Utilising a fieldbus protocol in a water quality monitoring system

Simon Watson

Utilising a Fieldbus Protocol in a Water Quality Monitoring System

Simon Watson

This thesis is submitted in fulfilment of the requirements

for the Degree of Master of Science

University of Durham
Science Site
South Road
DURHAM
DH1 3LE

School of Engineering

1996

Abstract

This thesis presents a new water quality monitoring system developed at the University of Durham in conjunction with Partech Instruments Ltd. The system uses a fieldbus protocol to create an open, distributed control network, replacing the dedicated products currently offered. Echelon LonWorks has been used to create three nodes: a suspended solids sensor, a general-purpose interactive monitoring tool, and a universal relay setpoint module. When connected, these nodes provide a means of activating relays when the suspended solids level reaches a definable level, while providing a numerical display for the operator. The sensor may be calibrated for a number of different applications.

The sensor uses infra-red light to monitor the light absorption and 90° scatter within the solution. By dynamically adjusting the intensity of the emitted light, the sensor is able to increase its range over conventional devices. Signal processing, linearisation and calibration operations are carried out within the sensor software. The final measurement is communicated as a LonWorks network variable, allowing the sensor to be treated as an interoperable device.

Several third-party products have been connected to the network and a high degree of interoperability demonstrated. Three network management software packages have been investigated, and their suitability assessed.

The final prototype system shows the power, flexibility and cost-saving that a fieldbus protocol can provide in an industrial control environment.

Acknowledgements

I would like to express my gratitude to the following:

To Dr. Clive Preece, my supervisor, for his help and support during the research, and for his advice towards this thesis.

To Partech Instruments Ltd. for their financial assistance, and to all the staff, particularly David Parker and Roger Henderson.

List of Contents

List of Figures

Declaration

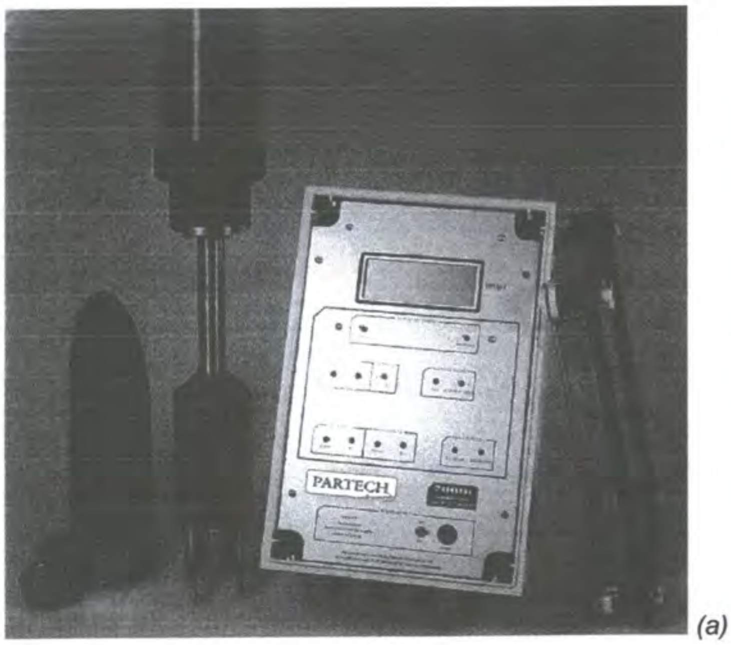
The work presented in this thesis is the author's own work and has not been previously submitted for any other degree.

Chapter 1

Introduction

The main aim of this project was to develop a prototype of a new commercial water quality monitoring system for Partech Instruments Ltd. - a small family-owned company that has been producing water monitoring systems for over thirty years. Partech designs and manufactures instruments specifically for the water and waste water industry, and has been the innovator of many principles now regarded as international standards. The measurement of water quality is becoming increasingly important in today's environmentally-conscious world, not only for raising the standard of water delivered to consumers, but for improving the efficiency of industrial processes and for preventing unwanted, and potentially hazardous, discharges into the natural water supply. Whilst there are many parameters associated with water quality, this project is concerned with only two: turbidity and suspended solids.

A modern monitoring system typically comprises a sensor and a controller (Figure 1.1.), often separated by some distance. The sensor optically measures the turbidity and / or the suspended solids content of the sample, translating this into an electrical signal which is communicated to the controller. The controller serves to further process this signal, generally performing linearisation and calibration calculations, before finally providing a visual indication of the result. A secondary function of the controller is to generate a controlled output signal representing the measured value, to interface with external control and monitoring systems.



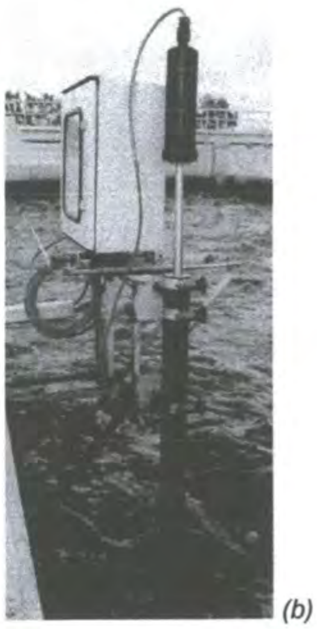


Figure 1.1. *(a)* A typical water monitoring system consisting of sensor and controller (unconnected). *(b)* A similar combination operating in a water treatment tank.

The first aim of this project was to develop the optical measuring process used within the sensor, along with its associated electronics, to improve the accuracy, repeatability, range of measurement, and to standardise on a single optical arrangement if possible. The second aim was to incorporate a processing capability within the sensor so that it becomes an 'intelligent' self-contained device, capable of outputting a final, meaningful measurement without the need for a dedicated controller.

At an early stage in the project planning, Partech decided that rather than develop a proprietary communications protocol for the output of the sensor, an existing fieldbus technology should be used. The main reason for this approach is was to avoid 're-inventing the wheel', instead taking advantage of existing tried and tested technology. A second advantage of having a fieldbus sensor is that such a device could instantly become an 'intelligent' node on a larger control network, capable of interacting with products from other manufacturers. This idea of 'interoperability' is becoming increasingly important within the control industry.

1.1. Water quality

Turbidity

The regulators in the UK and especially the USA are attaching increasing importance to turbidity as a water quality parameter. Turbidity is used as a measure of the 'clarity' or 'cloudiness' of a liquid. In the BSI standards document for water quality[1] clarity is defined as a 'reduction of transparency caused by the presence of undissolved matter'. It gives a good indication of the appearance of a water in an aesthetic sense, but it is also an indicator of the amount of suspended material. It is essentially an optical parameter and cannot be measured in any other way.

Turbidity is measured at many points through the water treatment process. In treated water it is useful as a 'catch all' water quality parameter, since many different water problems can result in a rise in turbidity. Turbidity rather than suspended solids is also widely used in the monitoring of surface waters, and as such gives a direct indication of the loss of light in the aquatic environment.

Turbidity has always been defined in an empirical way and different industries have defined turbidity to suit their own usage, leading to a variety of units and standard suspensions. Formazine is the most widely used standard suspension at present. It has some drawbacks, but its great merits is that it can be made in any laboratory with simple equipment to a tolerance which is acceptable for most purposes. In most applications the measured value of the turbidity of a suspension is depended on the instrument used and a turbidity value generally requires full details as to how the measurement was made in order to be meaningful.

Suspended Solids

Whilst turbidity measurement may be concerned with the appearance of a sample, suspended solids or sludge density are measured to give an indication of such things as loading or strength of a wastewater, efficiency of solids removal, or simply the dry solids content for disposal or further processing.

Suspended solids is defined gravimetrically, that is by the mass of dry solid material in the sample. The laboratory method involves manual filtration and drying of the sample residue. The process cannot easily be automated and so all online methods for suspended solids measurement are essentially inferential. This parameter is used rather than turbidity in the sewage treatment process to monitor the removal of solid material, and has been used as a basic measure of the polluting load in wastewater since the earliest days of such treatments.

Suspended solids measurement is dominated by optical instrumentation, much of which is similar or identical to the turbidimeters used in potable water applications. Sludge density can be measured using a wider range of techniques. This is possible because at the higher solids concentrations of sludges, properties of the sample other than the optical ones are significantly affected by the presence of the solids. The inferential nature of all the technologies currently in use for suspended solids or sludge density means that the success of an installation depends upon the relationship between the parameter measured, such as the loss of light transmitted across a gap, and the solids present in the sample. This relationship and its stability can seldom be taken for granted, requiring frequent user intervention to re-calibrate the instrument against a sample which has undergone recent gravimetric analysis.

1.2. Measurement types

Optical water quality sensors operate in two basic modes. The first is to measure the intensity of scattered light, the second is to measure the absorption of transmitted light. Often, both of these modes of operation are utilised within a single sensor.

Scattered light sensors

A schematic diagram of a scattered light sensor is shown in figure 1.2. The incident light is collimated by a lens before being allowed to pass through the sample, with a detector measuring the scattered light at some angle, commonly 90° as shown. The

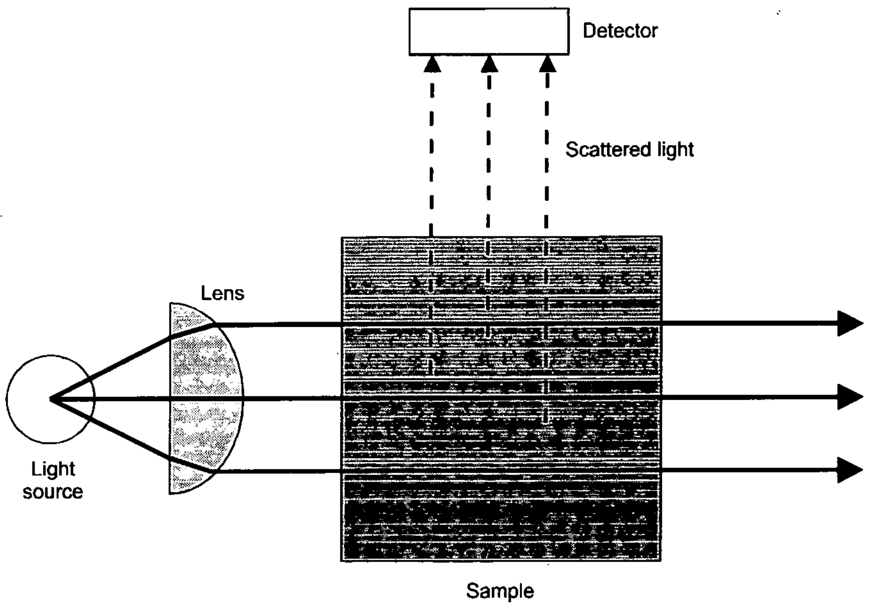


Figure 1.2. A simple scattered light sensor. Light scattered at 90° to the source light is detected.

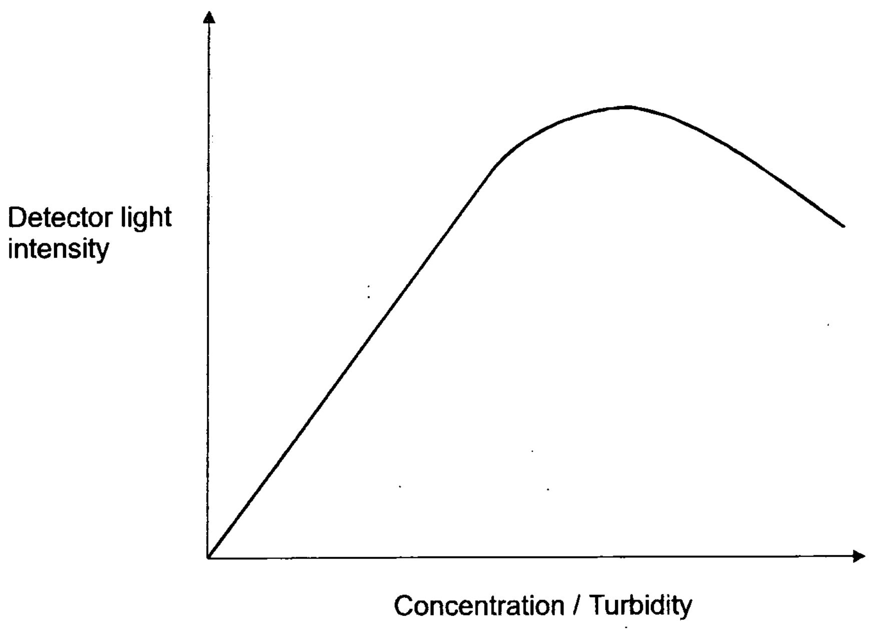


Figure 1.3. The general response of a scattered light sensor to increasing sample concentration (independent of units).

response of the detector to increasing turbidity is depicted in figure 1.3. It may be seen that at zero concentration the detector registers a near-zero output (an offset is often observed due to ambient light) which then rises almost linearly with increasing concentration. Eventually, with increasing concentration, the detector output falls as the scattered light itself is absorbed and scattered before reaching the detector.

Transmitted light sensors

A transmitted light sensor is shown schematically in figure 1.4, in which the reduction in intensity of light passing through the sample is measured by a detector in line with the incident light. The light will be both obscured and scattered by suspended particles in the sample. Some of the scattered light may reach the detector depending on the precise geometry of the instrument. The response of the detector to increasing concentrations is non-linear and illustrated in figure 1.5. It may be seen that at zero concentration the full light intensity reaches the detector. As the concentration increases the intensity falls off, following an exponential-like form, reaching near-zero for a totally opaque liquid.

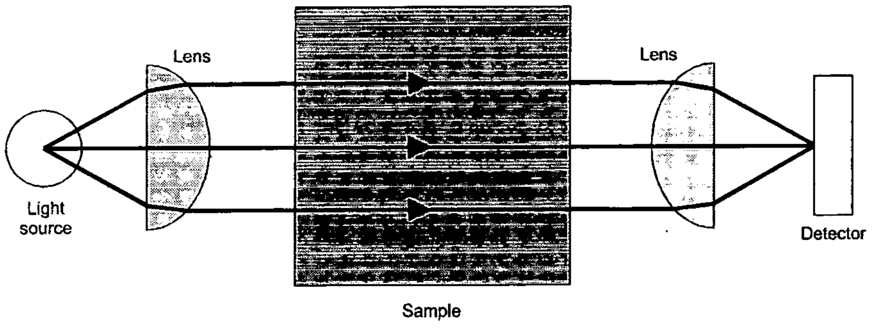


Figure 1.4. A simple transmitted light sensor. Only the light that is not scattered or absorbed by the sample is detected.

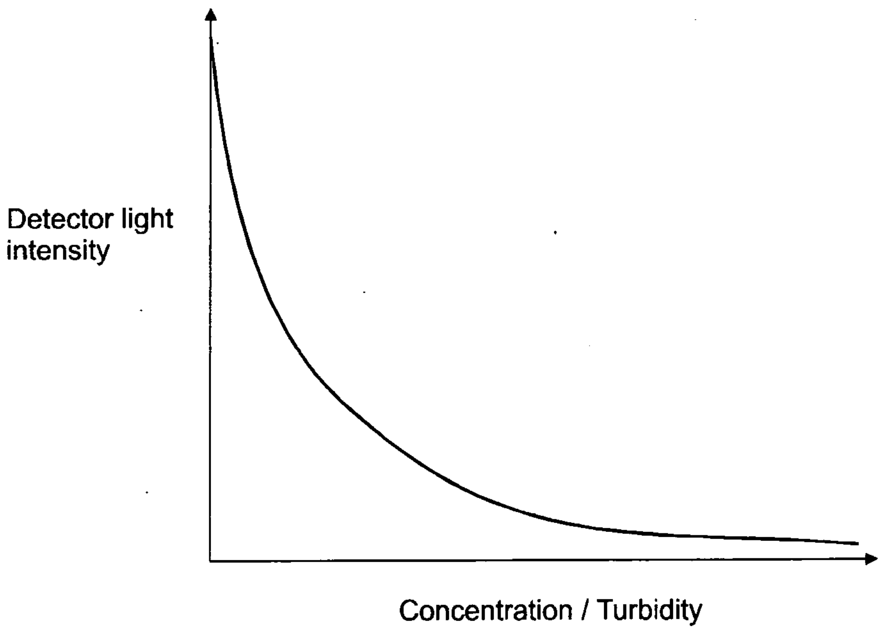


Figure 1.5. The general response of a tranmitted light sensor to increasing sample concentration (independent of units).

1.3. Fieldbus

While the sensing method used for obtaining a measurement is extremely important, of equal importance is the method used for communicating that measurement to the outside world. Existing Partech sensors use a very simple proprietary communications protocol which this project replaces with a fieldbus system. 'Fieldbus' is a generic term, describing a digital, bi-directional, serial-bus communications network protocol used to connect field-level devices to each other and to control systems. Considering the maturity of the process and automation industries, the lack of some de facto standard digital network is somewhat surprising. In reality, the most standard feature of today's industrial-control systems is still the 4- to 20-mA analogue signal that has been in use for decades. In an attempt to fill this void, there are now many rival fieldbus systems being promoted, each attempting to gain the critical market share needed to ensure success - some of the leading names are shown in figure 1.6.

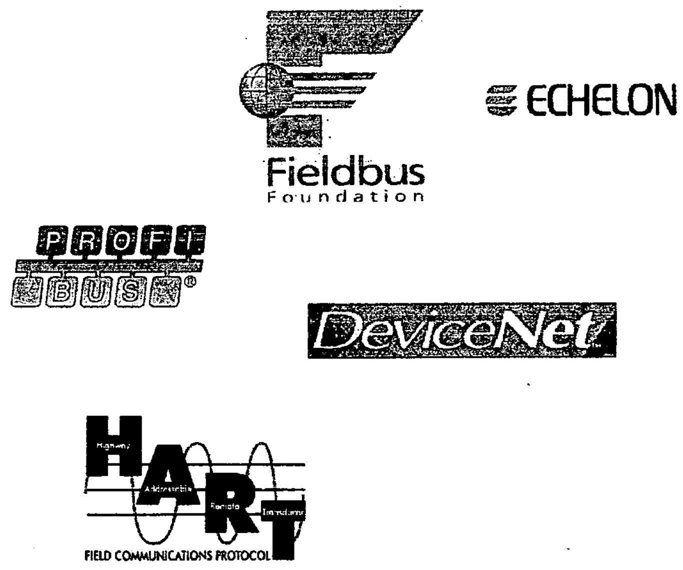
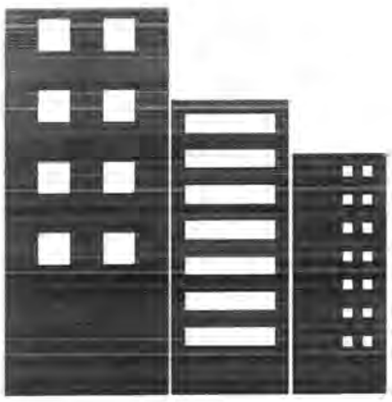
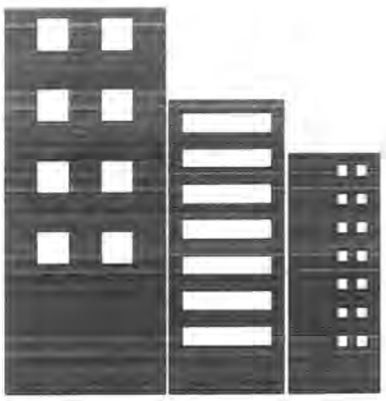


Figure 1.6. Some of the leading fieldbus contenders.

Through fieldbus, users will realise benefits such as reduced wiring, interoperability among devices from different manufacturers, enhanced field-level control, simpler integration and easier maintenance. Ultimately, by installing low-cost computing power in each field device, fieldbus will replace centralised control by distributed-control networks. This will enable greater manufacturing flexibility, productivity, higher quality products, and improved regulatory compliance.

An example of a fieldbus application





Imagine two buildings which appear to be identical. They both have very sophisticated control systems for maximising the comfort and convenience of the occupants while minimising energy consumption and cost. For example, in both buildings there are presence detectors in every room, and lighting is turned off if there are no people present. Occupancy sensors also tell the air conditioning system if cooling is really needed, and can also alert the security system if intruders have entered a restricted area after hours. The smoke sensors in the buildings, which alert the fire detection systems of potential fires, let the ventilating systems know which dampers to close to prevent the spread of smoke, and tell the lifts which floors to avoid as well. The occupants have the ability to control their environments - temperatures, light levels, etc. - down to individual offices or cubicles. Additionally, the systems provide the building owner and occupants with a wealth of information concerning energy usage and occupancy patterns to help allocate costs and plan future growth.

Although the buildings look identical, in fact there is a key difference between the two: The first uses proprietary, non-integrated control systems - intelligent but separate - each with its own proprietary communications technology. These separate systems have been made to work together by developing custom hardware and software gateways that convert information for the language of one system to that of another. The second building delivers the same functionality using an integrated network of intelligent, communicating devices that all 'speak' a common language, or protocol. They are connected in an integrated, interoperable control network, in which devices share information and co-operate without the need for special gateways or custom software.

For an occupant of one of the two buildings it would be difficult to determine which was which. Both buildings would be comfortable, responsive, and use the same amount of energy if they had the same activities going on inside. In fact, the only way to distinguish between the two buildings would be to make a change to the systems. For example, a large conference room may need to be divided into two smaller rooms. In the building with proprietary systems, the change will probably be difficult to make. Custom software will need to be written for the gateways to reflect the changes and keep the various systems coordinated. It will probably be necessary to add new wiring for all of the systems involved in the changes, since none of the proprietary systems share common wiring. If new components are required, they may only be available from the manufacturer of the original systems. The co-ordination of the different vendors involved requires added time and added cost.

In contrast, the building that is controlled by interoperable devices is much easier to change. It probably will not be necessary to add any new wiring, since the various devices share common communications media - even if they're made by different vendors. Devices from a number of suppliers can be considered based on the best combination of features, cost, delivery, support, etc. Integration of the devices and reconfiguration of the system is easy and straightforward because all of the devices are designed to communicate and cooperate in a standard way. In the building that takes advantage of fieldbus technology, changes are easier to plan, easier to execute, take less time, and cost less money.

Although the above example focuses on buildings, the same factors hold true in virtually all environments. Whether the problem is automating and controlling buildings, factories, industrial plants, oil refineries, homes, ships, or farms, the dominant costs for control systems are in the costs of installation - the design, the wiring, the set-up ,the testing - both for initial installation and for the inevitable changes that are required over time. In fact, the life cycle costs of controlling any plant, process, environment, or machine are usually dominated by the costs of making changes over the life of the system.

For virtually any control system, typically 70-80% of the total installed cost goes to design, installation (mostly wiring), and commissioning[2]. Only 20-30% of the cost is for the devices themselves. Every installation is unique and every job may require a specifier or integrator to 'reinvent the wheel'. Custom software, custom hardware, and special installation practices all take time and thus reduce profit. Interoperable products will allow specifiers and installers to respond more rapidly to change, be more competitive, and to sell into incremental markets.

A comparison with the computer industry

The computer industry has undergone a nearly total restructuring over the last ten years, driven in large part by the same forces now driving the controls industry. The computer business used to be dominated by mainframe suppliers and systems. These hierarchical systems were typically rigid and difficult to change, and used proprietary communications technologies with the result that the bulk of components were sourced from a single supplier. Mainframe systems later gave way to minicomputers, which had lower start up costs, making them easier to acquire by smaller companies and departments. These systems made greater use of open communications technologies (LANs, etc.) enabling a wider choice of products from a bigger field of suppliers.

The computer market today is dominated by suppliers of compatible PCs, workstations, and servers. The products are arranged in flat, peer-to-peer networks dominated by standard hardware platforms, standard operating systems, and standard networking technologies. Thousands of companies have participated in this exploding market, supplying hardware, software, services, and solutions. The largest markets are for fully compatible products that easily integrate with one another.

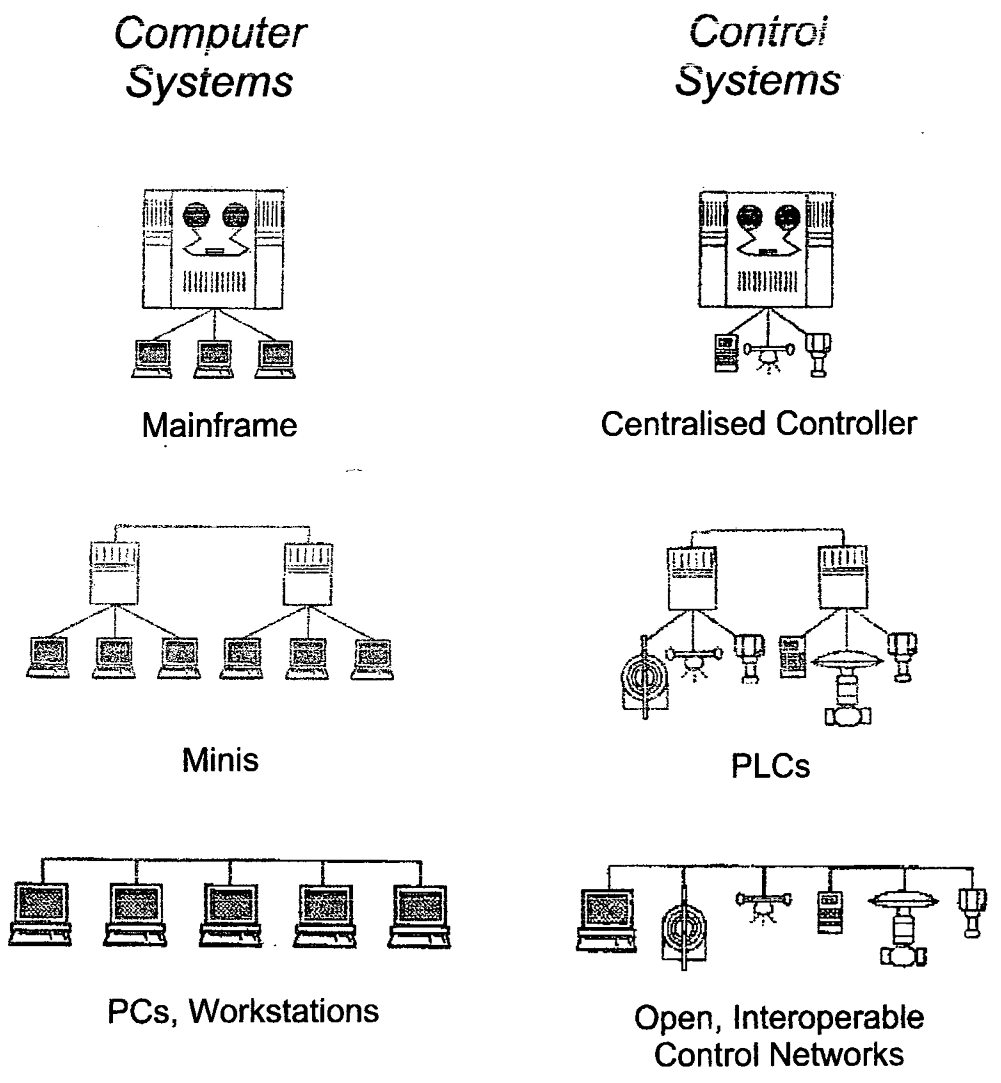


Figure 1.7. A comparison between the evolution of computer systems and control systems.

Noting these trends, the parallels with the control industry are unmistakable. Although, approximately a decade behind, the control industry has largely progressed from installations relying on a single central controller to ones using a larger number of programmable logic controllers (PLCs). The next step is to move to a completely distributed control network taking advantage of a fast field-level digital communications standard, to directly connect sensor and actuators.

The fieldbus contenders

The delay in the emergence of a widely agreed fieldbus standard has created a vacuum into which several suppliers of digital communication systems have stepped. Figure 1.8. presents a summary of the main contenders.

One of the most 'open' standards is being developed by the Fieldbus Foundation (FF) but, perhaps because of its desire to be open and internal negotiations between its board of representatives, has yet to publish an initial specification. The first FF products are not expected to be available until mid 1997 and then only for the slow version.

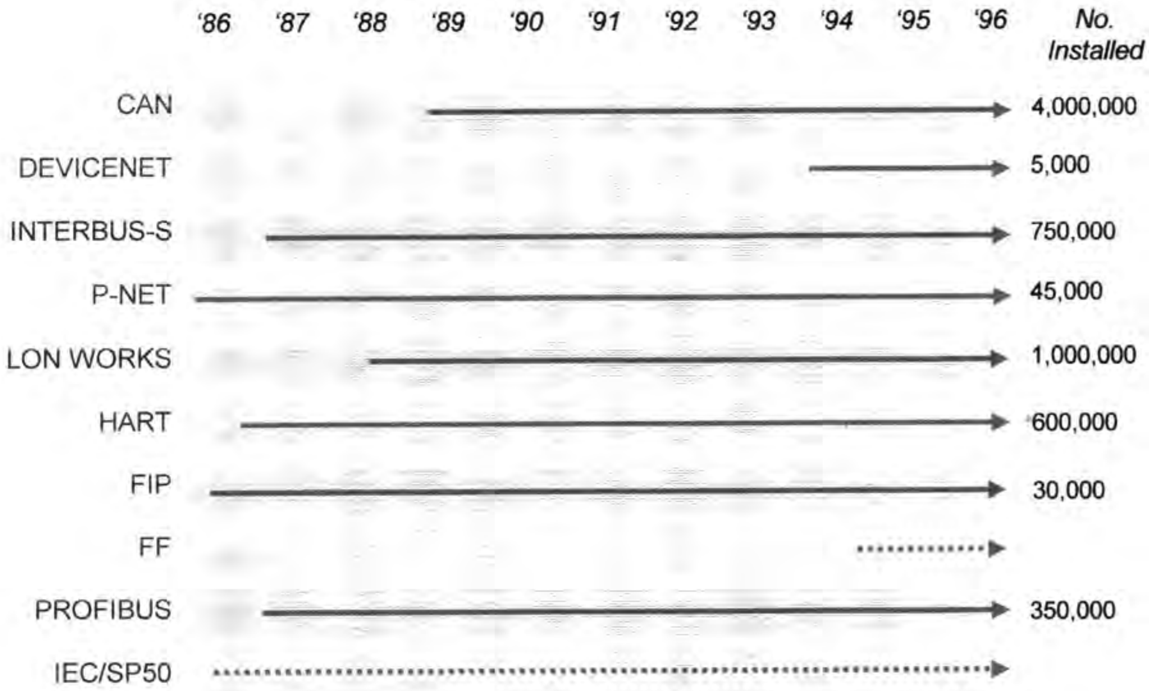


Figure 1.8. An indication of the progress of the main fieldbus contenders. The numbers on the right indicate the number of installed nodes, based on data supplied by relevant solutions providers/user groups. Source: Fieldcomms'95 [3].

FIP, the European wing of World-FIP (which merged with the InterOperable Systems Project, to form the Fieldbus Foundation), is continuing to market its products and services vigorously, although mainly concentrated in France and Italy. Profibus and Interbus-S are both German-developed systems hoping to expand out from niche markets to become truly international standards.

The Control Area Network (CAN) system was originally developed to simplify car wiring by providing a digital link between components such as switches, electric windows, lighting and instruments. CAN's attractions include the fact that it has been shown to work reliably in different environments and that dedicated integrated circuits are relatively cheap and readily available. Unfortunately, CAN's automotive origins also mean that it is not optimised for industrial applications. So, for example, the length of messages it can transmit is limited, as is the distance over which they can be sent. Systems such as DeviceNet, SDS (Smart Distributed System), and P-Net extend CAN, promising peer-to-peer communications, but have yet to show that they can serve as a full fieldbus standard.

The Hart (Highway Addressable Remote Transmitter) system adds digital communications to conventional 4- to 20-mA current loops. Although Hart does not pretend to be a true field bus, its supporters believe it has an important role to play, especially in process installations with a combination of analogue and digital equipment.

Another contender for the role of global industrial networking standard is Echelon® LonWorks®[3]. Despite being developed by a single commercial organisation, the system enjoys support from a diverse range of international companies, finding applications in factory control, environmental systems, petrochemicals, robotics, test equipment and building controls.

