University of Southern Queensland

Faculty of Engineering and Surveying

DEVELOPING A MONITORING AND CONTROL SYSTEM FOR IRRIGATION PUMPING UNITS

A dissertation submitted by

Daniel Whittred

in fulfilment of the requirements of

Courses ENG4111 and 4112 Research Project

towards the degree of

Bachelor of Engineering (Mechatronics)

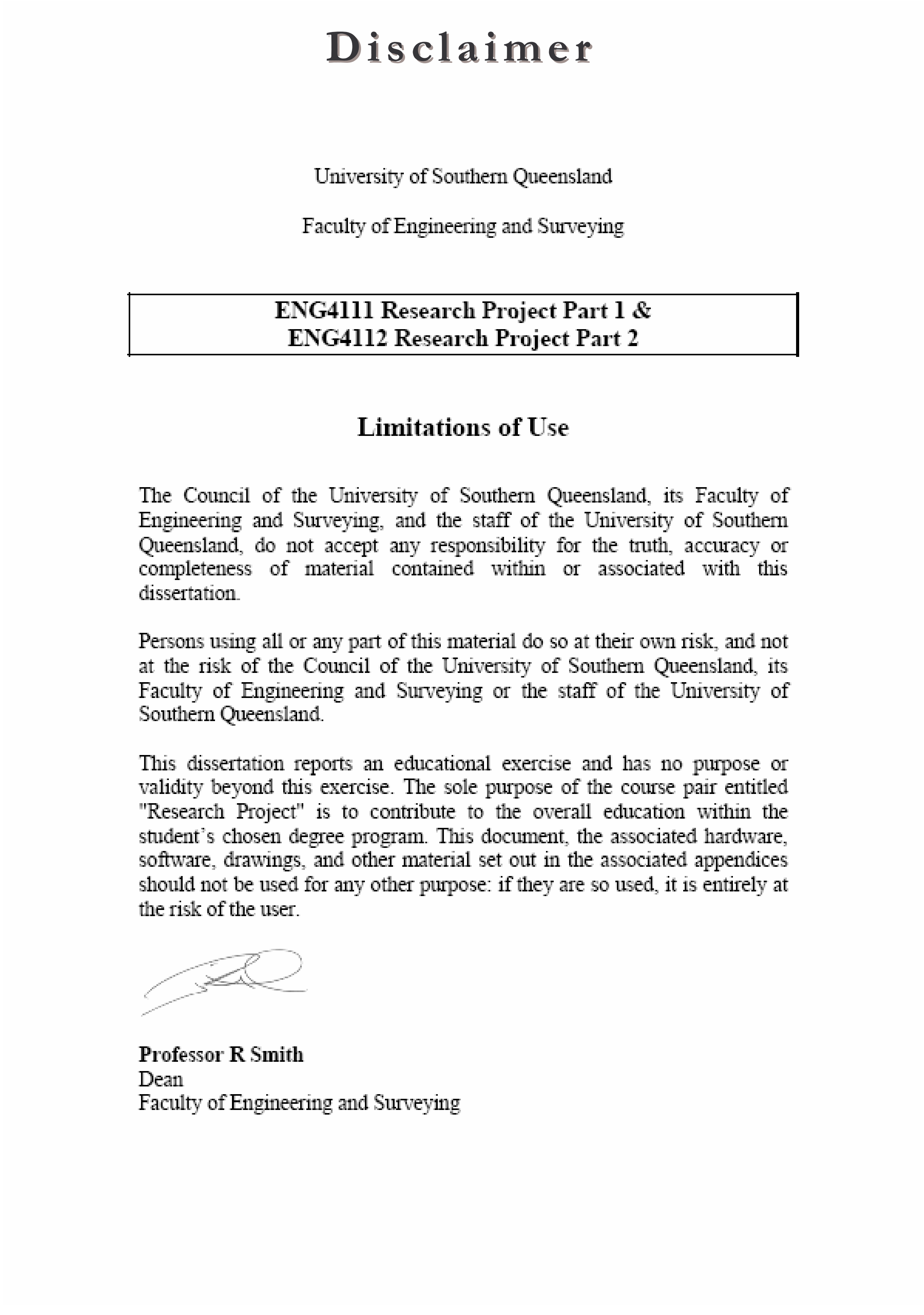
Submitted: November, 2006

Abstract

Production of a product with lower expenses is a situation that every farmer wants to be in and monitoring is an important tool used to provide understanding and a record of performance of a process. This project provides a way to monitor and log information about the performance of an irrigation pumping unit over an extended period of time, thereby allowing the farmer to use this data to make informed decisions on maintenance and overall efficiency of a pumping unit.

The system consists of two main elements: a user configurable data logging station and a second element that controls the interaction between the user and the first element. Furthermore, each of these elements has an extended set of features as follows: The data logging element is able to log data from a set of separate sensors with outputs ranging from a 0-5V analogue voltage through to a pulsed signal output. It also is able to communicate with a PC so that the information that is logged can be downloaded to a comma-separated-variable file for further analysis in a standard spread sheeting package such as Microsoft Excel. The interfacing element is made up of a numerical key pad for input of information into the system and a two by sixteen character liquid crystal display that presents the user with current readings from the sensors. There is a system of software procedures that allows these two components be independent and still communicate with each other thus allowing the input and display element to be in a separate location to that of the sensors and the data logging element.

Currently, the system is able to read inputs from four separate sensors and store data for over two months, however, these limits are both easily upgradeable to eight or more sensor inputs and enough memory to last for over twelve months of data.



Certification

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.

I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

Daniel Whittred
Student Number: 0050009703

Signature

Date

Acknowledgements

Assistance provided by the staff of the Faculty of Engineering and Surveying at the University of Southern Queensland and the staff of the National Centre for Engineering in Agriculture in providing components and facilities to work with the prototype for this project is gratefully acknowledged. Also the contribution of Steven Raine and Terry Byrne in the development of the system is greatly appreciated. Further acknowledgment must be given to my friends and family for their support over the year.

Table of Contents

List of Figures

CHAPTER 1 Introduction

Irrigation is one of the major uses of Australia's water. Over the 2003-04 period, 10,000 gigalitres of water was used for irrigation (Australian Bureau of Statistics, 2004). Irrigation is the method of providing crops with their water requirements and is used to some degree by almost all Australian farmers. The methods of irrigation are as diverse as Australia's culture, ranging from small installations of drip systems to ten or more span lateral move irrigators

In any irrigation installation there is a pump to get the water to the required pressure. This could be anything from a PTO driven pump to a variable-frequency drive electric pump. The flow of the water then enters a series of above or underground main pipelines which will take it to the area of the farm that needs to be irrigated. At the pump installation there are different points of information that will need to be measured. To operate an irrigation pump at its highest efficiency, there is need of information such as speed/flow rate, pressure as well as information about the motor itself – in the case of a diesel motor, information about temperature and fuel would be required. If you are pumping from a bore, it is useful to know things such as the standing water level (SWL) and pumping water level (PWL) of the bore.

The ability to log data from the pump, water source, and pipeline all at the same time allows for many different types of analysis. It is important to be able to analyse the performance of the pump over time for two main reasons, firstly, so that any degradation in the pumping system can be found. This may be of the form of wear in the motor, wear of the impellers in the pump, or even corrosion and clogging of the pipes. Secondly, for bores, it is important to measure the performance of the aquifer itself as extraction may lower the water-table or the underground stream may start to block with silt.

There are few options for farmers and other irrigation users when they are trying to control and log water use. There are devices that measure the volume of water that is applied to an area (directly on-site) as well as others that will measure the water levels of a bore or dam and devices that can be used to control a pump. However, these devices are separate and stand-alone. The read-outs are rarely located in the same area, making it difficult and time consuming for the farmer to check and use each of them.

The aim of this project is to design and build a system that senses different information about an irrigation pumping system. The signals from each of these sensors are to be then integrated into a unit that logs the data received and displays the relevant information to the user.

CHAPTER 2 Design Considerations

Farming in Australia is both diverse and extensive, with areas ranging from simple hobby farms with small acreages, through to large companies. To design an efficient solution for a system, such as this one, there is a need to consider all of the different options and situations that can occur. It is also important to do this systematically and thoroughly.

2.1 Irrigation Site Equipment to be Monitored

An irrigation site is made up of many different elements that need to be monitored, so that the performance of the equipment can be assessed. This involves measuring the level of the water source, the efficiency of the pump, as well as the delivery and application systems.

2.1.1 Water Source

The type of water source typically used in irrigation is an underground aquifer, or bore; however, the technology that is discussed in this dissertation also can be applied to other water sources such as dams and rivers.

Figure 2.1 shows the cross-section of a typical underground aquifer and how it responds in a pumping situation. As you can see, when the bore is not pumping, the water in the aquifer is termed the Static or Standing Water Level (SWL). When the bore is pumping, the water is drawn down around the bottom of the well and this is termed the Pumping Water Level (PWL). The difference in elevation between the SWL and PWL is termed the ‘pump drawdown’ and is a function of the recharge of the aquifer and the pumping rate.

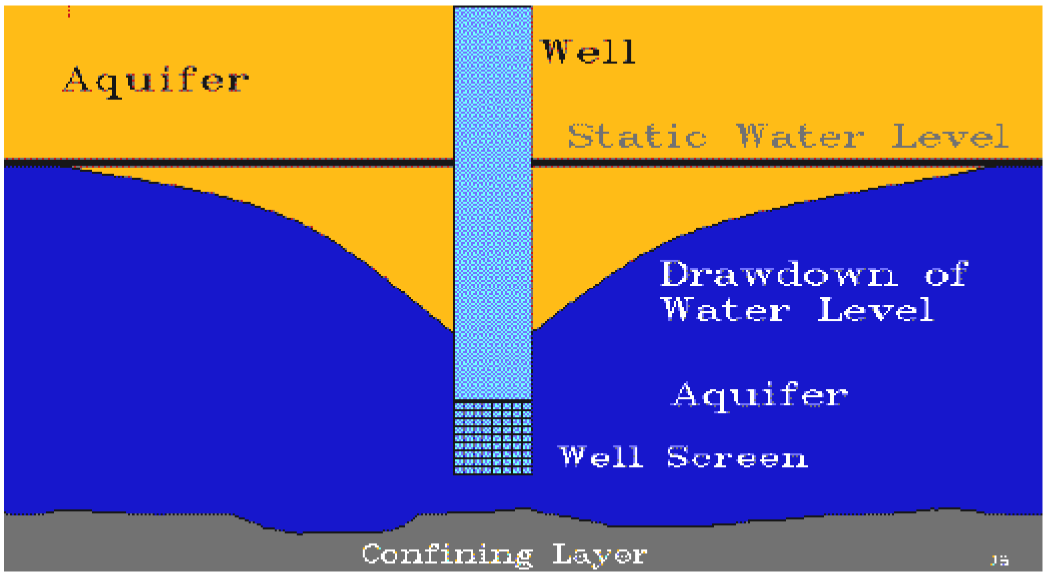


Figure 2.1: Pumping from an underground aquifer

It is important that when using a bore for irrigation consideration is given to how quickly the aquifer recharges. Often it is over used; meaning that there is not enough time given between pumping times or at times of drought. This means that the bore no longer has its PWL above the base of the bore casing, leaving space for the pump to start pumping air into the system. This leads to degradation of the whole irrigation system through loss of pressure and flow rate. Degradation of the flow can also occur when large particles clog the screen of the intake to the bore casing. Thus the overall flow of the system becomes a function of the mesh density of the well screen.

2.1.2 Pumps and drive mechanisms

Pumping systems vary from the simplest PTO (power take-off) driven pump through to a timer operated variable-frequency drive electric pump (See Figure 2.2).

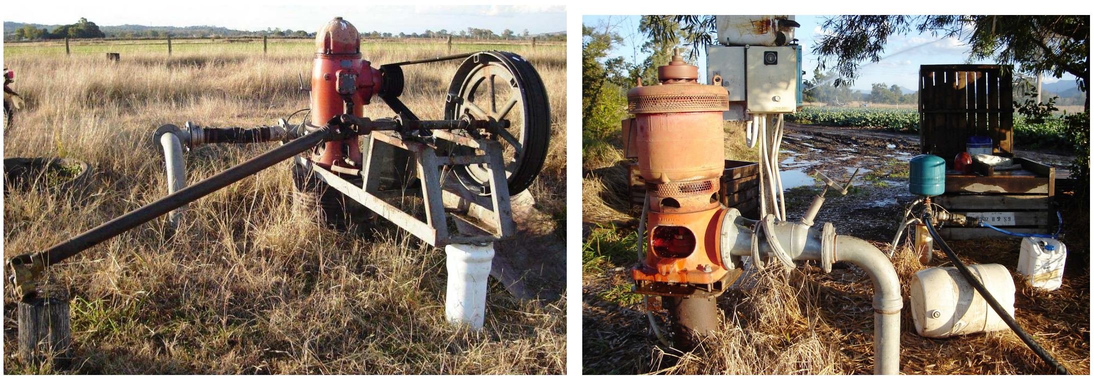


Figure 2.2: PTO driven (left) and electrically powered (right) irrigation pump

Each of these types of pump drive mechanisms (shown in Figure 2.2) offers a level of control and efficiency. The PTO driven pump is simply a shaft that connects to a tractor which then drives a pulley reduction system which is connected to the pump in the bore. The electrically driven pump differs in that it has the motor direct coupled onto the pump allowing for a more compact and simple design.

When it comes to the actual type of pump that can be used, there are many options, with the most common being centrifugal pumps, turbine pumps and electro-submersible pumps. Each of these pumps has different advantages and disadvantages depending on the need of the farmer.

Turbine pumps are commonly used in bores, have a limit to the size of the impeller and thus the pressure that can be developed at any given speed. To achieve a higher pressure from a turbine pump, it is necessary to add extra stages (more impellers) to the pump (Yiasoumi, B. 2003). Advantages of using a turbine pump for bores and wells include (New South Wales Department of Primary Industries, 2003):

- Driven by an engine.
- Less prone to damage by silt and sand in the water than an electro-submersible pump.
- It is easier to maintain than an electro-submersible unit.

It is important to measure the performance of a pump so that over time the effects of wear, maintenance and operating conditions can be observed and countered to keep the system running at its optimum levels of performance and efficiency.

2.1.3 Delivery and Application System

Water is provided to the crop through a system of above and underground piping that flows around the farmer's paddock. This line is normally tapped, with gate valves, at regular intervals around the headland of the crop so that water can be distributed to any part of the property. From the system of piping, the water is then applied to the crop by means of an irrigator or spray line system. This system can vary from anything such as a hard-hose reel or trolley irrigator through to a spray or drip line. For example, Figure 2.3 shows a simple trolley-winch irrigator used on a property at Rosevale. This type or irrigator pulls itself from one end of the paddock to the other by winding in a steel cable that is anchored at the other end of the paddock. It is important to monitor the hydraulics of the entire system because a change in what hydrant or outlet is open will result in a change in the hydraulics and thus can be used to recognise what hydrant valve is open.

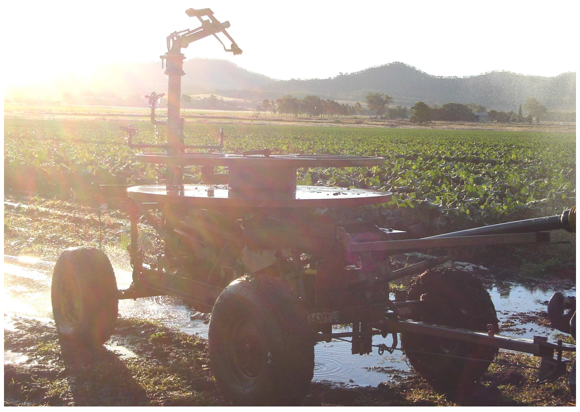


Figure 2.3: Trolley-winch irrigator after use

2.2 Sensors for evaluation of pumping unit performance

There are three main hydraulic measurements that are required to be found from a pumping system. These are the flow rate, water level (PWL and SWL), as well as the pressure. To evaluate the engine performance there is a need to consider three types of drive mechanism for the pump: tractor driven PTO; electric motor; or diesel motor. Depending on what sort of motor is driving the pump there will need to be different sensors involved to evaluate engine performance. If it is an electric motor then values for current and frequency will need to be sensed. For a diesel motor, measurements of fuel levels and temperature will need to be taken. Regardless of the motor type, there will also be a need to measure the revolutions per minute of the motor.

