and Interbus (Felser and Sauter 2002).

Strengths

- High data transfer rates
- Low pin count

Weaknesses

- Data Link Layer must be implemented in firmware, increasing size and complexity.

- Data collisions are destructive.

3.1.4 Universal Serial Bus (USB)

USB was developed as a low-cost bus solution to allow computer peripherals to be inter-connected without the need to plug expansion cards into the computer's ISA or PCI backplane. Its open architecture facilitates plug-and-play capabilities, allowing peripherals to be connected or disconnected without powering down or rebooting the computer.

3.1.4.1 USB Physical Layer

USB physical layer consists of a four wires; two are power (+5V & GND) with the remaining being a twisted pair for differential balanced data transmission. The bus has a characteristic impedance of around 90Ω and must be terminated with suitable impedance matching resistors on the signal lines. The clock is transmitted, encoded along with the differential data which is encoded non-return-to-zero, inverted (NRZI). Bit stuffing is also used for logic '1' transmission more than five bits long (put logic '0' after five bits of logic '1'). NRZI encoding method does not change the signal for transmission of a logic '1', but the signal level is inverted for each change to a logic'0'.

Three data rates are supported:

- High-speed (USB 2.0) at 480Mb/s, uses a 17.78mA constant current to reduce noise.
- Full-speed (USB 1.1) at 12 Mb/s
- A limited capability, low-speed mode at 1.5Mb/s requiring reduced EMI protection

Devices are physically connected to a host controller using a tiered star topology. Attachment points are provided by a special class of device called a hub which allows branching into a tree structure, subject to a limit of 5 levels of branching per controller. Up to 127 devices may be connected to a single host controller. Although physical connection of devices is tiered star, the host controller communicates with each logical device as if it were connected directly.

3.1.4.2 USB Protocol

The host controller polls the bus in a round-robin fashion and initiates all data transfers. When a device is connected to the bus the host controller assigns it a unique address. Devices may be added or removed from the bus at any time and enumeration of device addresses is an ongoing activity for the system software.

Bus transactions involve the transmission of up to three data packets. A transaction begins when the host controller transmits a 'token' packet describing the type and direction of the transaction, the USB device address and endpoint number. During a transaction, data is transferred to or from the addressed peripheral. The source of the transaction then transmits a data packet or indicates it has no data to transfer and the destination responds with a handshake packet.

The USB data transfer model between a source or destination on the host controller and an endpoint on a device is referred to as a pipe. The pipes are synonymous to byte streams, such as in the pipelines of Unix. The endpoints and pipes are enumerated from 0 to 15 in each direction, so a device can have up to 32 active pipes, 16 in to and 16 out of the host controller. Each endpoint can transfer data in one direction only, either into or out of the device. Endpoint 0 is reserved for bus management in both directions and requires 2 of the 32 endpoints - the USB specification states that all devices must implement endpoint 0. Data is transferred along the pipes in variable length packets (see picture of USB frame). Each pipe has a maximum packet length, typically 2n bytes, so a USB packet will typically contain something in the order of 8, 16, 32, 64, 128, 256, 512 or 1024 bytes.

The pipes are categorised into the following types:

- Control transfers - typically used for short, simple commands to the device, and a status response, used e.g. by the bus control pipe number 0, when a device is connected the host controller needs to learn about it and configure it.
- Isochronous transfers - at some guaranteed speed but with possible data loss. The required bandwidth is reserved for the device with less attention to the success of the transfer (whether or not the whole data arrived on time) since the

traffic included in this transfer type has a high tolerance for errors, e.g. real-time audio or video.

- Interrupt transfers - devices that need guaranteed quick responses (bounded latency), e.g. mouse and keyboards.
- Bulk transfers - large sporadic transfers using all remaining available bandwidth (but with no guarantees on bandwidth or latency). Bandwidth allocated in each transaction of the transfer varies according to the bus resources at the time. Bulk transfers are done in reliable mode - there is great deal of awareness to errors e.g. file transfers.

3.1.4.3 USB Error Detection and Handling

The USB specification assumes that the bit error rate of the USB transmission medium is similar to that of a computer backplane and that any electrical glitches are transient in nature. To provide protection against such transients, token and data packets are protected by CRCs. In token packets the CRC protected region is only 11 bits, so a CRC-5 ($G(x) = x^5 + x^2 + x^0$) provides adequate protection and also aligns the packet to byte boundary. USB data payload packets can be to 1023 bytes in length, so a CRC-16 ($G(x) = x^{16} + x^{15} + x^2 + x^0$) is used to provide reasonable protection of the data payload. Both these CRCs are capable of detecting single and double bit errors.

Errors can be handled in hardware or software. In hardware errors can be reported and failed transfers retransmitted. The host controller will attempt retransmission up to three times before informing the software of the failure. System software can be designed to recover in an implementation-specific way.

Strengths

- High data transfer rates
- Low pin count
- Variable frame size
- Plug-and-play capability
- Limited real-time performance in that the protocol can allocate bandwidth for data transfer in isochronous mode and low latencies.

Weaknesses

- USB is a polled. No node can transfer data on the bus without making an explicit request to the host controller – a centralised topology.
- Protocol is complex to implement.
- Developed as a bus replacement for computer PCI and ISA parallel bus, i.e. house/office environment.

3.1.5 Ethernet

Ethernet is a local-area network (LAN) bus architecture developed by Xerox Corporation in cooperation with DEC and Intel in 1976. It was originally developed as an inexpensive way of moving data quickly between computers connected together in a single room or building. The Ethernet specification served as the basis for the IEEE 802.3 standard, which specifies the physical and lower software layers.

3.1.5.1 Ethernet Physical Layer

The original Ethernet specification required coaxial cable (10Base5 and 10Base2) as the transmission medium, however this has been superceded with more economical twisted pair cables terminated with standard RJ-45 connectors. The network is based on a star topology with a hub or switch in the middle. All nodes attached to a hub share the total bandwidth whilst switches provide each sender and receiver pair with the full bandwidth and are significantly faster than hubs. Fibre-optic cable is also used as a transmission medium (100BaseT and Gigabit Ethernet) and more recently, wireless Ethernet (802.11b) technology became available. The following description of Ethernet protocol refers to a bus based on copper transmission medium.

3.1.5.2 Ethernet Protocol

Ethernet is a frame based protocol. The frame format is as follows:

Field Name	Length (bits)	Description
Destination Address	48	MAC address (not used in Data Link Layer)
Source Address	48	Where the message is going
Type	16	Not used in Data Link Layer
Data Payload	368 – 12000	46 – 1500 bytes
Frame Check Sequence	32	32 bit Cyclic Redundancy Check

Table 3.1 Ethernet frame.

Each frame has a source and destination address, both of which are identified in the frame's header. A frame placed on the bus eventually propagates to all nodes. Ethernet also has limited message broadcast capability in that all frames with a destination of zero are processed by all receiving nodes.

Carrier Sense Multiple Access/Collision Detect (CSMA/CD) scheme is implemented to manage bus arbitration and simultaneous demands on the network. Carrier sense (CS) describes the fact that a transmitter listens for the presence of a carrier wave signal before attempting to transmit a frame onto the bus. If a carrier is sensed, the node waits for the transmission in progress to finish before initiating its own transmission. Multiple access (MA) means that multiple nodes can transmit and receive data on the same bus. CSMA/CD is a layer 2 protocol in the OSI model.

Collision detection (CD) is used to enhance CSMA performance by terminating transmission as soon as a collision is detected. If two nodes transmit a frame simultaneously, an abnormal voltage will be present on the bus. Both nodes detect this collision at the end of a message transmission (Baker 2003) and "back-off" for a random period before retransmitting. Neither node has priority, so the first node to retransmit gains control of the bus. If the messages transmitted by the nodes collide again or collide with a message transmitted by a third node, there will be further delay. Every time a transmission fails, back-off times are increased using a truncated binary exponential back-off algorithm. These delays account for part of the overall system latency and this non-deterministic component in the bus architecture, is impossible to compensate for it at higher software levels. A well managed Ethernet network must be operated well below full capacity, to reduce the chance of such collisions. Even so there will always be an inherent, non-deterministic time delay (latency) component in system communications which has a relationship to the number of nodes on the bus, the back-off algorithm and amount of traffic on the bus at a given time. Since the original colliding messages are destroyed, this situation is called destructive arbitration.

3.1.5.3 Ethernet Error Detection and Handling

A CRC-32 ($G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + x^0$) is added to the end of the frame to provide error detection in the event of

transmission collisions occurring. Any frame with an invalid CRC is discarded by the receiver without further processing. This CRC exhibits very poor error detection properties in terms of Hamming distance per given block size. Error handling at the data link level is limited to detection of bit errors in the physical medium, and the detection and recovery from collisions. Provision of a complete error control mechanism is relegated to higher network layers (Ethernet Specification 1980).

Ethernet only provides "best effort delivery" of frames (Ethernet Specification 1980). This means that the network does not provide any guarantee that data is delivered or that a user achieves a certain quality of service or priority. Features such as recovery of lost or corrupted data are not implemented so that the network can operate more efficiently and make nodes inexpensive. To be more specific, the data link layer protocol does not provide any mechanism to recover corrupted frames which will be lost if they collide with one another or if the bus is subjected to sporadic bursts of electrical noise. It is up to the high-level protocol that is sending data over the network to ensure that frames are correctly received at the destination node.

Strengths

- Very high data throughput
- Variable frame size
- Standardised protocol
- Excellent noise tolerance and immunity.

Weaknesses

- Complex higher layer protocol.
- No real-time performance.
- High power consumption.
- Data collisions are destructive.
- A weak CRC-32 polynomial is used to protect the data payload.

3.1.6 CAN

The demands placed on a bus in a machine control environment are different from office networks, where Ethernet predominates. Certain activities require short,

guaranteed, deterministic latencies e.g. communication between a sensor and actuator node in a control loop. The Ethernet data link protocol provides no support for priority node operation (Ethernet Specification 1980) and it is therefore not possible of use it for such purposes. However, CAN fieldbus was developed specifically as an industrial network system for real-time distributed control and is widely used for this type of application.

CAN bus was originally conceptualised by Robert Bosch GmbH, Germany in 1986 (Bosch 1991) when they were requested to design a robust serial communication system between three electronic control units (ECUs) in vehicles by Mercedes. The CAN protocol became an ISO standard (ISO 11898) for serial data communication and now dominates the European automotive industry with manufacturers in the U.S. beginning to adopt it. CAN bus has also gained widespread popularity in industrial automation, aerospace (Stock 1998), medical equipment and marine instrumentation where it has led to a significant reduction in wiring loom weight and complexity (Fredriksson 1994).

3.1.6.1 CAN Physical Layer

It is possible to implement CAN bus on several different types of physical layers; the most common one is defined by the CAN standard, part ISO 11898-2. This is a balanced (differential) 2-wire interface (Richards 2002) running over either a shielded twisted pair (STP), unshielded twisted pair (UTP). For the physical layer, a twisted pair multi-drop cable is specified with a length ranging from 1,000m at 40Kbps to 40m at 1Mbps. The bus must be terminated with a single resistor at each end, which works as a load for the open-collector transceivers. Also, at higher data rates and longer cable lengths the resistor is required to minimise signal reflections (Richards 2002). Without termination resistors, fast driver edges can cause multiple reflections causing data corruption. Termination resistors also reduce electrical noise sensitivity due to the lower impedance. The value of each termination resistor is equal to the cable impedance (typically, 120Ω for twisted pairs).

When the data transfer rate is kept below 125Kbps, the bus has some degree of fault tolerance, so that signaling can continue even if one bus wire is cut or shorted to ground or the power supply rail. The motivation for this design is that the bus may continue to

operate after a car crash has severed one of the lines. In this mode, noise tolerance is reduced. Each node continues to monitor the faulty line and will resume dual-wire operation if the fault condition is removed. In mission critical, aerospace and satellite applications, fault tolerance and reliability are paramount. Fail-safe operation and system redundancy are achieved with an architecture consisting of a secondary CAN bus connected in parallel with the primary bus (Stock, 1998).

CAN bus has excellent noise tolerance and immunity. Information is carried on the bus as a voltage difference between the two lines. If both lines are at the same voltage, the signal is a recessive bit. If the CAN_H line is higher than the CAN_L line by 0.9V, the signal line is a dominant bit. There's no independent ground reference point for these two lines (Richards, 2002). The bus is therefore immune to any ground noise, which in a vehicle can be considerable. The signals on the two CAN lines will both be subject to the same electromagnetic influences, and so the difference in voltages between the two lines will not vary. Because of this, the bus is also immune to electromagnetic interference.

3.1.6.2 CAN Protocol

CAN communication is based on a carrier sense multiple access/bitwise arbitration (CSMA/BA) scheme. Every bit transmitted on the bus is defined as recessive or dominant (maps to logic '1' or '0'). All nodes can monitor the bus (carrier sense) and transmit at the same time (multiple access). If more than one node is transmitting, the result will carry a dominant bit if at least one node is transmitting a dominant bit. When a node transmits a dominant bit, it will see a dominant bit on the bus. In this case, the node will not know if another was attempting to transmit. If a node transmits a recessive bit, but senses a dominant bit then the node knows that someone else is on the bus.

The smart part of CAN bus arbitration is the first node's decision to "back-off" if another node transmits a dominant bit the first time the first node sends a recessive bit (bitwise arbitration). The identifier is the first part of the message transmitted; by the time the identifier has been sent, all nodes bar one will have backed off. The message identifier is called the arbitration field because it decides which messages get priority.

All nodes transmit a single, dominant bit when starting a message; start of message (SOM) bit. Listening nodes see bus activity and do not attempt to start a transmission until the current packet is complete. So the only possibility for collision is between nodes that simultaneously send an SOM bit. These nodes will remain synchronized for the duration of the packet or until all but one of them backs-off. After the SOM bit, the arbitration field is transmitted. The "winning" node will always be the one with the arbitration field of the highest value, because it's the one that will transmit a dominant bit first, while the other nodes are transmitting recessive bits. The numerical value of the arbitration field can be considered to be the message priority.

This is non-destructive bus arbitration, since the highest priority message is not destroyed. The transmitting node doesn't even know that a collision occurred. The only mechanism for a node to detect a collision is if it senses a different logic level on the bus from what it transmitted. So the successful node and any other listening nodes never see any evidence of a collision on the bus. Non-destructive bus arbitration occurs without corruption or delay of the higher priority message. The highest priority message always gets through, but at the expense of the lower-priority messages.

If a number of nodes clash, one will win out. After that message has completed, all of the "losers" will try again. In this second round, the next highest value arbitration field will win out, and the process will repeat. There's nothing to stop the highest value arbitration field from being transmitted again. This is similar to the situation in a pre-emptive real-time kernel where a high priority task could choose to run continuously and thereby prevent some lower priority tasks from completing their work. In both cases, it would be bad design to lock out lower priorities in this way, but it's important to realise that the CAN bus doesn't prevent this scenario. It is the responsibility of the system designer to ensure that no one message type monopolises the bus.

The arbitration field may be eleven or twenty-nine bits long, depending on whether the standard or extended messaging protocol is used. The first few bits are used for message priority and the remaining bits for identifying the message type. The CAN standard does not associate any meaning with those bits, but the many higher level protocols that sit on top of CAN do define them. For example, the J1939 standard allows one portion of the bits to represent a destination address, since the CAN protocol itself specifies a

source address for all packets, but doesn't mandate a destination address. This is quite reasonable since much of the traffic on an automotive bus consists of broadcasts of measured information, which isn't destined for one specific node.

Like Ethernet, CAN is a message (or frame) based protocol *[see table 3.2]*. Embedded in the CAN frame itself is both the priority and the contents of the data being transmitted. All nodes in the system receive all frames broadcast on the bus and acknowledge if the frame was properly received. Each node in the system determines whether the frame received should be immediately discarded or stored for processing at a later time. A single frame could be destined for one particular node to receive, or many nodes based on the network and system implementation. For example, an automotive airbag sensor can be connected via CAN to a safety system router node only. This router node takes in other safety system information and routes it to all other nodes on the safety system network. Then all the other nodes on the safety system network can receive the latest airbag sensor information from the router at the same time, acknowledge if the frame was received properly, and decide whether to act on this information or discard it.

Field name	Length (bits)	Description
Start-of-frame	1	Denotes the start of frame transmission
Identifier	11	A (unique) identifier for the data
Remote transmission request (RTR)	1	Must be dominant (0)
Identifier extension bit (IDE)	1	1 Must be dominant (0)
Reserved bit (r0)	1	Reserved bit (it must be set to dominant (0), but accepted as either dominant or recessive)
Data length code (DLC)	4	Number of bytes of data (0-8 bytes)
Data Payload	0-64	Data to be transmitted (length dictated by DLC field)
CRC	15	Cyclic redundancy check
CRC delimiter	1	Must be recessive (1)
ACK slot	1	Transmitter sends recessive (1) and any receiver can assert a dominant (0)
ACK delimiter	1	Must be recessive (1)
End-of-frame(EOF)	7	Must be recessive (1)

Table 3.2 CAN frame.

CAN protocol also allows a node to request information from other nodes (Gámiz et al. 2003). This is called a remote transmit request (RTR) where the node transmits a specific request for data to be sent to it. For example, a safety system in a car gets frequent updates from critical sensors like the airbags, but it may not receive frequent updates from other sensors like the oil pressure sensor or the low battery sensor to check they are functioning properly. The safety system can periodically request data from these other sensors to perform a built in self test (BIST). The system designer can utilise this feature to minimise bus traffic while still maintaining network integrity.

One additional benefit of this frame based protocol is that new nodes can be connected to the grid without the necessity to reconfigure other nodes to recognize this addition. This new node will start receiving messages from the grid and, based on the message ID, decide whether to process or discard the received information.

3.1.6.3 Deterministic Latency

Latency for a single message transmission is in the order of 300μs for an Ethernet network operating optimally, i.e. below full capacity. This figure increases non-linearly as the amount of available bandwidth on the bus decreases (Cheshire 1996). On the other hand CAN messages have a predictable maximum latency time which is because it detects message collisions at the beginning of the transmission and systematically resolves them (Baker 2003). A trigger message with no data and the highest priority can have a maximum latency time of 54μs on the bus if 1Mb/s transfer rate is used (Fredriksson 2000).

CANs real-time properties are analogous to those of a pre-emptive real-time kernel. In each case, the objective is to ensure the high priority work gets completed as soon as possible. It's still possible to miss a hard real-time deadline; however a high priority job should never miss its deadline because it was waiting for a low priority task to complete.

Practical investigations of CAN bus have been performed to ascertain its worst-case latency *in-situ* i.e. in a real machine system (Gámiz 2003). These investigations are of value to the system designer in the development phase of a project. Also computer simulations have been made of CAN and Ethernet to compare and evaluate the

performance of Ethernet as an embedded communication network (Hendry 1999) in softer real-time applications. The results predict that an Ethernet networks *average* latency is actually shorter than CAN. This is entirely down to the fact that the data transfer rate of Ethernet is more than a magnitude greater than that of CAN.

3.1.6.4 Fault Tolerance

CAN provides a number of fault tolerance mechanisms. One is the inclusion of a 2-bit acknowledgment field. During the acknowledgment time after each packet is sent, the transmitter sends a recessive bit while any receivers send a dominant bit. The transmitter can thus determine that at least one node has received the packet. This prevents a disconnected node from continuing its transmission, when there is no one listening.

CAN nodes have the ability to determine fault conditions and transition to another operating mode depending on the severity of the problem. They are able to recognise the differences between short-term disturbances and permanent failures and modify their behaviour accordingly. CAN nodes are capable of transitioning from normal mode (transmitting and receiving messages) to shutting down completely based on the severity of the errors detected. This is known as fault confinement and it prevents faulty nodes from monopolising the bandwidth on the network (Pazul 1999). It is therefore possible for the HLP layer to exercise control over the node and prevent it from becoming a "babbling idiot" (Bell 2002).

3.1.6.5 CAN Error Detection and Handling

CAN is high reliability, with the chance of an undetected error being calculated at less than 4.7×10^{-11} (Bosch, 1991), which equates to one error every 1000 years. This is partly because a frame consists of a small data payload (8 data bytes maximum) protected by a strong CRC-15 ($G(x) = x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + x^0$). This guarantees a Hamming bit length of 6 (Kooperman 2004), meaning that up to 5 consecutive corrupted bits can be detected by any node on the bus. In an evaluation of CAN SAE J1939 conducted by the US National Highway Safety Administration, no failures were recorded in 71 million transmissions during a 4000Km trip, despite the bus being purposefully loaded to capacity (Marsh 2000).

Strengths

- Very high reliability.
- Excellent noise tolerance and immunity.
- Reasonably high data throughput.
- Real-time performance – prioritised messages with deterministic latency.
- Low and micro power capability with standby and sleep modes for transceiver and controller hardware.
- CAN controller devices provide an efficient and simple hardware implementation of the physical and data link layer transport scheme.
- Data payload is protected by a strong CRC-15 polynomial.
- CAN is message based. All nodes receive all frame broadcasts.

Weaknesses

- Fixed frame size

3.1.7 Conclusion

USB is ubiquitous with computer peripheral connectivity and was therefore considered desirable for this application because of its pervasiveness, as well as "plug-and play" capabilities. However, the data link layer and protocol were originally developed for the purpose of replacing the ISA and PCI backplane in computers in the home/office environment, i.e. not intended for use in the field. The electrical characteristics of the physical bus and checksum strength reflect this. Additionally, it is a polled bus that exhibits the fundamental weakness of being a centralised topology.

Ethernet, although capable of huge data through-put, does not support any kind of real-time performance and provides no guarantee that data will be reliably transferred between nodes, only best effort of delivery (Intel and Zerox Digital. 1980). Finally, in its 10Base-T configuration it requires a central hub.

Within the spectrum of bus networks, CAN lies somewhere between multi-point RS-485 and Ethernet communication systems. Unlike RS-485 which leaves timing and protocols to the system developer, CAN handles these details in neatly and cleanly in hardware, reducing development time and code complexity. And unlike Ethernet which

requires the huge overhead of a complex, multi-layer protocol such as TCP/IP, the CAN protocol provides a simple transport scheme. There are numerous commercial low-power CAN controller devices available that implement this transport scheme in silicon. They are ideally suited for this type of embedded systems application where processor resources and battery power are at a premium.

The CAN protocol is optimised for systems that need to transfer data reliably between nodes connected on the bus (Stanczyk 2002). The CSMA/BA mechanism ensures that nodes have equal opportunity to gain access to the bus and packet collisions are handled efficiently and is transparent to the upper protocol layers. Efficient, robust message transfer with fault confinement is also a big plus for CAN because faulty nodes will automatically drop off the bus and no single node can bring the network down. This guarantees bandwidth will always be available for critical messages to be transmitted. Since the protocol is message-based, not address based, all nodes on the network receive all messages and acknowledge all messages, regardless of whether in needs the data or not. This allows the CAN to operate in node-to-node or broadcast messaging formats without having to send different types of messages. CAN is an event-driven protocol, which means there are no restrictions imposed on when nodes are allowed to place messages on the bus. It is concluded that CAN satisfies the requirements for a reliable, micro-power, homogeneous, peer-to-peer bus architecture on which to base the communication system.

3.2 Evaluation of Higher Layer Protocols (HLP) for CAN

A node may be connected to another node or even to the outside world via the CAN bus. To effectively manage system complexity a HLP is required to handle data transmission and reception between nodes on the network. The HLP resides on top of the physical and data link layers outlined in the original CAN specification (Bosch 1991).

The purpose of the HLP is best explained by using the following analogy. Functionality provided by CAN is similar to Latin letters in human communication. It is the base for writing a language, but is not enough to enable efficient communication. To specify a language, a stockpile of words, as well as grammar are required to construct sentences.

A system designer may specify their own CAN based language or choose from one of the existing, standardised CAN HLPs (Lennartsson and Fredriksson 2005). Returning to the analogy, the use of a dictionary and grammar is not an effective aid for ordering a drink in a foreign country. For such simple tasks phrase-books are available. They use a sub-set of the language and propose pre-defined sentences for specific situations, e.g. in a restaurant or hospital. In technical communication systems these phrase-books are called profiles.

3.2.1 HLP Requirements Outline

The HLP must fully support a decentralised network with "plug-and-play" capabilities for the nodes. For example, when a node is connected to the system it must have sufficient intelligence to continuously sense and discover the presence of other nodes and react appropriately. The ultimate outcome of this is that the machine system is capable of adapting its behaviour in response to external environmental factors. For example, whilst performing *in-situ* temperature monitoring, there may be a sudden increase in the ambient temperature. If this occurs, it may be useful to increase the sampling rate or perhaps open valves and turn on the pump to acquire a water sample.

Specification requirements for the HLP are shown below:

- Block memory transfers e.g. data file transfer
- Predefined start-up behaviour
- Soft real-time performance
- Plug-and-play capability (dynamically adding/removing nodes to the network)
- Global clock synchronization
- An event scheduler
- Support third-party OEM nodes on the network
- Remote firmware upgrades
- Remote access and control of hardware on the nodes for diagnostics and manual override.

Since the conception of CAN more than twenty years ago a number of standardised HLPs have been introduced for use as transport and application layers for use in

embedded system communication networks. The HLP for this application must satisfy the list of specification requirements outlined above otherwise a proprietary protocol will have to be developed instead. Many system designers still insist on "rolling their own" HLPs (Snowdon 2002; Amer 2002), however this practice should avoided for the following reasons:

- The proprietary protocol must be extensively documented, otherwise it can only be used by the people who created it.
- New team members have no other sources for learning other than the in-house documentation and possibly in-house cross training.
- Incompatibility with HLPs installed on third party OEM modules.
- No third party, off-the-shelf development and test tools are available for the protocol; they must be developed in the house

For embedded networking applications a standardised HLP is preferred as it avoids the pitfalls listed above (Olaf 2003). The benefits of adopting a standard HLP include huge savings in research & development time (Ganssle 1992) and allow the designer to maintain a degree of system compatibility with other equipment manufacturers (OEMs).

3.2.3 OSI Model Limitations

Many of the CAN HLPs are based on the Open Systems Interconnection (OSI) Reference Model. It is important to be aware that the OSI model was originally developed to describe the functionality of communication systems on the basis of a hierarchically layered architecture (Momut 2000). Its main purpose is to connect two nodes (clients) with each other so they can exchange information. The full OSI implementation has fundamental limitations when used for a machine system requiring reliable, real-time performance because it does not specify how long it takes data to propagate through the abstraction layers, resulting in a non-deterministic latency. This means there is no way for the system designer to precisely specify when an event will occur during the design phase (Fredriksson 1994). By its nature the CAN data link layer (layer 2) inherently supports multicasting i.e. all nodes receive all messages. Contrast this to a typical real-world OSI implementation, where TCP/IP is used to connect two clients in a communication network allowing them to share information. In this case,

point-to-point protocol (PPP) acts as a data link layer to establish a direct connection (using an address to specify the destination) between two nodes. Such networks are intended to be accessed by users (unknown at the design phase) during runtime. Not only does OSI ignore CANs broadcast capability it also means that each node requires some knowledge about the other nodes connected to the network to perform its function (Fredriksson 2005). This increases the size and complexity of code embedded on each node.

What follows is a brief discussion and evaluation of CAN HLPs commonly used in harsh environments including space, marine, military and industrial process control applications.

3.2.3 CAL (CAN Application Layer)

CAL was developed in 1993 by Philips Medical Systems and later adopted by the CAN in Automation (CiA) group. The protocol is based on a three-layer model, specifically designed for real-time control applications, and provides an application layer, communication and device profiles that standardise node functionality and system administration. These layers constitute a collapsed form of the seven-layer OSI model mapping onto the physical, data link and application layers (Woolever 1999). The intermediate layers are not needed because a fieldbus network usually consists of a single network segment only (no need for Transport and Network Layer, layer 3 and 4) and has no notion of "sessions" (layer 5) or a need for different internal data "presentation" (layer 6) (Boferenbrood 2005).

Advantages

- Supports transmission of larger data blocks between CAN devices
- Initialise and monitor nodes
- Protocol is well established and developed

Disadvantages

- Heavy OSI based protocol

3.2.4 CANopen

CANopen was developed in an Esprit project under the guidance of Bosch, and in 1995 the specification was handed over to the CAN in Automation (CiA) international users' and manufacturers' group. The communication profile is based on the CAL protocol (Boferenbrood, 2005) and finds use in medical equipment, off-road vehicles, maritime electronics, public transportation, building automation, etc.

CANopen specifies an application layer, communication and device profiles. It also provides communication objects for real-time data (Process Data Objects, PDO), configuration data (Service Data Objects, SDO), and special functions (time stamp, sync message, and emergency message) as well as network management data (boot-up message, NMT message, and error Control). These abstractions free the system designer from dealing with low level CAN details such as bit-timing and implementation-specific functions to focus on interchangeability of nodes and their integration into the network.

A full CANopen implementation requires substantial processor resources (40KB-60KB of code and 600 or more bytes of RAM). A minimal implementation is commercially available, known as MicroCANopen, which requires as little as 4KB of code and about 170 bytes of RAM. MicroCANopen is well suited for minimal CANopen slaves that are pre-configured and do not need to be re-configured during operation. A regular CANopen network would expect the presence of a CANopen network management (NMT) master to actually start and monitor the nodes. In deeply embedded applications where all nodes are pre-configured and know what they need to do, a master might not be required. MicroCANopen protocol assumes that no master is present and that all nodes startup automatically after power-up. CAN baud rate, the node ID and all PDO parameters are defined and hard-coded into the module. This lends itself well to this application and a COTS (commercial off the shelf) solution is appealing to facilitate system development and cross-compatibility.

Advantages

- Same advantages as CAL.
- Upon startup, each node transmits a boot-up message and continues to regularly transmit a heartbeat message in the specified heartbeat time interval.

- Nodes can be pre-configured and therefore no NMT is required during the start-up phase.
- MicroCANopen COTS implementation, tools, and protocol stacks are available.
- Widely adopted, which means it is possible to add COTS OEM nodes to the machine system.
- No royalties on deployed products.
- Documentation and support.

Disadvantages

- A full CANopen stack implementation and would require a more powerful microprocessor.

3.2.5 SDS (Smart Distributed System)

SDS was developed by Honeywell as a bus system to facilitate connection of intelligent sensors and actuators in industrial environments. These transducer nodes feature advanced device-level functions, system and device diagnostics.

Like CAL and CANopen, SDS is based on a minimal, three-layer OSI reference model implementation.

Advantages

- Small and effective way to connect small devices to a master controller.
- Master has 100% control of all nodes.
- Well established and widely used.

Disadvantages

- No support for communication between modules without a master PLC (a centralised topology).
- Limited to 64 nodes and a maximum of 126 addresses.

3.2.6 CAN Kingdom

The first version of CAN Kingdom was released by Kvaser in 1991 and the standard is currently being revised (V4.0). The protocol was developed for machine control purposes requiring hard real-time performance and high safety demands e.g. industrial robots, mobile hydraulics, power switchgears, etc.

Can Kingdom separates the system level as far as possible from the node level (abstraction). This is shortens development time, because it allows nodes to be developed in parallel by different teams while the system design is still in a very early stage (Fredriksson 1995). Nodes do not require prior knowledge about the network; however, a NMT master (the "King" in Can Kingdom terminology) must be present during configuration phase. During this phase, the "King" node, which contains complete knowledge about the system, coordinates all node activities. The "King" may not be involved in runtime communication between working applications in different nodes and so it is possible to remove it once the network configuration phase is complete.

CAN Kingdom, DeviceNet and SDS HLPs are discussed in further detail in "CAN HLP Brief Comparison" (Lernartson and Fredriksson 2005).

Advantages

- The system designer has full access and control of all nodes via the "King" node.
- The nodes serve the network. No system knowledge is required within any single node.
- Supports "plug-and-play" capabilities in a safe manner.
- Supports up larger data transfers.
- It is possible to pre-configure nodes and therefore the "King" is not required during the start-up phase.
- Number of nodes in a system is only limited by the CAN hardware.
- Supports a global clock.
- Full utilisation of the priorities in the CAN protocol for hard real-time control.

- Firmware implementation has a relatively small RAM and ROM footprint (typically 500-1500 byte code and 24-48 byte RAM), when compared to other HLPs.
- Nodes designed for some other higher layer protocols can be integrated into a CAN Kingdom network e.g. SDS and DeviceNet
- Open standard.

3.2.7 CANaerospace

The CANaerospace was developed by Stock Flight Systems (Stock 1998) in Germany and later standardised by NASA as a next generation general aviation bus within the Advanced General Aviation Experiments (Agate) program in 2001.

The protocol/data format definition is extremely light and designed for highly reliable communication between microcomputer-based systems in airborne applications. Its purpose is to create a standard for applications requiring efficient data flow monitoring and easy time-frame synchronisation within redundant systems. The definition is kept open to allow implementation of user-defined message types and protocols.

CANaerospace has found use in several interesting applications, including the Sunrise Telescope, a UV telescope for solar observations that circles Antarctica at 35 to 40Km altitude, hanging from a stratospheric balloon. Also in the Stratospheric Observatory For Infrared Astronomy (SOFIA), a NASA/DLR program for high altitude infrared astronomy. It is used in Eurocopter all-weather rescue helicopter and Eurofighter to network system flight state sensors and cockpit instruments and in Airbus A380 as test system.

Advantages

- Designed for flight or mission critical applications.
- Presently installed in several aircraft since 1998 and has demonstrated excellent reliability in a harsh environment.

Disadvantages

- A substantial part of the protocol definition is only relevant to airborne applications

3.2.8 CAN-SU (CAN for Spaceflight Usage)

CAN for Spaceflight Usage (CAN-SU) is a relatively simple higher layer protocol developed by Surrey Satellite Technology Limited (SSTL) in England. CAN-SU forces peer-to-peer addressing and is optimised for telemetry, telecommand and buffer transfer. It is designed to be scaleable, capable of large volume telemetry and to facilitate repeat build of spacecraft sub-systems for different missions (Woodroffe 2004).

Nodes are connected with a primary and redundant CAN bus architecture. On power up, a relay in each node switches to communicate on the primary bus. If a node does not receive a CAN message after five minutes, it assumes bus failure and switches to the redundant bus.

CAN-SU has been tested in LEO (low earth orbit) where it is exposed to radiation levels in the order of 1Krads. SSTL is pushing this technology forward to develop a latch-up immune, SEU (single event upset) tolerant, 100Krad bus architecture (RadCAN) for use in GEO (geo-stationary earth orbit). A 300Krad tolerant CAN controller device is also manufactured by the Atmel Corporation (AT7908E ATMEL CAN Controller for Space Application 4268B-AERO-10/04, 2004).

A substantial engineering effort has been invested in CAN to create a bus that is capable of tolerating high radiation levels. This is an essential requirement for any system used for control and monitoring of nuclear power plants or space applications, such as satellites. In such environments radiation levels are far above the typical 0.1 rad/year dose the surface of Earth is exposed and would cause permanent damage to unprotected electronic components. To survive these conditions, electronic systems must be radiation hardened using techniques such as silicon oxide or sapphire insulated substrates, wide band gap substrates, shielding, error correction, redundancy and watchdog timers.

Advantages

- Forces peer-to-peer addressing by placing the "From ID" in the CAN data field. This allows the receiving node to know where to acknowledge the request.
- Incorporates buffer transfer scheme to speed up larger data transfers.
- Features a redundant bus.
- Proven track record in extreme environments.

Disadvantages

- Redundant bus increases hardware size and complexity. Also, relays, being electro-mechanical devices are prone to failure and consume more power.

3.2.9 NMEA 2000

NMEA 2000 was developed by National Marine Electronics Association Committee (NMEA) in 1990 as data communication standard for ship-board electronic devices. The NMEA Standards Committee Working Group 2000 decided to use the CAN protocol as data link layer, and high-speed transceivers according to ISO 11898-2 as physical layer. The chosen higher-layer protocol is based on J1939 and ISO 11783. Some marine-specific additions were defined within the NMEA 2000 communication and application profile specification.

Disadvantages

- HLP is a closed standard.
- Only utilises a small part of CANs functionality.

3.2.10 Conclusion

Since the conception of CAN bus, over twenty years ago, there has been a requirement for an open HLP that would enable any node to be seamlessly integrated into a system network. This is reflected by the fact that CAL, J1939, NMEA2000, MilCAN and SDS protocols are based on either CANopen or DeviceNet, providing profiles and command sets for a specific application. Today, the CiA are working to further extend the CANopen standard in order to enable the use of TTCAN hardware within CANopen (an OSI based HLP) system networks. Kvaser took a more direct approach to avoid the

mismatch caused when imposing the OSI architecture on top of the CAN data link layer (implemented in hardware) by developing CAN Kingdom.

Both CANopen and Can Kingdom satisfy system requirements for CAN bus real-time performance for this application. Although CANopen is a widely adopted open standard, its large protocol stack demands a more powerful microprocessor than is otherwise necessary. However, the MicroCANopen stack (a minimal CANopen implementation) is targeted for use on microprocessors with limited RAM and ROM resources. MicroCANopen also constrains the system designer to pre-configure nodes at compile time with the additional benefit of improved system reliability as no NMT master in required during initialisation and operational phases of the machine system. It is also possible to pre-configure nodes in this way with CAN Kingdom and avoid the use of a "King" node.

Having evaluated the available options, the decision was made to utilise MicroCANopen as an HLP for machine system control. The protocol is light, well suited for this application and CANopen has been widely adopted in the science and engineering community (Etschberger 2008; Boferenbrood 2000; Yin 2003). There is substantial support in terms of documentation and development tools available (Embedded Systems Academy, Inc. 2008; Vector Informatik GmbH. 2008; National Instruments 2008). This provides a significant advantage over CAN Kingdom, as networking compatibility is maintained with more widely available third-party nodes that adopt this more commonly used and open standard. Finally, MicroCANopen also allows for possible future upgrade to full CANopen implementation without any changes to the communication channels.

Chapter 4 CAN and CANopen

With an appropriate bus network technology and higher layer protocol definition in place, it is possible move forward and develop a communication system based on a grid paradigm. This chapter gives a detailed explanation of how the low-level CAN controller hardware is configured and the measures taken to ensure it operates to maintain a safe, reliable and transparent communication link between nodes in the system network. It then goes on to describe which aspects of MicroCANopen, a minimal implementation of CANopen protocol, are leveraged in this deeply embedded application. The purpose is to demonstrate the feasibility of building a machine system where functionality is distributed evenly and homogeneously across many low-performance microprocessors on the network. This eliminates the fundamental weakness of a central controller unit, which is a single point of failure.

4.1 CAN Hardware Configuration

4.1.1 Bus Length

Configuring CAN controller hardware is not an arbitrary process. The final performance reliability of the system network is limited by component tolerances that affect bit timing. For example, the maximum allowable bus length will be reduced if oscillators of low accuracy or poor stability are used. Conversely, if maximum bus length is desired, the oscillator tolerances must be minimised. Data rates must also be considered because this third variable determines maximum bus length and maximum allowable oscillator tolerances.

Chapter 3 outlined how the CAN protocol is based on a non-destructive bitwise arbitration scheme so that multiple nodes are able to arbitrate for control of the network. This means it is necessary for all the nodes to detect and sample the bits within the same bit time. The relationship between propagation delay and oscillator tolerance can limit both the data rate and the bus length (Richards 2001). *Table 4.1* shows a selection of commonly accepted bus lengths versus data rates.

Bit Rate (Kb/s)	Bus Length (m)
1000	30
500	100
250	250
125	500
62.5	1000

Table 4.1 Data rate verses bus length.

The anticipated data traffic on the bus network for this control application is light and the nodes are located in close physical proximity to one another as they are mounted in a frame system that can be deployed by an ROV or manned submersible. This relaxes the network specification requirements somewhat, so that a nominal bit rate of 125 Kb/s and a total bus length of less than 3 m are considered more than adequate for this application.

4.1.2 Oscillator Tolerance

The bit timing for each node in the system network is derived from a surface mount ceramic resonator (f_{osc} = 16 MHz). In this situation, non-ideal resonator performance causes phase shifting and consequently drift will occur between nodes. The initial frequency accuracy of the resonator has to be considered as well as its long term and temperature stability to ensure it satisfied the CAN specification. The specification stipulates a worst-case oscillator tolerance of 1.58% (Bosch 1991) which must be conformed to for consistent, reliable communication between nodes. The manufacturer of this resonator (AVX Corp.) specifies an initial frequency accuracy tolerance of ±0.5% at a temperature of 25°C (Elliot 1995). It therefore follows that the maximum possible combined percentage deviation between two different resonators on the network will be a total of 1.0%. In addition to the frequency tolerance temperature stability quoted as being ±0.3% (deviation is a maximum of 0.6% between two individual resonators) over a temperature range of -20°C to +80°C. These two figures are added to yield the maximum theoretical deviation from the initial oscillation frequency,

$$\begin{aligned} Oscillator\ tolerance &= frequency\ accuracy + temperature\ stability \\ &= 0.5 + 0.3 = 0.8\% \end{aligned} \tag{4.1}$$

Also,

$$f_{oscMin} = 16\ MHz - (16\ MHz \cdot 0.8\ /\ 100) = 15.872\ MHz, \tag{4.2}$$

And,

$$f_{oscMax} = 16\ MHz + (16\ MHz \cdot 0.8\ /\ 100) = 16.128\ MHz, \quad (4.3)$$

, where f_{oscMin} and f_{oscMax} are the minimum and maximum deviations of frequency from the specified resonator frequency.