Selecting a fieldbus

Given the complicated and fragmented nature of the fieldbus marketplace, selecting one system over another is not an easy task, and has given rise to a great deal of confusion within the industry. Not surprisingly, the majority of end-users have opted to wait for a dominant standard to emerge out of the competing systems. For manufacturers, the fear of competitors gaining an advantage has forced many to gamble, although, the current feeling within the industry is that there is perhaps room for more than one 'standard' to enjoy success.

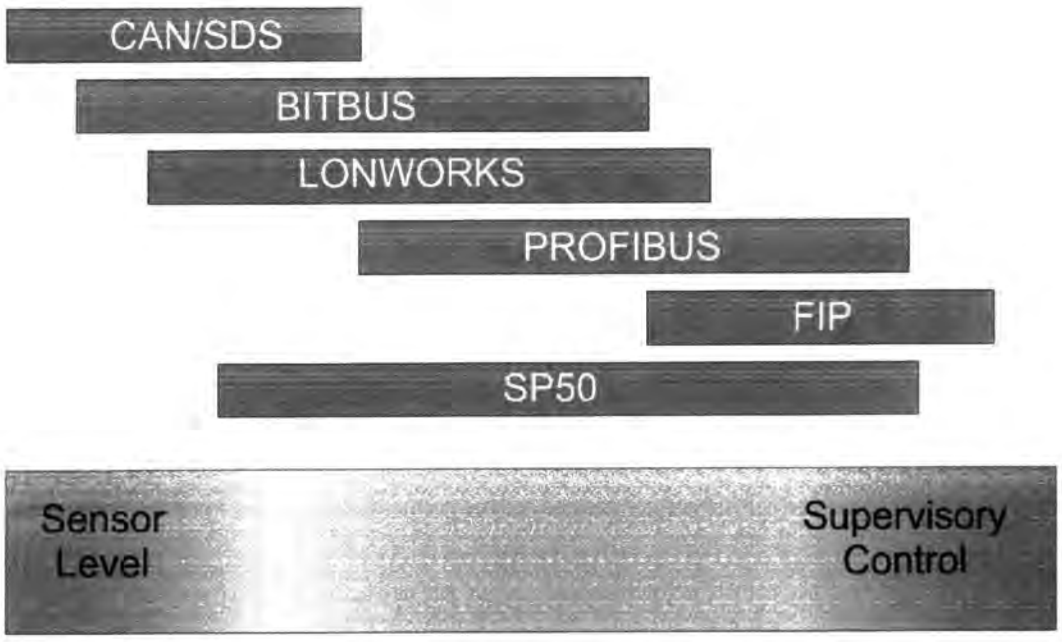


Figure 1.9. A comparison of the scope of competing fieldbus systems.

For certain applications, the benefit of interoperable products based on an open standard is not seen as a significant advantage. In these markets, more specific 'niche' systems will continue to flourish. For manufacturers wishing to target a broader spectrum of applications, the choice of a standard is likely to be heavily influenced by the investment required by each system, the quality of development tools, and the additional cost per product.

For this project the fieldbus specified was Echelon LonWorks. Partech Instruments' decision to standardise on this system was made for the following reasons:

1. **Peer-to-peer communications**. Distributing control to sensors, actuators and other devices removes the central controller as a potential system bottleneck and central point of failure. If devices can communicate directly with one another, eliminating extra passes through a central controller, it allows a control loop to be closed in the shortest time period, with minimum bandwidth and minimum risk of error or other failure. Distributed systems also tend to offer better scalability, performance, and flexibility.

2. **Low cost implementation**. LonWorks networks are constructed from 'nodes', each containing a Neuron® chip[3], combining an 8-bit microcontroller running a real-time kernel with networking hardware and software, as well as transducer interface logic and drivers, integrated into a single compact IC, available at a cost of \$3-\$4. A Neuron IC can be connected directly to the sensors and outputs it supervises, and is all that is needed to create an intelligent device or node, capable of communicating with any other node on the network.

3. **Media independence**. LonWorks supports a wide range of different media types due to the LonTalk protocol including support for twisted pair, optical fibre, link-power, radio, infra-red, and intrinsically safe transceivers, operating at data rates from 5kbit/s to 1.25Mbit/s.

4. **Interoperability**. The term 'interoperability' is often confused with 'interchangibility'. The two are different, however. Interchangibility means devices are exactly the same in all respects and can be exchanged with one another. Interoperability means that devices can share data with one another and be integrated into a system without the development of custom tools or gateways. LonWorks achieves interoperability on two different levels. The first level is managed by the Neuron IC embedded within each node, ensuring that the physical, link, network, transport, session and presentation layers of the LonTalk protocol are implemented in a consistent manner across the entire control network. The second level is managed by the application code running on each node. The application code uses 'network variables', chosen from a list of standard network variable types[4] (SNVTs) e.g. temperature, pressure, humidity, to exchange information with other nodes. Thus a pressure sensor from one supplier is able to communicate with a valve from another supplier using the standard network variable type for pressure.

1.4. Echelon LonWorks

LonWorks is a set of protocols, inexpensive integrated circuits, interoperable modules, and software tools allowing a complete distributed control network to be implemented.

At the core of each node on the network is a Neuron IC - a sophisticated VLSI device designed specifically for low-cost local operating network (LON) applications[5]. The Neuron IC has an eleven pin I/O interface with integrated hardware and firmware for connecting directly to motors, valves, display drivers, A/D converters, pressure sensors, thermistors, switches, relays, triacs, tachometers, other microprocessors, modems, etc. The IC incorporates three processors, of which two interact with a communication subsystem to make the transfer of information from node to node in a distributed control system an automatic process.

Neuron IC feature summary

- Three 8-bit pipelined processors
  Selectable input clock rates: 625 kHz to 10MHz
- On-chip memory
  Up to 2Kbyte static RAM
  Up to 2Kbyte EEPROM
  10Kbyte ROM
- 11 programmable I/O pins
  29 selectable modes of operation
  Programmable pull-ups (IO4 - IO7)
  20mA current sink (IO0 - IO3)
- Two 16-bit timer/counters for frequency timer I/O
- Up to 15 software timers
- Sleep mode for reduced current consumption while retaining operating state

- Network communications port
  Single-ended mode
  Differential mode
  Selectable transmission rates: 0.6 kbit/s to 1.25 Mbit/s
  40mA current output for differentially driving twisted-pair networks
  Optional collision detect input

- Firmware
  LonTalk protocol conforming to 7-layer OSI reference model
  I/O drivers
  Event-driven task scheduler

- Service pin for remote identification and diagnostics

- Unique 48-bit internal Neuron ID

- Channel Capacity: 280 packets/s throughput sustained; 700 packets/s peak at 10MHz

Each Neuron IC contains a unique 48-bit ID number assigned during manufacturing, allowing it to be addressed individually. During installation, a 'service' pin on the IC may be connected to 0V via a switch, causing the node to automatically transmit its ID number across the network. Additionally, the service pin may also drive an LED to indicate the state of the node and whether or not it is functioning correctly.

Network variables

The application program can declare a special class of static objects called network variables, which may be designated as either *input* or *output*. Assignment of a value to an output network variable causes propagation of that value to all nodes declaring an input network variable that is 'connected' to that output network variable. For example, a node that contains a temperature sensing device could declare an output network variable which contains the current temperature sensed by the node. Every time the node measures a new value for temperature, it updates the output network variable. Another node or nodes needing to know the current temperature, such as a heating control node, can then declare an input network variable for current temperature. Whenever the heating control node wants to use the value of the current temperature, it simply refers to the input network variable which will always contain the last temperature measured by the sensing node. This entire process is handled by the Neuron IC firmware and therefore appears totally transparent to the application code.

At installation time, the output network variable on the sensing node is 'connected' to the input network variable on the controlling node. This process of 'binding' network variables from different nodes together is typically performed by a network management tool - either a dedicated hardware device, or a computer software package. The binding is physically implemented by sending network management messages containing the necessary addressing information to the nodes to be connected. Nodes may also update their own binding information for simple networks, without external network management devices. Tables containing binding information are in the Neuron IC's EEPROM, allowing for easy updating.

The declaration of network variables within a node's application code occurs at compile time. The binding of a node's output network variable to the input of another node occurs at a later time, either before, during or after the node is installed in the network. Thus, although the physical connections (e.g. twisted-pair wiring) between all nodes on the network are fixed during installation, the logical connections (network variables) may be altered at anytime by simply attaching a network management tool to the network and reconfiguring the software within individual

Another important implication of the data driven model is that a sensor or actuator need only be concerned with itself. Temperature sensors need only know that they send out temperature values, not where to. Similarly an actuator does not need to know where the variable it depends on comes from - only that any important changes will appear on the network. Thus the network is optimised for the transfer of small messages, unlike Ethernet which is optimised for occasional large files.

In addition to the automatic propagation of network variables when their values change, nodes can be continually polled if critical to operation. However polling does not make efficient use of the network as, with traditional scan-based systems, a processor will inevitably spend most of its time reading states which haven't changed since the last time they were read.

Chapter 2

Theory

2.1. Light scattering by suspensions

Suspended solids, turbidity, and sludge density can all be inferred from the interaction of light with the particles in a sample suspension. When light passing through water meets suspended particles, the light is said to be scattered by the particles with some light being scattered in all directions (Figure 2.1.). This process is repeated many times at a microscopic or submicroscopic level to produce an overall pattern of light throughout the water sample. The same effect can be observed with other wave transmission situations such as surface waves moving across water. As soon as the wavefront reaches a boundary where there is a change in transmission characteristics it changes direction and may be split into two or more waves.

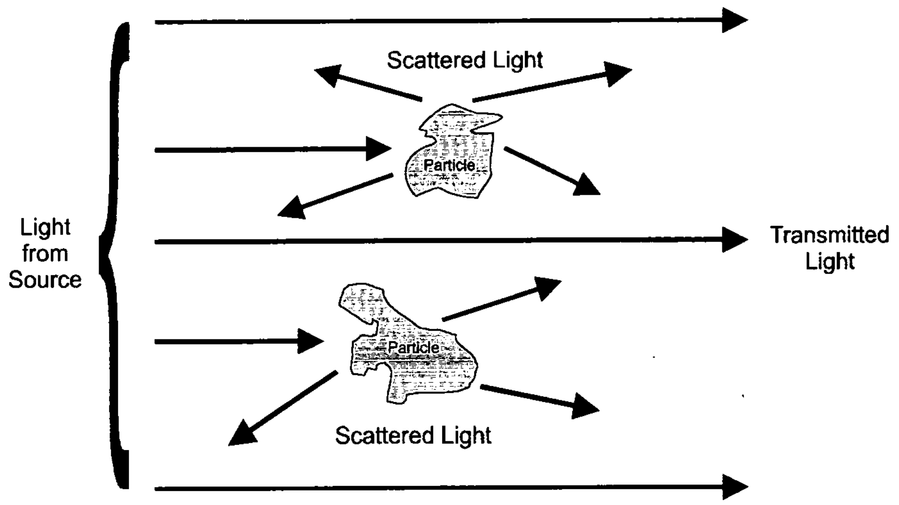


Figure 2.1. The light scattering effect of solid particles in suspension.

The way in which suspensions of particles scatter light is governed by the size of particles, their shape, their refractive index and the wavelength of the incident light. The theory of light scattering by regular-shaped objects such as spheres or cylinders was investigated towards the end of the 19th Century culminating in Mie's theoretical work in 1908. A good presentation of the theory is given by Van de Hulst[6]. For water applications the main factor governing the scattering behaviour is the ratio R of the wavelength of the incident light λ to particle size P. The manner of light scattering in suspensions can be divided into three main types depending on this ratio

$$R = \frac{P}{\lambda}$$

R ≤ 0.05. 'Small' particles.

Scattering in this region is known as Rayleigh scattering. Figure 2.2. shows the way in which the light intensity varies with scattering angle for a suspension of such particles. It may be seen that there is no strong bias in the intensity at different scattering angles.

Another feature of Rayleigh scattering is that the intensity is inversely proportional to the fourth power of the wavelength of the light. Hence for white light the blue shorter wavelength part of the visible spectrum is scattered much more efficiently than the red longer wavelength end of the visible spectrum. Everyday examples of Rayleigh scattering are smoke particles and the sky itself, both of which have the characteristic blue of this type of scattering. In water samples, fine colloidal material shows Rayleigh scattering.

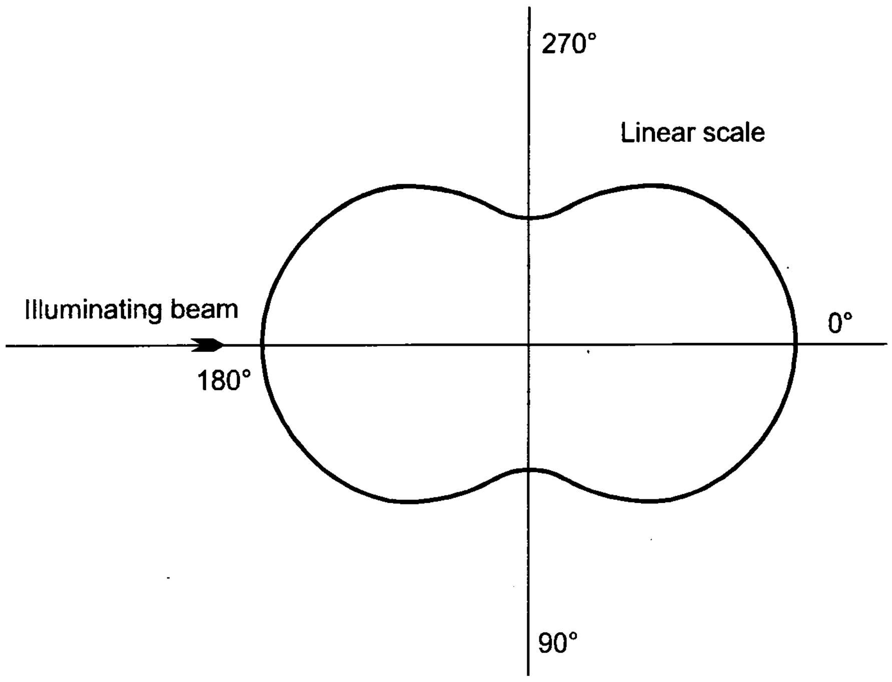


Figure 2.2. The angular distribution of Rayleigh scattered light.

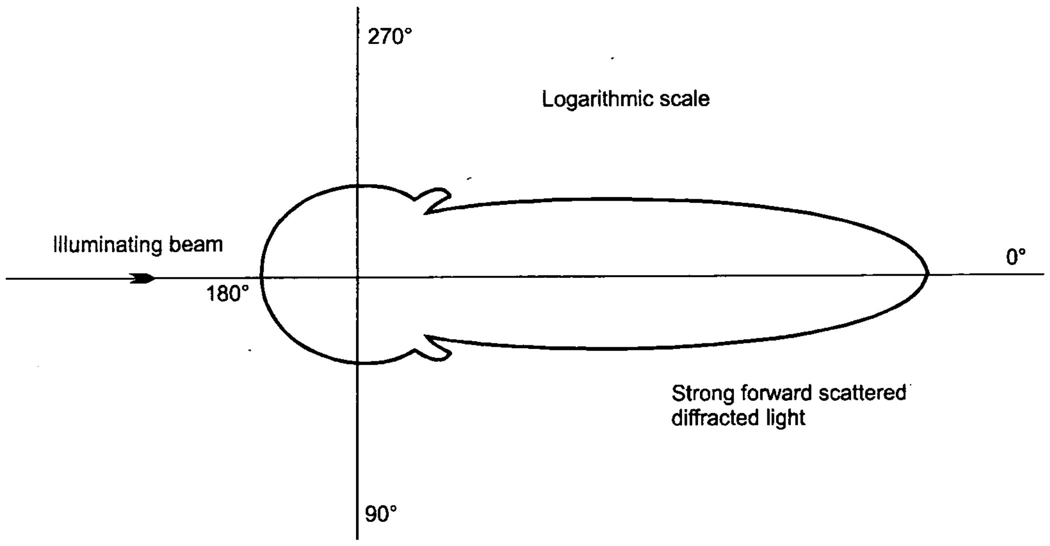


Figure 2.3. The angular distribution of light scattered by 'large' particles.

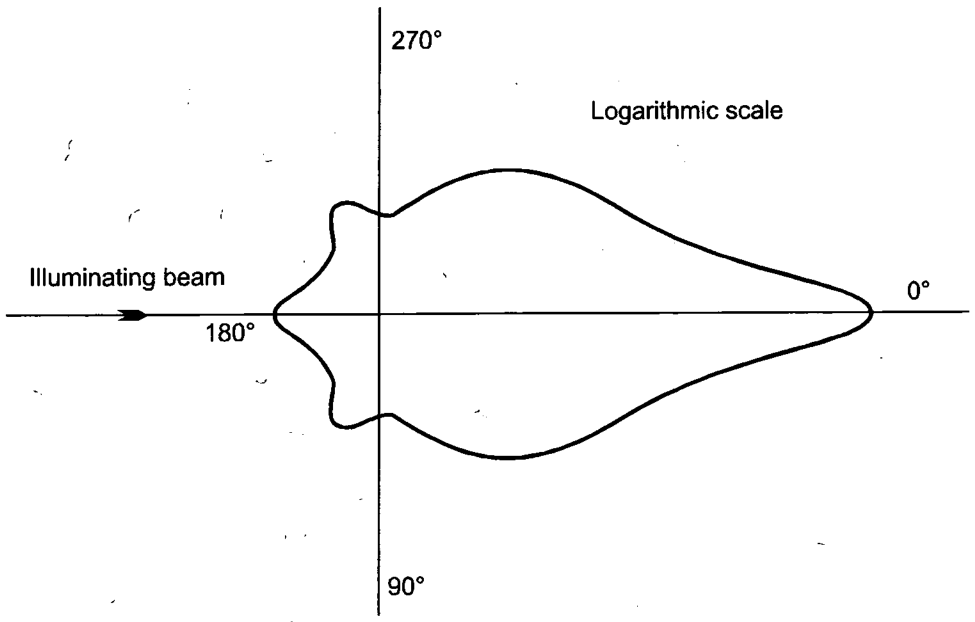


Figure 2.4. An example of the angular distribution of scattered light from particles of a few microns in size.

R ≥ 20. 'Large' particles.

The light scattering pattern is composed of two components. The first is due to reflection and refraction by the individual particles and produces a general illumination around the sample with no strong peaks. The intensity of this illumination depends on the surface qualities of the particles and increases in proportion to the surface area of the particles in the suspension. The second is a diffraction pattern centred on the 0° direction which is generally much stronger than the reflected and refracted light.

As the particle size increases to many times the wavelength of the incident light, the diffraction pattern moves closer to the 0° direction. The combined pattern is shown in Figure 2.3.

All sizes of particles which are covered by a general theory due to Mie.

In the previous cases Mie's light scattering theory reduces to the simplified situations described. For particle sizes between these two extreme cases, Mie's theory is required and the intensity pattern has to be worked out for the size, shape, and refractive index of the particles in a suspension. In general, the patterns are complicated and sometimes contain a series of quite sharp peaks, though in real situations the range of particle sizes and the spread of the illuminating wavelengths would cause the peaks to be less marked. An example of a pattern is shown in figure 2.4.

2.2. Light intensity and particle size

Section 2.1. has shown how the shape of light scattering patterns varies depending upon the relationship of the particle size to the wavelength of the incident light. The efficiency with which particles of the same kind scatter light also varies with the particle size and wavelength. When viewed as light scatter from one particle, the rule is simple, the smaller the particle the less light it scatters. However, this is not the most useful way to view the effects of size. A better way to look at the dependence of light scattering on particle size is shown in figure 2.5. where the three light scattering types are labelled.

The y-axis is the relative scattered light intensity for a suspension of fixed suspended solids with all the suspended solids particles of the same size, and the curve shows how this scattered light intensity would vary as the particle size changed keeping the suspended solids constant. The precise form of this curve will vary depending on the

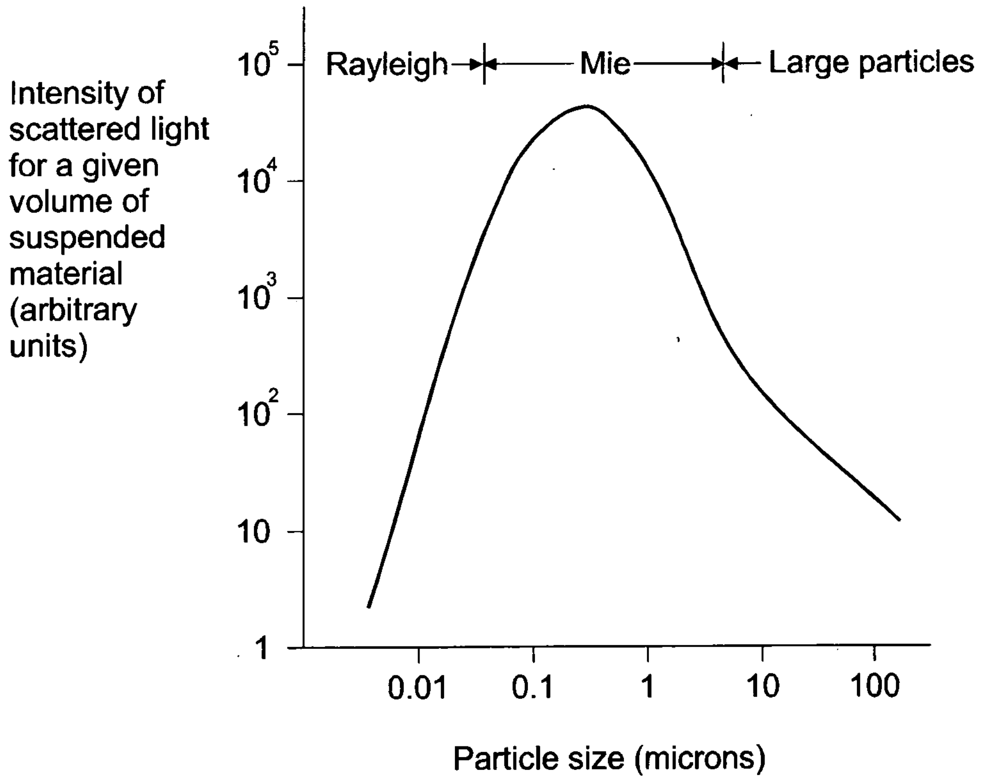


Figure 2.5. The dependence of scattered light intensity on particle size.

particles in the suspension and the nature of the sensing principle employed. However, the trend of a peak scattering efficiency at around half a micron with a rapid fall off for suspensions of larger or smaller particles generally holds true for most samples. Particle size therefore has a strong influence on the output of water quality sensors. Real samples usually contain a wide range of particle sizes, not just one size. The measured scattered light is made up of contributions from particles across a large part of the curve of figure 2.5.

For turbidity and suspended solids measurement in water applications, the most appropriate unit for particle size is the micron (1 micron = 1μm = 10^{-6} metres). Figure 2.6. illustrates the sizes of a range of objects in microns alongside the light scattering regimes described in section 2.1. It may be seen that the range of particle sizes of greatest interest for water applications is from around 0.1 microns up to 1000 micros (1mm) with the majority of this range being of the 'large' particle scattering type. Particles of the Rayleigh scattering type are not usually present in sufficient concentration to contribute significantly to the turbidity and are considered to be colour. In the UK, light absorption by particles below 0.45 microns in size is arbitrarily defined as part of the sample colour. This is a convenient distinction based on available filtration media though not on any physical or chemical change at this size.

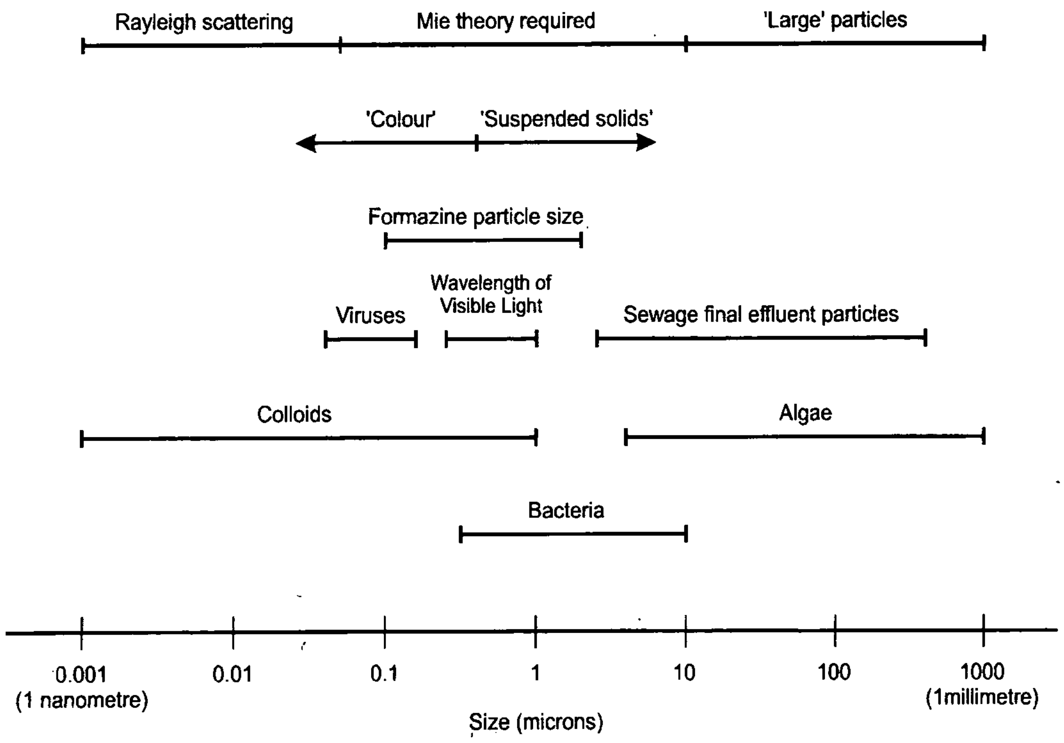


Figure 2.6. Sizes of various items in microns.

2.3. Observations from light scattering theory

The theory of light scattering by suspensions has a number of implications for turbidity measurement:

1. The greatest sensitivity to large particles is obtained by measuring scattered light at or close to 0°. This applies particularly to samples such as mineral particles in distribution, most sewage treatment samples, bacteria-sized and larger particles in water treatment.

2. Sensitivity to particles of less than about 0.2 microns in size is not strongly angle-dependent. If the turbidity is measured at or close to 0° on a sample containing larger particles, the contribution to the measured turbidity from the sub-micron sized particles will be small.

3. The ratio of 90° scatter to forward scatter intensity gives an indication of the proportion of large particles to sub-micron particles. This technique is used in the brewing industry to indicate the range of particle sizes causing the turbidity[7] but has not yet been applied in the water industry.

4. Changes in the particle size distribution of a sample will change the turbidity. This has important implications for sample stability. If a sample for analysis is allowed to flocculate the turbidity is likely to fall as the sample now contains a greater proportion of larger particles which scatter light less efficiently.

5. The amount of light measured by a sensor depends on the source light, the particle sizes, the nature of the particles in the suspension and the range of angles

scattering into the detector. Unless all these parameters are the same, two sensors will behave differently. In water applications, instruments are generally calibrated on formazine suspensions. The sensors are then used on a range of suspensions all of which scatter light in a different way from formazine. Hence the outputs of the sensors, correctly calibrated on formazine, are usually different when measuring other suspensions, with a recent study[8] showing differences of up to 50%.

2.4. Measurement methods

Practical sensor designs utilise a variety of measurement methods, generally based on the two basic types introduced in section 1.2., light scatter and transmitted light.

90° scatter

The most widely used optical arrangement amongst the scattered light sensors is the 90° design illustrated in figure 1.2. Such designs have a good sensitivity to a very wide range of particle sizes from colloidal material through to large particles in raw waters and the large mineral particles sometimes found in samples.

One of the basic problems in sensor design is to minimise 'stray light' Stray light is the general level of illumination due to multiple reflections within the instrument which can enter the scattered light detector and cause an increased zero reading. This effect can be removed electronically simply by applying a negative zero offset, but the most stable zero readings will be obtained by minimising the stray light and the size of the zero offset that must be applied. In 90° sensors the most intense light beam, which is the main transmitted beam, will be at 90° to the detector axis, and it is a relatively simple matter to keep the stray light to manageable levels. Another basic advantage of

90° types for the designer is that the light intensity at 90° varies quite slowly with changing angle as may be seen from figures 2.2., 2.3., and 2.4. This means that it will be possible to obtain good stability from the sensor even if there are small changes in alignment of the optical components.

Forward scatter

A popular configuration for sensor built as pipe sections is the forward scatter type sometimes known as 'low angle scatter'. The geometry of this type is shown in figure 2.7. It may be seen that it is optically more complicated than a 90° system. The main reason for the extra optical components is that the small amount of scattered light has to be detected without allowing the main beam of transmitted light to reach the detector. Forward scatter sensors have greater sensitivity to particles above about 0.5 microns in size, making them useful for applications where the larger particles are the ones of significance in the sample.

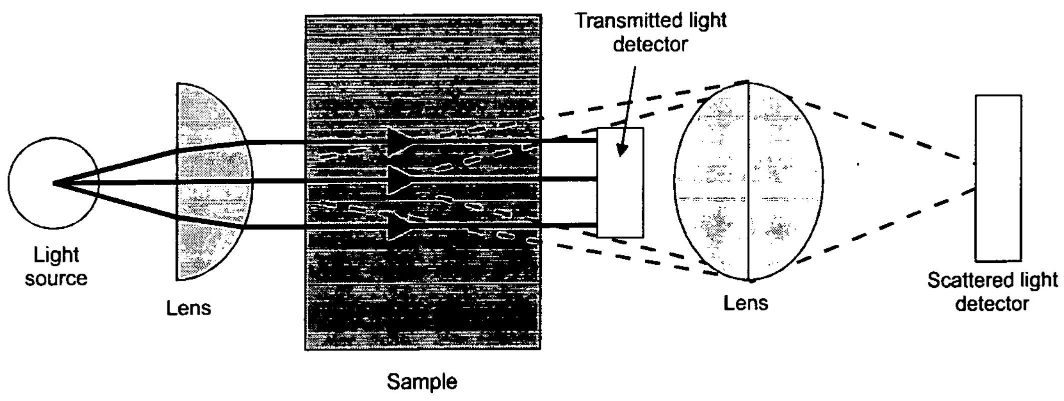


Figure 2.7. A forward scatter sensor in schematic form.

As may be seen from figure 2.4., the light intensity of a scattered light pattern varies rapidly close to the transmitted light beam and this requires a forward scatter sensor to be constructed from rigid stable materials otherwise poor stability will result.

Other scatter angles

Other angles of scatter commonly used include 25°, 135°, and 180°. The 25° type can be viewed as a compromise between the 90° and forward angle geometries.

The 180° type is well suited to instruments engineered as insertion probes, as the emitter and detector may be mounted side by side within a small space. Its sensitivity to different sizes of particle is similar to that of the 90° geometry. Fibre optics are often used in the construction of these probes. An advantage of this geometry is that the main light beam does not need to penetrate far into the sample to obtain a reading. This allows these probes to be used on high solids samples such as sludges and slurries.

Ratio sensors

It is quite common to employ a second detector to monitor the transmitted light as shown in figure 2.8. The electronics will then use the ratio of the scattered light to the transmitted light as the measure of turbidity thus correcting any changes in the light source and giving some compensation for colour effects. The scattered light and transmitted light may both be attenuated by colour in the sample, but if the light paths are of approximately equal length, any attenuation due to colour should divide out of the ratio.

