University of Southern Queensland
Faculty of Engineering and Surveying

Ambient Air Temperature Aeration Controller

A dissertation submitted by

Andrew Charles

in fulfilment of the requirements of

Courses Eng4111 and 4112 Research Project

towards the degree of

Bachelor of Engineering (Electrical and Electronic)

Submitted: November, 2006

Abstract

Aeration is an important component in the successful bulk storage of grain. Without it, grain can degrade in quality, destroying profits. To achieve the best results from aeration, an automatic aeration controller should be used. This device monitors the condition of ambient air and automatically activates aeration fans, during the coolest period of the day, to cool the stored grain. The system has been designed by researching existing control methods, in quest of improvements and alternatives, and developed into a working prototype.

A PICAXE microcontroller is used to process data and determine the optimal period to operate the aerators. A combined Relative Humidity and Temperature sensor is utilised to measure the state of the ambient air. The Relative Humidity is combined with a lookup table to determine an approximate wet-bulb temperature. By utilising wet-bulb temperature, a greater cooling effect is achieved through an evaporative cooling effect. A LCD display provides a user interface exhibiting useful data in relation to the device. The CSIRO Time Proportioning Control Method was implemented, providing Normal and Rapid outputs. An alternative method of calculating set-points, involving cumulative probability, was applied.

The final prototype constructed was tested successfully, logging data and determining set-points for operation. Improvements and further work aimed at improving the design are discussed.

University of Southern Queensland

Faculty of Engineering and Surveying

ENG4111 Research Project Part 1 &
ENG4112 Research Project Part 2

Limitations of Use

The Council of the University of Southern Queensland, its Faculty of Engineering and Surveying, and the staff of the University of Southern Queensland, do not accept any responsibility for the truth, accuracy or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk, and not at the risk of the Council of the University of Southern Queensland, its Faculty of Engineering and Surveying or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of the course pair entitled "Research Project" is to contribute to the overall education within the student's chosen degree program. This document, the associated hardware, software, drawings, and other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.

Professor R Smith
Dean
Faculty of Engineering and Surveying

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.

Andrew Charles

Student Number: 0050009343

_____
Signature

_____
Date

Acknowledgements

I would like to thank my supervisor, Mr Mark Norman, for his help and guidance throughout this project.

I would also like to thank my friends and family for their continuing support throughout my degree.

Table of Contents

List of Figures

List of Tables

Nomenclature

PIC	Programmable Integrated Circuit
EEPROM	Electrically Erasable Programmable Read Only Memory
LCD	Liquid Crystal Display
RH	Relative Humidity
CRC	Cyclic Redundancy Check
RTC	Real Time Clock
SPDT	Single Throw Double Throw
PCB	Printed Circuit Board
CSV	Comma Separated Variable
BCD	Binary Coded Decimal

Chapter 1 – Introduction

1.1 Background Information

Grain in agriculture is generally stored for periods of time after harvest. This is usually done as a stack of grain in the open air, or in a silo that contains the grain. A typical silo is shown in figure 1.1. Silos in agricultural applications generally range in size from a few tonnes to several hundreds of tonnes. Using a silo presents advantages over an open heap. The grain in a silo is not directly exposed to atmospheric conditions such as rain, this avoids wetting which can result in possible shooting of the seed. The likelihood of rodent infestation is also reduced significantly. Silos are available in either sealed or unsealed varieties, unsealed are not air tight while sealed silos can be made air tight for fumigation purposes. By storing the grain in a sealed container, it is exposed to complications that have the potential to degrade the quality of the grain.

Figure 1.1: A Sealed Silo

Aeration is used successfully in bulk grain storage to prevent quality degradation and can potentially offer improvements in quality. The process involves forcing

air through the grain bulk to cool and remove moisture from the grain. To achieve these benefits the air used in the aeration process must be carefully selected. An aeration controller monitors ambient air parameters and switches fans on at optimal times to condition stored grain according to the prevailing air conditions. The controller must switch the fans to provide a cooling effect to the grain, hence reducing insect/mould development, grain spoilage and moisture migration. Such an automated system has significant advantages over other control methods such as a timer or a manual switched system. These can be extremely inaccurate methods, operating at inappropriate times with the potential of undoing any gains previously made.

For highest efficiency the coolest possible air available must be used. As the day to day temperature cycle varies, a method of prediction must be utilized to forecast the minimum temperature for the current daily cycle. This data is used in order to calculate set-points for the aeration motor control. Most existing aerators are driven by mains supplied electric induction motors which offer no form of speed control. The controller operates as a simple on/off switch to the motors providing a variable, long pulse width run signal. This is demonstrated in figure 1.1 below.

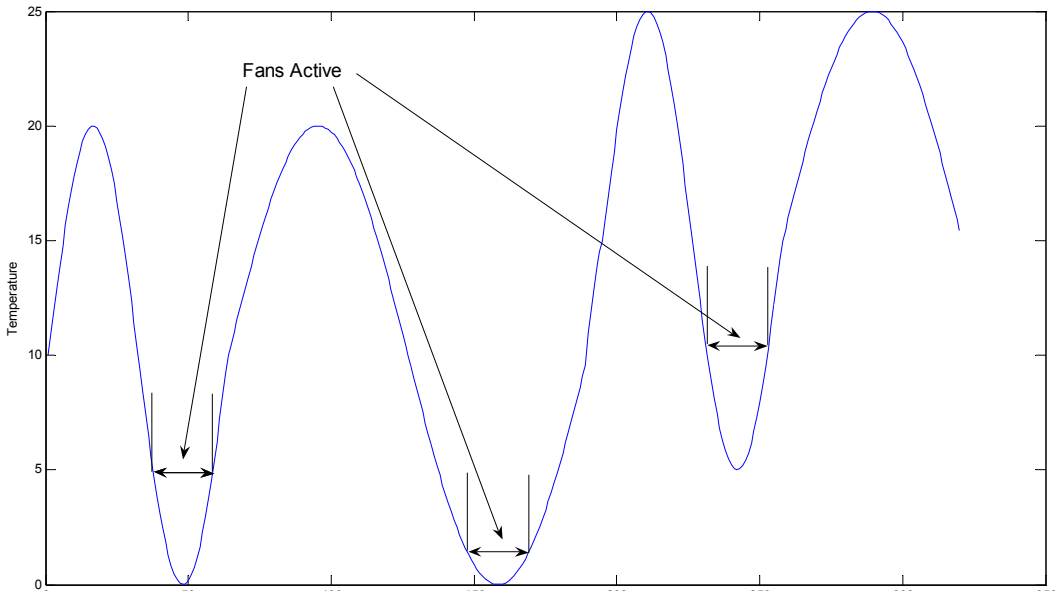


Figure 1.2: Fan Switching Periods

1.2 Project Aim

The aim of this project is to develop an aeration controller that will automatically switch aeration fans on and off depending on ambient air temperature. This will ensure the grain in storage is conditioned during the optimal time of coolest air.

1.3 Project Objectives

The following objectives were chosen to facilitate the completion of the project.

1. Research information on aeration control methods in grain storage.
2. Research and select sensors appropriate for ambient air measurement.
3. Research and select microprocessor with other hardware components.
4. Design and simulate software required for control of aeration fans.
5. Construct prototype and evaluate.

As time permits

6. Investigate remote monitoring of controller operation.

1.4 Dissertation Structure

The dissertation is organized as follows:

Chapter two covers the findings of the literature review carried out on the aeration process. The general process, categories, benefits and control methods will be discussed.

Chapter three is a risk assessment of the project. It outlines the possible risks in development and use of the controller and the measures required to minimise these risks. The development of a budget and timeline is also discussed.

Chapter four contains the detailed design information. Included is comprehensive information of components selected for the system, an explanation of how they are interfaced, the system schematic and the PCB design.

Chapter five presents the methods used for building the prototypes. Firstly the initial design that was built on a breadboard will be discussed. Secondly the final prototype, housed within a protective case will be presented.

Chapter six describes the software development for the project. Firstly, the PICAXE programming editor is introduced. The explanation of the software modules and their operation then follows.

Chapter seven presents the methods and analyses the results of the tests conducted on the design. Simulation is carried out to test the result of changing the sampling buffer size.

Chapter eight discusses the performance of the design compared to the aims, objectives and specifications. Recommendations for future work that would improve the system are made.

Chapter 2 – Aeration Process

This section will review current literature to establish the problems that need to be overcome and the processes that are best for achieving this. The following points will be considered:

1. The problems involved with the bulk storage of grain
2. Aeration benefits
3. Aeration categories and flow rates necessary
4. Aeration control methods currently used or in development

2.1 Storage Problems

There are a multiple reasons grain is stored for long periods of time. In agricultural situations the most common is usually an attempt to maximize profits. Other reasons include extending the harvest window (Newman 2002), allowing earlier harvest and also in managing income tax by delaying profits until the next financial year. These reasons justify grain storage but there are also problems inherent in the process.

Grain deteriorates while in storage, this may result in potential profit losses. The major causes of deterioration in bulk stored grain are insect activity, mould growth and moisture migration (McPhee 1998). Temperature difference gradients in the grain bulk cause moisture migration by convection currents. This moisture condenses on silo surfaces, causing spoilage of the grain in that area (McPhee 1998). Mould can develop if the moisture within the grain is too high. Insect development within the grain will naturally occur, however it is accelerated at higher temperatures. This can be a serious concern in some circumstances. Insect activity can cause hot spots within the grain; these areas promote extra insect growth and contribute further to moisture migration (Newman 2002).

2.2 What is Aeration

Fusae (2004) defines aeration as the process of passing cool air through grain to reduce its temperature to a level where insect development, mould growth and moisture migration are dramatically inhibited. The condition of this air needs to be chosen carefully to achieve any benefit. Aeration is accomplished with fans and ducting used to force the air through the grain. The fans are generally electrically driven as it is the most cost effective.

2.3 Aeration Benefits

Aeration provides many benefits and is very important to successful grain storage. Fusae (2004) lists the benefits as the following:

1. *Insect and mould activity is dramatically suppressed, reducing spoilage and weight loss of grain.*
2. *Temperature and moisture variations within the grain are prevented, thus avoiding hot spots and condensation due to moisture movement.*
3. *Cooling helps maintain grain quality – especially important for grain kept for seed.*
4. *Insecticides maintain their effectiveness for far longer when grain is cooled by aeration.*
5. *Aeration is a low energy process – running costs are low. Aeration can dramatically improve the storage life of dry grain. This practice has met with a wide acceptance in the past few years with farmers installing many of these units for improving the storage conditions in their silos.*

Aeration can also be an effective means of relieving pressure on a drying system by stalling the immediate need to dry over wet grain (Fusae 2004). The wet grain may be stored under aeration for a limited period while other grain is dried.

Aeration not only helps to keep the storage temperature in a safe range but will also reduce the decay rate of protectants by 3 to 5 times (McPhee 1998). This reduces the quantity of insecticide required; this reduction limits the extent of insects becoming resistant to the chemical.

2.4 Aeration Categories

Aeration can be divided into general categories based on air flow rate. Darby (1998) defines these categories as maintenance, up to 0.5 L/s/tonne, cooling, 0.5 to 2.5 L/s/tonne and drying, 2.5 to 20 L/s/tonne. The air flow rates of the aeration fan equipment used dictate the extent of ability of the aeration system. Most existing installed aeration systems are capable of air flows in the low cooling range. The air flow of the system must be sufficient for the application otherwise it will not perform effectively. This must be considered when designing the controller. The lower functions of maintenance and cooling can be performed by most systems (Darby 1998). Outputs from the controller to perform these functions can be interfaced to most existing systems. Higher functions, such as drying control periods, will be useless unless attached to a fan capable of the necessary air flow.

2.5 Aeration Control Methods

In aeration, there are a number of general methods available to switch the aeration fans on and off. Fully manual control involves the operator estimating and activating the fans at the appropriate times. This can be improved by manual charting of the temperature at various times and activating accordingly. Advancement on this method is through the use of an adjustable timer switch for control. The operator pre-sets the running time interval based on the estimate of time that will provide the most benefit. This provides the same accuracy as completely manual operation, but the operator will be relieved from manual

switching at inconvenient times. Clearly these methods are exceedingly inaccurate which is the reason a dedicated aeration controller, that monitors and selects the most beneficial air is the best option.

Commercial aeration controllers are available on the market. These devices are expensive for what is essentially a self adjusting thermostat. Many of these systems provide no way of viewing when the system ran the aerators, making verification of successful operation impossible. While monitoring the air temperature, existing systems do not display the value. An accurate present temperature display would be very useful in a grain storage environment. Many test devices used for measuring moisture of grain require the air temperature to be known. A readout of this value would eliminate the need for an additional thermometer. The control method employed by commercial aeration controllers are presented in the following sections. An alternative statistical method, logging a buffer of samples which can be used to verify system operation is explained.

2.5.1 Time Proportioning Controller

Research into controllers currently available, showed that the most common form of control was based on the CSIRO Time Proportioning Control Method. This process tracks the daily fluctuation of ambient air temperature and switches the aeration fan on during the coolest times in order to meet one of the two available aeration time fractions, 0.15 (normal mode) or 0.5 (rapid mode) (CSIRO, SGRL). A graphical example of this is shown in below in figure 2.1, the normal mode time period is displayed.