The graph in *figure 4.1* shows the resonator's temperature stability characteristics (Elliot 1995) and indicates that it will remain within the required tolerances for reliable performance in this application.

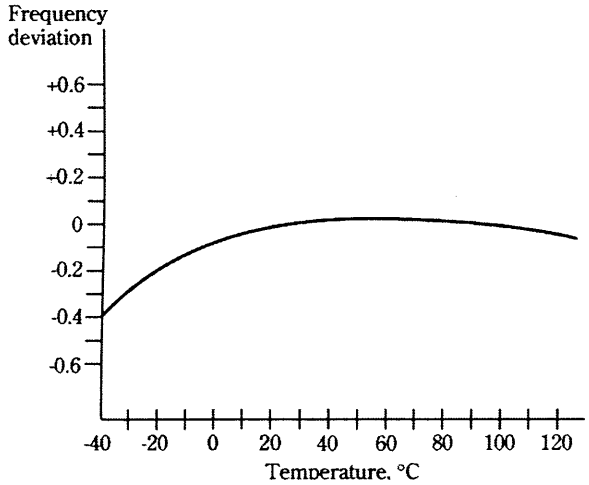


Figure 4.1 Temperature stability characteristics of CAN controller hardware surface mount resonator (reference temperature = 25°C).

4.1.3 Bit Timing

For communication to occur, all nodes the network must be configured to transmit and receive serial data at the same nominal bit rate. The CAN protocol uses non-return to zero (NRZ) coding where a logic '1' bit is represents a high value and a logic '0' represents a low value. As no clock signal is encoded within the data stream, it is possible that long sequences of consecutive bits with the same value may occur (i.e. because there is no logic transition, there is no edge) making it difficult for the receiving node to synchronise to the transmitter clock frequency and successfully decode the incoming data. Also, oscillator frequencies and transmission time can vary from node to node, causing phase distortion and signal skew, so the receiver must have some type of Phase Lock Loop (PLL) that responds to both the frequency and the phase of the input signals, automatically raising or lowering the frequency of the receiver oscillator until it is matched to the transmitter in both frequency and phase. Finally, since the data is NRZ

coded, it is necessary to include bit stuffing to ensure that an edge occurs at least every six bit times, so that the PLL can maintain this synchronisation.

The CAN controller hardware implements a DPLL that is configured to provide nominal timing and synchronise to the incoming serial data stream. The DPLL decomposes each bit time into multiple segments made up of minimal periods of time called the time quanta. Bus timing functions executed within the bit time frame, such as synchronisation to the local oscillator, network transmission delay compensation, and sample point positioning, are defined by the programmable bit timing logic of the DPLL. The nominal bit rate is the number of bits transmitted per second assuming an ideal transmitter with an ideal oscillator, in the absence of resynchronisation. The CAN standard states the maximum allowable nominal bit rate (f_{bit}) is 1Mb/s.

4.1.4 Programming Time Segments

The nominal bit time can be considered as being made up from non-overlapping discrete time segments as shown in *figure 4.2* below.

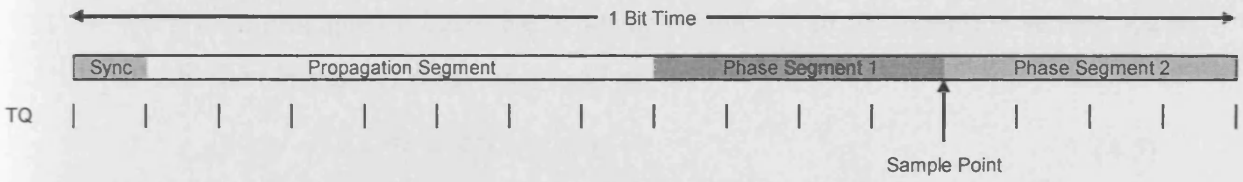


Figure 4.2 Bit time partitioning.

The time segments are divided into the following component segments:

4.1.4.1 Synchronisation Segment

The length of this segment is always 1 TQ. If there is a bit state change between the previous bit and the current bit, then the bus state change is expected to occur within this segment by the receiving nodes.

4.1.4.2 Propagation Segment

The phase buffer segment is used to compensate for physical delays within the network, i.e. signal propagation time on the bus line and the internal delay times of the nodes. This segment may be 1 to 8 TQ long. The system propagation delay (t_{prop}) is calculated as being the signal's roundtrip on the bus network (t_{bus}), the input comparator delay (t_{cmp}), and the output driver delay (t_{drv}). All the nodes in the system have similar

component delays, so the propagation delay can be expressed mathematically in the following formula,

$$t_{prop} = 2 \cdot (t_{bus} + t_{cmp} + t_{drv}) \quad (4.4)$$

The typical propagation delay expected from a bus cable (t_{bus}) is in the order of 5 ns/m (Burr-Brown Products, SN65HVD230 3.3V CAN Transceivers SLOS346E, 2001). The output and driver delays can be deduced from the fragment of the CAN transceiver data sheet shown below

Parameter	Description	Min	Typ	Max
t_{PLH}	Propagation delay time, low-to-high-level output	-	35 ns	85 ns
t_{PHL}	Propagation delay time, high-to-low-level output	-	70 ns	120 ns
$t_{sk(p)}$	Pulse skew (t_{PHL} - t_{PLH})	-	35 ns	-
t_r	Differential output signal rise time	25 ns	50 ns	100 ns
t_f	Differential output signal fall time	40 ns	55 ns	80 ns

Note: $C_L = 50\,pF$, $R_L = 60\Omega$

Table 4.2 Driver switching characteristics.

Given a bus length of 3 m, $t_{bus} = 3 \cdot 5 = 15\ ns$. Also, from the transceiver device data, the worst case propagation delay for is during a logic '1' to '0' transition setting, $t_{drv} = 70\ nS$ and the comparator delay, $t_{drv} = 55\ ns$, so

$$t_{prop} = 2 \cdot (15 + 70 + 55) = 280\ ns \quad (4.5)$$

4.1.4.3 Phase Buffer Segments

The phase buffer segments are used to compensate for edge phase errors on the bus. Phase segment 1 can be lengthened and phase segment 2 can be shortened by the resynchronisation process to optimally locate the sampling point of the received bit within the nominal bit time. The sampling point occurs at the end of phase segment 1. Phase segment 1 is programmable from 1 TQ to 8TQ in duration. Phase segment 2 provides delay before the next transmitted data transition and is also programmable from 1 TQ to 8 TQ in duration, or it may be defined to be equal to the greater of phase segment 1 or the Information Processing Time (IPT).

The nominal bit time can be defined as:

$$T_{bit} = t_{syncSegment} + t_{propSegent} + t_{phaseSegment1} + t_{phaseSegment2} \quad (4.6)$$

These time segments can be further decomposed into discrete units of time called time quanta (TQ). The length of each time quantum is based on the oscillator period (t_{osc}). The base TQ equals twice the oscillator period. The TQ length equals one TQ Clock period (t_{brpclk}), which is configured through a programmable baud rate prescaler (Richards 2001). This is described by the following equation:

$$TQ = 2 \cdot (BRP + 1) \cdot t_{osc} = 2 \cdot (BRP + 1) / f_{osc}, \quad (4.7)$$

, where BRP is the prescaler, a binary value represented by the CNF1.BRP<5:0> register in the controller hardware (see page 39 MCP2510 Datasheet). The nominal bit time (t_{bit}) for this application is defined as:

$$T_{bit} = 1 / f_{bit} = 1 / 125000 = 8\ \mu s, \quad (4.8)$$

Also

$$T_{osc} = 1 / f_{osc} = 1 / 16{,}000{,}000 = 0.0625\ \mu s, \quad (4.9)$$

, where f_{osc} is the resonator frequency (16 MHz).

Setting BRP to 3 (CNF1.BRP<5:0> binary value = 0b000011 = 0x03) gives:

$$TQ = 2 \cdot (3 + 1) \cdot 0.0625 = 0.5\ \mu s \quad (4.10)$$

The lengths of the time segments are set to the following values:

Synchronisation Segment = 1 TQ (time quantum)
Propagation Segment = 7 TQ
Phase Segment 1 = 4 TQ
Phase Segment 2 = 4 TQ

From Eq. (4.6) the total duration of one bit is calculated as:

$$\begin{aligned} T_{bit} &= TQ \cdot (\text{Synchronisation Segment} + \text{Propagation Segment} \\ &\quad + \text{Phase Segment 1} + \text{Phase Segment 2}) \\ &= 0.5 \cdot (1 + 7 + 4 + 4) = 8\ \mu s \end{aligned} \quad (4.11)$$

It can be seen that this value for t_{bit} agrees with that from Eq. (4.2) confirming that the hardware is configured for correct operation.

4.1.5 Sample Point

The sample point is the point of time at which the bus level is read and value of the received bit is determined. If the bit timing is slow (i.e. the bit rate is low) and contains many TQ, it is possible to specify multiple sampling of the bus line at the sample point. The value of the received bit is determined to be the value of the majority decision of three values. The three samples are taken at the sample point, and twice before with a time of TQ / 2 between each sample. The sampling of the bit takes place at approximately 70-80% of the bit time. In this case:

$$\begin{aligned} \text{Sampling point} &= 100 - 100 \cdot \textit{Phase Segment 2} / (\textit{Synchronisation Segment} + \textit{Propagation Segment} + \textit{Phase Segment 1} + \textit{Phase Segment 2}) \\ &= 100 - 100 \cdot 4 / (1 + 7 + 4 + 4) = 75\% \end{aligned} \quad (4.12)$$

The Information Processing Time (IPT) is the time segment, starting at the sample point, that is reserved for calculation of the subsequent bit level. The CAN specification (Bosch 1991) defines this time to be less than or equal to 2 TQ. The CAN controller hardware defines this time to be 2 TQ. Thus, phase segment 2 must be at least 2 TQ long.

4.1.6 Synchronisation

To compensate for the resonator frequency drift between nodes on the network, the CAN controller hardware must be able to synchronise to the relevant edge of the incoming signal to ensure an incoming message is decoded correctly. This synchronisation is achieved using two different mechanisms.

4.1.6.1 Hard Synchronisation

Hard synchronisation is performed only when there is a recessive to dominant edge transition (logic '1' to '0') during a bus idle condition, indicating the start of frame (SOF). The bit timing counter is reset to force the edge to lie within the synchronisation segment. At this point, all of the receivers are synchronised to the transmitter. Hard synchronisation occurs only once during a message. Also, resynchronisation may not occur during the same bit time (SOF) that hard synchronisation occurred.

4.1.6.2 Resynchronisation

Resynchronisation allows a receiver node to maintain the initial synchronisation that was established by the hard synchronisation. Without this, receiving nodes can get out of synchronisation due to oscillator frequency drift between nodes. Resynchronisation is achieved by the DPLL which compares the actual position of a recessive-to-dominant edge on the bus to the position of the expected edge (within the synchronisation segment) and adjusting the bit time as necessary. As a result of resynchronisation, phase segment 1 may be lengthened or phase segment 2 may be shortened. The amount of lengthening or shortening of the phase buffer segments has an upper bound given by the Synchronization Jump Width (SJW). The value of the SJW will be added to phase segment 1 or subtracted from phase segment 2. The SJW represents the loop filtering of the DPLL. The SJW is programmable between 1 TQ and 4 TQ.

Clocking information will only be derived from recessive to dominant transitions. The property that only a fixed maximum number of successive bits have the same value ensures resynchronisation to the bit stream during a frame. The phase error of an edge is given by the position of the edge relative to synchronisation segment, measured in TQ (Microchip MCP2510 Data Sheet, 1999).

For typical applications an SJW of 1 TQ is used, however if resonator frequencies of different nodes on the network is inaccurate or unstable a larger value may be required. Given that Eq. (4.1) calculates the selected resonator tolerance at 0.8% and from Eq. (4.2) the nominal bit rate is 8 μs then the allowable minimum SJW can be calculated using the following formula:

$$10t_{(bitA)} > 10t_{(bitB)} + t_{SJW(B)}. \tag{4.13}$$

, where $t_{bit(N)}$ = bit time of node "N" and $t_{SJW(n)}$ = SJW of node "N".

The oscillator tolerance between the slowest node and the fastest node is used to determine the minimum SJW. The equation assumes that node A is the slow node (longest bit time) and node B is the fast node (shortest bit time). The bit-stuffing rule guarantees that no more than five like bits in a row will be transmitted during a message frame. The only exception is at the end of the message that includes ten recessive bits (one ACK delimiter, seven end-of-frame bits, and three inter-frame space bits). Eq.

(4.6) is derived from the fact that resynchronisation only occurs on recessive-to-dominant edges which implies that there can be a maximum of ten bits between resynchronisation due to bit stuffing (Richards 2001).

From Eq. (4.8), the nominal bit time for the two nodes A and B is 8 μs and from Eq. (4.1), the resonator tolerance is 0.8%, the worst case difference in nominal bit times for the nodes can be determined as follows:

$$t_{bit(A)} = 8 + (8 \cdot 0.8 / 100) = 8.064\ \mu s, \tag{4.14}$$

And

$$t_{bit(B)} = 8 - (8 \cdot 0.8 / 100) = 7.936\ \mu s \tag{4.15}$$

Given that each bit is composed of 16 time quanta:

$$TQ_{(A)} = 8.064 / 16 = 504\ ns \tag{4.16}$$

and

$$TQ_{(B)} = 7.936 / 16 = 496\ ns \tag{4.17}$$

Rearranging Eq. (4.13) gives:

$$t_{SJW(B)} > 10t_{(bitA)} - 10t_{(bitB)} = 10 \cdot 504 - 10 \cdot 496 = 80\ ns \tag{4.18}$$

The number of time quanta in the synchronisation jump width is calculated as:

$$TQ_{SJW} = t_{SJW(B)} / TQ_{(B)} = 80 / 496 = 0.16 = 1\ TQ \tag{4.19}$$

Confirming that reliable data transfer for this application is satisfied by implementing basic SJW resynchronisation.

4.1.7 Transmit and Receive Buffers

The CAN controller hardware contains three transmit and two receive buffers *[see figure 4.3]*. Each of the transmit buffers occupies 14 bytes of SRAM and they are mapped into the hardware memory. Five bytes are used to hold the standard and extended identifiers and other message arbitration information. A further eight bytes contain the data payload of the message to be transmitted. The two receive buffers have

multiple acceptance filters and there is also a separate message assembly buffer (MAB) which functions as a further receive buffer.

The hardware acceptance filters are typically configured to pre-select which messages will be allowed into the receive buffers RXB0 or RXB1 (Richards 2001). The low bandwidth requirements for this project make it feasible to utilise just one receive (and one transmit) buffer, therefore keeping firmware complexity and size to a minimum. In this case, the control register bits are cleared, which ensures that the node receives all messages in RXB0 (RXB1 is not used). Transmit buffer TXB0 is used to hold all outgoing messages.

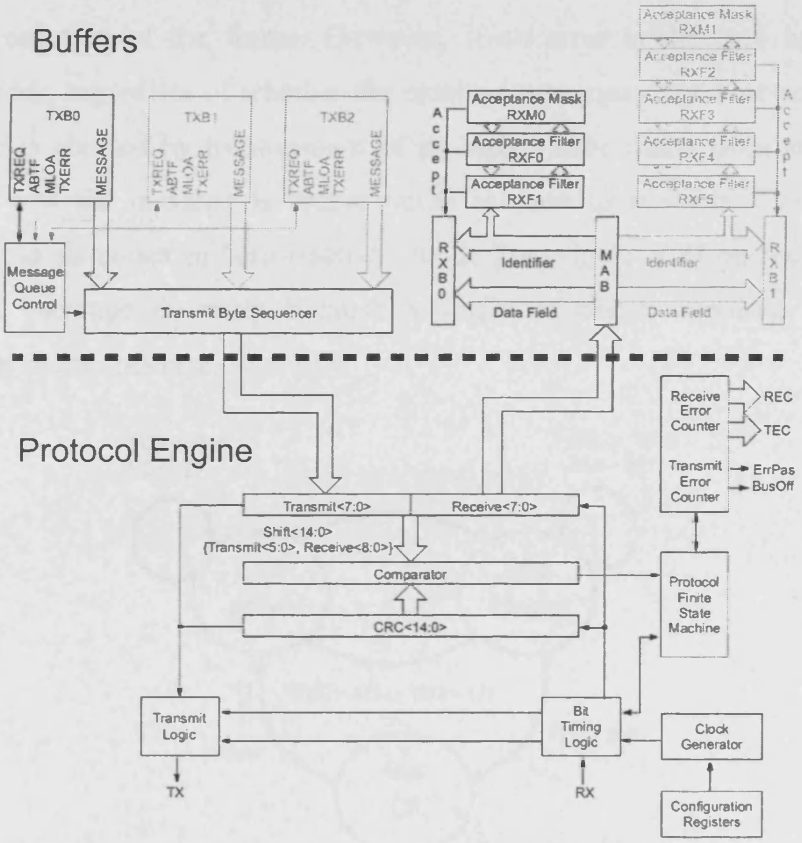


Figure 4.3 CAN Buffers and Protocol Engine.

4.1.8 Error Detection Mechanisms

The CAN controller hardware within each node fully implements a data link layer protocol to facilitate reliable data transfer between nodes on the network (physical layer) and minmise the requirement for error management at the application program level. There are numerous factors (environmental and intrinsic electrical noise, radiation, etc) that may cause unreliable data transfer across the network and the CAN

data link layer provides the functional and procedural means to detect and possibly correct errors that may occur in the physical layer. These mechanisms include cyclic redundancy check (CRC), flow control, error checking, frame acknowledgments and retransmission.

In low-level hardware, each node actively monitors the system network and checks the CRC of every message received. The CRC protects the entire frame from the first bit of the message identifier up to the last bit of the last data byte, with the calculated 15-bit checksum being transmitted after the last data bit. All nodes which receive the data frame calculate its checksum and then compare it with one sent with the frame. If the checksums match, the acknowledgement bit is set to the dominant state to confirm successful reception of the frame. However, if an error is detected by any of the receiving nodes, regardless of whether the message was meant for it or not, the current transmission is aborted by transmission of an active error frame from at least one of these nodes and the message is retransmitted as soon as possible. This mechanism guarantees that all nodes in "error-active" mode *[see figure 4.4]* on the network has received the message correctly because a single mismatch condition would have destroyed the faulty message.

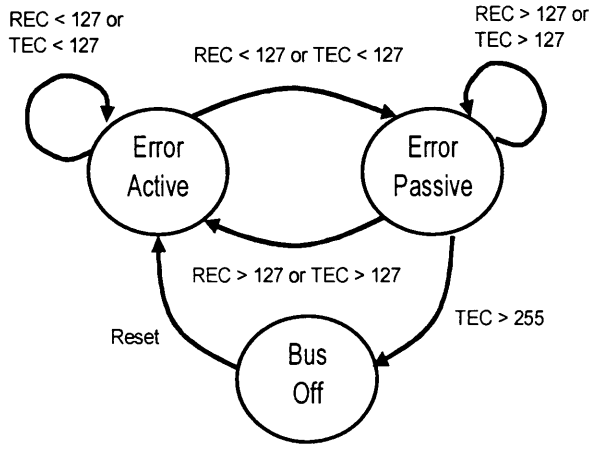


Figure 4.4 CAN Controller Error State Diagram.

Each CAN node is in one of the three error states "error-active", "error-passive" or "bus-off" depending to the value of the internal error counters. The error-active state is the usual state where the node can transmit messages and active error frames (made of dominant bits) without any restrictions. In the error-passive state, messages and passive error frames (made of recessive bits) may be transmitted. The bus-off state makes it

temporarily impossible for the station to participate in the bus communication. During this state, messages can neither be received nor transmitted.

The CAN controller hardware implements two error counters: the Receive Error Counter (REC) and the Transmit Error Counter. These are incremented with each error detected by the node, where the more severe the error, the greater the increment value. The counters are decremented when a message is successfully received or transmitted. If either of the two counters reach 127, the node switches into the error-passive state. In this state, the node can still send and receive messages, however will not actively destroy message frames on the network. It is possible for the node to self-recover and autonomously switch back to error-active state if the counters decrement below 127.

If the transmit error counter continues to increment and overflows then the node switches into the bus-off state forcing the node to shutdown and cease communication, until the bus-off recovery sequence is received. The bus-off recovery sequence consists of 128 occurrences of 11 consecutive recessive bits.

The node application program monitors the status of the controller hardware and if the bus-off state occurs it performs a controlled shutdown of the node systems, so that the remainder of the machine system can continue to function with minimal impact from the failed node.

4.2 CANopen Implementation

One of the huge benefits of a higher layer protocol is the guarantee of interchangeability between the same type of off-the-shelf devices (or nodes) from different manufacturers. This ensures interoperability between all devices that comply with this networking standard, thus simplifying the task of system integration. The documents that constitute the CANopen device profiles describe in detail how to use CANopen for a particular type of device, what communication parameters are available and how the object dictionary is set up (Pfeiffer 2003). Several CANopen services, such as service data objects, process data objects and network management, are implemented in this project to support tasks such as data transmission/reception between nodes on the system

network, control the connection procedures and monitor nodes on the network. The central point of connection for these communication objects is the Object Dictionary, which acts as a standard communication interface for every node on the system network.

A model representation of a node is shown below in *figure 4.5* to illustrate the relationships between the CANopen objects, node sub-systems and how they interface with the bus network.

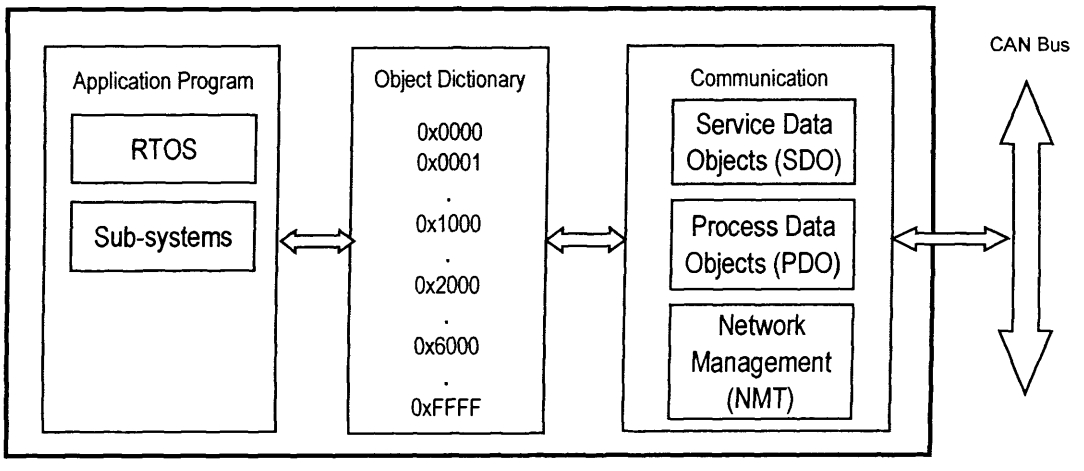


Figure 4.5 Node model.

4.2.1 The Object Dictionary Concept

All nodes implement their own Object Dictionary which is a table containing node-specific configuration and process data parameters in a standard format. The Object Dictionary is formal representation of data each node requires to operate and contains information about:

- Device Type Information
- Error Register
- Heartbeat Consumer Time
- Heartbeat Producer Time
- Identity Object
    1. Vendor ID
    2. Product Code
    3. Revision number
    4. Serial Number.

- NMT Startup (autostart)
- Time and Date
- Effluent temperature
- Effluent temperature + heating (flow rate sensing)
- Sea temperature
- Sample rate
- ADC Configuration (gain, buffer and offset)
- Valve Position (inlet, outlet, closed, unknown)
- Pump (on/off)
- Heater (on/off)
- Sensor calibration coefficients
- Sensor calibration formula
- State machine initialisation registers.

The structure of the Object Dictionary is mapped onto silicon (i.e. non-volatile registers in FRAM, ADC and RTC hardware) and makes it possible to access all important data, parameters and functions of a node using a logical addressing system (index, sub-index) from the "outside", i.e. via the network. An entry in the Object Dictionary is addressed using a hexadecimal index, which is 16 bits in size, allowing a maximum of 0xFFFF entries. Each entry is further subdivided into a maximum of 0xFF sub-entries (8-bit sub-index). Sub-entries are typically used to combine values of the same type, such as with an array, or to access connected values, such as with a data record. An Object Dictionary entry therefore has a 24-bit address, made up of the 16-bit index and 8-bit sub-index. The Object Dictionary can be sub-divided into blocks of 0x1000 entries each to define standardised regions as shown in the table below.

Index	Object
0x0000	Reserved
0x0001 - 0x001F	Static data types
0x0020 - 0x003F	Complex data types
0x0040 - 0x005F	Manufacturer specific data types
0x0060 - 0x007F	Device profile specific static data types
0x0080 - 0x009F	Device profile specific complex data types
0x00A0 - 0x0FFF	Reserved for future use
0x1000 - 0x1FFF	Communication entries (specified in DS-301)
0x2000 - 0x5FFF	Manufacturer specific entries
0x6000 - 0x9FFF	Standardised device entries
0xA000 - 0xFFFF	Reserved for future use

Table 4.3 CANopen Object Dictionary overview.

The Object Dictionary allows for structuring of node data (parameter values) as entries that are referenced through an index/sub-indexing system. Every entry in the object dictionary can be modified (if write access is permitted) using service data objects (SDO) or process data objects (PDO). These communication objects allow a node to be configured and controlled, as an object dictionary entry can also directly represent a property or a function of the node (e.g. the absolute position of a valve). With read access to an object dictionary entry, the node returns the parameter value of the entry. The data type and meaning of the value must be known to the enquirer. For this reason the object dictionary can also be represented in a standardised textual form, called an electronic data sheet (EDS), which describes each object dictionary entry with address index/sub-index), parameter name, data type, access type and default value. Other advantages include availability tools for checking EDS to ensure CANopen conformance of the design.

If an entry stores only one data parameter, then only one sub-entry is required at sub-index 0x00, however multiple data parameters are stored in separate sub-entries along with the highest sub-entry at sub-index 0x00. For example, if index 0x2000 stores a single 8-bit data parameter then:

Index 0x2000, sub-index 0x00 = 8-bit value

And if index 0x2001 stores two 8-bit values then:

Index 0x2001, sub-index 0x00 = 0x02 (number of highest sub-entry)
Index 0x2001, sub-index 0x01 = first 8-bit value
Index 0x2001, sub-index 0x02 = second 8-bit value

It is not necessary or practical to develop a node that maintains an entire object dictionary with all entries. Certain regions of the object dictionary are non-critical and can be purposefully left out, leaving gaps in the table. For example, the entries with indexes 0x0000 to 0x0FFF are not implemented in this application.

The table below lists the first seven entries from the object dictionary that represent a small part of the region from 0x0001 to 0x0FFF which specifies commonly used data types. These entries do not store any data parameters, however if physically implemented in a node; a read operation would return the data size of that data type in bytes. CANopen also supports definition of application specific data types and these would be stored in region 0x0040 to 0x005F.

Index	Data Type	Parameter Value (size in bytes)
0x0001	BOOLEAN	1
0x0002	INTEGER8	1
0x0003	INTEGER16	2
0x0004	INTEGER32	4
0x0005	UNSIGNED8	1
0x0006	UNSIGNED16	2
0x0007	UNSIGNED32	4

Table 4.4 Object Dictionary entries defining data types.

Although the region of the object dictionary that describes data types is not implemented here, references are made to the data type definitions by entries in the table beyond 0x1000. These entries are allocated for variable storage where; for example, if an entry is specified to be of type 'UNSIGNED8' then the parameter value in the corresponding object dictionary field will be 5.

4.2.2 Communication Entries

The communication entries in the object dictionary contain parameter values that describe most aspects of CANopen communication behavior of the node.

Index	Object	Name	Data type	Attribute
0x1000	Variable	Device type	UNSIGNED32	RO
0x1001	Variable	Error register	UNSIGNED8	RO
0x1002	Variable	Manufacturer status register	UNSIGNED32	RO
0x1003	Array	Pre-defined error field	UNSIGNED32	RW
0x1005	Variable	COBID of Sync object	UNSIGNED32	RW
0x1006	Variable	Communication cycle period	UNSIGNED32	RW
0x1007	Variable	Synchronous window length	UNSIGNED32	RW
0x1008	Variable	Device name	VISIBLE_STRING	CONST
0x1009	Variable	Hardware version	VISIBLE_STRING	CONST
0x100A	Variable	Software version	VISIBLE_STRING	CONST
0x100C	Variable	Guard time	UNSIGNED16	RW
0x100D	Variable	Life time factor	UNSIGNED8	RW
0x1010	Array	Store parameters	UNSIGNED32	RW
0x1011	Array	Restore default parameters	UNSIGNED32	RW
0x1012	Variable	COBID of TimeStamp object	UNSIGNED32	RW
0x1013	Variable	High resolution time stamp	UNSIGNED32	RW
0x1014	Variable	COBID of emergency object	UNSIGNED32	RW
0x1015	Variable	Inhibit time for emergency object	UNSIGNED16	RW
0x1016	Array	Consumer heartbeat time	UNSIGNED32	RW
0x1017	Variable	Producer heartbeat time	UNSIGNED16	RW
0x1018	Record	Identity object	UNSIGNED32	RO
0x1200 ... 0x127F	Record	1st ... 128th server SDO	SERVER_SDO_PARAMETERS	RW
0x1280 ...0x12FF	Record	1st ... 128th client SDO	CLIENT_SDO_PARAMETERS	RW
0x1400 ... 0x15FF	Record	1st ... 512th receive PDO	RXPDO_COMMUNICATION_PARAMETERS	RW
0x1600 ... 0x17FF	Array	1st ... 512th receive PDO mapping	RXPDO_MAPPING_PARAMETERS	RW
0x1800 ... 0x19FF	Record	1st ... 512th transmit PDO	TXPDO_COMMUNICATION_PARAMETERS	RW
0x1A00 ... 0x1BFF	Array	1st ...512th transmit PDO mapping	TXPDO_MAPPING_PARAMETERS	RW

Table 4.5 Communication Entry overview.

It should be noted that entries [0x1000, 0x00], [0x1001, 0x00], [0x1018, 0x00] and [0x1018, 0x01] are mandatory i.e. for CANopen conformance, all nodes must implement these entries their object dictionary

Index	Sub-index	Description	Data type	Attribute	Default Value
0x1000	-	Device type	UNSIGNED32	CONST	0x00000000
0x1001	-	Error register	UNSIGNED8	RO	0x00
0x1008	-	Device name	VISIBLE_STRING	CONST	"ISOSAMPLER"
0x1016	-	Heartbeat consumer time	UNSIGNED8	RO	0xii
0x1016	0x00	1.01s x 2 = 2.02s (2020ms) for node 1	UNSIGNED32	CONST	0x000107E4
0x1016	0x01	1.03s x 2 = 2.06s (2060ms) for node 2	UNSIGNED32	CONST	0x0002080C
0x1016	0x02	1.07s x 2 = 2.14s (2140ms) for node 3	UNSIGNED32	CONST	0x0003085C
0x1017	0x03	Node N (add other nodes within the communication group)			
0x1017	0x00	Heartbeat producer time 1.09s (1090ms)	UNSIGNED16	CONST	0x0442
0x1018	-	Identity	-	-	-
0x1018	0x00	Number of entries	UNSIGNED8	RO	0x04
0x1018	0x01	Vendor ID	UNSIGNED32	RO	0x00000000
0x1018	0x02	Product code	UNSIGNED32	RO	0x000000ii
0x1018	0x03	Revision number	UNSIGNED32	RO	0x000000ii
0x1018	0x04	Serial Number	UNSIGNED32	RO	0x000000ii
0x1F80	0x00	NMT Startup (Nodes that autostart report 0 in this entry)	UNSIGNED8	CONST	0x00

Table 4.6 Communication Entries implementation.

4.2.2.1 Device Type Entry [0x1000]

This entry stores the number of the device profile and also provides additional information (defined in the Device Profile Specification) about which features of the device profile are utilised in the node. The entry has the following structure:

Additional information																Device profile number															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 4.6 Device Type entry format.

If the node does not use a device profile then the Device Profile Number is zero and the additional information value field is left undefined or set to zero as well. The Device Type entry is mandatory and can be used as a way of dynamically scanning for nodes connected to the network.

4.2.2.2 Error Register Entry [0x1001]

When bits are set in the error register this indicates occurrences of certain types of errors as shown below:

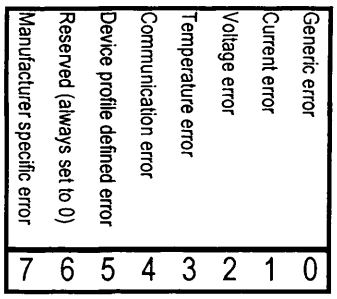


Figure 4.7 Error Register entry format.

The Generic Error field is a mandatory for full CANopen compliance and indicates the occurrence of any type of error in the node systems. The other fields are optional; however the "voltage" field is used as a flag that the node battery is low and the "temperature" field is used to flag that the electronic hardware is overheating, for example if the node were placed in a high temperature environment such as a hydrothermal vent plume.

4.2.2.3 Heartbeat Consumer Time Entry [0x1016, 0xii]

Nodes that work together as communication partners constantly monitor the heartbeat messages generated by other nodes within their group. The heartbeat consumer entry within each node's object dictionary specifies the maximum time that the receiving

node will wait (in milliseconds) for a heartbeat message from a specific transmitting node before generating an event. If no heartbeat is received within the specified period then the receiving node records it as being absent. Each 32-bit sub-index specifies the heartbeat consumer time for a specific node and has the following structure:

Reserved	Node ID	Heartbeat consumer time (ms)
31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Figure 4.8 Heartbeat Consumer Time entry format.

For example, if node #1 has a producer heartbeat time of 907ms then nodes within its communication group are configured to expect the heartbeat within 200% of this period i.e. $907 \cdot 2 = 1814$ms. Therefore,

Heartbeat consumer time = 0x00010716 (0x0716 = 1814 ms)

4.2.2.4 Heartbeat Producer Time Entry [0x1017]

The value of this entry specifies the time between transmission of the heartbeat messages in milliseconds. A value of zero disables transmission of heartbeat messages by a node. For example,

Heartbeat producer time = 0x03A0 (0x03A0 = 928 ms)

CANopen defines the attribute of heartbeat producer and consumer time entries as being writable i.e. the parameter value can be modified dynamically during runtime, however for security and in the interests of reducing code size these values are defined as constants.

4.2.2.5 Identity Entries [0x1018, 0xii]

The Identity entry contains basic information about the node in order to provide a standard method of distinguishing between different versions of a node.

The Vendor ID is a unique ID assigned to each CANopen vendor by CiA group to allow the source of the node to be identified. This Vendor ID entry is a numeric value of type UNSIGNED32 and consists of a unique number for each registered company and optionally for each department of that company (but only if required) as shown below.

Department	Company
31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Figure 4.9 Vendor ID entry format.

A copy of the CiA registration form is included in ***Appendix F***. It is proposed that the company be registered as "Cardiff University" and the department as "School of Earth, Ocean and Planetary Sciences".

The Revision Number entry has the following format:

Major Revision Number																Minor Revision Number															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 4.10 Revision Number entry format.

Where the minor revision number identifies different versions of the node where the CANopen behaviour has not changed and the major revision number identifies different versions of the node where the CANopen behaviour has changed. For example, if there are any differences in CANopen messages and object dictionary entries then major revision number should be incremented:

0x00030001 = Revision number 3.1

The Product Code and Serial Number entry field formats are manufacturer specific. For this application only the least significant byte is utilised all the rest are cleared to zero. Product codes are defined in accordance with the table below:

Node Type	Parameter Value
NMT	0x00000000
Temperature sensor	0x00000001
Pump	0x00000002
Pressure compensator	0x00000003
Master valve	0x00000004
Slave valve	0x00000005
Bridge	0x00000006

Table 4.7 Product Codes for the nodes that constitute the "Isosampler" system.

4.2.3 Manufacturer Specific Entries

This region of the Object Dictionary is used extensively by the node application program for storage of parameter values, representing node-specific data and configuration settings that are outside the CANopen standard. For example, each node features a real time clock where the current time of each node is made available in an object dictionary entry so that other nodes on the system network can read it. This achieved by defining the Object Dictionary in the manufacturer specific region as shown below:

Index 0x2000, sub-index 0x00 = 0x03 (number of highest sub-entry)

Index 0x2000, sub-index 0x01 = seconds (UNSIGNED8)

Index 0x2000, sub-index 0x02 = minutes (UNSIGNED8)

Index 0x2000, sub-index 0x03 = hours (UNSIGNED8)

A full implementation of the manufacturer specific entries in a typical node is shown in the table below.

Index	Sub-index	Description	Data type	Attribute	Default Value
0x2000	-	RTC time parameters	-	-	-
0x2000	0x00	Number of entries	UNSIGNED8	RO	0x08
0x2000	0x01	Seconds	UNSIGNED8	RW	0x01
0x2000	0x02	Minutes	UNSIGNED8	RW	0x01
0x2000	0x03	Hours	UNSIGNED8	RW	0x01
0x2000	0x04	Date	UNSIGNED8	RW	0x01
0x2000	0x05	Month	UNSIGNED8	RW	0x01
0x2000	0x06	Day	UNSIGNED8	RW	0x01
0x2000	0x07	Year	UNSIGNED8	RW	0x70
0x2000	0x08	Control (write protect)	UNSIGNED8	RW	0x00
0x2001	-	RTC alarm parameters	-	-	-
0x2001	0x00	Number of entries	UNSIGNED8	RO	0x08
0x2001	0x01	Seconds alarm threshold	UNSIGNED8	RW	0x7F
0x2001	0x02	Minutes alarm threshold (bit 7 alarm state)	UNSIGNED8	RW	0x7F
0x2001	0x03	Hours alarm threshold	UNSIGNED8	RW	0xBF
0x2001	0x04	Date alarm threshold	UNSIGNED8	RW	0x3F
0x2001	0x05	Month alarm threshold	UNSIGNED8	RW	0x1F
0x2001	0x06	Day alarm threshold	UNSIGNED8	RW	0x07
0x2001	0x07	Year alarm threshold	UNSIGNED8	RW	0xFF
0x2001	0x08	Clock Burst	UNSIGNED8	RW	0x3F
0x2002	-	ADC parameters	-	-	-
0x2002	0x00	Number of entries	UNSIGNED8	RO	0x04
0x2002	0x01	Setup Register	UNSIGNED8	RW	0xi0
0x2002	0x02	MUX (multiplexer control)	UNSIGNED8	RW	0x01
0x2002	0x03	ACR (analogue control register)	UNSIGNED8	CONST	0x50
0x2002	0x04	ODAC (offset DAC)	UNSIGNED8	CONST	0x00
0x2003	-	Actuator control	UNSIGNED8	RW	0xii
0x2004	-	Hardware error register	UNSIGNED8	RO	0xii
0x2005	-	Set compensator pressure (1 atmosphere)	UNSIGNED32	RW	0x47C5E680
0x2006	-	Bottle pressure	UNSIGNED32	RO	0xiiiiiiii
0x2007	-	Pump speed	UNSIGNED8	RW	0xii
0x2008	-	Valve position	UNSIGNED8	RO	0x0i
0x2009	-	Delta T (seconds)	UNSIGNED8	RW	0xii
0x200A	-	Battery voltage	UNSIGNED8	RO	0xii
0x200B	-	RS-232 Baudrate	UNSIGNED16	RW	0x4B00
0x200C	-	Count POR (power-on reset)	UNSIGNED8	RO	0xii
0x200D	-	Data representation register	UNSIGNED8	RW	0xii
0x200E	-	Pressure case temperature	REAL32	RO	0x00iiiiii
0x200F	-	Effluent temperature	REAL32	RO	0x00iiiiii
0x2010	-	Fluid flow velocity	REAL32	RO	0x00iiiiii
0x2011	-	Calibration parameters	-	-	-
0x2011	0x00	Number of entries	UNSIGNED8	RO	0x07
0x2011	0x01	Polynomial coefficient C1 (1.13287E-06)	REAL32	RW	0xiiiiiiii
0x2011	0x02	Polynomial coefficient C2 (6.12723E-04)	REAL32	RW	0xiiiiiiii
0x2011	0x03	Polynomial coefficient C3 (2.39747)	REAL32	RW	0xiiiiiiii
0x2011	0x04	Polynomial coefficient C4 (247.0073)	REAL32	RW	
0x2011	0x05	ADC resolution (24-bit = 2^24)	UNSIGNED32	CONST	0x000FFFFF
0x2011	0x06	ADC reference volts (1.000 V)	REAL32	CONST	0x3F800000
0x2011	0x07	RTD excitation current (0.0002 A)	REAL32	CONST	0x3951B717

Table 4.8 Manufacturer Specific Entries implementation.

4.2.3.1 RTC Time and Alarm Entries [0x2000, 0x08; 0x2001, 0x08]

The alarm entries are used to set a pre-defined time that will trigger valves to open and the pump to activate to acquire a sample in the bottle. Time data is stored as BCD (binary coded decimal).

4.2.3.2 ADC Entries [0x2002]

The Setup and MUX entries are used to control the gain and channel selection for the data acquisition system. The ODAC and ACR are configured at compile time and unchangeable.