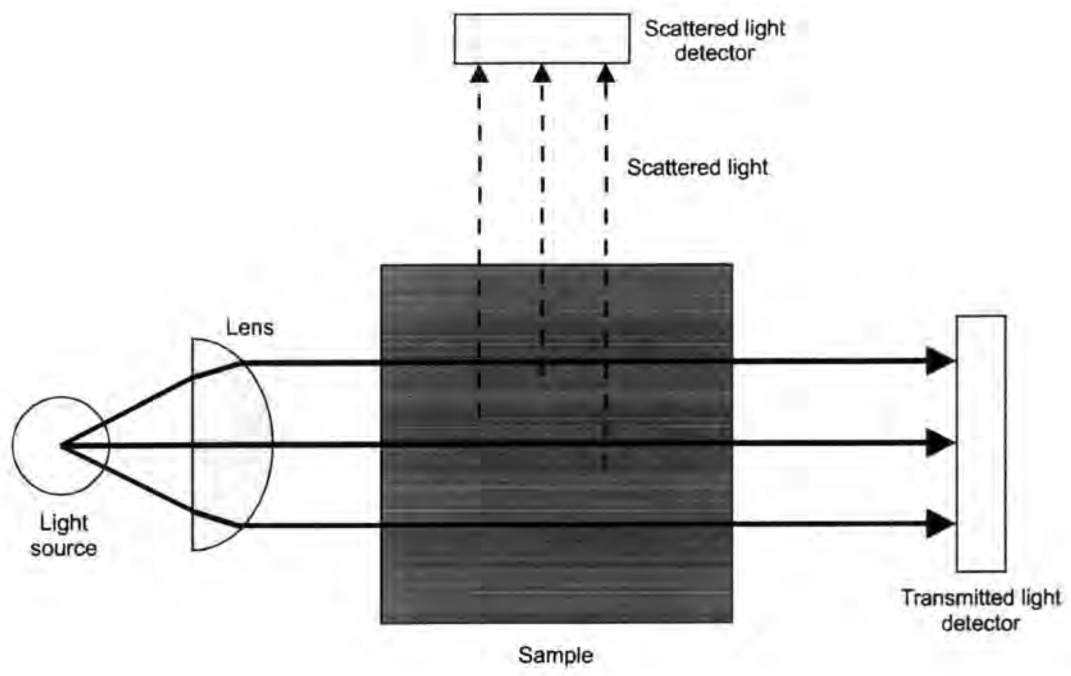


Figure 2.8. A ratio sensor design using 90° scatter.

As the concentration of the sample increases, increasing amounts of scattered light are taken out of the transmitted light beam, so that the simple ratio of scattered light to transmitted light can give poor linearity when used over a large turbidity range.

Transmitted light sensors

As a beam of light progresses through a suspension it is weakened as parts of the beam are removed by scattering. An alternative to measuring the intensity of the scattered light is to measure the loss of light in the transmitted beam. Sensors which operate on this principle generally assume a logarithmic relationship between the transmitted light and the solids concentration (see figure 1.5.) in the form of Beer's law (also known as Beer-Lambert, Lambert-Bouger law). The law[9], which was originally applied to liquids containing dissolved material, states that 'successive

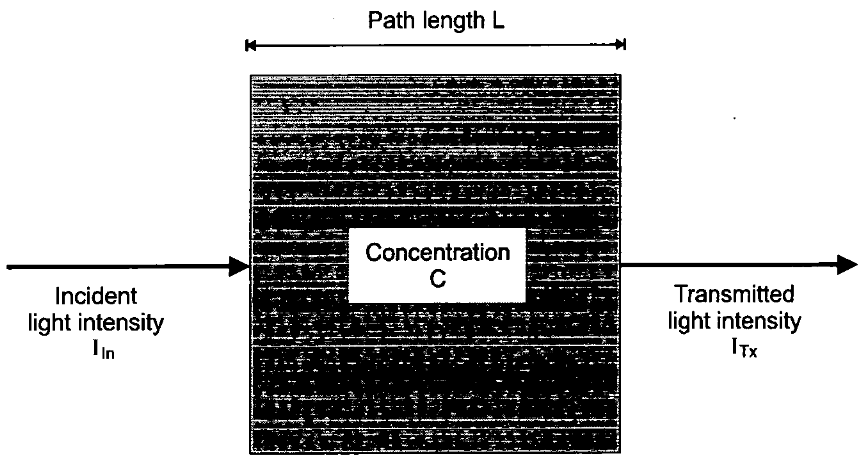


Figure 2.9. Beer's law is used to model the absorption of light by suspended material and is the basis for transmitted light sensors.

increments in the number of identical absorbing molecules in a given path length of monochromatic (i.e. a narrow range of light wavelengths) radiation absorb equal fractions of the intensity falling upon it' as illustrated in figure 2.9. From Beer's law, the reduction in light intensity for a concentration C and path length L is given by:

$$\log \frac{I_{In}}{I_{Tx}} \propto C \times L$$

A sensor configuration which uses transmitted light is shown in figure 1.4. Some of the scattered light may reach the detector depending on the precise geometry of the optical arrangement.

A potential disadvantage of transmitted light is that the sample colour interferes with the measurement process, giving rise to a positive bias signal in the reading. The measured value is increased by the absorbance due to the sample colour.

2.5. Light sources

There are two main light source types currently in use for optical water quality sensors: tungsten filament lamps (quartz halogen or inert as filled) and light emitting diodes (LEDs). Filament lamps normally emit a wide range of light wavelengths from the blue end of the visible spectrum to the infrared and commonly consume 10-20 watts of power. Sometimes a filter is used to remove the infrared component. LEDs produce a narrow range of wavelengths, typically 50 nanometres wide at the half-power points. Near infrared LEDs are a common choice, emitting light at about 880 nanometres, with a radiated power of a few milliwatts. Water absorbs infrared radiation very strongly beyond about 1 micron (1000 nanometres) and so wavelengths beyond 880 nanometres such as 950 nanometres (a common LED emission wavelength) will begin to be absorbed.

In general, the best sample penetration is achieved using near infrared LED light sources. This is due to colloidal material in the sample which scatters the blue light in a white light source much more effectively than the red and infrared. Multiple scattering non-linearity effects will therefore appear at lower solids concentrations with a white light source than with near infrared. There is also an improvement in sensitivity to larger particles at near infrared wavelengths. For use at low solids concentrations white light sources can give good results since, despite their lower efficiency as light radiators compared with LEDs, they can emit more total power as light than a normal-sized LED. While the total power output is useful in raising sensitivity, it can also cause problems in some designs due its promotion of photosynthesis and hence the growth of biological fouling.

2.6. The LonTalk™ Protocol

The Neuron IC implements a complete networking protocol, called LonTalk[10], using two of the three on-chip processors (Processor-1 and Processor-2). This networking protocol follows the ISO OSI reference model for network protocols; it allows application code running on Processor-3 to communicate with applications running on other Neuron IC-based nodes elsewhere on the same network. The following table shows the mapping of LonTalk services onto the seven-layer OSI reference model:

OSI Layer		Purpose	Services Provided	Processor
7	Application	Application compatibility	Standard network variable types	Application
6	Presentation	Data Interpretation	Network variables, foreign frame transmission	Network
5	Session	Remote actions	Request/response, authentication, network management	Network
4	Transport	End-to-end reliability	Acknowledged and unacknowledged, unicast and multicast, authentication, common ordering, duplicate detection	Network
3	Network	Destination addressing	Addressing, routers	Network
2	Link	Media access and framing	Framing, data encoding, CRC error checking, predictive CSMA, collision avoidance, priority, collision detection	MAC
1	Physical	Electrical Interconnect	Media-specific interfaces and modulation schemes	MAC, XCVR

The main features of the LonTalk protocol are:

1. Multiple media support

The protocol processing on the Neuron IC is media-independent. This allows nodes to support a wide variety of communications media, including twisted-pair, powerline, radio-frequency, infra-red, coaxial cable, and fibre optics.

2. Support for multiple communication channels

A channel is a physical transport medium for packets, with a network being composed of one or more channels. In order for packets to be transferred from one channel to another, a device called a 'router' is used to connect the channels. The protocol includes this feature to allow the construction of multiple media networks, so that network loading can be optimised by localising traffic.

3. Communications rates

Channels may be configured for different bit rates to allow trade-offs of distance, throughput, and power consumption. Channel throughput depends on bit rate (0.6 - 1250 kbit/s), oscillator frequencies and accuracy, transceiver characteristics, average size of packet, and the use of acknowledgements, priority and authentication.

4. Message services

The LonTalk protocol offers four basic types of message service:

Acknowledged end-to-end, where a message is sent to a node or group of nodes, and individual acknowledgements are expected from each receiver. If the acknowledgements are not all received, the sender times out and retries the transaction.

Request / response, where a message is sent to a node or group of nodes, and individual responses are expected from each receiver. The incoming message is processed by the application on the receiving side before a response is generated. The same retry and time-out options are available as above.

Unacknowledged repeated, where a message is sent to a node or group of nodes multiple times, and no response is expected. This method avoids network overloads caused by many node responses.

Unacknowledged, where a message is sent once and no response is expected. When using this service, the application must not be sensitive to the loss of a message.

5. Authentication

Authenticated messages allow the receivers of a message to determine if the sender is authorised to send that message. Authentication prevents unauthorised access to nodes and is implemented by distributing 48-bit keys to the nodes at installation time.

6. Priority

The protocol optionally offers a priority mechanism to improve the response time of critical packets. The user may specify priority time slots on a channel dedicated to priority nodes. Each priority time slot on a channel adds time to the transmission of every message, but now dedicated bandwidth is available at the end of each packet for priority access without any contention for the channel.

7. Collision avoidance

The LonTalk protocol uses a unique collision avoidance algorithm which has the property that under conditions of overload, the channel can still carry its maximum capacity, rather than have its throughput degrade due to excess collisions.

8. Collision detection

On communications media that support hardware collision detection (e.g. twisted pair), the LonTalk protocol can optionally cancel transmission of a packet as soon as a collision is detected by the transceiver. This allows the node to immediately retransmit any packet that has been damaged by a collision.

2.6.1. The LonTalk MAC sublayer

The LonTalk media access control (MAC) sublayer gives rise to many of the protocol's advantages in control applications - multiple-media communication, low data rates, sustained performance during heavy loads, large networks - and, because of its unique properties, is worth considering in detail.

The MAC sublayer is part of the Data Layer of the OSI Reference Model. Many different MAC algorithms exist in networks today, but the algorithm used by the LonTalk protocol belongs to the CSMA (Carrier Sense Multiple Access) family. The CSMA family of MAC algorithms requires a node to establish that the media is idle before it begins to transmit. However, each algorithm behaves differently once the idle state is detected. This results in very different network performance results under conditions of heavy data traffic.

Some CSMA algorithms use discrete time intervals called slots, or randomising slots, to access the medium. By limiting access to the medium by a given node to specific time slots, slotted media access greatly reduces the probability of two packets colliding. Slotted media access is used by p-persistent CSMA, including the LonTalk protocol.

The LonTalk protocol uses a new CSMA MAC algorithm called *predictive p-persistent* CSMA[11]. This retains the benefits of CSMA but overcomes its shortcomings for control applications. As in p-persistent CSMA, all LonWorks nodes randomise their access to the medium. This avoids the otherwise inevitable collision that results when two or more nodes are waiting for the network to go idle so that they can send a packet. If they wait for the same duration after backoff and before retry, repeated collisions will result. Randomising the access delay reduces collisions. In the LonTalk protocol, nodes randomise over a minimum of 16 different levels of delay called randomising slots. Thus the average delay in an idle network is eight slot widths.

In p-persistent CSMA when a node has a message to send, it does so in a given randomising slot with probability *p*. However, the LonTalk protocol carries the added improvement that *p* is dynamically adjusted based upon network load. When the network is idle, all nodes randomise over only 16 slots. When the estimated network load increases, nodes may randomise a large number of slots. The number of slots increases by a factor of *n*, where the range of *n* is from 1 to 63. Echelon calls *n* the estimated channel backlog, and represents the number of nodes with a packet to send during the next packet cycle.

This method of estimating the backlog and dynamically adjusting the media access allows the LonTalk protocol to have just a few randomising slots during periods of light load, while having the benefit of many randomising slots during periods of heavy load. Thus, media access delays are minimised during periods of light load, and collisions are minimised during periods of heavy load.

As noted earlier, the ability to adjust the number of randomising slots depends on the ability to estimate the channel backlog. In the LonTalk protocol, a transmitting node includes information in the packet on the number of acknowledgements expected as a result of sending that packet. All the nodes that receive the packet increment the estimated channel backlog by that amount. Likewise, the estimated channel backlog is decremented by 1 at the end of each packet cycle. The estimated channel backlog is never decremented below 1. Since LonTalk packets are typically acknowledged, 50% or more of the channel backlog is predictable at any time.

Chapter 3

Experimental Details

3.1. Experimental aims

The main aim of the experimental work of this project was to construct a working prototype fieldbus-based turbidity/suspended solids monitoring system. Partech Instruments required a working system to be developed which operated over a LonWorks network. In order to be able to demonstrate a typical working system, the sensor's output should be able to control at least two relay setpoints, each triggered at a different value. This is a common requirement of many industrial applications where the measured turbidity/suspended solids value must stay within a predefined range to ensure the process operates efficiently. Thus with two relay setpoints, if the measured value exceeds a high threshold or drops below a low threshold, a relay contact will close to alert an operator - or automatically control other equipment such as a pump or valve.

While such an arrangement constitutes a working monitoring system, most applications also require the measured value to be displayed numerically with appropriate units. Additionally, the operator must be able to easily calibrate the sensor in situ and to also adjust the setpoint values. This function could be provided by a computer or other dedicated node monitoring device, and need not be permanently attached to the control network. Any equipment used in the field must have an IP (Industrial Protection) rating of at least 65 to guarantee its continued operation in a dusty or wet environment.

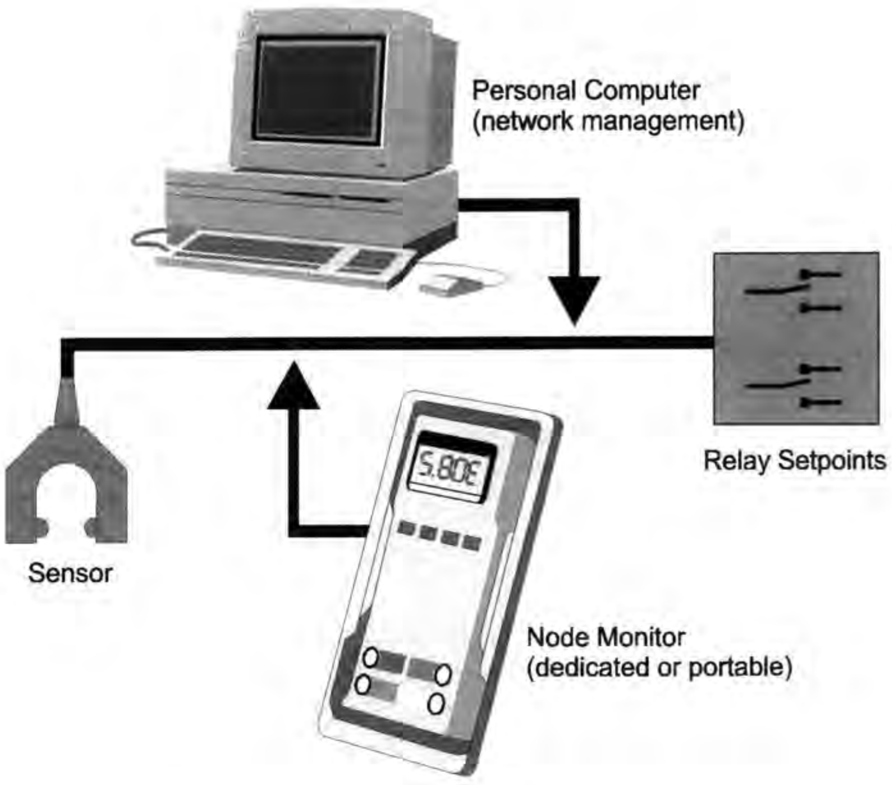


Figure 3.1. A schematic representation of the control network to be designed and constructed.

A secondary aim of the experimental work was to investigate suitable tools for installing and managing LonWorks control networks. Such tools should be able to bind network variables to one another, configure nodes, download new node application software, and generally provide the user with all network information required. Ideally, the user should be able to carry out these operations on a graphical representation of the network to make the process intuitive, rather than dealing with lists of textual or numerical information. It is not intended that a custom software application be created for water monitoring systems, rather that the suitability of existing commercial software packages for this purpose be evaluated.

3.2. Experimental methods

A wide range of different techniques and methods have been used in the course of the experimental work for this project. The five main areas to consider are sensor tests, mechanical design, electronic circuit and PCB design, embedded software programming, and PC Windows programming. All of these areas involved a variety of software tools running under Windows 95 on a Pentium-based PC with 16MB of RAM and 1GB of hard disk space.

Sensor tests

As the major aim of the project was to develop a working sensor, it is important to have a reliable, accurate, and consistent method of assessing the performance of sensor prototypes. Figure 3.2 depicts the experimental arrangement used for all sensor tests conducted in this project. The sensor to be tested is positioned in a suitable container, with its sensing area located as centrally as possible, before being clamped in place. The container rests on a magnetic stirrer which drives a small paddle in the bottom of the container to ensure that the sample being monitored by the sensor is not allowed to settle. For samples of high concentrations, an additional motor-driven stirrer, located within the container itself, was also used.

Applications for turbidity/suspended solids sensors involve a wide range of different samples to be monitored, each containing a variety of particles with different sizes, densities, surface properties and optical characteristics. In order to allow the performance of different sensors to be compared with one another, several standards have been established. The two standards used for this project are based upon

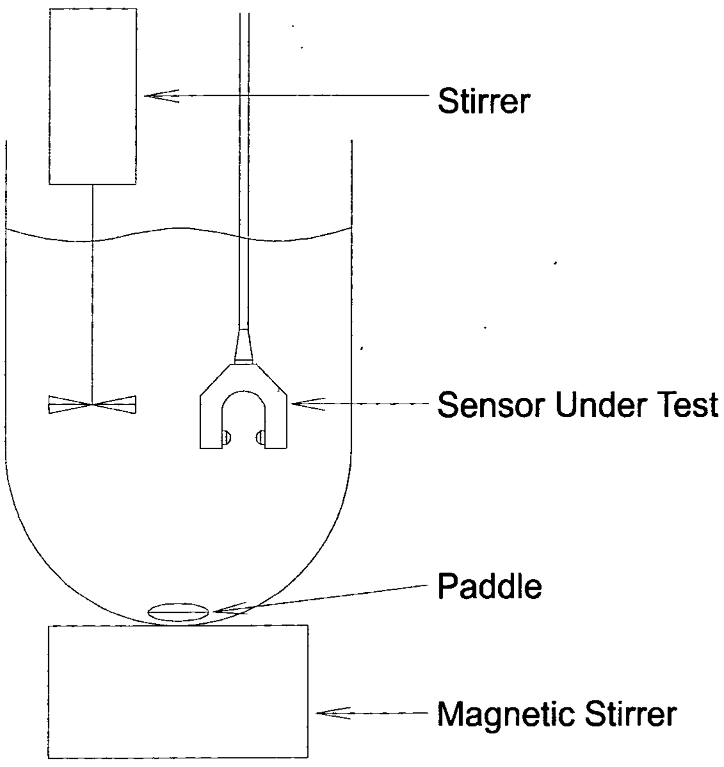


Figure 3.2. The experimental arrangement used for testing the performance of prototype sensors.

formazine and fuller's earth - materials traditionally chosen for their reproducibility, stability, and the wide range of solutions that they can create.

To prepare formazine, 10.00g of Hydrazinium Sulphate is dissolved in distilled water and diluted to 1000ml in a volumetric flask. Next, 100.00g of Hexamethylenetetramine is also dissolved in distilled water and diluted to 1000ml in a separate volumetric flask. These two solutions are thoroughly mixed in a large bottle and allowed to stand undisturbed for 24 hours at 25 degrees C. The resultant suspension has a turbidity of 4000 FTUs (Formazine Turbidity Units) and an appearance similar to that of milk. From this stock solution, suspensions of other FTU values can easily be created through further dilution. For a given volume of 4000 FTU formazine solution, V_f, diluted into volume V_w of distilled water, a suspension of A FTUs will be created such that:

$$A = 4000.\left(\frac{Vw}{Vf} + 1\right)^{-1}$$

In a typical sensor formazine test, the container is initially filled with 4 litres of distilled water and 4000 FTU formazine added by syringe in 1ml increments, increasing to 5ml and 10ml later. Although it is important to maintain gentle stirring of formazine solutions, too vigorous stirring can result in flocculation.

For fuller's earth sensor tests, a large quantity of fuller's earth (a fine greyish powder) is placed on a precision balance and the mass noted. The container is again initially filled with 4 litres of distilled water and small spatula-sized amounts of fuller's earth are incrementally transferred from the balance into the solution. After each increment the remaining mass on the balance is noted so that, at the end of the test, the masses can be subtracted to establish the actual amount in suspension after each increment. This method is used to reduce the systematic error that would result from repeatedly measuring small masses of fuller's earth. Generally, the biggest problem in fuller's earth tests is keeping the solution in suspension. For high concentrations, the resulting solution has a sludge-like consistency that requires vigorous stirring to prevent particles settling on the bottom of the container.

Throughout all sensor tests, care was taken to ensure that the ambient light level and ambient temperature remained constant.

Mechanical design

In order to develop the sensor, several different optical arrangements were tested. For each arrangement, a mechanical assembly was required to hold the optical and electronic components precisely in position while still allowing adjustments to be made. Each mechanical design was initially drawn using the Turbocad computer-aided design software package, and then produced within the engineering department.

Electronic circuit and PCB design

Given the nature of the project, electronic design formed a major part of the practical work undertaken. Various ideas were tested on breadboard and were left in this state during initial testing. Once the correct operation of a circuit had been verified, the schematic was entered into the Protel Advanced Schematic software package - a time consuming process due to the need to first enter many of the components used into the software's component library. The next step was to export the netlist of connections into the sister software package, Protel Advanced PCB. From here, the exact outline of the PCB would be entered, each component carefully placed within this outline, and tracks manually routed in accordance with the netlist. Once the design was complete, files would be generated for the top and bottom copper layers, along with a list of locations for each hole to be drilled. From these files, the final PCB could be produced by an external facility.

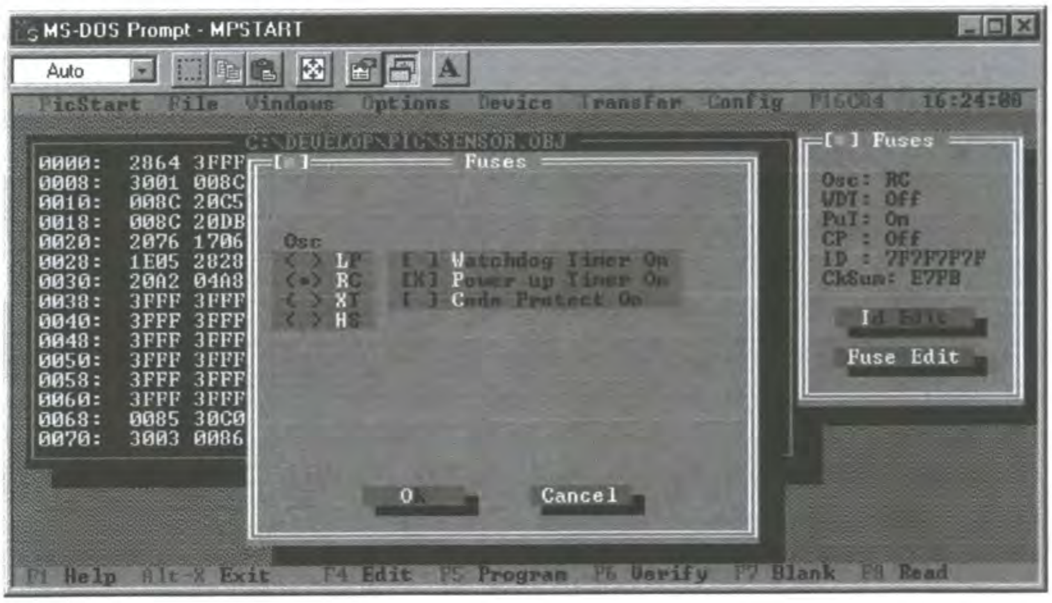
Each of the PCBs produced for this project were double-sided without solder resist or component identity layers. Once populated with components and tested, the PCB assembly was mounted in a suitable enclosure to improve its robustness and visual appeal.

Embedded software programming

Due to the project's emphasis of designing self-contained 'intelligent' devices, a large amount of embedded programming was needed. The three microcontrollers used in the project were the Philips 80C552, Arizona Microchip's PIC16C84 and Echelon's MC143150 Neuron IC (produced by Motorola) - each requiring their own specialist programming tools.

During the early stages of the project, a commercially-available microprocessor board containing a 80C552 8-bit microcontroller, 64K EPROM, 32K RAM, real-time clock, and RS-232 interface was used to log results from, and control, the first sensor prototype. The software was written in C, compiled on a PC using the Keil C51 compiler, and downloaded to an EPROM emulator tool connected to the PC's parallel port.

In the final sensor design, the PIC microcontroller proved to be an ideal companion to the Neuron IC, handling the faster real-time requirements of the system. The software for the PIC was coded in its native assembly language due to the limited 1K of EEPROM code space and requirements for speed. The source file was then passed to Arizona Microchip's own assembler, and the resulting object file loaded into the MSDOS-based program illustrated in figure 3.3.


MS-DOS Prompt - MPSTART
Auto
PicStart File Windows Options Device Transfer Config P16C84 16:24:08
Id Edit
Fuse Edit
Ok
Cancel
F1 Help Alt-X Exit F4 Edit F5 Program F6 Verify F7 Blank F8 Read

```
C:\DEVELOP\PIC\SENSOR.OBJ
0000:  2864 3FFF
0008:  3001 008C
0010:  008C 20C5
0018:  008C 20DB
0020:  2076 1706
0028:  1E05 2828
0030:  20A2 04A8
0038:  3FFF 3FFF
0040:  3FFF 3FFF
0048:  3FFF 3FFF
0050:  3FFF 3FFF
0058:  3FFF 3FFF
0060:  3FFF 3FFF
0068:  0085 30C0
0070:  3003 0086
```

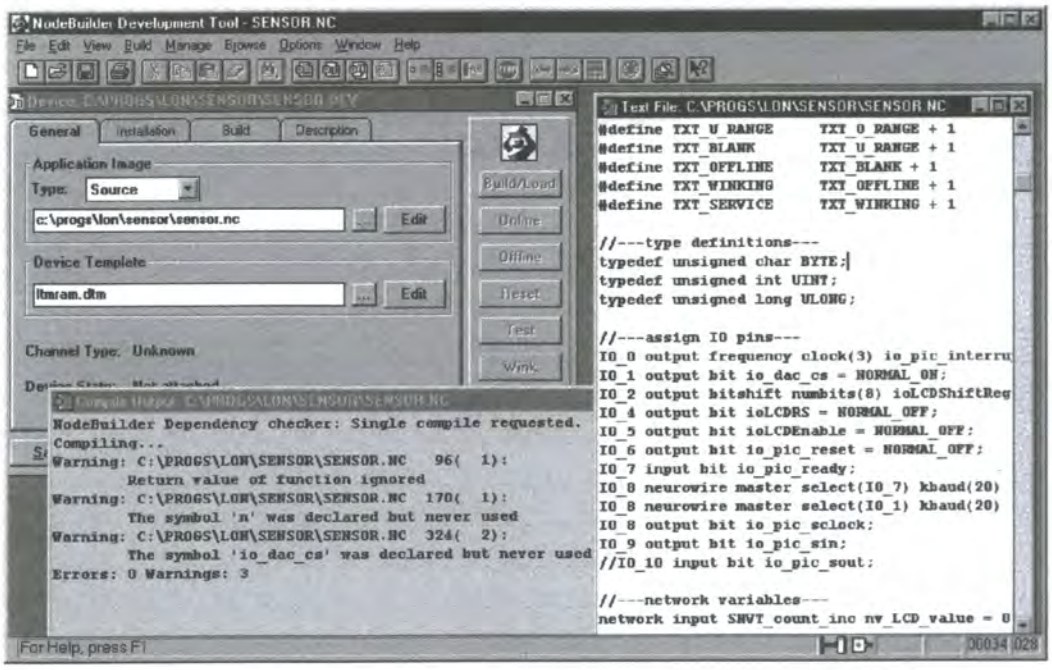
```
Fuses
Osc
< > LP     [ ] Watchdog Timer On
<•> RC     [X] Power up Timer On
< > XT     [ ] Code Protect On
< > HS
```

```
[■] Fuses
Osc: RC
WDT: Off
PuT: On
CP : Off
ID : 7F7F7F7F
CkSum: E7FB
```

Figure 3.3. The MSDOS software program used to program the PIC processor object code into the device and configure its fuses.

This ‘MPSTART’ program communicates with the dedicated PIC programmer via RS-232 and allows object code to be read or written to/from a wide variety of PIC processors. Additionally, memory locations may be individually edited as can the important configuration fuses that determine crucial aspects of the processor’s behaviour. The main drawback to this development system is the need to constantly transfer the PIC IC between the target hardware and the programmer.

The final, and most important. processor used for embedded software was the Neuron IC. The NodeBuilder™ development package supplied by Echelon consists of a Microsoft Windows-based software package (figure 3.4.), a PC expansion card to connect to a LonWorks network via a variety of transceivers, and a target development node.


NodeBuilder Development Tool - SENSOR.NC
File Edit View Build Manage Browse Options Window Help
Device: C:\PROGS\LON\SENSOR\SENSOR.DEV
General
Installation
Build
Description
Application Image
Type: Source
c:\progs\lon\sensor\sensor.nc
Edit
Device Template
ltmram.dtm
Edit
Channel Type: Unknown
Build/Load
Online
Offline
Reset
Test
Wink
Compile Output: C:\PROGS\LON\SENSOR\SENSOR.NC
Text File: C:\PROGS\LON\SENSOR\SENSOR.NC
For Help, press F1
00034 028

```
NodeBuilder Dependency checker: Single compile requested.
Compiling...
Warning: C:\PROGS\LON\SENSOR\SENSOR.NC   96(  1):
        Return value of function ignored
Warning: C:\PROGS\LON\SENSOR\SENSOR.NC  170(  1):
        The symbol 'n' was declared but never used
Warning: C:\PROGS\LON\SENSOR\SENSOR.NC  324(  2):
        The symbol 'io_dac_cs' was declared but never used
Errors: 0 Warnings: 3
```

```
#define TXT_U_RANGE       TXT_0_RANGE + 1
#define TXT_BLANK         TXT_U_RANGE + 1
#define TXT_OFFLINE       TXT_BLANK + 1
#define TXT_WINKING       TXT_OFFLINE + 1
#define TXT_SERVICE       TXT_WINKING + 1

//---type definitions---
typedef unsigned char BYTE;
typedef unsigned int UINT;
typedef unsigned long ULONG;

//---assign IO pins---
IO_0 output frequency clock(3) io_pic_interru
IO_1 output bit io_dac_cs = NORMAL_ON;
IO_2 output bitshift numbits(8) ioLCDShiftReg
IO_4 output bit ioLCDRS = NORMAL_OFF;
IO_5 output bit ioLCDEnable = NORMAL_OFF;
IO_6 output bit io_pic_reset = NORMAL_OFF;
IO_7 input bit io_pic_ready;
IO_8 neurowire master select(IO_7) kbaud(20)
IO_8 neurowire master select(IO_1) kbaud(20)
IO_8 output bit io_pic_sclock;
IO_9 output bit io_pic_sin;
//IO_10 input bit io_pic_sout;

//---network variables---
network input SNVT_count_inc nv_LCD_value = 0
```

Figure 3.4. A typical working session of Echelon's NodeBuilder development software for Microsoft Windows.

The development node is simply a general node incorporating a Neuron IC, 64K RAM, 64K FLASH memory, and appropriate transceiver, housed in a metal enclosure with an external power supply. The front panel provides service and reset LEDs and switches, as well as a connector providing access to the node's I/O lines.

Software is developed for a node in C, and is edited, compiled and downloaded over the network to the designated target node (any node on the network) by the NodeBuilder software. Every attribute of the node including its memory map can also be configured by this software.

During the development cycle code is generally downloaded into a node's RAM, changing to FLASH only when more permanent non-volatile storage is required. For the network devices developed in this project, initial prototyping took place using the NodeBuilder node, moving to dedicated target hardware once the I/O circuitry had been finalised and a PCB produced. The Neuron ICs are surface mount devices and so, in order to reduce the time and complexity of development, commercially available modules containing a Neuron IC, memory, transceiver, and support circuitry were employed. These modules had an extremely small footprint and could be easily mounted onto a secondary PCB.