2.2.1 Flow Rate

The flow rate of the water can be found using many different methods. The most common flow meter is an impeller like device that is inserted into the pipe, which reads back the speed of the impeller. This type of flow meter is classed as a turbine flow meter. The rotational speed of the impeller can be sensed by gears, magnetic pick-up or even photoelectric cell.

There are two types of ultrasonic meters, being Time-of-Flight or Doppler effect meters. Below is the definition given by the Engineering Search engine GlobalSpec (2006):

> *Doppler meters measure the frequency shifts caused by liquid flow. The frequency shift is proportional to the liquid's velocity. Time of flight meters use the speed of the signal travelling between two transducers that increases or decreases with the direction of transmission and the velocity of the liquid being measured.*

Ultrasonic meters, use sound frequencies to determine flow rates and have problems when trying to measure liquids with air gaps or solids suspended within them (GlobalSpec, 2006). Magnetic flow meters measure flow by applying Faraday's Law. The sensor is comprised of two electrodes that produce a magnetic field when energised. Thus a voltage is induced when a conductive liquid passes through the electrodes. This voltage is proportional to the electric field strength, diameter of the pipe and the velocity of the flow.

Depending on the brand of flow meter there will be different outputs ranging from an analogue voltage, current, frequency or pulse; through to a digital output signal.

2.2.2 Pressure

The pressure in a pipe is usually obtained by the use of a small outlet that has a thin silicon diaphragm with a piezoresistive bridge built on to it (Shultz, W 1997). As it is stretched, the resistor bridge produces a small differential voltage, which is then scaled up to allow it to be interfaced into a microcomputer. Figure 2.4 shows the Wheatstone bridge arrangement that is the essence of the piezoresistive sensor.

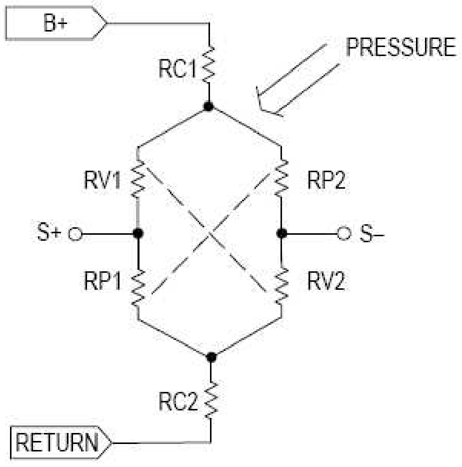


Figure 2.4: Pressure Sensor Circuit
(*Warren Shultz, Motorola Inc., 1997*)

2.2.3 Water Depth

Depths such as the standing water level (SWL) and pumping water level (PWL) can be calculated using a "bubbler". A bubbler is comprised of a tube (usually less than five millimetres in diameter) that is run down the side of the bore casing to the depth of the pump turbines. This line is then pressurised so that the air 'bubbles' out of the lower end of the pipe, thus the name. The pressure that causes this displacement of water in the tube can be measured. The distance of the line down the bore and the pressure needed to displace the volume are used to calculate the aquifer water level.

The bubbler is pressurised by either a micro-compressor or from gas stored in a cylinder. The latter option is more favourable in areas where there is no electricity as the gas could be replaced when the pressure became low via an external pump. In areas where electricity is readily available, the micro-compressor is activated only when the tank pressure is low or at a regulated interval to conserve electricity.

Water level can be measured at any time, though the best and most relevant cases will be when the pump is not running, allowing us to find the SWL of our underground aquifer, and during normal pump operation, so as to find the PWL. These depths are an important monitoring tool for the farmer as they can analyse the performance of their bore throughout the year and know when they are using it beyond its recharge rate.

2.2.4 Electric Motor Sensors

The main data that should be recorded from an electric motor are current and frequency. Due to the electric nature of this motor, the values can be scaled and then read directly by the microprocessor. These sensor values are an important tool in measuring the performance and efficiency of the electric motor.

2.2.5 Diesel Motor Sensors

Important data to be collected from the diesel motor include the temperature and fuel level. Temperature can be measured with small integrated circuits (IC's) that communicate directly with the microcontroller. Often there are specific temperature sensors such as the LM35 (Nation Semiconductor, 2005), which are calibrated to read the temperature in degrees centigrade with an accuracy of 0.25 °C. A simple electronic fuel gauge can be used to measure what fuel has been used over time. However for engine performance evaluation the fuel level can be measured using specialised flow meters that measure both the outflow and the return in the diesel motor.

2.2.6 Rotational Speed

One of the easiest methods for a user to measure how their system is performing is via the rotational speed of the motor. This value of rpm can be found via a simple digital tachometer. The output of this is usually a 0-5Vdc change that is relative to the speed of the motor.

2.3 Data Capture and Storage

To be able to achieve an accurate analysis of the performance of the bore, it is necessary to provide data that is fine enough to produce accurate results and yet consume as little storage space as possible. For example, when a bore pump is first started there will be a drop in the SWL over the first few minutes until it settles and becomes the PWL. Also during this time the pressure and flow will steadily increase to a certain level. Similarly, this pattern will occur again when the pump is stopped. This all may take three minutes and if the system was only sampling at a rate of once every five minutes or even two minutes, then the captured data will be very ragged and 'chunky'. However, if the sampling rate was taken to once every ten seconds, then over a 10-14 hour day of pumping, then the data would have stored over 5000 times in a day. Most of this time the values stored will be the same. As you can imagine, storing the data in this fashion would cause a 64k byte EEPROM to run out of memory after one day (when storing ten bytes of information per save).

There are two methods that could be used to save captured data more economically; the simplest is having two different sampling modes, one for the first three minutes where the data is sampled every 7-10 seconds, then after that it is sampled at 8-10 minutes intervals. Secondly, the data is checked for deviation about the mean, before it is saved. This is an adaptive method where the data is only stored if it has varied significantly compared to the last stored point. A more detailed description of these methods is discussed in Section 4.1.3.

2.3.1 Data loggers – Existing Vs Custom Made

Data loggers are specific microcontrollers that take information from sensors at a regular sampling interval and record the information into memory. When it comes to a choice of the data logger, there is a choice between using off the shelf parts, in this case a data logger, or designing and developing a custom system. From the range of existing data loggers on the market, it was found that to incorporate an off-the-shelf data logger would require excessive modifications and customisations.

The cost of purchasing a consumer data logger may be high and thus prohibitive to this project. A quote was obtained from Pacific Data Systems Pty. Ltd. in Australia and it was found that for a data logger that would perform the required tasks similar to the prototype system, it would cost between $2,000 and $3,000 (Please see Appendix B Data Logger Quote for further information).

To design a custom data logger, it is important to consider all aspects of how it will be controlled; ranging from the use of discrete IC components through to using a dedicated microcontroller to systematically perform tasks. With the relative low cost of small, low power PIC microcontrollers, it is often more viable to use a microcontroller, rather that using a complex design of discrete logic circuitry.

2.3.2 Use of Microcontrollers

Microcontrollers vary is size, shape, cost and functionality so that there is virtually one to suit every need. As described by Phythian (2006), a microcontroller is "a single chip computer which is tailored for a specific application". These chips can have anywhere from 8 to 32 bit word sizes and usually do not contain an operating system.

2.3.3 Other Integrated Circuits

Due to the digital nature of a microcontroller, that is it follows the pure logic of a bit equalling 1 or 0, and that many older sensors will have outputs that are analogue in nature (meaning they are a scaled value such as a voltage from 0-5), the microprocessor will need to be able to convert signals that are analogue into digital values so that it can process them. Such a device exists as an Analogue-to-Digital converter (referred to as ATD or ADC) and is often integrated into the main microcontroller, rather that being an add-on.

Another important feature of the microcontroller is that it should have a timer module that will allow capture from these external ports. Without this function it would be hard to facilitate the operation of a data logger. The microcontroller must have enough onboard storage memory to hold the necessary number of data samples or alternatively, it needs to be able to communicate with another class of external memory such as EEPROM or RAM. Finally, it must also be able to communicate with another storage medium so that the information from this can be downloaded to another computer for analysis. These requirements are discussed further in Section 3.1: Microcontroller.

2.4 Interfacing the system

When bringing all of the systems components together into a working device, there is need to make sure that the relative standards are met so that the device can be easily interfaced into the real world. An example of this is that the communication protocol that is used to transfer the information to the farmers PC must be of such a standard that the PC can understand this information that is sent to it. Similarly, if an RS232 connecting cord was used between the microcontroller and the PC, then certain wiring standards must be considered and designed appropriately.

2.4.1 Real-Time Display (LCD)

As stated in the aims of this project, the device will need to give a graphical output to the user while the pump is in operation. LCD displays are one of the best mediums on which to communicate with the user. While the unit is operating, it needs to display different readings, mainly items that are of immediate importance to the user. This would include the real-time speed of the motor, pressure and flow-rate. Ideally, information from all of the sensors should be able to be seen, either directly on the screen or through a menu system.

2.4.2 Downloading the Data to a PC

A necessary part of any data logger is the ability to download information back to the user so that they can analyse the information. This needs to be done simply and easily so that even people with limited computer skills are still able to access their logged data. There are many methods that can be used to transfer data to a users system. Firstly this can be through a connection between the data logger and the users PC; with this being a direct cable link or wirelessly. A common direct link is to send the data serially through a RS232 port onto the PC. Another is a matter of sending the data through a modem link. There are many different wireless standards ranging from

Radio Frequency (RF) signals, Bluetooth and GSM. Each of these standards have a different throughput and range.

2.4.3 Optional Control through the Microcontroller

The microcontroller also needs to be able to send output signals to control different aspects of the motor. For example the electric motor will be able to have its speed varied directly from this unit rather than separately from the motor. This can be easily facilitated by varying the frequency of the electrical signal that powers the motor. For a diesel motor, an electrically operated ram can be used to change the position of the throttle control.

2.4.4 Power Supply

In the situation that this device will be in there may be periods that the machine needs to go without power. If this device was used on an installation that was run by an electric motor then the device could include a simple transformer to provide the system with a regulated 12Vdc or 5Vdc. However, if the device was on an installation that was powered by a diesel motor or a tractor PTO then there would be no power to the unit. This means that it needs to have an internal battery that has a long life span and also an internal backup battery for when the main power is not available.

2.4.5 Housing and Connections

It will also be important to consider how the system interfaces with the real world. Simple things such as the housing and its aesthetics appeal may mean a lot to some customers. Likewise, it is important to ensure that the housing is strong enough to withstand the elements. Seemingly insignificant items, such as the way that the device’s cables connect and the durability of these cables can influence the decision for a person to buy a system.

2.5 Design Specifications

From the above design consideration we are able to provide the following design specifications:

Inputs:

- Able to handle up to six analogue inputs with an 8bit ATD
- Minimum of four digital inputs
- Able to count input pulses

Outputs:

- Minimum of eight digital output lines
- Serial output capabilities

Data Storage:

- Retention of information after power is removed
- Access to real time clock
- Sufficient memory for three months of readings and/or able to communicate with external memory

Human Interface:

- Two-line LCD
- Input number pad or keyboard

General:

- Robust (i.e. inbuilt static protection, shock resistance and capable of withstanding large temperature variances)
- Low cost

CHAPTER 3 Hardware Design

The main components of the hardware design are the choice of microcontroller, the board configuration, the human interface and how the system is powered.

3.1 Microcontroller

It is necessary to select the correct set of hardware that will be able to communicate with a range of sensors previously described (Chapter 2.2).

3.1.1 Choice of Microcontroller

There are many different microcontrollers on the market that would be suitable for this project. Ranging from the relatively high powered Freescale/Motorola HC12 through to the relatively small and simple PICmicro microcontrollers. This variance in power also has a corresponding variance in price, ranging from $300-$500 for the higher powered HC12, down to under $10 for a PIC based microcontroller.

Due to cost constraints, this product was designed using a PIC based microcontroller. PICs are generally the cheaper, with a 28 pin model being $12 (PIC16F876A, MicroZed Computers, Oct 2006). However, the disadvantage of using a PIC is that it requires special programming hardware. A PIC based microcontroller is the PICAXE (Revolution Education, United Kingdom). This processor only requires a low-powered PC to program it and is also reasonably cheap with a 28 pin package costing $21 (AXE010X from MicroZed Computers, viewed Oct 2006).

3.1.2 What is a PICAXE?

> *"A PICAXE is a standard PICmicro microcontroller that has been pre-programmed with the PICAXE bootstrap code. The use of this bootstrap code allows the user to simply reprogram the micro directly with a serial connection"* (Revolution Education Ltd, 2004).

The PICAXE system was originally designed by Revolution Education to be used in schools around the United Kingdom. Initially the PICAXE seems slightly limited in its capabilities. However, this is only due to the market strategy that Revolution Education is using. After further investigation, it was found that the PICAXE is very powerful and robust and thus makes it near perfect for use in this situation.

The original design was based around a 40 pin PICAXE chip called the 40X (A full pinout of this chip is supplied in Appendix C). From the pinout, it can be seen that the 40X has eight ATD channels and has another dedicated eight input and eight output channels as well as another eight selectable/interchangeable input/output pins. The PICAXE chips themselves also are able to perform to the specifications (Chapter 2.5). The advantage with the PICAXE chips is that they are specifically designed to be handled out of laboratory conditions, meaning that they have extremely high static protection built into them and they are rugged. Also, this type of design means that they are sold relatively cheaply, with a 40X costing $26.25 (AXE014X, MicroZed Computers, viewed Oct 2006).

Unfortunately, the 40X was unable to be used as the one loaned did not have the latest firmware making it not compatible with the latest programming software. However, the 28 pin model (also shown in Appendix C) operates in the exact same way as a 40X (same coding and hardware requirements), just with a few less I/O pins. Hence, all my future PICAXE references will discuss the 28X in replacement of a 40X chip.

3.1.3 Inter-Integrated-Circuit (I^2C) Bus

Other features of the model 'X' PICAXE chips are that they are able to communicate on the I^2C bus.

> *"The I^2C (Inter-Integrated-Circuit) bus was originally developed by Phillips Inc. for the transfer of data between ICs at the PCB level"* (Revolution Education Ltd, 2003).

The I^2C bus is made up of two wires the clock (SCL) and the data line (SDA) that are pulled high by normally a 4.7 kΩ resistor. The IC that controls the data on the bus is called the 'master' and is often a microcontroller, other ICs that are connected to the bus are referred to as 'slaves'. There is a theoretical limit of 112 'slave addresses' available, meaning that there can be a maximum of 112 different slave devices connected to the bus at the same time (provided that they all have unique slave addresses).

The main advantages of using a data bus such as this on a PICAXE chip is that it is extremely easy to use in the program coding. It also only uses two pins on the microcontroller and there are many different ICs that can be used on the I^2C bus (EEPROM memory, real-time-clocks, analogue-to-digital and digital-to-analogue converters, motor drivers).

3.2 Board Configuration

A final circuit design for the prototype is shown in Appendix A. This section describes each of the components and how they interface with each other.

3.2.1 28X Microcontroller Configuration

The initial prototype consisted of a 40X chip controlling all of the board components; LCD, keyboard, data capture. However, it was not as efficient to scan for a key press, and then perform the data capture (discussed further in Chapter 4). To facilitate this software issue, it was decided to use two separate PICAXE chips, the 28X to perform the data capture and storage routines and an 18 pin PICAXE chip, called the 18X, to perform the display and key press routines. To communicate between the two chips, a simple two-wire serial communication method which uses a minimum number of pins on the chip.