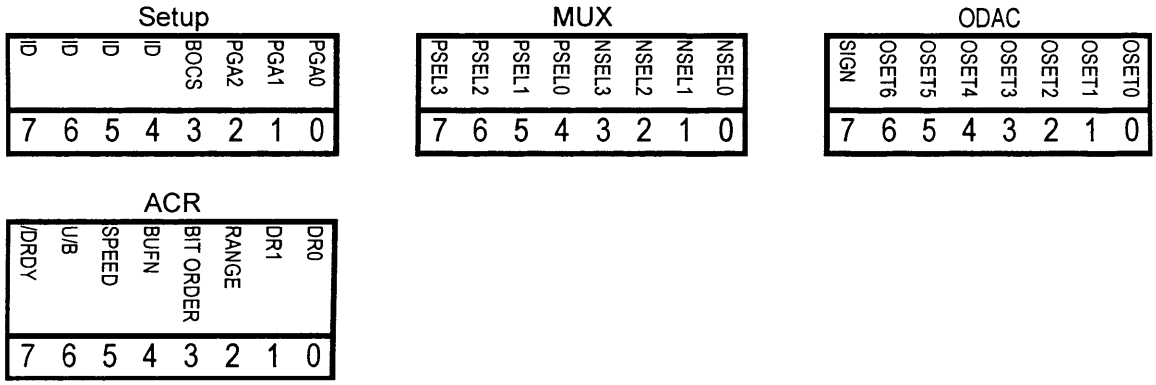


Figure 4.11 Format of entries containing ADC configuration parameters.

Name	Settings	Description
BOCS	0 = Disabled (default) 1 = Enabled	When the burnout current source bit is set, two current sources are enabled. The current source on the positive input channel sources approximately 2μA of current. The current source on the negative input channel sinks approximately 2μA. This allows for the detection of an open circuit (full-scale reading) or short circuit (0V differential reading) on the selected input differential pair.
PGA2:PGA1:PGA0	0b000 = 1 (default) 0b001 = 2 0b010 = 4 0b011 = 8 0b100 = 16 0b101 = 32 0b110 = 64 0b111 = 128	The programmable gain amplifier (PGA) can be configured for gains of 1, 2, 4, 8, 16, 32, 64, or 128 to improve the effective resolution of the ADC for a given gain range.
PSEL3: PSEL2: PSEL1: PSEL0	0b0000 = AIN0 (default) 0b0001 = AIN1 0b0010 = AIN2 0b0011 = AIN3 0b0100 = AIN4 0b0101 = AIN5 0b0110 = AIN6 0b0111 = AIN7	Positive channel select. The input multiplexer provides for any combination of differential inputs to be selected on any of the input channels which allows for eight true differential input channels.

NSEL3: NSEL2: NSEL1: NSEL0	0b1xxx = Reserved 0b0000 = AIN0 0b0001 = AIN1 (default) 0b0010 = AIN2 0b0011 = AIN3 0b0100 = AIN4 0b0101 = AIN5 0b0110 = AIN6 0b0111 = AIN7 0b1xxx = Reserved	Negative channel select.
DRDY	RO (read only)	This bit duplicates the state of the data ready (DRDY) signal.
U/B	0 = Bipolar (default) 1 = Unipolar	Data format.
SPEED	0 = f_{MOD} = f_{OSC} / 128 (default) 1 = f_{MOD} = f_{OSC} / 256	Modulator clock speed.
BUFN	0 = Buffer Disabled (default) 1 = Buffer Enabled	The input impedance of the ADC without the buffer is 5MΩ / PGA. With the buffer enabled the input voltage range is reduced, and the analog power-supply current is higher.
BIT ORDER	0 = Most Significant Bit Transmitted First (default) 1 = Least Significant Bit Transmitted First	Data is always shifted into the part most significant bit first. Data is always shifted out of the part most significant byte first. This configuration bit only controls the bit order within the byte of data that is shifted out.
RANGE	0 = Full-Scale Output Range Equal to ±V_{REF} (default). 1 = Full-Scale Output Range Equal to ±½ V_{REF}	Range select.
DR1: DR0	0b00 = 15Hz (default) 0b01 = 7.5Hz 0b10 = 3.75Hz 0b11 = Reserved	Data conversion rate. The modulator runs at a clock speed (f_{MOD}) that is derived from the external clock (f_{OSC}). The frequency division is determined by the SPEED bit in the SETUP register. Note: f_{OSC} = 2.4576MHz, SPEED = 0
SIGN	0 = Positive 1 = Negative	The offset digital to analogue converter (ODAC) register is an 8-bit value. The MSb is the sign
OSET6: OSET5: OSET4: OSET3: OSET2: OSET1: OSET0	See ADS1243 data sheet for details.	The input to the PGA can be shifted by half the full-scale input range of the PGA using the ODAC register. The seven LSbs provide the magnitude of the offset.

Table 4.9 Detailed register configurations for the ADC.

4.2.3.3 Actuator Control Entry [0x2003]

This single byte entry is used at to trigger RTOS tasks that are responsible for high-level

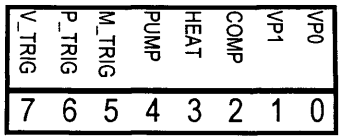


Figure 4.12 Actuator control entry format.

coordination of pump and valve nodes to acquire a fluid sample. The entry also contains several bit fields for direct manual control of actuator nodes on the network.

Name	Settings	Description
V_TRIG	COS trigger	Trigger slave valve control task (in slave valve)
P_TRIG	COS trigger	Trigger pump control task (in pump node)
M_TRIG	COS trigger	Manual triggering of the data acquisition system (DAQS) (in temp sensor node)
PUMP	0 = on; 1 = off (default)	Switch pump on/off.
HEAT	0 = on; 1 = off (default)	Switch heater on/off in flow velocity sensor.
COMP	0 = on; 1 = off (default)	Disable/enable pressure compensator.
VP1:VP0	0b00 = Closed 0b01 = Inlet open 0b10 = Outlet open 0b11 = Reserved	Sets valve position.

Table 4.10 Detailed actuator control register configurations.

4.2.3.4 Hardware Error Entry [0x2004]

When bits are set to logic '1' in the Hardware Error Entry, this indicates occurrences of various error conditions as shown below:

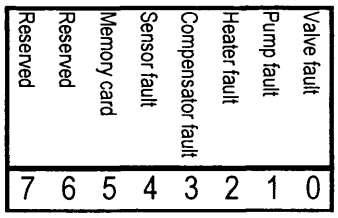


Figure 4.13 Hardware error entry format.

The nodes within a communication group must be "aware" of each other's operational status when cooperating to perform a task The Hardware Error Entry is essential to facilitate some form of error handling by the machine system in the event possible node sub-systems failure during operation. Its purpose is to ensure that hardware failures can be at least be detected so that the appropriate measures can be taken to minimise the impact the error has on the entire machine system. If a master node or host PC

connected to the network, this can be configured for use as diagnostic tool to detect and flag errors, so that repairs or node replacements can be made.

Also, with such a failure detection mechanism in place, it is feasible to handle errors at various levels within the system architecture. For example, if a node that detects an internal error, it should shut-down damaged sub-systems *[see **Chapter 6** for a detailed description of node subsystems]* and, if still capable of external communication, the error transmitted to the commanding node within its communication group. If the failed node is essential to the operation of the group, then the group is no longer capable of fulfilling its mission objectives. In this scenario, the whole group should perform an autonomous and controlled shut-down (without the requirement for intervention from a master controller or host) to preserve network bandwidth and conserve power.

4.2.3.5 Set Compensator Pressure Entry [0x2005]

This entry holds the target pressure to which the pressure compensator should maintain in N/m^2 (Pascals). Default value is one atmosphere (101,325 N/m^2). The parameter value can be dynamically changed via another node, for example for controlled depressurisation of the fluid within the bottle to remove or transfer a sample.

4.2.3.6 Bottle Pressure [0x2006]

This entry indicates the hydrostatic pressure within the sealed sampling bottle.

4.2.3.7 Set Pump Speed Entry [0x2007]

This entry can be modified to change the default pump speed and direction if required.

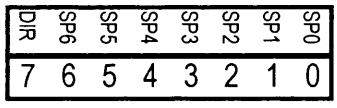


Figure 4.14 Set pump speed entry.

Name	Settings	Description
DIR	0 = forward (default) 1 = reverse	Pump direction.
SP6: SP5: SP4: SP3: SP2: SP1: SP0	0x00 = minimum 0x7F = maximum (default)	Pump Speed.

Table 4.11 Detailed pump speed control register configurations.

4.2.3.8 Valve Position Entry [0x2008]

This is a read only entry containing the physical position of the valve.

Valve Position	Parameter Value
Closed	0x00
Open to inlet	0x01
Open to Outlet	0x02
Unknown	0x03

Table 4.12 Valve position entry codes.

4.2.3.9 Delta T Entry [0x2009]

Sets the data acquisition system (DAQS) sampling rate between 1 and 255 seconds. A zero value in this field disables sampling.

4.2.3.10 Battery Voltage Entry [0x200A]

This entry contains a single byte value representing the voltage across the battery terminals. This gives a basic indication of the state of the internal battery pack in a node.

4.2.3.11 RS-232 Baudrate Entry [0x200B]

Legacy RS-232 UART allows direct connection of any node to a host PC for development and diagnostics. Communication rates of 4800 and 9600 and 19,200 baud are supported.

4.2.3.12 Count POR (power-on reset) [0x200C]

This register entry keeps track of occurrences of power-on/reset events. This is useful for diagnostic purposes to detect if the node systems are being inadvertently re-initialised, e.g. by an intermittent battery connector fault.

4.2.3.13 Data Representation Entry [0x200D]

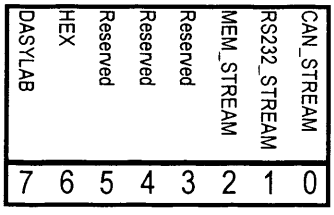


Figure 4.15 Data representation entry format.

Name	Settings	Description
DASYLAB	0 = hexadecimal (default) 1 = DASYLab	To maintain compatibility with DASYLab: 1. Add channel number to allow DASYLab to identify the channels. 2. Pad with an extra padding byte so sample is of type UNSIGNED32 (32-bits). 3. Store "ittle endian" format.
HEX	0 = text (default) 1 = hexadecimal	Data format is unprocessed hexadecimal or human readable text.
MEM_STREAM	0 = off (default) 1 = on	Stream data to SD/MMC memory storage media.
RS232_STREAM	0 = off (default) 1 = on	Stream data to RS-232 port.
CAN_STREAM	0 = off (default) 1 = on	Stream data to CAN bus.

Table 4.13 Detailed pump speed control register configurations.

4.2.3.14 Measurement Entries [0x200E, 0x200F, 0x2010]

Each node functions as a "smart transducer" producing fully calibrated flow velocity and temperature measurements from raw hexadecimal data and presenting these values as entries in the Object Dictionary where it can then be consumed by other nodes. For example, the pump node can use data from fluid flow velocity entry generated by the sensor node to monitor fluid flow rate and adjust its speed appropriately.

4.2.3.15 Calibration Entries [0x2010, 0x07]

A temperature sensor node is capable of calculating and exporting fully calibrated temperature measurements onto the network for consumption by pump or host nodes. The first stage of the calibration procedure is to convert the ADC output to a real voltage (V_{out}) using the following formula:

$$V_{out} = V_{ref} \cdot value / resolution \tag{4.20}$$

Where:

$$resolution = 2^{number\ of\ bits\ in\ ADC} = 2^{24}$$

$$V_{ref} = ADC\ reference\ voltage = 1.000\ V$$

The actual voltage input (V_{in}) to the ADC can now be calculated using:

$$V_{in} = V_{out} / gain \tag{4.21}$$

Where:

gain = ADC can be configured to amplify a signal by 1, 2, 4, 8, 16, 32, 64 or 128 times

The next stage is to convert the voltage into the physical resistance of the RTD measuring device:

$$r = Vin / RTD\ excitation\ current \tag{4.22}$$

Where:

RTD excitation current = 0.0002 A

r = RTD resistance in Ω

Finally, the polynomial to convert the ADC output voltage to an accurate calibrated temperature reading.

$$temperature\ (°C) = r\ (C1 \cdot r^2 + C2 \cdot r + C3) - C4 \tag{4.23}$$

The coefficients have the following values:

C1 = 1.13287E-06

C2 = 6.12723E-04

C3 = 2.39747

C4 = 247.0073

This polynomial was developed specifically for the application of temperature measurement in hydrothermal vent environments and its accuracy is greater than ±0.1°C within a temperature range of -50°C to 500°C.

All entries REAL32 data types conform to IEEE-754 32-bit single precision floating point format as shown below:

Sign	Exponent								Fraction																	
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 4.16 Format of entries containing REAL32 data types.

, where the most significant bit represents the sign; bits 30 to 23 represent an 8-bit exponent which is stored as excess 127 i.e. the exponent is biased by $2^{e-1} - 1$; and bits 22 to 0 represent the mantissa minus the most significant bit i.e. there is an implied bit to the left of the radix which is always one except for a zero value. The actual floating point value is calculated from the following formula:

$$(-1)^{sign} \cdot 2^{(exponent - 127)} \cdot 1.mantissa$$

So, as an example, the parameter value for entry [0x2010, 0x07], which represents RTD excitation current, can be calculated manually as follows:

The sign bit is zero; the biased exponent is 114, so the exponent is 114 – 127 = -13. Take the binary number to the right of the decimal point in the mantissa, convert this to decimal and divide by 2^{23}, where 23 is the number of bits taken up by the mantissa, to give 0.6384000 and then add 1 to this fraction. The floating-point value is given by:

$$(-1)^{0} \cdot 2^{-13} \cdot 1.6384000 = 0.0002$$

Working backwards from this result is more involved, however there are numerous online tools available that can perform the task of decimal floating point to 32-bit and 64-bit IEEE-754 hexadecimal representations along with their binary equivalents.

4.2.4 Service Data Objects (SDO)

An SDO message consists of a COB-ID and data payload containing eight bytes as shown below.

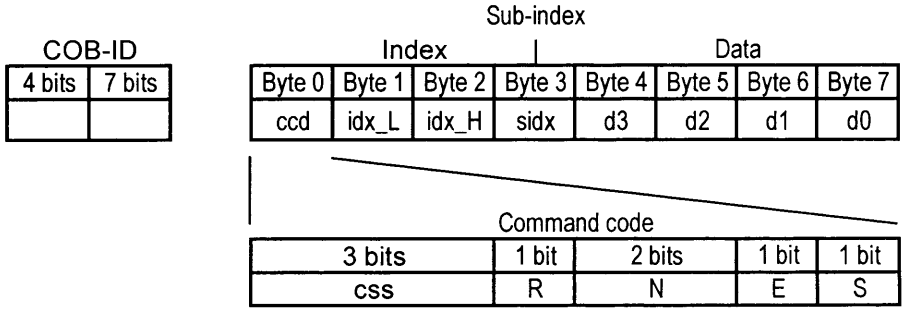


Figure 4.17 Generalised structure of an SDO message.

The data payload itself consists of:

- The command code (ccd), in which the SDO message type and length of the data value being transmitted are encrypted.
- The index and sub-index which point to the object whose data are being transported in the SDO message.
- From one up to four data bytes.

The ccd can be further reduced into the following component parts

- css (ccd<7:5>) is the client command specifier of the SDO transfer, this is 0 for SDO segment download, 1 for initiating download, 2 for initiating upload, 3 for SDO segment upload and 4 for aborting an SDO transfer.
- R (ccd<4>), reserved bit. Always set to 0.
- N (ccd<3:2>) is the number of bytes in the data part of the message which do not contain data, only valid if e and s are set. So, 0b00 = 4 bytes, 0b01 = 3 bytes, 0b10 = 2 bytes and 0b11 = 1 byte represent data payloads.
- E (ccd<1>), if this bit is set, indicates an expedited transfer, i.e. all data exchanged is contained within the message. If this bit is cleared then the message is a segmented transfer where data does not fit into one message and multiple messages are used.
- S (ccd<0>), if set, indicates that the data set size is specified in "N" (if "E" is set) or in the data part of the message.

The following, is a specific application example showing how to calculate the COB_ID and data payload field in a TSDO that writes system time data into the object dictionary of node # 2 on the system network. The time is represented by three data bytes (seconds, minutes and hours) and is stored in binary coded decimal format. The COB-ID is calculated as 0x582 *[described in section 4.2.5.2 Assigning CAN Message Identifiers].* The hexadecimal value for ccd is calculated by bit-shifting and then performing an OR operation on its constituent bit fields as follows:

css (ccd<7:5>)	*= 0b**001**00000*	*(1 decimal for initiating download)*
R (ccd<4>)	*= 0b000**0**0000*	*(not used)*
N (ccd<3:2>)	*= 0b0000**01**00*	*(3 byte data payload)*
E (ccd<1>)	*= 0b000000**1**0*	*(transfer is expedited)*
S (ccd<0>)	*= 0b0000000**1***	*(size of data payload is*

represented by "N")

Therefore:

$$ccd = 0b00100000 \mid 0b00000000 \mid 0b00000100 \mid 0b00000010 \mid 0b00000001$$
$$= 0b01000111$$
$$= 0x27$$

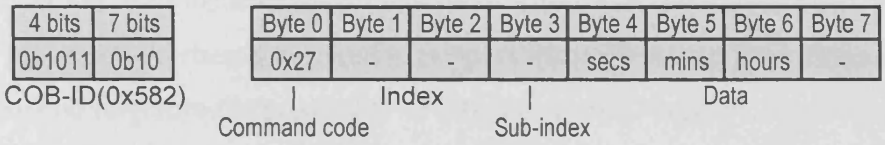


Figure 4.18 TSDO Configuration.

Index and data payload is stored "Little Endian" format where the low value byte is transmitted first meaning that the data must be packed and unpacked byte by byte before and after transmission *[figure 4.19]*.

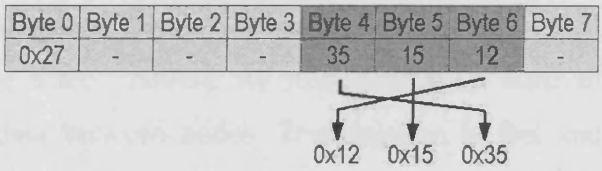


Figure 4.19 Conversion of numerical values greater than one byte.

Service data objects (SDOs) allow access to entries in the object dictionary through the index and sub-index. Each node not only implements it own object dictionary, it also implements a server that handles read and write requests of parameter values to the object dictionary. The SDO is typically used by a master node or configuration tool which acts as client. For example, the current time and date parameter values for a specific node or all nodes on the network can be configured before instrument deployment with a configuration tool running on a host PC. The client-server communication is initiated by the client in order to transfer parameter values to the server or fetch them from the server. In both cases, the client initiates communication and receives a response from the server. Data are transmitted by means of a TSDO in the client or server-SDO, and received by means of a RSDO.

SDOs have a higher COB-ID (the CAN identifier arbitration field) than PDOs and are therefore transmitted on the network with a lower priority i.e. if there is a collision between SDO and PDO messages on the network, the PDO will reach its destination

and the SDO will be destroyed and then retransmitted as the PDO has more dominant bits in its COB-ID field.

SDOs are not essential to the operation of the instrument, i.e. there are no requirements for dynamic reconfiguration of nodes during deployment, as the machine system behaviour is defined and fixed in the pre-compilation development phase. For this reason no effort has been made to implement them at the node level because of the large coding and processor overheads required to support them. However, the CANopen is an open protocol and therefore the possibility of integrating SDO support, should there be a demand for more sophisticated run-time behaviour, is not precluded later on. The mechanism for the exchange of process variables on this project is built entirely on optimised, direct PDO linkings between nodes with message transmission (PDO production) being purely event triggered.

4.2.5 Process Data Objects (PDO)

PDOs provide a more direct method for communication than SDOs for real-time exchange of process data between nodes. Transmission is fast and efficient because there is no additional overhead of administration data and no response is required from the receiver. The flexible data payload length of a PDO message also facilitates conservation of network bandwidth and allows greater data throughput. The data payload is a maximum of eight bytes, however if only two bytes are required then only two bytes are transmitted.

Data exchange with PDOs is based on a producer-consumer model (Foster 2002) and can be time or event triggered where a distinction is made between transmit process data objects (TPDOs) and receive process data objects (RPDOs). This terminology indicates whether a PDO is produced or consumed by a node. For each PDO in the system there is one node producing it (a TPDO) and at one or more nodes consuming it (an RPDO) as illustrated in *figure 4.20* below.

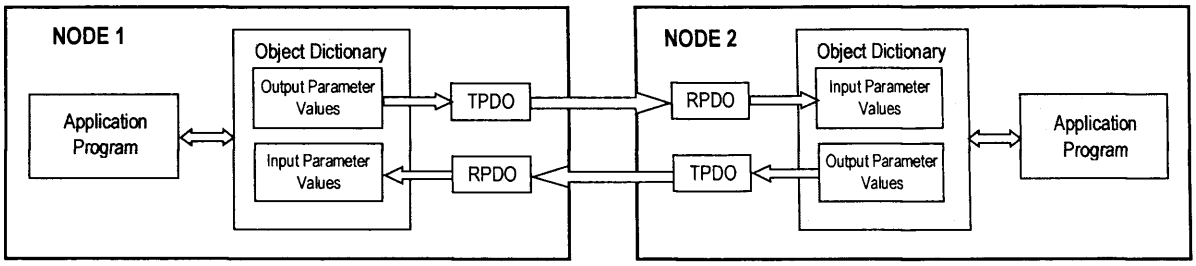


Figure 4.20 PDO communication model.

4.2.5.1 PDO Linking

In order to establish common ground for communication a default state for usage of the message identifiers is typically implemented. This is referred to as the pre-defined connection set and determines which COB-IDs should be used by which node by default (Pfeiffer 2003). The simple four node system outlined in ***Chapter 2*** is used to exemplify how PDO linking can be established between nodes to form a small group of communication partners that can operate collectively to perform the task of acquiring a fluid sample.

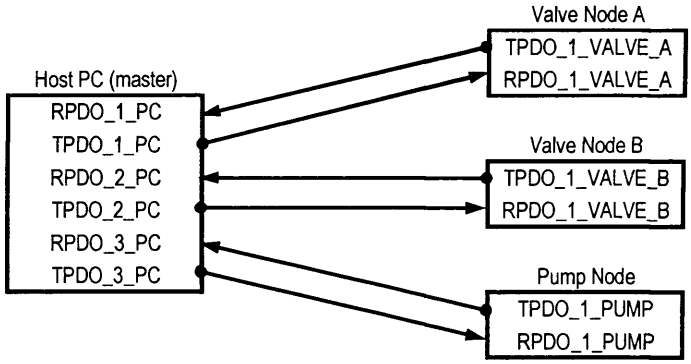


Figure 4.21 Default PDO linking for a four node system (Master-Slave model).

In this design topology there is no overlap of any specified TPDOs with RPDOs, that is, by default, none of the RPDOs utilise the same COB-ID as the TPDOs and therefore no PDO is directly linked. Only the master is able to receive all the TPDOs and only the master may generate the RPDOs to the slave nodes. Control of all the slave nodes is centralised to the host PC node (master) which bears the main computational burden and is responsible for ensuring the correct control sequencing of opening/closing both valves and switching the pump on and off to fulfill the task of successfully acquiring a fluid sample.

The centralised communication model described is not an optimal solution for this application for reasons discussed in ***Chapter 3***. However, because CAN is a multicast protocol, where any node can send/receive a message at any time, it is possible to configure TPDOs and RPDOs to share the same COB-ID. Direct links can therefore be established between nodes to create a more decentralised communication topology as shown below.

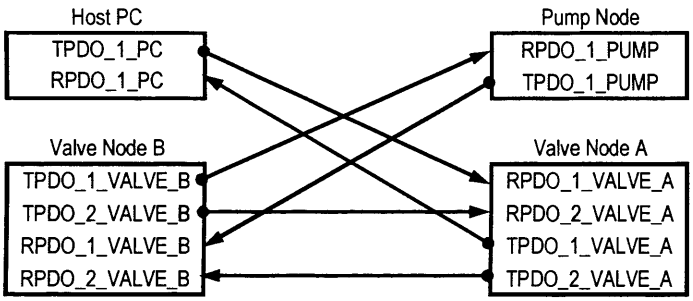


Figure 4.22 Optimised, direct PDO linking for a four node system.

From this diagram, it is apparent that the TPDOs and RPDOs are more uniformly distributed between nodes, which implies that the computational burden will also be more evenly spread throughout the system network. In this topology the host PC is used as a configuration tool to initialise the system and can be removed when the system is being deployed. The two valve nodes and pump node are communication partners. Valve Node A utilises its internal RTC to keep track of time and wait until the preset alarm time is reached (configured via the host PC). Upon the alarm being triggered Valve Node A opens and simultaneously sends a TPDO to Valve Node B commanding it to open as well. Valve node B then sends a TPDO to the pump node commanding it to turn on. Valve Node B then waits for a default preset time (approximately 5 minutes duration) before sending a TPDO to the pump node commanding it turn off. Finally, Valve Node B closes, completing the fluid sampling cycle.

This approach has several other advantages. As described earlier, the host PC node is only required during the configuration phase and can be removed after initialisation is completed. In this scenario, the grouping of two valve nodes and pump node to operate autonomously as a tightly knit unit of communication partners. This optimised, direct PDO linking topology is highly scaleable as behaviour of nodes in the system is

partitioned into groups that are virtually independent of each other for operation of the overall system.

To illustrate this point further, *figure 4.23* shows a seven node implementation based on a default PDO linking. The computational burden placed on the host PC node is now considerably greater and, in fact, will continue to increase approximately in proportion with the addition of nodes to the system network. The host node will require more resources in terms of processor power, number of lines of code and power consumption, to meet the increasing demands of new nodes as they are introduced to the network.

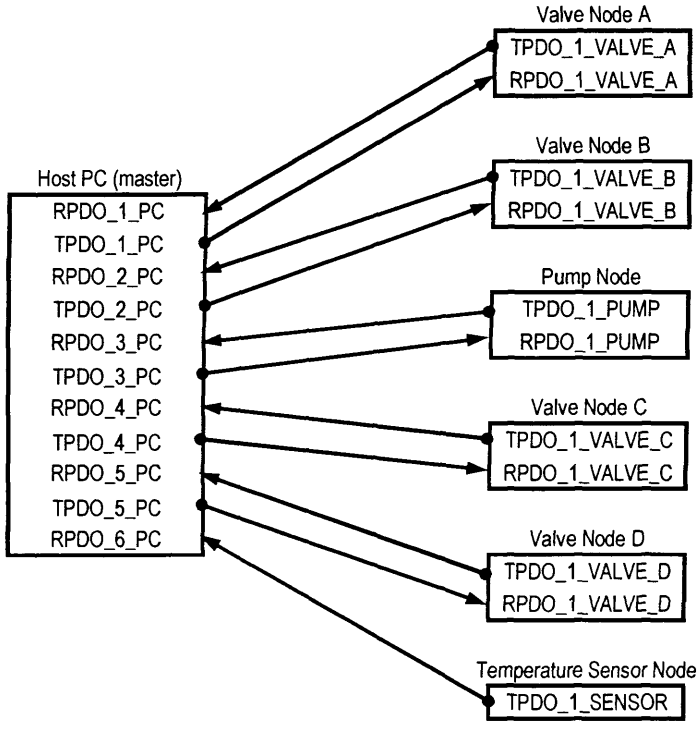


Figure 4.23 Default PDO linking for NASA seven node system (Master-Slave model).

Figure 4.24 below shows an optimised PDO linking, where again the computational burden is more uniformly distributed across the network.

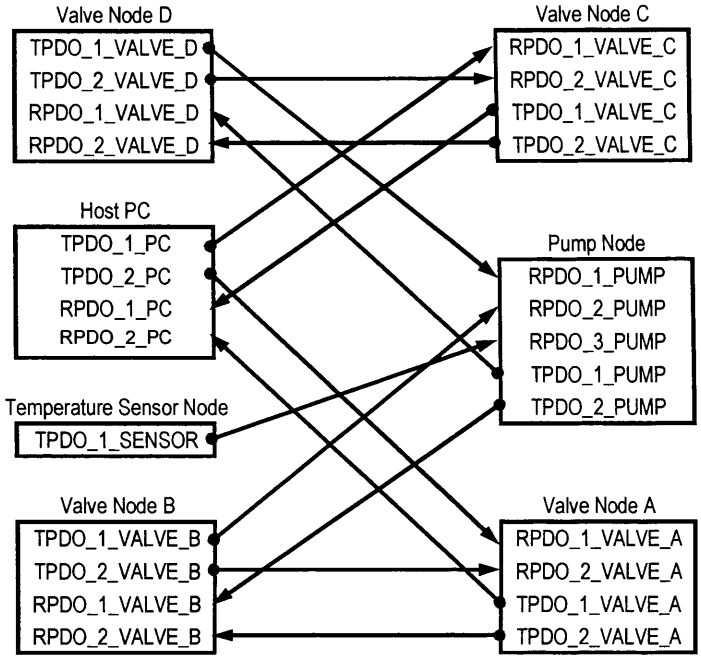


Figure 4.24 Optimised, direct PDO linking for NASA seven node system.

It is possible for system integrators, to further improve the reliability of the machine system by introducing additional nodes onto the network therefore adding additional redundancy. For example another pump node could be added to the network. Valve node are configured to receive a RPDO containing the status of the pump node and, if an error is detected, they can act on it to shutdown the primary pump node and default to using the secondary pump node.

Now that relationships have been clearly defined for communication between nodes on the system network it is possible to move forward and begin assigning CAN message identifiers (and priorities) to further elucidate their linkages with one another.

4.2.5.2 Assigning CAN Message Identifiers

For work with CANopen objects and data exchange, the CAN message can be expressed in a more simplified form, as many of the bits are used only by data link layer in the CAN controller hardware to ensure that data transmission is error-free. These bits are automatically inserted/removed from transmitted/received messages by the data link layer. The "Identifier" and "Data" bit fields constitute the simplified CANopen message. The "Identifier" corresponds to COB-ID (communication object identifier) and the "Data" field to the data payload of a CANopen message. The COB-ID defines transmission priority and identifies the communication object.

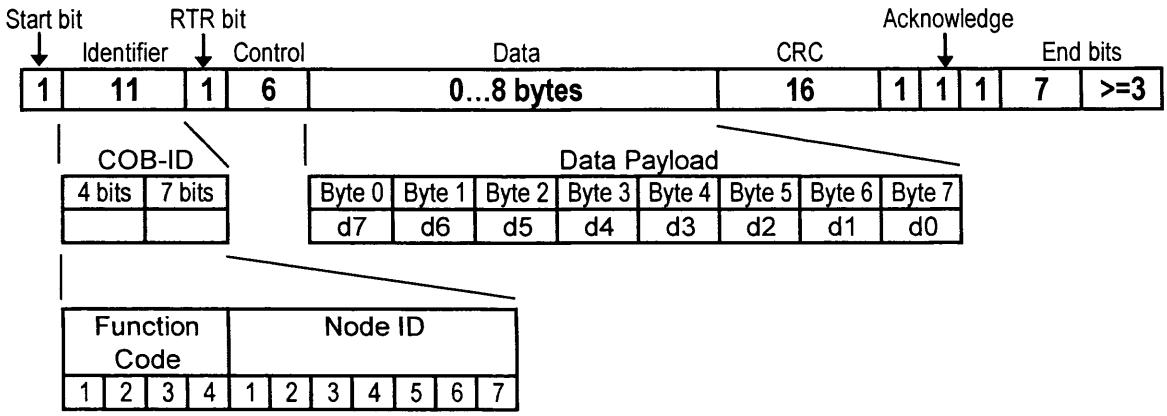


Figure 4.25 Structure of CANopen message.

Communication objects are classified according to their function code which also controls the message priority. Objects with a low value function code are assigned high priority in transmission, e.g. an object with the function code '1' will be transmitted before an object with function code '3', if both access the bus simultaneously.

All nodes are assigned a unique, 7-bit address (Node-ID) between 1 and 127 at compile time. Address '0' is reserved for broadcast transmissions, in which messages are sent to all nodes on the system network.

The table below shows the COB-IDs for CANopen communication objects, including the transmit process data objects (TPDOs), receive process data objects (RPDOs) and network management services (NMTs) used to build this system network.

Communication Object	Function Code	Node Address (Node-ID)	COB-ID (hex)	Slave Nodes
NMT node control	0000	0000000	0x000	Receive only
Sync	0001	0000000	0x080	Receive only
Emergency	0001	XXXXXXX	0x080 + NodeID	Transmit
PDOs	0011	XXXXXXX	0x180 + NodeID	TPDO_1
	0100	XXXXXXX	0x200 + NodeID	RPDO_1
	0101	XXXXXXX	0x280 + NodeID	TPDO_2
	0110	XXXXXXX	0x300 + NodeID	RPDO_2
	0111	XXXXXXX	0x380 + NodeID	TPDO_3
	1000	XXXXXXX	0x400 + NodeID	RPDO_3
	1001	XXXXXXX	0x480 + NodeID	TPDO_4
	1010	XXXXXXX	0x500 + NodeID	RPDO_4
SDOs	1011	XXXXXXX	0x580 + NodeID	Transmit
	1100	XXXXXXX	0x600 + NodeID	Receive
NMT node monitoring (heartbeat)	1110	XXXXXXX	0x700 + NodeID	Transmit
NMT Services	1111	110XXXX	-	-

Table 4.14 COB-IDs.

If number values are assigned to the nodes as shown below:

Node #	Description
1	Valve Node A
2	Valve Node C
3	Valve Node B
4	Valve Node D
5	Pump Node
6	Temperature Sensor Node
7	Host PC

, then the COB-ID for the TPDO_1 of Valve Node C (node # 2) can be calculated as follows:

$$NODE\_ADDRESS\_TPDO\_1 + NODE\text{-}ID = 0x180 + 2 = 0x182$$

The default PDO linkings for the NASA seven-node machine system can now be methodically calculated and COB-IDs assigned as shown in *figure 4.26.*

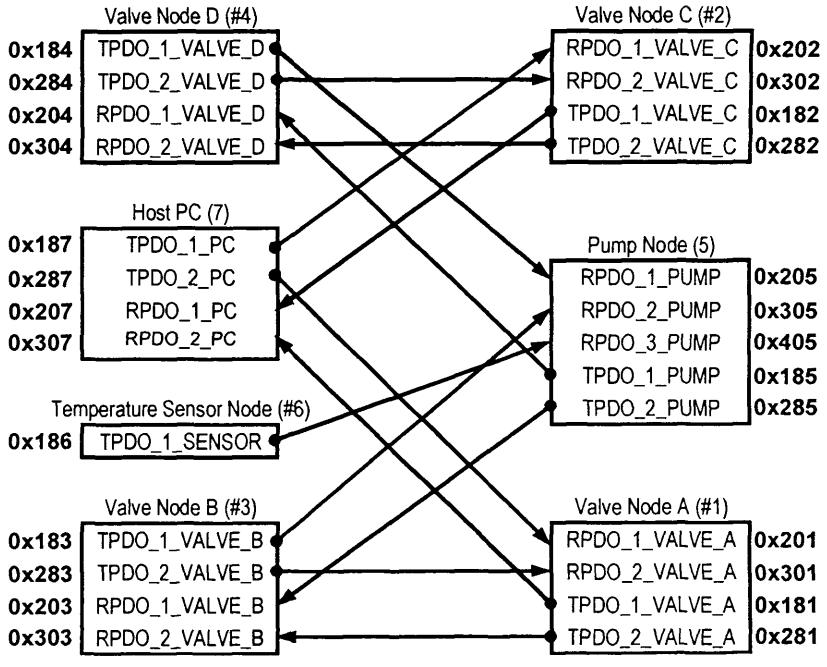


Figure 4.26 COB-IDs assigned to PDOs in 7 node NASA system for default linkage.

There is no overlap of any TPDOs and RPDOs in this default connection, which implies there no specified RPDO uses the same identifier as any TPDO and therefore no PDO is

directly linked. The efficiency and reliability of default PDO linkages described above can be improved by further optimisation to allow nodes to directly listen to the process data message they need to receive to perform their work. For example, RPDO_2_VALVE_B of node #3 is configured to directly consume TPDO_2_VALVE_A of node #1 by changing its COB-ID from 0x303 (default receive ID for RPDO_2_VALVE_B of node #3) to 0x281 (transmit ID for TPDO_2_VALVE_A of node #1). Making changes to the appropriate RPDOs in the whole system yields a more optimal solution for communication as shown in *figure 4.27*.

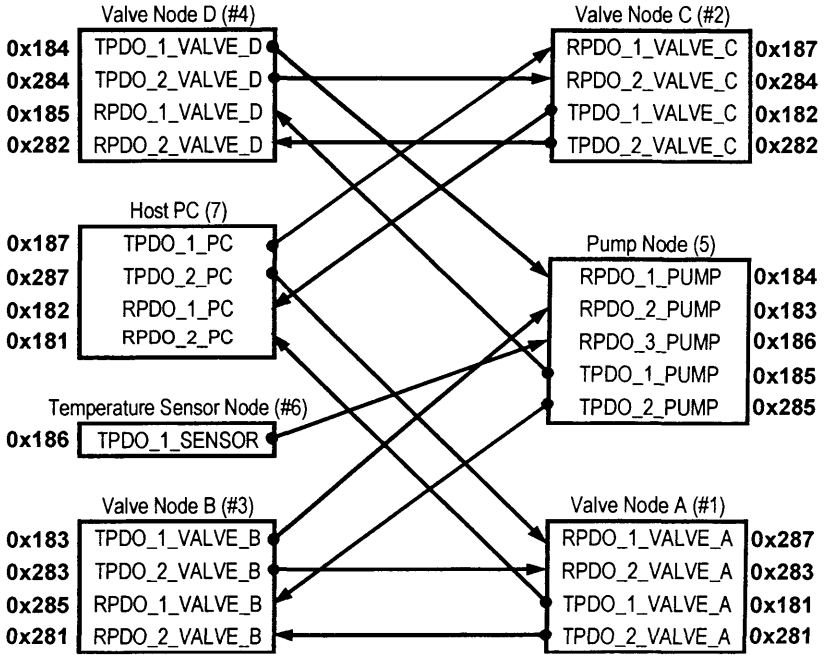


Figure 4.27 COB-IDs assigned to PDOs in 7 node NASA system for direct linkage.

Now that CAN identifiers are assigned, the next stage of the communication definition is to create mappings for the appropriate process data from the individual Object Dictionaries of nodes onto the message contents of their defined PDOs. The setup of the communication and mapping parameters for the PDO mappings are located in the Object Dictionary as outlined in the following two sections.

4.2.5.3 PDO Communication Parameters

In the Object Dictionary the index range from 0x1400 to 0x15FF is reserved for RPDO communication parameters and the range from 0x1800 to 0x19FF for TPDO communication parameters. These ranges allow a possible maximum of 512 (0x200) RPDOs and TPDOs can be configured in the Object Dictionary of a single node. The

parameters for the first RPDO (RPDO_1) are located at index 0x1400, the second at 0x1401 (RPDO2), the third at 0x1402 (RPDO3), etc and the parameters for the first TPDO (TPDO_1) are located at index 0x1800, the second at 0x1801 (TPDO2), the third at 0x1802 (TPDO3) and so on.

The parameters for RPDOs and TPDOs are accessed via a sub-index as shown in the table below:

Sub-index	Description	Data type
0	Number of entries	UNSIGNED8
1	COB-ID	UNSIGNED32
2	Transmission type	UNSIGNED8
3	Inhibit time	UNSIGNED16
4	Reserve	UNSIGNED8
5	Event time	UNSIGNED16

Table 4.15 RPDO and TPDO Communication Parameters.

It can be seen that there are a maximum of five entries for a PDO that can be supported. It is typical to implement the first two entries for an RPDO and in this application "Number of entries" is set to a value of 0x02 (there are two entries). The COB-ID entry is calculated by selecting the desired PDO from *figure 4.27* and setting the appropriate bits from the bit field below.

A	B	C	29-bit identifier MSbits (unused)															COB-ID										
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figure 4.28 COB-ID configuration

Where bit "A" is cleared to '0' to select an 11-bit COB-ID, bit "B" is set to '1' to indicate remote transmit requests are not allowed for the PDO and bit "C" is cleared to indicate that the node utlises this PDO, so the bit field (most significant 3 bits) has a value of 0b01000000000000000000000000000000 = 0x04000. As an example, the COB-ID entry for TPDO_1_VALVE_A (0x181) is configured by performing an OR operation with this field to yield a value of, 0x4000 | 0x181 = 0x4181.

For a TPDO one extra parameter, defined by the "Transmission type" entry, is required to specify how the message transmission is triggered. So "Number of entries" is set to a value of 0x03 and again the COB-ID value is taken from *figure 4.27*. A decision was made to base inter-node message transmission on only event driven (COS, change of state) trigger methods and not to implement time, polled and synchronised group polling

methods. The defined PDO linkings rely strictly on event triggered PDO transmission and there is no requirement to utilise the other types of communication methods supported by CANopen. Constraining communication behaviour in this way, with all nodes adopting the same method, reduces system complexity and aids in minimising the length of final system testing once nodes are integrated into the network. Ultimately, this will allow system developers to focus all test procedures on the chosen communication method, avoiding the additional test procedures required if multiple methods are mixed on one network (Pfeiffer 2003). Finally, in accordance with CANopen specification, the "Transmission type" entry for a TPDO is set to a value of 0xFE to indicate that the conditions that trigger message transmission are "manufacturer specific" events, for example, an alarm event from real-time clock hardware.