PC Windows programming

Many of the early sensor designs relied on a PC with a 48-bit I/O card to control the prototype measurement electronics. This method provided for a fast development cycle coupled with a great deal of flexibility. The programs to control the electronics were written in C using Microsoft's Visual C++ development environment. Rather than waste time creating traditional windows applications with their complicated, if attractive, user interfaces and large code overhead, all the programs produced were character-based 'console' applications. While very reminiscent of traditional MSDOS programs, 'console' applications provide true 32-bit multitasking and integrate well into the Windows 95 environment.

As well as controlling the sensor electronics, programs of this type were used to perform signal processing functions, test filter algorithms, and process logged data so that it could be imported into a spreadsheet or other Windows software package. By relying on parameter passing, many useful general-purpose utilities were created by the end of the project.

3.3. Developing a LonWorks node

Nodes are objects that interact with physically connected I/O devices and communicate with other nodes over a network, using the LonTalk protocol. All nodes include a Neuron IC for communication and control, and I/O interface for one or more I/O devices, and a transceiver to connect to the network (figure 3.5). The node's behaviour is defined by a program and configuration information contained in the node's memory. The user-definable portion of node memory consists of two parts: the application image and the network image.

The node's memory consists of three main sections: system image, application image, and network image. The *system image* includes the Neuron IC firmware that implements the LonTalk protocol, Neuron C runtime library, and task scheduler. The

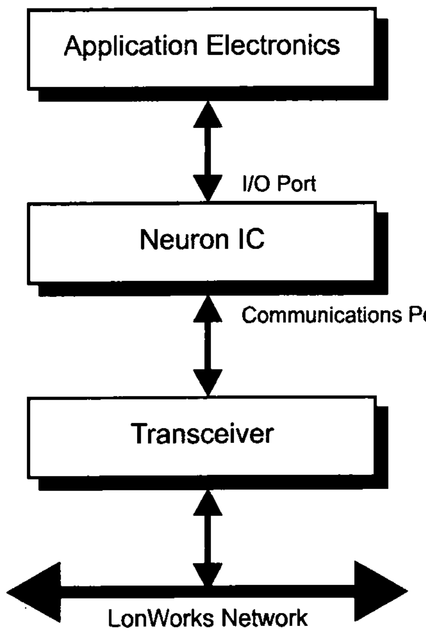


Figure 3.5. Block diagram of a generic LonWorks device.

```
when (reset)
when (nv_update_occurs(nv_temperature))
{
//code to execute when the node has been reset,
// or the network variable nv_temperature has been modified
}
```

The scheduler evaluates `when` clauses in round-robin fashion: each clause is evaluated by the scheduler and, if TRUE, the task associated with it is executed. If FALSE, the scheduler moves on to examine the next `when` clause. After the last clause, the cycle repeats. A system of priorities exists so that certain designated clauses will be evaluated in a separate order *every* time the scheduler runs. While a task is being executed, it cannot be interrupted, so good programming technique is needed to ensure that no task takes too much processing time. As an added safeguard a watchdog timer, built-in to each Neuron IC, is periodically reset by the scheduler. If a program enters a very long task, however, the watchdog timer may expire, causing the node to automatically reset.

Neuron I/O

The Neuron IC connects to application-specific external hardware via eleven pins named IO0 to IO10. These pins may be configured in numerous ways to provide flexible input and output functions with a minimum of external circuitry. The Neuron C language allows one or more pins to be declared as 'I/O objects'.

Each object can be thought of as a prewritten firmware routine in ROM which is accessed by the application program through the universal functions `io_in()` and `io_out()`. A variety of I/O objects are available: Direct, Timer/Counter, Serial, and Parallel.

- *Direct I/O Objects* are based on a logic level at the I/O pins; none of the Neuron IC's hardware timer/counters are used in conjunction with these objects.

Input Object Types	*Output Object Types*
`bit`	`bit`
`bitshift`	`bitshift`
`byte`	`byte`
`nibble`	`nibble`
`leveldetect`	`touch`
`touch`	

- *Timer / Counter I/O Objects* utilise the two timer/counter circuits in the Neuron IC, one whose input can be multiplexed and one with a dedicated input.

Input Object Types	*Output Object Types*
`dualslope`	`edgedivide`
`edgelog`	`frequency`
`ontime`	`oneshot`
`period`	`pulsecount`
`pulsecount`	`pulsewidth`
`quadrature`	`triac`
`totalcount`	`triggeredcount`

- *Serial I/O Objects* are used for transferring data serially over a pin or set of pins.

Input Object Types	*Output Object Types*
`infrared`	`serial`
`magcard`	`i2c`
`magtrack1`	
`serial`	
`i2c`	

- *Parallel I/O Objects* are used for high-speed bi-directional I/O.

Input Object Types	*Output Object Types*
`muxbus`	`muxbus`
`parallel`	`parallel`

Chapter 4

Sensor Design

4.1. Design aims

The main requirement of the sensor design is that it behaves as an interoperable device on a LonWorks network. The design of a LonWorks sensor conveniently falls into two main areas; the sensing technology and the network interface. As with any commercial product there is a danger of introducing too much complexity into a new design in one step. In order to reduce potential problems it was decided that the overall turbidity/suspended solids sensing performance and functionality of the

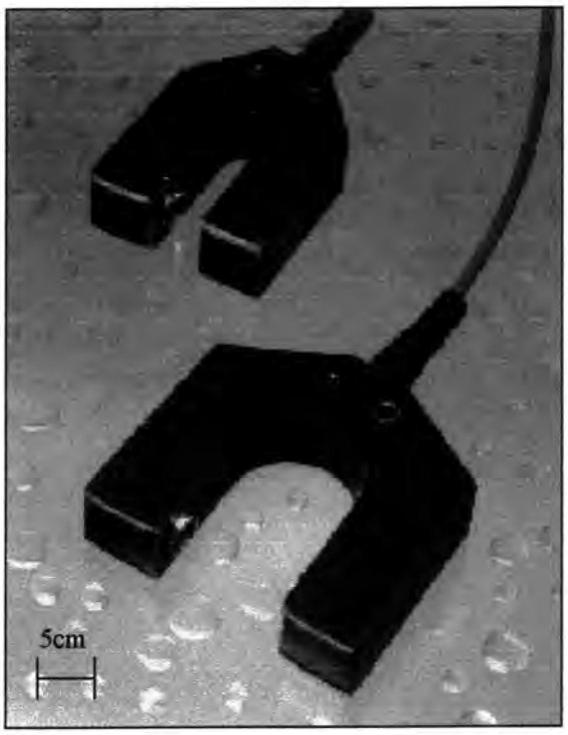


Figure 4.1. Two models from Partech's range of IR sensors. Note the different gap sizes, giving each sensor a different range and accuracy.

prototype sensor need not exceed the current generation of Partech Instruments sensors. Once the network interface has been perfected and proved to work, future designs can introduce more advanced sensing principles and processing techniques.

Partech currently manufacture a wide range of sensors for different applications, split between standard types and those capable of performing a 'self-cleaning' operation. Of the standard types, it is envisaged that the design outlined here will replace a popular range of sensors (see figure 4.1.) consisting of the IR8, IR15, IR40, IR100, and IR12LS. All of these sensors use infra-red light, with the first four types measuring light absorption while the latter measures light scatter. The numerical component of each part number refers to the path length over which absorption or scattering occurs. The reason that so many different size gaps are employed is because this parameter has such a dramatic effect on the sensitivity and maximum range of the sensor. For example, in applications where a high level of suspended solids is encountered the IR15 is commonly used, whereas for much lower levels where greater sensitivity is required, the IR100 would be preferred.

In order to try and reduce the number of sensors required, it was suggested that varying the intensity of the infra-red light emitted may have a similar effect to varying the gap size. Thus a single fixed-gap sensor could use low intensity light for sensitive applications, increasing the intensity to penetrate higher concentrations. In order to assess the feasibility of this and how complicated this feature would be to incorporate into the final design, a large amount of initial testing was proposed. A second suggested feature was for one sensor to measure both light absorption and light scatter, perhaps in a four-beam arrangement.

Whatever the sensing method employed, the processing aspect of the sensor design must be capable of linearising the raw measurement and presenting it to the network as a final turbidity/suspended solids value. Thus, the sensor's firmware must store one or more sets of linearisation data, either as a lookup table or a fixed-order equation. Additionally, the sensor must be able to be easily zeroed and re-calibrated over the network.

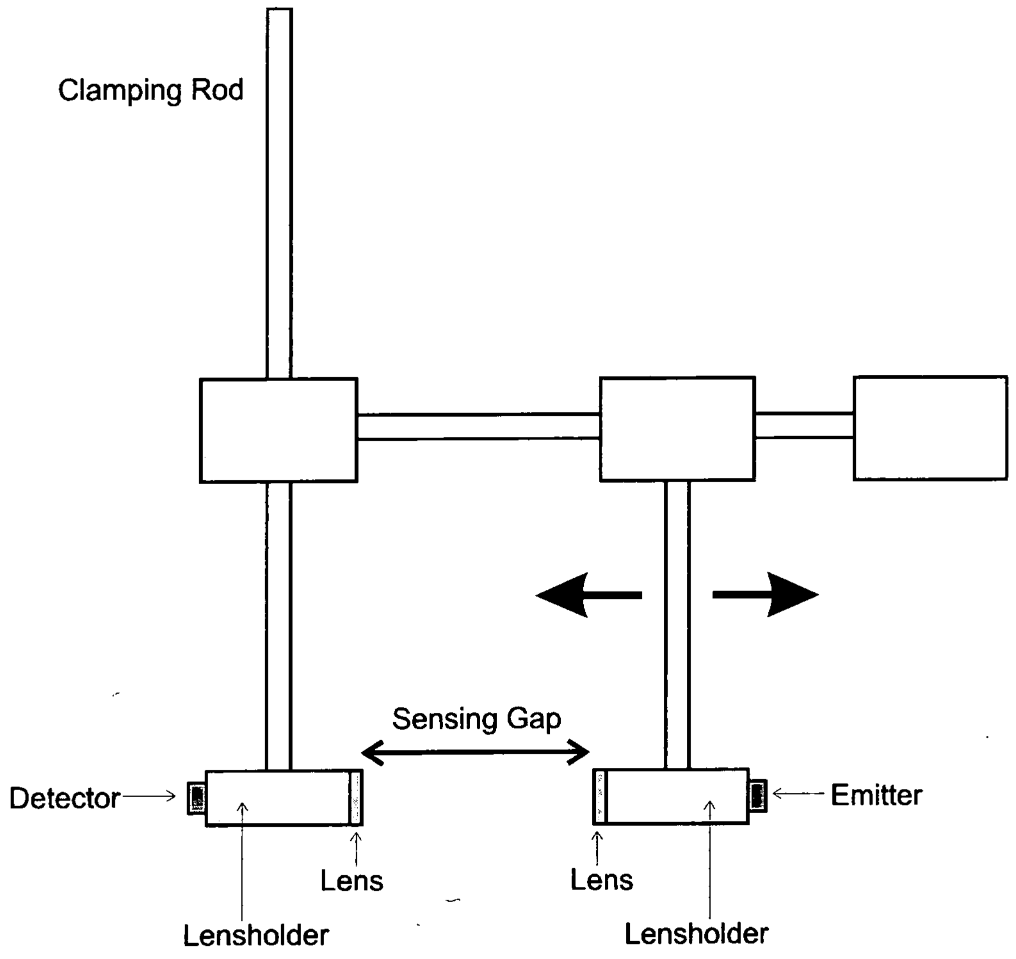


Figure 4.2. A side profile of the experimental arrangement used to investigate the relationship between emitter intensity and sensing gap.

4.2. Initial testing

Before a specification for the final design could be produced, it was necessary to try out a number of design ideas including the effect of varying the intensity of emitted light. In order to do this, the apparatus shown in figure 4.2. was produced to allow an emitter and detector to remain optically aligned in a solution, while the gap between them is adjusted. The emitter used was a Siemens 5mm high power infra-red LED with a wavelength of 880nm. For the detector, a Texas Instruments TSL260 light-to-voltage sensor was used, combining a photodiode and a transimpedance amplifier in a single package. The advantage of this device is that it offers a high irradiance responsitivity providing a large voltage output from a single 5V supply with a low dark (offset) voltage.

Both the emitter and the detector were mounted at the rear of their respective lens holders, exactly one focal length away from the lens mounted at the opposite end. The lenses used were the same bi-convex type employed in Partech's existing IR sensors, with a diameter of 13.97mm, a focal length of 11.20mm, and a refractive index of 1.513. Although moulded from a relatively low-grade glass, these lenses serve only to provide an approximate collimation of the LED's output in order to improve the efficiency, rather than to form a precise optical image.

Optical design

Once the optical components were selected, it was necessary to determine the optimum distance between the emitter/detector and their respective lenses. To produce a collimated beam, the image needs to be formed at infinity (a virtual image),

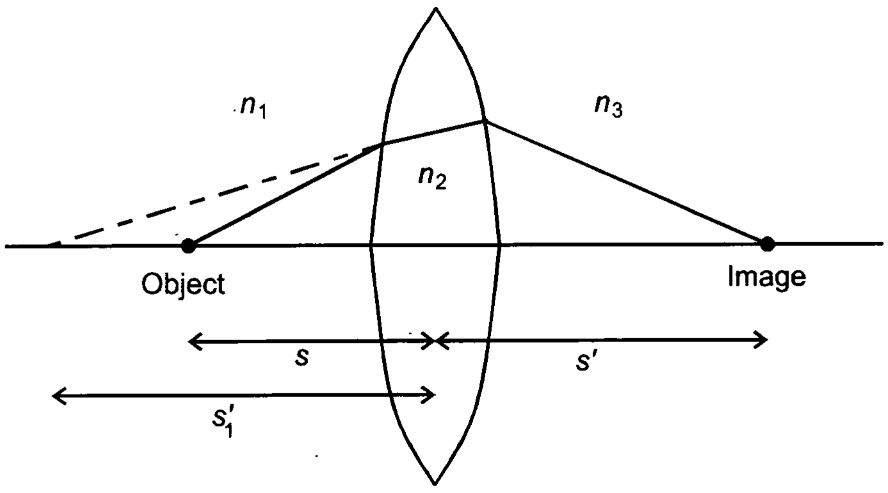


Figure 4.3. Refraction occurs at both surfaces of a thin bi-convex lens.

requiring the object to be situated at a distance of one focal length behind the lens. The focal length quoted by the manufacturer was only valid for the lenses being used in air, not at an air-water interface. In order to calculate the modified focal length of the lens, consider a very thin lens of index of refraction n_2 in media n_1 and n_3 respectively, as depicted in figure 4.3. The lens is assumed to be symmetrical with each surface having a radius of curvature, r, thus assuming that the lens is symmetrical. If an object is at distance s from the first surface (and therefore from the lens), the distance s'_1 of the image due to refraction at the first surface can be found using the small angle approximation[13]:

$$\frac{n_1}{s} + \frac{n_2}{s'_1} = \frac{n_2 - n_1}{r} \qquad 4\text{-}1$$

This image is not formed because the light is again refracted at the second surface. Rays in the glass refracted from the first surface diverge as if they came from a point behind the original object and, as they strike the second surface at the same angle, the image for the first surface becomes the object for the second. Since the lens is of negligible thickness, the object distance is equal in magnitude to s'_1, but because object distances in front of the surface are positive whereas image distances are negative there, the object distance for the second surface is $-s'_1$. Applying the same equation as before for the second surface gives the final image distance s'.

$$\frac{n_2}{-s'_1}+\frac{n_3}{s'}=\frac{n_3-n_2}{-r} \qquad 4\text{-}2$$

Adding Equations 4-1 and 4-2 eliminates s'_1, giving:

$$\frac{n_1}{s}+\frac{n_3}{s'}=\frac{2n_2-n_1-n_3}{r} \qquad 4\text{-}3$$

As mentioned previously, collimation is only achieved when the image distance, s', is infinite. At this point the object distance s is effectively the focal length f of the system:

$$f=\frac{r.n_1}{2n_2-n_1-n_3} \qquad 4\text{-}4$$

To solve for r, the system can be considered to be in air with $n_1 = n_3 = 1$, $n_2 = 1.513$, and $f = 11.20$mm. Equation 4-4 gives a value for the radius of curvature of the lens surfaces as $r = 11.49$mm. Now considering an air-water interface with $n_3 = 1.33$, and using the calculated value of r in Equation 4-4 yields the correct focal length to be $f = 16.58$mm.

A final consideration in the optical design was to ensure that the angular separation of both the emitter and detector matched the diameter of the lens, when situated a distance of 16.58mm apart. Simple trigonometry shows that the half-angle should be 22.8°. By selecting an LED with a half-angle of 20° (defined at the half maximum intensity point), the emitted radiation is efficiently collimated by the lens, avoiding the outside area where spherical aberrations occur.

Electronic interfacing

The output of the photodetector is a varying voltage and before any processing of this signal can be performed, it must be converted into a digital representation of the original analogue value. The resolution and speed of this digitisation stage determines how accurate a representation is recorded. Given that the physical turbidity/suspended solids value of typical applications changes extremely slowly, the digitisation speed was not thought to be important at this stage. However, it was not known just how high a resolution would be required of the analogue-to-digital converter (ADC), so a relatively high performance device was used.

The IC chosen for the task was an Analog Devices AD7710 signal conditioning ADC[14], with a maximum resolution of 24 bits, a maximum sampling rate of 1kHz, a self-calibrating internal reference, an input amplifier with programmable gains from 1 to 128, and a programmable low-pass digital filter. Internal settings may be modified and conversion results read using the high speed bi-directional serial port. Together with an external crystal, this IC offers an extremely accurate single-chip measurement system.

In order to vary the intensity of the infra-red emitter, an octal Darlington driver array IC was used to switch one of eight preset resistors in series with the LED. This also gave the potential for switching different values in parallel to achieve more than eight LED currents but this not feature was not fully exploited during testing.

Software control

Initially an 80C552-based microcontroller board was used to control the ADC, store readings, and transmit them to a PC via an RS-232 serial link, thus providing a completely embedded real-time control system. The main problem with this arrangement was the limitation of the microcontroller's 32K of RAM and the relatively slow serial link, which made logging many results an impossibility. It was for this reason that the PC was used to directly control the ADC using a 48 line programmable I/O card.

The PC software took the form of a simple command line menu-based application, with options to manually view ADC readings, manually adjust gain and LED intensity, automatically perform a measurement cycle, analyse logged data, and save/load logged data. The option to automatically control a measurement cycle was particularly useful as this would successively log ADC readings for a predefined time period at every LED intensity. Once logged, the minimum, maximum, mean, standard deviation, variation, mean variance, skew, and curtness of the resulting distribution were calculated and saved to disk, along with the corresponding sensor gap. In addition, at any time an extended data log of every individual reading could be saved to disk for future analysis.

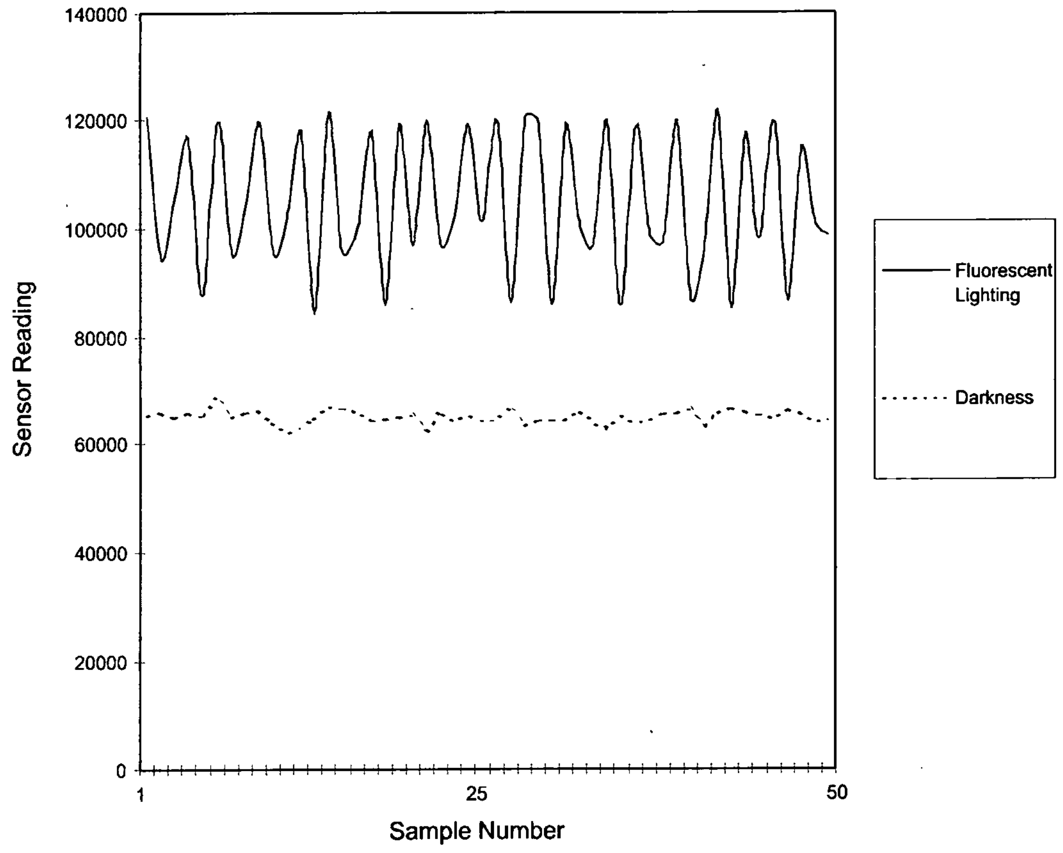


Figure 4.4. The effect of ambient light on fifty ADC values logged at 250Hz with the sensor elements in air.

Results

Before any serious testing, an initial logging run was conducted with the sensor elements in air and in normal room lighting conditions. Readings were logged at 250Hz and revealed large periodic fluctuations with frequency 50Hz, corresponding to mains variations in the intensity of the fluorescent room lighting. To remove this unwanted effect, the apparatus was operated in an almost dark environment with a much lower and essentially constant level of ambient light. Figure 4.4. illustrates the effect of this change in procedure.

The next stage was to measure the LED forward current for each of the defined resistor combinations. In each case the current was found to remain constant with time:

Intensity Number	**LED Forward Current (mA) ± 0.1mA**
0	0.0
1	0.5
2	1.7
3	3.7
4	7.5
5	13.9
6	22.0
7	33.0
8	47.3
9	60.0

Two sets of tests were carried out, one for formazine and another for fuller's earth. Before each test, the apparatus was cleaned and four litres of purified water placed in the container. For the formazine test, 25 increments of 4000 NTU stock solution were added, resulting in a maximum test solution of 800 NTU. The second test involved adding 25 increments of fuller's earth until a final concentration of 14.2g/l had been created. At each concentration the sensor gap was set to 20mm, 40mm, 60mm, 80mm, and 100mm, while the solution was kept in suspension by a stirrer. For each gap, the LED intensity was incrementally increased and a three second log made of the photodetector output at a variety of sampling rates and resolutions. At all times, the forward current through the LED was measured to ensure that the intensity levels remained constant.

The end result of this testing was a file containing a complete set of distribution characteristics for every combination of sensor gap, LED intensity, solution concentration and ADC settings. A separate program was used to scan this file, creating individual data files suitable for importing into a spreadsheet package. Graphs were then created for each sensor gap, consisting of a family of curves plotting the mean sensor reading (as a percentage of the full scale) against dilution/concentration for every intensity level. Figures 4.5. to 4.14. show the key results.

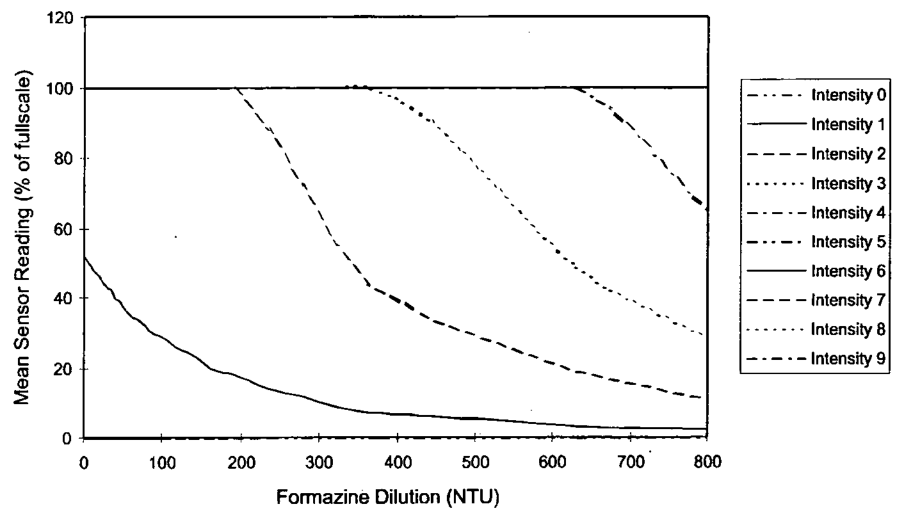


Figure 4.5. Mean sensor reading vs formazine dilution (gap = 20mm).

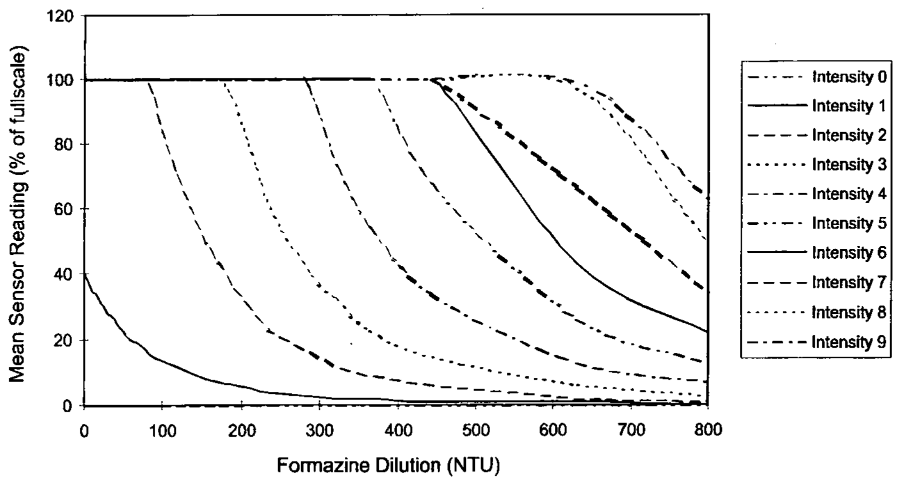


Figure 4.6. Mean sensor reading vs formazine dilution (gap = 40mm).

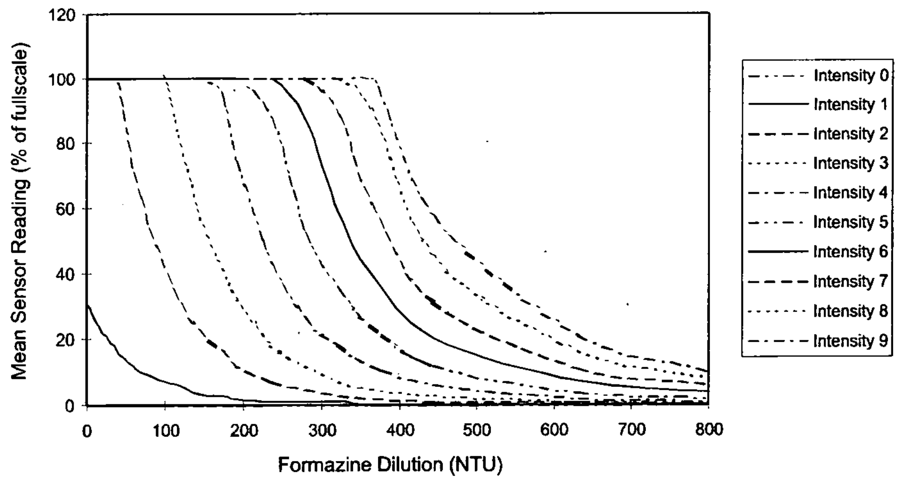


Figure 4.7. Mean sensor reading vs formazine dilution (gap = 60mm).

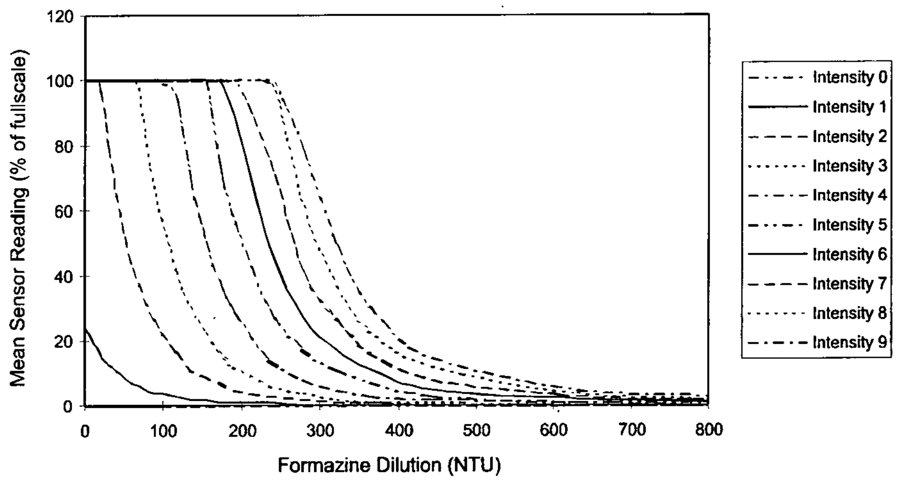


Figure 4.8. Mean sensor reading vs formazine dilution (gap = 80mm).

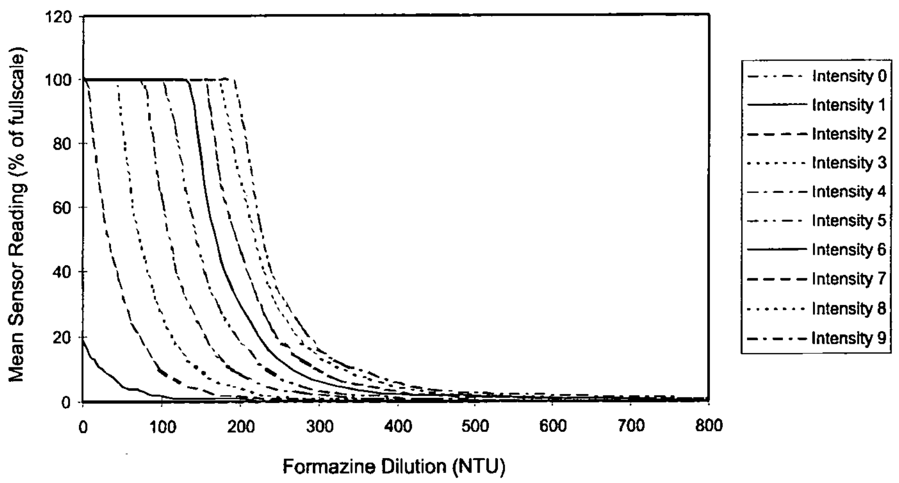


Figure 4.9. Mean sensor reading vs formazine dilution (gap = 100mm).

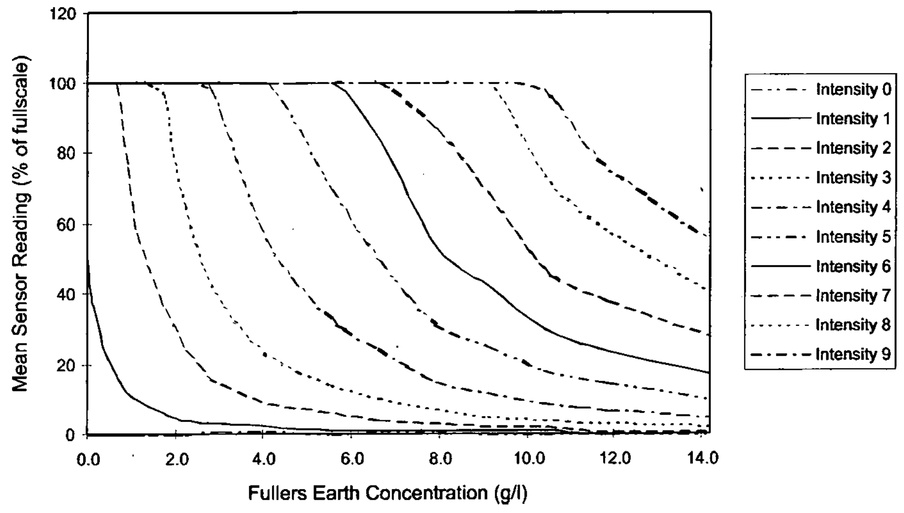


Figure 4.10. Mean sensor reading vs fuller's earth concentration. (gap = 20mm).