With this new board configuration, the 28X chip basically controls the data capture and storage from four analogue pins, reads the current time from the RTC and then stores all this information to an external EEPROM chip via the I^2C bus. The 28X is configured using the standard 5V power supply rails and uses the standard download circuit that is described on page 21 of the Picaxe Manual 1, (Revolution Education, 2004). This download circuit simply allows the programming of the 28X via a serial cable from a PC. Other standard configurations are the tying high of the 'reset' pin with a 4.7 kΩ resistor. This also allows a switch that pulls the 'reset' pin low to be used to either reset the PICAXE or turn it off. The final standard piece of hardware that needs to be connected to the 28X is a 4 MHz clock crystal. This external crystal allows for more precise timing than if it were an internal crystal.

Connected to the 28X are other ICs such as an EEPROM chip and RTC. These are both connected to the PICAXE via the I^2C bus. Pin 26 (Output5) is used as the Write Protect (WP) line for the EEPROM. This WP line is set high when the PICAXE wants to read from the EEPROM and is set low when it wants to write to the EEPROM. Also connected to pins 23 and 24 (Output2 and Output3) on the 28X are two LED's (red and green) that flash to signify when certain events are happening in the code such as green when a reading is being taken and red/green on an error. Pins 11 and 21 (In0 and Output0 respectively) are used to communicated with the 18X, with In0 used as an Interrupt-Request pin; meaning that when it is pulled high by the 18X, it runs a special sequence of code. Finally, pins 12 and 28 (In1 and Output7) are used to communicate with the users PC when an EEPROM memory download is initiated.

3.2.2 18X Microcontroller Configuration

The PICAXE 18X is configured in much the same way as the 28X (see above) in terms of power supply, reset circuit and serial download circuit. However, with the 18X there is no need for an external timing crystal as this is provided internally. Appendix C shows the standard connection and pinout for the PICAXE 18X chip. The 18X controls two main pieces of hardware – the keyboard and the LCD.

The LCD has its data lines, DB7|DB6|DB5|DB4, connected to pins 13-10 (Output7 to Output4), as well as two control lines, RS and SE. Also connected to the 18X is the keypad. An advantage of using the PICAXE system is that it has special instructions that are able to be used to communicate with a standard keyboard encoder, such as the one that is in any standard QWERTY keyboard. Thus the prototype uses a modified PC keyboard with only the number pad remaining. This meant that there were only two wires connecting the keyboard to the PICAXE, the clock line and the data line, that were each pulled high by a 4.7 kΩ resistor.

3.2.3 Memory EEPROM

To choose an EEPROM chip first the size need must be decided upon. At this point in time the calculations are based on average of one store per 10 minutes and on an average of a 10 hour pumping period. In each write, there will be eight bytes of data stored (Month, Date, Hour, Min, Sensor0, Sensor1, Sensor2, Sensor3). This gives us an expected number of bytes of memory used per day of:

$$\begin{aligned}\text{bytes per day} &= 8\times 6\times 10\\ &= 480\end{aligned}$$

Equ 3.1: Number of bytes stored per day

To get a minimum of three months of readings without having to download information, we will need to store at least $480\times 3\times 31\text{ days}\approx 45000\text{ bytes}$ of data. With the use of the PICAXE and the I^2C bus we can use an array of 8 separate 64k byte EEPROMs giving us 512 kB of memory to use. This will allow for almost three years of data to be stored (see Equ 3.2).

$$\begin{aligned}\frac{65536 \text{ bytes} \times 8}{480 \text{ bytes/day}} &= 1092 \text{ days} \\ &= 35 \text{ months}\end{aligned}$$

Equ 3.2: Days available using an 8 x 64k array of memory

The memory EEPROM that was used for my prototype was an I^2C CMOS Serial EEPROM (part 24LC256). It is only a 32k bytes memory, but was used as a ‘proof-of-concept’ only chip and the code would stay the same if it was replaced by a 64k byte chip. This particular memory was chosen as is cheap and reliable. Currently, this chip costs $3 (24LC256 from MicroZed.com.au, Oct 2006). From the manufactures datasheet (Microchip Technology Inc., 1998), the 24LC256 has the following features:

- Low power CMOS – maximum write current of 3 mA
- Temperature range of -40°C to +85°C
- 5 ms max write-cycle time
- 100,000 guaranteed erase/write cycles
- Data retention of more than 200 years
- Page write capability of 64-bytes

For use on the I^2C bus, these memory chips have three address pins that can be hardwired so that eight of the same chips can be used and yet each will have its own individual address. For my prototype, the address was wired to be at ‘1010[A2][A1][A0]X’; where A2, A1 and A0 where equal to a logic 0 and ‘X’ meaning “don’t care”. Also, these EEPROM chips have a write protect (WP) pin. This pin allows the read/write mode to be set before the chip is accessed to prevent data corruption.

3.2.4 Real-Time-Clock (RTC)

A RTC is needed for accurate timing as well as accurate time stamping of the captured data. The RTC that I chose to use for my prototype was the DS1307 made by Maxim/Dallas Semiconductor. It is an eight pin package that communicates on the I^2C bus and incorporates a backup lithium cell battery so that the time is kept even after the loss of power. It also has the ability to have an output that pulses at 1Hz, which can be used with a LED so that the LED will flash every second. A DS1307 costs $5 (DS1307 from MicroZed Computing, Oct 2006).

In my circuit the RTC was connected on the I^2C bus that of the 28X. It also had a red LED, in series with a 470 Ω resistor, connected to pin 7 (SQW/OUT) of the RTC allowing for a visual indication of each second. The setup of this IC was simple as it only had to be programmed once with the correct time and then while it was connected to its backup battery it was able to keep the time continually.

3.3 Human Interface

To allow this system to be used on-site, it is important to have an easy and simple interface that people with limited experience are able to use immediately. The ability to also provide direct real-time values to the farmer is an objective of this project.

3.3.1 Liquid Crystal Display

A simple 2-line x 16 character LCD module was used as the display medium for this system. The module that was used was a DSE Z 4170 (Dick Smiths Electronics, Australia) and costs $20 (DSE Z 4170, Oct 2006). MicroZed.com.au provides a PICAXE LCD module that requires only one wire to connect to the PICAXE system. This type of module allow the PICAXE microcontroller to send information as ASCII characters serially to the LCD module and the LCD module has its own microcontroller that formats the information and displays it. This type of module costs

$52.50 (AXE033 from MicroZed.com.au, Oct 2006). Using the AXE033 is a much more efficient use of microcontroller pins; however, using this system is not as powerful as directly programming the LCD module.

Connecting the LCD module to the 18X is via six lines, four being a data bus and 2 control lines. Figure 3.1 shows the output connections from the LCD module to the 18X. Lines D7 thru D4 are connected to the output pins Output7 thru Output4 of the 18X and the two control lines E and RS are connected to Output3 and Output2. The LCD works by the PICAXE putting information on the data bus (D7-D4) and when the enable (E) pin is pulsed, the LCD module reads what ever is on the data bus. Depending on whether RS is a high (character mode) or a low (instruction mode) will control whether the LCD module takes the information from the data bus as characters to be displayed or as an instruction.

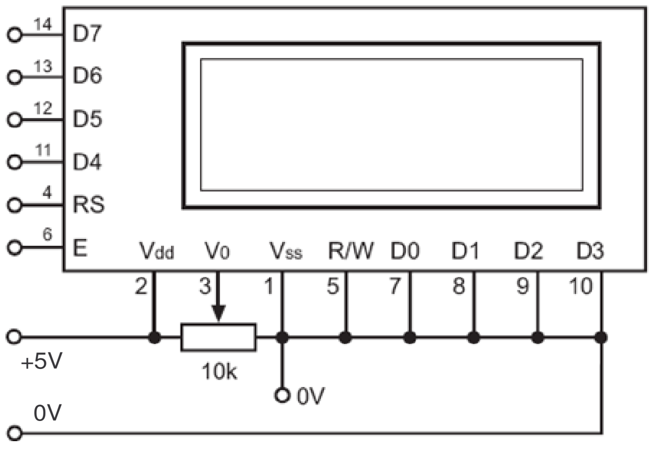


Figure 3.1: Connections for LCD module to PICAXE 18X

(*Image from PICAXE Manual 3, Revolution Education, p34, 2004*)

Often additional components are added to the circuit above, such as 330Ω resistors on the data and control lines so as to protect the LCD from static discharges. Also, the 10 kΩ potentiometer can be replaced by a connection to a digital-to-analogue converter that is connected to the microcontroller, allowing the contrast of the LCD module to be controlled by the microcontroller.

3.3.2 Keyboard

As previously mentioned, the keyboard used in the prototype was a modified QWERTY computer keyboard. This approach allowed me to take advantage of the built in PICAXE commands that will read data directly from a PC's keyboard encoder. Figure 3.2 shows the wiring connections used to connect the keyboard to the 18X. Due to the nature of this connection keys such as PAUSE and PRNT SCRN are not able to be used reliably as they have special multi-digit codes. The reason for using the 18X to read in the keypress' is because the '`keyin`' command that PICAXE uses to read the keyboard encoder value pauses all processing on the PICAXE, meaning that if this was used on the 28X (data logger), it would stall at the '`keyin`' instruction and not actually read any sensors until a key was pressed.

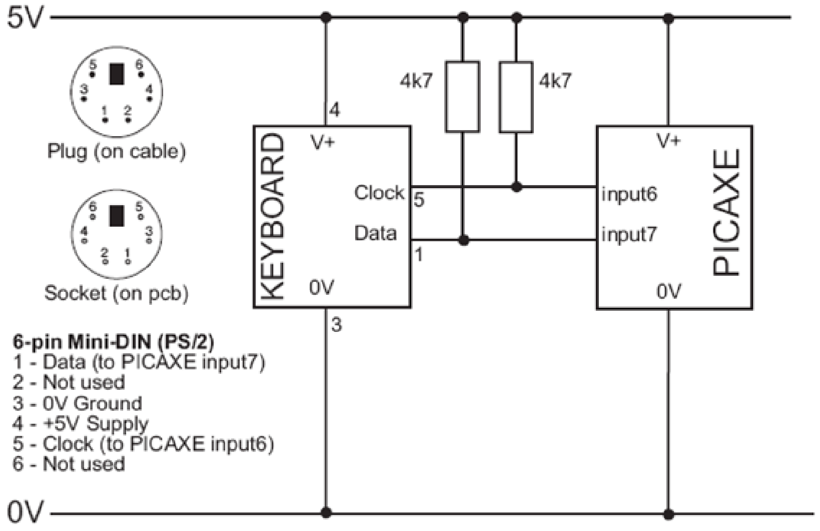


Figure 3.2: Wiring used to connect keyboard to PICAXE 18X

(*Image from PICAXE Manual 2, Revolution Education, p40, 2004*)

3.4 Powering the System

It was important to be able to provide a system such as this with an accurate and reliable power supply. Depending on the situation that the data logger is in, that is, whether there is access to AC power or whether it is in a remote location and has to rely on battery power, will determine how the data logger system performs.

3.4.1 AC Powered Supply

For the prototype, a simple wall plug pack was used to give an output of 9V DC. This was put through a 5V regulator (KA7805, Fairchild Semiconductor, South Portland, Maine U.S.A..) that gave the required voltage to the microcontrollers and other ICs. However, this initial setup did not provide a stable enough supply for the microcontrollers. A 1000 µF electrolytic capacitor was put across the 9V main supply to filter the supply before it went through the regulator and then also 10 nF capacitors across the supply just before the microcontrollers and the other ICs so as to filter out any high frequency noise.

Sensors are more of a challenge to power as they normally need more that the 5V that is supplied for the microcontrollers so that the analogue voltage of their output is able to be read by the microcontroller. However, this can be fixed simply by supplying the sensors directly from the 9V DC wall supply. This will be slightly filtered and regulated by the 1000 µF electrolytic capacitor, thus providing a reasonable supply for the sensors.

3.4.2 DC Powered Supply

In situations where there is no AC connection available for supplying the system, there is a need to find an alternative power source. This can range from a simple generator that can be connected into the same driveshaft that turns the pump. This could be driven by a tractor PTO or a diesel motor, regardless the source, it can used to drive a small generator that will not affect on the performance of the pump. Another option is the use of a large battery (such as an automotive truck battery) that is then either charged by a generator setup on the pump or a solar panel. Automotive batteries are usually 12 volts and this means that they can be used with the same regulator-capacitor setup as mentioned in Section 3.4.1. Additional code modifications can be made so as to use as little power as possible when the system is idle (this is discussed further in Section 4.1.3).

3.5 Housing

Due to the outdoor nature of this system, the housing for it needs to be able to withstand the elements of nature so as to protect the system inside. This means that care will have to be taken to make the housing waterproof and able to withstand vibration. Security can be a concern for farmers, thus it is important to ensure that the housing can be mounted securely.

CHAPTER 4 Software Design

An important part of any system that is computerised is the software. The software needs to be self contained and free of errors such that it will be able to accept any user input and perform the correct response. As previously stated (see Chapter 3.1), there is a need to separate the function of the prototype between two individual microcontrollers. This is to mainly facilitate the use of the `keyin` function which allows the PICAXE to read directly from a keyboard encoder. The `keyin` function works by pausing until it detects an input on the 'keyboard data' pin. While the microcontroller is waiting for an input, it is not able to perform any other functions, thus it would not be able to perform the necessary data logging functions. Thus, by letting a second microcontroller perform the functions of waiting for the users input and displaying information, the second microcontroller is free to log the data.

The prototype works with a PICAXE 28X performing the data logging functions and another PICAXE 18X controlling the user input and display. Figure 4.1 gives a breakdown of what software functions are used and how they are interrelated. Please note that all code associated with the prototype is contained in 0 for the data logger (28X) and in Appendix F for the control (18X) microcontroller.

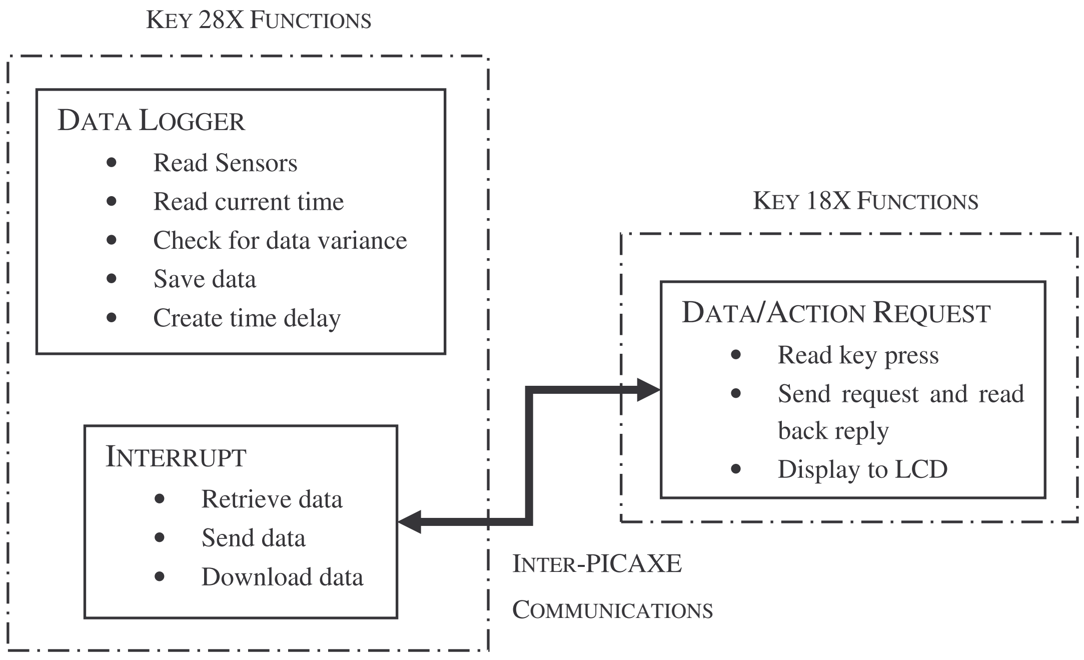


Figure 4.1: Key functions of microcontrollers

4.1 Data Logger

Using the 28X as the microcontroller that performs the data logging functions, means that it is able to continue other tasks while the 18X is working with the user. An outline of how the data logging system operates is shown below:

1. Turns on the green LED to signify the start of a sample
2. Reads the four sensor values into variable memory (Section 4.1.1)
3. Reads current time value (Section 4.1.2)
4. Performs the data variance routine (Section 4.1.3)
5. Saves the data to external EEPROM (Section 4.1.4)
6. Ends the flash of the green LED
7. Checks if memory address is at the end of available values and then loops back if it is the case
8. Stores the latest address used to onboard memory
9. Implements time delays (Section 4.1.5)
10. Loops back to number 1 to start another sample

4.1.1 Read Sensors

The data capture routine is named '`read_data`' and does nothing more than read in the ADC values of sensors 0-3 and saves these values to the variables named `data0`-`data3`. This function is called at the start of each loop so as to sample the sensor values. With the PICAXE 'X' chips, there is an option to read the data through the ADC with 10-bit resolution (`readadc10`) as opposed to the standard 8-bit resolution (`readadc`). The reason for using only the standard 8-bit resolution is that in this application there is little need for a higher resolution as the sensors themselves are usually not accurate enough to make use of the higher accuracy.