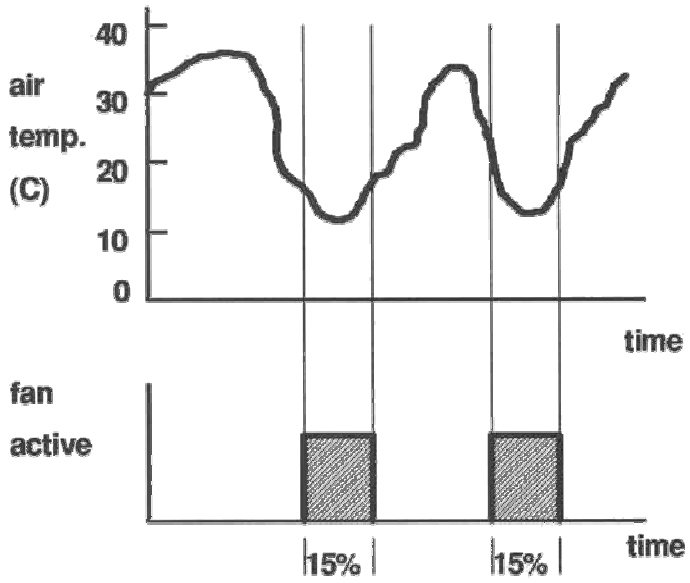


Figure 2.1: Time Proportioning Controller (Stored Grain Research Laboratory, CSIRO)

This method is simple in that it is essentially an open loop function. The temperature of the grain is ignored and the system attempts to keep the grain as cool as possible. The only input to the system is the current temperature; the output is selected from the two modes as appropriate.

The Rapid mode of 50% time duration is designed for use with hot, recently harvested grain. The longer running period provides a greater cooling effect, quickly removing the harvest heat from the grain before damage can result. Once this heat has been removed the Normal mode, active for 15% time duration is selected. This mode maintains the grain in a cool state by activating the fans long enough to remove heat absorbed from the surroundings and reach a point of stabilisation.

2.5.2 Adaptive Discounting

Research also revealed a new method of control recently became commercially available, it is called Adaptive Discounting. This method contains two components operating simultaneously. The adaptive component sequentially

propagates complete individual fronts through the store until the target average grain condition is achieved. The discounting action monitors the air selection process and 'releases' the set points to maximize the propagation rate (Darby 2000). Adaptive Discounting is a closed loop method utilising sensors within the grain to monitor its condition. This can also prove a disadvantage in that these sensors are a possible point of failure that other methods do not utilise. Each silo of grain being cooled requires its own sensor and control loop. This is an added complexity when compared to the time proportioning controller which can be simply multiplexed by connecting the outputs in parallel.

2.6 Temperature Prediction

As discussed in earlier sections, the success of aeration is based on the use of the coolest possible available air. The selection of the coolest period of air is not a simple process. The day to day weather pattern, and hence temperature is continuously changing. Due to these daily temperature fluctuations it is not known exactly what the coolest temperature will be and when it will occur. The controller is therefore required to forecast the coldest period, this prediction is generally based on previous temperature data. Two methods capable of performing this prediction are discussed below.

2.6.1 Gradient Control Method

Research available regarding existing aeration controllers using Time Proportioning Control, showed that they use a slope method for temperature prediction. A commercially available aeration controller, the Rimik AC12, is based on the CSIRO Time Proportioning Controller. It uses a self adjusting thermostat method to predict the relevant set-points. The process is shown graphically in figure 2.2.

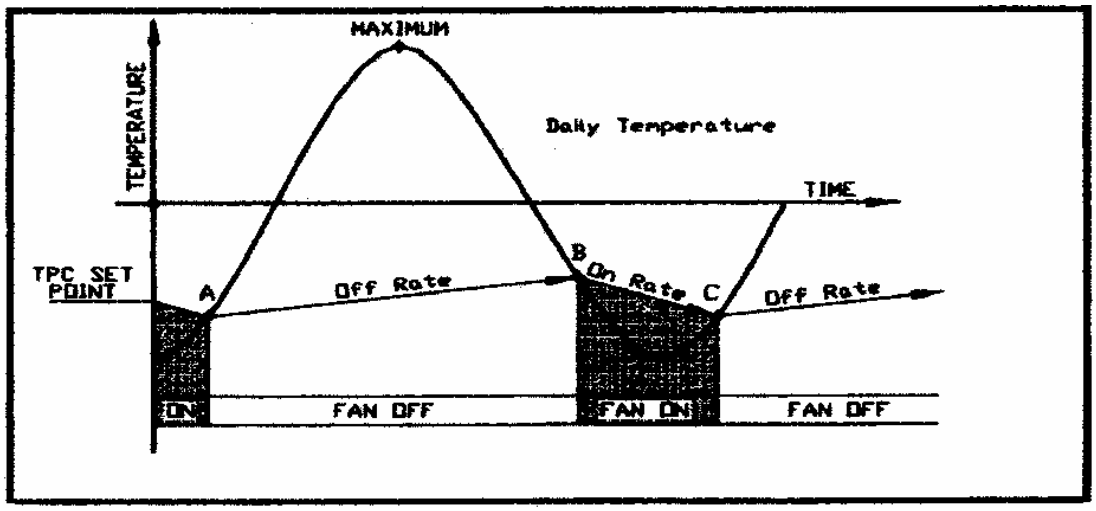


Figure 2.2: Slope Control Set-point Method (Fusae 2004)

Fusae (2004) describes the operation as follows. When first activated an initial set-point is established by the controller. In the case of figure 2.2, the initial temperature is less than the set-point. This being the case, the output is activated and the set-point is slowly ramped downwards. When the set-point intercepts the current temperature, the fans are switched off and the set-point gradient changes sign. The set-point is now ramped upwards until it intercepts the temperature. At this point the fans are again switched on. In this way the fans are only activated while the temperature is below the set-point slope, this being the coolest part of the temperature cycle. The gradient of the On Rate and Off Rate slopes determine the duty cycle of the fans. Due to the initial set-point used when the device is first switched on, it takes a number of cycles for the controller to stabilize and truly select the correct period of the coolest air.

Pros

- Simple operation and calculation within processor

Cons

- Takes time to stabilize
- Gradients must be pre calculated from typical temperature data
- On and Off set-points are skewed to different temperatures due to on rate slope, therefore not optimal selection of coolest period

To implement this method of control, temperature data must be analysed to determine the appropriate gradients. In figure 2.2, the temperature cycle is represented by a sine wave. The true temperature cycle will appear as a cyclic curve with a noise component due to the small fluctuations caused by atmospheric changes such as varying cloud cover.

2.6.2 Statistical Control Method

An alternative method proposed for predicting the coldest temperature is by statistical analysis of prior temperature data. This method involves storing a buffer of temperature samples and using these as a basis for the prediction of the current day's minimum temperature. This can be easily demonstrated by studying figure 2.3 below.

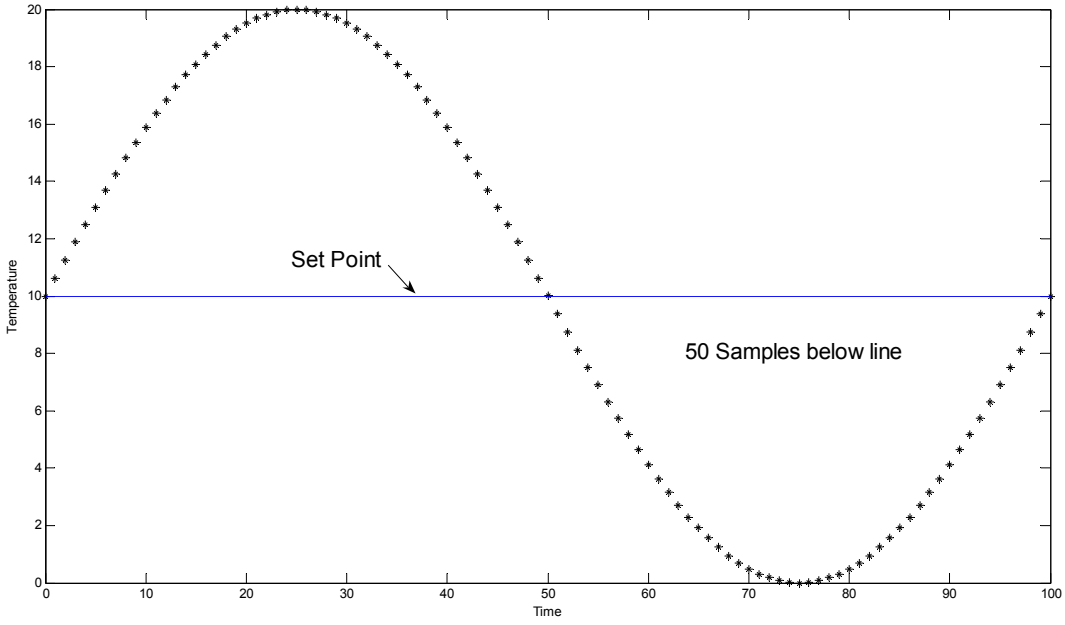


Figure 2.3: Statistical Method

In this example one hundred samples represent a days temperature cycle. To select the coolest 50% of the time, the lowest 50 samples are chosen. The buffer of samples must be a multiple of complete days to ensure the time percentage is correct. The temperature at which there are fifty samples below can be used as the predictive set-point for the current day.

In statistical terms, this is referred to as using the cumulative probability of the data. The cumulative probability of the samples is the probability of the signal being below a certain level (Leis 2002). This expressed mathematically for a continuous signal is:

$$F(X) = \Pr\{ x(t) \leq X \} \tag{2.1}$$

Where F(X) is the cumulative probability
x(t) is the continuous signal
X is the value which the cumulative probability is found to be below

For the value of X shown in figure 2.4 the cumulative probability is found by:

$$F(X) = \frac{t_1 + t_2 + t_3 + t_4}{T} \tag{2.2}$$

Where F(X) is the cumulative probability
T is the total time
t1 – t4 are the intervals for which the signal is below X

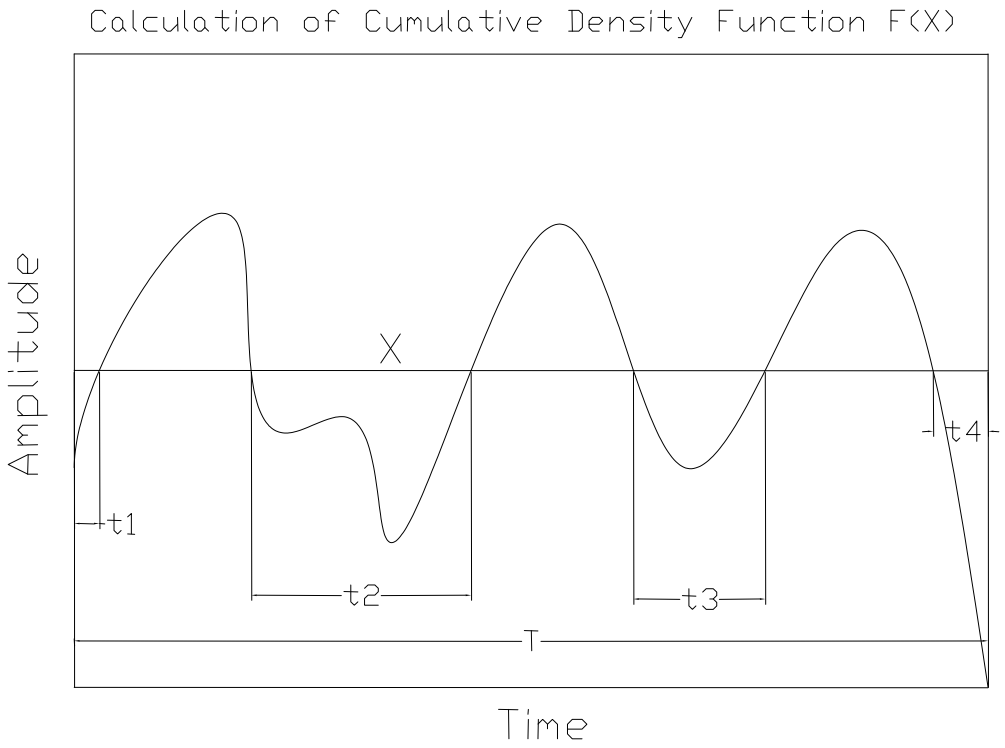


Figure 2.4: Cumulative Probability Below Value X

In an aeration control sense, to use the coolest F(X) percent of the time, the corresponding temperature value for X is found and this used as the predictive set-point. The Normal mode percentage of 15% is used as an example. The number of samples required to be below the set-point is established as 15% of the buffer. Starting at the lowest recorded temperature, the number of samples measured at this temperature is added to a total tally. The tally is incremented with the number of times the temperature occurred as the temperature is stepped through in ascending order. Once the required number of samples is reached the process is stopped. The temperature at which this occurs is the set-point.

The following figures provide an example of this process. A day's log of temperature, taken at a sampling interval of 15 minutes and resolution of one degree, is used as the statistical data. The histogram represents the number of times each temperature occurred. The cumulative probability graph is simply a cumulative sum of the histogram, scaled to a percentage. The lowest temperature which encompasses 15% of the samples is the set-point. Due to the one degree resolution in this example, there will be some error as a temperature bar containing exactly 15% of the samples is highly unlikely.

The dotted line in the figure shows the calculated set-point of 12 degrees. If the temperature drops below this value the Normal fan output will be activated. The Rapid set-point is calculated in the same way using 50% of the samples.