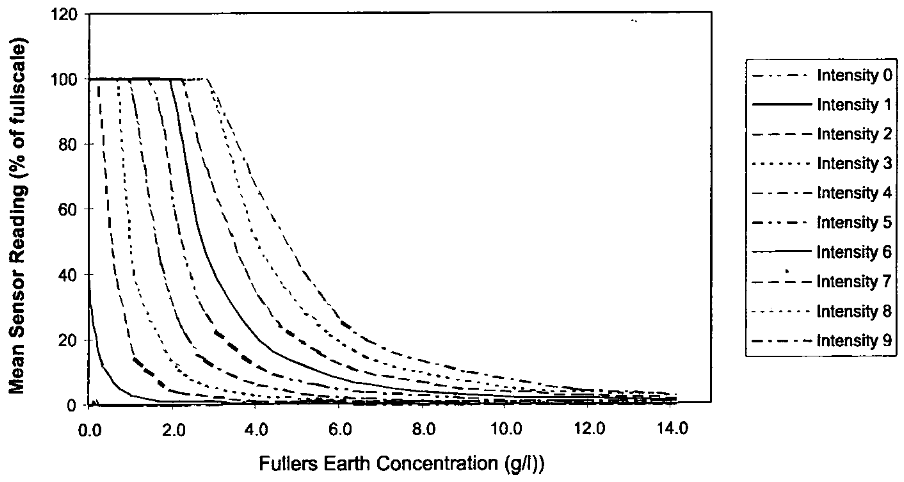


Figure 4.11. Mean sensor reading vs fuller's earth concentration. (gap = 40mm).

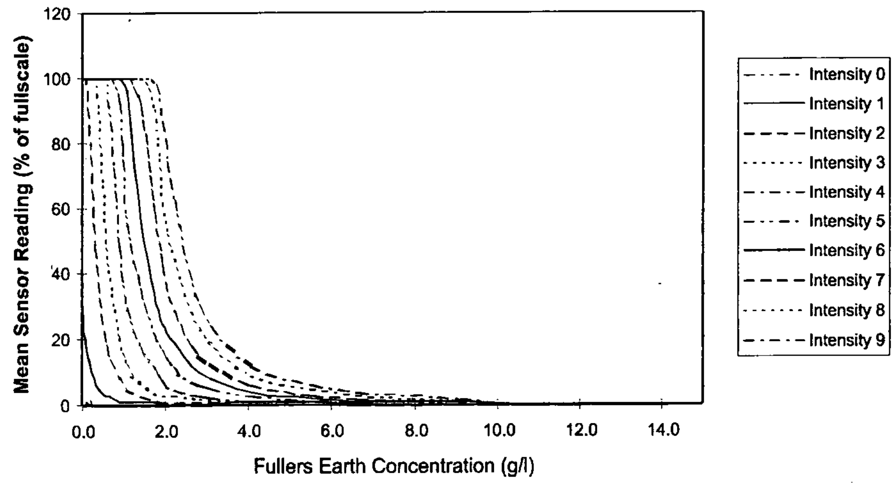


Figure 4.12. Mean sensor reading vs fuller's earth concentration. (gap = 60mm).

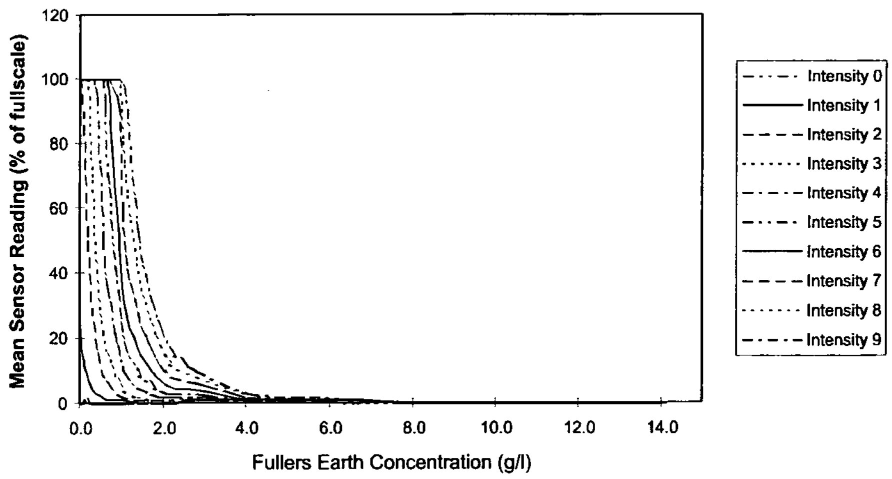


Figure 4.13. Mean sensor reading vs fuller's earth concentration. (gap = 80mm).

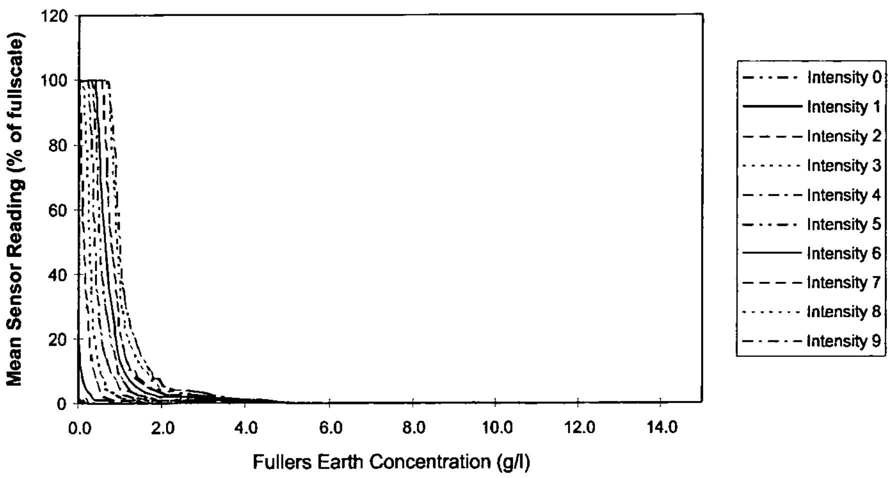


Figure 4.14. Mean sensor reading vs fuller's earth concentration. (gap = 100mm).

One of the most noticeable aspects of this series of graphs is that they all exhibit the same reverse-S shape, irrespective of test, gap, or intensity, confirming that this shape is a characteristic of light absorption in water. As the sensor gap increases, the measuring sensitivity (change in mean value per NTU or mg/l) also increases but at the expense of the maximum range available. Smaller gaps offer a reduction in sensitivity but are able to measure much higher values of concentration/dilution. Given the relatively high sensitivity of even the smallest gap (20mm), achieving this trade-off should not be too difficult.

While figures 4.5. to 4.14. are only based on readings digitised at 10Hz with 24 bit resolution, further analysis of the collected data showed that the standard deviation of the mean sensor value remained of the order of 0.1% for the majority of readings, increasing only slightly for high intensities in high levels of concentration/dilution. A particularly interesting observation is that the standard deviation does not increase as the effective resolution of the ADC decreases. Reducing the resolution from 24 bits to 12 bits produced no discernible rise in error. This would indicate that the spread of values in a logged distribution is due to some inherent optical property of the solution, rather than any inaccuracy in the digitisation process. A second observation from the tests is that the standard deviation appears to decrease slightly as the sampling rate increases. Thus, for accurate measurements in this application, digitising the photodetector output with 12-bit resolution at a high sampling rate is perfectly acceptable.

Perhaps the most important result evident from figures 4.5. to 4.14. is that by controlling the LED intensity, a single sensor gap is able to measure a much wider range than would otherwise be possible with any single fixed intensity. The maximum intensity used without causing saturation in formazine is illustrated in figure 4.15. As the mean value falls at a given intensity level, the next intensity level is able to be used to carry on measuring. Similarly in falling concentrations, as one intensity level saturates, reducing the LED intensity keeps the transmitted light within the range of the photodetector. In other words, electronically controlling the emitted radiation can control the radiation incident on the photodetector in a similar way to physically adjusting the emitter-detector separation. Of course, varying the LED intensity only

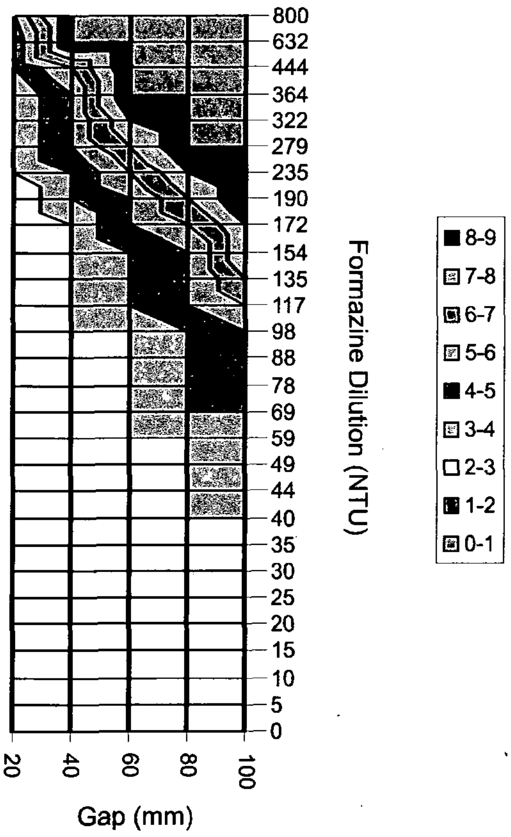


Figure 4.15. The maximum LED intensity used for each formazine dilution without saturating the photodetector.

serves to control the upper range of the sensor, whereas adjusting the gap also affects the actual measurement sensitivity.

By calculating the formazine concentration at which each intensity level gives 80% and 20% of full-scale, the effective range of each intensity level may be plotted (figure 4.16.). It can be seen that the degree of overlap between different intensity levels varies. Indeed, at certain levels of dilution there is no intensity that can provide a mean value between 20% and 80% of full-scale. This fact illustrates the coarseness of the discrete intensity levels used - a finer control of LED intensity would solve this problem.

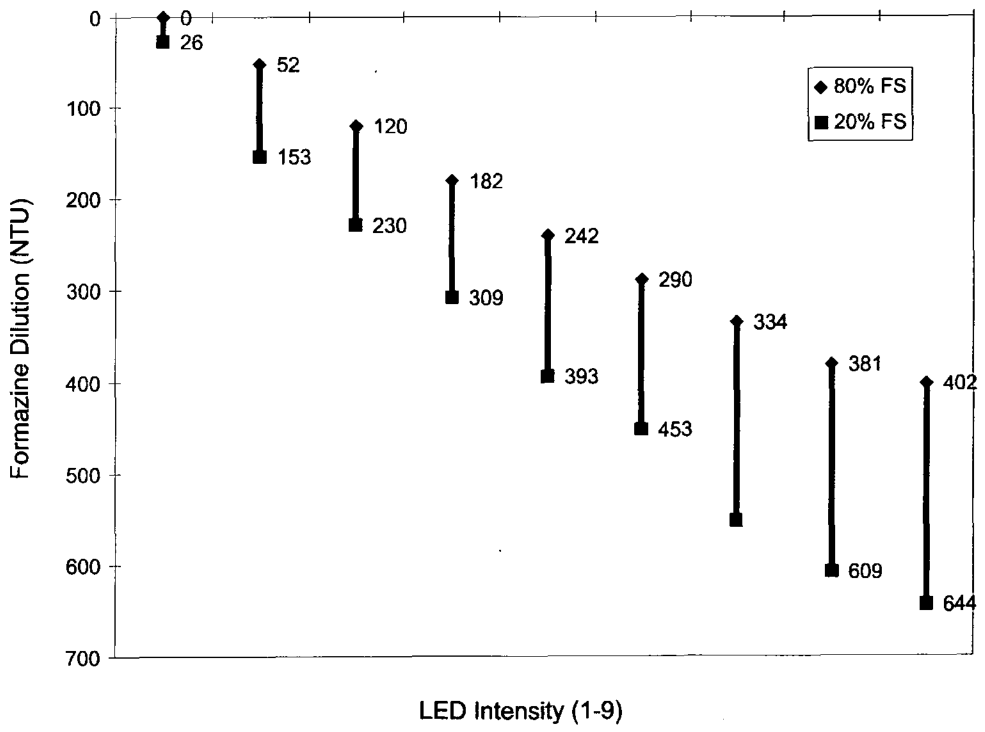


Figure 4.16. The 20% - 80% range of each LED intensity when used for measuring formazine with a sensor gap of 60mm.

While the highest intensity level used in the tests generated an LED current of 60mA, the device itself can handle a continuous current of 100mA increasing the range shown here. Additionally, the LED may be pulsed with currents of up to 1A extending the sensing range further still. A final point worth noting is that the programmable gain amplifier in the AD7710 device was set to unity gain throughout the experiment. The use of this feature is not required if the LED intensity is controlled to ensure that the light incident on the photodetector is always well within its measuring range

4.3. Design specification

By analysing the results of the testing and relating this to the initial design aims, it was possible to construct a detailed sensor specification. One of the most important problems raised by the testing was the apparent susceptibility to variations in the ambient light level - a 'feature' that is clearly not desirable in an industrial product. Partech's IR range of sensors modulate the infra-red beam so that, after de-modulation, the signal is much less affected by variations in ambient light. To incorporate this mechanism into the sensor design, the LED needs to be pulsed for a short period (in fact, just long enough for the ADC to digitise the photodetector output) and then turned off. By repeating this procedure many times a second, the effect of variations in ambient lighting will be averaged out. Increased compensation could come from digitising the photodetector output while the LED is turned off, and then subtracting this value from that with the LED turned on. Of course, it should be remembered that in the majority of applications the sensor has a good deal of inherent shielding from ambient light anyway.

The tests also revealed that there was little benefit in digitising the photodetector output with 24 bit resolution; 12 bit resolution was perfectly adequate. Instead, a high sampling rate seemed to reduce the standard deviation, improving accuracy. Also, the programmable gain amplifier feature of the ADC was found to be unnecessary.

Greater control over the LED intensity is required as the scale of nine levels used in the tests proved to be a little coarse. With the sensing gap fixed, the maximum emitted intensity essentially determines the maximum range of the sensor, so it is important to take advantage of the full LED forward current range. Given that the ambient light compensation requires the LED to be pulsed, extending the current range up to 1A should be possible.

In terms of mechanical design, it is desirable for the sensor to have the smallest gap possible. This ensures that the sensor's range is large, and the actual housing has a small profile to reduce possible fouling. Obviously the smaller the gap, the less sensitive the measurement process but the test results indicated that even with a 20mm gap, the sensitivity would be more than adequate for the vast majority of applications. Additionally, it would be useful to incorporate a second photodetector into the optical arrangement to monitor light scattered through 90°.

With these factors in mind, the following specification for the sensor design was produced:

- The mechanical design should align an LED and photodetector, separated by a 20mm gap. Additionally, provision should be made for a second photodetector to be aligned at 90° to the LED, along with a second LED.
- The sensor must be able to control up to two LEDs, varying the forward current between 0 and 1A with a resolution of at least 1mA, and be able to pulse each LED with a frequency of 250Hz or more.
- The output of each photodetector should be digitised with at least a 12 bit resolution at a sampling rate of no less than 250Hz.
- The sensor should perform appropriate signal processing, linearisation and calibration functions, presenting the final value as a single output network variable.

All calibration data should be stored locally, and the sensor able to re-calibrate itself upon receipt of a network message.

- The sensor should operate over a wide power supply range.
- A service pin LED and switch should be incorporated within the sensor.

4.4. Electronic design

Due to the change in the requirements of the ADC, the AD7710 device was no longer suitable for the task. Instead, a Maxim MAX186 IC was selected, featuring 12-bit resolution, 8 input channels, single +5V operation, low power modes, an internal track/hold, an internal reference and a 4-wire serial interface. With 8 analogue input channels, not only can two photodetectors be easily digitised, but there is also the potential for monitoring other transducers such as temperature (useful for compensation) and pressure (used to indicate depth). The serial interface allows the input channel and type of conversion to be selected, a conversion to be explicitly started, the conversion result to be read, and a power-down mode selected. A separate output indicates when a conversion is in progress. No external crystal is required.

In order to improve the resolution of the LED intensity, a digital-to-analogue converter (DAC) was used to provide a varying output voltage. The IC used, a Maxim MAX504, converts a 10-bit serial input value into a voltage between 0V and 4.096V. The circuit shown in figure 4.17. was used to set the forward current for the LEDs between 0mA and 1A for a DAC voltage of 0V and 4.096V respectively. This

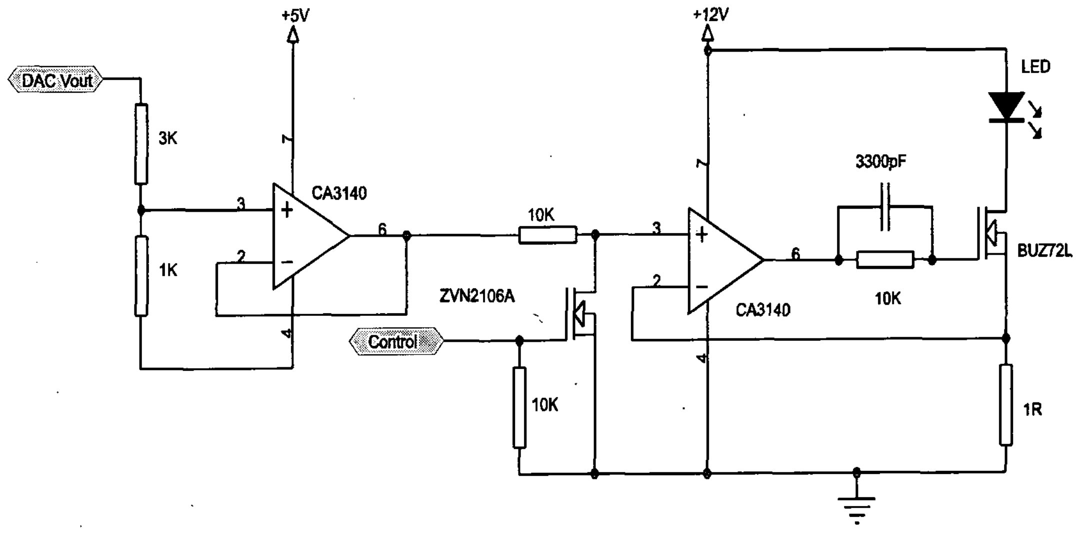


Figure 4.17. LED control circuit.

provides a forward current resolution of just under 1mA. The actual on/off control of each LED was achieved by a separate active-low digital control input.

It was originally envisaged that the Neuron IC would have direct control over both the ADC and DAC, via its in-built serial I/O model, in order to control the measurement process. However, when this system was implemented it was discovered that the Neuron IC could only handle serial communications at a maximum data rate of 10kbaud, even at the highest clock speed. While such a data rate is more than adequate for many applications, including communicating with the DAC, it only allows for around 400 discrete samples to be taken per second, even with zero conversion time. Given that two samples need to be taken to implement the ambient light compensation and that two photodetectors need to be monitored, the Neuron IC would clearly not be able to cope.

The solution to controlling the relatively high-speed measurement process was to use an Arizona Microchip PIC (programmable interface controller) microcontroller[15]. The exact device used was a PIC16C84 IC, providing a low-cost single-chip solution. This microcontroller includes a simple 35 instruction RISC core with 1K of EEPROM code memory, 64 bytes of static memory, and 13 I/O lines. An external RC network clocks the device at 4MHz. Five of the PIC's I/O lines are used to communicate with the Neuron IC, two used to control the LEDs, and another five used to communicate with the MAX186 ADC IC. An additional Neuron IC line controls the PIC's reset line, enabling the system to always power-up in a known state. Neuron I/O lines are still used to communicate directly with the MAX504 DAC IC.

During the first tests of this arrangement it soon became clear that the TSL260 photodetector, previously used, was not able to react fast enough to the short LED pulses. The rise time of its output seemed to be considerably slowed by its internal amplifier, creating a triangle-shaped waveform from the essentially square wave of radiation incident on it. To remedy this, the photodetector was replaced with a Siemens SFH2030F photodiode, featuring an integral daylight filter and a rise time of 5ns. The signal from this device was used as the input to a simple amplifier circuit based on a TLC271 IC, providing a fast response exactly matched to the voltage range of the ADC.

The power for the circuit was derived from an efficient switch-mode power supply, based on the LM2574HV IC, providing a regulated 12V output over a wide input range: 7V - 35V d.c. A simple 78L05 device was used to provide a 5V supply for the logic ICs used. The remaining circuit elements consisted of a status LED driven by a transistor, a simple reset circuit for the Neuron IC, and a service pin LED and switch.

Great care was taken to ensure that the power, digital, and analogue ground connections were kept separate, and only connected at the power supply. This is particularly important when using a sensitive ADC as even a small ground loop could affect the device's accuracy. The final schematic is shown in Appendix A.

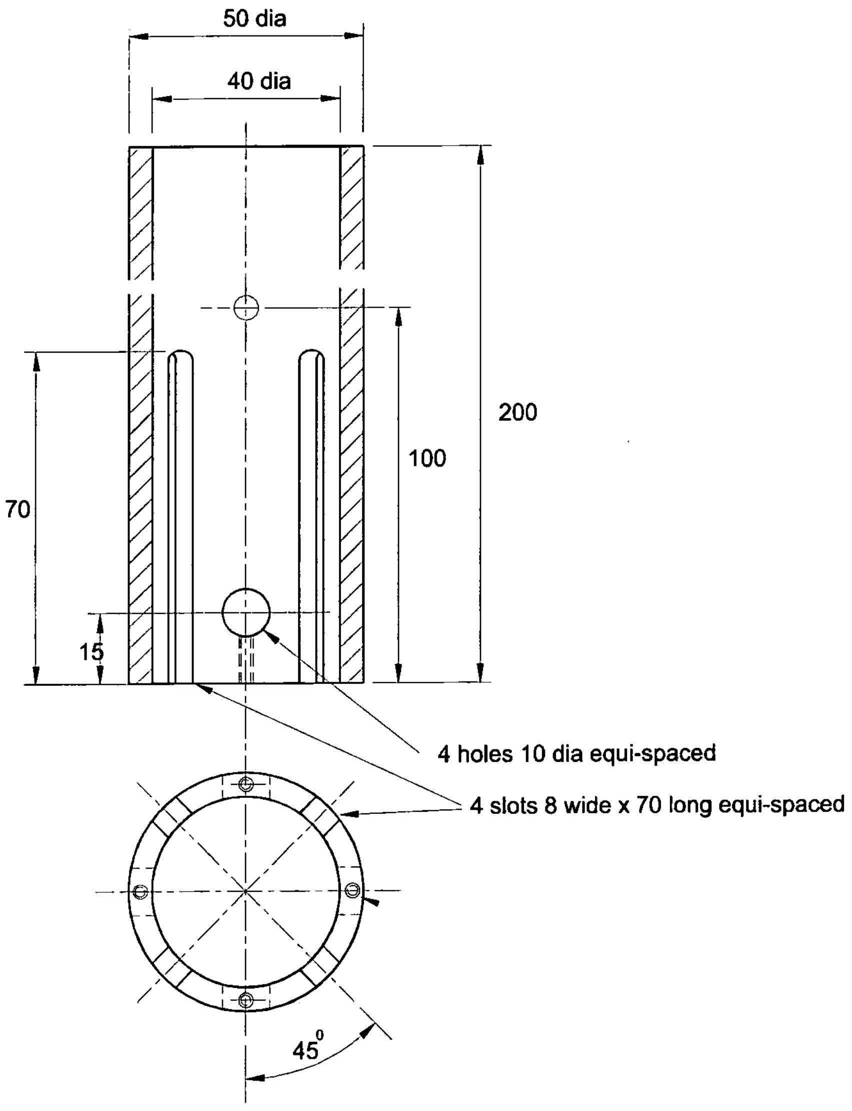

50 dia
40 dia
200
100
70
15
4 holes 10 dia equi-spaced
4 slots 8 wide x 70 long equi-spaced
45°

Figure 4.18. Mechanical design of sensor optical arrangement.

4.5. Mechanical design

Due to the prototype nature of the sensor, the electronics were housed separately from the optical arrangement, rather than attempt to create a custom sealed enclosure to accommodate the two. Figure 4.18. shows the mechanical design of the arrangement used to hold the optics in place. Four cylindrical lens holders were slotted into 10mm diameter holes and, once aligned, fixed in place with locking screws. Each lens holder contains a plano-convex lens, diameter 6.4mm, focal length 6.7mm, and either an LED or a photodiode. The advantage of using these lenses is that, with the plane face at the air-water interface, there is no refraction to/from the water, meaning that the focal length need not be re-calculated. The placement of the four lens holders, as shown in figure 4.19., allows both the transmitted and 90° scattered light to be monitored by each of the two photodiodes.

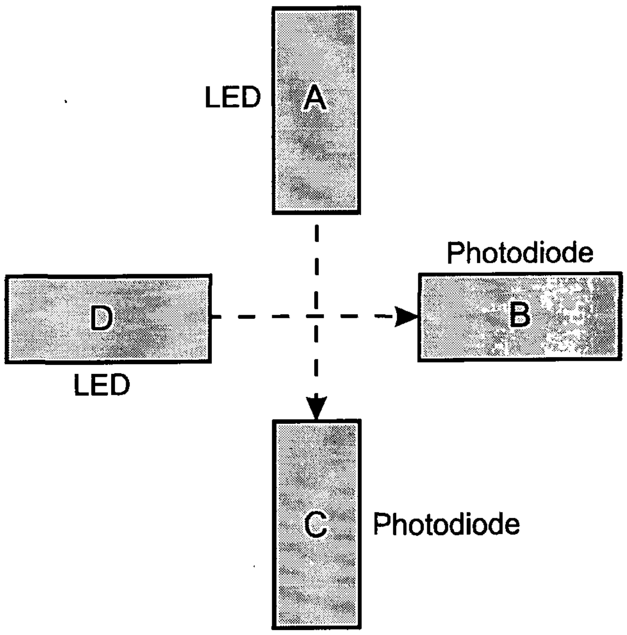


Figure 4.19. The optical arrangement of the sensor's four lens holders.

A length of screened cable was connected to each LED and photodiode, and sealant used to protect the lens holders from the ingress of water and dust. The PCB was mounted inside a separate enclosure, with four DIN connectors to match the screened cables from the main sensor assembly. An additional connector served to provide power and network communications.

4.6. PIC software design

The role of the PIC software is to control the measurement cycle, firing the LEDs as required and digitising the corresponding photodiode signals. Its aim is to acquire as many measurements in the time interval between communicating with the Neuron IC. This communication takes the form of a serial data transfer, with the PIC providing information about the number of measurements taken as well as the actual totals of each measurement made. From this, the Neuron software is able to calculate the mean value and perform ambient light compensation, in order to arrive at the final raw sensor measurement. The order of the measurement cycle is detailed below:

Stage	LED A	LED D	Photodiode	Description
1	Off	Off	C	Ambient level C
2	On	Off	C	0° scatter C
3	Off	On	C	90° scatter C
4	Off	Off	B	Ambient level B
5	On	Off	B	0° scatter B
6	Off	On	B	90° scatter B

The organisation of this sequence is such that the number of stages between each LED firing is kept to a maximum. This is important because when pulsing an LED with a high current, not only must the duration of the pulse not exceed a predefined time period, but the duty cycle (ratio of on-time to off-time) must also stay below a critical value, dependent on current. Another feature of the measurement sequence is that the change between photodiodes, and therefore ADC channels, is kept to a minimum, reducing the number of control bytes that need to be sent to the MAX186.

When the PIC processor resets, it immediately turns off both LEDs, initialises its various data structures, and then puts itself into a low-power sleep mode. It only awakes from this mode when the Neuron IC toggles an I/O line, generating an interrupt within the PIC. Upon vectoring to the interrupt address, the PIC checks the state of one of its I/O lines to determine what action to take. If this input is low then communication with the Neuron takes place, otherwise the PIC commences a single measurement cycle. This involves using simple lookup tables to determine the state of the LEDs, the ADC channel to use, and which total to add the digitised value to. Once the LEDs states are set, the conversion is started and, when complete, the LEDs turned off. The 12-bit result is read, and added to the relevant 32-bit total. At the end of the complete cycle, the 16-bit count of the number of measurements taken is incremented by one, and the sleep mode entered.

When the PIC determines that the Neuron IC wishes to transfer the accumulated measurement data, it jumps to a routine that uses two of its I/O lines to implement a simple serial protocol, with the PIC acting as a slave. This means that it is up to the Neuron IC to provide the necessary clock pulses and to ensure that the two devices remain synchronised. The six 32-bit totals and the one 16-bit count are transferred to the Neuron IC. Once successfully completed, the PIC resets this information before again entering a power-down mode. The complete listing of the PIC assembly language program can be found in Appendix B.

The advantage of this arrangement is that the overall task of producing a measurement from the sensor is split between two microcontrollers, according to their particular abilities. The PIC is able to undertake the high speed serial communication required to control the ADC and accumulate readings, while not getting involved in the more numerically-intensive processes. The Neuron IC can then utilise its high-level floating point routines to further process the sensor data, accumulated over a relatively long time period, and make the result available to the network. Also, the Neuron IC can control the number of measurements the PIC takes per second by varying the frequency of the pulse train that generates the interrupts. Because the PIC is in a low-power mode between interrupts, by reducing the frequency, the Neuron IC can also reduce the sensor's power consumption as desired.

4.7. Neuron software design

As has already been discussed, the PIC relieves the Neuron software from having to directly control the measurement process. Instead, the role of the Neuron IC is to further process the measurement data that it transfers from the PIC, performing averaging, damping, and linearisation operations. When reset, the Neuron software initialises its data structures, resets the PIC, and activates a 'frequency' I/O object to generate a square-wave output that provides the PIC processor with a constant stream of interrupts. A software timer, *measure_timer*, is employed to trigger communication with the PIC at regular intervals. A single function, *PicComms*, allows for a bi-directional transfer of a given number of bits to/from a given memory location. If, at any time, a problem is encountered when communicating with the PIC, the Neuron software resets the device and attempts to recover from the situation. When the measurement timer expires, an associated task uses *PicComms* to transfer the six 32-bit totals and the 16-bit count into local variables. Integer arithmetic is used to subtract the ambient light totals from the other totals, and to calculate the final mean values for both the transmitted light and the 90° light scatter.

Due to the prototype nature of the sensor, it was decided to concentrate on the transmitted light measurement and to try and achieve a level of performance to match the existing range of IR sensors, which are based solely on light transmission. The first stage in the process of translating the averaged sensor value into a final suspended solids value is to apply a damping algorithm. This is already used in Partech's current products and is required to smooth out the inevitable fluctuations that occur when measuring such a turbulent parameter. Many applications require the sensor to be situated in a fast flowing stream of water which creates bubbles and air pockets around the sensor, causing spurious values to be produced. It is in some ways similar to the electrical noise that can corrupt an analogue signal. What is needed is a method of removing the noise to reveal the general trend of the data beneath.

The first, and most obvious, method used was a simple rolling average. The last *n* sensor measurements are stored, the newest always replacing the oldest, and the average calculated every time a new measurement is incorporated. The effect of this process on a set of sample data can be seen in figure 4.20. The processed value certainly has much less variation when compared with the unprocessed signal, but the initial slow response of the rolling average, and when there is a sudden change in the data, is also apparent. This slow response is due to the fact that the rolling average 'buffer' needs to become full before the benefit of the averaging will be seen, and that all the values in the buffer have an equal weighting. Obviously, decreasing *n* will improve the response time but this will also have a detrimental effect on the level of the averaging.