4.1.2 Read Current Time from RTC

Reading the time from the RTC is done via the I^2C bus. This means that there has to be initialisation routines used before the RTC can be accessed. By using the `i2cslave` command we are able to set bus parameters such as slave address (`%11010000`), bus speed (`i2cslow`, 100kHz) and the register address size (`i2cword`). With the bus parameters set we are then able to start reading information from the RTC, from register 0, by using the `readi2c` command. The values read will be seconds, minutes, hour, date and month and they will be stored in the variables `sec`, `mins`, `hour`, `date` and `month` respectively.

Once this information has been read from the RTC, the sub-routine `bcd_decimal` is called to convert the time variables, which are in binary coded decimal (BCD) into decimal values so that they can be used in mathematical operations. `bcd_decimal` takes each variable value and converts the low-nibble of the byte into hexadecimal by multiplying by ${}^{10}\!/_{16}$. The same is done with the high-nibble. These are both added together and stored back into the variable, thus converting BCD to hexadecimal. After the `bcd_decimal` routine has completed it returns to `read_time_now` which then returns to the main program.

4.1.3 Check for Data Variance

As previously mentioned, it would be wise to add a strategy for reducing the amount of data points stored based on whether the latest point has varied significantly from the previous stored point. This type of data storage will allows for less waste from repeatedly storing the same data point.

4.1.4 Save Data to External EEPROM

Data for the prototype is stored by saving each different byte (month, date, hour, etc.) in a different section of memory. Graphically, we can imagine that the 32k (32,768 bytes) of memory is used is split up into eight separate sections, each dedicated to its own type of information (see Figure 4.2). As you can see, the 32k is broken into eight lots of 4096 bytes, which each contain their own pieces of logged data.

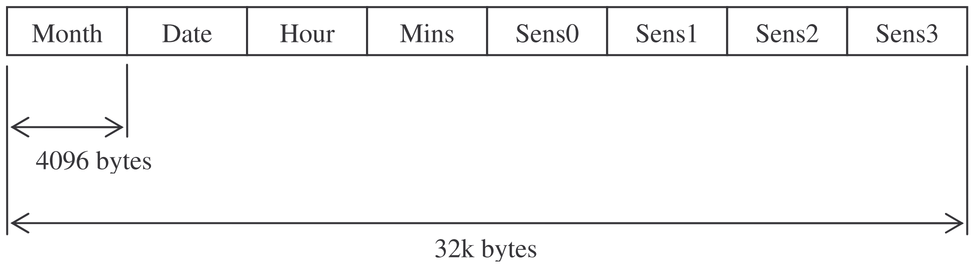


Figure 4.2: Breakup of a 32k External EEPROM

This type of storage management allows for a simple offset addressing method. It works by simply using a variable (`address`) that contains the memory address that we want to store information into. As we store the first piece of data (the value for month) we use the memory address contained in `address`, then for the second piece of data (date) we use the memory address of `address` + 4096 (the value 4096 is stored in `memoffset`). This is continued with each subsequent piece of data having an offset from the base address value that is a multiple of 4096. At the end of the loop the `address` value is incremented and the next piece of data is able to be stored sequentially.

As the memory is steadily filled, it is important that when the address reaches its upper limit that the data logger is still able to store data. The best way to do this is by running a simple check in every loop that tests to see if the value of `address` has reached a predetermined limit (4096 in this prototype), when this occurs the value for `address` is set back to zero, allowing the memory to effectively loop and overwrite on itself. This means that there will always be 4096 readings stored in the memory.

The actual software sequence of storing the data is as follows: first, because the external EEPROM is connected to the microcontroller via the I^{2}C bus, we must first set the bus parameters to ensure that the communications are sent to the correct device. This is achieved by setting the `i2cslave` parameters as `%10100000`, `i2cfast`, `i2cword`. We then set a temporary variable to equal the address that we want to store to (including the offset) and store our byte of data to this address with the `writei2c` command. A pause of 10-20 milliseconds is required to allow for transfer of the data on the I^{2}C bus. At this point a simple check is made to ensure that the data has been stored, without error, to the external EEPROM. This is achieved by reading the data back from the address is was stored to and then comparing it to the data that was to be originally stored. If an error is found, the microcontroller jumps to an error routine that just has the LED's flashing to warn the user that the error has occurred. If the data has read back correctly, then it moves on and stores the next piece of data.

4.1.5 Time Delays

At the end of each loop the real-time-clock is used again to create an accurate time delay. It begins by using the function described in Section 4.1.2: Read Current Time from RTC to accurately acquire the current time. An offset of the desired delay is then added to this by adding each component (hour, minutes and seconds) of the offset to the current values for the time. The current time is read again and is compared to the offset values to see whether the current time is equal to the desired time. If it is not, then this process is repeated until it is. When this happens, the program returns to the main loop and starts the next sample.

4.2 Data/Action Request from Control PICAXE

As has been seen previously in Figure 4.1, the PICAXE 18X controls how the user interfaces with the data logger, with this being done through the control of the keyboard and the LCD. A basic outline of the 18X control PICAXE's main functions is as follows:

1. On power up, initialises LCD and pauses for half a second to avoid power up error from keyboard
2. Waits for a key press; uses this value to determine the what request is to be sent to the 28X (See Section 4.2.1)
3. Sends data request to 28X and receives information back from 28X
4. Displays this information on LCD
5. Loops to back number 2 to wait for the next key press

It was found that during the testing that an error often occurred at startup, with the 18X picking up noise on the line as a key press, thus there was a need for a small delay to allow for both the keyboard and the microcontroller to be fully started before it started reading the keyboard's data line.

4.2.1 Keyboard Routine

Next in the sequence of functions for the 18X is that routine that is used to capture the key that is pressed on the keyboard. This is facilitated by the use of the `keyin` function, which works by pausing the microcontroller until it registers a key press. As mentioned at the start of Chapter 4, this function is the main reason why there are two separate microcontrollers. The `keyin` function, as described in the PICAXE manual 3 (BASIC Commands), is used:

> *"…to wait for a new key press from a computer keyboard (connected directly to the PICAXE - not the keyboard used whilst programming). All processing stops until the new key press is received. The value of the key press received is placed in the predefined variable '`keyvalue`'."*

The value that is left in `keyvalue` corresponds to an array of numbers that will give the ASCII equivalent of the key that was pressed.

4.2.2 Send Data Request

The data request is a predefined set of codes that are used in communicating between the PICAXE's. A set of five codes were used in the prototype and are show in Table 4.1; however, these codes could easily be expanded to include further sensors or add extra control on the 28X.

Table 4.1: Current Request Codes

Code	Data – Description
0	Data request for latest reading from sensor 0
1	Data request for latest reading from sensor 1
2	Data request for latest reading from sensor 2
3	Data request for latest reading from sensor 3
	Action – Description
4	Control request to initiate download of readings from the 28X

With the request number (found from the key press) now stored in `drequest`, we are able to send this value to the 28X via the serial inter-PICAXE communications (See Chapter 4.4 for specific details). The 28X responds with either a value that will need to be displayed or if the request code was "4" then the 18X displays a message.

4.2.3 LCD Routine

After the data is received back (stored in `din`) from the 28X, the 18X performs as series of steps to format the data and display it on the LCD. For the example of a sensor reading, the 18X will take `din` (which is a hexadecimal value received from the 28X) and run it through a routine called `dispnum`, which formats the value. This value from the 28X will be in the range of 0 to 255 and when it is passed through `dispnum` it is converted into ASCII values that are able to be directly transferred to the LCD. The `dispnum` routine works by splitting the hexadecimal value into it hundreds, tens and units and then adding 48 to each of these values to get the ASCII equivalent value. Each of these individual digits is then sent to the LCD module to be displayed.

For example, if the decimal value of 128 was read back from the 28X it would be split into the digits 1, 2 and 8 and then have 48 added to them to turn the digits into ASCII values, which could then be displayed on the LCD. In an example where a control code is sent to the 28X, the 18X will not wait to receive data from the 28X, however it will display a predefined message from its on-board memory. For example, when a control code "4" is sent to the 28X, the 18X knows that it has initiated a download to a PC, thus the 18X begins displaying the message, "Downloading Memory…", that is stored at position 128 in its EEPROM memory. The routines that are used for initialising the LCD and to write instructions and characters (`initlcd`, `wrins` and `wrchr` respectively) are all standard routines provided in the PICAXE manuals and can be seen at the end of Appendix F.

4.3 Interrupt Routine on 28X

Interrupts are a data transfer technique that is initiated by hardware and having the transfer controlled by the software (Phythian, 2006.1). Generally, interrupts are a moderate speed I/O technique and has moderate hardware costs. However, the main reason for using interrupts is because the software does not have to continually poll an input to check for a change of state. The other possible choices for transferring the data would have been Direct-Memory-Access (DMA) or programmed I/O. Neither of these would have been sufficient as DMA requires external hardware to implement and therefore is more costly and programmed I/O would not be fast enough and would use an excessive amount of processor time.

The PICAXE 28X is able to handle interrupts on its input port, however, by default there are no interrupts enabled on the device. Thus during the initialisation routine the interrupt is enabled, so that it becomes active when pin0 goes high. The general sequence of events that occur when handling an interrupt is as follows:

- Control is transferred to the service routine
- Service of the input/output request
- Acknowledge the source of the interrupt
- Resume original operation
  (Phythian, 2006.1)

The 28X works slightly different as it is a data transfer sequence as opposed to an input/output routine. This means that the 28X handles an interrupt as follows:

- Control is transferred to interrupt service routine
- Acknowledge is given to source of interrupt
- Request is serviced
- Normal operation resumes

Once the interrupt routine has been initiated the 28X, the chip runs a simple branch test to determine what action it should take. This action will depend on what code was sent to the 28X from the 18X; a ‘0’-‘3’ will cause it to retrieve data and send that data back to the 18X, while a ‘4’ will cause the 28X to start downloading data a PC.

4.3.1 Retrieve Latest Data

A control code of ‘0’ to ‘3’ from the 18X causes the 28X to retrieve the latest data captured from the respective sensor (sensor0 through to sensor3). For the 28X to achieve this, it needs to first read the external EEPROM at the address associated with the respective sensor. So, for sensor2, the 28X would have received the control code ‘2’ and to find that latest data from sensor2 it has to read the latest address and add to it the required offset (`memoffset` x 6; see Figure 4.2). Using this address, it then reads the data from that location in the external EEPROM via the I^2C bus and stores to the variable `dsend`. The 28X then moves onto the next routine, `dsendout`.

4.3.2 Send Data to 18X

Using the data now stored in `dsend` and the process of inter-PICAXE communications (as described in Chapter 4.4) the 28X is able to communicate with the 18X and transmit the data to it serially. Once the 18X confirms that it has received the data successfully, the 28X moves onto the routine `finterrupt`, which finishes the interrupt by pulling the previous values of the variables used from the stack. It is a simple matter of then re-enabling the interrupt for input1 and using the command `return` to return the microcontroller to its data logging mode.

4.3.3 Downloading Data to PC

Downloading the data to the PC is one of the more involved procedures on the 28X. Its job is to reverse the process of storing the captured data in such a fashion that the end-user is able to read and understand the final information. Not only must the 28X transfer the information out of the external EEPROM, it must also send the information in such a way that the software on the PC can understand and save it. From this it can be seen that there are two main components of this section: the PICAXE 28X memory dump routine and the PC serial-in software.

Memory Dump Routine

When the 28X receives a control code of '4' from the 18X, it branches to the memdump routine, which transmits all of the captured data, from the oldest data point to the newest. The oldest data point in the memory, provided that the data has already been filled once, is simply the current address plus one (see Figure 4.3 for a graphical representation). Data is transmitted on output7 at a baud rate of 4800, inverted, with eight data bits, no parity and one stop bit. The format of the serial communications (eight bits, no parity, and one stop bit) is a set PICAXE protocol that is hard-coded into the BASIC routines and cannot be changed. The rate of 4800 is the fastest rate available on the PICAXE 'X' parts. The data itself is transmitted in the format as follows: Month, Date, Hour, Minute, Sensor0, Sensor1, Sensor2, Sensor3; the inclusion of the commas to separate 'columns' of data allows the PC software to store the data as a "Comma-Separated-Variable" file.

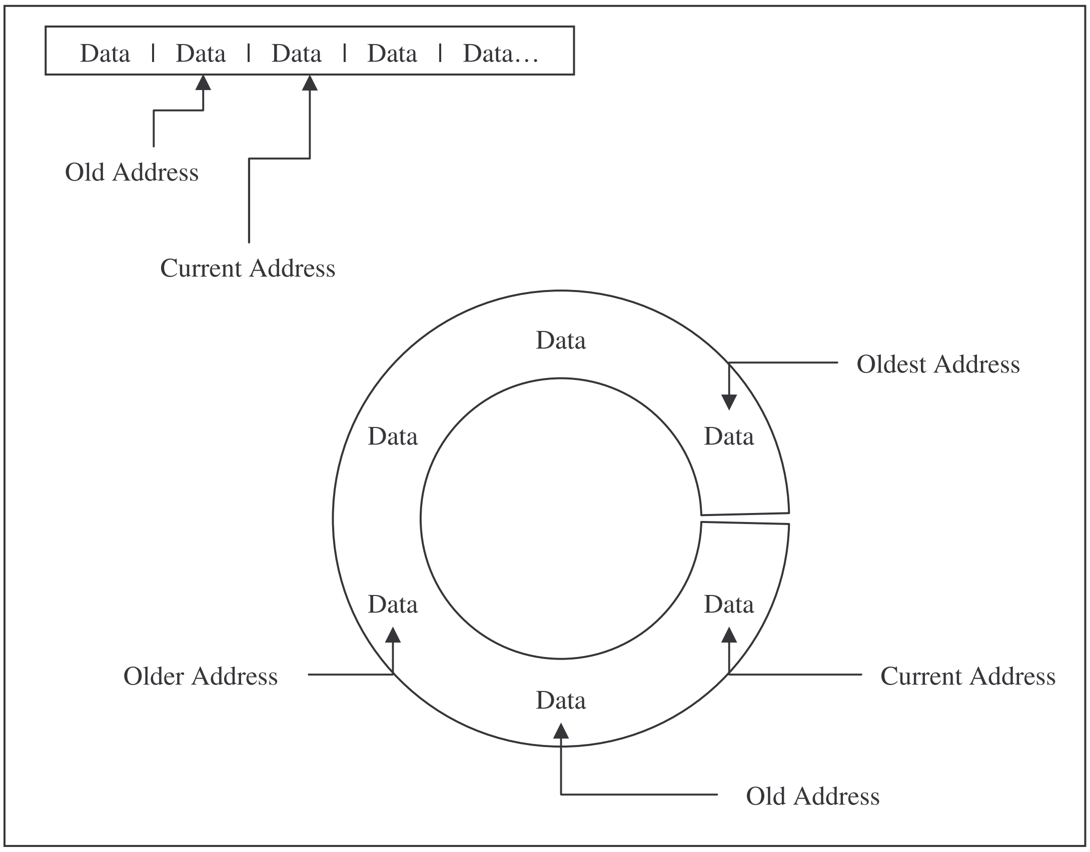


Figure 4.3: Graphical representation of data overwrite structure

To start the data transmission, the 28X sends the titles of the data columns, each separated by a comma, with a return and a line feed at the end, which sets the pointer at the start of the next line. Next the microcontroller reads the data from the EEPROM via the I^{2}C bus so that each piece of data is stored in its own variable (month in `month`, date in `date`, etc.). These variables are then transmitted in the same way as the titles were, with the contents of each variable being transmitted as an ASCII value followed by a comma and have a line feed and return at the end of the variables. The data is transmitted as ASCII values so that values, such as $AF, will be displayed as real decimal values, in this case $AF is 175. Once this has been transmitted, the 28X loops back to the start of the `memdump` routine where it increments address and repeats the process until the value of the address is equal to the address of the first data read. It is at this point that the microcontroller has finished downloading the memory and sends a message of "Download Complete" to the PC so that the user knows to close the program. To finish servicing the interrupt, the 28X branches to `finterrupt` and as in Section 4.3.2, it re-enables the interrupt for input0 and returns to normal operation.