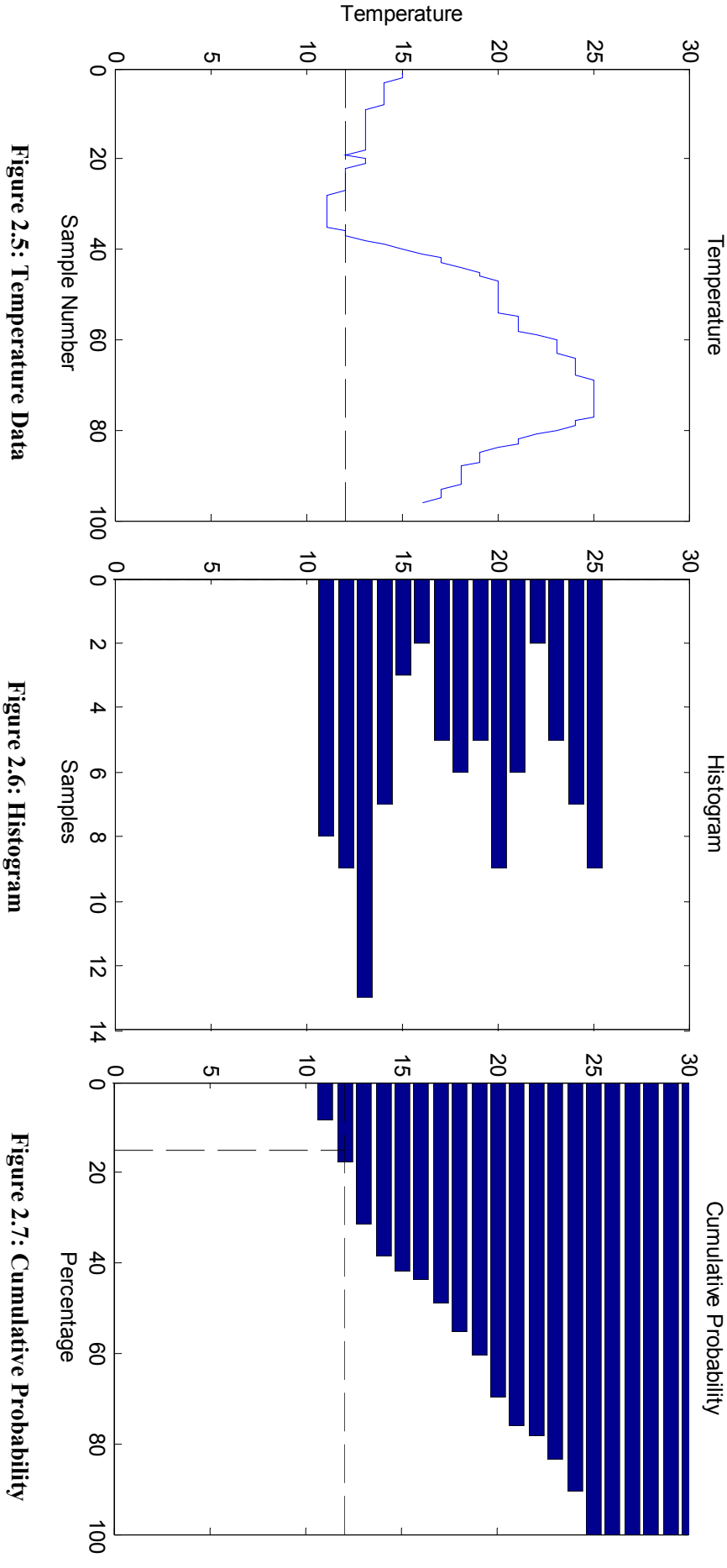


Figure 2.5: Temperature Data

Figure 2.6: Histogram

Figure 2.7: Cumulative Probability

When using this method, a number of previous day's data may be used to smooth out daily fluctuations by effectively taking an average.

Pros

- Self calculating to climate
- Optimal symmetrical on and off switching based on data

Cons

- Takes time to collect samples initially
- Storage required for data
- More complex than Gradient Method

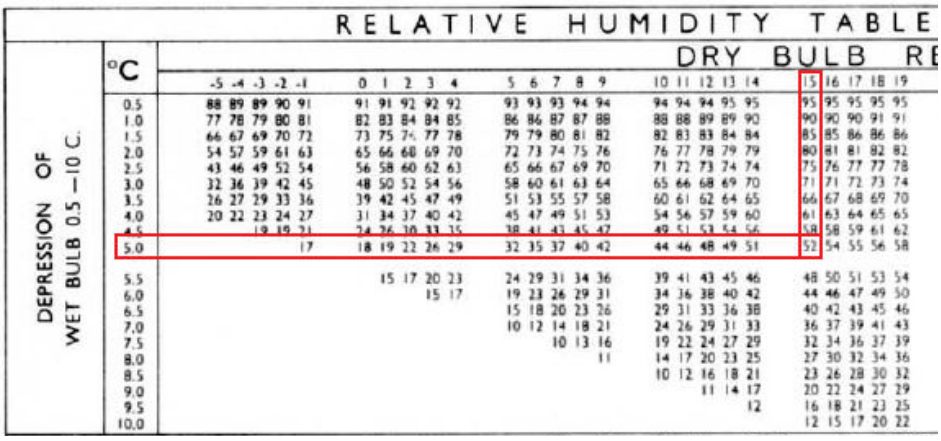
2.7 Temperature Sensing

The basis of aeration is air of correct condition being passed through the grain. The basis of aeration control is the selection of appropriate air. Ambient air contains an amount of water vapour. This amount can be measured in terms of Relative Humidity. A method of calculating Relative Humidity (RH) is by means of a hygrometer, also known as a Wet and Dry Bulb Thermometer.

Figure 2.8: Hygrometer (source: http://www.bom.gov.au/info/weatherkit/section2/hygro.shtml)

This method involves comparing the temperature indicated by a normal thermometer to that given by a thermometer which has its bulb wrapped in a wet moisture absorbent material. The rate of evaporation from the wet-bulb thermometer depends on the humidity of the air - evaporation is slower when the air is already full of water vapour. For this reason, the difference in the temperatures indicated by the two thermometers gives a measure of atmospheric humidity (BOM, 2006). The RH is calculated from a table comprising the dry bulb temperature and the difference between dry and wet bulb temperatures.

Wet bulb temperature itself is an obscure measurement, usually only used to determine RH. It is however a means of factoring the amount of evaporative cooling potential available by the air. Aeration controllers using wet-bulb temperature rather than dry-bulb temperature achieve greater cooling due to an evaporative cooling effect. Air that is dryer than the grain will absorb some of the moisture as it is blown through the grain bulk. This is the same principal used in evaporative type air conditioners; the efficiency of these is reduced during times of high humidity. Despite aeration controllers that use wet bulb thermometers being more expensive, the increased efficiency is a desirable characteristic. One solution is that rather than using an actual wet and dry bulb thermometer, the wet-bulb temperature can be approximated using the dry-bulb temperature and the relative humidity. These two characteristics can be measured relatively cheaply and the same table used to calculate RH can be used in reverse to approximate the Wet Bulb temperature of the air. An excerpt from a typical table is shown in figure 2.9.



RELATIVE HUMIDITY TABLE

DRY BULB RE

DEPRESSION OF WET BULB 0.5 — 10 C.

°C	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0.5	88	89	89	90	91	91	91	92	92	92	93	93	93	94	94	94	94	94	95	95	95	95	95	95	95
1.0	77	78	79	80	81	82	83	84	84	85	86	86	87	87	88	88	88	89	89	90	90	90	90	91	91
1.5	66	67	69	70	72	73	75	76	77	78	79	79	80	81	82	82	83	83	84	84	85	85	86	86	86
2.0	54	57	59	61	63	65	66	68	69	70	72	73	74	75	76	76	77	78	79	79	80	81	81	82	82
2.5	43	46	49	52	54	56	58	60	62	63	65	66	67	69	70	71	72	73	74	74	75	76	77	77	78
3.0	32	36	39	42	45	48	50	52	54	56	58	60	61	63	64	65	66	68	69	70	71	71	72	73	74
3.5	26	27	29	33	36	39	42	45	47	49	51	53	55	57	58	60	61	62	64	65	66	67	68	69	70
4.0	20	22	23	24	27	31	34	37	40	42	45	47	49	51	53	54	56	57	59	60	61	63	64	65	65
4.5				19	19	24	26	30	33	35	38	41	43	45	47	49	51	53	54	56	58	58	59	61	62
5.0					17	18	19	22	26	29	32	35	37	40	42	44	46	48	49	51	52	54	55	56	58
5.5						15	17	20	23		24	29	31	34	36	39	41	43	45	46	48	50	51	53	54
6.0								15	17		19	23	26	29	31	34	36	38	40	42	44	46	47	49	50
6.5											15	18	20	23	26	29	31	33	36	38	40	42	43	45	46
7.0											10	12	14	18	21	24	26	29	31	33	36	37	39	41	43
7.5													10	13	16	19	22	24	27	29	32	34	36	37	39
8.0															11	14	17	20	23	25	27	30	32	34	36
8.5																10	12	16	18	21	23	26	28	30	32
9.0																		11	14	17	20	22	24	27	29
9.5																				12	16	18	21	23	25
10.0																					12	15	17	20	22

Figure 2.9: Relative Humidity Table Excerpt

Source: (BOM 2001)

A complete copy of this table is presented in Appendix B. As an example calculation take the red lines shown on the table. A dry bulb temperature of 15°C is measured along with a Relative Humidity of 52%. By finding these values within the table a wet bulb depression of 5°C is found. The depression is subtracted from the dry bulb temperature, 15°C - 5°C = 10°C, and thus the wet bulb temperature is estimated.

The accuracy of this method of determining wet-bulb temperature varies. At lower temperatures the humidity range is much smaller. Thus the depression value will vary less with a change in humidity. To obtain a very accurate value a larger table would be required. This would require larger memory storage within the controller to accommodate a larger lookup table. The table presented in the above figure is deemed accurate for this prototype. The depression resolution is half a degree and the dry-bulb resolution is in whole degree intervals.

Monitoring the humidity of the air also provides a further advantage. The device is aimed solely at using cool air. Early dewy mornings, and when the temperature drops just before rain storms, are times when the controller may operate the fans (Fusae 2005). The cool temperature associated with these conditions will be less than the set-points, activating the fans. These are cool periods where the RH of the air will be high. This high humidity will transfer moisture to the grain along with the cooling. Usually there is a great gain in cooling during these times and

the amount of wetting occurring is small (Fusae 2005). If however drying is preferred to cooling, the humidity may be used as a form of high moisture cut out. The system could be prevented from running during times of high moisture. Such a feature would be unavailable in a system monitoring dry-bulb temperature only.

Chapter 3 – Risk Analysis and Project Management

3.1 Assessment of Consequential Effects

The controller intended to be produced will be responsible for the control of aeration fans maintaining a stockpile of grain. Failure of the device could potentially result in loss of product income. Incorrect operating times could possibly result in grain swelling; this could potentially cause damage to the silo structure as the grain volume increases to more than the silo can hold. Adequate testing of the design is necessary to ensure that the controller is capable of the task. Use of this device should be relatively simple as incorrect operation could produce catastrophic results.

The small scale of this project, containing few parts and limited production does not present any significant hazard to the environment during production, operation or disposal. Devices of this kind already exist and the project poses no major ethical questions. The device produced should be designed to be safe to use, install and dispose of. This product can be considered safe for all foreseeable actions involving its use.

3.2 Safety Issues

To evaluate and reduce safety hazards during and after this project the following risk assessment was developed.

To evaluate the level of risk posed, the likelihood of an event happening and the potential consequences are assessed. Definitions for the probability scale used for assessing risk are:

Extremely Slight = Practically impossible

Very Slight = Conceivable but very unlikely

Slight = Possible but unlikely

Significant = Possible

Substantial = To be expected

3.3 Risk Management Chart

To limit the risks associated with this project, the possible risks are firstly identified. These risks perceived include those present during the construction and use of the device. Measures for controlling and limiting these risks are also included. These risks have been checked and reviewed during the project interval.

Table 3.1 Risk Assessment

Description of Hazard	**People at Risk**	**Risk Severity**	**Probability**
Equipment being dropped on feet	Myself	Bruising	Slight
Control Measures	Always wear fully enclosed footwear Take extra care		
Injury during construction of prototype	Myself	Cuts or scratches to hands and fingers	Slight
Control Measures	Use correct tools for the task Use tools in the correct manner Handle components carefully		
Injury from heat from soldering/ electrical components	Myself	Burns	Slight
Control Measures	Allow components to cool Avoid touching components when on		
Soldering fumes	Myself	Toxic inhalation Eye irritation	Very slight
Control Measures	Keep well ventilated Stop if irritation occurs		
Electrocution	Myself Future users	Small shock to death	Significant to me Future uses slight
Control Measures	Carry out as much testing as possible without power Test using low voltage power supply to avoid mains voltages Exercise caution Installation of live wiring by a licensed electrician		

3.4 Resource Planning and Timeline

To facilitate the completion of the project, a list of expected resources and associated costs were compiled. From this a budget of approximately $250 was calculated. The table of resources is presented in Appendix J. To set deadlines for work to be completed, a timeline was constructed consisting of work to be completed, and time allotted. This timeline was updated through the course of the project as required, as a better understanding of activities and their timeframes became available. The timeline is presented for reference in Appendix K.

Chapter 4 – System Design

4.1 Description of Intended Use

The Aeration Controller is intended to be permanently mounted in an undercover position and not exposed to environmental factors such as rain. The temperature sensor will be remote and attached to the main unit by a cable, allowing it to be mounted in an open location that will reflect the true atmospheric conditions. As this system is being designed for control of electrically driven motors a 240v mains supply should be available and the power supply for the system will be derived from this source.

The outputs from the controller will be separate Normal and Rapid signals for control of contactors. This will allow selection of required cooling rate for each silo external to the aeration controller and minimize the number of outputs required by the microcontroller.

Due to the slow rate of change that occurs with ambient temperature, high clock speed of the microprocessor was not considered an important characteristic. A display will provide a visual indication of current temperature along with other useful features such as time and set-point levels.

4.2 Control Method

In order to determine the required hardware such as memory type and amount, it was first necessary to select the control method to be used. The statistical control method described in chapter 1.6.2 was chosen for this design. The advantage of self tuning will allow the controller to adapt to any climate by simply collecting samples. These samples however will need to be stored in memory, contributing

to the complexity of the hardware. A potential benefit of the stored samples is the possibility for them to be read back to a PC allowing the aeration controller to also perform the function of a data logger. The symmetrical set-points for both on and off switching were also deemed beneficial compared to the gradient method, which by using a slope and intercept would switch on and off earlier than ideal, thus using warmer air than necessary and then wasting cooler air time respectively. For these reasons the statistical method was chosen over the gradient method.