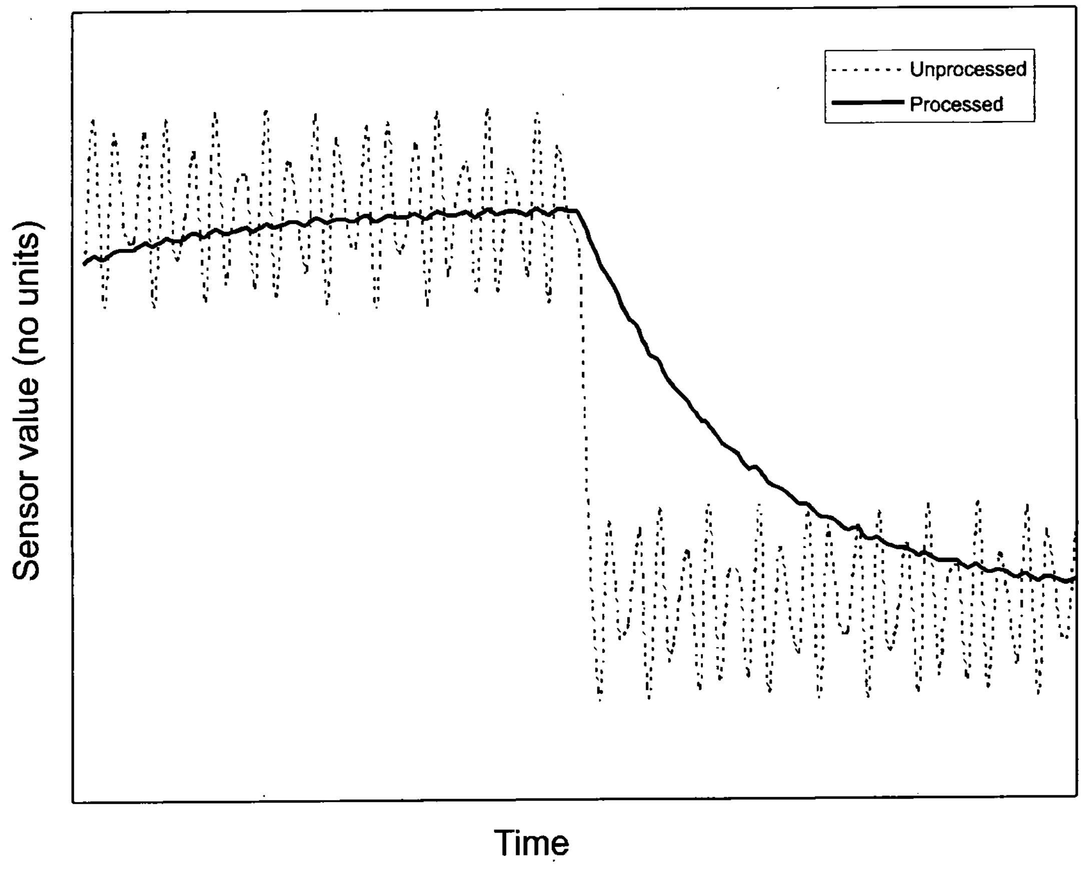


Figure 4.20. The effect of a rolling average on a sensor value changing with time.

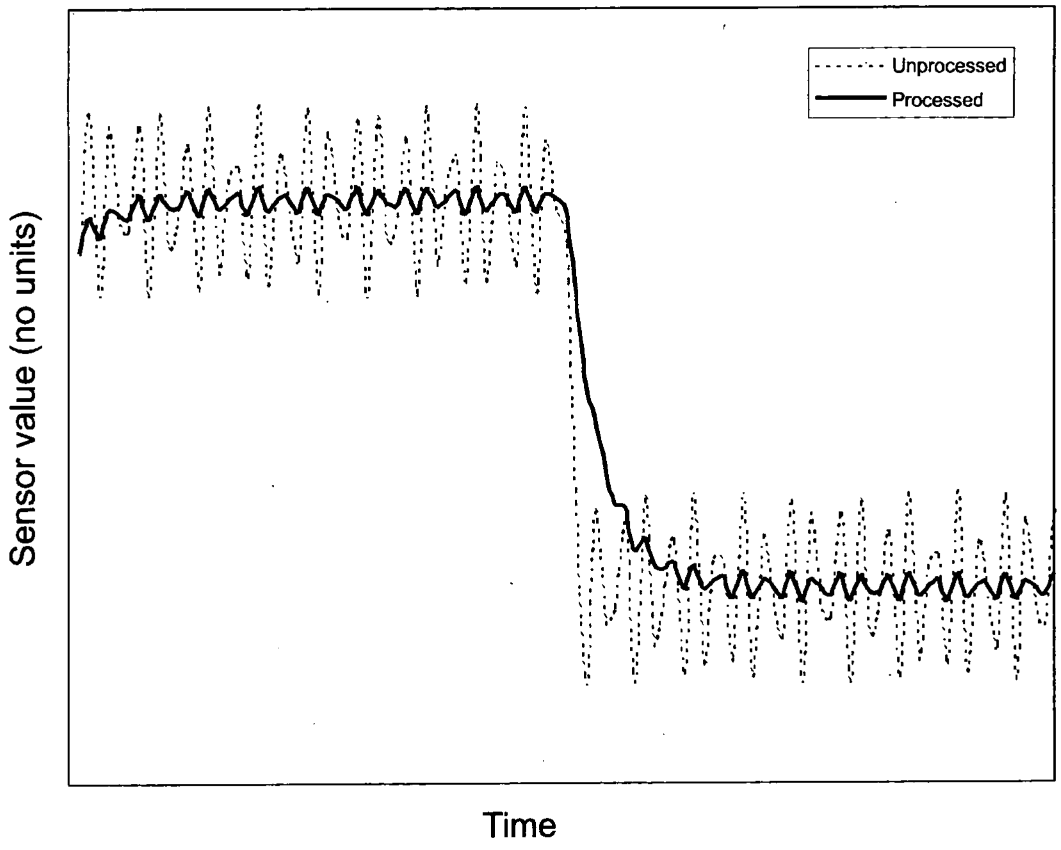


Figure 4.21. The effect of a single pole infinite impulse response filter on a sensor value changing with time.

An alternative method is based on the following equation:

$$y(n) = x(1-a) + a.y(n-1)$$

This is an averaging filter which assigns prior readings an exponentially-decreasing weight. Recent values are weighted more, with the exponential decay determined by the value of a ($0<a<1$). Figure 4.21. illustrates the result of this on the same sample data as before. The processed data is again far 'smoother' than the unprocessed signal but, this time, its response to a step change in the underlying data is much improved. Essentially, this is an example of a single pole infinite impulse response (IIR) filter with a controlling the trade-off between smoothing and response time.

Having arrived at a stable sensor value, the next step is to translate this into a suspended solids value. An immediate observation from figures 4.5. to 4.15. is that as the suspended solid/turbidity value of the solution increased, the sensor value itself decreased. This is a result of the ADC digitising the level of the light transmitted through the solution, rather than the level of the light absorbed. To rectify this, the sensor value is subtracted from the full-scale value to ensure that a low ADC reading will correspond to a low suspended solids value.

It is important that the sensor provides a linear response to the parameter to be measured. One method of achieving this would be to store a 'lookup table' of sensor values, each with a corresponding linear parameter value that could be scaled and offset. However, given the characteristic shape of the sensor reading response, a more elegant solution is to simply store the equation of the curve that best matches this shape. Once this has been done, any sensor value can be accurately translated into a linear measurement. The general equation of the curve used to describe the sensor response is identical to that used in current Partech controllers, and is therefore known to perform well in a wide variety of applications:

$$y(x) = \frac{1-\exp(x/2a)}{a} + \frac{1-\exp(x/10b)}{b} + \frac{1-\exp(x/100c)}{c} + \frac{1-\exp(x/1000d)}{d}$$

The coefficients, *a*, *b*, *c*, *d*, govern the precise shape of the sensor's response curve. In order to ensure that the calculated suspended solids value is zero at the correct point, an offset is subtracted from *y(x)* - generally the sensor reading in clean water. Having done this, the software can directly convert the raw sensor measurement into a suspended solids value for the type of solution used to derive the curve coefficients. The problem with this is that the sensor needs to be able to monitor the wide range of different solutions found in industrial applications, not just the type of solution used for testing in the laboratory. Fortunately, application information accumulated by Partech shows that the majority of industrial effluent shares the same basic response curve as that for formazine and fuller's earth. Therefore, the software only needs to linearly scale this basic shape to fit a spot measurement supplied by the user. This calibration phase does not modify the curve coefficients but simply calculates a scaling constant. For applications that require a radically different type of response, an alternative set of coefficients would need to be calculated.

The greatest challenge in writing the Neuron software was not the implementation of the procedure described to calculate a single measurement, but rather how best to control this process over the network. In order to provide a degree of interoperability, standard network variable types were used to output the final measurement and also to allow the user to calibrate and zero the sensor. One problem encountered was the lack of a suitable SNVT (Standard Network Variable Type) for communicating suspended solids measurements. To overcome this, a temporary unofficial SNVT type (SNVT_suspended_solids) with range -1E38 to +1E38 and units 'mg/l' was created and also included within the node monitor device. However, to promote interoperability, an input network variable of this type was also defined to set the full scale value of the measurement. This allows the measured value to be expressed as a percentage of this full scale (-163.84% to +163.385%), and communicated as a standard percentage level SNVT.

A simple discrete level network variable was included to allow the user to easily zero the sensor. When the software detects that this variable state has changed from OFF to ON, the current sensor value is used as the offset to subtract from future measurements. Calibration works in a similar way, but requires the user to supply the desired measurement value via an input network variable of type SNVT_suspended_solids. When this variable is updated, the software calculates a new scaling value based on the current sensor value and the supplied calibration figure. During both the zero and calibration phases, the output network variables are not updated. This is visually conveyed to the user by the status LED activating to indicate that the sensor is in a non-measuring state. Similarly, the LED is active while the sensor is off-line. Internal error conditions are reported by flashing this LED.

The following table summarises the variables that represent the sensor's network interface:

Name	**Type**	**Dir**	**Description**
nvoMeasurement	SNVT_suspended_solids	Output	Fully processed and linearised measurement
nvoSensorValue	SNVT_count_inc	Output	Raw sensor value prior to linearisation
nvoPercentage	SNVT_lev_percent	Output	Measured value as a percentage of full scale
nviFullScale	SNVT_suspended_solids	Input	Measurement full scale
nviZero	SNVT_lev_disc	Input	Replace the sensor offset with the current sensor value
nviCalibration	SNVT_suspended_solids	Input	Value to be used in calibration
nviIntensity	SNVT_count_inc	Input	Intensity of LEDs

4.8. Results

Assessing the performance of this sensor is especially difficult as it is not intended for one single application. The precise response curve, and therefore the linearity, could vary greatly according to the nature of the solution being monitored. In reality, regular cleaning and calibration of the sensor will be required to ensure its continued effective operation. As long as the exponential equation is an adequate representation of the sensor's response, the linearity of the suspended solids measurement will be at least as good as Partech's current IR range of sensors. The precise resolution of the measurement varies across the sensing range, but for typical applications, where only a rough on-line indication is required, this is not an issue.

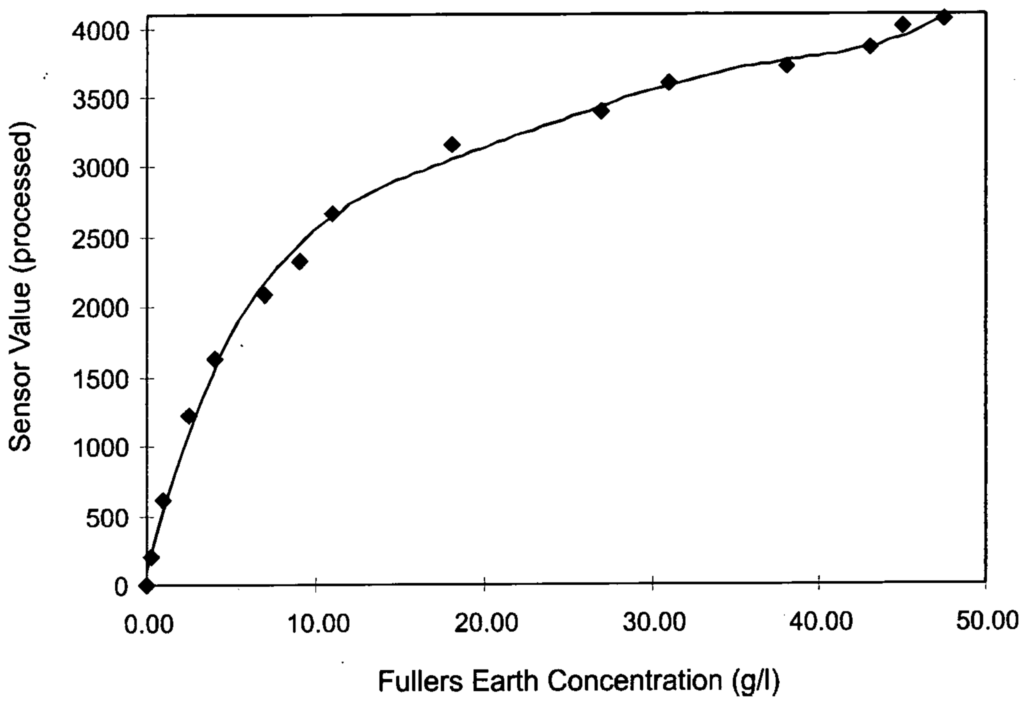


Figure 4.22. The variation of sensor value (inverted, offset, damped) with fuller's earth concentraion for a fixed LED control value of 60.

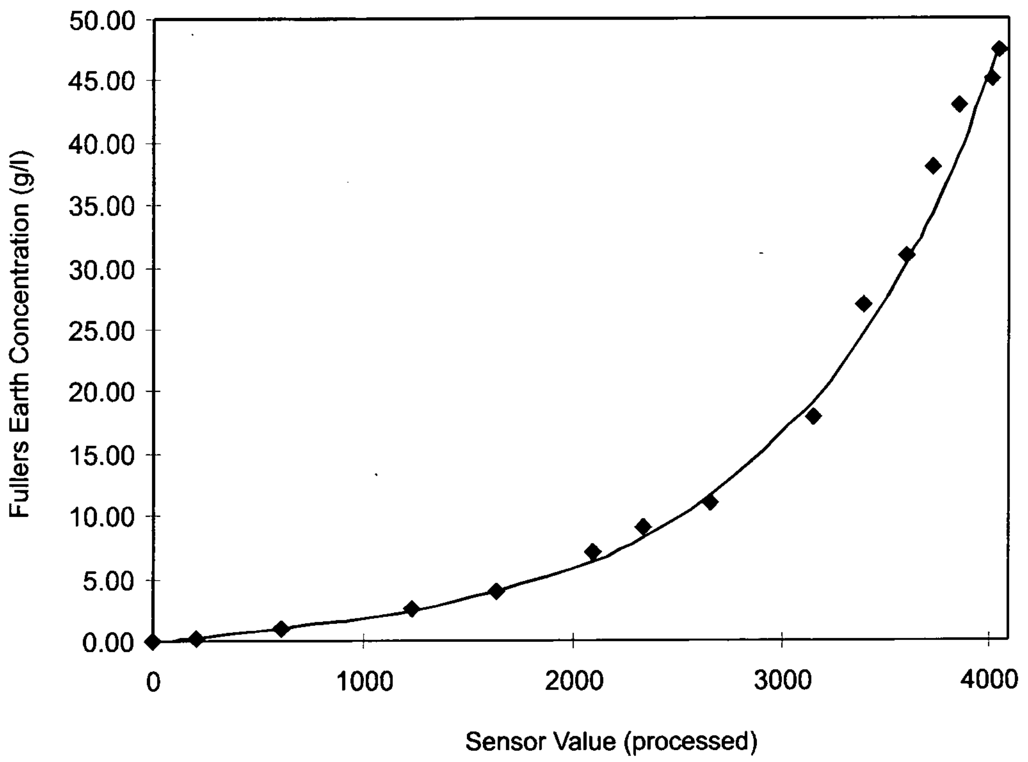


Figure 4.23. Fitting a suitable equation to the test data of figure 4.22.

In order to evaluate the success of this sensor as a replacement for one or more sensors in Partech's IR range, some means of comparison is required. The most obvious test to conduct is an incremental fuller's earth solution, performed in the same manner as before. Figure 4.22. shows the results of this test in the form of the sensor value, having been inverted, damped and offset, plotted against the solution concentration. The LED was set to a constant DAC value of 60 which, as can be seen, caused the emitted light to be completely absorbed at 50g/l. In order to linearise this data, the software must construct an internal equation to relate the sensor value to an actual suspended solids measurement. The degree to which this equation can be made to fit the data is illustrated in figure 4.23. which shows the internal equation superimposed on the original test data points. The coefficients of this equation are

a = 470, b = 270, c = 220, d = 68 with a scaling value of -280. By plotting this calculated suspended solids value against the experimental suspended solids value, shown in figure 4.24., the linearity of the sensor is seen to be extremely good at low values, becoming slightly poorer as the concentration increases. This may well be due to the increased turbulence present in higher concentration fuller's earth solutions. Even so, the straight line fit has an R^2 value of 0.9929 indicating a high degree of accuracy in the measurement. This compares favourably with the Partech IR15 sensor normally used for this application, which achieved an R^2 value of 0.9134 in the same test, although having a slightly higher range and correspondingly smaller linear gradient.

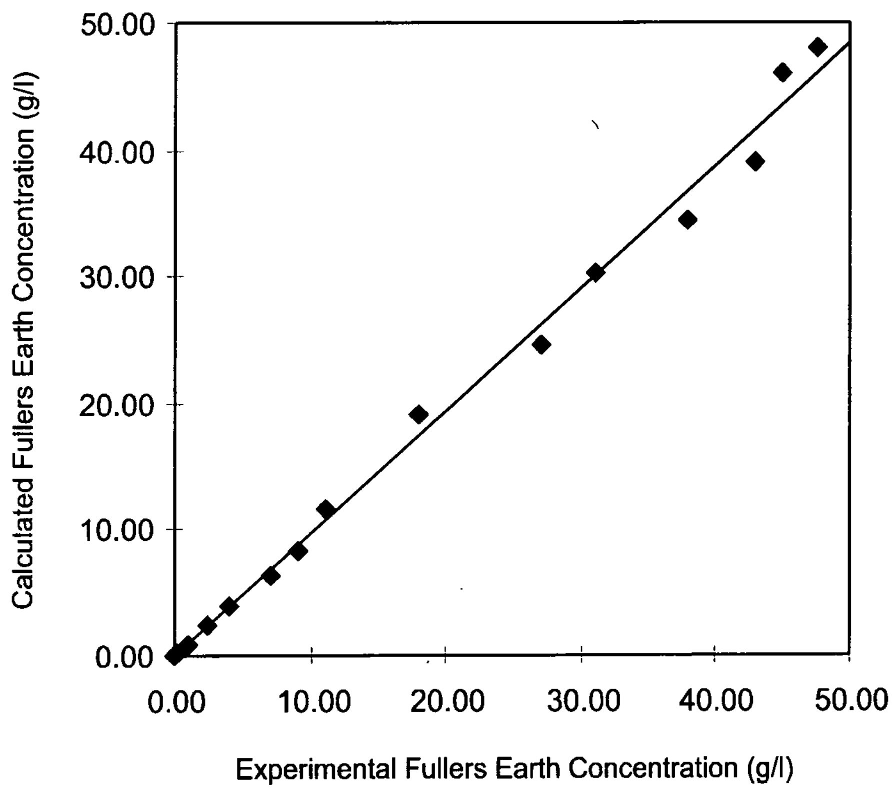


Figure 4.24. Analysing the linearity of the measurement process for fuller's earth.

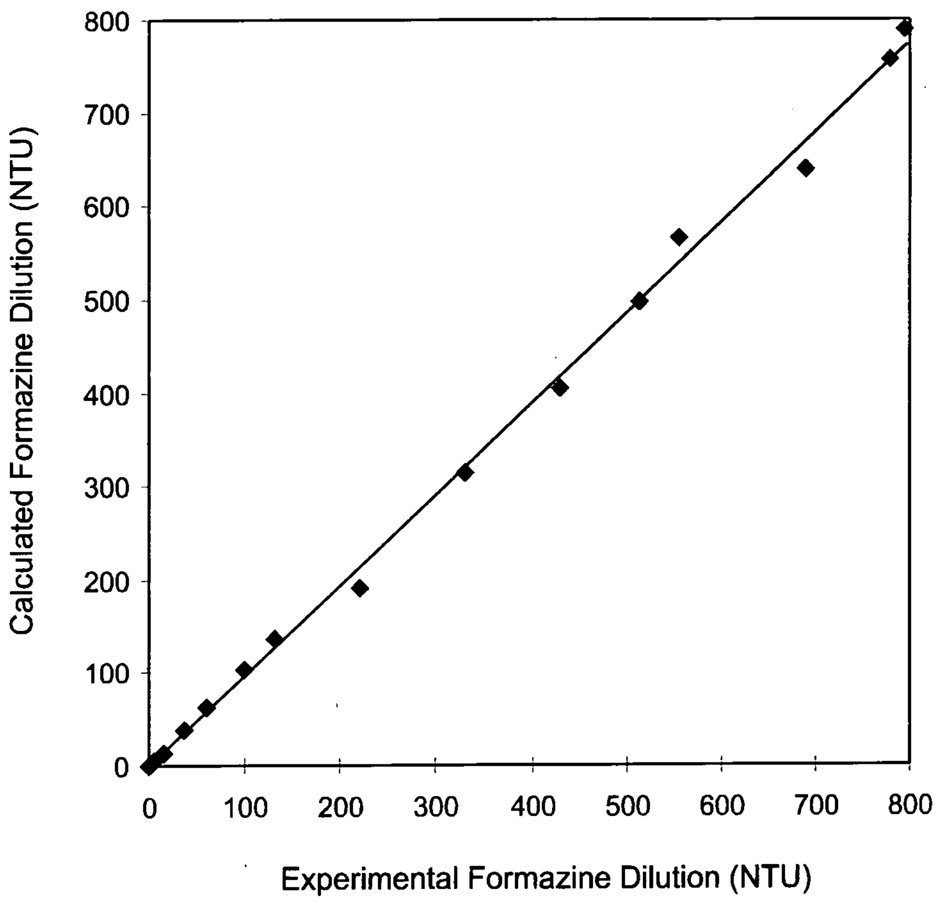


Figure 4.25. Analysing the linearity of the measurement process for formazine.

The test was repeated for an incremental formazine dilution and the linearity again examined. The same curve coefficients as before were used but the scaling factor was changed to -4600 to match the new test data. The resulting graph, shown in figure 4.25., has an R^2 value of 0.9976, again comparing favourably with an IR15 sensor. This confirms that the exponential curve, described by its coefficients, is a good representation of both fuller's earth and formazine solutions.

Chapter 5

Node Monitor Design

5.1. Design aims

The role of a 'node monitor' within this project is to continuously display an on-line turbidity/suspended solids value from the sensor, and to allow the operator to calibrate the sensor and to adjust the setpoint values of the relay module when required. In the context of LonWorks, this translates into a requirement to communicate with another node and to display/modify its network variables. This functionality is essentially provided by the NodeBuilder software, albeit targeted at the developer and therefore in a form too complicated for trivial use. An alternative could be to write a simple Windows application, perhaps using Visual Basic in conjunction with Echelon's DDE server. While this would certainly provide a working solution, and the possibility of portability using a laptop PC, the use of a dedicated computer is not cost effective for situations where continual monitoring is required. Instead, the decision was taken to develop a low-cost dedicated node monitoring device, capable of being easily used in the field.

The next decision made was whether the node monitor design should be specific to this project, or if it should be a universal tool suitable for other applications as well. It would be relatively straightforward to create a simple node with an input network variable for the sensor and two output network variables for each of the relay setpoint values. Such a device would have a very limited capability, not being able to cope with multiple turbidity/suspended solids sensors or multiple relay modules, and only being able to display one type of network variable. By expanding on this original idea, a design was created for a universal LonWorks tool, capable of communicating with a large number of nodes and displaying any type of network variable. A list of the basic features required was then drawn up:

- The node monitor should be a low-cost device, allowing it to be used as a permanently installed man-machine interface (MMI) in multiple locations, rather than viewed as a single portable tool for occasional use
- The node monitor should receive a service pin message from any node and add its addressing information to an internal database of 64 nodes. Each node in the database should have a user-definable 16 character name for easy identification.
- The user should be able to easily select any of the nodes stored in the internal database, prompting the node monitor to communicate with that node and retrieve the appropriate information held in its memory, particularly that relating to its network variable structures.
- Once the network variable information has been retrieved, the user should be able to select a network variable to monitor.
- The node monitor should poll the selected network variable at a user-definable interval and display its value in its native numerical format with correct units.
- The user interface should consist of a 16 character by 2 row alphanumeric liquid crystal display (LCD), and a set of dedicated keys in order to keep the device simple and quick to use in the field.

5.2. Electronic design

In order to satisfy the requirement for a low-cost solution the design is based around a single Neuron IC, rather than including an additional microprocessor. Although the node monitor's I/O may appear to be relatively simple at first - an alphanumeric LCD and a handful of keys - providing this functionality with just eleven I/O lines was the biggest electronic design problem encountered.

The user interface demanded a minimum of seven keys which, combined with the four data lines and three control lines (enable, register select, and read/write) needed to drive the LCD, exceeded the available I/O by three lines. The first solution tried was to drive the LCD from the outputs of an eight bit shift register using just three Neuron I/O lines (clock, data, latch). This system worked but required four complete shift cycles to write a byte into the LCD (control bits, control and data bits, control and data bits with enable = 1, control and data bits with enable = 0), reducing the response of the display. Also any synchronisation problems, due to electrical noise or software interrupts, caused spurious characters to appear on the display.

A second circuit provided a more elegant solution, driving six of the LCD lines (four data bits, enable, register select) directly from the Neuron IC and using another three to drive a 3 to 8 line decoder IC. The outputs of this IC are used to sequentially scan one terminal of each of the seven keys, with the other terminal of the keys connected via diodes to a common Neuron input. The eighth decoder output is used to control the LCD read/write line which only changes state occasionally. The one remaining Neuron I/O line is used to provide a pulse-width modulated signal which, after being converted to a variable voltage by a simple op-amp circuit, allows the LCD contrast to be adjusted under software control. This last feature may initially appear to be something of a luxury considering the limited I/O available, but past experience has shown that the contrast of LCDs is extremely temperature-dependent. For instruments used in outdoor conditions a wide temperature swing can be expected, and by implementing a software contrast control, the issue of providing an environmentally-sealed analogue contrast control is neatly avoided.

The following table summarises how the Neuron IC's eleven I/O lines are allocated:

I/O Line	Direction	I/O Object Type	Purpose
IO0	Output	PULSEWIDTH	LCD contrast
IO1 - IO4	Output	NIBBLE	3 to 8 line decoder IC
IO4 - IO7	Bi-directional	NIBBLE	LCD data bits D4 - D7
IO8	Input	BIT	Universal key input
IO9	Output	BIT	LCD enable
IO10	Output	BIT	LCD register select

An efficient switch-mode power supply, based on the LM2574HV IC, was included in the electronic design to provide a regulated 5V source over a wide input range: 7V - 35V d.c. In order for the node monitor to communicate with the NodeBuilder development tool, and to enable future software updates in the field, a service pin switch was included, although a service pin LED was viewed as unnecessary. The last remaining circuit element to be added was a simple reset circuit for the Neuron IC in accordance with the manufacturer's guidelines. The final node monitor schematic is shown in Appendix D.

The schematic was used to design a 150mm × 80mm double-sided printed circuit board.

5.3. Mechanical design

The first stage in the mechanical design process was the design of the instrument's front panel - once the number of available keys had been finalised in the electronic design. The layout, shown in figure 5.1., achieves a relatively compact form factor without cramping access to any of the functions.

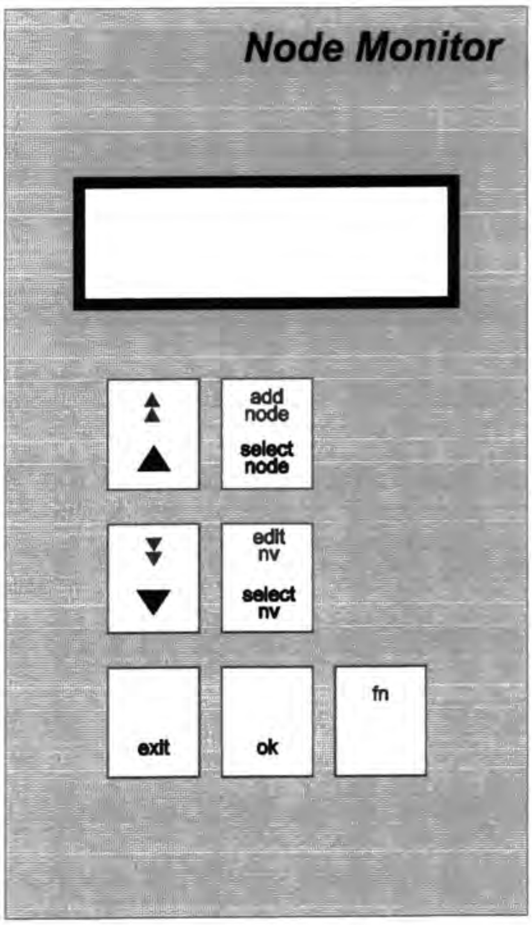


Figure 5.1. The node monitor front panel overlay design.

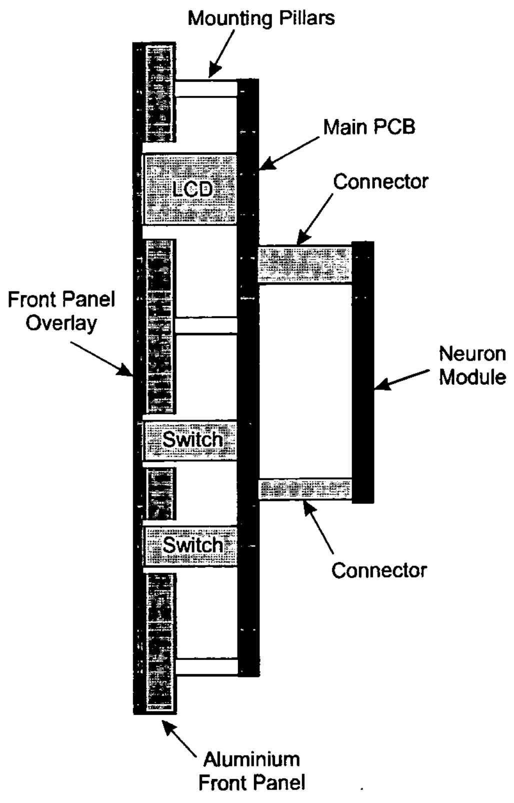


Figure 5.2. A cross-sectional view of the mechanical structure.

The next stage was to select an enclosure that closely matched the front panel area while still providing enough depth for the electronics. The ideal solution in a commercial environment would be to produce a membrane keypad front panel and incorporate the Neuron IC electronics onto the main PCB. However, due to the prototype nature of this project, there was no justification for investing in such a high-cost solution. Instead, the chosen method of construction was to use an aluminium front panel with cut-outs, on which a colour-printed plastic overlay was applied. This overlay has a transparent area through which to view the LCD, and is flexible enough to be able to push the key switches through. Behind this arrangement, the PCB is attached on mounting pillars so that the LCD and key switches sit precisely against the underside of the overlay. The Neuron IC module then connects to the opposite side of the PCB. Figure 5.2. illustrates this construction method in cross-sectional form.

This whole assembly then forms the recessed lid of an enclosure so that the aluminium front panel is supported and retained by four screws. A rubber seal below the front panel provides IP65 protection. The final step in the mechanical design was to mount a connector, for power and network connections, together with a service pin switch in the top face of the enclosure. The end result is an extremely robust device which, while a little bulky, still fits comfortably into one hand. The backlit LCD has a wide viewing angle and the keys have a much more positive click action than a typical membrane keypad.

5.4. Software design

The greatest challenge in the node monitor design was undoubtedly that of the software. Designing a coherent user interface within the confines of a 16 × 2 character display, and then implementing it in the event-driven Neuron C proved particularly difficult.