4.2.5.4 PDO Mapping Parameters

Within the Object Dictionary, the index range from 0x1600 to 0x17FF is reserved for the RPDO mapping parameters, and the area from 0x1A00 to 0x1BFF is reserved for TPDO mapping parameters. The communication parameter index range size is identical to the mapping parameter index range, which ensures straightforward correlation of a given PDOs' mapping and communication parameter values. The CANopen PDO mapping parameters determine which Object Dictionary entries are mapped or placed within a TPDO or RPDO. These mappings allow PDO contents to be customised to meet the specific communication requirements of the machine system so that it can perform its tasks efficiently. The mappings provide substantial flexibility in that they can be either dynamic or static, where dynamic mappings can be re-configured during runtime operation by using a system configuration tool or master. However, for this type of deeply embedded application, where reliability is paramount, this kind of flexibility is considered an unnecessary risk factor as it opens up possibilities for inadvertent changes to be made to CAN message structure that could result in a catastrophic system failure. In this case, the additional CANopen flexibility is a potential liability and therefore static PDO mappings are implemented by hard-coding them into the firmware of each node. This introduces a degree of protection because it constrains the machine system from the possibility of incorrect reconfiguration during runtime operation. A further benefit of static PDO mapping is that fewer resources in terms of microprocessor time, code and data space are required for its implementation.

The CANopen PDO mappings are designed to work with Object Dictionary data at both the byte and bit level, making it possible to map single data bits into a CAN message up to a maximum of 64 bits per data payload. There is a processing penalty associated when storing data in bit level format because of the additional overhead of having to pack/unpack each bit in/out of each data byte. Despite this, extensive use is made of bit level data is made here as many of the hardware devices within each node implement initialisation registers that operate at this level.

A PDO mapping parameter associates a specific Object Dictionary entry with its parameters index, sub-index and length (in bits). These three parameters are coded into one 32-bit value as shown below.

Index	Sub-index	Length (bits)
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0

Figure 4.29 Structure of a 32-bit Mapping Parameter.

A PDO mapping is composed of one or more of these mapping parameters. The 64-bit PDO is filled entry-by-entry with the data from the Object Dictionary entries specified in the single mapping parameters as shown below.

Sub-index	Description	Data type
0	Number of entries	UNSIGNED8
1	1st OD entry mapped	UNSIGNED32
2	2nd OD entry mapped	UNSIGNED32
3	3rd OD entry mapped	UNSIGNED32
-	-	-
64	64th OD entry mapped	UNSIGNED32

Table 4.16 Format of a PDO Mapping Record.

4.2.5.5 Message Contents

Now that the concepts of PDO mapping and communication parameters are established, it is possible to proceed and associate or map the required data parameters from the Object Dictionary entries into the PDOs to define the CAN message contents. No formal methodologies or smart analytical techniques presently exist for doing this, and the only approach to define PDOs is that of iterative refinement. One of the challenges is to decide which process variables can be best combines within one PDO. Typically, one would send command bytes from one node to several others. A decision must be

made whether to send the bytes one by one i.e. one separate PDO for each receiving device or should they still be combined into one PDO and sent together. The latter approach optimises the use of available network bandwidth, however has the disadvantage that each receiving node receives data that it does not need. In general, the benefit of bandwidth optimisation outweighs the disadvantage of handling some additional unnecessary data. CANopen supports receiving such unwanted data by using dummy mapping. A receiver may map unwanted data of an RPDO directly to so-called dummy entries, meaning the unwanted data is ignored. Mappings for Node #1 (Valve A) are shown below and a complete set of mapping definitions for the NASA 7-node instrument are given in ***Appendix J.***

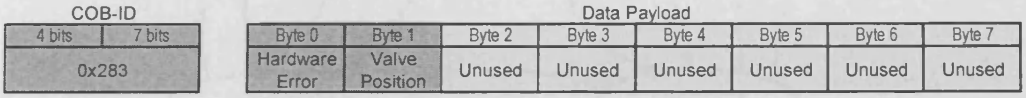


Figure 4.30 Message contents for RPDO_1_VALVE_A.

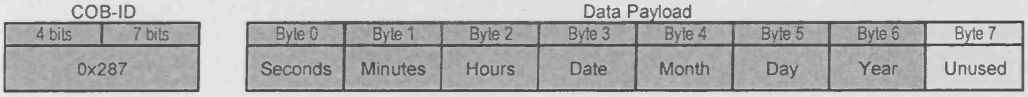


Figure 4.31 Message contents for RPDO_2_VALVE_A.

Index	Sub-index	Description	Data type	Attribute	Default Value
0x1400	-	RPDO_1_VALVE_A communication parameters	RECORD	-	-
0x1400	0x00	Number of entries	UNSIGNED8	CONST	0x02
0x1400	0x01	COB-ID (= 0x4000 | 0x283)	UNSIGNED32	RO	0x4283
0x1400	0x02	Transmission type	UNSIGNED8	RO	0xFF
0x1401	-	RPDO_2_VALVE_A communication parameters	RECORD	-	-
0x1401	0x00	Number of entries	UNSIGNED8	CONST	0x02
0x1401	0x01	COB-ID (= 0x4000 | 0x287)	UNSIGNED32	RO	0x4287
0x1401	0x02	Transmission type	UNSIGNED8	RO	0xFF
0x1600	-	RPDO_1_VALVE_A mapping	RECORD	-	-
0x1600	0x00	Number of mapped entries	UNSIGNED8	CONST	0x02
0x1600	0x01	PDO mapping (Hardware error = 0x2004 0x00 0x08)	UNSIGNED32	RO	0x20040008
0x1600	0x02	PDO mapping (Valve Position = 0x2008 0x00 0x08)	UNSIGNED32	RO	0x20080008
0x1601	-	RPDO_2_VALVE_A mapping	RECORD	-	-
0x1601	0x00	Number of mapped entries	UNSIGNED8	CONST	0x07
0x1601	0x01	PDO mapping (RTC seconds = 0x2000 0x01 0x08)	UNSIGNED32	R0	0x20000108
0x1601	0x02	PDO mapping (RTC minutes = 0x2000 0x02 0x08)	UNSIGNED32	R0	0x20000208
0x1601	0x03	PDO mapping (RTC hours = 0x2000 0x03 0x08)	UNSIGNED32	RO	0x20000308
0x1601	0x04	PDO mapping (RTC date = 0x2000 0x04 0x08)	UNSIGNED32	RO	0x20000408
0x1601	0x05	PDO mapping (RTC month = 0x2000 0x05 0x08)	UNSIGNED32	RO	0x20000508
0x1601	0x06	PDO mapping (RTC day = 0x2000 0x06 0x08)	UNSIGNED32	RO	0x20000608
0x1601	0x07	PDO mapping (RTC year = 0x2000 0x07 0x08)	UNSIGNED32	RO	0x20000708

Table 4.17 RPDO mappings for Node #1 (Valve A).

4.2.6 Network Management (NMT)

Each node in the network system implements a NMT state machine which allows it to exist in different operating states *[figure 4.32]*. Some of these state transitions can be made autonomously by the node by itself and others can only be made on receiving a control message from a NMT master node. The NMT master message can be targeted at

an individual node or broadcast to all nodes on the network simultaneously and contains the new state that the node(s) should switch to.

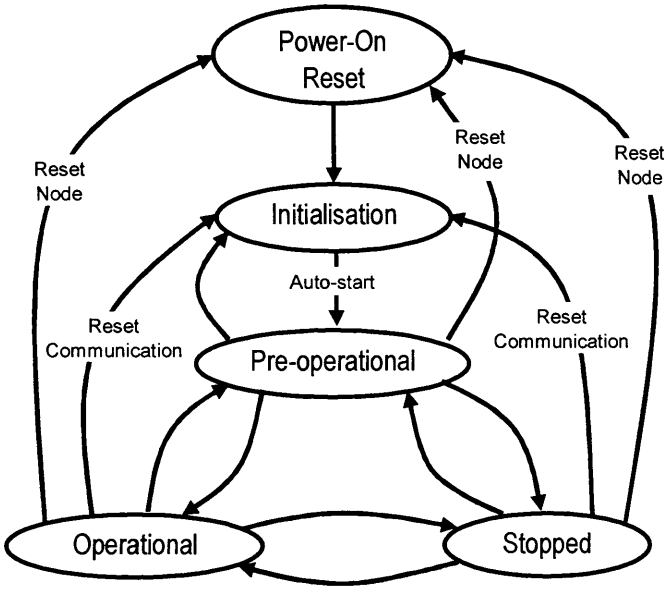


Figure 4.32 Modified node network management (NMT) state machine.

A NMT master node can switch nodes back and forth between the three major states: pre-operational, operational and stopped. In the pre-operational state a node participates in all communication related to SDOs, emergencies, timestamps and heartbeats. The operational mode adds PDO communication, allowing the node to exchange and work with process data. In the stopped state a node stops all communication, except for minimal NMT services.

The NMT master can also request two different reset actions. Upon receiving the "Reset Communication" command a node will reset the CAN/CANopen interfaces, switching the node into its initialisation state where it transmits a boot-up message. In this state the node does not consume any messages. The "Reset Node" command effectively re-boots the node sub-systems by forcing a reset of its hardware, all peripherals and firmware.

The DS301 CANopen specification does not provide a mechanism for a node to switch from the pre-operational to operational state without waiting for a message from a NMT master. This is problem for deeply embedded networks, such as this one where it is not possible for the NMT master to gain access to the system. One solution is to implement a minimal NMT master node that broadcasts the NMT "Go Operational" message to get the network system up and running. A decision has been made to ignore this aspect the

standard by allowing nodes to auto-start. This allows the machine system to operate autonomously without the presence of a NMT master in deployment situations. Control is distributed as evenly as possible throughout the nodes on the network in an effort to maximise reliability. There are, however two scenarios where a NMT master is advantageous and even essential for operation, for example the NMT master could act as a sentinel, watching over all nodes to check they remain within operational parameters to improve the ability of the system to cope with mission threatening scenarios. If a node fails or an alarm/emergency message is received it can initiate the appropriate recovery or shutdown procedures. Also, a host PC NMT master is required to initialise certain parameters during startup such as times for specific action to be triggered.

4.2.7 Heartbeat

All nodes advertise their presence on the system network by periodically transmitting a heartbeat which consists of a single byte CAN message containing their current NMT state. A decision was made to implement heartbeat rather than node guarding services to decentralise "plug-and-play" functionality. With node guarding it is the NMT master's responsibility to poll ("guard") all nodes for their current NMT state information and if a node does not respond within a specified time, the NMT master can take appropriate action. The NMT master is essential and its failure would result in the entire machine system ceasing to function. This is not the case with the heartbeat method, as no single node is essential to the continued operation of the heartbeat mechanism. Another advantage of the heartbeat method over node guarding is that the network bandwidth required for monitoring is halved as no polling is required.

The heartbeat producer time is set by configuring OD entry [0x1017, 0x00]; its value is a 16-bit integer and represents the heartbeat period in milliseconds. The heartbeat producer times of each node are set to individual prime number values to minimise the frequency of message collisions *[figure 4.33]*. Although the CAN data link layer is implemented in hardware and resolves such collisions automatically, it is good engineering practice to take steps avoid message collisions, wherever the opportunity arises to optimise available network bandwidth and improve system efficiency.

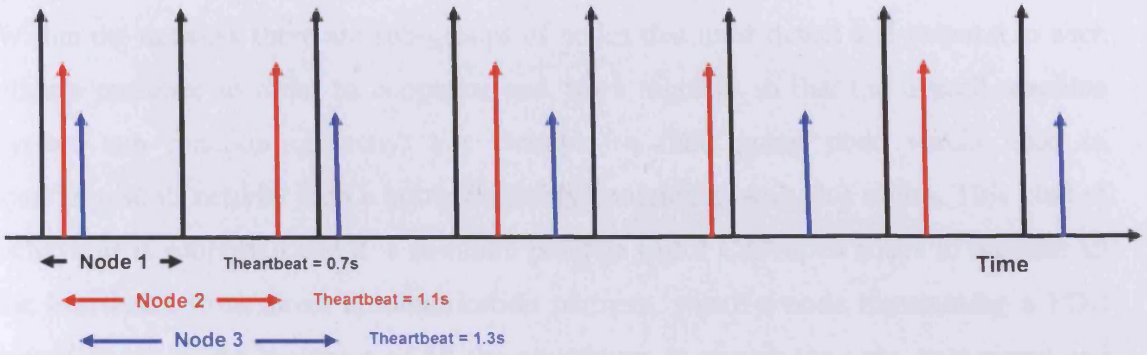


Figure 4.33 Heartbeat producer times for three nodes in the system network.

A nominal base heartbeat period in the region of one second is chosen, as it offers a reasonable compromise between system responsiveness in detecting addition/removal of nodes and maintaining the number of heartbeat messages on the network within manageable levels, so they are not consuming excessive bandwidth. All node operating systems implement a timer with 10 ms resolution, which puts a minimum limit on the granularity of heartbeat message transmissions.

Node #	Description	Heartbeat Frequency (s)
1	Valve Node A	1.01
2	Valve Node C	1.03
3	Valve Node B	1.07
4	Valve Node D	1.09
5	Pump Node	1.13
6	Temperature Sensor Node	1.27
7	Host PC	Not implemented

Table 4.18 Heartbeat periods for 7 node system developed for NASA.

Setting heartbeat producer times to prime number values will not necessarily eliminate heartbeat collisions completely because of imperfections in the node systems hardware and firmware. Inter-node resonator clock drift and the fact that the heartbeat message can be delayed while higher priority tasks take precedence will introduce "jitter" which will cause heartbeat producer times to vary form the ideal. This allows the possibility of collisions between producer heartbeat messages, however further measures can be taken to minimise this by spreading the heartbeat periods around i.e. selecting prime numbers that are further apart will decrease the frequency of message collisions.

Within the network there are sub-groups of nodes that must detect and respond to each other's presence in order to cooperate and work together so that the overall machine system can function correctly. For example, a fluid pump node would need to synchronise its activity with a bottle assembly containing two valve nodes. This kind of behaviour is supported and it is common practice to for CANopen nodes to monitor all the heartbeats from direct communication partners, where a node transmitting a PDO would listen to the heartbeat of all the consumers to ensure they are operational and therefore capable of processing the PDO (Pfeiffer 2003).

The seven node system for NASA contains several such sub-groups working as communication partners *[figure 4.27]*. In some cases there some overlap between sub-groups, for example the pump is shared between the two valve sub-groups, A + B and also forms a group with the temperature sensor node (sub-group C). Valve Node A requires the presence of Valve Node B and the Pump Node to fulfill its function and therefore monitors heartbeat messages generated by them to confirm they are operational. This also applies to Valve Node C and its communication partners, Valve Node D and the Pump Node. It is useful for the Pump Node to monitor fluid temperature and flow rate so that it can self-regulate its pumping rate, so that a known volume of fluid passes through bottle to ensure it is properly flushed and an uncontaminated sample is acquired.

To monitor the heartbeat of a node, the consumer's time is set in the range of 150% to 200% of the producer's time. The consumer's time functions as a time-out and the heartbeat is considered lost if it does not occur within the set time (OD entry [0x1016, 0xii]).

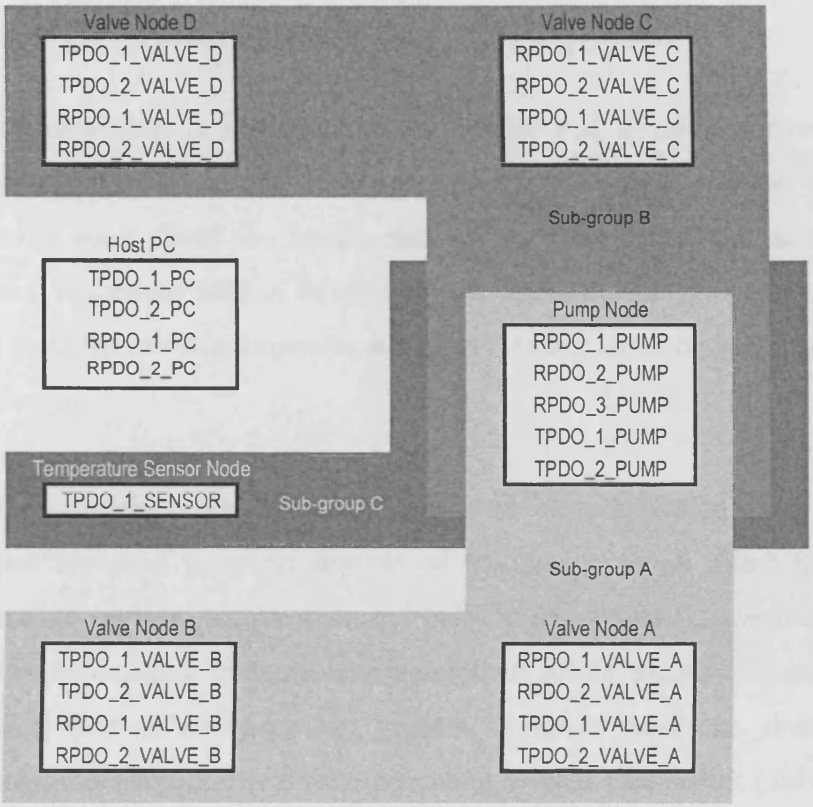


Figure 4.34 Overlap between communication partners.

Chapter 5 Firmware Architecture

An important objective of this thesis is to specify and develop a manageable and scalable instrument architecture, striving to create a non-obsolescent or "future-proof" design. In many ways, these two requirements are at odds with one another, creating a conflict, which makes the task of developing the firmware infrastructure a challenging undertaking that requires some consideration and forward planning in the early stages of the project design.

This chapter explains how top-level system decomposition and partitioning methodologies are used in many aspects of the design in an effort to "design-in" robustness into the system architecture and build maintainable firmware infrastructure. It then goes on to describe in detail how a commercially available, minimal CANopen implementation (MicroCANopen) and custom designed hardware abstraction layer (HAL) are integrated into the real-time operating system framework (Salvo RTOS) to realise a node system.

5.1 Design Considerations

Computer programming is a relatively new field and in many ways can still be regarded as being in its infancy. To quote H.M. Deitel (1999), "Just as architects design buildings by employing the collective wisdom of their profession, so should programmers design programs. Our field is younger than architecture is, and so our collective wisdom is considerably sparser.". However, despite its short history, much insight has been gained about structured programming and a substantial body of information and techniques are available to aid the system developer in producing programs that are easier to test, debug, modify and even prove correct in a mathematical sense. Several of these methodologies are employed here to impose some structure and order over the development process, including decomposition, a high degree of modularisation, extensive partitioning and standardisation within various levels of both the electronic (hardware) and code embedded within the hardware (firmware). It is useful to deconstruct node systems in this way during certain phases of its design process as they are entirely different disciplines, therefore this chapter is devoted to the firmware architecture and ***Chapter 6*** outlines the hardware design considerations.

The firmware is written in ANSI 'C' (Kerningham & Ritchie 1988) and adheres to standards documentation (Ganssle 2000) based on the established Borland Turbo 'C' Reference libraries style. Abstraction and partitioning are achieved using a hardware abstraction layer (HAL), commercial off the shelf real-time operating system (RTOS) and layered models. The functionality is described with system state diagrams, flow charts and textual descriptions. ***Appendix N*** contains a reference manual with detailed function descriptions, call graphs and the source code listing for this project can be found in ***Appendix O***.

5.1.1 More on Partitioning

Firmware complexity scales non-linearly with size (Ganssle 2000) and also arises from having a large number of dependencies between firmware modules. The Constructive Cost Model (COCOMO) is frequently used to estimate the effort it will take to develop a software product (Boehm 1981). The effort is measured in number of programmer-months and is proportional to the number of lines of code (NOC) in the project source files. To flatten the COCOMO complexity curve (Ganssle 2000) a high degree of partitioning (Holub 1995) is used here to minimise cross linkage (and therefore dependencies) in both firmware and hardware domains. The following partitioning techniques (Ganssle 2000) are used extensively in this project to flatten the complexity/size curve as far as possible:

- Partition by encapsulation. Functions are written to implement specific tasks. Where possible their size is limited to around 30 lines or so (enough to fill the text editor screen) which makes them easier to maintain and reuse. If a function gets substantially longer than this then it is possibly becoming over-complicated and it may be worth examining it to see if it can be broken down into two or more separate sub-functions.
- Partition by adding hardware. Three microprocessors are used on each node, as well as dedicated hardware blocks for communication, measurement, timing and control purposes *[see **Chapter 6** Electronic Hardware Platform].*
- Partition by using an RTOS. The commercial RTOS allows an application to be defined in terms of tasks that communicate with one another using defined messaging mechanisms.

- Partition by feature management and incremental development. The RTOS kernel encourages incremental development because features can be added incrementally to the application. A basic test harness was designed and installed on the target processor. This consisted of the RTOS kernel and tasks for LED control, serial communication with a host PC and kicking the watchdog. Once this was up and running, the system was built-up by incrementally adding drivers for hardware (ADC, RTC, etc) and testing as we went along.
- Finally, partition by top-down decomposition as described above. When there is cross linkage between modules, then interfaces are defined or established protocols are adopted wherever possible. Top-level system decomposition and tools like UML allow the system designer an overview and elucidate module inter-relationships and dependencies.

A more structured approach can now be taken to the design and organisation of the system architecture.

5.2 Node Infrastructure

5.2.1 Node Model

The node internal architecture closely follows the generalised model for a "smart transducer" as shown below.

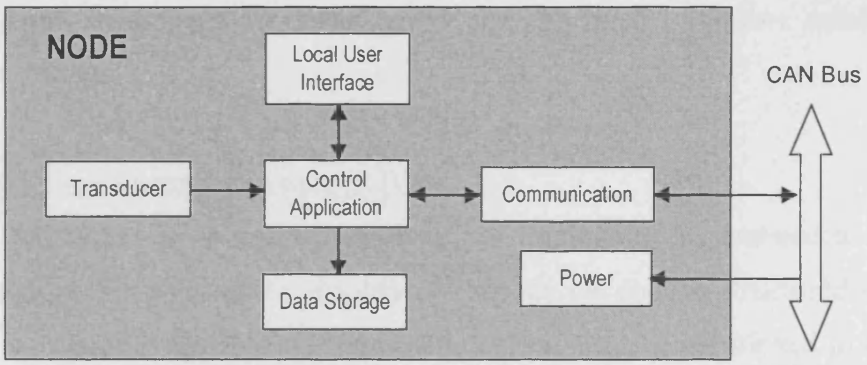


Figure 5.1 Generalised model of a node.

By definition, a smart transducer features integrated intelligence closer to the point of measurement and control, basic computation capability and the ability to communicate data and information in a standardised digital format, i.e. in this case, the CANopen specification. This approach is not only lends itself well to the development of a grid

architecture, but also has further benefits of faciltating more cost effective integration and maintenance of the completed machine system network during its operational life.

5.2.2 Node Systems

The internal architecture of a node can also be considered using a process of ordered hierarchical (top-level) decomposition, where the system is deconstructed into its component subsystems. A diagrammatic representation of the node subsystems and the relationship between firmware and hardware is shown in *figure 5.2*.

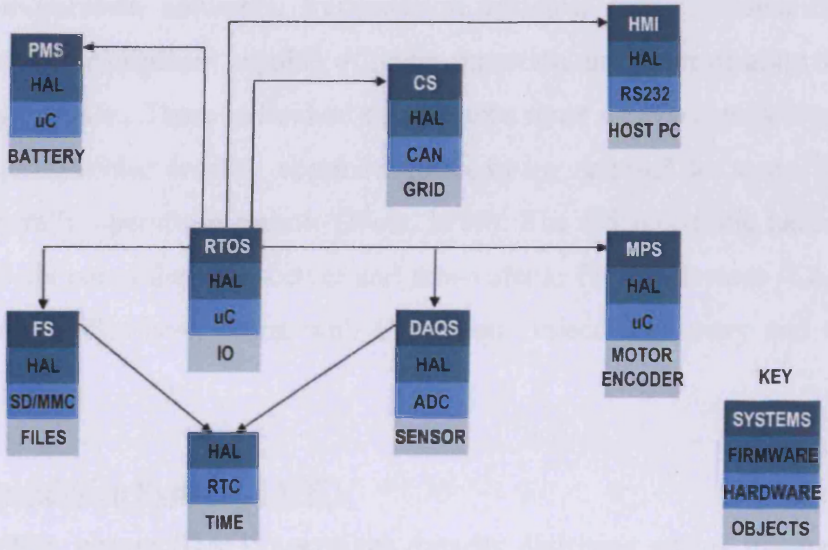


Figure5.2 Node sub-systems.

This overview of the system architecture is a guide for specifying cross-linkages between these subsystems and how they are expected to interact during the operational phase. The node subsystems are listed below and described in further detail in the sections that follow.

5.2.2.1 Real Time Operating System (RTOS)

The RTOS has emerged as one of the dominant paradigms for embedded systems programming and is a great aid to simplifying the way the code is structured when an application is managing multiple processes and devices. Code is partitioned in the time domain, with activities running concurrently, and procedurally where each task is dedicated to one specific function. Any mechanism that breaks the code into small independent blocks reduces algorithmic complexity (Lewis 2000) and yields better programs faster (Ganssle 2001). Application performance can be characterised in a relatively straightforward manner, regardless of size and complexity, making it easier to

add and test conceptual additions to the kernel as they are developed encouraging a system architecture that is scaleable and easier to maintain. Extensive use of this RTOS capability was made in this project, where firmware was incrementally integrated into the kernel framework to test the hardware and develop the sub-systems outlined below.

5.2.2.2 Communication System (CS)

A communications system can be described as a facility capable of providing data transfer between persons and equipment. The system usually consists of a collection of individual communication networks, transmission systems, relay stations, tributary stations, and terminal equipment capable of interconnection and interoperation so as to form an integrated whole. These individual components must serve a common purpose, be technically compatible, employ common procedures, respond to some form of control, and generally operate in unison (Weik 1989). The CS electronic hardware is composed of a CAN controller, transceiver and non-volatile FRAM devices *[**Chapter 6** Electronic Hardware Platform]* along with CANopen Object Dictionary and support firmware.

5.2.2.3 Data Acquisition System (DAQS)

The data acquisition system (DAQS) acquires data by digitising analog channels and storing the data in digital form. A DAQS can be a standalone system or interfaced to a host computer as is the case with a node. The node DAQS is composed of the temperature sensing transducer, ADC, microprocessor and control firmware.

5.2.2.4 Motor Positioning System (MPS)

The motor positioning system (MPS) is required for motor control in the gear pump, 3-way valve and isobaric pressure compensator nodes. The MPS is composed of the motor, gearbox, position encoders, intelligent H-bridge microprocessor and control firmware. This combination of hardware allows complete control of the motor speed and position.

5.2.2.5 File System (FS)

A file system (FS) can be described simply as a method for storing and organizing computer files to make them easily accessible, however it can also be considered as a set of abstract data types that are implemented for the storage, hierarchical organisation,

manipulation, navigation, access, and retrieval of data. This kind of abstraction is a powerful tool and implementing a FS frees the system designer to think of information in terms of files containing data without having to constantly be concerned with management the mechanisms responsible for the correct functioning of the storage media.

The decision was made to implement a file allocation table (FAT) FS because it is relatively uncomplicated and is supported by virtually all existing operating systems for personal computers. FAT is often used to share data between several operating systems booting on the same computer (a multi-boot environment). It is also used on solid-state memory cards and other similar devices. The most common implementations have a serious drawback in that when files are deleted and new files written to the media, their fragments tend to become scattered over the entire media making reading and writing a slow process. De-fragmentation is one solution to this, but is often a lengthy process in itself and has to be repeated regularly to keep the FAT FS clean.

The FAT16 FS on each node is utilised for storage of measurements acquired by the DAQS in hexadecimal or text format and is capable of addressing storage media cards from 32MB to 4GB capacity. Files stored in (8.3 name format). Also, the HLP will support direct transfer of sensor data from node to PC over the network as it may be impossible to gain physical access to the machine system in some environments.

5.2.2.6 Power Management System (PMS)

When connected to the network, nodes are capable of utilising an external power source (if it is available) to fast-charge their internal rechargeable batteries. The battery pack acts as power reservoir and is effectively a non-interruptible power supply, buffering nodes from external spurious power surges and longer term power interruptions. Once the batteries are fully charged the PMS switches to trickle-charge mode to prevent "gassing" and cell damage. The internal SPI bus gives the master CPU access to registers on the slave containing status information such as battery voltage and charge rate. Some of these registers are mapped into the object dictionary so they are accessible by other nodes on the network.

5.2.2.7 Human-Machine Interface (HMI)

To work with the machine system, a mechanism must be in place to facilitate operator control and access its status information. This mechanism is formally called the human-machine interface (HMI). A text based user interface (presentation layer) was developed to interpret a terse script language and decompose it to bytecode, which the machine system would then be capable of implementing.

This open design architecture does not preclude the possibility of adding a graphical user interface (GUI) on the host PC at a later date. The GUI or web browser based interface could be developed using Borland Builder or perhaps one of the virtual instrumentation or dataflow languages such as DASY*Lab*® or LabVIEW™. The GUI would use the same clearly defined syntax to communicate with the machine system, eliminating the potential requirement for huge changes to the firmware code residing on the nodes.

5.2.2.8 Hardware Abstraction Layer (HAL)

A hardware abstraction layer (HAL) is system of firmware device drivers that reside between the physical hardware of a computer and the RTOS that runs on the microprocessor. The drivers integrate into the RTOS framework and are called by tasks. Operating systems having a defined HAL are easily portable across different hardware. This is especially important for embedded systems that run on numerous types of microprocessors.

Device drivers encapsulate and bind data to a function so that it is hidden from higher firmware levels. Device drivers are written in a consistent programming style that follows a documented firmware standards manual (Ganssle 2000). Ultimately the intention is to build a library of device driver routines that can be integrated into other projects. A library is also desirable because the firmware development is now "frozen", that is, complete. Also, the build time from libraries is faster than from source files which must be compiled before they are linked together to produce the downloadable hex for the target.

5.3 Implementing MicroCANopen

MicroCANopen was introduced by Embedded Systems Academy as a minimal CANopen implementation targeted at deeply embedded applications with limited resources (requiring as little as 4 KBytes of program space and about 170 Bytes of memory). The designers built an API (application programmers interface) with two software communication interfaces. At the lowest level, there is an interface between the MicroCANopen protocol stack and the CAN controller hardware and at the highest level, an interface to the user application.

5.3.1 Hardware Driver Interface

The hardware driver interface (defined in `can.h`) provides basic functionality to transmit/receive CAN messages and manages all microprocessor specific issues that occur in the embedded application. This includes real-time performance issues, such as CAN interrupt service routines and supporting transmit and receive buffers for CAN messages. ***Appendix O.2*** contains a detailed source code listing and a brief overview of the functions that are implemented by the hardware driver is given below.

5.3.1.1 CanGetStatus Driver Function

This function is defined as follows:

```
UNSIGNED8 CanGetStatus(void);
```

This function returns the global status variable. The status variable can be changed anytime by this module, for example from within an interrupt service routine or by any of the other functions in this module.

5.3.1.2 CanInit Driver Function

This function is defined as follows:

```
UNSIGNED8 CanInit(UNSIGNED 16 baudrate);
```

This function implements the initialisation of the CAN interface. Allowed values for the baudrate are: 1000, 800, 500, 250, 125, 50, 25 or 10. The system baudrate is initialised to 125.

5.3.1.3 CanSetFilters Driver Function

This function is defined as follows:

```
UNSIGNED8 CanSetFilters(UNSIGNED16 CAN_ID);
```

This function implements the configuration of CAN controller hardware mask and filters for receive buffers.

5.3.1.4 CanPushMessage Driver Function

This function is defined as follows:

```
UNSIGNED8 CanPushMessage(CAN_MSG *pTransmitBuf);
```

"CanPushMessage" function implements a CAN transmit queue. With each function call a message is added to the queue, where the function returns '1' if message was added to the transmit queue and '0' if queue is full, i.e. the message was not added. The MicroCANopen stack will not try to add messages to the queue "back-to-back". With each call to `MCO_ProcessStack()`, a maximum of one message is added to the queue. The network load is low for this application, so the firmware queue is configured to hold just one message.

5.3.1.5 CanPullMessage Driver Function

This function is defined as follows:

```
UNSIGNED8 CanPullMessage(CAN_MSG *pTransmitBuf);
```

The "CanPullMessage" function implements a CAN receive queue, where each function call pulls a message from the queue. If a CAN message was pulled from receive queue the function returns '1', otherwise '0'. Only one receive buffer is utlised in the CAN controller hardware and a minmal software queue of length '1'.

5.3.1.6 CanGetTime Driver Function

This function is defined as follows:

```
UNSIGNED16 CanGetTime(void);
```

"CanGetTime" typically reads a 1 ms timer tick. To ensure data consistency, the timer interrupt incrementing the timer tick is disabled while executing this function. Systems that do not have capability to support a 1ms tick can increment the timer tick as a multiple of this to conserve processing resources. *Section 5.4.2* briefly outlines how the system interrupt service routine is configured to provide a timer with a minimum granularity of 10ms.

5.3.1.7 CanIsTimeExpired Driver Function

This function is defined as follows:

```
UNSIGNED8 CanGetTime(UNSIGNED16 timestamp);
```

This function compares a `UNSIGNED16` timestamp to the internal timer tick and returns '1' if the timestamp expired/passed. The maximum timer runtime measurable is 0x8000 (about 32 seconds).

5.3.2 Application Level Interface

Both shared data memory (process image) and function calls are used to implement an interface between MicroCANopen the application program. The application interface consists of two sets of functions. There is a set of API functions (defined in `mco.h`) that can be called from within the application to initialise the MicroCANopen and to pass process data to the communication stack. The second set of functions are call-back functions that MicroCANopen calls to inform the application of important events or the reception of process data.

5.3.2.1 MCO_Init API Function

This function is defined as follows:

```
void MCO_Init (
  UNSIGNED16 Baudrate,
  UNSIGNED8  Node_ID,
  UNSIGNED16 heartbeat
);
```

, and is used to initialise the MicroCANopen protocol stack. It is called from within the user call-back function MCO_ResetApplication which is called by MicroCANopen upon receiving a Reset command. The baud rate may be one of the values 1000, 800, 500, 250, 125, 50, 25 or 10. The node ID may be in the range of 1 to 127 and the

heartbeat time is specified in milliseconds. For example, to generate a heartbeat message once every 2 seconds, the value is set to 2000.

5.3.2.2 MCO_Init_RPDO API Function

This function is defined as follows:

```
void MCO_InitRPDO (
  UNSIGNED8  PDO_NR,
  UNSIGNED16 CAN_ID,
  UNSIGNED8  len,
  UNSIGNED8  offset
);
```

This function initialises a receive PDO. Once initialised, the MicroCANopen stack automatically writes received data to the destination pointer location. The parameter PDO_NR represents the number of receive RPDOs implemented by the node and is within the range of 1 to 4, the CAN_ID can be set to 0x0000 if the CANopen default ID should be used. The variable, `len` defines the number of data bytes in RPDO and `offset` to data location within the process image

5.3.2.3 MCO_Init_TPDO API Function

This function is defined as follows:

```
void MCO_InitTPDO (
  UNSIGNED8  PDO_NR,
  UNSIGNED16 CAN_ID,
  UNSIGNED16 event_tim,
  UNSIGNED16 inhibit_tim,
  UNSIGNED8  len,
  UNSIGNED8  offset
);
```

This function initialises a transmit PDO by placing one or more variables into one CAN message as described in ***Appendix H***. The application can directly change the data at any time, however to ensure data consistency, the it must not write to the data while function `MCO_ProcessStack()` executes. A change-of-state (COS) triggering mechanism is utilsed here to initiate PDO transmission. This is implemented by setting `event_tim` to zero and `inhibit_tim` to a no-zero value that represents a minimal timeout period between message transmissions in milliseconds. *Listing 5.1* shows the initialisation of RPDOS and TPDOs in "`initdos.h`" for Node #1 (Valve A).

```
/* To use pure change-of-state (COS) transmission, the parameter "event time" must be
zero. */
#define INITPDOS_CALLS \
/* Node #1 (Valve A) */
MCO_InitRPDO(1, 0x283, 2, P200400_HARDWARE_ERROR_); \
MCO_InitRPDO(2, 0x287, 7, P200001_RTC_SECONDS_); \
/* Note: TPDO 1 is reserved for future use */
MCO_InitTPDO(2, 0x281, 0, 0, 1, P200300_ACTUATOR_CONTROL_); \
```

Listing 5.1 initpdos.h contains the RPDO and TPDO initialisations

5.3.2.4 MCO_ProcessStack API Function

This function is defined as follows:

```
UNSIGNED8 MCO_ProcessStack(void);
```

, and implements the main CANopen protocol stack. It is called at frequent intervals from within the `for(;;)` in main to ensure that CANopen communication is not high priority, whilst, at the same time minimising CAN message latency. Once initialised, the CANopen stack automatically handles transmitting of PDOs.

The flowcharts in ***Appendix G*** fully describe the operation of the CANopen protocol stack. The main function is illustrated in ***[Appendix G.1]*** which checks whether this is the first time the function is called, it then polls the next receive message from the driver. If a message was received and it is a NMT master message or an RPDO message, the associated code section are executed as shown in ***Appendix G.2, G.3 and G.4.*** If no message is received or the message received is not for the local node to handle, `MCO_ProcessStack()` continues with potential transmissions that are eligible. First, the TPDO transmissions are checked and if no TPDO transmission is eligible, the heartbeat producer time is verified. If it is expired, a heartbeat message is generated.

Only one TPDO per call to `MCO_ProcessStack()` is checked for transmission. This done to avoid bursts of back-to-back TPDOs that may be ready for transmission. When using an Event Time (i.e. TPDO is transmitted every *'x'* milliseconds), the handling of the TPDO is simple. If the time is expired, re-set the timer and transmit the TPDO. However, if COS detection is used with an Inhibit Time, the first step is to check if the TPDO is already due for transmission and is just waiting for the Inhibit Time to expire. If that is not the case, the last transmitted data needs to be compared with the current data (trying to detect the COS). If data changed, it needs to be copied to the transmit

buffer. However, it can only be transmitted if the Inhibit Time has already expired – otherwise transmission needs to wait. The Inhibit Time is reset with every transmission of the TPDO.

When handling a receive PDO, the stack runs through a loop checking all configured RPDOs to see if the identifier of the message received matches any of the identifiers used for the RPDOs. If a match is found, the data received is copied to the appropriate process variable.

5.3.2.5 MCO_ResetApplication Call-Back Function

This function resets the application and is called from within the CANopen protocol stack, if a NMT master message was received that demanded "Reset Application". A `while(1)` infinite loop is used to implement a "watchdog trap" which guarantees a reliable hardware reset of the embedded microprocessor.

```
void MCO_ResetApplication(void)
{   /* Call-back function to reset application. */
    while (1)
    {  /* waits until watchdog hardware causes the processor to reset */
       /* Watchdog hardware will reset processor reset within 250 ms*/
    }
}
```

Listing 5.2 MCO_ResetApplication function.

5.3.2.6 MCO_ResetCommunication Call-Back Function

This function both resets and initialises both the CAN interface and the CANopen protocol stack. It is called from within the CANopen protocol stack, if a NMT master message was received that demanded "Reset Communication". This function calls `MCO_Init` and `MCO_InitTPDO/MCO_InitRPDO`.

```
void MCO_ResetCommunication(void)
{
  UNSIGNED16 can_bps = 125;    /* Note: This version always uses 125 Kbps */
  UNSIGNED8 node_id = NODE_ID;

  EA = 0;

  /* NOTE: Serial number must be present in process image at location */
  /* gProcImg[P101804_SERIAL_NUMBER] */

  MCO_Init(can_bps,node_id,1000);

  if (node_id != 0)
  {  /* Node #1 (Valve A) */
    MCO_InitRPDO(1, 0x283, 2, P200400_HARDWARE_ERROR_);
    MCO_InitRPDO(2, 0x287, 7, P200001_RTC_SECONDS_);
    /* Note: TPDO 1 is reserved for future use */
    MCO_InitTPDO(2, 0x281, 0, 0, 1, P200300_ACTUATOR_CONTROL_);
  }
}
```

Listing 5.3 MCO_ResetCommunication function.

5.3.2.7 MCO_FatalError Call-Back Function

This function is called if a fatal error occurred. Error codes of `mcohwxxx.c` are in the range of 0x8000 to 0x87FF. Error codes of `mco.c` are in the range of 0x8800 to 0x8FFF. All other error codes may be used by the application.

```
void MCO_FatalError(UNSIGNED16 ErrCode)  /* the error code */
{   /* Turn on red LED to indicate an error has occured */
 /*  error_state(ErrCode & 0xFF); */
 PIN_RED = ON;
}
```

Listing 5.4 MCO_FatalError function.

5.3.2.8 Configuration of the Process Image

In order to offer a generic method for addressing and exchanging the data communicated via CANopen, process data is organised into a process image. The process image is implemented as an array of bytes and is used as shared memory to facilitate access for MicroCANopen as well as from the application program. The process image contains all process data variables that are communicated via CANopen. Access functions are provided to allow the application program to read or write data from or to the process image. The length of the process image in bytes is defined by a constant, `PROCIMG_SIZE` in file "`procimg.h`" and must be in the range of 0x01 to 0xFE (values 0x00 and 0xFF are reserved).