An initial period of measurements was decided to be three days. This period would allow a quick response to climatic changes while also preserving averaging information. Ambient temperature changes slowly, sampling too frequently would only serve to increase the amount of storage needed without increasing accuracy. An initial estimate of fifteen minutes was chosen as the sampling interval. Over a three day period this amounts to a total of 288 samples that require storing in the buffer.

The two operational modes, Normal and Rapid, require operational periods of 14% and 50% respectively. To achieve this, the cumulative density is found from the buffer of samples, the temperature that is higher than the lowest 14% or 41 samples is used as the Normal set-point. Similarly the temperature higher than 50% or 144 samples is used as the Rapid set-point. At each fifteen minute interval, a new temperature reading is measured and stored, the new set-points will be calculated and the current temperature compared to these set-points. The outputs will be activated as appropriate.

The figure below demonstrates the set-points and the output states which correspond.

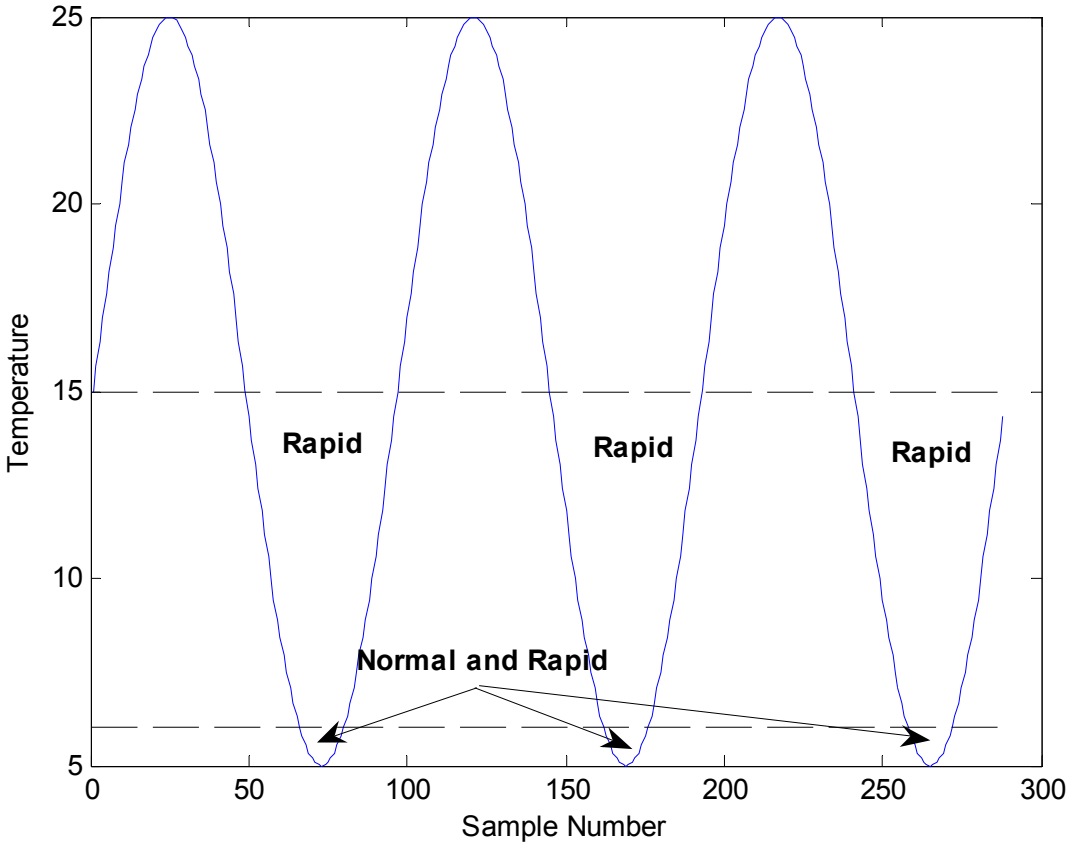


Figure 4.1: Rapid and Normal Output Conditions

To perform the calculation, the histogram will firstly be found. From the histogram the cumulative distribution will be found by addition of the histogram in ascending order. To encompass all likely temperatures that may occur, the range of wet-bulb temperatures stored is from -20 to 49°C. Any values that may occur outside this range will be clipped. The resolution of measurement is in whole degrees. This requires temporary storage of 70 individual bytes, each containing the number of times the corresponding temperature has been recorded in memory. An 8 bit variable can store a maximum value of 255. The total number of samples within memory is 288. If a temperature value were to be recorded more than 255 times, then an overflow condition would occur within the histogram. This is an unrealistic situation as the same temperature occurring 88.5% of the time over the course of three days would never occur. The only possible cause for this would be a failure of the sensor; the same temperature could possibly then be read from the sensor continually. The error handling

routines within the temperature measurement protocols will prevent this from occurring.

To utilise the improved cooling effect of wet-bulb temperature the lookup table must be stored in memory. The table is split into 20 rows and 56 columns. This is equivalent to 1120 bytes. This capacity must be added to the measurement buffer when selecting memory requirements.

4.3 Component Selection

This section will review and select appropriate components to perform essential functions within the aeration controller system. The main task involved is the development of the processor that will control all I/O. The important functions include sensing temperature, determining optimum period, controlling the display, checking input switches and switching two sets of contacts to control the fan motors. These functions are not speed critical and will not require a high speed processor.

Selection criteria were determined based on the above requirements.

- Affordability
- Ease of programming
- Moderate memory capacity
- Low power consumption
- Ease of interface with peripherals

Based on the above criteria a Programmable Integrated Circuit or PIC microcontroller was considered the most suitable option for the tasks. The PICAXE family of chips was chosen due to the low cost, freely available programming software and basic programming language to facilitate ease of software development.

4.3.1 PICAXE Microcontroller

A PIC is an integrated circuit that comprises memory, I/O ports, processor and its program in a single chip. This has the major advantage of less hardware complexity and cost compared to a microprocessor. The PICAXE microcontroller is a PICmicro pre-loaded with a bootstrap program, allowing the chip to be programmed in circuit from a PC serial port. The bootstrap program downloads and stores the new program to memory without the need to completely erase and reprogram the chip. The PICAXE chips utilize flash memory, which depending on the particular model, allows reprogramming from one thousand to one hundred thousand times. This is extremely valuable in a development environment where a single write chip would have to be disposed of after each write. The program assembler using the BASIC language is freely available for download from the PICAXE website, www.picaxe.co.uk. A particularly useful ability is the debug command which displays all variables on the programming PC, speeding up debugging.

The PICAXE system is available in a range of I/O pin sizes and program memory lengths. Special functions are available in the higher end families. The X range of parts have the maximum program memory at 2K Bytes, this is equal to approximately 600 lines of basic code. The smallest pin-out X part, the 18X, having 18 pins was selected. The base chip used and loaded with the bootstrap program is the PICmicro 16F88-I/P. All hardware specifications are obtained from its datasheet.

A major reason for the choice of this component is its ability to communicate with devices using the I2C (Inter Integrated Circuit) protocol. This interface allows a large number of peripherals to be connected in parallel to a single bus, driven by two microcontroller pins. A large range of devices are readily available that utilise the I2C protocol. The I2C protocol is capable of two clock speeds, 100 kHz and 400 kHz. Some devices can operate at the higher speed while others are

limited to the lower. The 18X utilises an internal clock oscillator. This is less accurate than an external ceramic oscillator commonly used with the larger PICAXE parts. As a result the 18X is limited to the lower 100 kHz clock speed; this is to reduce the likelihood of errors that could result from the less accurate clock frequency.

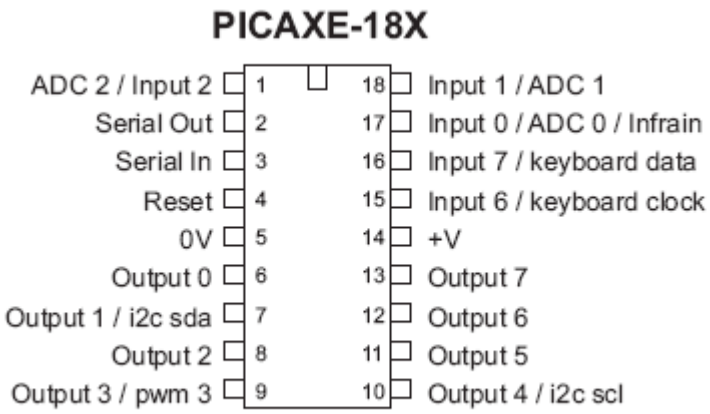


Figure 4.2: PICAXE 18X Pin-out

Source: www.picaxe.co.uk

Table 4.1: PICAXE 18X Features

Pins	Program Memory (lines)	I/O Pins	Outputs	Inputs	ADC	Data Memory
18	600	14	9	5	3	256

This chip has sufficient I/O to fulfil the requirements of the project, with most devices utilising the I2C bus. The 18X was selected in preference to the 28X or 40X to reduce the unnecessary cost of extra I/O pins.

While extremely versatile, the PICAXE family of chips is also limited in its capabilities. Variables can not contain negative numbers, also fractions or floating point numbers are also unavailable. Numbers that exceed the variable size will overflow without any warning. These limitations may be overcome by careful programming. Ensuring that a variable can never overflow by checking its

maximum possible size is one solution. Floating point numbers can be avoided to a limited accuracy by multiplying the number by ten for example to preserve one tenth accuracy. Negative numbers can also be avoided by adding an offset so that the lowest negative number does not go below zero.

4.3.2 Memory

The method of control requires a buffer of temperature samples to be stored in memory. This could be volatile or non-volatile. Using non-volatile memory has the advantage that the samples will be retained in the event of a power failure. If the sample data is lost the system will be unable to calculate the appropriate set-points until new data can be recorded. This would result in lost running hours, an unwanted occurrence. For this reason a non-volatile memory has been chosen for the sample buffer. The lookup table for wet-bulb temperature calculation must be non-volatile. It can be combined with the sampling buffer as both are of a non-volatile nature.

Possible non-volatile memories considered include EEPROM (Electrically Erasable Programmable Read Only Memory) and static ram with a backup battery. Both forms are able to be written to and read from at will. Static ram has the advantage of an unlimited number of write cycles, whereas EEPROM is capable of approximately one million write cycles. By using a circular buffer, the number of writes to each EEPROM location can be dramatically reduced. This reduction is sufficient to make EEPROM the most suitable choice as it will not require extra hardware in the form of a backup battery that would be necessary for memory retention if using static ram.

Serial EEPROM was selected in order to minimize data lines. An I2C interface was chosen as it can be easily connected to the PICAXE 18X. The 24LC range of parts by Microchip was deemed suitable for this application. These range in memory capacity from 128 bytes to 64K bytes. An estimate was made to the

amount of memory required, with some extra capacity included for expansion. The 24LC16B was chosen with a capacity of 2K bytes. This device is shown in the figure below.

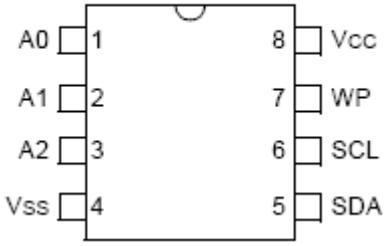


Figure 4.3: Microchip 24LC16B

Source: Microchip (2005)

The specifications state that this EEPROM can be written to a minimum of one million times. The 2K of memory is split into 8 blocks, each of 256 bytes in size. To address the device a four bit control byte must be sent, followed by a further four bits selecting the desired block. This configuration minimizes cost but limits the I2C bus to a maximum of one 24LC16; otherwise an address conflict will occur.

4.3.3 Temperature Sensor

Temperature sensors are available in a wide range of types, accuracies and interfaces. Electrical temperature measurement is usually performed by a temperature sensitive component which changes resistance according to temperature. This can be as simple as a basic thermistor or as complicated as a fully integrated chip. Common interfaces are a variable voltage, variable current, variable frequency or a digital output signal. Thermistors have the advantage of being inexpensive and simple; however this lack of complexity is also a disadvantage. Integrated chip type temperature sensors are available fully calibrated, greatly increasing accuracy.

The PICAXE 18X has a function for reading the temperature from a Dallas Semiconductor DS18B20 one wire temperature sensor. This sensor is capable of measurement in the range of -55°C to 125°C. Using this sensor allows simplification of software development as a single basic command can be used to take a reading. This sensor is suitable for monitoring the dry-bulb temperature of the air. This sensor was selected for initial software development, to allow the set-point calculation to be tested without other routines. A more suitable sensor however was selected for use in the final prototype.

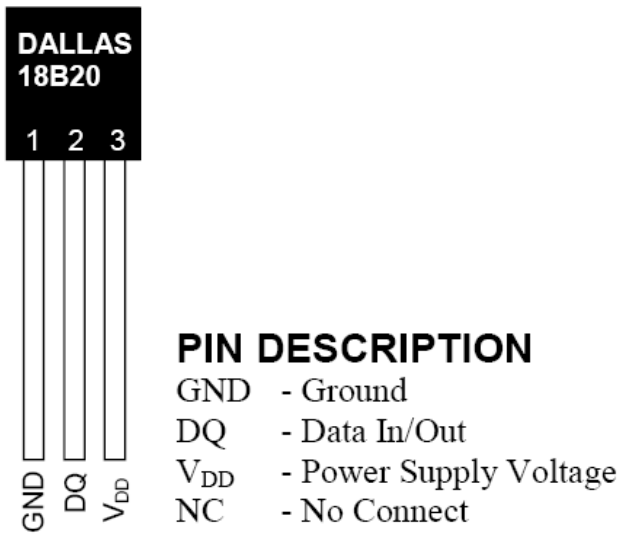


Figure 4.4: Dallas Semiconductor DS18B20

Source: Dallas Semiconductor

The temperature sensor selected for the final version of this project was the Sensirion SHT15. It is an integrated single chip temperature and humidity sensor, pre-calibrated at production. It uses a digital two wire interface, allowing for minimal I/O. It is extremely small and has a very low power requirement. By combining the temperature and humidity sensor into a single unit with a digital output, a minimum number of data lines are required. To ensure accurate communications the sensor is also capable of a CRC-8 (Cyclic Redundancy Check 8 bit). This ensures to a high degree of certainty that the received measurement has not been corrupted by noise.