PC Software

At this stage of the design, the PC software is very simplistic, yet performs all the tasks that are required. A small command-line serial terminal program (Schmidt, 2001) was used as a basis for receiving and capturing the data transmitted to the PC. When executed this program is able to store the data that it receives into a document. A comma-separated-variable file format was chosen as this format is easily opened by Microsoft Excel. A copy of the "Readme" file that was created to go with this PC software is included in Appendix G.

4.4 Inter-PICAXE Communication

Due to the way that the tasks are split up between the two PICAXE chips, it is necessary for them to be able to communicate between themselves. To facilitate this, the necessary code was written to allow for serial communication between the two PICAXE's, with the 28X acting as the 'master' and the 18X working as a 'slave' (See Figure 4.4). The communication process works as follows:

The 18X sends a request for communication by pulling Output1 (pin 7) high and then waiting until it receives a "|" on its Input1 (pin 18); when this happens it initiates an interrupt on the 28X via its Input0 (pin 11). The 28X then acknowledges that it has seen the interrupt by sending the "|" character to the 18X out on Output0 (pin 21) and this allows the 18X to continue. The 18X sets in Output1 to low and then pauses for a tenth of a second to allow for the 28X to finish its transmission and be ready to receive data. At this point, the 18X then is able to send the request code to the 28X, which is just the number that was pressed on the keyboard. The request code is compared to a table by the 28X and depending on what it is, it will either send a value back to the 18X or it will initiate an external EEPROM memory dump. If the request code is 0, 1, 2 or 3 it will send the current value of the respective sensor to back to the 18X and then waits for the acknowledge signal from the 18X. If the information is received correctly, the 18X will send an acknowledge signal back to the 28X allowing it to continue servicing the interrupt and the 18X will display the information received on the LCD, then return and wait for another key press.

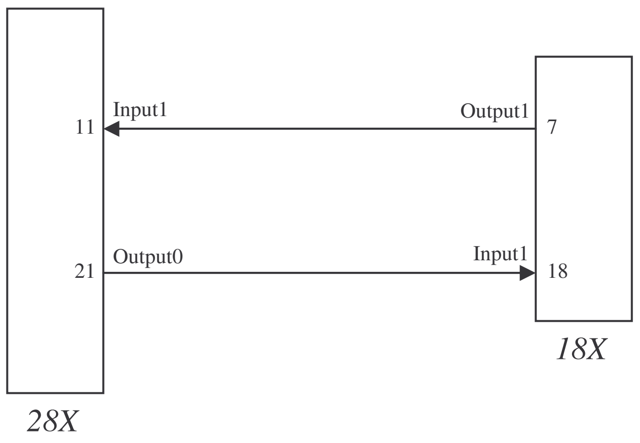


Figure 4.4: Inter-PICAXE Communication Wiring

CHAPTER 5 Evaluation and Testing

To evaluate the performance of any system, there is a need for it to be tested for its response to an expected set of inputs as well as its ability to handle any unexpected signals. For the prototype that was built in this project there were two main areas of testing performed on the device; simulated bench testing and field trials.

5.1 Bench Testing

While a system is being developed, it is common practice to run simulated bench tests on the device to evaluate its performance. This normally occurs with a series of simulated inputs being given to the device while the outputs are recorded and compared to the theoretical expectations.

5.1.1 Testing Site Setup

The bench tests were performed with the circuit constructed on a “breadboard”, which is a type of solderless testing bed that allows for semi-permanent connections between components and the microcontrollers. A 9vdc “plug-pack” transformer was used to power the circuitry by running it through a 5v voltage regulator, such as the KA7805 (Fairchild Semiconductor, South Portland, Maine U.S.A.), which allows for an easily produced Transistor-Transistor-Logic (TTL) power supply. It was also necessary to have a PC connected to both the 18X and the 28X to allow for downloading edited and debugged programs. The PC also acted as the PC that would be used by the farmer to download the captured data. Subsequently, the PC that was used had dual serial ports, was running Windows XP and both the serial communication program and the PICAXE Programming Editor.

5.1.2 Input/Output Configuration

To create a simulated set of inputs for the data logger, two rotary potentiometers (variable resistors; Figure 5.1 shows an example) were used, with the power rails connected to the supply of the microcontrollers. This type of connection allows for the manual production of a variable voltage between zero and five volts, which is in the necessary range for the ADC on the PICAXE to read the signal, meaning the actual value then stored by the PICAXE will be a reading of 0-255. To simplify readings and wirings, the first potentiometer acts as the input for Sensor0 and Sensor2, while the second potentiometer is used for Sensor1 and Sensor3. As previously mentioned in Section 3.3.1, the LCD used allowed for an output of two lines by sixteen characters, giving enough space to display current bore statistics and other messages.

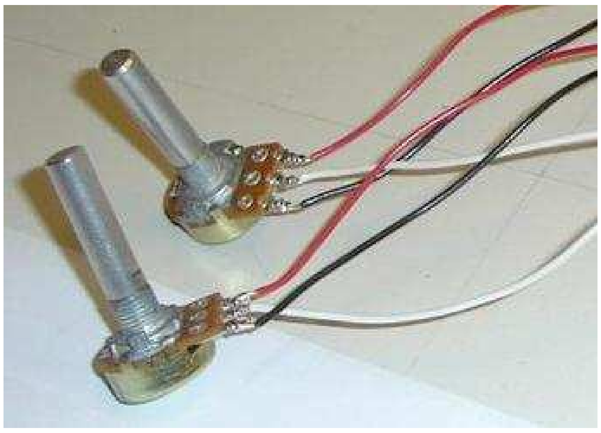


Figure 5.1: Potentiometers used in prototype

5.1.3 Test Procedure and Results

Testing was done to determine whether the designed system was able to perform as described in the Project Specification (listed in Appendix A). This means that there were different tests for each of the four main sections.

Test 1 – Data Logging

Firstly, the 28X was tested on its ability to capture, store and retrieve the data from the sensors. The procedure for this test was simply setting one potentiometer to almost zero and the other at a relatively high number. Note that the first potentiometer must not be actually set at zero as then it would not be possible to distinguish between a true reading and a failed reading. With the potentiometers set, the program was sent to the 28X, with it set to capture 100 readings, with a one second interval between each reading. The timing of the delay was controlled with the RTC. After the readings had completed, the memory of the external EEPROM was downloaded via the PC serial transfer program and saved to a comma-separated-variable file.

Test 1 – Results

Initially, the preliminary test was done without the RTC as the delay mechanism and the `pause` command used to create a delay. This test failed as the internal software read/write check (see Section 4.1.4) continually produced an error due to the value being sent to the EEPROM and the value read immediately back from the EEPROM were not the same. After a large amount of extra research was performed, it was found that the original datasheet provided by Revolution Education for the construction of an 18X Data Logging Chip (AXE110, 2004) used an incorrect circuit schematic. It was found that in the original schematic, the pin numbers for the I^2C Serial Data Line and the I^2C Serial Clock Line were swapped when compared to the pinout of the 28X chip itself. Once these lines were swapped, the data was saved to the EEPROM without error.

With this working, the RTC was then used to create the time delay, rather than the `pause` command. Again, the software error occurred when the data was attempted to be read back from the external EEPROM. After analysis of the circuit using a multi-meter and an oscilloscope, the error was found to be being caused by high frequency noise occurring on the I^2C bus thus giving erroneous results. It was for this reason that the smoothing capacitors were added before the power supplies of the microprocessors and the other IC's in the final design (see Section 3.4.1). Using this new design, the test was restarted and new data was collected for which a portion is displayed in Table 5.1

below. This test gave valid results and verified the four main functions of the data logger: the ability to read data from the sensors, save the data accurately to the EEPROM, retrieve the data and send in to the PC. As can be seen, the data logger responded as expected with Sensor0 and Sensor2 both having the same low value and Sensor1 and Sensor3 both having the same high value.

Table 5.1: Extract of data returned from first test

Month	Date	Hour	Min	Sensor 0	Sensor 1	Sensor 2	Sensor 3
9	20	14	48	12	219	12	219
9	20	14	48	12	219	12	219
9	20	14	48	12	219	12	219
9	20	14	48	12	219	12	219
9	20	14	48	12	219	12	219
9	20	14	48	12	219	12	219
9	20	14	48	12	219	12	219
9	20	14	48	12	219	12	219
9	20	14	48	12	219	12	219
9	20	14	48	12	219	12	219
9	20	14	48	12	219	12	219
9	20	14	48	12	219	12	219
9	20	14	48	12	219	12	219

Test 2 – Keyboard and LCD

The second test evaluated the performance of the communication between the keyboard, PICAXE 18X and the LCD module. A test program was loaded to the 18X that found the value of the key pressed on the keyboard and compared it to an array of ASCII values stored in the 18X's onboard memory. This ASCII value could then be sent serially to the LCD, so that the key pressed would be registered on the LCD – a press of an "a" displayed the letter "a" on the LCD. This process was continually looped so that it was not possible to stop it unless the microprocessor was turned off.

Test 2 – Results

After testing a number of different keys, it was found that all of the alphabetic keys responded as expected as well as the numeral keys on the number pad. The miscellaneous keys on the number pad (/, *, -, + and "enter") also responded correctly. However, it was noted that use of the "Num Lock" key produced mixed results and further investigation of the PICAXE manual found that the response of the number pad was documented to respond erratically depending on the state of the "Num Lock" LED (Revolution Education, 2004, vol. 2, p 38). With this information it was decided that the "Num Lock" key would not be used as an input key or otherwise.

Test 3 – Inter-PICAXE Communications

The third test involved the testing of the inter-PICAXE communications, which are described in Chapter 4.4. The `debug` command was used rather extensively in this test as it is able to send back to the PC all the values that are contained in the PICAXE's internal variable memory. Due to the `debug` command working via the serial connection to the PC, the test had to be performed in two parts; first the 28X receiving data that was sent to it and then the 18X receiving its data back. A hard-coded value of "1" was sent to the 28X, meaning that it was commanded to find the latest information for Sensor1 and transmit that back to the 18X, which would then display this value on the LCD. A second part was carried out so that the 28X was sent a "4", which is the command to initiate an external EEPROM dump.

Test 3 – Results

With the use of the interrupt procedure to start either the data transfer or the memory download, it was important to ensure that entry into the interrupt and the subsequent exit from the routine worked perfectly, otherwise the 28X may become trapped inside the routine and not return to the critical procedure of data logging. When first tested, it was found that the 28X received its command successfully (whether that was to return data or dump the memory) and performed the required action. Once completed, the 28X returned to its main routine, yet if a second interrupt was attempted (without the 28X being reset), the microcontroller would not register the interrupt. It was found that this behaviour was due to the microcontroller disabling the interrupts when entering an interrupt routine – this is so that the interrupt source will not be recognised multiple times by error (Revolution Education, vol. 2, p 79). A single line of code, used to enable the interrupt, was added just before the `return` command so as to allow the interrupt to be used more than once. With this error resolved, the rest of the test found the system performing to expectations, with the 18X displaying the correct value for each sensor and the 28X downloading the EEPROM on command.

Test 4 – Complete System

The final test performed, was a test of the whole system working together. Meaning, that the user would be able to press a number on the key pad and the 28X would stop its data logging and either start downloading the EEPROM memory or the 18X would display the current value of the sensor corresponding to the number pressed. Thus, it was this procedure that was tested.

Test 4 – Results

When examining a complete system, it is important that the system can handle a variety of inputs, including erroneous ones. With both intended (0-4 on the number pad) and non-intended key presses (any other key) saw this system working correctly. When from 0-3 was pressed the most recent value from the respective sensor was displayed on the LCD and when 4 was pressed, the 28X initiated its download routine, which produced a comma-separated-variable file on the connected PC. If a key was pressed in error, such as a 7 or a alphabetic key, the 18X registered it as an outside it's know range and notified the user and ignored that key press.

5.2 Field Trials

As mentioned previously, there is a need for on-site, field trials, as they are able to test the response of how the system would operate in real life use. This type often accounts for the toll that small changes, such as temperature and vibration, have on electronic components. Unfortunately, due to time and other circumstantial constraints, field tests were not conducted on this system. However, the following was actually carried out in preparation of testing.

5.2.1 Site Description

Multiple sites were considered for field testing of this system, including a small property at Rosevale and the Agricultural Plot that is associated with the University of Southern Queensland. Ultimately, the Agricultural Plot was chosen for its locality and existing equipment, as the site runs a small motor and an existing flow meter. Section 5.2.2 discusses the sensors that were installed on the site and their specifications.

The power that was provided on-site was a single-phase mains AC outlet. This was used to provide the power for a compressor for the bubbler line as well as to the data logging system. To power such a system that contains both high current drain sensors and low voltage microcontrollers, it becomes necessary to consider how each component will be supplied with power. As discussed in Chapter 3.4, most microcontrollers and other IC's run from an extremely well regulated 5V power supply.

5.2.2 Sensors Used – Typical Sensor Set

The set of sensors that were prepared for testing included the following:

• Line Pressure:	Druck PTX 1400 Pressure Transmitter (General Electrics, Boston, USA, 2005)
• Bubbler Pressure:	Druck PTX 1400 Pressure Transmitter (General Electrics, Boston, USA, 2005)
• Flow Rate:	Kent H4000 Woltmann cold water meter (Elster Metering Limited, Bedfordshire, UK, 2005)
• Temperature:	Dallas Semi. DS18B20 1-wire digital thermometer (Dallas Semiconductor Corporation, Texas, USA, 2006)

These four sensors were chosen to showcase the system's ability to handle almost any input type that was thrown at it as they are a set of typical sensors that are available on today's market.

As shown in Table 5.2, there are three main outputs, namely pulsed, analogue voltage, and a current loop. These outputs are the most common outputs that will be found on a bore installation. Thus for the original testing purposes, we are able to use just the three output types stated below.

Table 5.2: Sensor Specifications (**sourced from Elster H4000, Druck limited ptx/pmp 1400, Dalas Semi. DS18B20 datasheets**)

Sensor	Output	Input requirements	Existing Hardware
PTX 1400	4-20mA	9 to 28 V	New
H4000	Pulsed Voltage	4.5 to 16 V	Similar existing
DS18B20	0-5V	3 to 5.5 V	New

5.2.3 Proposed Test Procedure

The procedure that would have been used to test this system would have been similar to test four used in Section 5.1.3, where the entire system is run with all components active. More precisely, the data logging system would be activated when the pump is started and the system would start data logging immediately. Once the pump has had a chance to settle at an appropriate speed, the user would test to find the current values for each of the sensors. The pump would then be allowed to run for an extended period (overnight or all-day) and a similar procedure would be used when the pump was shut down. This could then be repeated for a few days to a week, giving an accurate account of the performance of the system in different environments (day/night, high vibration when the pump is running). Once this series of tests have been completed, the data that was captured could be downloaded and analysed to find the performance of the system. Depending on accuracy of the captured data from the data logger, conclusions could be drawn on how well the system performed in a real-life environment.