A single variable of the process image can be addressed by specifying an offset and a length. The offset specifies where in the process image the first byte of a variable is stored and the length specifies how many bytes are used to store the variable. The offset

may have a value from 0 to 0xFE. Using an offset of 0x0FF indicates that the offset is invalid or unused. If numeric values are stored in multiple byte variables, then the default byte order is CANopen compatible: Little Endian – the lower bytes are stored at the lower offset.

The process image variable definitions are contained in the header file "`pimg.h`", where each Object Dictionary entry Index and Sub-index is assigned to a variable and the PDOs are configured to contain one or multiple process data variables. *Figure 5.5* below shows mapping entry definitions for RPDOs and TPDOs by node #1 (valve A).

```
#ifndef _PIMGH_
#define _PIMGH_

/* Maximum number of receive PDOs */
#define NR_OF_RPDOS       2

/* Maximum number of transmit PDOs */
#define NR_OF_TPDOS       2

/* Mapping entries for RPDO1 [1400] */
/* [2004,00] UNSIGNED8 */
#define P200400_HARDWARE_ERROR_ 0x00000000
/* [2008,00] UNSIGNED8 */
#define P200800_VALVE_POSITION_ 0x00000001

/* Mapping entries for RPDO2 [1401] */
/* [2000,01] UNSIGNED8 */
#define P200001_RTC_SECONDS_ 0x00000002
/* [2000,02] UNSIGNED8 */
#define P200002_RTC_MINUTES_ 0x00000003
/* [2000,03] UNSIGNED8 */
#define P200003_RTC_HOURS_ 0x00000004
/* [2000,04] UNSIGNED8 */
#define P200004_RTC_DATE_ 0x00000005
/* [2000,05] UNSIGNED8 */
#define P200005_RTC_MONTH_ 0x00000006
/* [2000,06] UNSIGNED8 */
#define P200006_RTC_DAY_ 0x00000007
/* [2000,07] UNSIGNED8 */
#define P200007_RTC_YEAR_ 0x00000008

/* Mapping entries for TPDO1 [1800] */
/* Reserved - not in use */

/* Mapping entries for TPDO2 [1801] */
/* [2003,00] UNSIGNED8 */
#define P200300_ACTUATOR_CONTROL_ 0x00000009

#endif /* #ifndef __PIMGH */
```

Listing 5.5 Node #1 (Valve A) Mapping Entries for RPDOs and TPDOs

To simplify accessing the process image and to allow for easy reconfiguration of process images, `#define` statements are used to define the offsets to the individual variables in the process image. These are defined in the file "`procimg.h`" that is included to all code modules requiring access to the process image.

```
#ifndef _PROCIMG_H
#define _PROCIMG_H

/* DEFINES: Definition of the process image
Modify these for your application. */

/* Define the size of the process image */
#define PROCIMG_SIZE 10

/* Define process variables: offsets into the process image  */

#define HARDWARE_ERROR 0x00
#define VALVE_POSITION 0x01

#define RTC_SECONDS 0x02
#define RTC_MINUTES 0x03
#define RTC_HOURS 0x04
#define RTC_DATE 0x05
#define RTC_MONTH 0x06
#define RTC_DAY 0x07
#define RTC_YEAR 0x08

#define ACTUATOR_CONTROL 0x09

#endif /* #ifndef __PROC_IMG_H */
```

Listing 5.6 Process image offset definitions.

All variables mapped into a CAN message (PDO) must be located consecutively in the process image so that the entire contents of a PDO can copied byte-by-byte from/to the process image. The application program directly accesses the data in the process image. For example, *gProcImg[offset] = x*

5.4 Implementing Salvo RTOS

The firmware architecture for each node is based on a commercially available RTOS developed by Pumpkin Inc. (Kalman 2003) which is expressly designed for low-cost embedded systems with severely limited ROM (Read Only Memory) and RAM (Random Access Memory) resources. Typical applications use between 1 and 2 KBytes ROM and 50 to 100 Bytes RAM. The RTOS is highly configurable (Valenti and Kalman 2001) and scalable, with a full set of run-time features including priority based, cooperative multitasking, event services (semaphores, messages, message queues and event flags), real-time delays and elapsed-time services. Multitasking is priority based, with support for fifteen different priority levels where tasks with the same priority execute in a round-robin fashion. Salvo is written in ANSI C, with only a small number of processor-specific extensions, facilitating portability of any programs.

5.4.1 The Multitasking RTOS Approach

In this paradigm, tasks operate concurrently rather than in the conventional main super-loop and interrupt service routine running in the background with undefined or unpredictable latency (where latency is defined as the time delay between the moment a task is initiated, and the moment a task actually begins). The RTOS maintains each task in one of a number of states as shown in *figure 5.3* below.

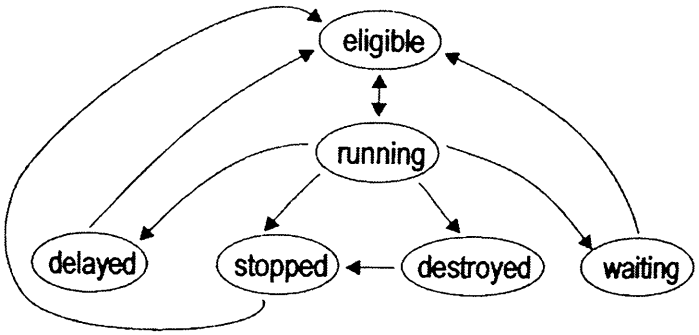


Figure 5.3 Task states

Tasks can exist in many different states; periodic tasks are likely to be delayed at any particular instant, low-priority tasks may be eligible but unable to run because a higher priority task is already running and some tasks are likely to be waiting for an event. It is the responsibility of the scheduler to manage all these tasks and ensure that each task runs when it should. A more detailed explanation of Salvo RTOS can be found in Salvo User Manual (Kalman 2003). This high level of abstraction lends it self well to the challenge of developing smart or complex machine behavior encourages efficient use of limited microprocessor resources and clock cycles. Because the RTOS is event driven, it is possible to minimise power consumption while maximising use of the microprocessor. This solution is also advantageous because it embraces the important principle of code reusability and is supplied with comprehensive documentation.

These sophisticated messaging mechanisms facilitate reliable and safe communication between tasks without having to resort to using global variables. Using global variables is especially bad practice in a mission critical application like this because they are available to all functions. This practice means firmware is vulnerable to potential conflicts if tasks attempt to modify global variables simultaneously.

5.4.2 OSTimer()

Node firmware makes extensive use of Salvos delay services, to delay a tasks or sequences of functions within a given task. The listing below shows part of the interrupt service routine that implements a 10ms timer by calling `OSTimer()` at a periodic rate of approximately 100Hz. The periodic rate is derived from the processor hardware timer overflow, which generates an interrupt.

```
#define TMR0_RELOAD    157     /* for 100Hz (10ms) ints @ 20MHz TM0 prescaler 1:4 */

void interrupt IntVector(void)
{
    if (T0IE && T0IF)
    {   /* System timer */
        T0IF = 0;
        TMR0 -= TMR0_RELOAD;
        OSTimer();
    }
}
```

Listing 5.7 Calling OSTimer() at the System Tick Rate.

5.4.3 Main()

The firmware code within all nodes contains the `main()` function essential part to any 'C' programs. The first few lines of code in `main()` make calls to several functions responsible for initialising and configuring the PIC processor hardware, RTOS and MicroCANopen protocol stack. `PICInit()` configures the processor digital I/O pins as inputs or outputs and assigns names to them *[**Appendix O.11**]*. `OSInit()` initialises all the RTOS data structures, pointers and counters (this function must be called before any other calls to RTOS functions are made or unpredictable behaviour can occur). The then code drops into an infinite `for()` loop where the scheduler (`OSSched()`) operates in a tight loop checking for eligible tasks. Only tasks which are in the eligible state can run, and each call to `OSSched()` results in the most eligible task running until the next context switch within that task. In order for multitasking to continue, `OSSched()` must be called repeatedly (Kalman 2003). Finally, the MicroCANopen operates as a background task by calling the protocol stack (`MCO_ProcessStack()`). This is done in order to avoid CANopen communication being a high priority within the system.

```
Void main(void)
{
  PICInit();                              /* Initialise PIC ports, serial, SPI          */
  puts(NodeInit(buf, &systemRegister));   /* Initialise system to a known state         */
  newLine();
  MCO_ResetCommunication();               /* Initialise CANopen protocol stack          */
  OSInit();                               /* Initialise RTOS                            */

  OSCreateTask((OStypeTFP)TaskStrobe, (OstypeTcbP)TASK_STROBE_P,
(OstypePrio)PRIO_TASK_STROBE);
  OSCreateTask(TaskRs232In, TASK_GET_KEY_P, PRIO_GET_KEY);
  OSCreateTask(TaskBist, TASK_BIST_P, PRIO_BIST);
  OSCreateTask(TaskDump, TASK_DUMP_P, PRIO_DUMP);
  OSCreateTask(TaskAdcSample, TASK_ADC_SAMPLE_P, PRIO_ADC_SAMPLE);
  OSCreateTask(TaskFlashInit, TASK_FLASH_INIT_P, PRIO_FLASH_INIT);
  OSCreateEFlag(EFLAG_SYSTEM_P, EFLAG_SYSTEM_CB_P, (BIT_ADC_BUSY | BIT_BIST));

  OSEi(); /* Enable interrupts */
  for (;;)
  {
    if (gProcImg[P200300_ACTUATOR_CONTROL_])
    {   /* Trigger DAQS to make a measurement */
      OSSetEFlag(EFLAG_SYSTEM_P, BIT_MEASURE);
    }
    if (gProcImg[P200300_ACTUATOR_CONTROL_])
    {   /* Trigger MPS to acquire fluid sample */
      OSSetEFlag(EFLAG_SYSTEM_P, BIT_FLUID_SAMPLE);
    }

    if (BIT_FLASH_DET & systemRegister.porState)
    { /* Simple user semaphore to detect insertion of flash card */
      GIE = 0;
      SystemRegister.porState &= (UNSIGNED8)~BIT_FLASH_DET;     /* Clear the flag */
      GIE = 1;
      OSSetEFlag(EFLAG_SYSTEM_P, BIT_FLASH_INIT);   /* Wake task */
    }
    MCO_ProcessStack();     /* Operate on CANopen protocol stack */
    OSSched();              /* Run highest-priority eligible task */
  }
}
```

Listing 5.8 Main()

5.4.4 Tasks

5.4.4.1 TaskStrobe()

The primary purpose of this task is to periodically strobe or "kick" the watchdog timer hardware. It has the highest priority of all tasks, as it is essential that the delay time between strobes is no greater than 250 ms or the watchdog will reset node hardware.

It is also responsible for timekeeping duties that trigger events in node sub-systems. This part of the code is implementation dependent as there are some functional differences between nodes. For example, some nodes utilise the RTC to trigger temperature measurements or to control the open of a fluid sampling bottle, whereas other nodes rely on incoming CAN messages to trigger them and therefore have no requirement for time keeping. The listing below is a generic sample of code found in the node firmware.

```
Void TaskStrobe(void)
{ /* Kick watchdog every 250ms and trigger any eligible tasks. Priority 2 */
  static char tock, tick;
  static unsigned int secs;
  for (;;)
  {
    PIN_GREEN = !PIN_GREEN;  /* Toggle LED on/off */
    Strobe();
    RtcGets(&rtcTime);
    Tick = (BcdToBin(rtcTime.seconds) & 0x01); /* Use mask to see if LSb in RTC has
changed */
    if ((tick ^ tock))
    {   /* Derive system ticks from change of state of LSb */
      secs++;   /* Increment seconds counter */
      tock = tick;
    }
    if (secs >= systemRegister.deltaT)
    {   /* Trigger DAQS to make a measurement */
      OSSetEFlag(EFLAG_SYSTEM_P, BIT_MEASURE);
      Secs = 0x0000;   /* Reset seconds counter */
    }
    if (RtcGetBitAlarmStatus())
    {   /* Trigger MPS to acquire fluid sample */
      OSSetEFlag(EFLAG_SYSTEM_P, BIT_FLUID_SAMPLE);
    }
    OS_Delay(T_STROBE, TaskStrobe1); /* 250ms delay */
  }
}
```

Listing 5.9 TaskStrobe()

5.4.4.2 TaskFluidSample()

This task is responsible for acquiring a fluid sample. Referring back to ***Chapter 4***, it can be seen that "Valve Node A", "Valve Node B" and the "Pump Node" operate as a subgroup of communication partners with "Valve Node A" acting as the timekeeper and having the responsibility of initiating the activity to acquire the fluid sample *[see section 4.2.5.1 PDO Linking]*. Each node in the subgroup implements this task, however there are some essential differences in the implementation. The code for "Valve Node A" is shown below *[listing 5.10]*. In this case the task is triggered by the event flag `(EFLAG_SYSTEM_P, BIT_FLUID_SAMPLE)`, where `TaskStrobe()` polls the RTC alarm every 250 ms to check if it has been set and communicates with `TaskFluidSample()` via the event flag to signal when the alarm is set.

```
Void TaskFluidSample(void)
{ /* This task uses MPS to acquire a fluid sample. Priority 5 */
  for (;;)
  { /* Trigger task when BIT_FLUID_SAMPLE is set */
    OS_WaitEFlag(EFLAG_SYSTEM_P, BIT_FLUID_SAMPLE, OSANY_BITS, OSNO_TIMEOUT,
TaskFluidSample1);
    OSClrEFlag(EFLAG_SYSTEM_P, BIT_FLUID_SAMPLE);
    MotorPutc((UNSIGNED8)(VALVE | (UNSIGNED8)VALVE_OPEN_INLET)); /* Open master
valve*/
    /* Open slave valve - CAN message transmit is triggered on COS */
    gProcImg[P200300_ACTUATOR_TRIGGER_] |= 0x80; /* Set V_TRIG bit in actuator control
reg. */
    OS_Delay(T_VALVE, TaskFluidSample2);      /* Wait for 6 minutes */
    MotorPutc((UNSIGNED8)(VALVE | (UNSIGNED8)VALVE_CLOSE)); /* Close master valve */
    GProcImg[P200300_ACTUATOR_TRIGGER_] &= 0x7F; /* Clear V_TRIG bit in actuator
control reg. */
  }
}
```

Listing 5.10 TaskFluidSample() implementation for "Valve Node A"

In "Valve Node B" the situation is a little different *[listing 5.11]*. Here the task is still triggered by the event flag `(EFLAG_SYSTEM_P, BIT_FLUID_SAMPLE)`, however an incoming CAN message (from "Valve Node A") sets `BIT_FLUID_SAMPLE` within the event flag.

```
Void TaskFluidSample(void)
{ /* This task uses MPS to acquire a fluid sample. Priority 5 */
  for (;;)
  { /* Trigger task when BIT_FLUID_SAMPLE is set */
    OS_WaitEFlag(EFLAG_SYSTEM_P, BIT_FLUID_SAMPLE, OSANY_BITS, OSNO_TIMEOUT,
TaskFluidSample1);
    OSClrEFlag(EFLAG_SYSTEM_P, BIT_FLUID_SAMPLE);
    MotorPutc((UNSIGNED8)(VALVE | (UNSIGNED8)VALVE_OPEN)); /* Open valve*/
    OS_Delay(T_VALVE, TaskFluidSample2);      /* Wait 30 seconds for valves to open */
    /* Switch pump on - CAN message transmit is triggered on COS */
    gProcImg[P200300_ACTUATOR_TRIGGER_] |= 0x40; /* Set P_TRIG bit in actuator control
reg. */
    OS_Delay(T_VALVE, TaskFluidSample3);      /* Wait for 5 minutes and 30 seconds */
    MotorPutc((UNSIGNED8)(VALVE | (UNSIGNED8)VALVE_CLOSE)); /* Close valve */
    GProcImg[P200300_ACTUATOR_TRIGGER_] &= 0xBF; /* Clear P_TRIG bit in actuator
control reg. */
  }
}
```

Listing 5.11 TaskFluidSample() implementation for "Valve Node B"

The implementation of TaskFluidSample() within the pump node is shown below.

```
void TaskFluidSample(void)
{ /* This task uses MPS to acquire a fluid sample. Priority 5 */
  for (;;)
  { /* Trigger task when BIT_FLUID_SAMPLE is set */
    OS_WaitEFlag(EFLAG_SYSTEM_P, BIT_FLUID_SAMPLE, OSANY_BITS, OSNO_TIMEOUT,
TaskFluidSample1);
    OSClrEFlag(EFLAG_SYSTEM_P, BIT_FLUID_SAMPLE);
    MotorPutc((UNSIGNED8)(PUMP_ON)); /* Switch pump on */
    OS_Delay(T_PUMP, TaskFluidSample2);      /* Pump fluid for 5 minutes */
    MotorPutc((UNSIGNED8)(PUMP_OFF);  /* Switch pump off */
  }
}
```

Listing 5.12 TaskFluidSample() implementation for "Pump Node"

From the above listings, it is apparent that these nodes all rely on some type of event to trigger various actions which are localised to themselves or to one other communication partner, effectively sharing the computational load evenly amongst the subgroup, i.e. the control is decentralised. This modular approach is highly scaleable and maintainable because the number of lines of code within any one node is limited to a few thousand lines and it does not increase excessively as new nodes are added to the network *[see **Chapter 4**, CANopen Implementation].*

5.4.4.3 TaskMeasure()

This task is implemented in the temperature sensor node only, where its purpose is to perform accurate, high resolution temperature measurements. This activity involves coordination the data acquisition system ADC and RTC hardware, where an ADC measurement is triggered on an event flag (EFLAG_SYSTEM_P, BIT_MEASURE) set by the RTC.

```
void TaskMeasure(void)
{ /* This task is part of the DAQS and controls the rate of measurement. Priority 5 */
  for (;;)
  {
    OS_WaitEFlag(EFLAG_SYSTEM_P, BIT_MEASURE, OSANY_BITS, OSNO_TIMEOUT, TaskAdcSample1);
    OSClrEFlag(EFLAG_SYSTEM_P, BIT_MESASURE); /* Trigger task when BIT_MEASURE is set */

    OS_WaitEFlag(EFLAG_SYSTEM_P, BIT_ADC_BUSY, OSANY_BITS, OSNO_TIMEOUT,
TaskAdcSample2);
    OSClrEFlag(EFLAG_SYSTEM_P, BIT_ADC_BUSY);  /* Mutual exclusion locking of ADC
resource */
    PIN_RTD_ON = LOW;          /* Enable the REF200 200uA current excitation & Vref */
    PIN_ADC_ON = HIGH;                    /* Power-up ADC */

    OS_Delay(4, TaskAdcSample3);          /* Time for C21 to charge up and stabilise */

    AdcConvert(EFFLUENT_FLOW);
    OS_Delay(T_ADC, TaskAdcSample4);      /* Time for conversion to complete (minimum
34ms) */
    AdcGetData((UNSIGNED8 *)&(adcChannel.one));

    AdcConvert(EFFLUENT_TEMP);
    OS_Delay(T_ADC, TaskAdcSample5);      /* Time for conversion to complete (minimum
34ms) */
    AdcGetData((UNSIGNED8 *)&(adcChannel.two));

    AdcConvert(PCB_TEMP);
    OS_Delay(TADC, TaskAdcSample6);       /* Time for conversion to complete (minimum
34ms) */
    AdcGetData((UNSIGNED8 *)&(adcChannel.three));

    PIN_RTD_ON = HIGH;          /* Disable the REF200 200uA current excitation & Vref */
    PIN_ADC_ON = LOW;                     /* Shutdown ADC */
    OSSetEFlag(EFLAG_SYSTEM_P, BIT_ADC_BUSY);   /* Finished using ADC */
    OSSetEFlag(EFLAG_SYSTEM_P, BIT_DUMP);  /* Trigger task to store/display data */
  }
}
```

Listing 5.13 TaskMeasure()

The ADC hardware requires a minimum duration of 40ms to perform a complete measurement conversion on one channel before it can store the acquired sample as an array of three bytes. Rather than simply waiting in a delay loop and doing nothing, TaskMeasure() is put into a delayed state for 40ms where context switching causes the code to jump back into the OS scheduler so the processor is free to undertake other duties. For perspective, a delay of 40ms wastes a considerable quantity of clock cycles when using a 20MHz resonator with this processor architecture, as calculated below:

$$f_{osc} = Resonator\ Frequency\ /\ 4 = 20MHz/4 = 5MHz \tag{5.1}$$

$$t_{osc} = 1/f_{osc} = 1/5MHz = 200ns \tag{5.2}$$

The computational core is based RISC architecture where one instruction is processed every clock cycle, therefore,

$$\begin{aligned} Number\ of\ instructions\ processed &= Delay\ time/instruction\ cycle\ time \\ &= 40\ ms / 200ns \end{aligned} \tag{5.3}$$

= 200,000 instructions

This task also deals with controlled power-up and shutdown of ADC excitation current source and the ADC to keep power consumption of these devices to a minimum.

The services offered by the RTOS support management of resources by use of binary semaphores or event flags. In this case, event flags are utilised to implement mutual exclusion or locking of the on board ADC to prevent tasks from simultaneously attempting to access it, which would result in unpredictable behaviour.

Before exiting from this task an event flag (EFLAG_SYSTEM_P, BIT_DUMP) triggers TaskStream().

5.4.4.4 TaskStream()

This task is implemented by the temperature sensor node, where its function is to stream measurement data to a memory storage device (Secure Digital or Multimedia Card), to the RS-232 comms port or onto the CAN bus. There is also capability to configure the temperature sensor node to output the stream in different formats, including calibrated measurement data as text, raw hexadecimal and in DASY*Lab*® compatible format. In this case, the task performs a mass data storage function, however it could also be utlised for logging status of a node or to keep a track of activities and events to maintain a history or operational profile.

```
void TaskStream(void)
{   /* Priority 7 */
    //static UNSIGNED8 fileCount = 0x00; /* Option to write data to a series of several
separate files. All these files share the same root name (between 4 and 7 characters)
followed by consecutive numbers automatically assigned by the program. */
    //static WORD blockCount = 0xFFFF; /* Count of blocks written to file. When
blockCount == blockSize close file and open a new file.  */
    //static UNSIGNED8 fileHeader[1];         /* TODO: structure containing information
about the file */
    DATA text;

  for (;;)
  {
    OS_WaitEFlag(EFLAG_SYSTEM_P, BIT_DUMP, OSANY_BITS, OSNO_TIMEOUT, TaskDump1);
    OSClrEFlag(EFLAG_SYSTEM_P, BIT_DUMP);

//        if (BIT_HEX & systemRegister.porState)
//        {   /* Hex format  */
//            if (BIT_DASYLAB & systemRegister.porState)
//            {   /* Discard timestamp in DASYLab compatible mode */
//                text.time[0] = rtcTime.seconds;
//                text.time[1] = rtcTime.minutes;
//                text.time[2] = rtcTime.hours;
//                text.time[3] = rtcTime.day;
```

```
//                text.time[4] = rtcTime.month;
//                text.time[5] = rtcTime.year;
//                text.time[6] = '\0';  /* Append terminating null */
//            }
            /*
            To maintain compatibility with DASYLab:
            1. Add channel number to allow DASYLab to identify the channels
            2. Pad with an extra padding byte so sample is of type DWORD (32bits)
            3. Store 'little endian' format
            */
//            text.temperature.one[0] = 0x01;              /* Channel number */
//            text.temperature.one[1] = adcChannel.one[2]; /* LSB */
//            text.temperature.one[2] = adcChannel.one[1]; /*     */
//            text.temperature.one[3] = adcChannel.one[0]; /* MSB */
//            text.temperature.one[4] = 0x00;              /* Padding byte */
//            if (BIT_DASYLAB & systemRegister.porState) text.temperature.one[5] = '\0';
/* Don't need terminating null on string in DASYLab mode */

//            text.temperature.two[0] = 0x02;
//            text.temperature.two[1] = adcChannel.two[2];
//            text.temperature.two[2] = adcChannel.two[1];
//            text.temperature.two[3] = adcChannel.two[0];
//            text.temperature.two[4] = 0x00;
//            if (BIT_DASYLAB & systemRegister.porState) text.temperature.two[5] = '\0';
/* Don't need terminating null on string in DASYLab mode */

//            text.temperature.three[0] = 0x03;
//            text.temperature.three[1] = adcChannel.three[2];
//            text.temperature.three[2] = adcChannel.three[1];
//            text.temperature.three[3] = adcChannel.three[0];
//            text.temperature.three[4] = 0x00;
//            if (BIT_DASYLAB & systemRegister.porState) text.temperature.three[5] =
'\0';   /* Don't need terminating null on string in DASYLab mode */
//        }
//        else
        {  /* Text format */
            //ClearBuf((UNSIGNED8 *)&(text.temperature.three));
            TimeToString(&rtcTime, (UNSIGNED8 *)&(text.time));
            TemperatureToString(adcChannel.one, (UNSIGNED8 *)&(text.temperature.one),
(systemRegister.porState & BIT_UNITS));
            TemperatureToString(adcChannel.two, (UNSIGNED8 *)&(text.temperature.two),
(systemRegister.porState & BIT_UNITS));
            TemperatureToString(adcChannel.three, (UNSIGNED8
*)&(text.temperature.three), (systemRegister.porState & BIT_UNITS));
        }

        if (BIT_RS232_STREAM & systemRegister.porState)
        {   /* Stream data to RS-232 to host PC */
            puts((UNSIGNED8 *)&(text.time));
            puts((UNSIGNED8 *)&(text.temperature.one));
            puts((UNSIGNED8 *)&(text.temperature.two));
            puts((UNSIGNED8 *)&(text.temperature.three));
            putch(';'); newLine();
        }

        if (BIT_FLASH_STREAM & systemRegister.porState)
        {   /* Stream data to SD/MMC storage media */
//            if (blockCount++ >= (systemRegister.blockSize - 1))
//            {   /* Chain file series */
//                char index[3];
//                blockCount = 0x0000;
//                CharToHex(fileCount, index);  /* Convert file size to text format */
//                systemRegister.filename[6] = *(index + 0);
//                systemRegister.filename[7] = *(index + 1);
//                systemRegister.filename[9] = 'a';
//               systemRegister.filename[10] = 's';
//                systemRegister.filename[11] = 'c';
//                if (BIT_HEX & systemRegister.porState)
//                {
//                    systemRegister.filename[9] = 'h';
//                    systemRegister.filename[10] = 'e';
//                    systemRegister.filename[11] = 'x';
//                }
//                systemRegister.filename[12] = '\0';
//               fclose(out);      /* Close */
//                fopen(systemRegister.filename, out);
 //                fputs(fileHeader, out);
```

```
//                puts(systemRegister.filename); newLine(); /* Display filename on
filename increment */
//                fileCount++;    /* */
//            }
      putch('.'); newLine();
      fputs((UNSIGNED8 *)&(text.time), out); /* Timestamp */
      fputs((UNSIGNED8 *)&(text.temperature.one), out); /* Channel one */
      fputs((UNSIGNED8 *)&(text.temperature.two), out); /* Channel two*/
      fputs((UNSIGNED8 *)&(text.temperature.three), out); /* Channel three */
      fputc(';', out); /* Delimiter */
      fputc((char)CR, out); fputc((char)LF, out); /* newLine(); */
    }

    if (BIT_CAN_STREAM & systemRegister.porState)
    {   /* Stream data to CAN bus */
      CanPushMessage(pTransmitBuf) /* Send CAN message */
    }
        //OSClrEFlag(EFLAG_SAMPLE_P, RECORD);
  }
}
```

Listing 5.14 TaskStream()

5.4.4.5 TaskRS232()

`TaskRs232()` supports terse a command line interface for communication via legacy RS-232 interface for test and diagnostic purposes.

```
void TaskRs232(void)  /* TaskRs232 Priority 4 */
{
  static char key;
  for (;;)
  {
    if (kbhit())
    {   /* If there are characters in the FIFO */
      key = toupper(getch());  /* Some basic input validation. Make uppercase */
      if (CR == key)
      {    /* Parse the command-line and process */
        newLine();
        CliParse(buf, &command, argument);/* Parse the command-line */
        ClearBuf(buf);    /* Clear all characters in the buffer */
        CliProcess(command, argument); /* Process the command-line */
        ClearBuf(argument);    /* Clear all characters in the buffer */
        command = 0x00;
        //newLine(); //putch(command); putch(' '); puts(argument); newLine();   /*
Display command & argument */
      }
      else
      {   /* Build the command-line */
        if (isprint(key))
        {
          putch(key);  /* Display the key pressed */
          CliBuild(key, buf);  /* Build the command-line */
        }
      }
    }
    OS_Delay(T_KEYSCAN, TaskGetKey1);
  }
}
```

Listing 5.15 TaskRS232()

This task makes use of another useful RTOS service is the queue, which is configured as a first in first out (FIFO) buffer for short term storage of characters received on the UART. When a character is received, a short interrupt service routine (ISR) quickly

inserts it into the buffer. Every 250ms a task within the RTOS checks the FIFO and removes any characters within the buffer.

5.4.5 OSIdleHook()

Salvo's scheduler normally runs in a tight loop when no tasks are eligible to run, i.e. when it is idling. Use is made of the "Idle" function hook service to switch on an amber LED to indicate that the operating system is busy. The ultimate aim is to utilise this service to decrease clock speed or put the processor into sleep mode while the processor is idle to reduce power consumption to conserve battery life and extend deployment duration capability.

5.5 Putting it all together

MAKE files are utilised to describe how source code modules are compiled, assembled and linked to build the final ROM image that resides on the target processor, effectively acting as a blueprint for building the 'C' source code and translating it into machine code that will run on the target microprocessor. They also describe the tools used to build the final executable. These MAKE files allow repeatable, documented and portable firmware builds and the source code can rapidly be recompiled for other processor targets. For example, this powerful technique is used for test and development purposes, where hardware-specific header files are replaced so that test code can be compiled to run as an executable on the PC rather than downloading it to the target microprocessor. This speeds up the development cycle and promotes more thorough testing of the code. Project specific information (i.e. source code files, target, etc) is located in a project file (`isosamp.pjt`). This allows code to be built for different target processors by altering the "`PROCESSOR`" and "`COMPILER`" options. A full description of the tool chain is given in the project MAKE files header section with command-line syntax for driving the compiler, linker, boot-loader, programmer and terminal emulator. ***Appendix K*** shows the MAKE files used to build the firmware that is downloaded into each node.

The test compilation listed below shows resource usage for a full system build that includes the Salvo RTOS kernel, RS-232 command line interface, all drivers in the HAL and the MicroCANopen protocol. Despite the fact that this build incorporates more

functionality than a typical node will require, there is ample program and memory space available for further development work with this particular processor core

```
Program ROM    $000000 - $000003  $000004 (       4) bytes
Program ROM    $000008 - $000057  $000050 (      80) bytes
Program ROM    $000096 - $0005FF  $00056A (    1386) bytes
Program ROM    $000844 - $003FFD  $0037BA (   14266) bytes
Program ROM    $004000 - $0070FB  $0030FC (   12540) bytes
                                  $006E74 (   28276) bytes total Program ROM

RAM data       $000012 - $0001F6  $0001E5 (     485) bytes
RAM data       $000396 - $0005FF  $00026A (     618) bytes
                                  $00044F (    1103) bytes total RAM data

Near RAM       $000000 - $00000F  $000010 (      16) bytes total Near RAM
Near bits      $000080 - $00008B  $00000C (      12) bits  total Near bits
ROM data       $000058 - $000095  $00003E (      62) bytes
ROM data       $000600 - $000843  $000244 (     580) bytes
                                  $000282 (     642) bytes total ROM data

Config Data    $300000 - $30000D  $00000E (      14) bytes total Config Data

Program statistics:

Total ROM used    28918 bytes (88.3%)
Total RAM used     1121 bytes (73.0%)   Near RAM used       18 bytes (14.1%)
```

Listing 5.16 Memory usage map

Chapter 6 Electronic Hardware Platform

Reliability and micro-power operation specification requirements played a large part in shaping the final electronic hardware solution for the nodes in the machine system network. To this end, a custom single board computer (SBC) was developed at Cardiff University, UK *[figure 6.1]* to be embedded within each node. This chapter gives an overview of the features of the SBC and a detailed description of the design and function of the on-board low-level hardware infrastructure.

6.1 Single Board Computer (SBC)

A SBC is a complete computer built on a single printed circuit board (PCB), including microprocessor, random access memory (RAM), input/output (IO), real-time clock (RTC) and networking capability via Bosch controller area network (CAN) bus.

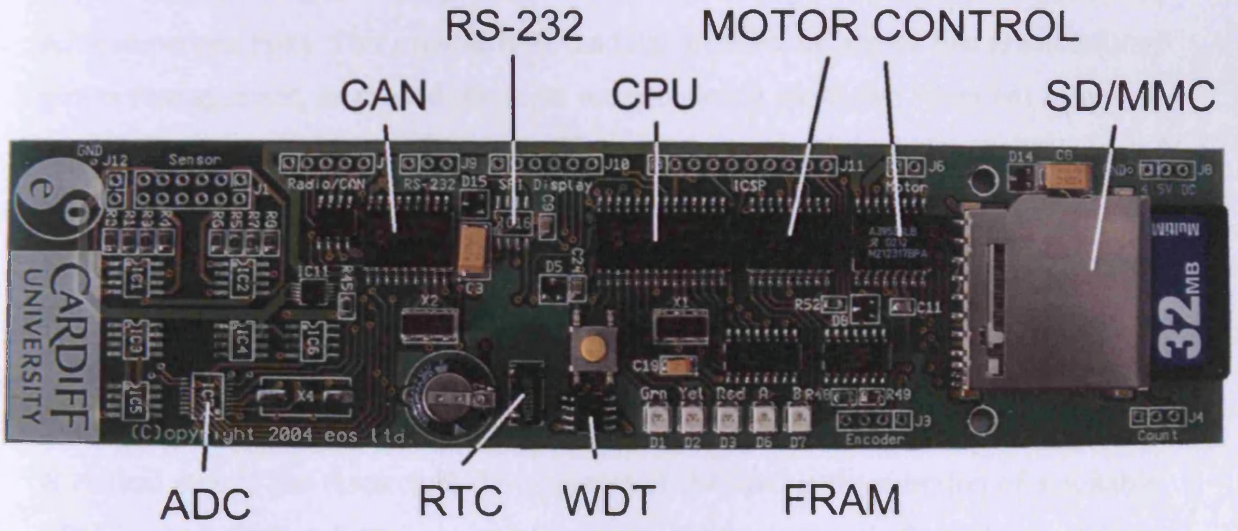


Figure 6.1 Single Board Computer (SBC)

A high level of integration, reduced component and connector count make this SBC small, light, power efficient in comparison to many other commercially available SBCs. The hardware and firmware were developed concurrently and integrated early on in the design process to yield the best possible performance in terms of power consumption and stability. For example, "Windows XPe" (embedded) or Linux operating systems running on a commercially available SBC may not have direct access to shutdown various hardware devices to conserve power unless the platform specifically supports it and a driver has been created. The SBC hardware architecture is designed with the aim

of maximising system reliability and reducing power consumption to an absolute minimum. This is achieved by partitioning the hardware into separate functional blocks where each block is dedicated "slave" device optimised to assist the master in completing a specific task. In this application the tasks are communication, time keeping, measurement, motor positioning and power management. These blocks are sometimes referred to as hardware accelerators because they operate concurrently with the "master" CPU to improve speed performance of the system. The master communicates with the slaves via a dedicated serial bus (serial peripheral interface, SPI), interrupt and power enable pins. Slave blocks can raise interrupts to signal occurrence of internal and external events (such as a measurement being completed or an incoming message) to trigger the master, which takes appropriate action to process the interrupt request. The ISR (interrupt service routine) firmware residing in the master (and some slaves) can configure the system according to its instantaneous computational requirements by means of power enable pins. These control pins are used to gate supply voltages to components as and when they are required by the system to achieve certain tasks. This event-driven, modular architecture allows fine granularity of power management, as unused resources are completely shutdown when not in use. A spreadsheet was created *[see **Appendix E**]* to enable accurate calculations to be made of expected system power requirements for different deployment scenarios. Finally, the form factor of the SBC was determined by the physical dimensions of other components within the pressure case such as the battery pack and motor/gearbox.

6.2 Central Processing Unit (CPU)

A critical step in the research & development of the SBC is the selection of a suitable CPU or computational core on which to base the hardware platform. Each node is battery powered and is expected to function reliably for many months, so a low power, low-performance microprocessor is an appropriate core for this application. The term "low-performance" is typically used to describe 8 or 16-bit microprocessors with limited program, data memory and hardware resources). For longer duration deployments, the capability to switch the device into sleep mode becomes essential. The device should obviously have adequate program space to accommodate the existing firmware code, but also space for any future improvements or features. The ability to

program the device in a high-level language, such as 'C' when embedded in the circuit is useful for testing purposes and for future firmware upgrades.

An enhanced RISC (reduced instruction set computer) Microchip 18-bit core PIC (Peripheral Interface Controller) microprocessor was selected as the core on which to base each SBC. This low-power device is equipped with on-board services such as serial communication UARTS, timers, PWM and a total of 28 pins, of which 22 are available for general purpose digital input/output. The enhanced RISC architecture provides fast computation and enables code to be executed at the microprocessors internal clock speed (five million instructions per second with an external 20MHz crystal). This microprocessor also has a C compiler optimised architecture/instruction set.

6.3 Serial Peripheral Interface (SPI) Bus

There are two realistic propositions for a local serial communication bus between devices on a node circuit board, the SPI bus and the I2C (Inter-IC) bus. SPI was chosen because of its relatively low protocol overhead compared to the I2C bus and also because of the wider selection of low voltage (3.3V) SPI devices currently available on the market to choose from. The SPI bus is a very loose standard for controlling almost any digital electronics device that accepts a clocked serial stream of bits. SPI is cost effective, in that it does not take up much real-estate on an integrated circuit (IC), and effectively multiplies the pins, the expensive part of the IC. It is easily implemented by "bit-bashing" in firmware and a few standard IO pins of a microprocessor. It is the most flexible choice when many different types of serial peripheral devices may be present, and there is a single "master" microprocessor.

The microprocessor on the SBC accesses peripheral hardware via the local SPI bus and chip-select scheme *[figure 6.2]*. In this scheme the microprocessor acts as a "master" and selects a "slave" using the 3:8 de-multiplexer (74HC138 3-to-8 Line Decoder DS005120 Fairchild Semiconductor Corporation, 1983). Only the selected slave device drives its output, all other slaves are in a passive high impedance state. The output remains active as long as long as the slave is selected by its address. This approach rigidly enforces bus arbitration at the hardware level by ensuring exclusive selection of

only one slave at any time. The 3:8 de-multiplexer also reduces the number of digital control pins (A0, A1 and A2) the master needs to dedicate to the task of chip selection. All the SPI peripheral devices used in this circuit design are low power, operating at

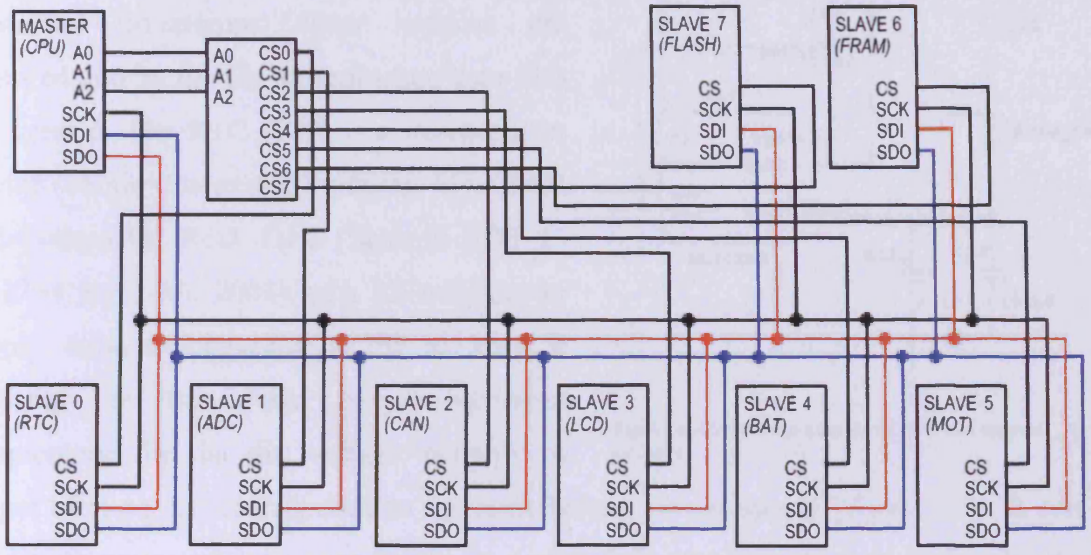


Figure6.2 SPI Master with independent slave devices

3.3V and can be put into "sleep" mode by the master microprocessor to further reduce power consumption to make them true "micro-power" devices. In sleep mode a node would be drawing an immeasurably small current from the batteries. Self-discharge, due internal cell resistance of the battery would be the predominant cause of power drain.

Partitioning the hardware by use of task specific devices that communicate over a common SPI bus presents the master with a well-defined and uniform interface. This simplifies firmware design and reduces the number of lines of code (NOC) required to implement the internal bus control algorithm.