Perhaps the most obvious way to code the user interface would be for the program to call a different C function depending on which feature the user selects. Each function could basically be a continuous loop to poll keys, updating the LCD as necessary, returning only when a different feature is selected. Thus there would be one function to poll and display network variables, one to receive service pin messages from new nodes, another to scroll through a list of nodes, and so on. The problem with implementing a design of this nature in Neuron C is that it does not respect the event-driven nature of the IC's firmware. As section 3.3 describes, Neuron C programs consist of a series of tasks which are only executed by the scheduler when a particular condition is true. Each task must complete its execution as quickly as possible so that other tasks may use the application processor's time. If any one task used a continuous loop to poll the state of I/O, then the Neuron IC would remain in a continuous wait loop until the watchdog timer reset the operation.

Any successful Neuron C program must therefore be event-driven. Indeed, for many real-time and control applications this requirement makes programs considerably easier to write and more logical in operation. Applying this principle to the node monitor software, it was logical to consider each key press as a separate event. The problem is then designing the software so that such events are handled in the correct way. For example, pressing the 'up' key may increment a number if the user is editing a network variable, but would scroll through a list if the user is selecting which network variable to edit; in other words, the task associated with each key event needs to be different, depending on which software feature is currently being used.

The first step in formulating the solution to this problem, was to breakdown the program operation into a series of 'modes':

Add node, Select node, Select NV, Edit NV, Display NV (default), Setup.

(NV = network variable)

Having done this, each mode was implemented as a separate C function with a global variable *mode* storing the number of the current mode of operation. Each function takes a single parameter *msg*, the 8-bit message being passed to it, and consists of a `switch(msg)` clause containing a case for every message that the function wishes to process. The function then returns a value to indicate the success of the message processing operation. Each of these functions uses a common area of global memory to store variables whose state needs to be maintained between messages.

The next step is to generate the messages to pass to these 'handler' functions. In the case of key presses this was achieved by defining a key scanning routine, executed every 40ms to determine the state of each key switch. Once a key press has been detected, debounced, and processed, the appropriate message is despatched to the current mode handler function. This despatch process merely consists of using *mode* to look up the corresponding mode function pointer in a global array, and then indirectly calling this function, passing the message as the only parameter. Service pin messages are detected by a dedicated task called when an explicit LonTalk message is received on domain zero.

The mechanism by which the mode changes also relies on message passing. When a handler function receives a key press message indicating that the user wishes to change mode, it first de-initialises its own data structures before sending the message MSG_INIT to the handler function for the new mode. Each of the mode functions processes this message, using it to initialise their own data structures and to display appropriate text on the LCD. If, for some reason, the function determines that the mode cannot be selected at that time (e.g. selecting a node before any nodes have been added to the database) then it returns an error value. The calling function checks this return value and only if it indicates a success will it finally change *mode* to that of the new mode, causing all future messages to be directed there instead. This process is also used then the application is reset to put the software into the default mode (Display NV).

The message processing architecture is an extremely powerful one and is particularly well suited to user interfaces and real-time applications. Once the user interface had successfully been implemented and tested, the LonWorks-specific functions were created. The most complicated of these is the function concerned with reading the memory of another node, to determine its network variable structure. It does this by repeatedly sending the NM_read_memory_response to the target node, using Neuron ID addressing. The internal Neuron IC memory map is complex, with many of the data structures being optional and two different network variable structure versions being in use, requiring a great deal of cumbersome program logic to correctly deal with every situation and ensure that the correct byte is read from the correct address. The situation is made more complicated by Echelon's recommendation that no more than 16 bytes be read at a time to ensure that network buffers are not exceeded.

The next difficulty encountered in developing the node monitor software was how to poll, format, display and edit any network variable type, while minimising the amount of specialised code required. A global structure was created, containing the name, number of bytes, units, and number of decimal places for each standard network variable type. The vast majority of SNVTs (Standard Network Variable Type) may be expressed either as `unsigned int`, `unsigned long`, `signed long`, or `float_type`. Additionally there are a large number of SNVT structures that may have several fields of different data types. Another common type is the 8-bit enumeration that can take one of a few possible values, each denoted by a unique text string. Inevitably, there are about ten SNVTs that defy classification and require custom routines. Wherever possible, the polled network variable data is referred to by a pointer in order to use the same routines for every SNVT type. The Neuron C libraries do not include a `printf()` or similar function to format an integer type as an ASCII string, so a special routine had to be written to perform this task, including support for a variable number of decimal places.

The rest of the software was fairly straight-forward to develop, though time-consuming due to the large amount of code and lengthy NodeBuilder download times. An extra feature was included to view the self-documentation information for a network variable (up to 1023 characters) as a scrolling message, as well as displaying the full name of the SNVT type being monitored. It was also discovered that when a large number of nodes were stored in the database, the process of selecting the desired one became rather slow. To rectify this, the sort order of the list is dynamically changed so that the most recently selected nodes appear at the start.

Despite the node monitor's heavy involvement with network variables, the software does not declare any network variables of its own. By relying on polling rather than binding, the node monitor never needs to write to the memory of the node it is monitoring, and was found to be an extremely quick and unobtrusive network tool.

Chapter 6

Relay Setpoint Module Design

6.1. Design aims

The simple aim of designing this module is to allow the variable output from the sensor to trigger external equipment, via a relay, at defined levels - or setpoints. Ideally, such equipment should incorporate a LonWorks interface, allowing it to be 'bound' to the sensor, thus maintaining a total fieldbus system. Realistically, however, not all equipment has yet evolved a fieldbus capability, or is so simple (e.g. a single indicator) that it is not cost-effective. Therefore, there is currently a genuine requirement for this type of module.

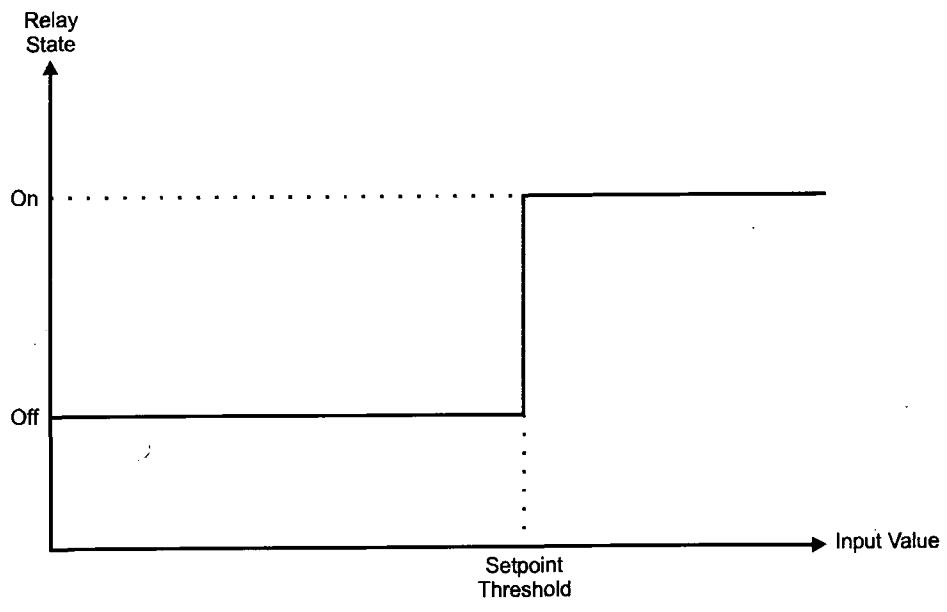


Figure 6.1. The standard behaviour of a single setpoint.

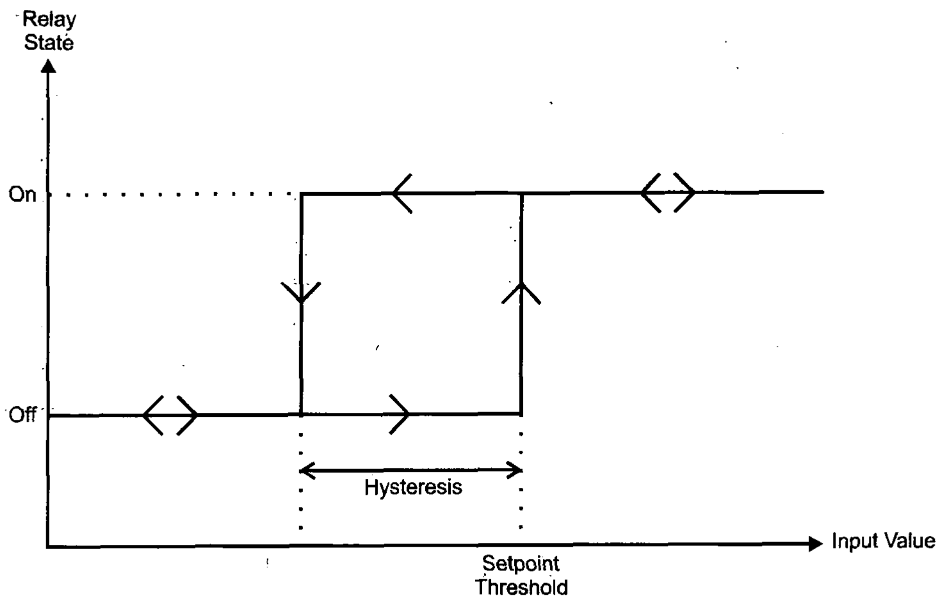


Figure 6.2. A single setpoint, modified to include hysteresis.

Figure 6.1. depicts how this behaviour should work for a single setpoint. While this type of 'threshold switching' operation is well-suited to a wide range of uses, many other applications benefit from the introduction of a hysteresis band. The effect of this is to separate the threshold at which the setpoint turns on from the threshold at which it turns off, as illustrated in figure 6.2. Of course, if the hysteresis is set to zero, the response is equivalent to that shown in figure 6.1.

Having established the type of response required, the specification for the module may be defined as:

- The module should control two setpoints, each having a separate threshold and hysteresis band.

- The module should be a low-cost, DIN-rail mounted device, capable of operating over a wide power supply range.
- Each relay should be able to switch 5A at 30V d.c. and allow the user to connect to normally-open and normally-closed contacts.
- The front-panel should incorporate LEDs to indicate the state of each setpoint and the service pin, along with a push-button to activate the service pin.

6.2. Electronic design

The functionality outlined in the specification was easily implemented using a single Neuron IC. Indeed, because certain Neuron IC I/O lines can drive loads of up to 20mA, the only external circuitry required was a Darlington driver array IC to power the relays, whose coil current is slightly too high to be interfaced directly. Apart from this, the switch-mode power supply used previously was incorporated, along with the standard Neuron IC reset circuit.

The following table summarises how the Neuron IC's eleven I/O lines are allocated:

I/O Line	Direction	I/O Object Type	Purpose
IO0	Output	BIT	Setpoint 1 LED
IO1	Output	BIT	Setpoint 2 LED
IO2	Output	BIT	Setpoint 1 Relay
IO3	Output	BIT	Setpoint 2 Relay
IO4 - IO10			NOT USED

Given the simplistic nature of the circuitry, it was decided that the design and production of a dedicated PCB was not required. Instead, the components were neatly assembled on a small area of 'tripad' stripboard, with an array of connectors along one edge to connect to the front panel and external terminals.

6.3. Mechanical design

An attractive, commercially-available DIN-rail enclosure (55mm × 110mm × 75mm), conforming to EN50-022, was selected and the plastic front panel modified to accept three LEDs and a push switch as shown in figure 6.3. The enclosure incorporates two rows of eight screw terminals, of which ten are internally connected to the horizontally-mounted circuit board.

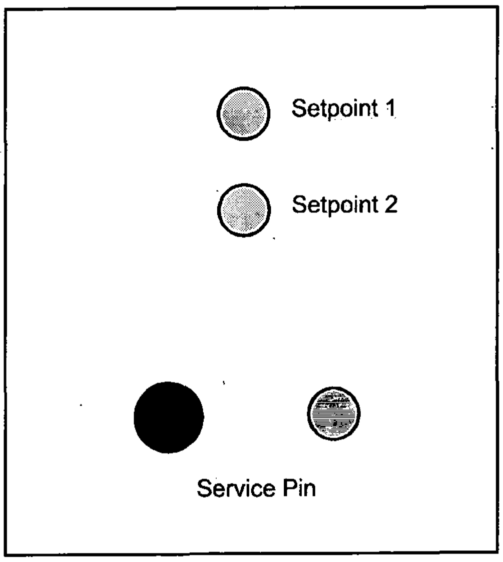


Figure 6.3. The relay setpoint module front panel design.

6.4. Software design

The starting point for the software was deciding on the network variables that would be required:

NV Name	Direction	Type	Purpose
nviValue	Input	Level Percent	Varying input value.
nviSetpoint1	Input	Level Percent	Setpoint 1 threshold.
nviSetpoint2	Input	Level Percent	Setpoint 2 threshold.
nviHysteresis1	Input	Level Percent	Setpoint 1 hysteresis band.
nviHysteresis2	Input	Level Percent	Setpoint 2 hysteresis band.

The type 'Level Percent' was chosen as this is a good general purpose SNVT to use, and matches one of the output network variables from the sensor.

Having established these five input network variables, it becomes obvious that when anyone of them is updated the state of the setpoints must be recalculated. Therefore, a function `UpdateSetpoints()` was created to do just this, and is called from the `when (nv_update_occurs(...))` and `when (reset)` event functions. If there is no hysteresis then the role of `UpdateSetpoints()` is easy - simply compare *nviValue* with both *nviSetpoint1* and *nviSetpoint2*, setting the state of the relevant relay. Examining figure 6.2, it can be seen that the precise threshold at which the setpoint state changes is dependent on the direction in which the value is changing. In other words, when the value is increasing, the setpoint changes from OFF to ON at a different threshold to that at which it changes from ON to OFF when it is decreasing. Initially, the idea of analysing the direction of change may appear difficult to translate into computer code, but in fact the solution is extremely simple. The following three steps illustrate how the algorithm is implemented:

1. If *nviValue* is greater than *nviSetpoint1*, set the setpoint state to ON.
2. If *nviValue* is less than (*nviSetpoint1* - *nviHysteresis1*), set the setpoint state to OFF.
3. Otherwise, make no change to the setpoint state.

It is this final step that effectively implements the hysteresis loop by ensuring that the setpoint maintains its state throughout the hysteresis band, irrespective of the direction of change. The same algorithm is used to calculate the setpoint 2. Appendix E contains the complete program listing.

Chapter 7

Network Operation

7.1. Topology

Each of the three nodes constructed incorporates a free topology twisted-pair network transceiver. The advantage of this transceiver type over others is that the network wiring may be constructed in any topology, such as those shown in figure 7.1. Looped networks are especially useful in fault-tolerant systems because even if one part of the loop is broken, the network will continue to operate. The maximum data rate available with free topology transceivers is 78 kbits/s with a maximum distance of 500m. Routers and repeaters may be used to extend this distance and to interface the network to other networks based on different transceivers.

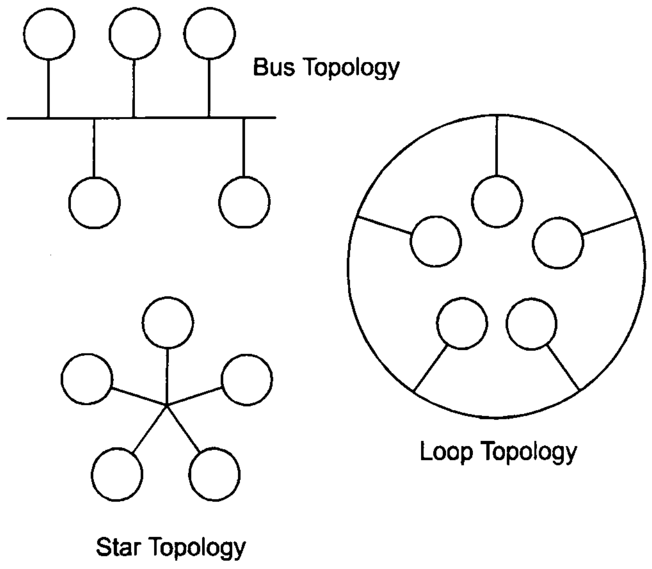


Figure 7.1. Different network topologies possible with free topology transceivers.

During the course of the experimentation, standard two pair cable was used - one twisted pair carried the network communications while the other pair carried 24V d.c. Simple DIN connectors were then used to connect each node to a 'splitter box', along with connections from the PC and the NodeBuilder. The idea of this was to easily enable each node to be physically connected/disconnected to/from the network. The total length of network cable was relatively short and no problems were encountered.

7.2. Network management

The network topology determines the electrical connections between nodes. In order to control the logical connections, a network management tool is required. Such tools can be subdivided into installation, diagnostic, maintenance and development categories. In this context, installation is defined as the process of loading a node with address, binding, and configuration data - it does not include the physical installation of the node.

No network management tool is included with the NodeBuilder development system, so it was not initially possible to test the operation of the nodes with each other by binding network variables. In an attempt to find a solution to this problem, several third party tools that claim to fulfil this role were tested to assess their suitability. Not only is such a tool vital in a development environment, but in order for Partech to be able to supply a complete system, an appropriate network management tool must be included.

A good network management tool should be able to perform the majority of the following tasks:

- Discovery of new nodes as they are physically attached to the network.
- Network deconfiguration.
- Node commissioning.
- Receiving service pin messages.
- Importing node self-documentation and self-identification information.
- Importing node external interface files.
- Copying configuration network variable values from one node to another.
- Installing, removing, and replacing nodes.
- Connecting and disconnecting network variables and message tags.
- Loading application images into nodes.
- Querying and setting node properties, such as locations, priority slots, self-documentation, and network variable attributes.
- Resetting nodes.
- Winking nodes.
- Testing nodes.

Additionally, the software package should be intuitive, and easy to use in the field on a laptop computer. Three different packages were assessed: LonMaker™ (Echelon), EasyLon Toolbox Lite (Gesytech), and Visual Control (Dayton General Systems).

LonMaker

Echelon's own LonMaker tool is designed to support all the steps required to install and maintain LonWorks networks of up to 2540 devices, including node installation, establishing node connections, network commissioning, and network maintenance. The software relies on a precise description of the network, in terms of domains, channels, locations, routers and devices, being stored in a database on the PC's hard disk. The first step in installing a network is to use a separate application, LonProfiler, to construct a parts catalogue of all the types of devices that need to be installed. This information is then exported into LonMaker, from where the user can install each node into the network, choosing its type from the predefined list of parts. Binding

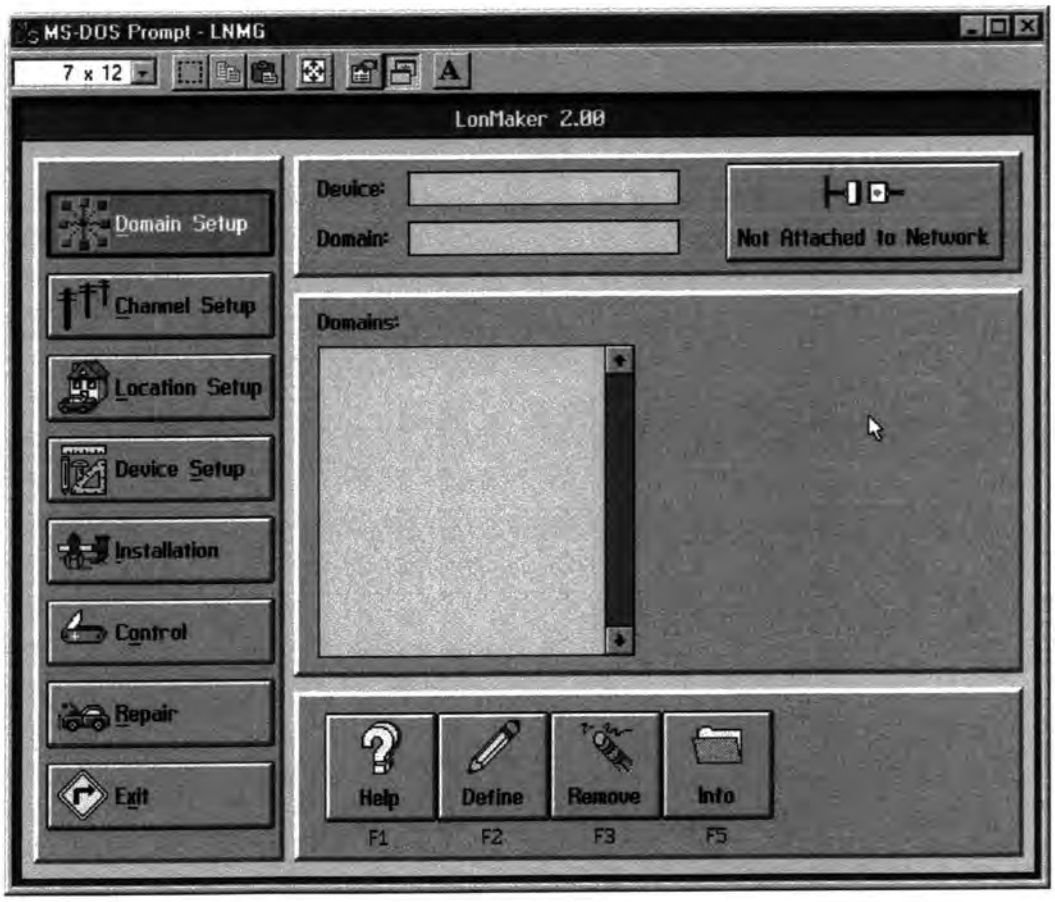
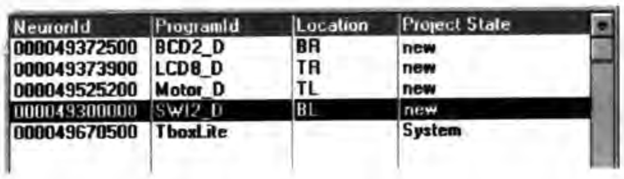


Figure 7.2. The MSDOS-based LonMaker network management software.

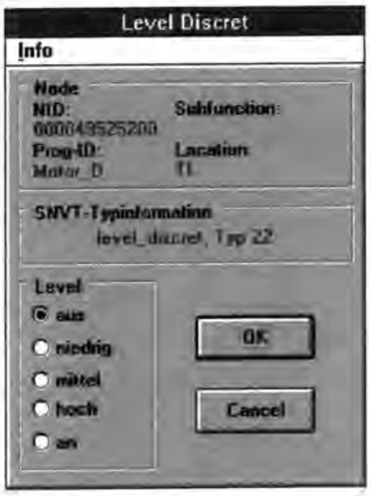
takes the form of selecting pairs of network variables from lists, before finally implementing the connections. Faulty nodes may be replaced, and any network variable viewed or modified.

Considering the user interface of the Windows-based NodeBuilder application, it is surprising to find that LonMaker is an MSDOS application, and unable to operate simultaneously with the NodeBuilder software. As figure 7.2. shows, it has an attractive user interface but, in practice, this is extremely cumbersome and time consuming to use. LonProfiler is an even more primitive character-based application. Having said this however, the software proved to be extremely reliable and performed all the tasks it claimed to be able to do.



NeuronId	ProgramId	Location	Project State
000049372500	BCD2_D	BR	new
000049373900	LCD8_D	TR	new
000049525200	Motor_D	TL	new
000049300000	SWI2_D	BL	new
000049670500	TboxLite		System

Selecting a node on the network.



Level Discret

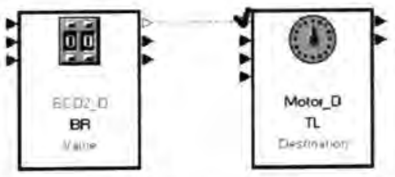
Info

Node
NID: 000049525200
Subfunction:
Prog-ID: Motor_D
Location: TL

SNVT-Typinformation
level_discret, Typ 22

Level
[✓ aus]
niedrig
mittel
hoch
an

Editing a network variable.



Graphically binding network variables.

Figure 7.3. Some of the elements that make up Easylon Toolbox Lite.

Easylon Toolbox Lite

Gesytec's Easylon product is a collection of six separate Windows applications, designed to work together on the same project, but each with its own function:

- The *Network info* tool is used to query node information from the network.
- The *Node Information* tool allows investigating and changing the status of individual nodes. This can be used to check the correct installation of an application onto the network.
- The *Network Variable Information* tool allows values of network variables to be read and modified on individual nodes.
- The *Graphical Binder* provides a graphical representation of available nodes, allowing connections to be drawn between one output and one or more input network variables.
- The *List Binder* allows connections to be made between one output and one or more input network variables by 'drag-and-drop' editing of a list of available network variables from all available nodes in the network.
- The *Loader* tool is used to bind network variables and to download an application to a node in the network.

Curiously, it is not possible to run two of these applications simultaneously and, together with the fact that each application appears to have been produced with Visual Basic, the whole system feels a little disjointed and clumsy to operate. Performance and reliability was not as good as LonMaker, although the graphical binder allows network variable connections to be handled more intuitively. Figure 7.3. illustrates some of Easylon's features.

Visual Control

Dayton General Systems' Visual Control product differs from the previous two because its primary use is not as a general network management tool. The software allows the function of a node to be defined visually by graphically combining different control elements. Several predefined math, control, and logic devices are available. The inputs and outputs of each element may be linked together, and additional network variables defined. Once the design of a node is complete, the graphical representation of its function is translated into C code, compiled, and downloaded. In this way, LonWorks nodes can be created without investing in the NodeBuilder development tool.

The reason for evaluating Visual Control is that it also incorporates a simple network management module to control nodes and to bind network variables. The 'bind wizard' (see figure 7.4) is similar in operation to the list binder in Easylon. The first step is to select a node on the network, and then one of that node's output network variables. Next, a second node is selected, causing its input network variables that match the selected output type to be displayed. Having decided on the pair of variables to be bound, the final stage is to update the binding information table in each node. While simple in operation, the elegant user interface and fast response of the software make it the easiest and most intuitive of the three packages to use. Visual Control is also written to take advantage of Echelon's LonWorks Network Services (LNS), which is the next-generation platform for designing client-server network management tools.

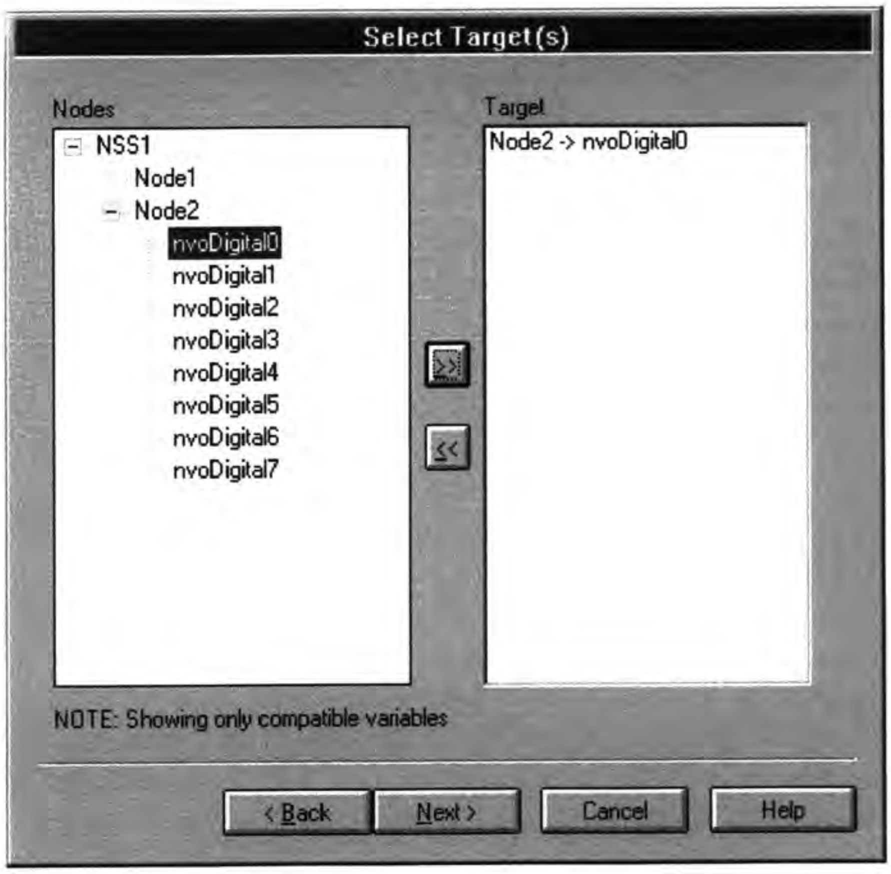


Figure 7.4. Visual Control's Bind Wizard is used to define network variable connections.

7.3. Network use

The sensor node, node monitor, and relay setpoint module were electrically connected as described in section 7.1. to form a self-contained control network. The node monitor could be used to zero and calibrate the sensor, and to monitor its output values. By using one of the network management tools described, the sensor node's ouput network variable *nvoPercentage* could be bound to the relay module's input network variable *nviValue*. Once bound, a small increase in the suspended solids content of the solution being monitored by the sensor caused a relay to activate. A further increase triggered the second relay. Thus, a LonWorks control network was performing the same role as that of a traditional Partech monitoring system.

In order to demonstrate a trivial level of interoperability, LonWorks control modules from other manufacturers were added to the network. These consisted of two Smart Controls modules, offering a variety of analogue and digital inputs and outputs, along with two Weidmuller DIN-rail modules, providing digital inputs and additional relay outputs. The nodes were easily added into the node monitor's internal database, after which the self-documentation and network variable information of each node could be viewed. Various combinations of network variables between the new modules and the three original nodes were bound. Visual Control was also used to create new software for one of the Smart Controls module, providing a simple means of overriding the sensor's control of the relay module via a push switch. This level of power and flexibility would not have been possible with the previous control systems supplied by Partech Instruments.

Chapter 8

Conclusions

The primary aim of this project was to construct a working prototype of a water quality monitoring system based on a fieldbus communications protocol. Clearly, that has been achieved. Within the context of the experimental work conducted, LonWorks has shown itself to be a robust and reliable solution to the problem of creating a distributed control system. The final prototype network demonstrated that the traditional Partech Instruments 'one sensor, one dedicated controller' system can be matched by fieldbus-based products, both in terms of features and performance. In addition, the level of flexibility and interoperability offered by LonWorks allows systems to evolve and expand in a controlled way. By removing centralised controllers, instead distributing the control strategy over an entire network of intelligent nodes, the system acquires an inherent fault-tolerance. Any centralised control room simply becomes a place to monitor and update parameters, rather than a critical centre of communications.

The sensor design has been shown to work at a basic level, and that its performance is at least as good as Partech's existing sensor of that type. Undoubtedly, much work is required to enhance the design and to explore the potential benefits of the electronics that are not currently utilised. An example of this is the potential to incorporate light scatter measurements. There is a great deal of scope for improving the sensor's software, particularly devising an algorithm for dynamically modifying the LED intensity. Also, by more intelligent monitoring of the two photodiode signals it would be possible to detect abnormal fouling, LED failure, or if the ambient light level was too high. More advanced signal processing would further improve the sensor's response, and for portable applications, the power consumption could be reduced significantly. The ability to store all calibration information within the actual sensor is a great advantage over previous designs, as it allows the sensor to be easily moved to a different point on the network without affecting its operation. Once a standard network variable type has been created for suspended solids, the sensor will become a truly interoperable device.

The node monitor proved to be an unexpected benefit of the experimental work. This device provided an invaluable insight into the state of many nodes, something that was often difficult or not possible to achieve with Echelon's own development system. Given its universal functionality and low cost, the node monitor could be an extremely marketable product, suitable for many applications. A more sophisticated version could incorporate a graphical display which would allow for a more intuitive user interface with the addition of more features, such as the ability to bind network variables together. There appears to be a real need for such a hand-held network management tool.

The main point illustrated by the construction of the relay setpoint module is just how easy it can be to develop a complete, working LonWorks node. By using an off-the-shelf Neuron IC module, the external electronics required is minimal. Also, because of the event driven nature of Neuron C, constructing real-time control systems is much more straightforward than on other embedded platforms. The role performed by the relay module, while essential for a working suspended solids monitoring system, could also be done by a variety of other third-party products. This fact demonstrates that interoperability is a reality for LonWorks systems, and that the increased choice is an advantage for the end user. In the future, the need for general-purpose I/O modules will decrease as more products, such as valves, switches, and temperature sensors, become native interoperable LonWorks nodes in their own right.