CHAPTER 6 Conclusion and Recommendations for further development

This document has outlined and discussed the design of a data logging system as previously described in Chapter 1. It provides a means of evaluating the performance of an irrigation pumping unit to allow the farmer to better understand how well the underground aquifer is performing, whether the pump or pipes need maintenance and provides a simple way to record water use over time. Compared to current market options for a similar device, this prototype came in at a fraction of the costs.

The current system allows for a data capture and storage for up to four sensors for a period of two months. This is based on a proof-of-concept set of hardware and can be easily upgraded to allow for capture from eight sensors and store data for over twelve months. The system also displays the information to the user and allows for the user to make a selection on what information is being displayed. However, contrary to the Project Specification, this system does not allow for the user to control the functions of the motor (speed, etc.). This feature was not implemented due to time constraints. Software was also written to provide download capabilities to the users PC so that a database of records can be kept.

Future work could be carried out on this project so as to provide extra features as well as field testing on the system. The current system splits the task of data logging and the user interface between two microcontrollers, with serial data transfer happening between them. This setup allows for further separation of the components so that conceivably, the data logging element would be situated on the pumping site and broadcasting information to user control element, which could then be located at the residence of the user or even made portable. Extra work could also be done on creating a printed circuit board (PCB) for the system, as the current model uses a semi-permanent breadboard. Likewise, the current housing does not meet the design specification of being waterproof and rugged and so could be improved upon. This system could also be modified to work in other areas of monitoring, such as stock water or other industrial applications.

This project provides a simple and cost effective solution for a farmer that wishes to record and monitor the performance of an irrigation pumping unit.

References

Australian Bureau of Statistics, 2006, *Water Account 2004*, viewed 17 May 2006, <http://www.abs.gov.au>.

Dallas Semiconductor 2006, *DS18B20 Programmable Resolution 1-Wire Digital Thermometer: Datasheet*, Maxim Integrated Products, Dallas, Texas, viewed 1 November 2006, <http://www.maxim-ic.com/>.

Druck Limited 2006, *PTX/PMP 1400 Industrial Pressure Sensors: Datasheet*, General Electric Company, Boston, MA, viewed 1 November 2006, <http://www.gesensing.com/>.

Elster Metering Limited 2006, *H4000 Woltmann cold water meters: Datasheet*, Elster Metering Limited, Bedfordshire, United Kingdom, viewed 20 October 2006, <www.elstermetering.com>.

Fairchild Semiconductor International 2006, Maine, USA, viewed 15 October 2006, <http://www.fairchildsemi.com/>.

GlobalSpec, 2006, Engineering Search Engine, *Common Flow Meters*, viewed 21 May 2006, <http://flow-meters.globalspec.com/>.

Microchip Technology Inc. 1998, *24AA256/24LC256 CMOS Serial EEPROM: Datasheet*, Microchip Technology Incorporated, USA, viewed 12 August 2006, <http://www.microchip.com>.

MicroZed Computers Pty. Ltd. 2006, Revolution Education, United Kingdom, viewed 20 October 2006, <http://www.microzed.com.au/>.

Phythian, M 2006, *ELE2303 Embedded Systems Design: Study Book*, University of Southern Queensland, Toowoomba

Revolution Education Ltd. 2006, Revolution Education Limited, Bath, United Kingdom, viewed 15 June 2006, <http://www.rev-ed.co.uk>.

Shultz, W 1997, *Interfacing Semiconductor Pressure Sensors to Microcomputers*, Motorola Inc, Colorado.

Yiasoumi, B. (Irrigation Officer, New South Wales Department of Primary Industries), *Agfact E5.8*, Third Ed, viewed 14 November 2003, <http://www.agric.nsw.gov.au/reader/4061>.
Young, F 2006, *ENG4111/4112 Project Reference Book*, University of Southern Queensland, Toowoomba

Appendices

Appendix A Project Specification

University of Southern Queensland – Faculty of Engineering and Surveying

ENG 4111/4112 Research Project

PROJECT SPECIFICATION

FOR:	Daniel WHITTRED
TOPIC:	*DEVELOPING A MONITORING AND CONTROL SYSTEM FOR IRRIGATION PUMPING UNITS*
SUPERVISOR:	Dr. Steven Raine
SPONSORSHIP:	Faculty of Engineering, USQ; NCEA
PROJECT AIM:	*To design and build a system that senses different information about an irrigation pumping system; The signals from each of these sensors are to be then integrated into a unit that logs the data received and displays the relevant information to the user.*
PROGRAMME:	Issue A, 24 March 2006

1. Research background information on current sensing techniques to find the range of possible inputs that can be expected in both a 'retro-fit' application and a new installation.
2. From background review, decide on an appropriate micro-controller that will accept the necessary range of inputs as well as be able to convert expected analogue signals to digital signals. The micro-controller will also need to be able to write and retrieve from external memory. Other outputs will be needed to allow communication with the device either via a remote link or directly to a serial port of a pc. Specifically:
    a. Sensors will include flow meters, pressure sensors and sensors to measure the efficiency of the motor, whether that is a diesel or electric motor.
    b. Data storage will need to be of a size that will allow for around 6 months of data.
    c. Data needs to be displayed in such a way that the user can view all necessary information immediately by a display that will communicate with main system.
3. Test the system with simulated data and adjust the system as necessary.
4. Allow user to be able to set flow or pressure rates with keypad or toggle (+/-) buttons and the microcontroller will adjust the motor and flow to achieve the desired pressure.
5. Design PC software to analyse long term data collected and archive.

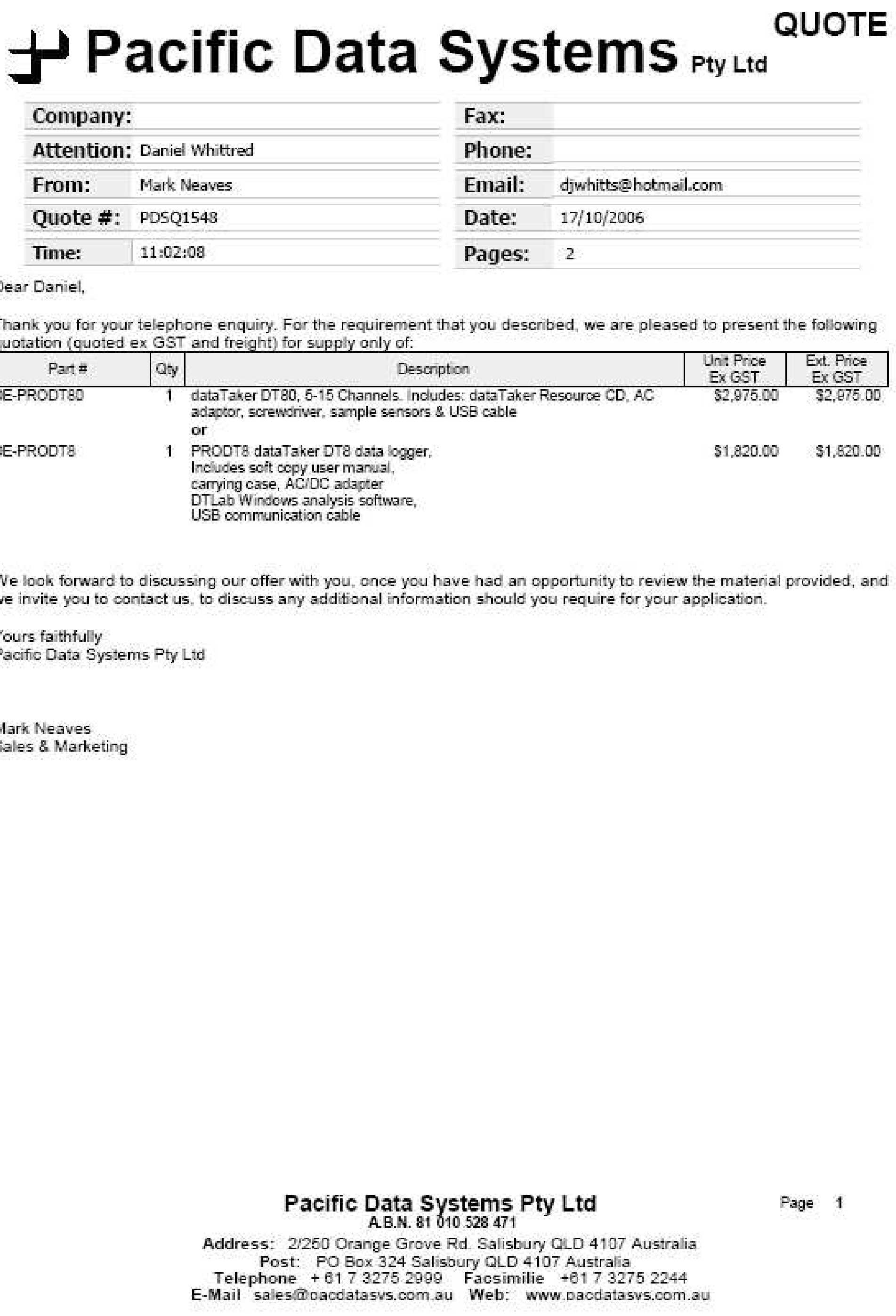
As time permits:

6. Code internal algorithms to make the system self diagnosable and able to warn the user of pending problems.
7. Design the system so that it can be integrated into the current NCEA 'Irrimate' sensors.

AGREED: ___________________ (student) _____________________ (Supervisor)

____ / ____ / ____ ____ / ____ / ____

Appendix B Data Logger Quote



Pacific Data Systems Pty Ltd

QUOTE

Company:		**Fax:**	
Attention:	Daniel Whittred	**Phone:**	
From:	Mark Neaves	**Email:**	djwhitts@hotmail.com
Quote #:	PDSQ1548	**Date:**	17/10/2006
Time:	11:02:08	**Pages:**	2

Dear Daniel,

Thank you for your telephone enquiry. For the requirement that you described, we are pleased to present the following quotation (quoted ex GST and freight) for supply only of:

Part #	Qty	Description	Unit Price Ex GST	Ext. Price Ex GST
DE-PRODT80	1	dataTaker DT80, 5-15 Channels. Includes: dataTaker Resource CD, AC adaptor, screwdriver, sample sensors & USB cable or	$2,975.00	$2,975.00
DE-PRODT8	1	PRODT8 dataTaker DT8 data logger, Includes soft copy user manual, carrying case, AC/DC adapter DTLab Windows analysis software, USB communication cable	$1,820.00	$1,820.00

We look forward to discussing our offer with you, once you have had an opportunity to review the material provided, and we invite you to contact us, to discuss any additional information should you require for your application.

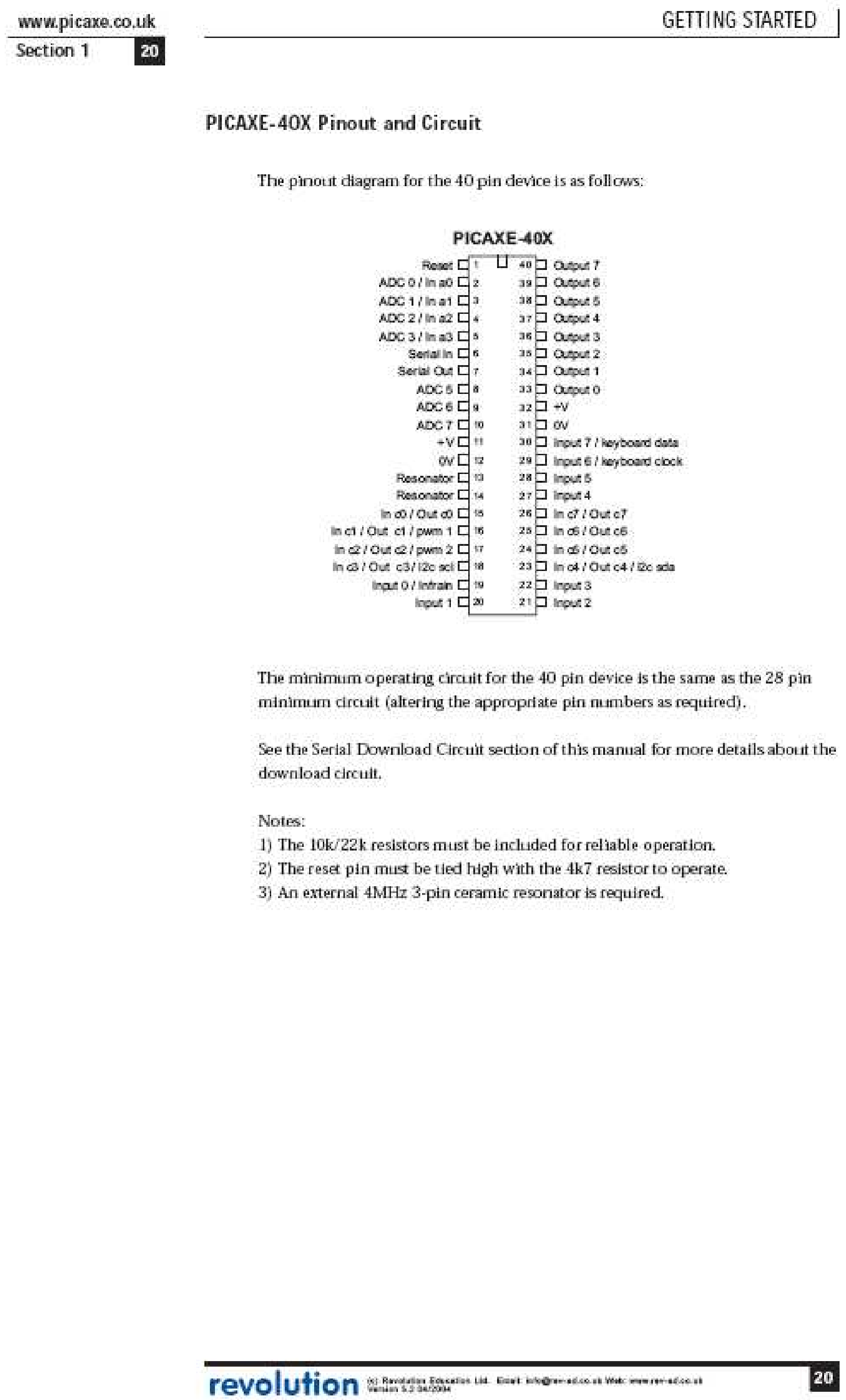
Yours faithfully
Pacific Data Systems Pty Ltd

Mark Neaves
Sales & Marketing

Pacific Data Systems Pty Ltd
A.B.N. 81 010 528 471
Address: 2/250 Orange Grove Rd. Salisbury QLD 4107 Australia
Post: PO Box 324 Salisbury QLD 4107 Australia
Telephone +61 7 3275 2999 Facsimilie +61 7 3275 2244
E-Mail sales@pacdatasys.com.au Web: www.pacdatasys.com.au

Page 1

Appendix C Selected PICAXE Chip Pinouts

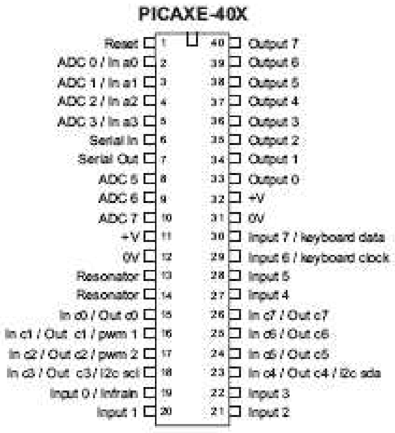


www.picaxe.co.uk
Section 1 20

GETTING STARTED

PICAXE-40X Pinout and Circuit

The pinout diagram for the 40 pin device is as follows:



The minimum operating circuit for the 40 pin device is the same as the 28 pin minimum circuit (altering the appropriate pin numbers as required).