6.4 Real-Time Clock (RTC)

The RTC is central to the node design and is utilised by the on board DAQS to create formatted timestamps when samples are acquired and by the file system when data files are created. The RTC (IC9) is a micro-power device (Maxim Integrated Products, MAX6902 SPI-Compatible Real Time Clock in SOT-23 19-2134 Rev. 0.0, 2001) with it's own power supply back-up capacitor (C23) to keep it "ticking" in the event of a brownout (momentary dip in the voltage supply) or longer term power interruptions to the node battery power supply *[figure 6.3]*. It can handle spurious and long-term breaks in power for up to two days. It is possible to implement clock and calendar functions performed by the RTC in firmware, however a dedicated external RTC frees the "master" microprocessor for higher level duties and reduces the number of lines of code (NOC) required. The HLP supports RTC register updates, meaning nodes are capable of synchronizing their RTC to an external time source.

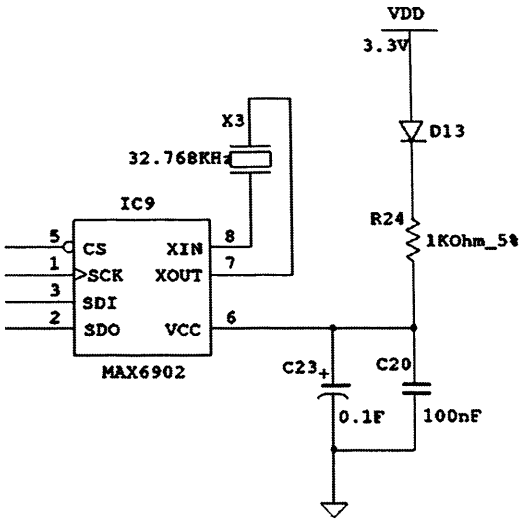


Figure 6.3 Schematic diagram of RTC and support circuitry.

6.5 Analogue-to-Digital Converter (ADC)

The ADC (Burr-Brown Products, ADS1242/1243 24-bit Analogue-To-Digital Converter SBAS235, 2001) is an integral part of the DAQS hardware *[figure 6.4]*, where it is required to digitise voltage signal outputs from the sensors. In this application the sensor is a thin-film platinum resistance temperature device (RTD), however other types of transducers may be connected with suitable signal conditioning to ensure that the input voltage to the ADC (IC4) stays within a 0 to 3V range. The ADC is an eight channel, 24-bit delta-sigma type with 1024 tap finite impulse response (FIR) comb digital filter with notches in the frequency response 50Hz and 60Hz to reject mains induced noise. There is an onboard programmable gain amplifier (PGA) that can be set for gains from 1 to 128 times in power of two steps and selectable high impedance (5MΩ) buffer. The ADC also has sleep and stand-by mode capability to minimise power consumption when not in use. To further reduce power consumption to

an absolute minimum the 200μA power source can also be switched off by the DAQS via an analogue switch (IC13).

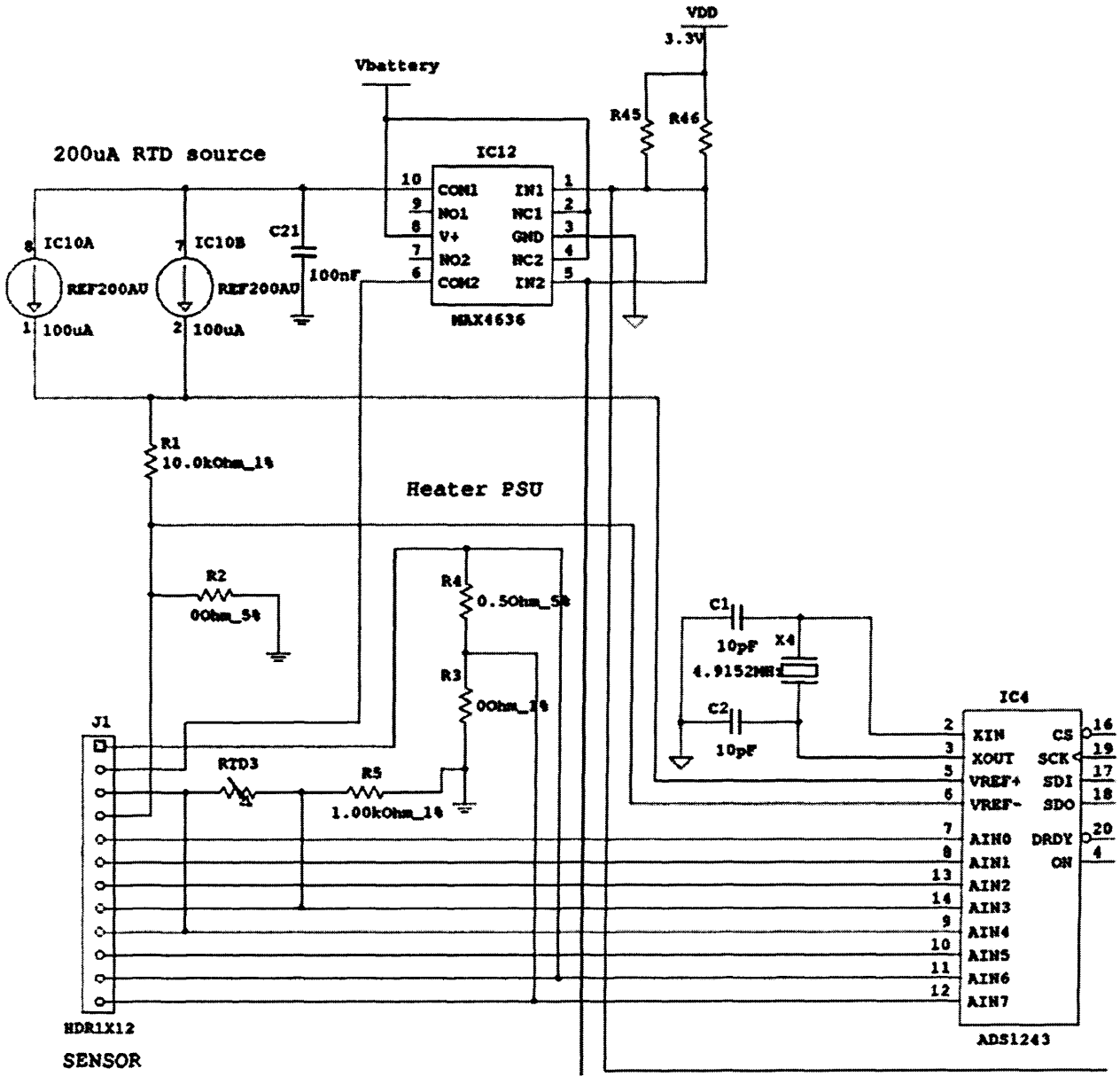


Figure 6.4 Schematic diagram showing DAQS hardware.

Internal buffering can be selected to provide very high input impedance (5MΩ) for direct connection of the low-level voltage signal generated by the RTD sensing element. The programmable gain amplifier (PGA) allows the DAQS to be configured for use in different environments. For example, a high PGA gain would be used when performing high-resolution temperature profiling throughout a water column. Conversely, the temperature measurements of water temperature taken at the mouth of a hydrothermal

vent require a wider dynamic range and lower resolution, so the PGA gain would be lowered to accommodate the larger signal output from the RTD.

6.6 Transducer

A transducer is a device that transforms one form of energy into another. In this case, temperature and fluid flow rate are converted into an electrical signal or voltage. The inaccessible and harsh marine environment in which the transducer must function is similar to some industrial and commercial applications. It must be able to withstand high temperatures, pressures and some unpleasant chemistry, so reliability is paramount. The measurement point is not in the same place as the indication or control point and there is a requirement for further processing of the measurement in controllers, loggers or computers. There are three types of transducer commonly used in industrial process and control: resistance temperature devices (RTDs), thermistors and thermocouples.

A PT100 DIN IEC-751 thin-film RTD was selected for this application because it has a relatively linear response over a wide temperature range offers excellent accuracy, best long-term stability with the ultimate benefit that sensors will be interchangeable. In many industrial processes, for example, cracking towers in petroleum refineries, a group of temperature measurements must be related to one another. A series of platinum RTDs that sense slow changing temperatures can be configured into a resistive ladder (Linear Technology Corporation, LTC2404/LTC2408 4/8-Channel 24-Bit µPower No Latency $\Delta\Sigma^{tm}$ ADCs, 1999). This approach allows a single excitation current passed through the entire ladder, reducing total supply current consumption and eliminating current source mismatch and drift errors. An accurate 200µA constant current source (REF200 Dual Current Source/Current Sink PDS-851D Burr-Brown Corporation, 1988) excites the RTDs and a precision reference resistor. The resistance of any of the RTDs is determined by measuring the voltage across it, as compared to the voltage drop across the reference resistor. The accuracy of a measurement from this ratiometric implementation is dependent on the reference resistor having a low temperature coefficient. High precision 100R and 200R resistors may be inserted in series with the RTDs in ladder. Insertion of two known source resistors into the ladder makes it feasible for the DAQS to perform a full self-calibration for offset and gain correction.

The RTDs are bonded to pure gold "thermal windows" with high thermal conductivity epoxy resin and the complete sensor probe assembly is encapsulated in thermally resistive epoxy. Although these RTDs are capable of operation over -200°C to 650°C temperature range, the resins are only rated to a 250°C maximum. The sensor probe and schematic are shown in *figure 6.5* below.

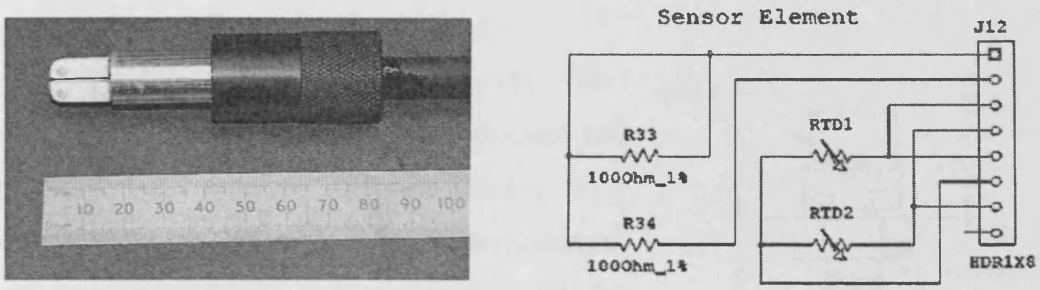


Figure 6.5 Prototype calorimetric flow sensor and schematic diagram.

6.7 Controller Area Network (CAN)

The CAN hardware interface is based on a dedicated CAN controller device (Microchip Technology Inc., MCP2510 Stand-Alone CAN Controller with SPI DS21291E, 2002) and low voltage, low-power transceiver (Burr-Brown Products, SN65HVD230 3.3V CAN Transceivers SLOS346E, 2001). This controller handles the CAN protocol, message packets and data collisions, freeing the master microprocessor from these lower level duties. Because the CAN hardware manages the entire packet, including CRC checks, the overhead on the processor is far less than it would be for an equivalent serial port. Failed messages are retried automatically, with no software interaction. More detailed information on the CAN bus protocol can be found in ***Chapter 3*** *Communication System Design*.

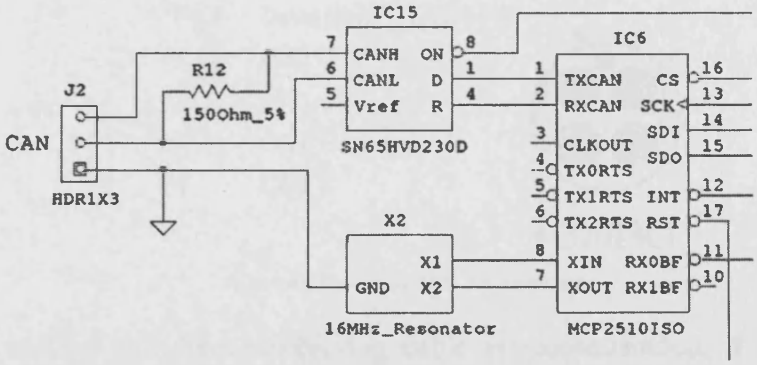


Figure 6.6 Schematic diagram showing CAN hardware.

CAN bus was originally developed for the automotive market where power consumption and battery life are important. For this reason the CAN controller has a sleep mode, where it is be awakened when a message is received on the bus. CAN is desirable for this application because it is fault tolerant and its message transmission error rate is low (Bosch 1991) with fantastically reliable data transfer calculated at one error every thousand years.

For this specific marine application the SBC computer is housed in a titanium pressure case and the physical bus interface between nodes is a specialised four-pin *[figure 6.8]*, sub-miniature, deep-sea connector that is pressure-rated to 10,000 PSI *[figure 6.7]*. The titanium pressure cases and connectors underwent further certification (up to 9,500 PSI external pressure) in the laboratory with a pressure chamber rig. Two of these bulkhead connectors are mounted on the base of every node pressure case housing to facilitate daisy-chaining of nodes. Nodes can easily be added and removed from the system network for repair or re-configuration of the instrument without the need for significant re-wiring or disruption the rest of the bus. One disadvantage of this topology is that damage to the cable could cause a break in the bus and loss of the termination resistor at one end of the data lines, so more errors will occur. However, the CAN protocol supports lower data rates (125Kbps) where the termination resistor is not so critical. It is therefore possible for the instrument to continue functioning as two independent units if suitable error detection and adaptive system firmware is implemented.

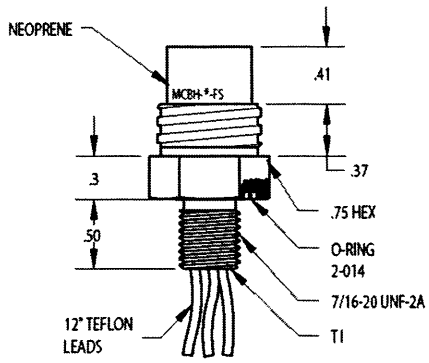


Figure 6.7 CAN bus connector.

Pin #	Description
1	GND
2	+5V
3	CAN_H
4	CAN_L

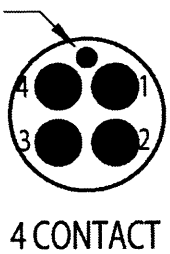


Figure 6.8 Pin-out of CAN bus connectors.

A shielded twisted pair inter-connecting cable is recommended, if the instrument is operating in an electrically harsh environment where it would be exposed high levels of

electromagnetic interference (EMI) or radio frequency interference (RFI). However, this is not an essential requirement for typical marine deployments.

6.8 Power Supply

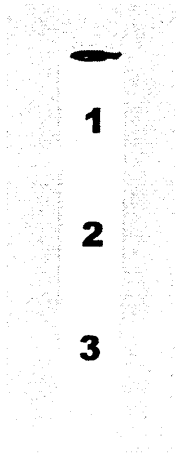
A decision was made early on in this project to use a 3.3V power supply rail for all the digital electronics on the SBC. This was dictated by the voltage requirements of suitable flash memory devices available at the time with the benefit of a significant decrease in power consumption compared with 5V. The 3.3V rail is derived from a low drop out linear voltage regulator (STMicroelectronics, LE00AB/C Series Very Low Drop Voltage Regulators with Inhibit, 1998) which provides a stable output from wide voltage input, allowing the SBC to operate from a power source with 3.6V to 6V range.

Several different rechargeable battery technologies were considered for powering node electronics including, nickel-cadmium, lead-acid and lithium-ion chemistries:

- Nickel-Cadmium - NiCd batteries are capable of supplying high currents, have a long service life and exhibit a relatively flat discharge characteristic over time. The most appropriate charging method is constant current and pulse charging. A periodical full discharge is important to prolong life. The major disadvantage of NiCd technology is low energy density and self-discharge.
- Lead-Acid - Lead-acid batteries have one of the lowest self-discharge rates of all rechargeable battery chemistries, making them appropriate for standby applications. Charging method requires a constant voltage. Disadvantages are low energy density and limited number of full discharge cycles. Also terminal voltage should not be allowed to drop below 2.1V or irreversible damage will occur because of insoluble lead sulphate deposition on the plates inside the cell.
- Lithium-Ion - Li-Ion offers high energy density, low weight and low self-discharge in comparison to NiCd batteries. High terminal voltage of 3.6V means that the SBC could be powered from a single cell. The major limitation of Li-Ion batteries is the requirement to maintain voltage and current within safe operating limits during charge/discharge cycle. Batteries are prone to failure after two or three years even when not in use.

Although Li-Ion technology offers high energy density it was rejected because of longevity and safety issues when used in the deep marine environment. Li-Ion (and lead-acid) batteries are all too easily damaged because of neglecting to keep charge/discharge currents within safe operational limits. NiCd batteries are more forgiving to this kind of mal-treatment, however all chemistries are damaged by overcharging because the cells begin to loose electrolyte via "gassing" and plates may even buckle if overheating occurs. Ni-Cads have the capacity to deliver high instantaneous currents (higher than their Ni-Mh replacements), making them well suited for motor start applications such as this. For these reasons they were selected as a power source for node electronic systems.

A battery stack was specified *[figure 6.9]* consisting of three high temperature conformance NiCd cells to deliver a nominal 3.6V to the voltage regulator input.



- Nominal output voltage: 3.6V
- Capacity (energy storage): 2.5Ah
- Battery technology,: Nickel Cadmium (NiCd)
- External length / height: 150mm
- Diameter: 26mm
- Maximum operating temperature: 65°C
- Minimum operating temperature: -20°C
- Terminal type: Solder tag
- Weight: 216g
- Internal impedance: 15 ~ 22 mΩ / cell
- Cycle life: ≥500

Figure 6.9 Rechargeable battery pack

A built-in power management system (PMS) based on a 12-bit core PIC microprocessor (Microchip Technology Inc., PIC12F629/675 Data Sheet 8-Pin FLASH-based 8-bit CMOS Microcontrollers DS41190C, 2003) continuously monitors battery voltage and supervises the amount the charging current. There is no SPI hardware support on PIC12F675 microprocessor, so SPI is implemented using "bit-bashing" code.

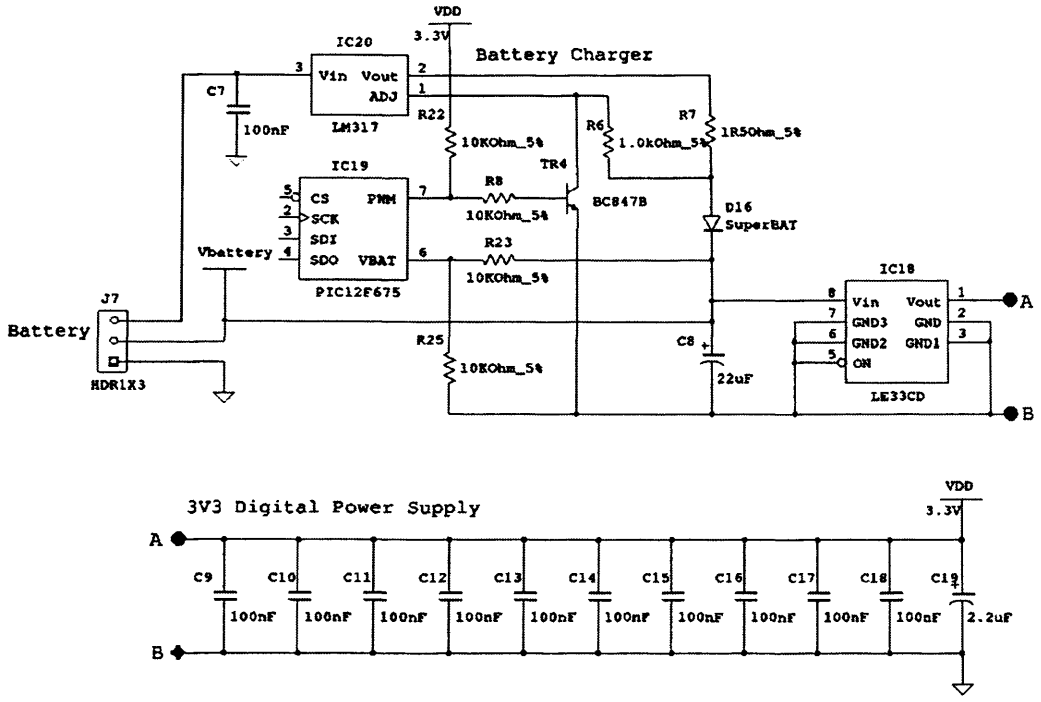


Figure 6.10 PMS schematic diagram.

The charger circuit is built around a LM317 (national Semiconductor, LM117/LM317A/LM317 3-Terminal adjustable Voltage Regulator DS009063, 2006) constant current charging circuit under the direct control of the slave microprocessor *[figure 6.10]*. The voltage between the adjustment terminal and the output terminal of the LM317 (IC20) is always 1.25V, so by connecting the adjustment terminal to the battery load and placing a resistor (R7) between the battery and the output terminal, a constant current of 1.25/R is established. A value of 3.3Ω for R7 sets the maximum possible charging current at about 350mA. A diode is used in series with the input to prevent the batteries from applying a reverse voltage to the regulator when the external power is not available. The microprocessor (IC19) can control the charging current with a PWM pin which is tied high via R22, to turn TR4 on and disable charging. This safety precaution prevents the batteries from being overcharged if the microprocessor fails for any reason. By varying the duty cycle of PWM it is possible to adjust the charging rate. IC18 provides a regulated 3.3V supply rail to all active digital devices, which are decoupled with 100nF capacitors situated in close proximity to them.

6.9 Motor Controller

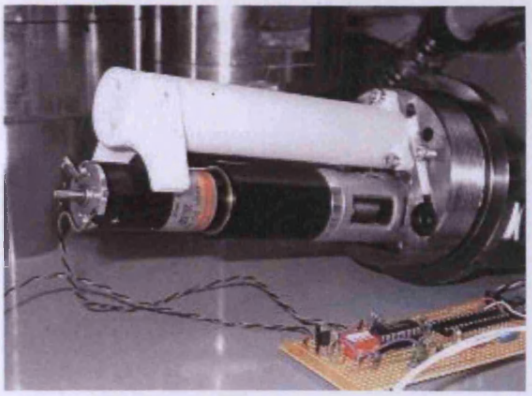


Figure 6.11 Maxon motor/gear box assembly and prototype controller hardware undergoing test at Cardiff University, UK (2003).

The motor controller circuit *[figure 6.12]* is a sub-system of the MPS and comprises an intelligent H-bridge device (Allegro Microsystems, Inc., 3953 Full-Bridge PWM Motor Driver 29319.8, 1995) and second slave PIC microprocessor (Microchip Technology Inc., PIC16F818/819 Data Sheet 18/20-Pin Enhanced Microcontrollers with nano Watt Technology DS39598C, 2002). The slave microprocessor is programmed to accept single bytecode SPI commands (from the CPU master microprocessor), interpret them to drive pins on the H-bridge device and ultimately actuate the motor *[figure 6.11]*. This architecture minimises cross-linkage and effectively encapsulates motor control tasks (valve positioning, pump control, etc) within a functional block to free the main microprocessor to handle higher level duties such as communications and management of node systems. It also completely partitions the code in the slave from the master during the development stage making it easier to maintain and encouraging better, faster, cheaper and more reliable firmware.

Maxon motors and planetary gearboxes were chosen for this application because of their low start/operating currents, compact size, low electrical noise characteristics and proven track record in extreme environments. This meant that the batteries and the H-bridge controller could to be de-rated leading to an overall reduction of the pressure case dimensions and cost for a complete node. The H-bridge device features safety over-current protection and dynamic braking capability. A pulse width modulation (PWM) control scheme is used to set motor speed for "soft-starting" and accurate positioning. This drastically reduces power consumed by the motor and electrical noise from the point at which the brushes contact the rotating commutator.

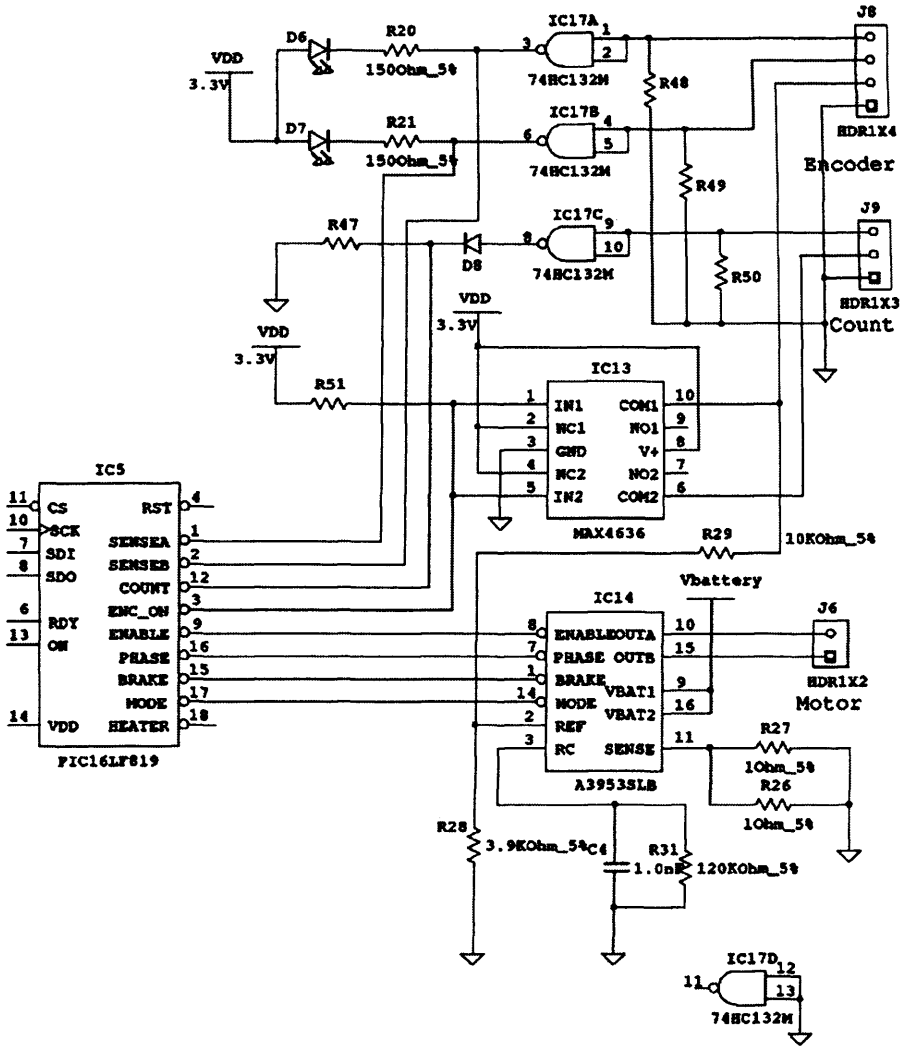


Figure 6.12 Motor controller schematic diagram.

6.10 Encoders

To achieve accurate movement and speed control of the motor shaft some form of sensor is required to feedback position information to the slave processor. The MPS uses two totally independent optical encoder circuits to achieve this reliably and a level of redundancy. For example, firmware has been written to open and close a valve with either of the encoder circuits. One incremental encoder counts motor shaft revolutions. This data is further processed to give information about motor speed and position. A second encoder *[figure 6.13]* after the gearbox gives absolute positions for either of the two valve ports being open or in the closed position.

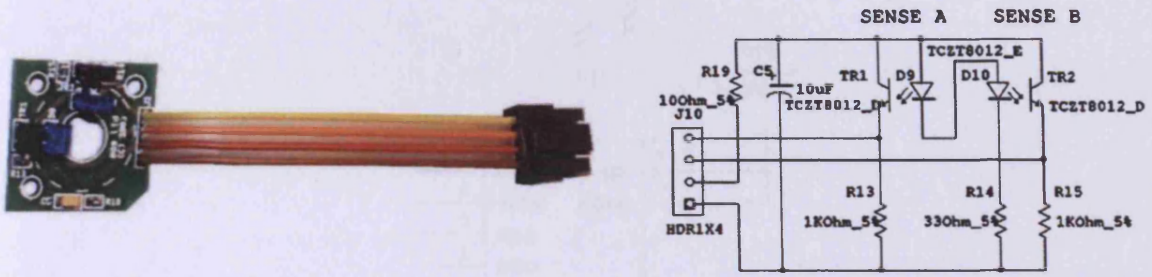


Figure6.13 Absolute position encoder and schematic diagram

Low-power matched infra-red optical detector/emitter pairs are used in both encoders. Being non-contact sensors, they offer good immunity to mechanical noise (foreign particles such as dust ingress or condensation), vibration and electrical noise (voltage spikes from the motor stopping and starting). The signal from the detector is post-processed by feeding it through a complementary metal oxide semiconductor (CMOS) Schmitt trigger *[refer back to figure 6.15]* to introduce a little hysteresis and eliminate multiple edge transitions; this ensures a clean CMOS logic level signal is presented to the microprocessor digital inputs. A solid-state switch (Maxim, Fast, Low-Voltage, Dual 4Ω SPDT CMOS Analog Switches MAX4636, 2003) is used to completely shut the encoder circuitry down when not in use to further reduce power consumption when the MPS is not in use.

6.11 Ferro-electric Random Access Memory (FRAM)

A SPI FRAM device (Ramtron International Corporation, FM25CL64 64Kb FRAM Serial 3V Memory Rev. 2.1, 2003) functions as a non-volatile storage medium and contains the Object Dictionary; a formal representation of node configuration settings and process data ***[Chapter 4** Implementing CAN and CANopen]* that is accessible to all nodes connected to the network. The complete structure of the Object Dictionary is described in the CANopen standard (Pfeiffer 2003). The configuration settings play an important role during power on reset (POR) to ensure that state machine initialises the system in a known state i.e. its previous state after power-down or reset.

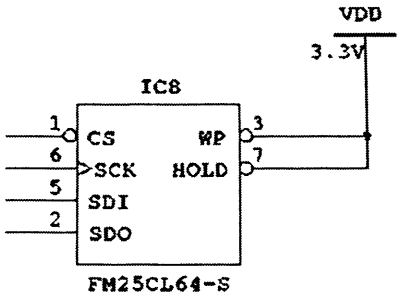


Figure 6.14 Schematic diagram showing FRAM used for non-volatile data storage.

FRAM was chosen over electrically erasable programmable read-only memory (EEPROM) because of its superior electrical performance in terms of low-power consumption, speed and durability. It is an extremely durable device for this purpose as it has virtually unlimited write cycles and zero wait states for writing - for comparison EEPROM has a maximum of 1,000,000 write cycles (Microchip 1993). This also simplifies firmware design.

6.12 Secure Digital/ Multi-Media Card (SD/MMC) Mass Data Storage

The SD (SanDisk Corporation, Secure Digital Card Product Manual 80-13-00169 Rev. 1.9, 2003) or MMC (SanDisk Corporation, MultiMediaCard Product Manual 80-13-00089 Rev. 5.1, 2002) card constitutes part of the file system (FS). The card is used for storing large amounts of data acquired by the DAQS in hexadecimal or text format in files. Data files can be recovered by physically removing the SD/MMC and inserting it into a personal computer (PC) or personal data appliance (PDA). The master CPU can detect insertion/removal of the card via an interrupt line and perform the appropriate action.

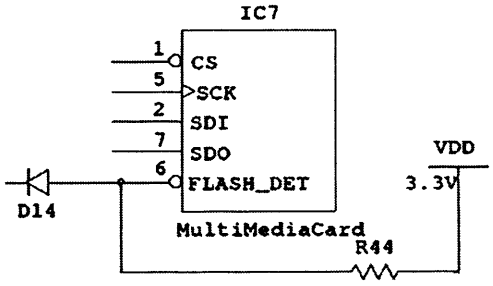


Figure 6.15 Schematic diagram of SD/MMC hardware.

Although the Interfacing the SD/MMC card to the SPI bus *[figure 6.15]* was a relatively simple part of the hardware design, implementing the FAT16 FS and SD/MMC driver

required several thousand of lines of support code and took many months to debug so that it functioned correctly *[see **Chapter 7** Results].*

6.13 Liquid Crystal Display (LCD)

There is the facility to connect an external LCD via an SPI connector on the main circuit board. The main purpose of this is to aid test and debugging, however the SPI can also be used for interfacing another board to add more functionality at a later time.

6.14 Watch Dog Timer (WDT)

The WDT will perform a hard reset of a system unless some sequence is performed that generally indicates the system is alive, such as a write operation from an onboard processor. During normal operation, firmware strobes the WDT at regular intervals to prevent the timer from running out. WDT duties are performed by dedicated external hardware (Dallas Semiconductor, DS1832 3.3 volt MicroMonitor Chip 112099), rather than the microprocessors internal one, for high system reliability. The WDT is strobed every 250ms in a controlled manner by a single task within the RTOS framework. This hardware can also detect low voltage (brownout) conditions and reset the PIC microprocessor to prevent the system behaving in an unpredictable manner.

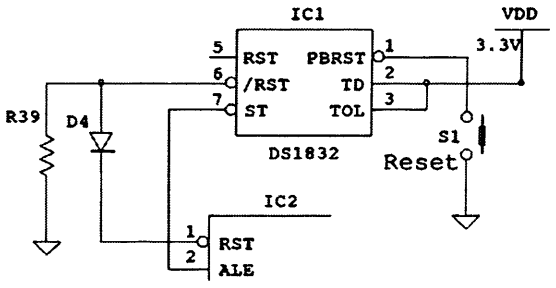


Figure 6.16 Schematic diagram of watchdog timer circuitry.

The importance of a dependable WDT cannot be over emphasised. It is a crucial component of the hardware and firmware in high reliability embedded systems and a mechanism to restart the program if it wanders off or the hardware fails (Gannsle 1992). For example, in 1994 the Clemantine spacecraft deployed to map the moon was abandoned after its software crashed causing it to turn on thrusters and dump fuel. A WDT was not used because of objections by the lead designer. In contrast the Pathfinder mission was saved by a WDT when its firmware crashed. The team found the bug and uploaded new code to a target system 40 million miles away on Mars (Ganssle 2004).

The WDT is an essential component of any radiation hardened system. If radiation causes the processor to operate incorrectly, it is unlikely the firmware will work correctly enough to clear the WDT. The watchdog is the last line of defence and eventually times out to force a hard reset to the system.

6.15 Input/Output (IO)

A legacy serial RS-232 interface *[figure 6.17]* allows each node to be directly connected to a host computer or PC. It supports data transfer rates up to 19,200 Baud. The host acts as a HMI to facilitate system configuration for autonomous use or direct manual operation.

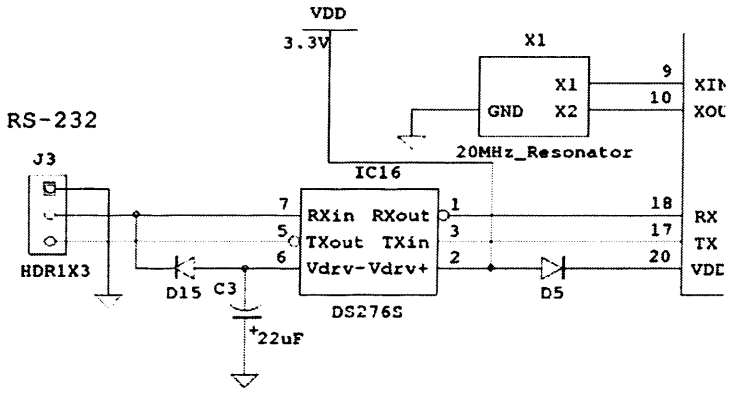


Figure 6.17 RS-232 interface schematic diagram

Several light emitting diode (LED) indicators are included on the circuit for system diagnostics. This is common practice and extremely useful in the test and debugging stage as it gives a simple and clear visual indication of the systems operational status. The green LED flashes every time the WDT is strobed. Yellow LED indicates when the system is busy, or more accurately when the RTOS scheduler is busy. The red LED indicates if an error condition has occurred and will remain lit until the error is cleared or fixed. Typical error conditions include external hardware failure, write errors and timeouts. There are also two orange LEDs to indicate the status of the valve position encoder *[refer back to figure 6.12]*.

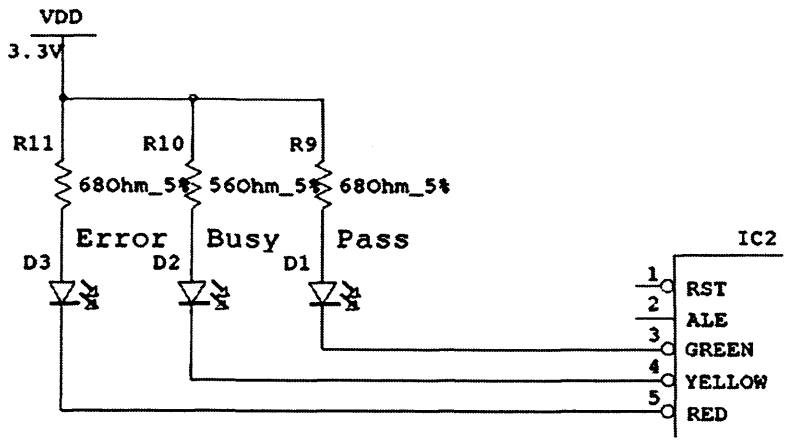


Figure 6.18 Schematic diagram showing LEDs.

6.16 Printed Circuit Board (PCB)

The PCB is of two-layer, fibre-glass reinforced epoxy design with plated through holes finished in tin with green solder resist mask. Much of the bottom layer is a ground plane and a substantial amount of copper on top layer is used as a 3.3V power rail. Components were placed on both sides of the board in order to keep PCB size to a minimum. Standard pitch surface mount components (SOIC and 0805 outline) are used predominantly with some fine-pitch and through-hole devices. This was a deliberate decision to allow high component density yet retain the ability to re-work and modify the board in-house without requiring specialist tools. The high component density made the task of constraining component layout on a two layer PCB challenging, however it has the advantages of being less expensive to manufacture and easier to debug, as there are no "buried" vias or hidden tracks.

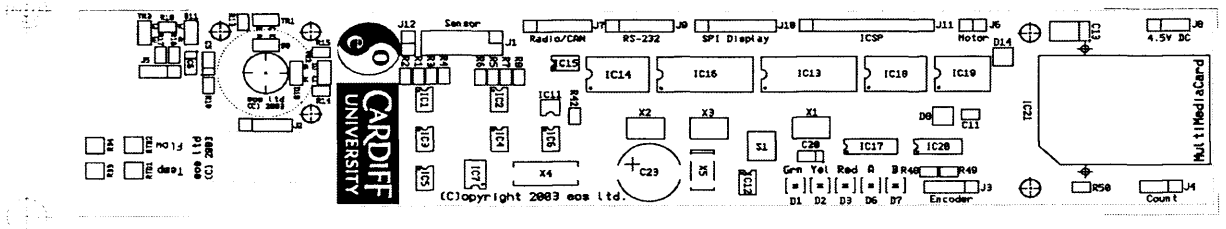


Figure6.19 SBC (mainboard), enoder and sensor PCB silkscreen.

The PCB was drawn with the aid of Electronics Workbench's "Ultiboard" CAD software to create gerber design files. It underwent three revisions over an eighteen month period before the final batch of production boards was built. Prototypes were assembled by hand, tested and modified at COAS, Oregon State University and PCBFabExpress fabricated the full production run of sixty PCBs. To minimise costs all

four boards (SBC, encoders and sensor) were placed on a single wafer in the design file *[figure 6.19]*. Megatech of Oregon undertook final assembly of the production PCBs.

6.17 In-Circuit Serial Programming (ICSP)

ICSP facilitates the production and development process by making debugging easier and future upgrades of the firmware possible. A SIL (single in-line) connector and isolation circuitry were included on the PCB design to allow firmware updates on the CPU master and motor controller slave PIC microprocessors when they are soldered in place. An "MELabs" serial programmer with command-line control capability was used to download hex files into the target processors. It connected directly to the USB port on the host development PC running the programmer software under "Windows" 98/Me/NT/2000/XP. A makefile was written to control the programmer and automate the process as described in ***Chapter 5***.

6.18 Summary

A series of three prototype SBC circuit board revisions were developed and tested before proceeding to manufacture a complete batch of sixty production PCBs. Megatech of Oregon did an excellent job of soldering components onto these PCBs during the final assembly. A visual inspection and electrical test (performed using a basic RTOS kernel with) was made on a sample selection of ten boards *[figure 6.20]* and revealed zero manufacturing defects. Complete schematics for the SBC and a detailed bill of materials (BOM) can be found in ***Appendix C***.

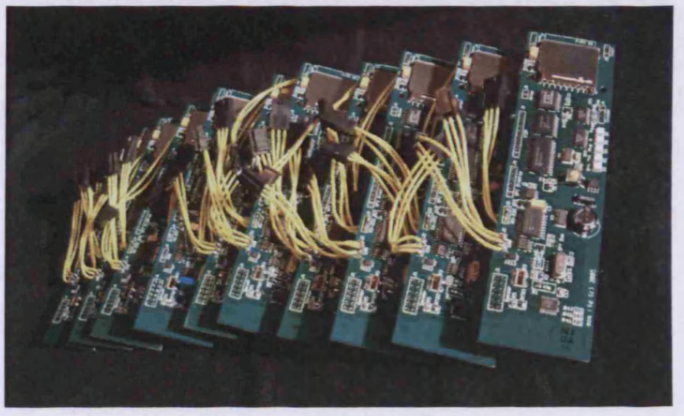


Figure 6.20 Ten of sixty assembled SBC circuit boards.