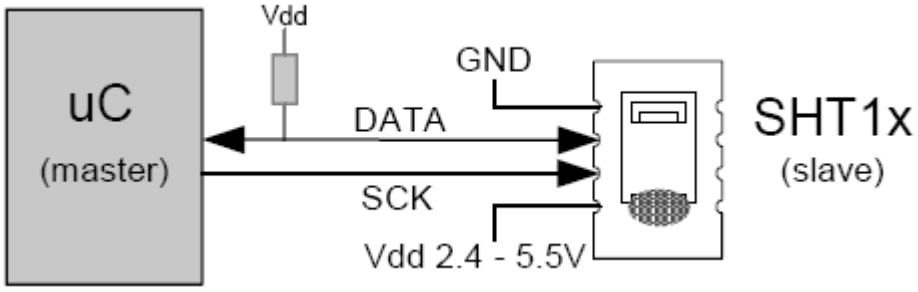


Figure 4.5: Typical SHT1x Connection

Source: Sensirion (2003)

The sensor has two configurable measurement resolutions; these are 14bit Temperature and 12 bit Relative Humidity, or 12bit Temperature and 8 bit Relative Humidity. Auxiliary features include low voltage detection and a heating element that will increases the temperature of the sensor by approximately 5°C. This heater can be used to prevent condensation in high humidity environments.

The extremely small size of this device would make hand soldering extremely difficult. A supplier was found that provided the device on its own small PCB, with a connector plug supplied.

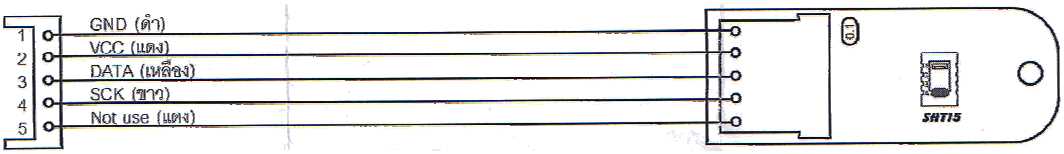


Figure 4.6: Sensirion SHT15

The serial interface used by this device is not compatible with I2C. To interface with the PICAXE custom routines were written that operate at the bit wise level. The timing diagram for this device can be found in Appendix C.

The standard configuration of the SHT15 temperature system requires a clock line and a bi-directional data line, pulled high by a pull up resistor. The PICAXE 18X does not have bi-directional data line. To overcome this problem a resistor was placed between an output and an input line. When the SHT15 is transmitting, the

output line is made high by the microcontroller. The SHT15 then pulls the line low to signify a low bit or leaves the collector open to signify a high; this is read by the input pin of the microcontroller. To transmit to the SHT15, the microcontroller switches the output line either high or low as appropriate. Unfortunately this configuration uses an extra line that would not be needed if a bi-directional line was available. The second output line is a clock signal, which is toggled high and low when the data is ready to be transmitted or received. This configuration uses a total of three pins of the microcontroller, two outputs and one input. The configuration is shown in the following figure.

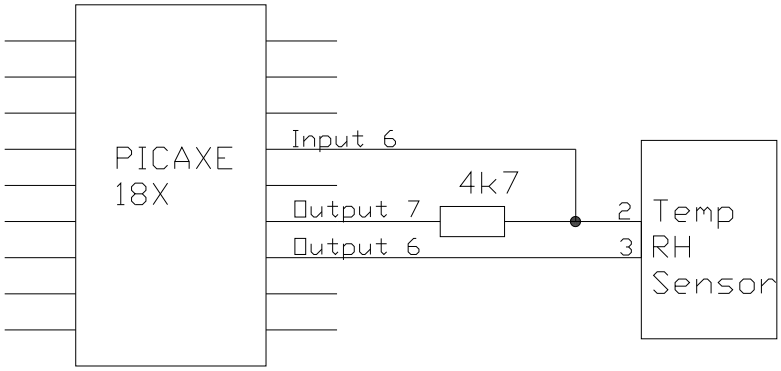


Figure 4.7: Sensor Pin Connections

4.3.4 Contact Switching

Most aeration fans are powered by electric induction motors and powered by the mains supply. This being a 240V single phase or even a 415V three phase supply. Switching of these motors is usually conducted by a set of electrical contactors. In order for the microcontroller to activate the fans and to provide electrical isolation, a set of relays was chosen to perform the switching of the larger contactors. Each output, Normal and Rapid, has its own relay for connection to the appropriate contactor. In typical installations each silo has its own switch, with the positions Off, Normal, Rapid and Manual. This allows the use of a single output for each mode being switched by the controller, while still being capable of

operating a large number of fans at the appropriate rate for the condition of the grain each silo is storing.

The relays selected for this application are produced by OMRON. They contain 12V coils with a resistance of approximately 275Ω, equating to 0.53W power consumption and 44mA of current. The switching side is a SPDT (Single Pole Double Throw) contact rated at 240VAC and 10A. These ratings are sufficient for switching industrial contactors which use minimal current at 240VAC.

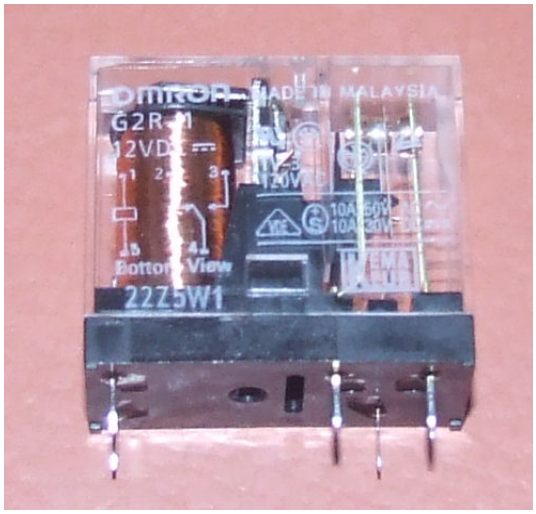


Figure 4.8: Switching Relay

4.3.5 Relay Driving Circuitry

The 16F88-I/P chip has a limited maximum output current per pin. The datasheet states the absolute maximum current either sourced or sunk by an I/O pin as 25mA. This is not sufficient to drive one of the selected relays as they require approximately 44mA each. Therefore a form of current driving circuitry is required to operate the relays. The selected device is a ULN2003A IC which contains 8 high current Darlington pairs. All outputs are capable of sinking 500mA of current. Each Darlington pair has a 2.7kΩ base resistor built in, allowing direct connection to TTL or 5V CMOS outputs.

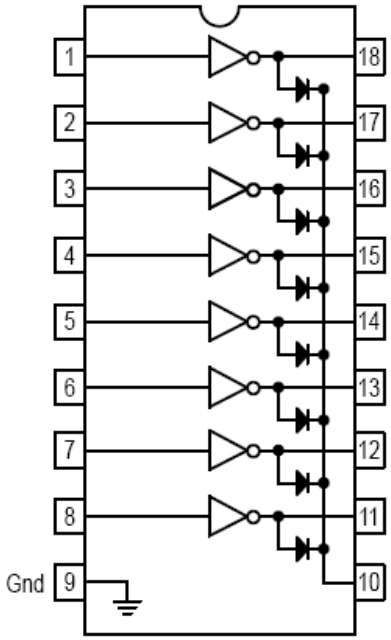


Figure 4.9: ULN2803A

Source: Texas Instruments (2005)

All Outputs are situated along one side of the IC with their respective inputs directly opposite, allowing easy routing of connections. The ULN2803A will be used to sink current powering the two relays and also two LED's that will provide a visual indication of the status of the Normal and Rapid outputs.

4.3.6 Power Supply

To ensure accuracy of the system it is important to have a continuous string of samples. A backup power supply is therefore required to allow the controller to keep sampling in the event of a failure of the mains supply. A sealed lead acid battery has been chosen as the backup supply source. These are completely maintenance free and sealed, preventing explosive gases escaping during charging. To simplify the design the battery will be continuously float charged, with the controller permanently drawing power from the battery. The power requirements for the controller are estimated as very small, less than 200mA. Most power outages are limited to a maximum of a few hours, as a result the smallest 12V

sealed lead acid battery easily available was selected being 1.3AH. This capacity should allow the system to continue sampling data for a period of approximately 6 hours, longer than any power outage can be expected for. The service life of the battery under constant float charging should be up to 10 years.

Figure 4.10: Sealed Lead Acid Battery

The PICAXE chip, LCD and EEPROM require a 5V power supply. A L7805CV voltage regulator has been selected to step down the voltage to a constant 5V supply. This component is rated at a maximum of 1 Amp and utilises overheating protection, making it virtually indestructible. Filter capacitors were also included to eliminate any unwanted ripple component introduced by the charger. A TO-220 package has been chosen.

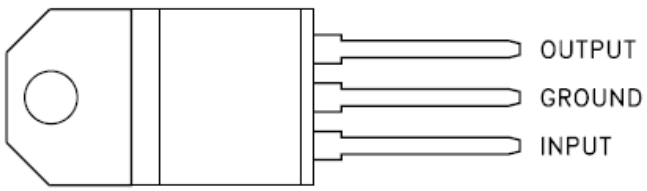


Figure 4.11: L7805CV 5V Regulator
Source: STMicroelectronics (2003)

This configuration ensures that there will be no power outage experienced by the microcontroller. Measurements will thus be taken uninterrupted, preserving accuracy.

4.3.7 Real Time Clock

In order to free the microprocessor from time keeping tasks, and provide an accurate time base, an external RTC (Real Time Clock) is used. RTC's are available in a range of configurations and interfaces. The part selected is a DS1307 serial clock produced by Dallas Semiconductor. It is an I2C device and can be easily connected with the other devices on the I2C bus, without the need for any extra interfacing hardware. The chip counts seconds, minutes, hours, date, month, day of the week and year, including leap year compensation to the year 2100. Once the chip is initialized with the current time and date it will automatically keep the correct time with no further input needed.

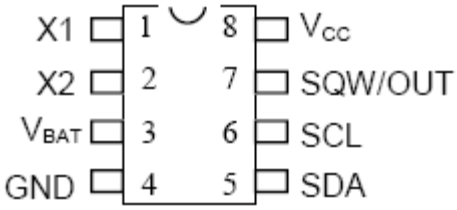


Figure 4.12: DS1307 RTC
Source: Dallas Semiconductor

The DS1307 also contains 56 bytes of ram, non-volatile when backed up by a lithium cell. The address map of the device is shown below.

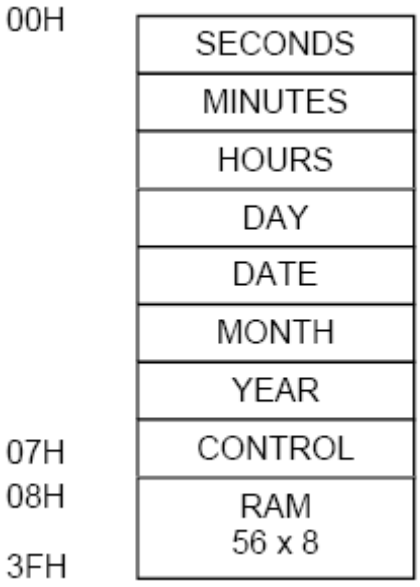


Figure 4.13: DS1307 Address Map

Source: Dallas Semiconductor

The device has extremely low power consumption, less than 500nA when using the backup battery. The device switches from its main supply to the backup battery automatically when the voltage level drops.

4.3.8 Human Interface

A user friendly interface is necessary to convey information about operation and easily set/monitor parameters. For basic status information, two LED's will be used to constantly display the state of the Normal and Rapid outputs. For more detailed information, a Liquid Crystal Display (LCD) will be incorporated. LCD's are available in a range of types. These include number displays such as on a wrist watch, graphical displays used for displaying graphics and character displays, such as a typical fax machine may use. These may be selected in backlit or non-backlit models. The AXE033 display module was chosen as it has a number of advantages over a standalone LCD. An onboard driver chip allows display information to be sent via a single serial line or by I2C interface. The I2C mode was once again chosen to conserve I/O. Pull up resistors are built into the

module and therefore not needed for the EEPROM when the module is connected to the same I2C bus.

Fitted is also a connector for the DS1307 RTC to plug straight in, along with a battery holder for a CR2032 lithium coin cell backup battery. This battery allows the RTC to keep time in the event of a power failure.

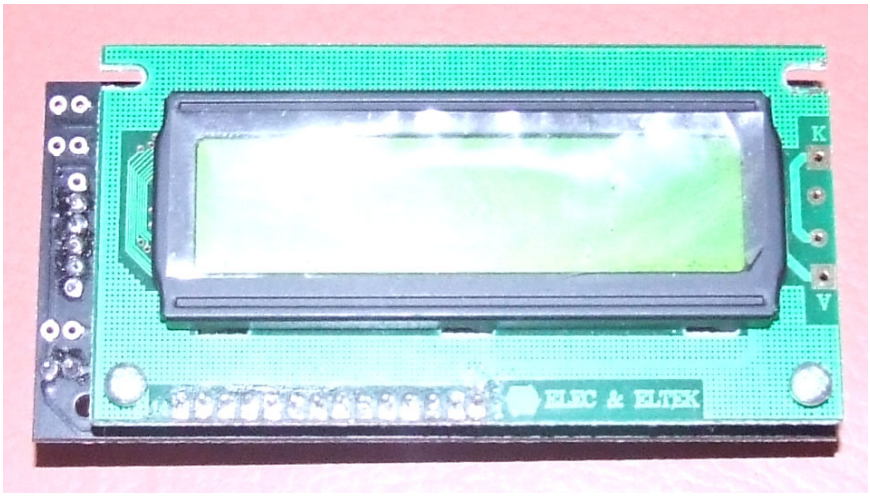


Figure 4.14: AXE033 LCD Module

The firmware chip decodes the I2C information sent and converts it to parallel data which is then sent to the display. The LCD display is a HD44780 compatible model, this being the most common standard for character LCD displays. The LCD is a double line type with sixteen characters per line. It contains a full ASCII character map, allowing any character in the following figure to be displayed at any position on the screen.