See the Serial Download Circuit section of this manual for more details about the download circuit.

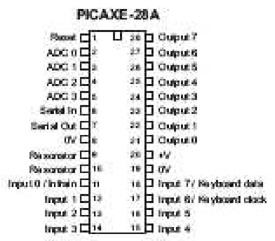
Notes:
1) The 10k/22k resistors must be included for reliable operation.
2) The reset pin must be tied high with the 4k7 resistor to operate.
3) An external 4MHz 3-pin ceramic resonator is required.

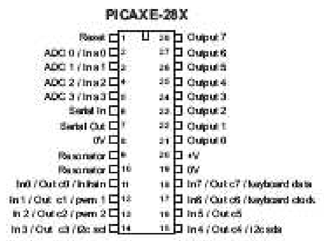
revolution (c) Revolution Education Ltd. Email: info@rev-ed.co.uk Web: www.rev-ed.co.uk Version 5.2 04/2004 20

(Revolution Education, p20, picaxe manual 1, 2004)

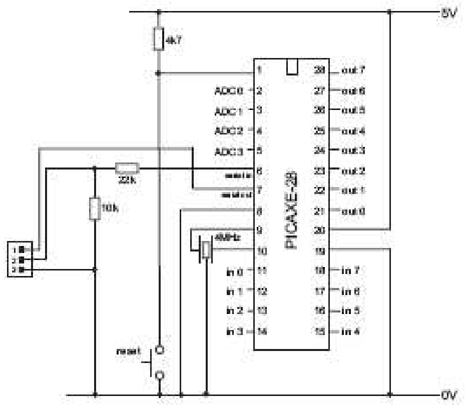
PICAXE-28A/28X Pinout and Circuit

The pinout diagrams for the 28 pin devices are as follows:





The minimum operating circuit for the 28 pin devices is:



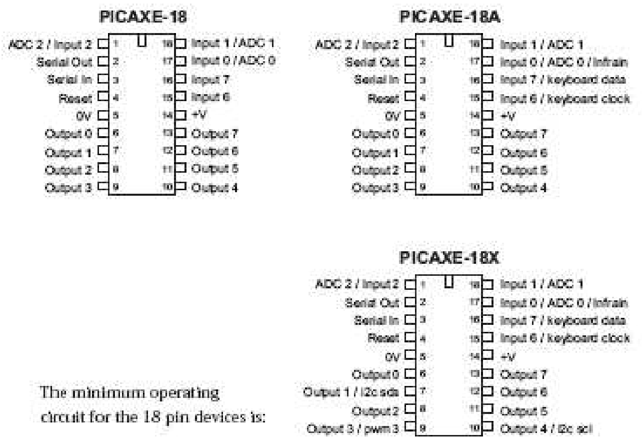
See the Serial Download Circuit section of this manual for more details about the download circuit.

Notes:
1) The 10k/22k resistors must be included for reliable operation.
2) The reset pin must be tied high with the 4k7 resistor to operate.
3) An external 4MHz 3-pin ceramic resonator is required.

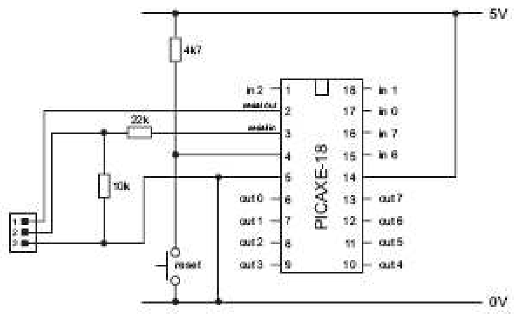
(Revolution Education, p19, picaxe manual 1, 2004)

PICAXE-18/18A/18X Pinout and Circuit

The pinout diagrams for the 18 pin devices are as follows:



The minimum operating circuit for the 18 pin devices is:

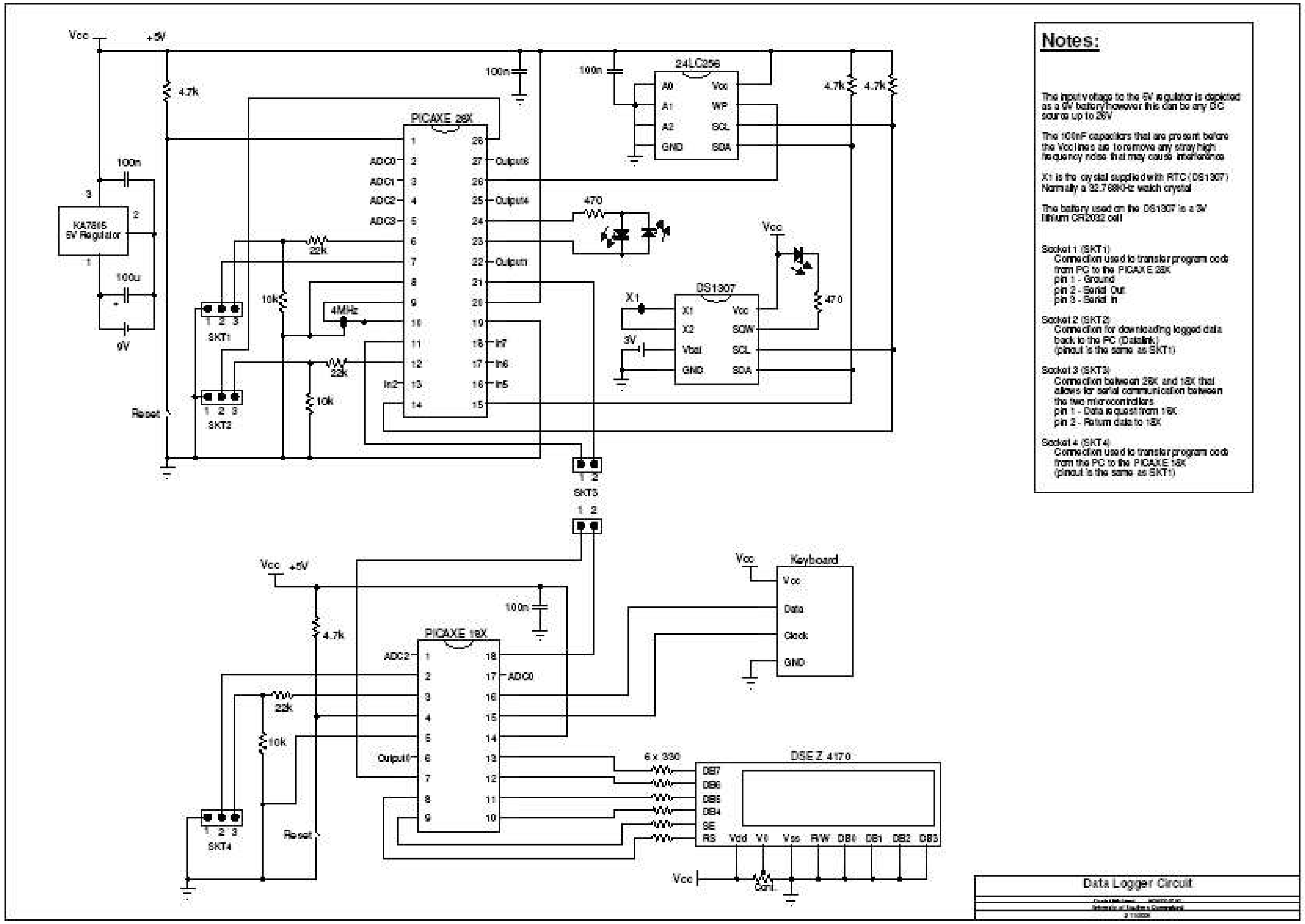


See the Serial Download Circuit section of this manual for more details about the download circuit.

Notes:
1) The 10k/22k resistors must be included for reliable operation.
2) The reset pin must be tied high with the 4k7 resistor to operate.
3) No external resonator is required as the chips have an internal resonator.

(Revolution Education, p18, picaxe manual 1, 2004)

Appendix D Prototype Circuit Diagram



Notes:

The input voltage to the 5V regulator is depicted as a 9V battery however this can be any DC source up to 26V

The 100nF capacitors that are present before the Vcc lines are to remove any stray high frequency noise that may cause interference

X1 is the crystal supplied with RTC (DS1307) Normally a 32.768KHz watch crystal

The battery used on the DS1307 is a 3V lithium CR2032 cell

Socket 1 (SKT1)
Connection used to transfer program code from PC to the PICAXE 28X
pin 1 - Ground
pin 2 - Serial Out
pin 3 - Serial In

Socket 2 (SKT2)
Connection for downloading logged data back to the PC (Datalink)
(pinout is the same as SKT1)

Socket 3 (SKT3)
Connection between 28X and 18X that allows for serial communication between the two microcontrollers
pin 1 - Data request from 18X
pin 2 - Return data to 18X

Socket 4 (SKT4)
Connection used to transfer program code from the PC to the PICAXE 18X
(pinout is the same as SKT1)