Chapter 7 Testing and Results

The preceding chapters outlined the conceptualisation, development and implementation of an open and flexible network architecture capable of supporting reliable machine operation in extreme environments. The "lego-brick" modularity made it possible to build and test instrument sub-systems incrementally in the laboratory and in real sea-trials where both useful engineering and science data were obtained. The first section of this chapter gives a detailed description of the experimental work and the performance results obtained from applying various node sub-systems to geophysical monitoring and sampling fluids in the deep ocean. The second section then progresses to describe the approach taken to integrate these nodes into a network to realise a decentralised "plug and play" machine system architecture and the results of the tests that followed.

7.1 Validation of Node Operation

Before any sea-trials were undertaken, the node electronic hardware was tested in the workshop. Firmware drivers were written and incrementally introduced into the RTOS framework. During this stage, the driver firmware typically underwent two or three revisions; initially to establish communication with a device and then further refinements to reduce size, efficiency and readability of the code. Once confidence had been gained with the hardware/firmware interaction, it was possible to move forward and test the various sub-systems in real sea trials. This section describes the bench tests performed and the results obtained from practical deployments at sea.

7.1.1 Valve Control Consumer Node

A test harness was initially constructed in 2003 whilst working at Cardiff University *[shown in figure 6.11 of **Chapter 6**]* to circumvent the immediate requirement for writing a functional firmware driver and facilitate immediate testing of motor controller hardware. The harness was used to test that the H-bridge device and battery power source were capable of supplying adequate current to drive the motor/gearbox combination under full load i.e. with the valve assembly connected. Also, tests were undertaken to ensure that the gearbox had enough mechanical advantage to physically open and close the valve. Under nominal load the motor draws 250mA i.e. when the valve is rotating, however during start-up requires at least 400mA to overcome physical friction of valve

ball. This figure could conceivably be greater if the valve is subjected to temperature extremes, as mismatches in expansion coefficients of valve parts manufactured from different materials (some are PEEK and some titanium) will affect clearances. To compensate for this, some de-rating was applied in the H-bridge over-current protection circuit, which was set to trip at 500mA.

Full device driver firmware code was completed several months later at Oregon State University, US where further bench tests were undertaken to validate the complete MPS hardware and firmware. The tests checked that SPI communication between the master processor and slave was reliably established and that the byte commands were interpreted correctly *[source code for SPI slave and master devices is given in* ***Appendix Q.7, Q.8*** *and* ***Q.13****]*. Partitioning the motor control firmware within a second slave processor and coupling it to the master via a standardised SPI communication protocol proved advantageous, as it allowed code changes to be made to the master without perturbing the stability of the motor control firmware. Finally, the driver code utilised the absolute position encoder for sensing the valve state when opening and closing the valve; the incremental encoder is reserved for monitoring motor speed.

Figure 7.1 Preparing the instrument for deployment.

In order to ascertain how the valve nodes performed in real situation, a sea-trial was arranged so that a single bottle assembly could be deployed several hundred meters below the Pacific from the oceanographic vessel "Elakha" *[figure 7.1]* during 2004. Unfortunately, the system failed to function under water because of incorrect mechanical assembly with the consequence of allowing seawater into the pressure case. This caused severe corrosion damage to the electronics, motors and batteries and it was not possible to service them. Despite this setback, further tests were performed in a sealed pressure test chamber in the laboratory, at 9500 PSI. Test procedures were more limited because it was not possible to communicate directly with the system and the water temperature in the test chamber was warmer than on the seabed (typically 4°C), however it was established that the MPS functioned correctly in a high- pressure aqueous environment. Tests confirmed seal integrity, that the motor/gearbox had sufficient mechanical advantage to open and close the valve and

that the control electronics and power supply could deliver adequate current to drive the motor.

Figure 7.2 Instrument MPS testing at sea. The "Elakha" heading away from Hatfield Marine Science Centre (OSU) on the Pacific Ocean.

7.1.2 Temperature Sensor Producer Node

A substantial part of the temperature sensor node development effort involved the investigation and implementation of calibration procedures. The calibration protocol was then embedded within the CANopen Object Dictionary so that node could produce processed measurement data with inherent meaning, whilst also ensuring that calibration was traceable. This follows the present trend of embedding signal processing within the node housing to realise a "smart transducer". This section gives a detailed description of temperature and flow-rate calibration tests and the results obtained.

7.1.2.1 Temperature Calibration

The PT100 RTD response is approximately linear over a narrow temperature range, however this assumption yields significant errors over wider ranges. For example, between 0 and 100°C, the error at 50°C would be 0.4°C. To enable precision measurement over a wide range, it was necessary to correct the resistance to give a more accurate representation of temperature. This correction was made in firmware by applying the polynomial (from DIN IEC-751) given in Eq. 4.23 as shown in *figure 7.3* below.

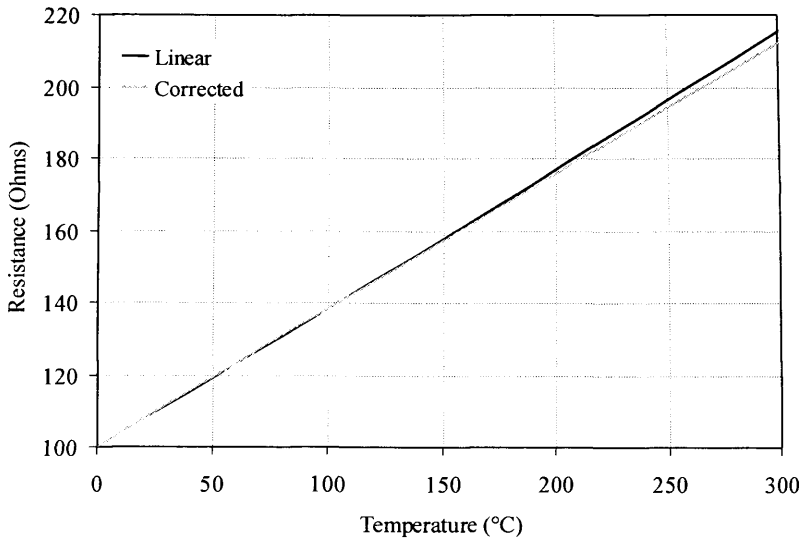


Figure 7.3 Temperature Calibration Characteristic Curve for PT100 RTD.

The accuracy of the ADC hardware and conversion routine was initially validated by substituting the RTD sensor element with precision reference resistors of value 100, 200 and 300Ω. The table below shows calibration results after gain and offset calibration has been performed.

Dummy Load Resistance (Ω)	Mean Measured Temperature (°C)
100	0.1 (0.0)
200	265.9 (266.0)
300	559.1 (558.0)

Once it was established that the DAQS was functioning correctly the dummy load was replaced with the sensor head so that basic calibration tests could be performed. The overall accuracy of the system was tested by first placing the sensor head into ice water (0°C) to obtain the ice point resistance (100Ω) and then into boiling distilled water (100°C) where the RTD resistance is 138.5Ω. In each case the sensor head was given five minutes to reach equilibrium with the environment before recording the measurement. A standard laboratory mercury thermometer was used as a reference to establish accuracy within 0.5°C.

7.1.2.2 Flow-rate Calibration

Flow measurements are based on a calorimetric principle where the two matched PT100 devices are housed in the sensor probe tip and thermally insulated from each other. One of these devices monitors the reference media temperature while the other is

indirectly heated several °C above this. As fluid moves across the sensor, heat is carried away from the PT100 resulting in a resistance change i.e. an output signal that is proportional to flow-rate *[figure 7.4]*.

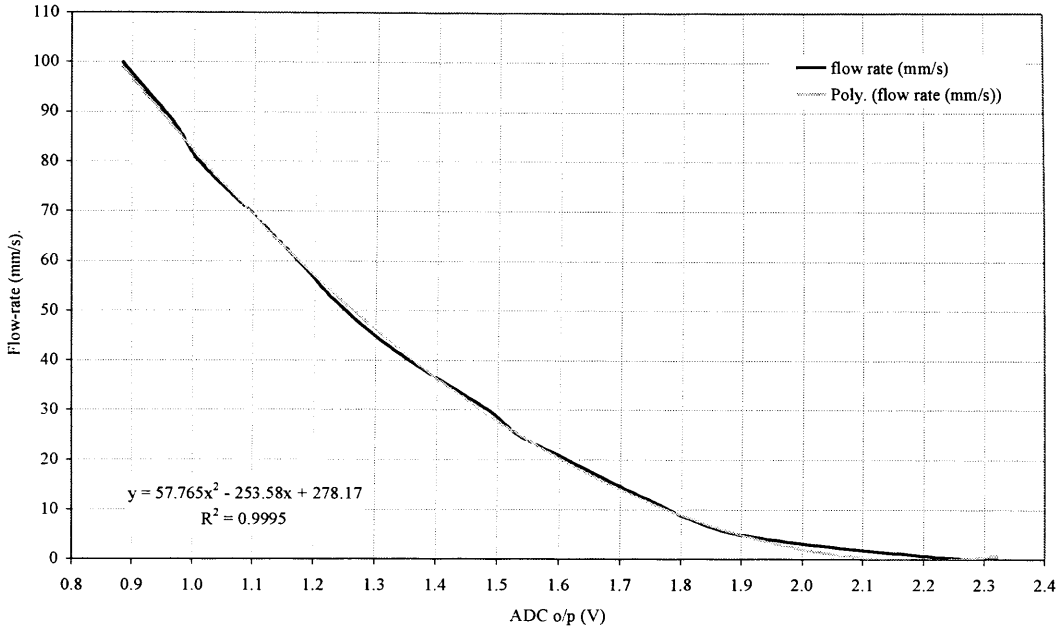


Figure 7.4 Flow Sensor Calibration Curve.

This method can work in any fluid irrespective of its electrical conductivity, density, viscosity and contamination, and within a wide range of temperatures and pressures. Accuracy of operation depends on the sensor converting a specific flow-rate into a consistently equivalent voltage. PT100 DIN IEC-751 thin-film RTDs were specified for this sensor because they have closely matched temperature response curves. This is important because the response curves of the PT100 devices must be kept close to parallel to achieve low temperature drift to give an accurate flow-rate measurement. Also the thermal path coupling both devices to the external flow environment should be identical or a mismatch will be introduced; again causing the response curves to deviate from each other.

One shortcoming of this particular implementation was that the heater is energised with a constant voltage source and the total power dissipation in the heater element changes with ambient temperature. This effect has been minimised by using a precision resistor with a low temperature coefficient for the heater element. Use of a constant power source, where the calorific power of the sensor is kept constant, can further reduce this effect. A precision 0.5Ω resistor (R4) was included on the circuit for the purpose of

monitoring heater current to allow power dissipation to be calculated and corrections made. There was not the time to develop firmware to perform the heater flow correction, however a system calibration was performed and the results plotted on a graph to obtain a characteristic curve shown above in *figure 7.4*.

7.1.2.3 Accuracy and Precision Issues

For a 1°C change in temperature, a PT100 RTD sensor responds with a 0.385Ω change in resistance. The implication of this is that even a small error in measurement of the resistance, such as the resistance of the wires leading to the sensor, will cause a large error in the measurement accuracy. For this reason, a modified version of the precision four-wire implementation was implemented (as described in ***section 6.6***) with dedicated wiring for the excitation current, and separate wiring to measure the voltage across the sensor element. The output voltage was sampled using a 24-bit delta-sigma ADC with built-in differential programmable gain amplifier with high input impedance buffer to minimise loading on PT100 sensor *[**section 6.5**]*. It was found that performing measurements with the buffer disabled resulted in significant errors because of the significant loading effect of the unbuffered amplifier input stage (Burr-Brown Products, ADS1242/1243 24-bit Analogue-To-Digital Converter SBAS235, 2001).

Another potential source of error that should be given consideration, are external and internal electromagnetic noise sources that add an unknown voltage in series with the PT100 voltage resulting in overall degradation of DAQS accuracy and resolution. This effect was kept to a minimum by selecting a high-specification ADC and using dedicated return (or reference) and power supply planes on the circuit board. The overall performance of the DAQS that proved to be accurate with high resolution, where the best test equipment available in the workshop could only determine that the resolution was better than 0.1°C with an accuracy better than 0.5°C - well below the expected performance of the DAQS.

It should also be noted that the excitation current (I_{ref}) through the sensor causes some self-heating: for example, a current of 1mA through a 100Ω resistor will generate 100μW of heat. If the sensor element is unable to dissipate this heat, its resistance will increase which will add a positive error voltage resulting in it registering an artificially high temperature. To minimise this effect the heater current is kept as low as possible at

I_{ref} = 200μA. This approach results in a trade-off of resolution against accuracy. Reducing the excitation current improves the accuracy, however the sensitivity is lower and therefore poorer resolution. Conversely, increasing the current reduces accuracy (because of the self-heating effect), however there is increased sensitivity resulting improved signal noise ratio and therefore better resolution. Fortunately, the sensor is embedded in an aqueous environment in good thermal contact with its environment as water has a high specific heat capacity. The sensor is therefore well coupled to an effective heatsink, removing a substantial part of this residual heat. Again, this effect was so small it was not possible to measure it with the test equipment available.

A 1.0V reference voltage (V_{ref}) was chosen to make best possible use of the available resolution ADC. The voltage reference defines the allowable voltage range of across the ADC inputs and also sets the size of the LSB. A smaller voltage for V_{ref} means that all the output codes are constrained to a smaller voltage range and that the LSB will be a smaller voltage. With a lower reference voltage, more of the output codes are used with the smaller range of input voltages. This is the same as providing gain in the ADC. For example, a nominal voltage reference is 5V; however with a V_{ref} of 2.5V the effect is a gain of 2 times, and a V_{ref} of 1.25V would be equivalent to a gain of 4 times.

The reference voltage for the ADC is generated by sourcing 200μA through a 5KΩ (R1) precision resistor (R_{ref}). This was the most precise and stable resistor that could be obtained, however, it should be noted that the quoted resistance tolerance is ±0.01% and drift is 10ppm/°C. Although no suitable equipment was available to test the effect these errors can introduce the DAQS accuracy, they were calculated to give an indication of expected performance. For example, the initial resistance of R_{ref} can be calculated as follows:

$$R_{ref} = 5000 \pm 0.01\% = 5000\Omega \pm 0.5\Omega \quad (7.1)$$

This gives a reference voltage error of,

$$V_{error} = I_{ref} \cdot R_{ref} = 0.0002 \cdot 5000 \pm 0.5 = 1.0000 \pm 0.0001 \; Volts \quad (7.2)$$

The reference voltage drift is proportional to ambient temperature. For example, for a 20°C temperature change, the maximum deviation in the reference resistance would be:

$$R_{ref} = 5000 \pm (5000 \cdot 20 \cdot 10 / 1000000) = 5000\Omega \pm 1\Omega \quad (7.3)$$

This gives a reference voltage error of,

$$V_{error} = I_{ref} \cdot R_{ref} = 0.0002 \cdot 5000 \pm 1 = 1.000 \pm 0.0002 \, Volts \quad (7.4)$$

The total voltage error is given by the addition of the errors from Eq. (7.2) and Eq. (7.4)

$$V_{totalerror} = 1.000 \pm (0.0001 + 0.0002) \, Volts = 1.000 \pm 0.0003 \, Volts \quad (7.5)$$

This error is too small to have any significant effect on DAQS precision, however will limit the ultimate measurement accuracy to ±0.03%.

The ADC itself is also subject to offset error (7.5 ppm of full-scale = 0.00075%), offset drift (0.02% of full-scale/°C) and gain error drift (0.5 ppm/°C). These errors can be calculated in a similar manner as described above. Finally, the current source has an initial accuracy of 200μA ±0.5% and drift of 25 ppm/°C, however the ratiometric sensor architecture nullifies this potential source of error.

During sea-trials the instrument, was deployed with the ADC input buffer enabled, an amplification gain of 16 (PGA = 16) times and 1.0V reference voltage. This information was used in conjunction with the typical characteristics for the ADC, given in the data sheet to calculate the expected dynamic range and resolution of the complete DAQS. For this particular configuration the effective resolution or effective number of bits (ENOB) for the ADC is 17. The PGA setting and V_{ref} introduce two gain factors that allow the ADC to achieve 23-bit resolution because PGA = 16 is equivalent to 4 bit-shifts and reducing V_{ref} is equivalent to just over 2 bit-shifts. This gives a total of 6 extra bits (17 + 6 = 23-bits). The system voltage resolution is calculated as:

$$Voltage\ Resolution = (V_{ref} / PGA) / 2^{ENOB} = (1/16) / 2^{17} = 477nV \quad (7.6)$$

Now, a 0.385Ω change (ΔR) in PT100 resistance corresponds to a 1°C temperature change, which translates to a ADC voltage input change of:

$$\Delta V = (I_{ref} \cdot \Delta R) \cdot PGA = (200\mu A \cdot 0.385) \cdot 16 = 1.232mV \quad (7.7)$$

, when using 200μA excitation current. So it follows that (for this particular ADC configuration with PGA = 16 and buffer enabled) the DAQS will be capable of resolving a temperature change of:

$$T_{resolution} = 477nV \ / \ 1.232mV = 0.0004°C \quad (7.8)$$

It is also important to establish the dynamic range of the instrument. The maximum allowable analogue input voltage for ADC is:

$$V_{ref} / PGA \ = 1 / 16 = 62.5mV \quad (7.9)$$

, which translates to an RTD resistance of:

$$R_{RTD} = V / I_{ref} = 62.5mV / 200\mu A = 312.5\Omega \quad (7.10)$$

Again, using the linear relationship for conversion of resistance to temperature, this sets an upper limit on the measurable temperature of:

$$T_{max} = (312.5 - 100) / 0.385 = 552°C \quad (7.11)$$

, at this PGA setting. Altering the PGA setting will change this upper limit, for example a PGA setting of 32 would reduce the upper limit by a factor of two (276°C). The excitation current is common to all PT100 devices in the sensor ladder network *[as described in **Section 6.5**]* and the current source (REF200 Dual Current Source/Current Sink PDS-851D Burr-Brown Corporation, 1988) voltage is derived from the battery terminals (V_{bat} = 4.5V). This gives enough voltage headroom for the current source compliance and allows all PT100 devices in the ladder to attain the 1.0V required to represent full-scale temperature (552°C).

Examining the lower limit of the scale, it should be noted that the minimum allowable analogue input voltage level for the ADC is +0.05V (50mV) with the input buffer enabled. To maintain the voltage input above this, a 1KΩ resistor was inserted at the bottom of the ladder to offset the voltage by:

$$V_{offset} = I_{ref} \cdot R_{offset} = 200\mu A \cdot 1K\Omega = 200mV \quad (7.12)$$

This puts the signal well within the analogue input voltage range of the ADC, allowing the DAQS to measure temperature down to the limits of the PT100 (below –200°C).

7.1.2.4 Sea-Trial

In 2005, one SBC was embedded within a pressure case to make a data logger instrument that was then integrated into the ROV "Bathysaurus" on board the Norwegian research ship G.O. Sars. Communication was established via an RS-232 interface, which allowed the operator to control of parameters, such as system sample-rate and ADC gain and telemeter real-time data to the surface. Incoming data was displayed, plotted and logged using a DASY*Lab*® worksheet running on a PC in the ROV control room. The instrument was initially configured to perform high-resolution fluid flow-rate and temperature measurements at Mohns Ridge vent system (near Jan Mayen) in the Arctic Ocean during several "BioDeep" dives. Deployments were made on diffuse flow areas where the instrument successfully telemetered data from 540 meters below sea level back to the ship above. Fluid outflow was detected from chimney structures at +0.5°C, against ambient background temperatures of -0.3°C. Flow anomalies were also identified on top of extensive bacterial mats or mounds (Schultz 2005).

Further dives were made which led to the discovery of a hydrothermal vent field ("Soria Moria") where attempts were then made to measure the temperature and flow rate of effluent directly above a "white smoker". The sensor was deployed directly above the vent chimney and indicated temperatures of 260°C. This was repeated at another site, indicating that the fluids were venting at the point of phase separation and "smoker" plume flow-rates of approximately 0.5m/s were measured (Schultz 2005). These temperatures were outside operational limits of the epoxy resins used for the sensor manufacture and consequently one of them suffered fracture damage *[figure 7.5]*, possibly due to mismatch expansion coefficients of materials used in fabrication.

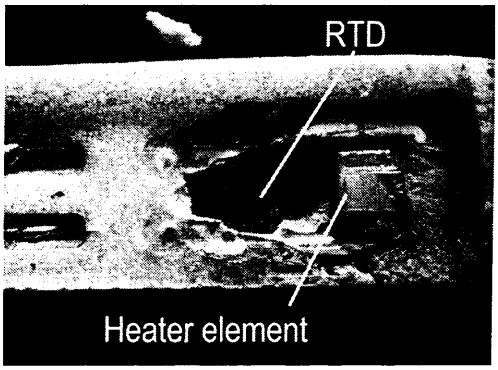


Figure 7.5 Thermal stress damage to sensor deployed at Mohns Ridge.

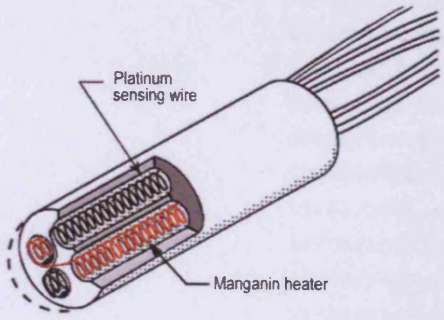


Figure 7.6 Improved sensor element custom manufactured by RdF Corporation, USA.

These valuable results led to further development and an improved sensor design shown in *figure 7.6*. The coiled sensor element was manufactured by inserting a helical platinum coil into a powder-filled, insulating mandrel. This construction minimises the mismatch of coefficients of thermal expansion between different materials producing a strain-free sensing element. The result is a tougher sensor element with a higher temperature coefficient, improved stability and accuracy. This cylindrical geometry also improves calibration accuracy as the sensor is less orientation dependent.

7.1.3 Data Storage Consumer Node

The code required to support the file system (data storage on FAT16 formatted SD/MMC media cards) was by far the most complex firmware driver developed for this project. The task of merely establishing SPI communication with the media cards was more complex than for other SPI devices, as their native mode of operation is a propriety "Multimediacard mode" protocol - the card must first be initialised to work in SPI mode. There was a further overhead of "housekeeping" to maintain a data buffer data in SRAM, as the media card read/write operate on data blocks of 512 byte length. Additionally, the support code required to maintain and update the FAT, opening and closing files, writing to files at the byte and string level (no functions were developed for reading data from files) and converting file information into human readable time and date format introduced another considerable overhead.

The file format was designed so that it could be imported seamlessly into MS Excel spreadsheets or read as a data stream by DASY*Lab*® virtual instrumentation data-flow language (Note: the file header is required to maintain full compatibility with DASY*Lab*). Data within the file was stored as text as shown in *listing 7.1*. Although this is not the most efficient or compact method of storing data, it is human readable in the event of file corruption and can be read using a basic text editor, such as "Notepad".

```
Recording Date : 12/08/2006;19:04:20
Block Length :
Delta : 5
Number of Channels : 3
Date;Time;Chan 0;Chan 1;Chan 2;
12/08/2006;18:05:10;37.796;56.523;29.367;
12/08/2006;18:05:15;37.835;56.773;28.906;
12/08/2006;18:05:20;37.640;56.671;28.921;
12/08/2006;18:05:25;37.695;56.328;29.125;
12/08/2006;18:05:30;37.656;56.585;29.148;
```

Listing 7.1 Data file format

The single board computer file system was given a thorough "shakedown" as part of the final integration and validation phase of the EXOCET/D project during the first leg of the MoMARETO cruise on board the French oceanographic research vessel "Pourquoi Pas?" in August 2006. The main objective of the science cruise was to study the spatial and temporal dynamics of hydrothermal vent communities colonising the MoMAR zone, located on the Azores Triple Junction. A part of this objective relied on the implementation of dedicated instrumentation for in-situ temperature and flow-rate measurement of "smoker" effluent.

Figure 7.7 The author with chief scientist, Pierre-Marie Sarradin installing the node and sensor on ROV Victor.

It was assumed that it would be possible to integrate the node into the communication framework of ROV Victor via the RS-232 interface, which would have allowed real-time control and telemetry as on the previous Norwegian cruise. Once on board "Porquoi Mois!" a custom interface cable was spliced together to allow the RS-232 interfaces on the node to be connected to those on ROV Victor. All attempts at establishing communication failed because of an unforeseen incompatibility issue between the two different RS-232 interface implementations. A quick decision was taken to deploy the instrument in autonomous logging mode so that it would acquire the measurement data and store it on internal memory storage media card without the requirement for interaction with a control operator. This scenario highlights the advantages of adopting a commonly accepted standard interface and protocol to

facilitate communication between scientific instruments developed by different manufacturers and research institutions.

Figure 7.8 Sensor-head deployed on a mussel assemblage on Menez Gwen (MAR).

Several ROV dives were made to the ocean floor to various sites along the Mid-Atlantic Ridge (MAR) close to the Azores, where the instrument was operated in autonomous mode. Within the duration of each dive, there were several opportunities to deploy the sensor head using the ROV robot arm. *Figure 7.8* illustrates the measurement of diffuse effluent flow through a mussel assemblage on Menez Gwen. Upon the completion of each dive, the instrument was recovered from the ROV and taken to the laboratory space aboard the vessel and the pressure case was opened so that the 32MB media card could be removed. The card was then plugged into a portable computer so that the data file could be imported into an MS Excel spreadsheet for post-processing and presentation. The instrument flawlessly recorded temperature and flow-rate data files during all dives and a sample plots is shown below.

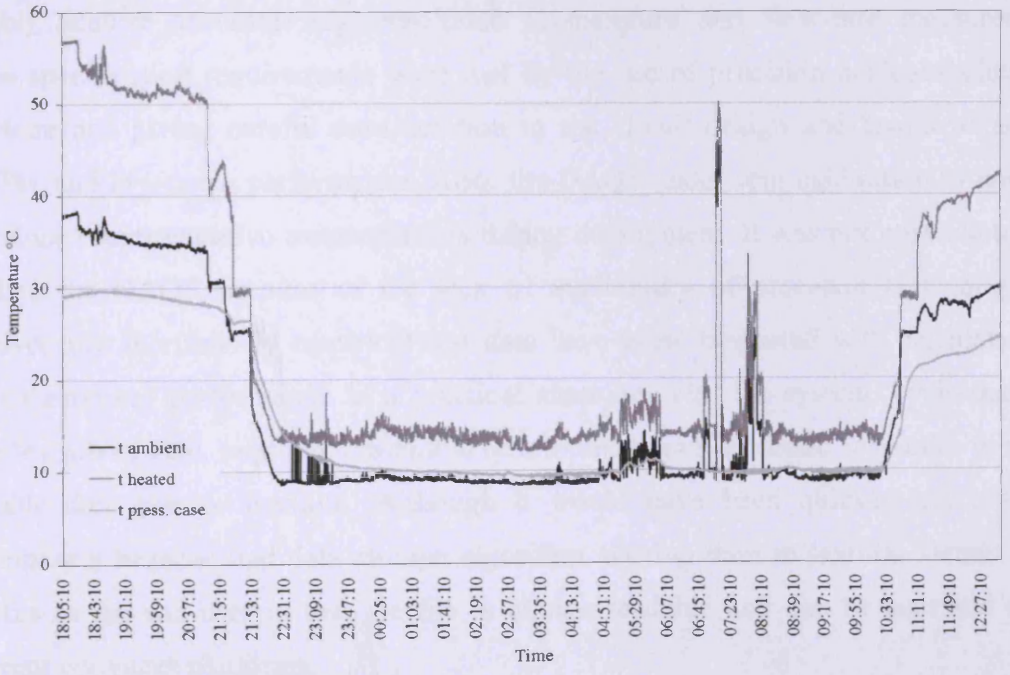


Figure 7.9 Menez Gwen Victor Dive 287 12th August 2006

For this particular session, the sample distance is 5 seconds and the resulting data set is composed of almost 40,000 datum measurements. The time history represents temperature profiles for a complete dive for ROV Victor being deployed at 10-00pm in the evening and then retrieved at the following morning at around 10-30am. The "t ambient" plot displays a continuous time history of the ambient sea temperature, with individual sensor deployment events centered at 5-30am, 7-00am and 7-45am. The "t heater" plot displays the temperature of the indirectly heated temperature sensor used for calorimetric flow-rate measurement. This tracks approximately parallel to "t ambient" with an offset component of around 5°C in a zero flow-rate water regime, however in air the offset increases to around 30°C. This change can be seen at the beginning and end of the time history while the ROV is out of the water. Finally, "t pressure" shows the temperature within the pressure case. The pressure case is decoupled from the external aqueous environment and is an effective low-pass filter. Consequently the trace is smoother with reduced temperature fluctuation than measurements taken outside the pressure case.

7.1.4 Performance and Reliability

Considerable effort was focused on testing and validating node sub-systems, especially the DAQS and file system. These sub-systems are core sub-systems required to build an instrument with useful data logging capability. It was important that the instrument could capably acquire accurate, high-resolution temperature and flow-rate measurements. These specification requirements were met by the use of precision analogue electronic hardware and giving careful consideration to the circuit design and layout to achieve stability and low-noise performance. Also, the DAQS underwent calibration to ensure it reproduced representative measurements during deployment. It was not possible to fully validate the DAQS because of the lack of availability of precision test equipment, however any shortfalls in empirical test data have been supported with calculations to predict expected performance in a practical situation. The file system, by its nature a complex sub-system, required substantial development and processor resources to realise a viable data storage medium. Although it would have been quicker and easier to implement a hexadecimal data storage algorithm, storing data in textural format offers benefits to the end-user in that the file is human readable and can be accessed across different computer platforms.

All the single board computer (SBC) systems preformed within specification with no unpredictable behaviour or firmware "lock-up" scenarios being encountered. As a last line of defence against such events a hardware watchdog timer was also incorporated into the system design at an early stage in the development. This forces a hard reset of the system in the event of unpredictable firmware "lock-up" and ensure that control can be reestablished. This approach is itself not a complete guarantee against system failure, however provides a substantial improvement over not implementing a watchdog at all.

7.2 Node Communication on the Network

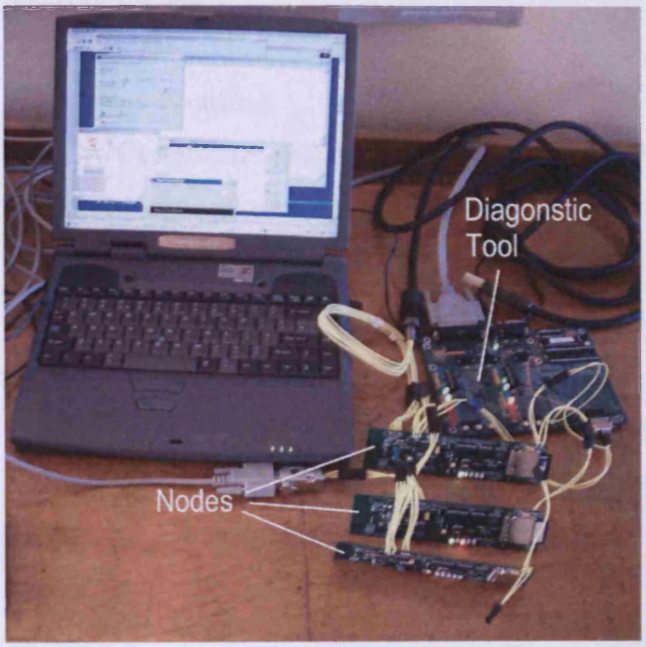


Figure 7.10 Four-node system under test.

The preceding tests validated the correct functioning of node electronic hardware (single board computer) and firmware (kernel and drivers). The next stage involved connecting several nodes together to establish communication and test how they interact with one another. Extensive use was made of Microchip's CAN controller development kit as a tool to aid in establishing inter-node communication. The software allowed direct manipulation of the MCP2510 at the bit and byte levels with a Register Template, while providing high-level control with a second Basic Template. One node was controlled by the PC, which operated as a microprocessor using the provided software, and a second node was controlled by an embedded microprocessor pre-programmed to monitor bus traffic. The two nodes were connected via a CAN bus that was also routed off-board through a 9-way serial connector, allowing external nodes to be added to the network. The development board was modified *[figure 7.11]* so that this connector also supplied power for external nodes, enabling them to be "daisy-chained" together.

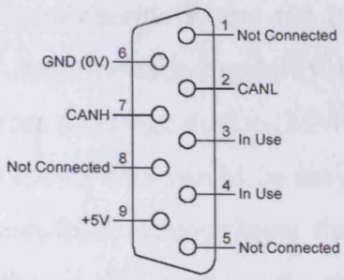


Figure 7.11 Pin-out for CAN Serial Connector.

The Basic Template was utilised as a high-level diagnostic tool for testing network communication, these tests included monitoring bus loading, logging and displaying received messages and also transmitting messages.

7.2.1 Bus Loading

The CAN controller on the development board is configured to receive all messages by default which makes it possible to continuously monitor bus loading (or traffic). The "Bus Status" window was used to display nominal load on the CAN bus as a percentage and the total number of messages received and transmitted. *Figure 7.12* shows a typical experimental run where a single sensor producer node was connected to the network and configured to transmit three channels of temperature data every second.

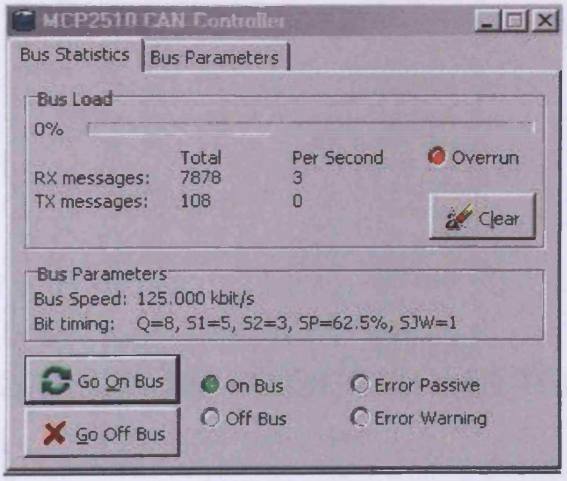


Figure 7.12 Bus Status Window

It can be seen that three messages are received every second and that the bus loading is so low that it registers zero. As discussed in ***Chapter 3***, network traffic for this application is low even at moderately high sampling rates (a typically sampling rate would be 15 seconds). This accurately reflects the volume of network traffic in a real system, such as the proposed NASA and NERC sampling instruments. However, there is

the possibility of normal operating scenarios where the bus traffic would rise to high levels for brief periods if mass data transfer capability is implemented, for example download of measurement data from a storage device (MMC/SD). In this case, the COB-ID for messages containing this type of data would be assigned a relatively low priority to prevent the node producing them from monopolising the bus. This priority based bus arbitration mechanism is one of the significant strengths that the CAN protocol exhibits over other bus technologies (such as Ethernet, for example) especially in the context of mission critical applications.

7.2.2 Message Integrity

The "Output" window gives a more detailed view of the contents of received (and transmitted) CAN messages. To aid readability of the CAN message contents, this window was configured to display data in hexadecimal format rather than decimal. This feature was utlised to check that communication medium was functioning reliably, i.e. message contents were not corrupted or that any messages were "lost". *Figure 7.13* shows messages being received from the temperature sensor producer node.

Output Window

Ident	Flg	Len	D0	1	2	3	4	5	6	D7	Time	Dir
0186		4	01	57	DE	6F					2690.272	R
0186		4	02	57	D6	2E					2690.272	R
0186		4	03	57	FE	4C					2690.276	R
0186		4	01	57	DB	0D					2691.277	R
0186		4	02	57	D5	3C					2691.277	R
0186		4	03	57	FF	6B					2691.281	R
0186		4	01	57	DD	92					2692.281	R
0186		4	03	57	FD	9F					2692.298	R
0186		4	01	57	DF	53					2693.292	R
0186		4	02	57	D3	F4					2693.292	R
0186		4	03	57	FE	0B					2693.295	R
0186		4	01	57	DC	D1					2694.296	R
0186		4	02	57	D6	28					2694.296	R
0186		4	03	57	FF	83					2694.300	R
0186		4	01	57	DB	BE					2695.301	R
0186		4	02	57	D3	65					2695.301	R
0186		4	03	57	FA	5E					2695.305	R

Figure 7.13 Output Window

Nodes were left on "soak-test" for a thirty-minute period in an effort to detect missing/corrupted messages, however no problems were observed.

Further tests were undertaken by introducing two additional nodes into the network as shown in *figure 7.10* above. The first of these nodes was compiled and programmed as a producer node to asynchronously transmit a messages with COB-ID 0x287 (647 decimal) via user control onto the bus. The second was programmed as a consumer

node to process only messages that match the COB-ID (0x287) from the first node and ignore all other bus traffic. The consumer node acknowledged reception of this message by flashing a red LED. The CAN controller development board was also utilised as producer node in this test to transmit the same COB-ID and message contents in an attempt to force a data collision on the bus. The figure below shows the "History List" window, which was used to transmit a predefined message continuously at defined interval or in "one-shot" mode when the send button is pressed.

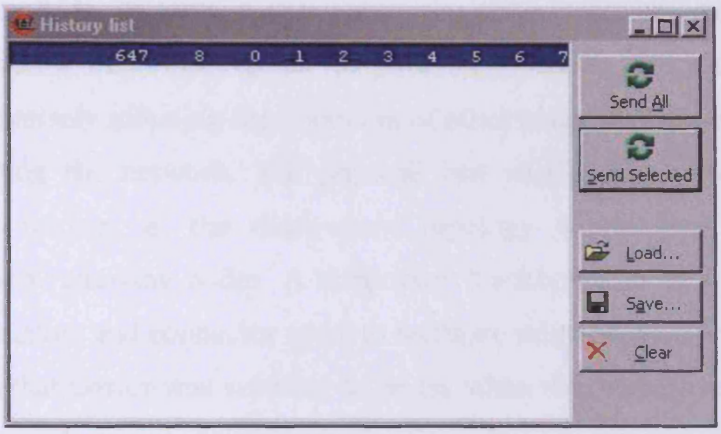


Figure 7.14 History List Window.

This window also made it possible to collect transmitted messages by saving them to a file to facilitate analysis of bus traffic. Again, CAN proved to be a reliable and transparent transmission medium with no observed data corruption.

7.2.3 "Plug-and-Play" Capability

With the CAN data link layer in place it was possible to progress and implement higher level functionality, where nodes can be added and removed dynamically to the network and exchange data with one another. The RTOS and MicroCANopen stack provided the framework for defining this behaviour. The freely downloadable version of MicroCANopen utlised imposed several restrictions in that it does not support heartbeat consumer and only a 254-byte Object Dictionary. Despite this there were adequate resources to construct and test a minimal system consisting of a temperature sensor and data-logging nodes.

A temperature sensor node was configured to produce TPDO (a CAN message containing temperature process data) once every second. A data-logger node was then directly linked to consume these messages by mapping its RPDO to the same COB_ID.

A small amount of additional code was required within the data-logger node so that it could unpack the data payload from the incoming CAN frame and transfer it to the 32MB media card *[see section 7.1.3]*. This direct linking between the nodes was monitored by the CAN development board that operated as a third node monitoring bus traffic. Additionally, a PC was connected to the RS-232 interface on the data-logger node to check that the node was receiving and interpreting the CAN frames correctly.

Further tests were made by disconnecting/reconnecting the temperature sensor node from the network to confirm that nodes could be dynamically removed/added from/to the network without adversely affecting the operation of other nodes. For example corrupting messages or stopping the network. The physical bus wiring had to be modified to accommodate this testing, as the daisy-chain topology would have been broken communication when removing nodes. A temporary "backbone" loom was constructed with termination resistors and connector spurs to facilitate this. A separate power bus was also used to ensure that power was supplied to nodes when they were removed from the communication network to prevent them resetting. The monitoring tools indicated reliable communication between nodes on the network. Upon disconnecting the sensor node the data-logger stopped storing data and on reconnecting started again demonstrating a level of "plug and-play" capability as expected.

7.3 Summary

The first section of this chapter described the tests undertaken in the workshop and during sea-trials to evaluate the electronic hardware (single board computer) and firmware drivers (hardware abstraction layer) that constitute the node sub-systems. These drivers were incrementally integrated into the RTOS kernel so that their operation and interaction could be clearly evaluated. The resulting computational platform proved to be stable, exhibited low power consumption *[see **Appendix E** SBC Power Budget]* and operated as intended during deployment at sea where it produced useful scientific data (Schultz 2005).

This platform was utilised to test and interface a demonstration of MicroCANopen higher level protocol (HLP) to the existing CAN controller driver and further integrate it into the RTOS framework. Several single board computers were taken and programmed

to function as producer and/or consumer nodes to evaluate the feasibility of the grid-enabled instrument architecture. It was established that the single board computer platform had the necessary resources to adequately support the communication system and required HLP features on which to base a viable decentralised instrument system. Several tests were performed to assess and validate reliable communication between nodes, where the results indicated that the CAN data link layer handled communication effectively and transparently with no corruption or loss of data. Additionally, data was inserted/removed from CAN frames allowing information to reliably propagate between nodes to realise "plug-and play" capability.

Chapter 8 Conclusion

This thesis has presented the case for adopting a decentralised network topology as an appropriate framework on which to base a grid-enabled machine system for use in inaccessible or "hostile" environments. To achieve this objective, commercial off-the-shelf technologies and standards were leveraged wherever possible. The physical and data link layers of the communication system were implemented in CAN hardware with high level control being established under the MicroCANopen protocol stack and Salvo real-time operating system kernel. This strategy helped to expedite the development process and ensure the work does not become obsolete in the near future, as widely accepted, documented standards are adhered to. It also opens up the opportunity for end-users to directly connect the instrument to modules developed by other manufacturers (who adopt the same standards) to form aggregate systems without the prerequisite for any specialised knowledge regarding the interface.