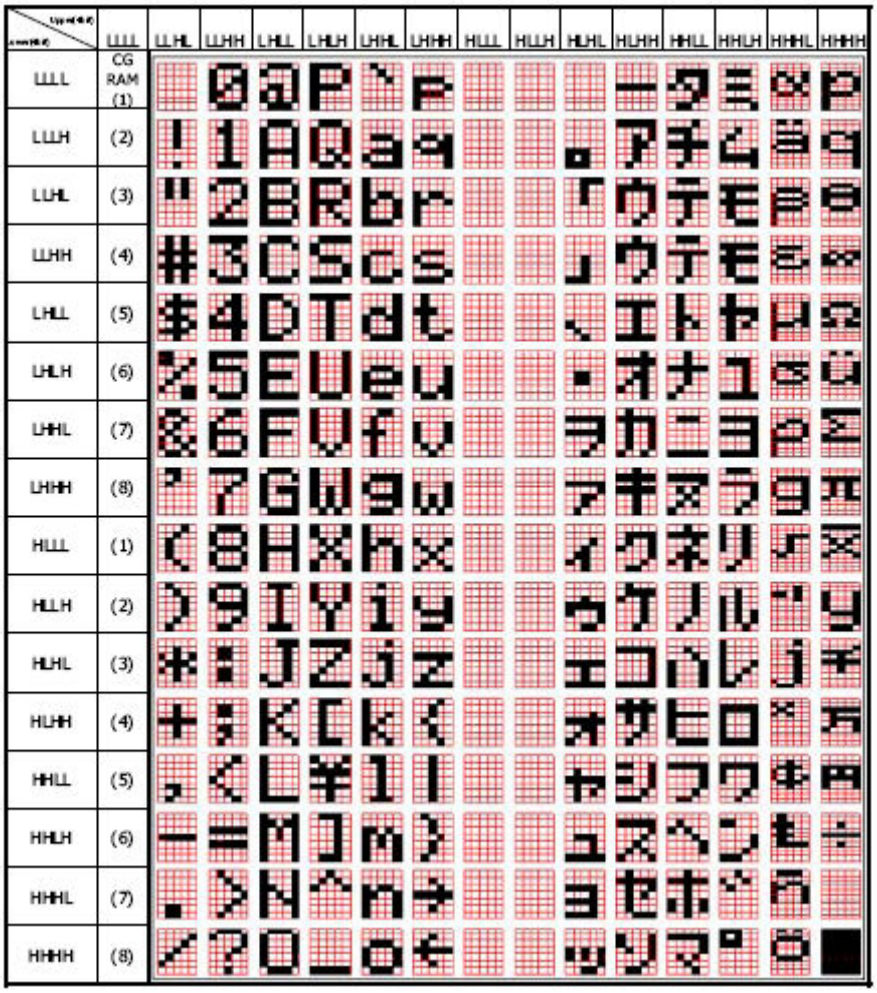


Figure 4.15: LCD Character Map

Source: Revolution Education (2004)

To write to the LCD the I2C slave address is set to binary 11010000. To display a character the character string is sent followed by the byte 255, this signifies the end of transmission and causes the firmware chip to write to the LCD. To send an instruction to the LCD, the byte 254 is first sent, followed by the instruction and finally byte 255. A wide range of instructions are available to perform a number of functions such as change the position of the cursor, switch the display on or off, scroll the display left or right and many others. A list of these commands is presented in Appendix D.

To input data it was initially decided that two switches would be sufficient. Parameters would be changed similar to a digital wrist watch, a single key

incrementing the value, and another key to switch modes. Spare input lines on the PICAXE were ensured so that additional buttons could be added if this setup was found to be difficult to operate. The switches selected are of simple push button SPST (Single Pole Single Throw), momentary, normally open design. The switch input lines are to be high when the switch is operated. This is achieved by use of pull down resistors. When a switch is pushed the supply voltage is lost across the pull down resistor and the microcontroller pin sees 5V, signalling a high. With any form of mechanical switch, switch bounce will exist. To simplify and minimise hardware, switch bounce was compensated for in software.

The two LED's used are green and contained within a 16mm bezel. They are rated at 2V and 20mA. To achieve the correct current from the 5V supply 120Ω resistors are placed in series with each LED. The Darlington driver chip is used to sink the current activating each LED.

4.4 System Schematic

The schematic for the system was drawn using AutoCAD, it is presented in figure 4.16.

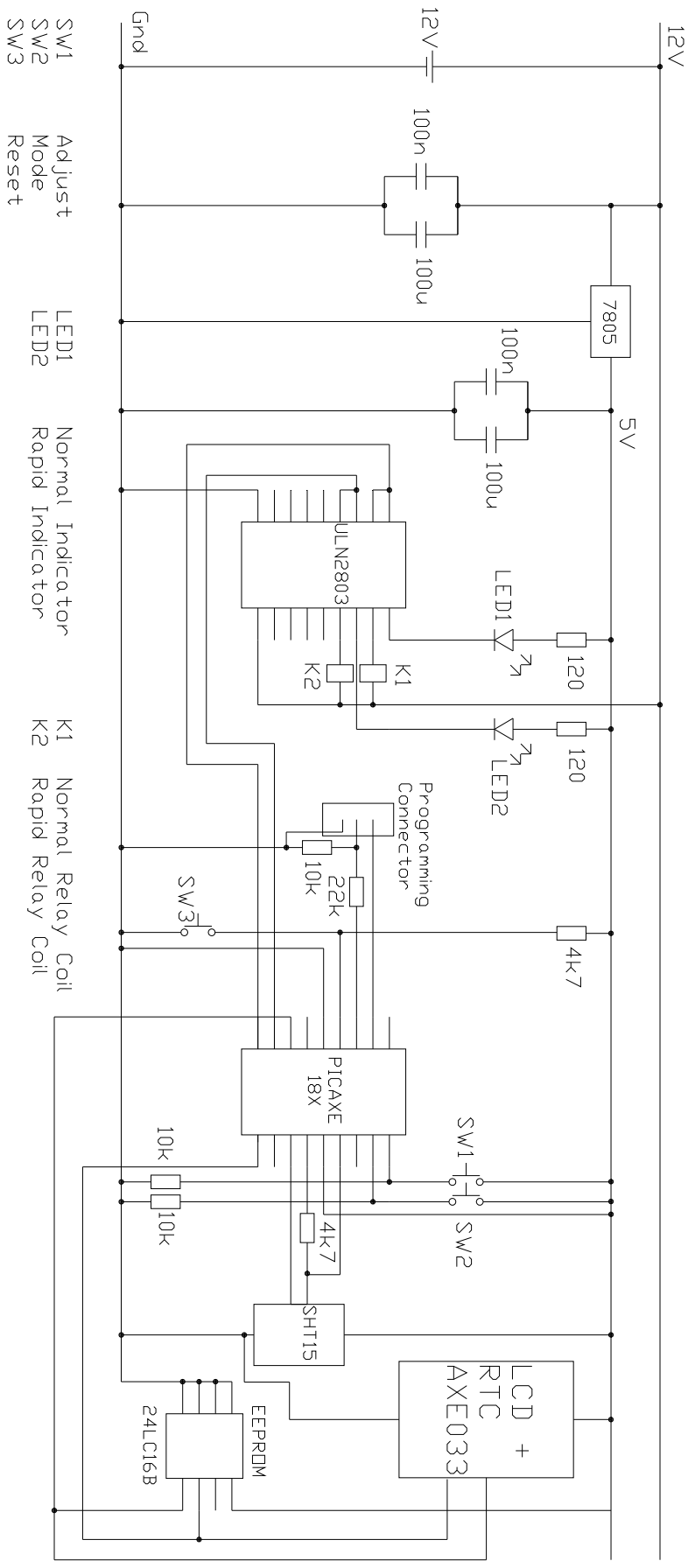


Figure 4.16: System Schematic

4.9 PCB

The Protel DXP software package was used to design a PCB (Printed Circuit Board) for the system components. Due to time constraints, the PCB was designed but not constructed. The PCB designed is a double sided type. The auto route command was used to automatically place the tracks on the PCB. The figure below is a representation of the design; the red is the top layer while the blue is the bottom layer.

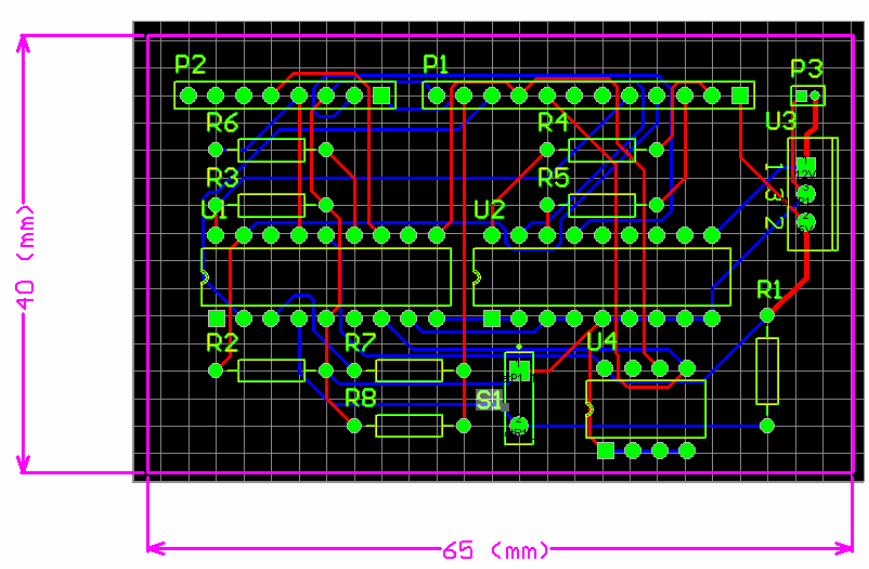


Figure 4.17: PCB

Further design details are provided in Appendix E.

Chapter 5 – Hardware Implementation and Testing

5.1 Initial Testing and Construction

To test the viability of the hardware and that the configuration would operate as expected, the design was initially bread boarded. This allowed any hardware problems to be identified and any necessary modifications to be easily made. To begin the PICAXE was mounted on the bread board and the required serial interfacing circuitry added. The 5V regulator was used and this connected to a 12v source for testing and the serial cable connected. This was to verify the ability of the chip to communicate with the programmer. Once successful the EEPROM and ULN2803A IC's were mounted on the board and connected as per the schematic. A LED and current limiting resistor were connected to the ULN2803A. The LCD display module required some assembly. The display was mounted on the firmware PCB and the header pin connections soldered into place. The display was powered and the contrast turned to maximum to check the display was active.

This breadboard platform was used as a base to become familiar with the PICAXE system. Simple test programs were written to perform straightforward functions, such as flash the LED at intervals. Once simple functions were mastered, the display and switches were connected to the prototype. Further simple programs were written to display strings on the display after a key press. Additional testing was undertaken with routines to test the writing and reading to the EEPROM, and display these values on the LCD. The RTC was tested in similar fashion, by setting the time with the keys and also displaying the time on the LCD.

Finally the connections to the DS18B20 Temperature sensor were added. A basic routine was written to read the temperature from the device and display this on the LCD. Correct operation was verified by using heat from the hand to warm the sensor and check the displayed temperature rose accordingly. The sensor was

then placed within a freezer and negative values checked. The reading of the ambient temperature was compared to a digital thermometer, the result was a very similar reading, and thus the sensor was deemed to be operating correctly.

After the bulk of the program appeared to be working correctly, the SHT15 sensor was integrated into the design. An image of the completed breadboard prototype is shown below.

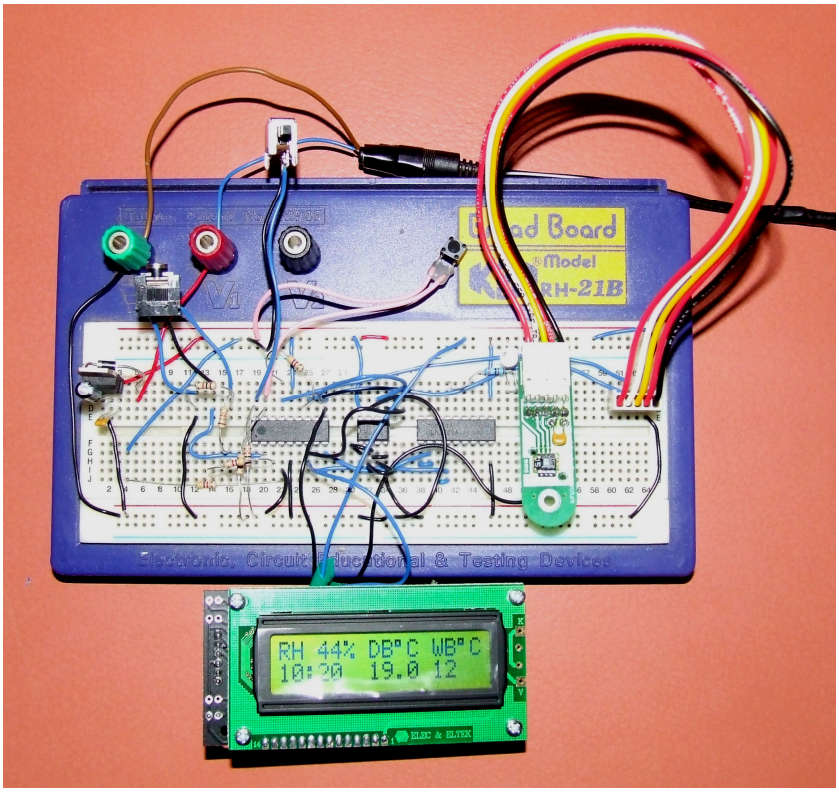


Figure 5.1: Bread Board Prototype

5.2 Final Prototype Construction

The main controller was designed to be mounted in an undercover area. This being the case and in an agricultural environment, the main concern is accumulation of dust. The final prototype was housed in a case containing the all peripherals except the temperature sensor, which was mounted remotely to allow mounting in a position that will reflect an accurate ambient temperature and

humidity. As the design is intended to be mounted undercover, UV radiation is of no concern. A polycarbonate box with 3mm walls measuring 222(L) x 146(W) x 55(H) mm was selected, large enough to contain all the components including the SLA battery. It features a tongue and groove sealing system with a neoprene gasket, and the stainless-steel lid-fixing screws thread into brass inserts that are outside the sealed area. The case has a claimed IP65 rating, meaning total protection from dust and low pressure jets of water. The case was expected to be mounted with its base on a wall and the lid forming the front and user interface.