Appendix E 28X Data Logger Code

```
'Title - PICAXE 28X 4-channel Data Logger
'Coded by Daniel Whittred


' ***********************************************************************************************
' ***** Options  ********************************************************************************
' ***********************************************************************************************

'Sensors:
'Sensor 0 - Sen0          (leg 2)
'Sensor 1 - Sensor 1      (leg 3)
'Sensor 2 - Sensor 2      (leg 4)
'Sensor 3 - Sen3          (leg 5)

'Memory:
'No of readings = max in memory -> 32k/9bytes
'1 x 24LC256

'Outputs:
'Bi-colour LED (LED will flash green as readings are taken)
'Logging Period:(using DS1307 RTC)
'Hours:  0 Mins:  0 Secs:  1



' ***********************************************************************************************
' ***** Symbols  ********************************************************************************
' ***********************************************************************************************

'Symbol definitions
symbol data0 = b0                 'input from sensor 1
symbol data1 = b1                 'input from sensor 2
symbol data2 = b2                 'input from sensor 3
symbol data3 = b3                 'input from sensor 4

symbol address    = w6            '(b13:b12)       - variable used to describe address for storing
sensor logs
symbol temp_word  = w5    '(b11:b10)       - temporary variable of 'word' length
symbol temp_word2 = w4            '(b9:b8) - another temporary word variable
symbol temp_byte  = b9            'temporary variable of byte length

symbol dsend      = b9            'variable used to hold data to be sent to control picaxe
symbol drequest   = b8            'variable used to hold information about the data that the control
picaxe has requested

symbol month = b4                 'value of the month returned from the RTC
symbol date  = b5                 '   "            " date   "        "        "
symbol hour  = b6                 '   "            " hour   "        "        "
symbol mins  = b7                  '   "             " minutes         "        "        "
symbol sec   = b8                  '   "             " seonds "        "        "

symbol COM   = 44         'constant value for 'comma'
symbol RET   = 13         'constant value for 'carriage return'
symbol LFEED = 10                  'constant value for 'line feed'

symbol memoffset = 70              'value used to split memory into chunks - 4096

'Preload sensor names and title into data memory
EEPROM 0, (0,0)                    'inital starting address
EEPROM 16,("Sensor 0")             'name for first sensor
EEPROM 32,("Sensor 1")             'name for second sensor
EEPROM 48,("Sensor 2")             'name for third sensor
EEPROM 64,("Sensor 3")             'name for fourth sensor
'EEPROM 80,("4-ch Datalogger")'name of title for project




' ***************************************************************************************************
' ***** MAIN  ***************************************************************************************
' ***************************************************************************************************

  main:
'~~~~~~~

'initialise the picaxe
         gosub init

loop:
'read in data
         high 3                    'turn on green LED to signify start of data sample
         gosub read_data           'sample sensor values
         gosub read_time_now       'read current time/date

'save data to external eeprom
         gosub save_data
         low 3                     'end of flash LED and sampling period

'increment address and save in onboard eeprom
         let address = address + 1                   'increment address
         if address <= memoffset then continue       'check whether address is still less than
                                                     'max allowed, if so, jump to 'continue'

'memory full -  loop back and overwrite
         let address = 0           'initialise memory address back to zero

continue:
         gosub save_data

         write 5,b12               'save low byte of address to onboard eeprom
```

```
        write 6,b13             'save high byte of address to onboard eeprom

        gosub time_delays       'implement time delay

'now loop back and do next reading
        goto loop



' *****************************************************************************************************
' ***** Sub-routines  *********************************************************************************
' *****************************************************************************************************

  init:
'~~~~~~~
'Called at the start of the main program to initialise paramaters
'such as interrupts and starting eeprom address

        setint %00000001,%00000001'set interrupt to activate on a low on input pin0
        high 5                           'write protect eeprom

'reload the last address from data memory
        read 5,b12                       'store into low part of variable 'address'
        read 6,b13                       'store into high part of address

        return



  read_data:
'~~~~~~~~~~~~
'Performs the main data reading function where data is read in from each
'input

        readadc 0,data0         'read adc pin0 and store value to data0
        readadc 1,data1         '       "   pin1 "       "       "   data1
        readadc 2,data2         '       "   pin2 "       "       "   data2
        readadc 3,data3         '       "   pin3 "       "       "   data3

        return



  save_data:
'~~~~~~~~~~~~
'Take values contained in data0-data3 and stores them with a
'timestamp into the external eeprom

        low 5                                   'write enable external eeprom

        i2cslave %10100000, i2cfast, i2cword    'initialise external eeprom on i2c bus
```

```
temp_word = address                        'assign a temporary address value of 'address'
writei2c temp_word,(month)                 'store the current value of 'month' into the
                                           `external eeprom
pause 10                                   'wait the write time of 10ms
high 5
readi2c temp_word,(temp_byte)              'read back the just stored value
if temp_byte <> month then ee_error        'compare the read back value to the original –
                                           `if they are not the same then branch to an
                                           `error routine
low 5
temp_word = address + memoffset            'as above but with an address offset of
                                           `'memoffset'
writei2c temp_word,(date)                  '        etc....
pause 10
high 5                                     '        |
readi2c temp_word,(temp_byte)              '
if temp_byte <> date then ee_error         '        |

low 5                                      '
temp_word = address + memoffset*2          '        |
writei2c temp_word,(hour)                  '
pause 10
high 5                                     '        |
readi2c temp_word,(temp_byte)              '        V
if temp_byte <> hour then ee_error         '

low 5                                      '        |
temp_word = address + memoffset*3          '
writei2c temp_word,(mins)                  '        |
pause 20
high 5                                     '
readi2c temp_word,(temp_byte)              '        |
if temp_byte <> mins then ee_error         '

low 5                                      '        |
temp_word = address + memoffset*4          '        V
writei2c temp_word,(data0)                 '
pause 20
high 5                                     '        |
readi2c temp_word,(temp_byte)              '
if temp_byte <> data0 then ee_error        '        |

low 5
temp_word = address + memoffset*5          '        |
writei2c temp_word,(data1)                 '
pause 20
high 5                                     '        |
readi2c temp_word,(temp_byte)              '        V
if temp_byte <> data1 then ee_error        '
```

```
        low 5                                   '       |
        temp_word = address + memoffset*6       '
        writei2c temp_word,(data2)              '       |
        pause 20
        high 5                                  '
        readi2c temp_word,(temp_byte)           '       |
        if temp_byte <> data2 then ee_error     '

        low 5                                   '       |
        temp_word = address + memoffset*7       '       V
        writei2c temp_word,(data3)              '
        pause 20
        high 5                                  '       |
        readi2c temp_word,(temp_byte)           '       V
        if temp_byte <> data3 then ee_error     '

        high 5                                  'write protect eeprom

        return



  time_delays:
'~~~~~~~~~~~~~~
'Reads the time from the RTC and adds an offset (length of delay), then checks the
'RTC again to see if is now the time of orignal_time + offset. If it is then it
'is then it moves on, otherwise it keeps checking

'read current time
        gosub read_time_now      'returns currents time and date in decimal format

        let data0 = sec          'temporarily store the time
        let data1 = mins
        let data2 = hour

'add time delay offset of: 0hour:0mins:1sec
Add_Secs:
        let data0 = data0 + 1            'add sec offset to current time
        if data0 < 60 then Add_Mins      'if less than 60sec goto adding mins offset
                                         'otherwise compensate by adding 1min
        let data0 = data0 - 60           'set sec back to 0
        let data1 = data1 + 1            'increment mins

Add_Mins:
        let data1 = data1 + 0            'mins offset
        if data1 < 60 then Add_Hours     'if greater then 60mins, compensate by incrementing hour
        let data1 = data1 - 60
        let data2 = data2 + 1

Add_Hours:
        let data2 = data2 + 0            'hour offset
```

```
        if data2 < 24 then check_time      'if hours less then 24 move on; else reset hour to 0
        let data2 = data2 - 24

'read rtc to test alarm
check_time:
        gosub read_time_now                'get current time/date

        if sec <> data0 then check_time    'test if current time equals desired time
        if mins <> data1 then check_time
        if hour <> data2 then check_time

        return



  ee_error:
'~~~~~~~~~~~
'This routine only runs when a write-back check fails in the save_data routine
'It will occur if there is either an problem with the external eeprom (memory is
'corrupt) or if there is a problem with communication on the i2c bus. Can only be
'resolved by a complete system restart. Also called if there is an error in
'communication with the control picaxe.

'flash LED red/green
        high 2                     'red LED on
        low 3                      'green LED off
        pause 500                  '0.5sec pause
        low 2                      'red LED off
        high 3                     'green LED on
        pause 500                  '0.5sec pause
        goto ee_error              'continually loop back to start of routine



  bcd_decimal:
'~~~~~~~~~~~~~~
'Function to convert BCD values from the RTC to decimal

        let temp_byte = sec & %11110000 / 16 * 10
        let sec = sec & %00001111 + temp_byte
        let temp_byte = mins & %11110000 / 16 * 10
        let mins = mins & %00001111 + temp_byte
        let temp_byte = hour & %11110000 / 16 * 10
        let hour = hour & %00001111 + temp_byte
        let temp_byte = date & %11110000 / 16 * 10
        let date = date & %00001111 + temp_byte
        let temp_byte = month & %11110000 / 16 * 10
        let month = month & %00001111 + temp_byte

        return
```

```
  read_time_now:
'~~~~~~~~~~~~~~~~
'Communicates with the RTC to get the current date and time and returns
'the values converted from bcd to decimal

        i2cslave %11010000, i2cslow, i2cbyte                'set i2c RTC paramaters
        readi2c 0,(sec,mins,hour,temp_byte,date,month)      'get current date/time

        gosub bcd_decimal                 'Convert to decimal

        return



' ***************************************************************************************************
' ***** Interrupt Routine  **************************************************************************
' ***************************************************************************************************

  interrupt:
'~~~~~~~~~~~~
'An interrupt is caused by pin0 going high which is either a command or a request
'for data from the control picaxe. This routine is broken into several more
'subroutines to handle the required flow of information

'stack variables so that they do not get accidently changed
        poke 80,b9               'push onto stack temp_byte
        poke 81,b8               '       "       "    sec
        poke 82,b12              '       "       "    low-byte of address
        poke 83,b13              '       "       "    high-byte of address
        poke 84,b10              '       "       "    lsb of temp_word
        poke 85,b11              '       "       "    msb of temp_word

'send confirmation of interrupt back to control picaxe and ask what data is needed
        serout 0,N2400,("|")                        '"|" confirmation code
        serin 0,N2400,("|"),drequest                'receive data only if preceded by '|' and store
to 'drequest'

'get current external eeprom address
        i2cslave %10100000, i2cslow, i2cword        'set external eeprom i2c bus paramaters

        read 5,b12                                  'read in low part of address of latest data log
                                                    'from eeprom
        read 6,b13                                  'read high part of address

        if address <> 0 then testreq                'loop back from 0 to max memory rather than
                                                    'going into negatives
        let address = memoffset + 1

'test what request code is given by control picaxe
```

```
testreq:
        branch drequest,(d0,d1,d2,d3,memdump)               'DATA Request
                                                                    '0 -> current sensor0 value
                                                                    '1 ->    "         sensor1   "
                                                                    '2 ->    "    sensor2   "
                                                                    '3 ->    "    sensor3   "
                                                                    '
                                                            'CONTROL Request
                                                                    '4  ->  external  eeprom  memory
dump

        goto ee_error             'unknown request code sent from control picaxe - error!

'if data is to be sent back to control picaxe
dsendout:
        serout 0,N2400,("|",dsend)         'send out data (in 'dsend') with a preceding '|'
        serin 0,N2400,("|")                'wait until control picaxe confirms data has been
                                           'recieved


'finish interrupt and restore values
finterrupt:
        peek 80,b9                'pull from stack temp_byte
        peek 81,b8                '       "       "    sec
        peek 82,b12               '       "       "    low-byte of address
        peek 83,b13               '       "       "    high-byte of address
        peek 84,b10               '       "       "    lsb of temp_word
        peek 85,b11               '       "       "    msb of temp_word

        setint 1,1
        return



'****************************************************************************************************
'***** Request Code Sub-Routines  *******************************************************************
'****************************************************************************************************

  d0:
'~~~~~
'Data request for current value (latest value stored) for sensor0

        let temp_word = address - 1 + memoffset*4   'get a temporary address value for previous store
        readi2c temp_word,(dsend)                   'read  value  from  external  eeprom  and  store  in
'dsend'
        goto dsendout                               'send the data to the control picaxe



  d1:
'~~~~~
```

```
'Data request for current value (latest value stored) for sensor1

        let temp_word = address - 1 + memoffset*5
        readi2c temp_word,(dsend)
        goto dsendout




  d2:
'~~~~~
'Data request for current value (latest value stored) for sensor2

        let temp_word = address - 1 + memoffset*6
        readi2c temp_word,(dsend)
        goto dsendout




  d3:
'~~~~~
'Data request for current value (latest value stored) for sensor3

        let temp_word = address - 1 + memoffset*7
        readi2c temp_word,(dsend)
        goto dsendout




  memdump:
'~~~~~~~~~~
'Perform "Datalink" routine to dump all of the external eeprom's contents via the serial connection.
'Use of the terminal program will allow you to save this memory dump to a *.csv file for later
‘analysis.
'Quality of data is checked to prevent the transmission of a large amount of consecutive zero values.
'Memory download starts from the oldest piece of stored data in the external eeprom and continues
‘until it gets to the most recent piece of information

        serout 0,N2400,("|","|")  'send out '|' with a preceding '|'
        serin 0,N2400,("|")               'wait until control picaxe confirms data has been recieved

        let temp_byte = 0         'set a qualifer flag (used later on)
        poke 91,b9

        'pause 800

'transmit the titles
        serout      7,N4800,("Month",COM,"Date",COM,"Hour",COM,"Min",COM,"Sensor      0",COM,"Sensor
1",COM,"Sensor 2",COM,"Sensor 3",COM," ",LFEED)

'reload the last address from data memory
        read 5,b12                                'store into low part of variable 'address'
```

```
        read 6,b13                              'store into high part of address
        let address = 0
        let temp_word2 = address                'hold original address for later
        poke 90,b9

next_reading:
        high 3                                          'green LED on
        let address = address + 1                       'increment address to get oldest data in
memory
        poke 91,b9
        peek 90,b9
        if address = temp_word2 then all_done           'check that we arn't at original
                                                        'address yet
        if address <= memoffset then continue_send      'check that address is not past its
                                                        'limit
        let address = 0                                 'if past, then reset to zero

continue_send:
'read back the data
        high 5                                          'write protect external eeprom
        i2cslave %10100000, i2cfast, i2cword            'set external eeprom i2c bus paramaters

        temp_word = address                             'read data back... etc...
        readi2c temp_word,(month)
        debug
        wait 2



        temp_word = address + memoffset
        readi2c temp_word,(date)
        debug
        wait 2

        temp_word = address + memoffset*2
        readi2c temp_word,(hour)
        debug
        wait 2



        temp_word = address + memoffset*3
        readi2c temp_word,(mins)
        debug
        wait 2



        temp_word = address + memoffset*4
        readi2c temp_word,(data0)
        debug
        wait 2
```

```
         temp_word = address + memoffset*5
         readi2c temp_word,(data1)
         debug
         wait 2


         temp_word = address + memoffset*6
         readi2c temp_word,(data2)
         debug
         wait 2


         temp_word = address + memoffset*7
         readi2c temp_word,(data3)
         debug
         wait 2

'check for quality of data - is there more than three consecutive 'zero' readings?
         peek 91,b9
         if data0 <> 0 or data1 <> 0 or data2 <> 0 or data3 <> 0 then goto good_data    'not zero is
                                                                                          `good data
         let temp_byte = temp_byte + 1              'increment qualifier
         if temp_byte < 3 then goto ok_data         'if not third consectutive zero, but is
                                                    `after a zero, then ok data
         let temp_byte = temp_byte - 1              'bad data (all zeros) - decrement qualifier
         goto next_reading                          'read from next address but dont send data

good_data:
         let temp_byte = 0          'make sure qualifier equals zero

ok_data:                            'data is still sent, but qualifier is allowed to increase
         low 3                      'turn green LED off

'transmit data
         serout
7,N4800,(#address,COM,#month,COM,#date,COM,#hour,COM,#mins,COM,#data0,COM,#data1,COM,#data2,COM,#data3
,COM,LFEED)

         goto next_reading          'next reading



  all_done:
'~~~~~~~~~~~
'memory dump is complete - return to main interrupt routine
         low 3                                      'make sure gren LED is off
         serout 7,N4800,("Download Complete")
         goto finterrupt                            'return to main interrupt routine

`EOF
```

Appendix F 18X Control Code

```
******** Definitions ************************

symbol lcdd      = b1
symbol dispdata  = w5
symbol length    = b2     'length of data to be sent
symbol din                = b13
symbol address   = b10
symbol drequest  = b12
symbol temp_byte = b0

EEPROM $00,("?9?5312C?A864?'?")    ' Function keys
EEPROM $10,("?????Q1???ZSAW2?")    ' Main keyboard keys
EEPROM $20,("?CXDE43?? VFTR5?")
EEPROM $30,("?NBHGY6???MJU78?")
EEPROM $40,("?,KIO09??./L;P-?")
EEPROM $50,("??'?[=?????]????")
EEPROM $60,("?????????1?47???")    ' Numeric keypad keys
EEPROM $70,("0.2568??B+3-*9??")
EEPROM $80,("Downloading   Memory...")


'***************************
'** START OF MAIN PROGRAM **
'***************************


init:
        gosub initlcd
        'let address = 100
        'let drequest = 4 '0=data0 1=pressure       2=flow   3=data3
        pause 500                  'small delay to avoid start up error

main:
        keyin
        read keyvalue,drequest
        let drequest = drequest - 48


dreq:                      'data request - get data from main picaxe
        high 1
        serin 1,N2400,("|")
        low 1
        pause 80
        serout 1,N2400,("|",drequest)

readin:
        serin 1,N2400,("|"),din
```

```
         serout 1,N2400,("|")

         if drequest <> 4 then goto continue

         for b4 = 128 to 139
                 read b4, lcdd
                 gosub wrchr
         next b4
         let lcdd = 192
         gosub wrins
         for b4 = 140 to 151
                 read b4, lcdd
                 gosub wrchr
         next b4
         goto main


continue:
         let lcdd=1
         gosub wrins

         gosub dispnum
         goto xloop




xloop:
         let lcdd=13
         gosub wrins

         goto main








'*****************
' SUB-PROCEDURES
'*****************
` provided by rev. ed. In PICAXE manual 3

initlcd:
         let pins = 0             'Clear all output lines
         let b3 = 0                        'Reset variable b3
         pause 80                          'Wait 80 ms for LCD to reset.
         let pins = %00110000     'Set to 8-bit operation.
         pulsout 3,1              'Send data by pulsing 'enable'
         pause 10                          'Wait 10 ms
         pulsout 3,1              'Send data again
```

```
        pulsout 3,1               'Send data again
        let pins = 32             'Set to 4-bit operation.
        pulsout 3,1               'Send data.
        pulsout 3,1               'Send data again.
        pause 10
        let pins = 128            'Set to two line operation
        pulsout 3,1               'Send data.
        let lcdd = 14             'Screen on, cursor on instruction
        gosub wrins               'Write instruction to LCD
        let lcdd=1                        'clear screen
        gosub wrins
        let lcdd=13                       'flash cursor
        gosub wrins
        return



wrchr:
        let pins = lcdd & 240     'Mask the high nibble of b1 into b2.
        high 2                    'Make sure RS is high
        pulsout 3,1               'Pulse the enable pin to send data.
        let b2 = lcdd * 16        'Put low nibble of b1 into b2.
        let pins = b2 & 240       'Mask the high nibble of b2
        high 2                    'Make sure RS is high
        pulsout 3,1               'Pulse enable pin to send data.
        return



wrins:
        let pins = lcdd & 240     'Mask the high nibble of b1 into b2.
        low 2
        pulsout 3,1               'Pulse the enable pin to send data.
        let b2 = lcdd * 16                'Put low nibble of b1 into b2.
        let pins = b2 & 240       'Mask the high nibble of b2
        low 2
        pulsout 3,1               'Pulse enable pin to send data.
        return



dispnum:
        lcdd = din / 100 + 48 ' Hundreds
        if lcdd < 49 then goto dtens
        GOSUB wrchr
dtens:
        let temp_byte = lcdd
        lcdd = din / 10 // 10 + 48 ' Tens
        if temp_byte < 49 AND lcdd < 49 then goto dunits
        GOSUB wrchr
dunits:
        lcdd = din // 10 + 48 ' Units
        GOSUB wrchr
        return


`EOF
```

Appendix G PC Software Readme.txt File

```
Firstly, all credit for the program 'serialterm.exe' is given to A. Schmidt
and a copy of his readme is supplied at the end of this file.


*************
INSTRUCTIONS:
*************

To dump the information from your datalogger to a *.csv file (comma separated
variable file) first connect your download cable to com port 1 of your computer.
Then double-click “runme.bat”. This will run “serialterm.exe” with the correct
baudrate, parity, com port, etc. Running “runme.bat” will bring up a dialog box
on your computer. When this happens, press the “download” button on your datalogger.
Depending on the amount of memory that you have in your datalogger, you may
have to wait over a minute for this to complete. When the download has completed
(the numbers will have stopped appearing on the screen), press “escape” on your
keyboard to end the transfer program. Your data has now been downloaded to a file
in the same directory as this readme and will be called “datalogger.csv”. From
here, the *.csv file can be opened directly into Microsoft Excel.

Please note that the download program will overwrite any existing file named
"datalogger.csv" that is contained in this directory.




********************************************************************************
Serialterm.exe Readme follows:

    Commandline Serial Terminal - Version 1.1 by A. Schmidt, Oct 2001
    Lancaster University - http://www.comp.lancs.ac.uk/~albrecht/


    serialterm port [speed] [DisplayMode] [Separator] [Echo][logfilename]
       port ::= com1 | com2 | com3 | com4 | com5 | com6
       speed::= 300 | 4800 | 9600 | 19200 | 38400 | 57600 | 115200 | 230400
       DisplayMode::= ascii | hex | decimal
       Separator::= empty | space | newline | tab
       Echo::= no | yes
       logfilename::= <anyname> (if not provided no log is written)

    Usage examples:
       serialterm com1 115200 hex space no logfile.txt
       open the terminal on port com1 with 115200 bit/s, print hex code of
       incoming characters, seperate them by space, no local echo, save
       output to the file 'logfile.txt'

       serialterm com2 19200 decimal tab yes
       open the terminal on port com2 with 19200 bit/s, print decimal code
       of incoming characters, seperate them by tabs, do local echo, no logfile
```