Currently, by far the weakest link in a LonWorks control system is the network management tool. Of the tools examined, none reached a standard of performance and, particularly, ease of use that would be required in most applications. Before distributed control solutions can be offered as a realistic alternative to current technology, the quality of these tools must improve. Having said this, there are alternative software packages available and many new products are in development. As Visual Control showed, the next generation of LNS products is likely to be a significant improvement.

Throughout the project, it has been shown how a complex communications protocol, implemented using VLSI integrated circuits, can offer a deceptively simple and elegant solution to control networking problems. Comparing such technology with the standard 4-20mA protocol and PLCs currently used in industry is like comparing a modern PC to a mainframe computer of twenty years ago. Without digital communications, the control industry will remain in the dark ages of analogue control. Ultimately, fieldbus will enable greater manufacturing flexibility, productivity, higher quality products, and improved regulatory compliance. Control networks are destined to become an essential part of the fabric of connection and co-operation.

References

1. BSI. Water Quality Part2. *Physical, Chemical and Biochemical Methods. Section 2.13. Determination of turbidity.* BS 6068:Section 2.13:1984.Van de Hulst H.C. *Light scattering by small particles.* Dover. New York. 1981.

2. *Interoperable control systems.* The LonMark Interoperability Association. 1994.

3. *Fieldcomms'95 conference report.* 1995.

4. *The SNVT Master List and Programmer's Guide.* Echelon Corporation. October 1995.

5. *LonWorks Technology Device Data.* DL159/D Rev1. Motorola. 1996.

6. Van de Hulst H.C., *Light Scattering by Small Particles.* Dover. New York. 1981.

7. Rawlinson E. *Filtration Optimisation.* Process Industry Journal. pp40-1. October 1992.

8. Hart V.S., Johnson C.E., Letterman R.D. *An analysis of Low Level Turbidity Measurements.*
J AWWA. pp40-45. 1992. 84. No12.

9. Standing Committee of Analysts. *Ultraviolet and Visible Solution Spectrophotometry and Colorimetry.* 1980. HMSO.

10. *LonTalk Protocol.* Echelon Corporation. April 1993.

11. *Enhanced Media Access Control with LonTalk Protocol.* Echelon Corporation. January 1995.

12. *Neuron C Programming Guide.* Echelon Corporation. 1995.

13. Tipler P. *Physics.* 3rd Edition. pp 1022. Worth. 1991.

14. *Data Converters Design-in Reference Manual.* Analog Devices. 1994.

15. *PIC Device Data.* Arizona Microchip. 1995.

Bibliography

Martin J.D. *Signals and Processes A foundation Course.* Pitman. London. 1991.

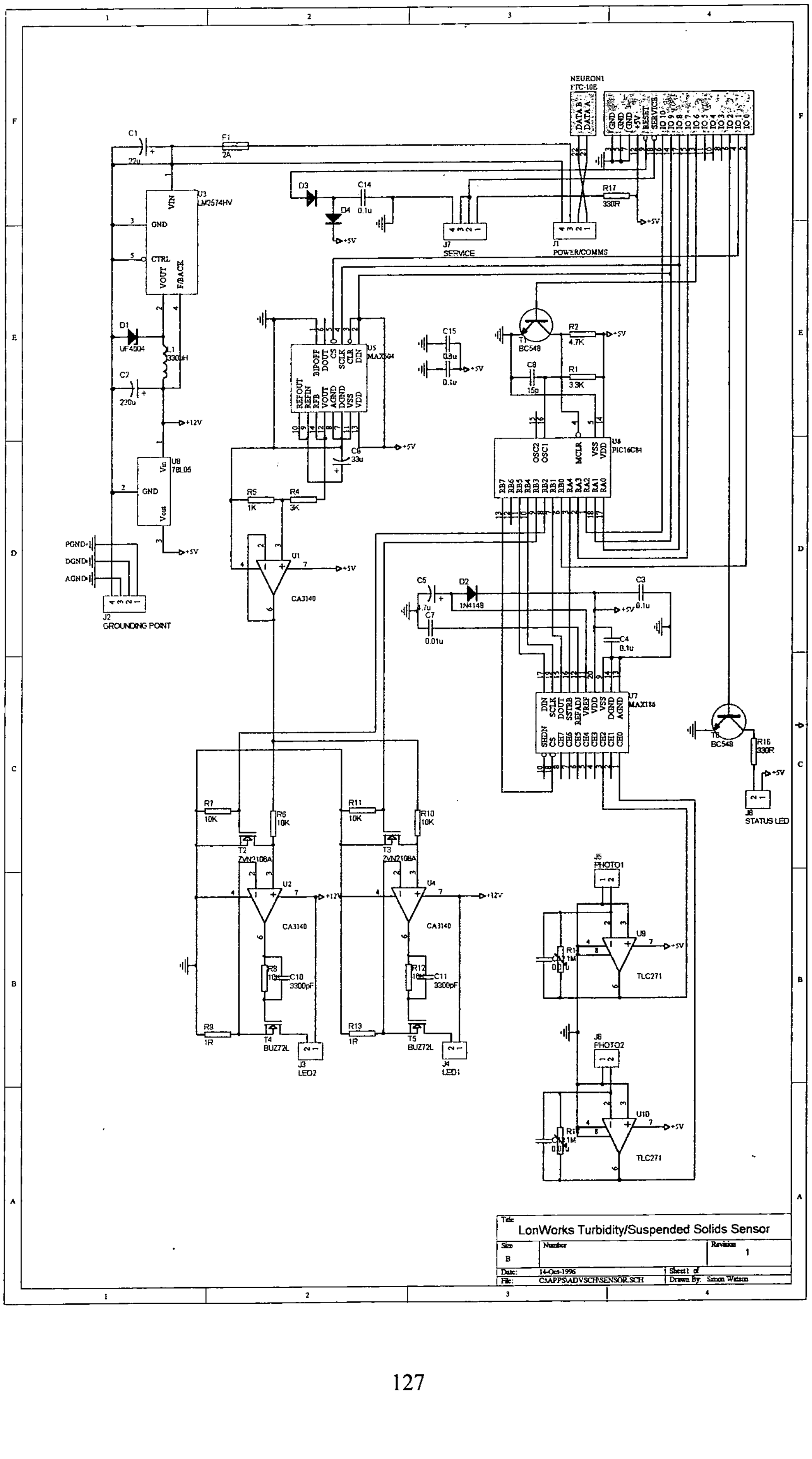
Van de Hulst H.C. *Light scattering by small particles.* Dover. New York. 1981.

Lynn P.A. *An introduction to the analysis and processing of signals.* 2nd Ed. Macmillan. London. 1982.

LonWorks Technology Device Data. DL159/D. Rev1. Motorola. 1996.

Martin J.D. *Signals and Processes A foundation Course.* Pitman. London. 1991.

Appendix A - Sensor Schematic



Appendix B - Sensor PIC Program

```
;***************************
;******* SENSOR.ASM ********
;***************************

;all values stored LSB first
;serial communciations = MSB first

        LIST    P=PIC16C84, F=INHX8M
        INCLUDE "P16CXX.INC"
        RADIX   DEC

;---declare symbols---
;PORT B bits
INT     equ     0
DIN     equ     1
LED1    equ     2
LED2    equ     3
DCLKOUT equ     4
DOUT    equ     5
CS_DAC  equ     6
CS_ADC  equ     7

;PORT A bits
SCLKIN  equ     0
SIN     equ     1
SOUT    equ     2
SREADY  equ     3
BUSY    equ     4

;port details
PORTA_DEFAULT   equ     B'01000'
PORTB_DEFAULT   equ     B'11000000'
PORTA_DDR       equ     B'10011'
PORTB_DDR       equ     B'00000011'
;program variables
VA      equ     0x0c
VB      equ     0x0d
VC      equ     0x0e
VD      equ     0x0f
COUNT           equ     0x10
C0_TOTAL        equ     COUNT + 2
C90_TOTAL       equ     C0_TOTAL + 4
B0_TOTAL        equ     C90_TOTAL + 4
B90_TOTAL       equ     B0_TOTAL + 4
STAGE           equ     B90_TOTAL + 4
ADC_RESULT      equ     STAGE + 1
AMBIENT         equ     ADC_RESULT + 2

;---declare macros---
PAGE0   macro
        bcf     STATUS, RP0
        endm
PAGE1   macro
        bsf     STATUS, RP0
        endm

;---code---
;////////////////////////////////////
;reset address
        org     0
        goto    start

;////////////////////////////////////
;interrupt address
      org    4
```

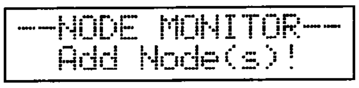
```
        btfsc   PORTA, SIN              ;check action rquired
        goto    interrupt_reading

;.......................................
interrupt_comms
;get command code from host
;        movlw   VD
;        movwf   FSR
;        movlw   1
;        movwf   VA
;        call    host_rx

;upload average data
command1
        movlw   COUNT+17
        movwf   FSR
        movlw   18
        movwf   VA
        call    host_tx

        call    reset_store             ;reset count and totals
        goto    interrupt_end

;.......................................
interrupt_reading

        bcf     PORTB, CS_ADC           ;enable ADC

;reset stage counter
        movlw   0
        movwf   STAGE

;transmit stage0 C7-C4
        movlw   8
        call    adc_write_nibble

;start of loop for each measurement stage
stage_start

;setup LEDs
        movlw   HIGH led_control
        movwf   PCLATH
        movf    STAGE, 0
        call    led_control             ;get LED states into W
        movwf   PORTB                   ;output LED states

;transmit stage C3-C0 to start conversion
        movlw   B'00001110'
        call    adc_write_nibble

;turn-off LEDs
        movlw   0
        movwf   PORTB

;wait until ADC no longer busy
adc_wait
        btfss   PORTA, BUSY
        goto    adc_wait

;read conversion result & store as 12-bit value
        call    adc_read_byte           ;read B11-B4
        movwf   ADC_RESULT              ;load into temporary LSB
        movwf   ADC_RESULT + 1          ;load into temporary MSB
        swapf   ADC_RESULT + 1, 1
        movlw   B'00001111'
        andwf   ADC_RESULT + 1, 1
        andwf   ADC_RESULT, 1

        movlw   HIGH adc_control        ;get C7-C4 of next stage
        movwf   PCLATH
        incf    STAGE, 0
        call    adc_control
        call    adc_read_write_nibble   ;read B3-B0 & write C7-C4
        swapf   ADC_RESULT, 1
        iorwf   ADC_RESULT, 1

;determine if reading is ambient (no total)
        movlw   HIGH total_index
        movwf   PCLATH
        movf    STAGE, 0
        call    total_index             ;get total index into W
        movwf   FSR
        btfss   FSR, 7
        goto    stage_compensate

;store adc reading as ambient
        movf    ADC_RESULT, 0
        movwf   AMBIENT
        movf    ADC_RESULT+1, 0
        movwf   AMBIENT+1
        goto    stage_next

;compensate for ambient light
stage_compensate
        goto    stage_total

        movf    AMBIENT, 0
        subwf   ADC_RESULT, 1           ;subtract first digit
        btfss   STATUS, C               ;check carry
        incf    AMBIENT+1, 1
        movf    AMBIENT+1, 0
        subwf   ADC_RESULT+1, 1         ;subtract second digit
        btfsc   STATUS, C               ;check carry
        goto    stage_total

;trap negative result
        movlw   0                       ;zero ADC_RESULT
        movwf   ADC_RESULT
        movwf   ADC_RESULT+1

;calculate address of corresponding total
stage_total
        movlw   C0_TOTAL
        addwf   FSR, 1
        call    add_adc                 ;add ADC_RESULT to total

;increment STAGE, and repeat loop if < 6
stage_next
        incf    STAGE, 1                ;increment STAGE and keep change
        movf    STAGE, 0
        sublw   5
        btfsc   STATUS, C
        goto    stage_start             ;repeat loop

        bsf     PORTB, CS_ADC           ;de-select ADC
        call    increment_count         ;increment 16-bit counter

interrupt_end
        bcf     INTCON, INTF            ;clear interrupt flag
        RETFIE                          ;return from interrupt


;/////////////////////////////////////
;start address
      org     100
start
;intialise registers
        PAGE0
        movlw   B'00010000'             ;interrupt enables
        movwf   INTCON
        movlw   PORTA_DEFAULT           ;reset PORTA outputs
        movwf   PORTA
        movlw   PORTB_DEFAULT           ;reset PORTB outputs
        movwf   PORTB
        PAGE1
        movlw   B'11000000'             ;device options
        movwf   OPTION_REG - 0x80
        movlw   PORTA_DDR               ;set PORTA I/O setup
      movwf TRISA - 0x80
        movlw   PORTB_DDR               ;set PORTB I/O setup
        movwf   TRISB - 0x80
        PAGE0
        call    reset_store             ;reset count and totals

        bsf     INTCON, GIE             ;enable global interrupts

sleep_loop
        sleep                           ;enter 'sleep' mode
        goto    sleep_loop

;/////////////////////////////////////
adc_write_nibble
;Transmits four bits to ADC
;W = lower nibble contains bits for transmission
;CORRUPTS: VA

        bcf     PORTB, DCLKOUT          ;initial condition
        movwf   VA

        bcf     PORTB, DOUT
        btfsc   VA, 3                   ;bit 3
        bsf     PORTB, DOUT
        bsf     PORTB, DCLKOUT
        bcf     PORTB, DCLKOUT

        bcf     PORTB, DOUT
        btfsc   VA, 2                   ;bit 2
        bsf     PORTB, DOUT
        bsf     PORTB, DCLKOUT
        bcf     PORTB, DCLKOUT

        bcf     PORTB, DOUT
        btfsc   VA, 1                   ;bit 1
        bsf     PORTB, DOUT
        bsf     PORTB, DCLKOUT
        bcf     PORTB, DCLKOUT

        bcf     PORTB, DOUT
        btfsc   VA, 0                   ;bit 0
        bsf     PORTB, DOUT
        bsf     PORTB, DCLKOUT

        bcf     PORTB, DCLKOUT          ;final conditions
        bcf     PORTB, DOUT
        return

;/////////////////////////////////////
adc_read_byte
;Receive 8 bits from ADC
;CORRUPTS: Nothing
;RETURNS: Received byte in W

        bcf     PORTB, DOUT             ;initial conditions
        bsf     PORTB, DCLKOUT
        movlw   0

        bcf     PORTB, DCLKOUT
        btfsc   PORTB, DIN              ;bit 7
        iorlw   128
        bsf     PORTB, DCLKOUT
        bcf     PORTB, DCLKOUT
        btfsc   PORTB, DIN              ;bit 6
        iorlw   64
        bsf     PORTB, DCLKOUT

        bcf     PORTB, DCLKOUT
        btfsc   PORTB, DIN              ;bit 5
        iorlw   32
        bsf     PORTB, DCLKOUT

        bcf     PORTB, DCLKOUT
        btfsc   PORTB, DIN              ;bit 4
        iorlw   16
        bsf     PORTB, DCLKOUT

        bcf     PORTB, DCLKOUT
        btfsc   PORTB, DIN              ;bit 3
        iorlw   8
        bsf     PORTB, DCLKOUT

        bcf     PORTB, DCLKOUT
        btfsc   PORTB, DIN              ;bit 2
        iorlw   4
        bsf     PORTB, DCLKOUT

        bcf     PORTB, DCLKOUT
        btfsc   PORTB, DIN              ;bit 1
        iorlw   2
        bsf     PORTB, DCLKOUT

        bcf     PORTB, DCLKOUT
        btfsc   PORTB, DIN              ;bit 0
        iorlw   1

        return


;/////////////////////////////////////
adc_read_write_nibble
;Receive/Transmit 4 bits to/from ADC
;W = Nibble to transmit in lower 4 bits
;CORRUPTS: VA
;RETURNS: Received nibble in lower 4 bits of W

        bcf     PORTB, DCLKOUT          ;initial conditions
        movwf   VA
        movlw   0

        bcf     PORTB, DOUT
        btfsc   VA, 3                   ;transmit bit 3
        bsf     PORTB, DOUT
        bsf     PORTB, DCLKOUT
        bcf     PORTB, DCLKOUT
        btfsc   PORTB, DIN              ;receive bit 3
        iorlw   8

        bcf     PORTB, DOUT
        btfsc   VA, 2                   ;transmit bit 2
        bsf     PORTB, DOUT
        bsf     PORTB, DCLKOUT
        bcf     PORTB, DCLKOUT
        btfsc   PORTB, DIN              ;receive bit 2
        iorlw   4

        bcf     PORTB, DOUT
        btfsc   VA, 1                   ;transmit bit 1
        bsf     PORTB, DOUT
        bsf     PORTB, DCLKOUT
        bcf     PORTB, DCLKOUT
        btfsc   PORTB, DIN              ;receive bit 1
        iorlw   2
        bcf     PORTB, DOUT
        btfsc   VA, 0                   ;transmit bit 0
        bsf     PORTB, DOUT
        bsf     PORTB, DCLKOUT
        bcf     PORTB, DCLKOUT
        btfsc   PORTB, DIN              ;receive bit 0
        iorlw   1

        return


;////////////////////////////////////
host_tx
;VA = no. bytes to transmit
;FSR = address of last byte in transmit buffer
;CORRUPTS: W, VB

        bcf     PORTA, SREADY

;start next byte
ht_loop0
        movlw   8
        movwf   VB

;start next bit
ht_loop1
        bcf     PORTA, SOUT

;set SOUT

        rlf     0, 1
        btfsc   STATUS, C
        bsf     PORTA, SOUT

ht_loop2
        btfss   PORTA, SCLKIN           ;wait for clock to go high
        goto    ht_loop2

ht_loop3
        btfsc   PORTA, SCLKIN           ;wait for clock to go low
        goto    ht_loop3

        decfsz  VB, 1                   ;loop round for next bit (if
required)
        goto    ht_loop1

        decf    FSR, 1        ;adjust pointer to next byte for transmission
        decfsz  VA, 1         ;decrement no. bytes
        goto    ht_loop0      ;loop around for next byte (if required)

ht_loop4
        btfss   PORTA, SCLKIN           ;wait for clock to go high
        goto    ht_loop4
        bsf     PORTA, SREADY
ht_loop5
        btfsc   PORTA, SCLKIN           ;wait for clock to go low
        goto    ht_loop5

        return

;////////////////////////////////////
host_rx
;VA = no. bytes to receive
;FSR = address of last byte in receive buffer
;CORRUPTS: W, VB

        bcf     PORTA, SOUT

;start next byte
hr_loop0
```

```
        movlw   128
        movwf   VB
        movlw   0
        movwf   0
        bcf     PORTB, LED2
        bcf     PORTA, SREADY

;start next bit
hr_loop2
        btfss   PORTA, SCLKIN           ;wait for clock to go high
        goto    hr_loop2

;get SIN
        btfsc   PORTA, SIN
        goto    hr_set
hr_clear
        goto    hr_loop3
hr_set
        movf    VB, 0
        iorwf   0, 1

hr_loop3
        btfsc   PORTA, SCLKIN           ;wait for clock to go low
        goto    hr_loop3

        bcf     STATUS, C
        rrf     VB, 1
        btfss   STATUS, C
        goto    hr_loop2

        decf    FSR, 1          ;adjust pointer to next byte for
transmission
        decfsz  VA, 1               ;decrement no. bytes
        goto    hr_loop0            ;loop around for next byte (if required)

hr_loop4
        btfss   PORTA, SCLKIN           ;wait for clock to go high
        goto    hr_loop4
        bsf     PORTA, SREADY
hr_loop5
        btfsc   PORTA, SCLKIN           ;wait for clock to go low
        goto    hr_loop5

        return

;/////////////////////////////////////
;Set COUNT, all C0, C90, B0, B90 TOTALS, and STAGE = 0
;CORRUPTS: VA, W, FSR
;
reset_store
        movlw   STAGE
        movwf   FSR
        movlw   STAGE-COUNT+1
        movwf   VA
        movlw   0
rs_loop1
        movwf   0
        decf    FSR, 1
        decfsz  VA, 1
        goto    rs_loop1

        return;

;/////////////////////////////////////
add_adc
;Add ADC_RESULT to 32-bit value addressed by FSR
;FSR = Address of total to use
;CORRUPTS: FSR, W

;1st byte
        movf    ADC_RESULT, 0           ;get 1st byte
        addwf   0, 1                    ;add byte to total
        btfsc   STATUS, C               ;check for carry
        incf    ADC_RESULT+1, 1

;2nd byte
        incf    FSR, 1
        movf    ADC_RESULT+1, 0         ;get 2nd byte
        addwf   0, 1                    ;add byte to total
        btfss   STATUS, C               ;check for carry
        return;

;3rd byte
        movlw   1
        incf    FSR, 1
        addwf   0, 1                    ;increment 3rd byte
        btfss   STATUS, C               ;check for carry
        return;

;4th byte
        incf    FSR, 1
        addwf   0, 1                    ;increment 4th byte

        return;


;//////////////////////////////////////
;Increment 16-bit counter
increment_count
        movlw   1
        addwf   COUNT, 1
        btfsc   STATUS, C
        incf    COUNT+1, 1
        return;

;//////////////////////////////////////
;---declare data tables---
        org 0x3e0

led_control
        addwf   PCL, 1
        retlw   0
        retlw   4
        retlw   8
        retlw   0
        retlw   8
        retlw   4

adc_control
        addwf   PCL, 1
        retlw   8
        retlw   8
        retlw   8
        retlw   9
        retlw   9
        retlw   9
        retlw   8

total_index
        addwf   PCL, 1
        retlw   128
        retlw   0
        retlw   4
        retlw   128
        retlw   8
        retlw   12

;//////////////////////////////////////
        end
```

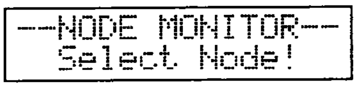
Appendix C - Node Monitor User Guide

The default node monitor mode of operation is that used to display the selected network variable. Pressing EXIT in any mode will return the user to this point.

If no nodes are in the database then 'Add Node!' is displayed, and once nodes are added 'Select Node!' is shown.

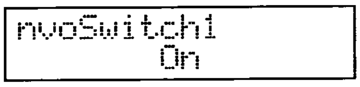


```
--NODE MONITOR--
  Add Node(s)!
```

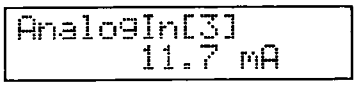


```
--NODE MONITOR--
  Select Node!
```

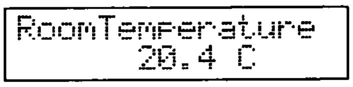
Once a network variable has been selected, this mode will display the name of the network variable on the top line, with its current value on the bottom line of the LCD.



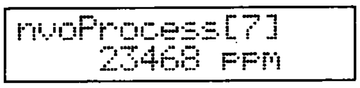
```
nvoSwitch1
       On
```



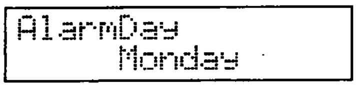
```
AnalogIn[3]
       11.7 mA
```



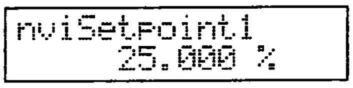
```
RoomTemperature
       20.4 C
```



```
nvoProcess[7]
    23468 FPM
```

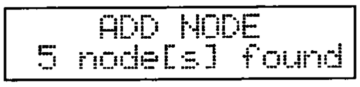


```
AlarmDay
       Monday
```



```
nviSetpoint1
      25.000 %
```

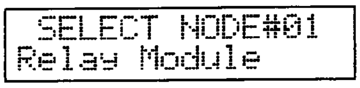
Add Node



```
    ADD NODE
 5 node[s] found
```

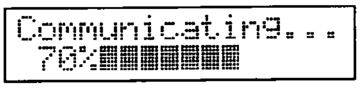
This mode allows new nodes to be added to the node monitor's internal database. Every time a service pin message is received from a node, the count of nodes found will be increased. When OK is pressed, the new nodes are integrated into the database, with duplicates automatically discarded.

Select Node



```
 SELECT NODE#01
Relay Module
```

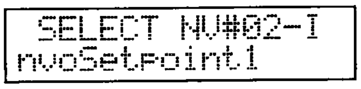
To select a node store in the node monitor's internal database, use UP and DOWN to scroll through the list of available node names. The index number at the end of the top line indicates the current position in the list. Once the desired node has been located, press OK to confirm this choice. The node monitor will now communicate with the chosen node and retrieve its network variable information. A bargraph indicates the progress of this process.



```
Communicating...
 70%████████
```

When completed, the node monitor will return to the default mode and begin monitoring the first network variable in the list.

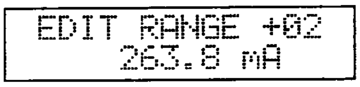
Select Network Variable



```
 SELECT NV#02-I
nvoSetPoint1
```

To select a different network variable to the one being monitored, use UP and DOWN to scroll through the list of available variables. The index number at the end of the top line indicates the current position in the list, and whether that network variable is an input or output type. Once the desired network variable has been located, press OK to confirm this choice. The node monitor will immediately begin monitoring this variable instead.

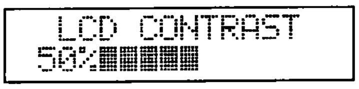
Edit Network Variable



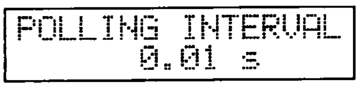
```
  EDIT RANGE +02
     263.8 mA
```

When this option is selected, the last polled value of the current network variable is frozen and displayed on the bottom line of the display. The UP and DOWN keys increment and decrement the value in steps of 10^n, where n is the edit range number shown on the top line of the display. The value of n may be adjusted by holding down FUNCTION while pressing UP or DOWN. Pressing EXIT discards any changes made, while OK updates the network variable with the edited value.

Setup



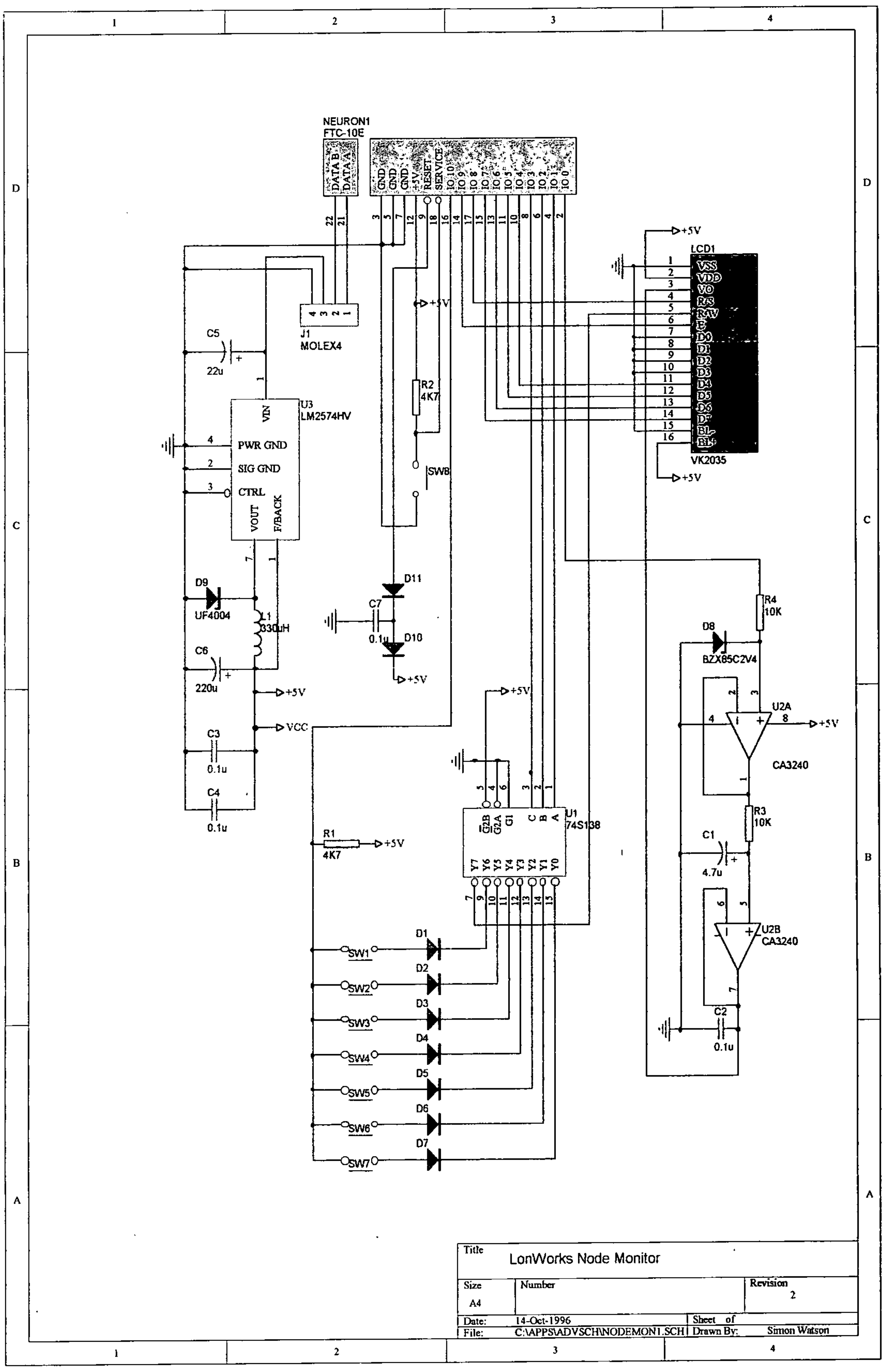
```
  LCD CONTRAST
 50%■■■■■
```



```
POLLING INTERVAL
      0.01 s
```

Repeatedly pressing SETUP steps through each available setup option in turn. For each option, use UP and DOWN to alter the value, followed by OK to make the change.

Appendix D - Node Monitor Schematic



Appendix E - Relay Module Neuron C Program

```
//RELAY.NC

#include <snvt_lev.h>
#include <access.h>
#include <float.h>

//---define constants---
#define NORMAL_OFF        0
#define NORMAL_ON         1
#define INVERTED_OFF            1
#define INVERTED_ON       0
#define NV_TYPE_VALIDATE  12345
#define NV_TYPE_DEFAULT         52

//---type definitions---
typedef unsigned char BYTE;
typedef unsigned int UINT;
typedef unsigned long ULONG;

//---assign IO pins---
IO_0 output bit io_led1 = INVERTED_OFF;
IO_1 output bit io_led2 = INVERTED_OFF;
IO_2 output bit io_relay1 = INVERTED_OFF;
IO_3 output bit io_relay2 = INVERTED_OFF;

//---network variables---
network input SNVT_lev_percent nviValue;
network input SNVT_lev_percent nviSetpoint1 = 5000;
network input SNVT_lev_percent nviSetpoint2 = 10000;

//--global variables---

//---timers---

//---function prototypes---
void UpdateSetpoints(void);


////////////////////////////////
when (reset)
{
UpdateSetpoints();
}


when (nv_update_occurs(nviValue))
{
UpdateSetpoints();
}


when (nv_update_occurs(nviSetpoint1))
{
UpdateSetpoints();
}


when (nv_update_occurs(nviSetpoint2))
{
UpdateSetpoints();
}


when (wink)
{
BYTE n;
for (n = 0; n < 5; n++)
	{
	io_out(io_led1, 1);
	io_out(io_led2, 0);
	delay(5000);
	io_out(io_led1, 0);
	io_out(io_led2, 1);
	delay(5000);
	}
UpdateSetpoints();
}


void UpdateSetpoints(void)
{
//calculate setpoint1
if (nviValue < nviSetpoint1)
	{
	io_out(io_relay1, INVERTED_OFF);
	io_out(io_led1, INVERTED_OFF);
	}
else
	{
	io_out(io_relay1, INVERTED_ON);
	io_out(io_led1, INVERTED_ON);
	}
//calculate setpoint2
if (nviValue < nviSetpoint2)
	{
	io_out(io_relay2, INVERTED_OFF);
	io_out(io_led2, INVERTED_OFF);
	}
else
	{
	io_out(io_relay2, INVERTED_ON);
	io_out(io_led2, INVERTED_ON);
	}
}
```