The positions of components were marked on the lid and holes were drilled in the case to fit the LED's, programming cable connector and switches. The mounting holes for the display were drilled next. Initial holes were drilled to allow a hacksaw blade to be used to cut out the slot to allow the display to be viewed. A file was then used to create a smooth finish to the slot.

Holes were drilled in the body of the box to allow cable glands to be mounted for entry of the power and sensor cables. A separate smaller case was selected to house the temperature/humidity sensor. A hole was drilled in the side of this box to fit a cable gland. The SHT15 sensor was mounted inside this box.

The LED's, switches, connector and display were fitted to the lid. A piece of clear plastic was stuck to the inside of the lid over the LCD slot. This was to provide sealing to prevent dust or moisture from directly contacting the LCD screen. Wires were connected and soldered to the appropriate connections. Header pin connector plugs were soldered to the LCD and PCB ends of the wires to allow easy connection.

The remaining hardware components were assembled and soldered to a small prototyping board, using single core insulated wire as connections. The bare prototyping board is shown below.

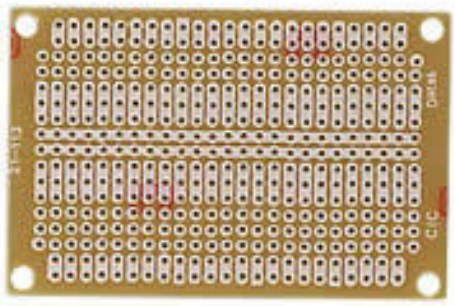


Figure 5.2: Prototyping Board

Connecting pins were kept towards the outside edges, header pins provide the connection between the lid and also the sensor. A screw type connector allows connection to the battery. The completed board is shown in figure 5.3.

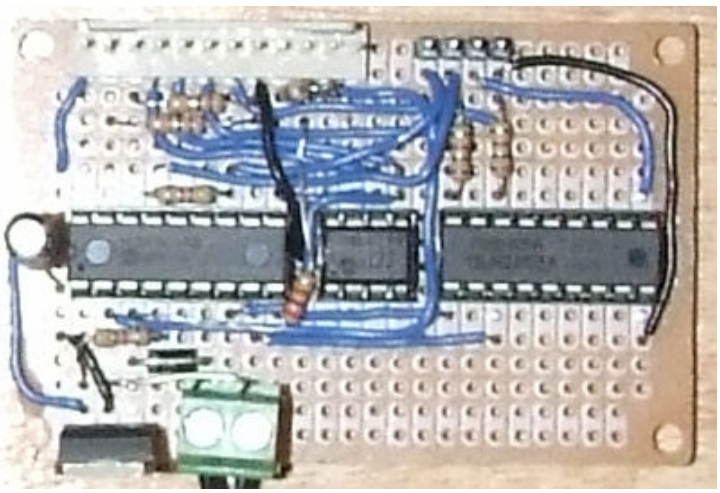


Figure 5.3: Completed Circuit on Prototyping Board

The charging cable was passed through the cable gland and connected to the battery. The battery leads also attach to the connectors on the board, providing power. Initially the sensor was connected to the board by a piece of 5 core round cable approximately 1.5 metres long. Only four of the wires were used with the fifth left unconnected. Communication with the sensor could not be established through this cable. After investigation it was deemed that the length of cable was resulting in cross-talk between the clock and data lines. To solve this problem the type of cable was changed. A piece of ordinary 4 core phone cable was used again of length approx 1.5 metres. To minimise cross-talk the power and ground lines were run as the middle two conductors, separating the data and clock lines.

This configuration presented no problems with communication being established straight away.

The completed prototype housed in the protective case is shown in the following figures.

Figure 5.4: Controller Displaying Main Page

Figure 5.5: Controller Displaying Set-points

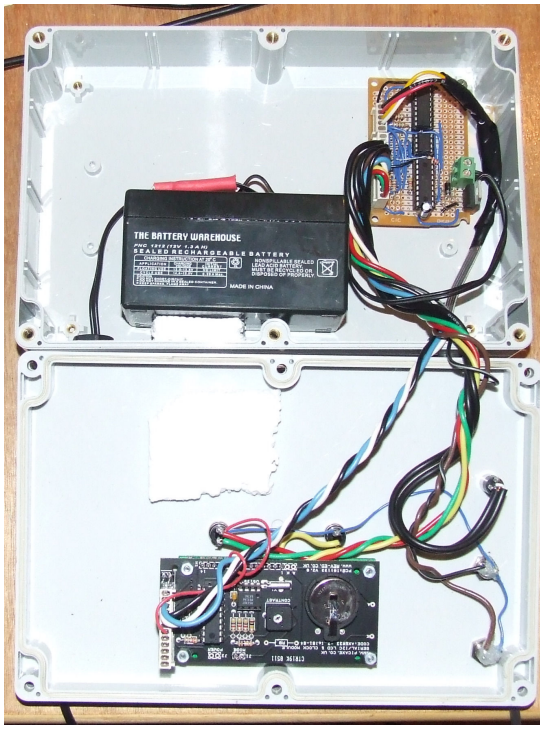


Figure 5.6: Controller Internals

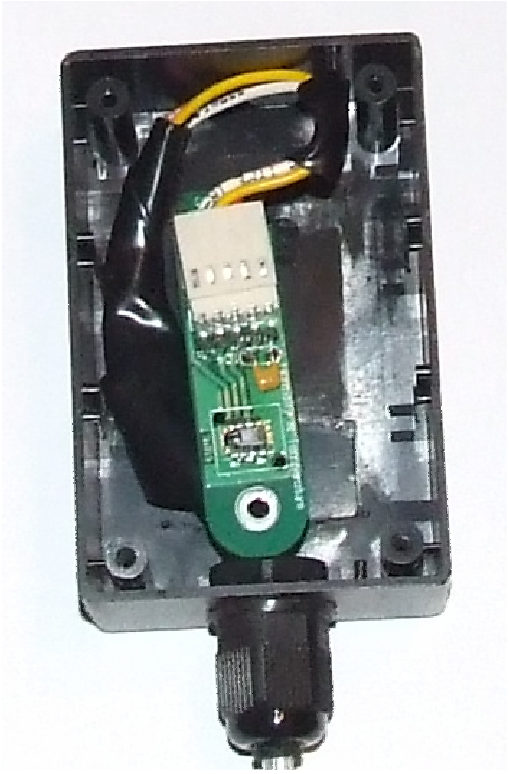


Figure 5.7: Remote Sensor

Chapter 6 - Software Development

The simulation and parameters determined for the statistical method of control in previous sections were adopted as the aim for the PICAXE software. The intended approach was bottom up implementation, the lower modules performing specific functions were separated from the main program. This approach allowed testing of individual sub-sections of code without reliance on the main loop. The final BASIC code is presented in Appendix F.

6.1 PICAXE Programming Editor

The development of the code for this project was completed entirely within the PCIAXE Programming Editor. This application is designed for use exclusively with PICAXE chips; as such a range of special features for use with them has been built in. The programming language used is BASIC, this language is simple and easy to follow, but without some of the useful commands available in higher level languages such as C.

The Editor can interface directly with a PICAXE chip through use of a serial port. The laptop PC used to program the chips was not fitted with a serial port. As a result a USB to Serial adapter was needed. This connected to the serial programming cable which through a 3.5mm stereo plug provides the connection to the PICAXE.

The editor also contains a data link terminal, for retrieving data from the chip via the programming cable and a debug window, which allows the value of all variables to be displayed at the time a debug command is used in the program. A syntax check can be performed, alerting the programmer to a syntax error and also displaying the amount of memory the program will occupy. The figure below is an example of the PICAXE Programming Editor window.

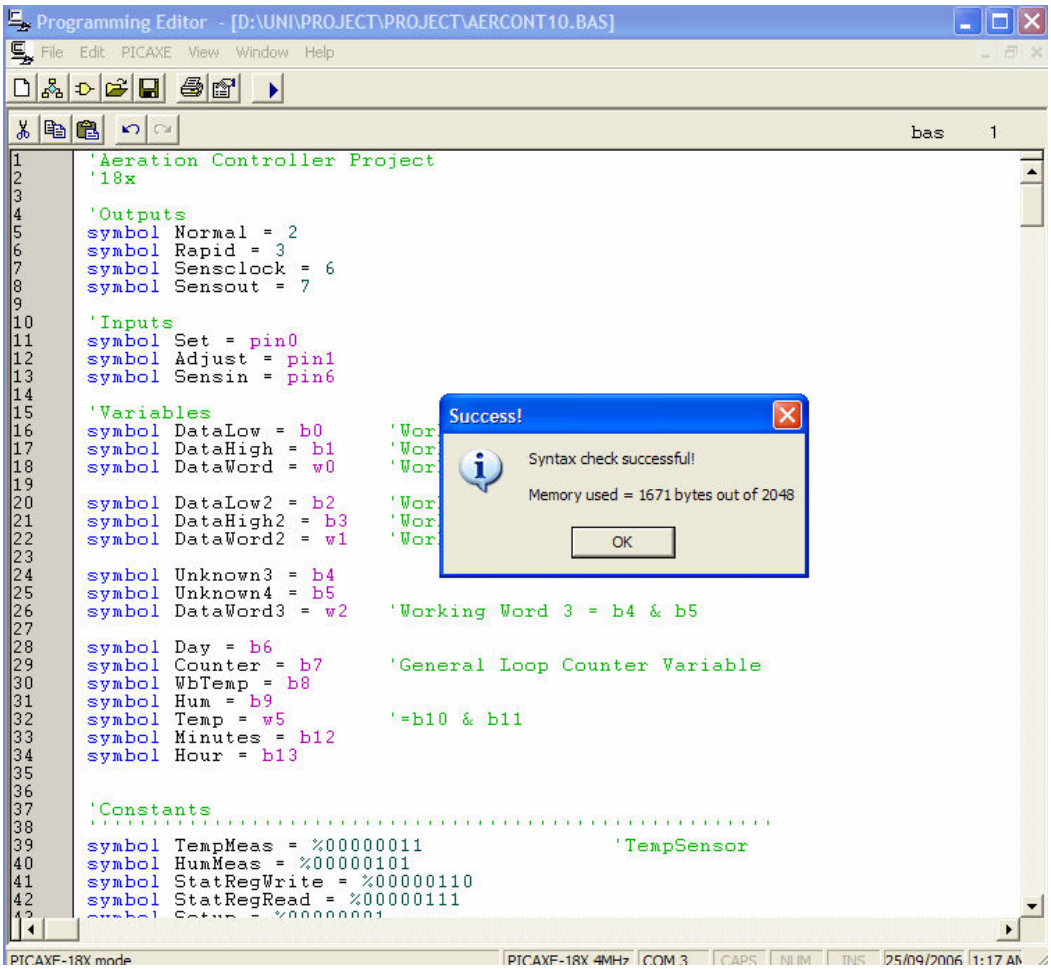


Figure 6.1: PICAXE Programming Editor

6.2 Code Development

The measurement, logging and calculation of set-points are the main functions of the program. Auxiliary functions include the display of information on the LCD display, outputting data for graphing and receiving input from the switches. A simplified flow chart for the system operation is shown below in figure 6.2.

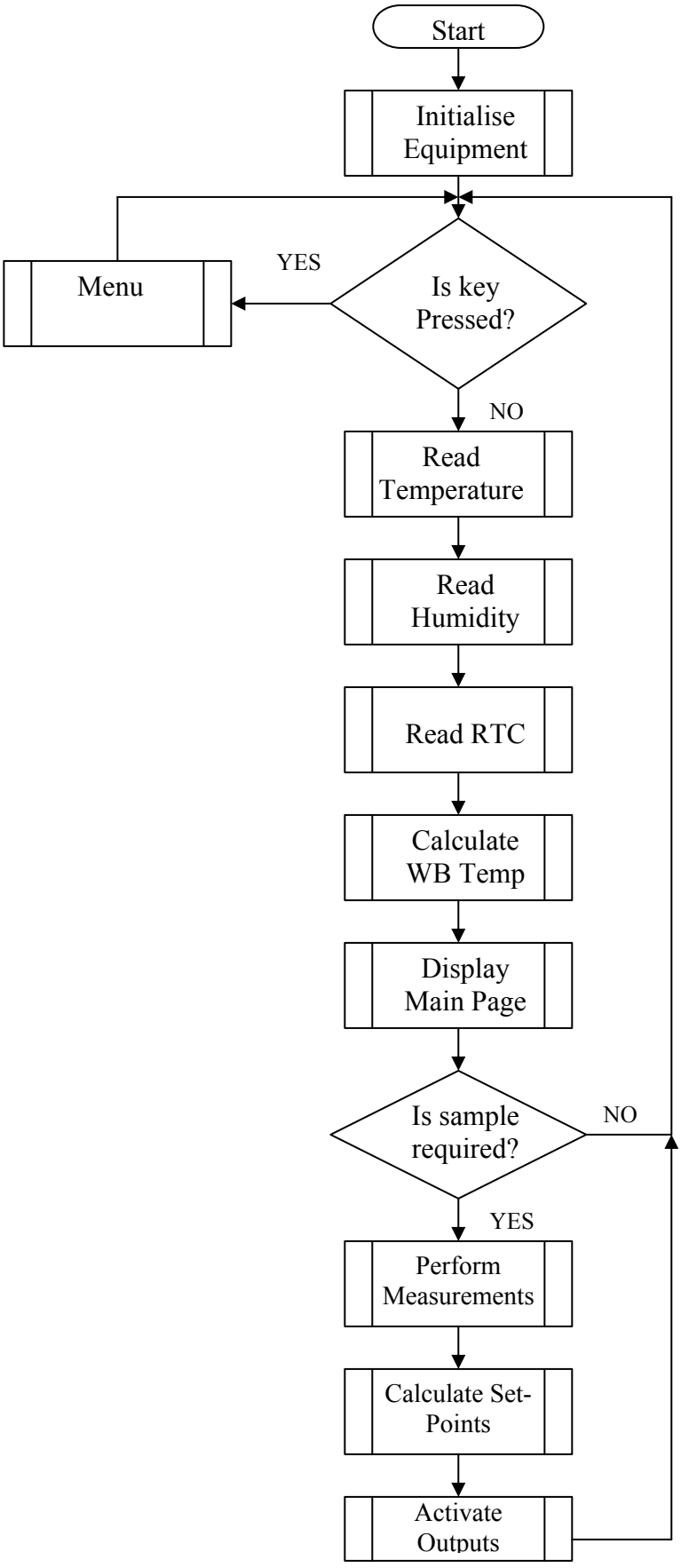


Figure 6.2: System Flow Diagram

The code will be discussed according to functional sections.