8.1 Current State

Much of this research effort has been focused on developing a standardised framework to support the development of robust instrumentation systems. To this end, a body of documentation was produced, including a library of firmware routines and methodologies for integrating them into an RTOS kernel, running on a custom-built single board computer. The construction of the single board computer itself was based on modular hardware architecture to enable it to meet the micro-power and high reliably specification requirements of this project. This electronic hardware and firmware infrastructure developed herein was then utilised to build a decentralised microbial sampler instrument (the machine system) composed of a group of nodes (individual machines) that coordinate their activities to work together concurrently to accomplish a given task. The power supply and processor resources were distributed homogeneously throughout a network of nodes to yield an inherently robust and reliable system. Additionally, the use of open interface and communication standards create an extensible architecture where it will be possible to fully support "plug-and-play" functionality, so that nodes can be dynamically added or removed to/from the network to meet mission-specific requirements or for repair of faulty nodes.

The electronic hardware and low-level firmware drivers that constitute the node subsystems have gone through several test and development iterations. These subsystems were observed to function and interact in a stable and predictable manner when integrated within the RTOS kernel running on the single board computer platform. The adoption of standardised interfaces and modularisation of components at the node and sub-node level have made it feasible to assemble decentralised machine systems that can do useful work. These findings are supported by the application of this technological development to the construction of a new microbial sampler instrument. Subsystems from the instrument were tested at in several different situations to evaluate their performance at-sea. The DAQS and file system were successfully validated on hydrothermal vent fields, "Soria Moria" on Mohns ridge (Schultz 2005) in the Arctic Ocean ("BioDeep" cruise, 2005) and "Lucky Strike" and "Menez Gwen" on the Mid-Atlantic ridge (MoMARETO EXOCET/D cruise 2006). The motor positioning system test failed at sea because of improperly installed seals, however testing in a pressure chamber established the valve positioning and pump control algorithms worked according to specification. Further funding is expected to become available in 2008 from ChEsSo (Biogeography of Deep-Water Chemosynthetic Ecosystems in the Southern Ocean), a newly funded UK Consortium Bid for the exploration and investigation of chemosynthetic habitats in the Antarctic region with cruises planned for 2009/2010 in Bransfield Strait. It is anticipated that this opportunity will allow a full system test to take place with the aim of obtaining viable microbial samples.

Although the sea-trials have confirmed that the instrument subsystems function within design specification, in its current state, it still requires specialist-engineering support during preparation, deployment, recovery and post-cruise refurbishment. This to be expected with any sophisticated new instrumentation system, however it does limit the utility of the system and the contribution it can afford to the scientific community as it is difficult and costly to deploy. The next challenge is to find a solution that ensures that resulting instrument systems are easier and more accessible to use. A step towards realising this goal is to develop the technology to a state where it would be possible for a system integrator to bring together the component subsystems into one system, without having to resort to direct low-level code manipulation. Taking this strategy further to its ultimate conclusion, the ideal goal is to create instrumentation systems that can be modified by end-users (scientists, researchers and marine technicians) to fulfill

mission-specific requirements. The next section puts forward a development "roadmap" in an effort to resolve this ease of use issue.

8.2 Future Work

8.2.1 Short-term

In the immediate short-term there are several additions that can be made to the node electronic hardware and firmware to improve the overall system reliability performance and ease of use. These additional features and the benefits they will offer are outlined below.

8.2.1.1 Communication System (CS)

The CAN hardware performed within specification during bench testing, providing a reliable and transparent communication medium for data transfer between nodes. However, one concern was raised during the 2006 EXOCET/D cruise, where an engineer from the French research institute, Ifremer highlighted the issue of equipment isolation. Ifremer stipulate that all marine electrical equipment should be isolated to prevent electrolytic corrosion, which can result from external current sources, such as a ship battery or power supply. The current that causes the electrolytic action is referred to as a "stray current" and usually emanates from a poorly installed electrical circuit or an inadequate earth (ground) arrangement. This could be the result of a poorly installed electrical circuit or a current leak due to damp connections.

These stray currents can be eliminated by galvanically isolating the CAN hardware interface on all nodes to prevent ground loops to stop the flow of current. This is achieved by placing two opto-coupler devices between the CAN controller logic levels and the CAN transceiver. The transceiver then requires an isolated power supply, either supplied through a small DC/DC converter on each board or by running a common power supply through the cable that all the boards share. Similar work had been carried-out for Ifremer several years ago to build a stand-alone data-logger with a galvanically isolated RS-232 interface. The principle for galvanically isolated CAN is the same and the hardware implementation is almost equivalent for both interfaces.

During the bus network testing and debugging phase, nodes were configured to receive all messages by default and message filtering was performed in firmware to establish that the COB-IB could be reliably packed and unpacked into the CAN frame. The data link layer in the CAN controller hardware should ultimately be utilised for this purpose, as it frees the application layer to enable the whole process to be handled more cleanly and efficiently. In fact, the single board computer hardware supports interrupts from the CAN controller to the main processor, which makes it feasible to shut the main processor down to minimise power consumption and rely on the interrupt to "wake" the node up upon the reception of a valid COB-ID.

Moving up from the data link layer to the MicoCANopen higher layers, further work should be undertaken to formalise the method of communication between the MicroCANopen protocol stack and the Salvo RTOS tasks *[see **Section 5.4**]*. This first revision of the firmware was put together to enable nodes to trigger one another when an event occurs. Presently, the communication mechanism relies on RTOS event flags and MicroCANopen change of state (COS) triggering to transfer bit-level control information from an RTOS task running on one node to another task within another node on the network. This seems to be a valid approach, however the event flag and Object Dictionary definitions certainly need formalising as presently, duplicate information is held in more than one place in memory. It is therefore desirable to modify some of the variable definitions and perhaps represent the data in structures, to optimise the code and make it more readable. On the other hand, there may be a more logical or efficient way of implementing this communication mechanism and perhaps further research could be undertaken to investigate this.

The system would benefit from some additional code to support the existing PDO definitions, specifically the "Hardware Error" register entry, which should be utilised to propagate status information between communication partners *[as described in section 4.2.3.4 Hardware Error Entry [0x2004]]*. Adding support code for this mapping would endow the machine system with capability to autonomously handle hardware failures in a graceful manner. Finally, one or two new PDO definitions are also needed to support some additional functionality in the machine system. For example, the system would be easier to use if the system clock embedded within every node on the network could be set simultaneously from the host computer. The broadcast capability of the CAN

protocol supports this kind of behaviour as all nodes on the CAN bus can receive all messages.

8.2.1.2 Data Acquisition System (DAQS)

Much effort was focused on producing an accurate, high-precision DAQS with wide dynamic range to meet the demands of temperature and flow-rate measurement in the marine environment. On one hand, substantial dynamic range was required to encompass temperatures from just below 0°C (e.g., on the Arctic ocean floor) to above 400°C (e.g., within a "smoker" plume) and, on the other, high-resolution, to enable milli-degree precision temperature measurement (e.g., profiling within the water column). Experimental results and assertions based on calculation confirmed that the DAQS hardware and firmware meets these specification requirements, however further work should be undertaken to quantify the performance of the sensor element and housing assembly (sensor head). The sensor head was custom manufactured and presently there is no data on its bulk thermal response time, which is required to establish how long it takes to reach equilibrium with the environment. This data is required to ensure that measurements can be performed properly and that accurate measurements can be obtained.

It was assumed that sensor element and its housing (sensor head) exhibited a first order Bessel filter response, where the transition from the pass band to the stop band is wide before the final –6dB/octave roll-off is achieved. This could be accomplished by a simple test to determine the impulse response of the sensor head. To establish this, the firmware would be modified to achieve a sampling rate about 1/10 second and data logged on to memory card. The sensor head would be taken from room temperature and plunged into a beaker of hot water. The data obtained from this experiment could then be imported into DASY*Lab* where a Fourier analysis can be applied to calculate the impulse and frequency response of the sensor head.

8.2.1.3 Motor Positioning System (MPS)

Work on the MPS firmware was completed sometime before the author acquired any knowledge or experience in performing embedded firmware compilations based on Salvo RTOS kernel. The present code implementation is therefore built on the more

"traditional" main "superloop" with interrupt service routine running in the background. Although, experimental tests revealed that this approach worked adequately in practice as a valve and pump controller, experience indicated that significant improvements could be obtained, in terms of reliability and ease of code maintenance, by introducing RTOS paradigm onto target processor (PIC16F819). The application layer requirements are not excessively demanding for motor control in terms of processor resources (the PIC16F819 ROM is limited to 3.5KBytes with only 256 Bytes of RAM) and a compilation of the RTOS kernel with drivers was successfully made to test the feasibility of this approach. Introducing RTOS services, such as time-outs and inter-task communication mechanisms would increase the level of security for variable data (defensive programming) and allow for finer granularity of control through multitasking. This code revision would not be a huge undertaking and require approximately 2-3 weeks to complete.

8.2.1.4 Files System (FS)

The FS is a complex subsystem that utilises a large proportion of processor memory and program space resources for its implementation. Despite this, significant work still remains to be undertaken to develop this minimal implementation so that it can satisfy requirements:

- Generalise the FAT driver so that it can work with all types of media cards. In it's current state, it has been used with 32MB SanDisk MMC. The next stage should be to develop the drivers to work with other SanDisk MMCs (8MB, 16MB and 64MB) and then move onto different manufacturers. Finally, generalise the driver to operate with SD cards. Some of this work has already been completed and the FAT can recognise both MMC and SD cards and read the directory structure.
- Rather than storing one single contiguous file to memory, it would more secure to break it up into a "chain" series of files. A natural extension of this would be to support multiple files being opened for reading and writing. This would allow more sophisticated instrument behaviour.
- Implement functions (fgetc and fgets) for reading data from files. This extra functionality would make it possible to store node configuration settings on the media card in the form of an initialisation file (isosamp.ini). These setting could include sampling rate, file names and other system settings.

- Capability to allow the system to format the media card. Presently cards must be formatted on an external computer.
- Improve the robustness of the data storage algorithm. In some ways the requirements of this minimal FAT implementation are more demanding than a typical full implementation found on a personal computer (PC). For example, if there is a power failure or "brown-out" on a PC then the file that is being written to disk will be corrupted and the data may be impossible to retrieve the data. When the machine boots-up again, there is no record that the event happened, let alone any mechanisms in place to prevent loss of data. Obviously there is no way predict a power failure before it happens, however measures can be put in place to minimise its impact should it occur. For example, breaking the file up into a chain series prevents loss of all the data and memory persistence allows the event to event to be recognised. Counters should be implemented to keep track of resets and the current file. Count should be held in non-volatile memory so that the FS can keep track of the currently open file and pick-up where it left off in the event of a power failure.

These extra requirements constitute such a large overhead, that the main computational core on the single board computer simply cannot support all of this additional functionality. The core (PIC18F252) features 32KBytes of program space (ROM) and 1536 Bytes of memory space (RAM) of which a total of 28918 Bytes (88.3%) ROM and 1121 Bytes (73.0%) RAM are being used for the current compilation. Any significant conceptual additions will almost certainly push requirements beyond the available resources. An upgrade path exists and during the span of this project an improved core (PIC18F4685) became available. This new device has the same pin-out and physical package, however with considerably more ROM (96 KBytes) and over twice the RAM (3328 Bytes). This is an 18-bit core, which is part of the same family and should present no significant problems when porting the existing C code to the new platform.

8.2.1.5 Power Management System (PMS)

Project time constraints did not allow for the opportunity to begin development of the PMS firmware. This system would manage changing of batteries from external power available on the bus, measure battery voltage and allow this information to be accessed by the application layer so that the appropriate action can be taken in the event of any

problems with the batteries. No RTOS test compilations have been undertaken involving RTOS and no drivers have been ported to for the target (PIC12F675). Although ROM and RAM resources are severely limited on this core, at 1Kbyte and 64 Bytes respectively, the Salvo user manual indicates that it is practical to implement the RTOS on this core and application notes support this with examples of applications being developed for other "low-performance" PIC cores. Also, there is no hardware SPI interface, so this would need to be implemented in by "bit-bashing" in firmware. Previous experience gained in assembly language bit-bash implementations for a similar application (RS-232 data receive and transmit on a PIC16F84) indicates that the SPI reception and transmission routines would require in the region of 50 to 80 instructions ROM space. Data reception is slightly more complex than transmission to implement. Working in such a limited programming environment has the advantage of constraining code complexity and can yield an elegant solution that is an optimally "good fit" for this type of application.

8.2.1.6 Human-Machine Interface (HMI)

Two different approaches were taken to designing an interface to facilitate user interaction with the instrument in the field, both utilising RS-232 interface for communication. A HMI was initially developed using DASY*Lab* data-flow language to construct a virtual front-panel for control of the instrument and data display. DASY*Lab* made it possible to develop a worksheet that would allow an operator to configure the instrument and manipulate/display incoming data in real-time as charts, digital meters and even perform calibration "on the fly". This approach was initially appealing, however despite its ease of use and professional appearance limitations were encountered trying to coerce DASY*Lab* to perform certain operations. For example, it was not possible to develop a worksheet to set/read the onboard system clock using DASY*Lab*s' visual programming language. The language ultimately proved to be too restrictive, as it was optimised for data manipulation and presentation, not for reading/writing data bytes to or from specific hardware registers.

As a consequence of this a terse command line interface based communication protocol was developed to allow better access to configuration registers and control of the instrument. In this case, a text editor ("TeraTerm") application running on the host PC was used to communicate with the instrument. Although not as visually impressive as

the DASY*Lab* graphical user interface (GUI), the command line protocol does not preclude the use of a custom GUI at a later time that shares the same communication protocol definition. The next stage is to map this onto CAN messages (PDOs) to allow access and control of any node on the network from the host computer.

8.2.1.7 Hardware Abstraction Layer (HAL)

After undergoing several revisions, it is considered that the device drivers that makeup the HAL are in a state where no further development of the firmware is required. A reference manual was written to support these driver functions, containing a systematic and documented library of the routines ***[Appendices M and N]***. The final step would be to compile the source code to object files and then combine them into a single library file. This requires the use of a librarian program to create the library file (file.lib) and the process is explained in the PICC ANSI C Compiler User Guide (Hi-Tech Software 2002). This library would add a significant level of security to the HAL by preventing accidental modification of the driver functions and also speed up the compilation process.

8.2.2 Long-term

With completion of the design improvements outlined above, the firmware framework can now be considered as an application programmers interface (API) that a software programmer could utilise to develop a fully operational machine system. However, further work is required to develop off-the-shelf technology that supports "plug-and-play" capability where nodes, such as input/output modules, sensors and actuators from different manufacturers are interchangeable with one another, to allow a systems integrator to assemble a fully functioning system from the subsystem components. The proposed IEEE 1451.6 standard (Kang 2000) goes partway to addressing these challenges, in that it provides a basic communications link for transducer nodes; however does not support dynamic programming of the nodes. The section below describes some of the available tools and methodologies that can be utilised to aid the process of system integration.

8.2.2.1 Electronic Data Sheets (EDS)

In this context, electronic data sheets are files that define the capabilities of CANopen nodes. EDS files offer a standardised way of specifying supported Object Dictionary

entries to describe how CANopen nodes can be integrated into system networks and also provide in-house documentation of the node. This standard electronically readable file format has facilitated the development of application software tools such as bus monitors, analysers and configuration tools that are capable of recognising which Object Dictionary entries are available in CANopen nodes. Any manufacturer of a CANopen node or module should supply the EDS when selling or making it available to third parties to aid system integrators in assembling the system.

The EDS is a text file similar to the ".ini" files used on "Microsoft Windows" operating systems and a regular ASCII text editor such as "Notepad" could be used to read and/or modify it. This is not recommended practice, however, as in order to maintain CANopen conformance, entries must not only have the appropriate parameters but some are also cross-referenced. There are commercially available software tools that support the creation and maintenance of EDS files by allowing the system designer to add/remove Object Dictionary entries to the "drag and drop" level. One such tool is "CANopen Architect" which allows set-up and specification of Object Dictionaries as well as generation of EDS files and even the necessary 'C' code header files and tables for use by MicroCANopen. Finally, another useful tool developed by National Instruments, utilises the EDS file for the CANopen Conformance Test. This test not only validates CANopen compliance, but also tests if a given node implements all the Object Dictionary entries specified in its EDS file.

8.2.2.2 CAN Bootloader

The feasibility of installing a bootloader within each node should be investigated. This would enable faster programming and firmware updates to be made remotely over the CAN bus (Foster 2003) from a host computer or configuration tool. This facility makes it possible to retrospectively reprogram the device to fix any errors or update it with the latest firmware version. For example, the recent Cassini-Huygens Titan mission experienced severe difficulties (Oberg 2004) which could have been more easily overcome with the ability to be reprogrammed remotely.

Providing bootloading capability over a CAN bus network is non-trivial and requires some forethought. For example, a system with a number of nodes may have identical firmware in several nodes. Again, the broadcast nature of the CAN protocol means that

all nodes on the CAN bus can receive all messages, therefore it might be more efficient to program these identical nodes in a single pass. However, in other cases where a node or many nodes are unique, it may only be necessary to establish peer-to-peer communications to program the device. Another interesting situation is bootloading in an active and functioning system. In this scenario, one or more of the nodes are taken off-line to update their firmware, yet the functionality of the entire system is not completely disabled. These issues are discussed in greater depth in Microchip Application Note 247 "A CAN Bootloader for PIC18F CAN Microcontrollers" (Foster 2003).

8.3 Benefits for External Parties

This research effort has set out to establish a methodology for development of flexible and robust instruments capable of operating nominally under extreme environmental conditions. The hydrothermal vent environment was targeted for the reason that academic and industrial partners provided the scientific motivation and access to the necessary deep-ocean research vehicles and platforms to allow the development of technology that could be applied to the construction of a microbial fluid sampling instrument. It is anticipated that this technology can be generalised and adapted for wider applications. For example, conventional oceanographic instruments, such as remotely operated vehicles (ROVs), Conductivity, Temperature and Depth (CTD) arrays, autonomous underwater vehicles (AUVs) would all benefit from a decentralised "plug-and-play" architecture. Such an approach would guarantee compatibility between instruments from different manufacturers and standardardisation would extend their useful working life (standards endure for decades, whereas technology can arrive and disappear within a decade).

Also, there is an emerging market for modular "plug-an-play" wiring harnesses and rad-hardened hardware/software interfaces for use on spacecraft (which embraces micro-satellites, space probes and multistage vehicles, such as rockets and space planes) in the commercial and military sectors. An application was submitted to the Department of Defense (DoD) in 2005 for funding from the Small Business Innovation Research program (Ref: SBIR AF05-031). The SBIR program funds early-stage R & D in small technology companies and is designed to stimulate technological innovation and

increase private sector commercialisation of federal R & D. More specifically, SBIR AF05-031 related to the design, build, and test an innovative, modular micro-satellite bus with interfaces to reduce system integration and facilitate rapid integration of a variety of payloads.

On this occasion the application was unsuccessful, however there is clearly an ongoing interest by the DoD in this type of technology, as a recent literature SBIR search found a funding program pursuing mechanical attachment solutions that allow for the rapid, on-demand assembly of satellites built from stocked components (Ref: SBIR AF071-288). Looking at other aerospace sectors, the emerging commercial space access industry will undoubtedly drive this technology forward, as reliable modular bus technology is required for space vehicles, such as multistage rockets, an intrinsically electrical noisy and mechanically harsh environment. Finally, looking beyond sub-orbital and low-earth orbit missions, NASA Aimes Research Centers' astrobiological research program (Flynn 2003) are motivated to continue developing this fault-tolerant bio-containment technology for deep space exploration. The ultimate goal of finding life on Earth in environments that are similar to conditions posited to exist on other planetary bodies such as Mars and Jupiter's moon, Europa.

8.4 A Final Word

Several ambitious concepts have been proposed in this thesis, which began with an evaluation of the feasibility of a union of "low-performance" PIC microprocessor core with Salvo real-time operating system and MicroCANopen protocol stack. These technologies were then leveraged to develop a low-power single board computer with system resources capable of supporting an open, peer-to-peer network with "plug-and-play" capability. A formal methodology was documented which describes how to utilise this new computational platform and distribute the firmware evenly across a network of nodes to construct a grid-enabled decentralised machine system. In practice, this modular approach to implementation proved beneficial within the university research and multidisciplinary science cruise context, as it was possible to incrementally test the instrument and acquire useful science data during development of the project.

Experimental testing confirmed that, CAN bus and CANopen higher level protocol operated as a reliable and transparent communication medium, making it possible to perform a basic proof of the grid concept and decentralised control. The research has raised interesting

questions on how to partition machine control amongst the nodes in the system and exchange information efficiently between them to maximise reliability of the system as a whole. Also, it became apparent that a more formal methodology would be useful to aid in maintaining the firmware for the different types of nodes within the system. The electronic data sheets partly address this in that they provide a system level mechanism to define the Object Dictionary, however more systematic housekeeping is also required for the source code files. Although these ideas require further investigation and refinement before the "Holy-Grail" solution of an off-the-shelf instrument is realised, it is hoped that the issues addressed in this work will contribute to the future development of machines for scientific and commercial exploration.

Appendix A References

Alicke, F.; Bartholdy, F.; Blozis, S.; Dehmelt, F.; Forstner, P.; Holland, N.; Huchzermeier, J. 2000. *Comparing Bus Solutions* Application Report SLLA067 Texas Instruments. [Online]. Available at: http://www.ti.com. [Accessed: 12 August 2008].

Amer, W. 2002. *Design of a PIC Microcontroller based Analogue Data Acquisition and Processing System.* MSc Thesis, Cardiff School of Engineering.

Arnaud, B. 2004. An Integrated Approach to Ocean Observatory Data Acquisition/Management and Infrastructure Control Using Web Services. *Marine Technology Society Journal*, Vol. 38, No. 2, pp. 155-163.

Baird, S .L. 2005. Deep Sea Exploration: Earth's Final Frontier: Only a Portion of the Potential of the Oceans Has Been Tapped, but It Is Clear That Exploring and Improving Our Understanding of the Ocean and Its Influence on Global Events Are among Our Most Important Challenges Today. *Journal Article : The Technology Teacher Journal.* Vol.65.

Baker, B. C. 2003. *Ease into the Flexible CANbus Network* DS21837A. [Online]. Available at: http://www.microchip.com. [Accessed: 12 August 2008], pp. 1.

Behar, A.; Matthews, J.; Venkateswaran, K.; Bruckner, J. 2006. A Deep Sea Hydrothermal Vent Bio-Sampler for Large Volume In-Situ Filtration of Hydrothermal Vent Fluids. *Pasadena, CA: JPL, NASA.*

Bell, J. 2002. *Network Protocols used in the Automotive Industry.* University of Wales, Aberystwyth. [Online]. Available at: http://www.aber.ac.uk/compsci/research/mbsg/fmeaprojects/softfmeatechreports/systems/protocols.pdf, July 2002 [Accessed: 12 July 2008], pp. 3.

Blandin, J. and Leon, P. 1998. Network Architectures for Underwater Systems: Two Applications of the CAN bus. *OCEANS '98 Conference Proceedings*, 28 Sep-1 Oct 1998, Vol. 1. Nice, France, pp. 503-507.

Boehm, B. 1981. *Software Engineering Economics*. Prentice Hall.
Boferenbrood, H. 2000. Design and Implementation of the ATLAS Detector Control System at CERN. *IEEE Transactions on Nuclear Science*, Vol. 51. Issue 3, pp. 495-501.

Bosch, R. 1991. *CAN Specification Version 2.0.* [Online]. Available at: http://www.semiconductors.bosch.de/pdf/can2spec.pdf, pp. 8-9, 25-26.

CAN-in-Automation, 1996. CAL (CAN Application Layer) for Industrial Applications Version 1.1, *CiA Draft Standard DS-201 to DS-207.*

Capsum. 2005. *METS Underwater Methane Sensor* data sheet. [Online]. Available at: http://www.capsum.com/capsum_online/mets.html [Accessed: 10 August 2008].

Cheshire, S. 1996. *It's the Latency, Stupid.* [Online]. Available at: http://www.stuartcheshire.org/rants/latency.html [Accessed: 20 August 2008], pp. 2.

Deitel, H. M. and Deitel P. J. 1999. *C: How To Program.* Second Edition Prentice Hall, pp. 126-133.

Deming, J. W. 2007. Of Ice and Microbes. *American Astronomical Society Meeting*, Seattle, 10-14 January 2007.

Delany, J. R. 2000. Neptune: Real-time Ocean and Earth Sciences at the Scale of a Tectonic Plate. *Oceanography*, Volume 13, No.2, pp. 71-79.

D'Hondt, S. 2002. Metabolic Activity of Subsurface Life in Deep-Sea Sediments. *Science*, 15 March 2002, Vol. 295, No. 5562, pp. 2067-2070.

Elliot, P. 1995. *Surface Mount Ceramic Resonators.* AVX Corporation , pp. 3.

Embedded Systems Academy, Inc. 2008. *P CANopen Inspector.* [Online]. Available at: http://www.canopenstore.com/test-tools.html [Accessed: 11 August 2008].

Embedded Systems Academy, Inc. 2008. *CANopen Architect EDS.* [Online]. Available at: http://www.canopenstore.com/configuration-and-analysis-tools.html [Accessed: 11 August 2008].

Etschberger, K. 2008. *A Failure Tolerant CANopen System for Marine Automation Systems.* [Online]. Available at: http://www.canopensolutions.com/english/articles/ar_2_e.shtml [Accessed: 25 August 2008].

Felser, M. and Sauter, T. 2002. The Fieldbus War: History or Short Break Between Battles. *Factory Communication Systems.* pp.73-80.

Foster, I. 2002. What is the Grid? A Three Point Checklist. *GRIDtoday*, Vol. 1, No. 6. July 2002.

Fredriksson, L. 1994. *Controller Area Networks and the Protocol CAN for Machine Control Systems.* [Online]. Available at: http://www.kvaser.com/index.htm [Accessed: 10 August 2008].

Fredriksson, L. 1995. *A CAN Kingdom Rev 3.01.* [Online]. Available at: http://www.kvaser.com/index.htm [Accessed: 10 August 2008].

Fredriksson, L. 2005. *On the Difference between CANopen and CAN Kingdom.* [Online]. Available at: http://www.kvaser.com/index.htm [Accessed: 10 August 2008].

Gamiz, J.; Samitier, J.; Fuertes, J.M.; Rubies, O. 2003. Practical Evaluation of Messages Latencies in CAN. *Emerging Technologies and Factory Automation*, Vol. 1, Sept 2003, pp. 185-192.

Ganssle, J. G. 1992. *The Art of Programming Embedded Systems.* Academic Press, Inc., pp. 45-51, 51-53.

Ganssle, J. G. 2000. *The Art of Designing Embedded Systems.* Newnes Butterworth-Heinemann, pp. 37-48, 51, 83-85, 203-221.

Ganssle, J. G. 2005. *Embedded Systems at Sea.* [Online]. Available at: http://www.ganssle.com/articles.htm [Accessed: 10 August 2008].

Ganssle, J. G. 2004. *Great Watchdogs Version 1.2.* [Online]. Available at: http://www.ganssle.com/articles.htm [Accessed: 10 August 2008].

Ganssle, J. G. 2005. *Keep it small - Get the Product out Faster by Better Partitioning.* [Online]. Available at: http://www.ganssle.com/articles.htm [Accessed: 10 August 2008].

Hawthorne, M. J. and Perry D. E. 2004. Architectural Styles for Adaptable Self-Healing Dependable Systems Empirical Software Engineering Lab (ESEL), pp. 3.

Hendry, G. R. 1999. *Standard Ethernet as an Embedded Communication Network.* MSc, Department of Electrical and Computer Engineering, Carnegie Mellon University, pp. 1, 24-28, 31-32.

Holub, A. I. 1995. *Enough Rope to Shoot yourself in the Foot* McGraw-Hill, pp. 3.

Jannasch, H. W. and Maddox, W. S. 1967. A Note on Bacteriological Sampling in Seawater. *Journal of Marine Research*, Vol. 25, pp. 185-189.

Jannasch, H. W. and Wirsen, C. O. 1977. Retrievel of Concentrated Un-decompressed Microbial Populations from the Deep Sea. *Applied and Environmental Microbiology*, Vol. 33, pp. 642-646.

Kalman, A. 2004. *Salvo Real Time Operating System User Manual* Pumpkin Real Time Software. [Online]. Available at: http://www.pumpkininc.com [Accessed: 2 June 2005].

Kleinknecht, H. 1999. *CAN Calibration Protocol.* Version 2.1. Association for Standardisation of Automation and Measuring Systems. [Online]. Available at: http://www.asam.net [Accessed: 21 July 2008].

Kooperman, P. and Chakravarty T. 2004. Cyclic Redundancy Code (CRC) Polynomial Selection for Embedded Networks. *Proceedings of the International Conference on Dependable Systems and Networks*, June 2004, pp 145.

Lee, K. 2000. Distributed Measurement and Control Based on the IEEE 1451 Smart Transducer Interface Standard. *IEEE Transactions on Instrumentation and Measurement.* Vol. 49 No.3, June 2000, pp. 2-3.

Lennartsson, K. F. and Fredriksson, L. 2005. *CAN HLP Brief Comparison.* [Online]. Available at: http://www.kvaser.com/index.htm [Accessed: 10 August 2008].

Lewis, J. P. Large Limits to Software Estimation. *ACM Software engineering Notes* Vol. 26, No. 4, July 2001, pp. 54-59.

Malahoff, A.; Gregory, T.; Bossuyt, A.; Donachie, S.; Alarn, M. 2002. A Seamless system for the Collection and Cultivation of Extremeophiles from Deep-Ocean Hydrothermal Vents. *IEEE Journal of Oceanic Engineering,* Vol. 27 No. 4.

Marsh, D. 2000. Drive by wire fuels network-highway race *EDN Europe*, [Online]. Available at: http://www.edn.com/article/CA83747.html [Accessed: 10 August 2008], pp. 178.

Martin, W. and Russell, M. 2004. The Rocky Roots of the acetyl-CoA Pathway. *Trends in Biochemical Sciences*, Vol. 29, No. 7, July 2004, pp. 358-363.

Maxim Integrated Products. 2002. *AN58 Crystal Considerations with Dallas Real-Time Clocks (RTCs).* [Online]. Available at: http://www.maxim-ic.com. [Accessed: 12 August 2008].

Microchip Technology Inc., 1992. *AN536 Basic Serial EEPROM Operation data sheet DS00536C.* [Online]. Available at: http://www.microchip.com. [Accessed: 12 August 2008].

Microchip Technology Inc. 1993. *AN537 Everything a System Engineer Needs to Know About Serial EEPROM Endurance data sheet DS00537A.* [Online]. Available at: http://www.microchip.com. [Accessed: 12 August 2008].

Momut, R. January 2000. *Open Systems Interconnection (OSI) Reference Model.* Arescom Inc., pp 1-3.

Naganuma, T.; Kyo M.; Ueki, T.; Takeda, K.; Ishibashi, J. 1998. A New, Automatic Hydrothermal Fluid Sampler Using a Shape-Memory Alloy. *Journal of Oceanography*, Vol. 54, pp. 241-246.

National Instruments. 2008. *CANopen LabView Library.* [Online]. Available at: http://sine.ni.com/nips/cds/view/p/lang/en/nid/202614 [Accessed: 17 August 2008].

Nibet, E. G. and Sleep, N .H. 2001. The Habitat and Nature of Early Life. *Nature*, Vol. 409, 22 Feb 2001, pp. 1083-1091.

Niskin, S. J. 1962. A Water Sampler for Microbial Studies. *Deep Sea-Research*, Vol. 9, pp. 501-503.

NOAA. 2008. *National Oceanic and Atmospheric Administration.* [Online]. Available at: http://www.noaa.gov/ocean.html. [Accessed: 5 July 2008].

Nolte, T. January 2000. *Reducing Pessimism in CAN response time analysis.* Mälardalem Real-Time Research Centre. Department of Computer Engineering, Sweden.

Oberg, J. 2004. Titan Calling – How a Swedish Engineer saved a once-in-a-lifetime mission to Saturn's Mysterious Moon. *IEEE Spectrum.* October 2004. pp.33.

Opluštil, V.; Gáspár, L.; Svacina, D.; Szabó, S. 2004. COTS (Commercial Off The Shelf) Distributed System for Critical Application. *Proceedings of the 11th IEEE International Conference and Workshop on Engineering of Computer based Systems.* pp 464-468.

Pazul, K. 1999. AN713 *Control Area Network (CAN) Basics Microchip Technology* Inc., pp. 3. [Online]. Available at: http://www.microchip.com. [Accessed: 12 August 2008].

Pedersen, R. B.; Thorseth, I. H.; Hellevang, B.; Schultz, A.; Taylor, P.; Knudsen, H. P.; Steinsbu, B. O. 2005. Two Vent Fields Discovered at the Ultraslow Spreading Arctic Ridge System. *American Geophysical Union.*

Person, R.; Blandin, J.; Stout, J.M.; Briole, P.; Ballu, V.; Etiope, G.; Ferentinos, G.; Masson, M.; Smolders, S.; Lykousis, V. 2003. ASSEM: a new concept of observatory applied to long term seabed monitoring of geohazards. *OCEANS 2003 Conference Proceedings.*
Vol. 1, pp. 86-90.

Person, P.; Aoustin, Y.; Blandin, J.; Marvaldi, J.; Rolin, J. 2006. From Bottom Landers to Observatory Networks. *Anals of Geophysics.* Vol. 49. April/June 2006.

Pfeiffer, O. 2003. Approximating CANopen - Embedded Systems Design. *British Library Direct*, Vol. 16, Part 9, pp 28-35.

Pfeiffer, O. 2003. *Embedded Networking with CAN and CANopen.* RTC Books pp. 27, 83-91, 114.

Rajbharti, N. 2001. AN738 *PIC18C CAN Routines in 'C'.* [Online]. Available at: http://www.microchip.com. [Accessed: 12 August 2008], pp. 1.

Rajbharti, N. 2002. *The Microchip TCP/IP Stack* DS00833B. [Online]. Available at: http://www.microchip.com. [Accessed: 12 August 2008], pp. 30.

Richards, P. 2002. AN228 *A CAN Physical Layer Discussion.* [Online]. Available at: http://www.microchip.com. [Accessed: 12 August 2008], pp. 1, 3, 9.

Richards, P. 2000. *Using the CAN Developer's Kit.* [Online]. Available at: http://www.microchip.com. [Accessed: 12 August 2008].

Richards, P. 2001. AN739 *An In-depth Look at the MCP2510.* [Online]. Available at: http://www.microchip.com. [Accessed: 12 August 2008].

Richards, P. 2001. AN754 *Understanding Microchip's CAN Module.* [Online]. Available at: http://www.microchip.com. [Accessed: 12 August 2008], pp 1-6.

Richey, R. 1997. AN606 *Low Power Design Using PICmicro Microcontrollers* DS00606B. . [Online]. Available at: http://www.microchip.com. [Accessed: 12 August 2008], pp. 1.

Sarradin, P.M. Sarrazin, J. Allais, A.G. Almeida, D. Brandou, V. Boetius, A. Buffier, E. Coiras, E. Colaco, A. Cormack, A. Dentrecolas, S. Desbruyeres, D. Dorval, P. du Buf, H. Dupont, J. Godfroy, A. Gouillou, M. Gronemann, J. Hamel, G. Hamon, M. Hoge, U. Lane, D. Le Gall, C. Leroux, D. Legrand, J. Leon, P. Leveque, J.P. Masson, M. Olu, K. Pascoal, A. Sauter, E. Sanfilippo, L. Savino, E. Sebastiao, L. Serrao Santos, R. Shillito, B. Simeoni, P. Schultz, A. Sudreau, J.P. Taylor, P. Vuillemin, R. Waldmann, C. Wenzhofer, F. Zal, F. 2007. EXtreme ecosystem studies in the deep OCEan: Technological Developments. *IEEE/OES Oceans 2007 Conference – Europe.* 18-21 June 2007, pp. 1-6.

Schultz, A.; Pedersen, R. B.; Thorseth, I. H.; Taylor, P.; Flynn, M. 2005. Fluid Flow Rate, Temperature and Heat Flux at Mohns Ridge Vent Fields: Evidence from Isosampler Measurements for Phase Separated Hydrothermal Circulation Along the Arctic Ridge System. *American Geophysical Union.*

Seabird. 2008. *Submersible Pump data sheet.* [Online]. Available at: http://www.seabird.com/products/spec_sheets/5Tdata.htm [Accessed: 10 August 2008].

Seabird. 2008. *SBE 3S Oceanographic Temperature Sensor.* [Online]. Available at: http://www.seabird.com/products/spec_sheets/modular.htm [Accessed: 10 August 2008].

Seabird. 2008. *SBE 50 Digital Oceanographic Pressure Sensor.* [Online]. Available at: http://www.seabird.com/products/spec_sheets/50data.htm [Accessed: 10 August 2008].

Snowdon, D. 2002. *Hardware and Software Infrastructure for the Optimisation of Sunswift II.* BEng, School of Electrical Engineering & Telecommunications, University of New South Wales.

Stanczyk, M. 2000. *AN212 Smart Sensor CAN Node using the MCP2510 and PIC16F876.* [Online]. Available at: http://www.microchip.com. [Accessed: 12 August 2008], pp. 1, 3.

Stock, M. 1998. *CAN Aerospace Interface Specification for Airborne CAN Stock Flight Systems.* [Online]. Available at: http://www.a2tech.eu/CANaerospace001.html [Accessed: 10 August 2008], pp. 53.

Tabor, P.S., Deming, J. W., Ohwada, K., Davies, H., Waxman, M., Colwell, R. R. 1981. A Pressure-Retaining Deep Ocean Sampler and Transfer System for Measurement of Microbial Activity in the Deep-Sea. *Microbial Ecology*, Vol. 7, pp. 51-65.

Taylor, C.D.; Dohertyb, K.W.; Molyneauxa, S. J.; Morrison, A.T.; Billingsd, J.D.; Engstromb, I.B., Pfitschb, D.W.; Honjob, S. 2006. Autonomous Microbial Sampler (AMS), A Device for the Uncontaminated Collection of Multiple Microbial Samples from Submarine Hydrothermal Vents and other Aquatic Environments. Biology Department, Woods Hole Oceanographic Institution. *Deep Sea Research Part I: Oceanographic research Papers* Volume 53, Issue 5, May 2006, pp. 894-916.

Twose, C.; Parker, R.; Okabe, T.; Kirihara, K.; Nakata, H.; Kondo, T.; Karasawa, H.; and Cappetti, G. 2000. Development of a High Temperature Borehole Fluid Sampler and its Field Experiments in the Larderello Geothermal Field, Italia Proceedings World Geothermal Congress. *Published by International Geothermal Association.*

Compaq, Intel, Microsoft & NEC Corporation, pp. 17, 1998. *Universal Serial Bus Specification* Revision 1.1.

Valenti, C. and Kalman, A. 2001. AN777 *Multi-Tasking on the PIC16F877 with the SALVO RTOS* DS00777B Microchip Technology Inc. [Online]. Available at: http://www.microchip.com. [Accessed: 5 May 2006], pp. 1.

Vector Informatik GmbH. 2008. *Solutions for your CANopen Networking.* [Online]. Available at: http://www.canopen-solutions.com/canopen_index_en.html [Accessed: 11 August 2008].

Vig, J. R. 2005. *Quartz Crystal Resonators and Oscillators for Frequency Control and Timing Applications.* [Online]. Available at: http://www.rakon.com.

Wächtershäuser, G. 2000. Origin of Life: Life as We Don't Know it. *Science*, Vol. 289, No. 5483, 25 Aug 2000, pp. 1307-1308.

Weik, M. H. 1989. *Communications Standard Dictionary.* Second Edition. New York: Van Nostrand Reinhold.

Wetlabs. 2008. *ECO Fluorometer User's Guide Revision AF.* [Online]. Available at: http://www.wetlabs.com/products/eflcombo/fl.htm [Accessed: 10 August 2008].

Wetlabs. 2007. *WETStar User's Guide Revision N.* [Online]. Available at: http://www.wetlabs.com/products/wetstar/wsx.htm [Accessed: 6 August 2008].

Woodroffe, A. and Madle, P. *Application and experience of CAN as a low cost OBDH bus system.* Surrey Satellite Technologies Ltd. [Online]. Available at: http://www.klabs.org/mapld04/abstracts/woodroffe_a.pdf [Accessed: 16 November 2004].

Woolever, B. 1999. *SDS Physical Layer Specification* – Version 2.0. Honeywell Inc. MICRO SWITCH Division.

Yin, E. 2003. *Implementing CANopen in Autonomous Underwater Vehicles.* Stanford University. [Online]. Available at:

http://www.mbari.org/education/internship/03interns/03papers/EYin.pdf [Accessed: 7 October 2004], pp. 1-2.

Zeltwanger, H. 2004. Cascadable Sensor Network. *Sensors & Transducers Magazine*, Vol.41, Issue 3, March 2004, pp.197-200.

Zobell, C .E. 1941. Apparatus for Collecting Water Samples from Different Depths for Bacteriological Analysis. *Journal of Marine Research*, Vol. 4, pp 173-178.