The PICAXE contains 14 general byte variables that may be used in mathematical calculations and program branches. These variables are allocated to storage as in the table below.

Table 6.1: Variable Allocation

DataWord (Temporary Word)	Byte 0	DataLow	Temporary Byte
	Byte 1	DataHigh	Temporary Byte
DataWord2 (Temporary Word)	Byte 2	DataLow2	Temporary Byte
	Byte 3	DataHigh2	Temporary Byte
	Byte 4	NormalSet	Normal Set-point
	Byte 5	RapidSet	Rapid Set-point
	Byte 6	Day	Current day of week
	Byte 7	Counter	Temporary Counter
	Byte 8	WbTemp	Wet-bulb Temperature
	Byte 9	Hum	Humidity
Temp	Byte 10		Dry-Bulb Temperature
	Byte 11		
	Byte 12	Minutes	Current Minute
	Byte 13	Hour	Current Hour

6.2.1 Temperature and Humidity Sensor Interfacing

The purpose of this section of code is to interface the PICAXE with the SHT15 combined temperature and humidity sensor. As the device is not compatible with I2C devices, a custom routine had to be written. To begin communication with the device a transmission start sequence must be performed. This sequence is shown below.

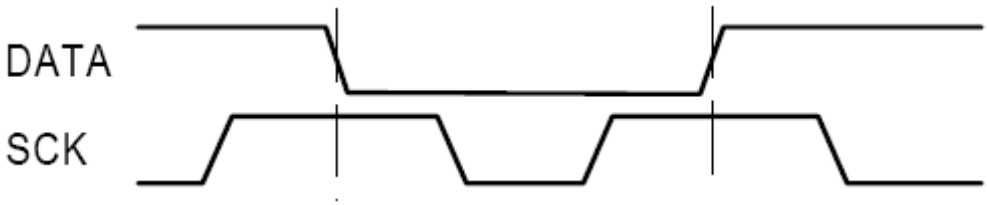


Figure 6.3: Transmission Start Sequence

Source: Sensirion

From this point a range of commands may be entered, these are summarized in the table below.

Table 6.2: SHT15 Commands

Source: Sensirion

Command	Code
Reserved	0000x
Measure Temperature	**00011**
Measure Humidity	**00101**
Read Status Register	00111
Write Status Register	00110
Reserved	0101x-1110x
Soft reset, resets the interface, clears the status register to default values wait minimum 11 ms before next command	**11110**

Each command byte must be followed by an acknowledge signal. The SHT15 controls the data line and pulls it low for a ninth clock cycle, signifying the successful receipt of the command byte. The initial code performs a status register write, setting the resolution of the sensor output to a 12bit temperature measurement and an 8bit humidity measurement. To achieve this binary byte 00000001 is written to the status register upon start-up. The status register contents and corresponding configurations are shown below.

Table 6.3: SHT15 Status Register

Source: Sensirion

Bit	Type	Description	Default	
7		reserved	0	
6	R	End of Battery (low voltage detection) '0' for Vdd > 2.47 '1' for Vdd < 2.47	X	No default value, bit is only updated after a measurement
5		reserved	0	
4		reserved	0	
3		For Testing only, do not use	0	
2	R/W	Heater	0	off
1	R/W	no reload from OTP	0	reload
0	R/W	'1' = 8bit RH / 12bit Temperature resolution '0' = 12bit RH / 14bit Temperature resolution	0	12bit RH 14bit Temp.

Upon receiving a measurement command for either temperature or humidity, the microcontroller must wait a period of time for the measurement to be completed. This period is dependant upon the resolution of the measurement. In the 12/8 bit configuration used the measurements require 11/15 ms respectively. During this time the sensor must control the data line, keeping it high until the measurement is complete. At this time the sensor pulls the data line low, signifying completion and the microcontroller may again start toggling the clock line. Two bytes of data are then sent by the sensor. These are most significant bit first, right justified. If the measurement is only of 8 bits size then the first byte transmitted is empty. Following the measurement data is an 8 bit CRC checksum. Following each byte of data transmitted by the sensor, the microcontroller must provide an acknowledge signal by pulling the data line low and toggling the clock line. If the CRC byte is not used communication can be terminated after the last measurement byte by a high acknowledge. The sensor will automatically return to low current sleep mode once the measurement and communication have ended.

With the communication protocol needs established code was developed to perform these functions. A subroutine called TransStart performs the transmission start protocol by toggling the data and clock lines as appropriate. Separate subroutines named WriteData and ReadData were written to send and

receive data to and from the sensor. Each bit operation is performed by multiplying the byte by two; this is equivalent in binary to a right shift of one bit. Acknowledge routines were created to provide a high or low acknowledge routine as appropriate. With these lower directly interfacing routines written it was possible to construct the measurement routines that would convert the read in bytes to physical values.

The temperature sensor output is inherently linear, to convert the digital value to a corresponding temperature the following formula is used.

$$\text{Temperature} = d_1 + d_2 \bullet SO_T \quad \textbf{(6.1)}$$

The values for this formula vary with supply voltage and desired temperature unit according to the following tables.

Table 6.4: Temperature Conversion Coefficients

Source: Sensirion

VDD	d_1 [°C]	d_1 [°f]
5V	-40.00	-40.00
4V	-39.75	-39.50
3.5V	-39.66	-39.35
3V	-39.60	-39.28
2.5V	-39.55	-39.23

SO_T	d_2 [°C]	d_2 [°f]
14bit	0.01	0.018
12bit	0.04	0.072

The measurement scale to be used for this project is °C, the supply voltage for the sensor is 5V. Due to the inability of the PICAXE to perform decimal calculations, the formula must be reconstructed. To preserve one decimal place of accuracy, the measurement value must be multiplied by ten. The multiplication coefficient d_2 can be rewritten in terms of a division by taking the reciprocal, 1/0.04 = 25. The coefficient d_1 must also be multiplied by ten, resulting -400. The value d_1 is the offset above zero that converts the range to start at -40°C. Due to the inability of the PICAXE to handle negative numbers, the negative offset of 400 will not be added, data will be stored in this offset form with the value zero signifying a temperature of -40°C. Multiplication by ten results in a number too large to be

stored in a single byte variable, a double byte or word variable was therefore allocated to hold the temperature value.

The digital relative humidity measurement unlike the temperature measurement is non-linear. A conversion formula requiring floating point multiplications results in a maximum error of ±0.1%. This formula can not be performed by the PICAXE, instead a simpler two range linear formula was be used. This simpler formula results in a maximum error of ±0.8%. This formula is presented below with the ranges it is validity.

$$RH_{real} = (a*SO + b) / 256 \quad \textbf{(6.2)}$$

Where SO is the 8 bit humidity sensor output

Table 6.5: Temperature Conversion Coefficients

Source: Sensirion

Validity	a	b
$0 \leq SO \leq 107$	143	-512
$108 \leq SO \leq 255$	111	2893

This equation can be performed by the PICAXE in this form. While performing the calculation a temporary word variable is required to prevent overflow. Underflow must also be checked in the event that SO×143 is less than b. In this case the intermediate value is set to b so that after subtraction the final result is zero.

6.2.2 Cyclic Redundancy Check

The SHT15 temperature sensor is capable of performing an 8bit CRC checksum. This check can be performed in two ways, a bitwise method and a byte wise method. To perform the bitwise method, the receiver must emulate the structure of the generator. The process is presented in Appendix G. The final byte after all

steps are complete is the CRC value. Figure 6.4 shows the internal structure of the CRC-8 generator.

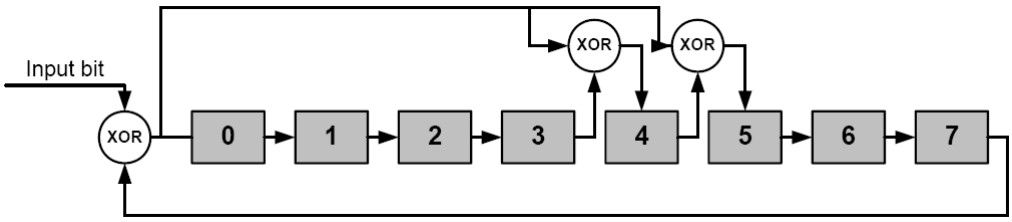


Figure 6.4: Internal CRC-8 Generator Structure

Source: Sensirion

The byte wise method requires the CRC data to be stored within a 256 byte lookup table. Values are XOR'd and the result used as an index into the table. The final byte obtained is the CRC value. In both these methods the CRC value calculated is compared to that received. If the values are identical then the data has been transmitted successfully and is not corrupt. If the values differ then the data has been corrupted and is not valid. The CRC has not been implemented in this prototype due to limited memory capacity.

6.2.3 Wet Bulb Temperature Calculation

This section of code has the specific function of calculating an approximate wet-bulb temperature from the dry-bulb temperature and relative humidity measurements. The conversion table presented in Appendix B is the basis for this calculation. The initial step prior to device operation is to load the EEPROM with the lookup table. The maximum number of elements in a column is twenty. As discussed in the System Design section, the EEPROM is divided into eight blocks, each of 256 bytes. The maximum number of columns that will completely fit within a block is twelve. Therefore five EEPROM block are used, each containing twelve columns of the lookup table. Each column represents a whole degree of dry-bulb temperature from the range -5°C to 50°C. Each row within a column contains a humidity value, each step through the column to reach a

humidity value is equivalent to half a degree depression. This is graphically represented in Figure 6.5 below.

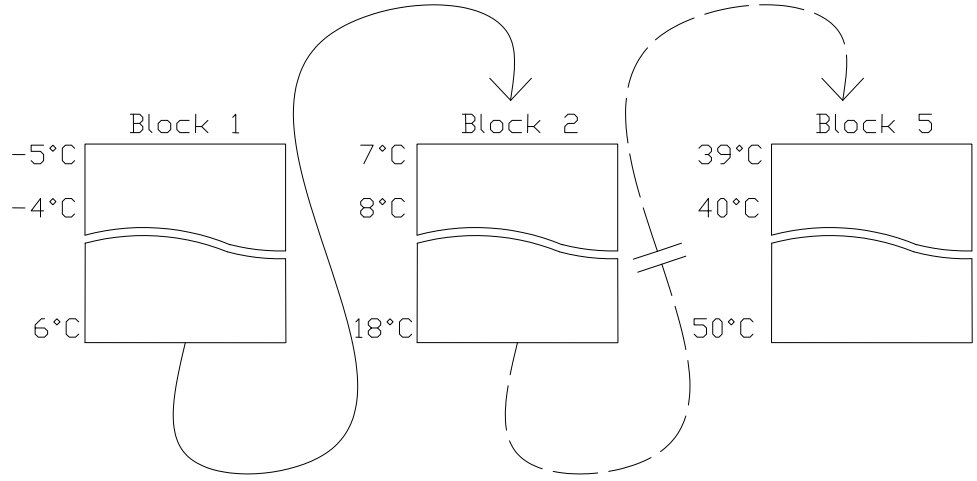


Figure 6.5: Wet Bulb Lookup Table Calculation

This lookup table must firstly be loaded into the EEPROM memory. A dummy BASIC program was written with the sole purpose of storing this table in the memory. The values were hard coded as instructions within the program memory for transfer to the EEPROM. This software was loaded into the PICAXE and allowed to run, values were transmitted back to the PC serial interface and verified.

The table is quantized into half degree steps of depression. This results in a non-continuous range of humidity values within the lookup table. This causes an element of rounding within the calculation, however this will impact minimally on the overall operation of the design as the resolution of data storage used is one degree. Dry bulb measurements below -5°C or above 50°C should rarely occur. In the event of a measurement outside this range occurring, the depression will be calculated based on the closest extreme of data within the lookup table.

To implement this in software a separate sub-routine was written. The dry-bulb temperature is used to determine which block of EEPROM contains the relevant lookup table column. The temperature is used as an index into the block, each element of the column is then stepped through and compared to the humidity

value. Once the value from the table is less than the measured humidity, the index into the table column is multiplied by five. This is the depression in degrees scaled by 10, the form the dry bulb temperature is in. The depression is subtracted form the dry-bulb temperature and then divided by ten. This results in a wet-bulb temperature value, quantized to one degree intervals and offset above zero by 40 to avoid negative numbers. This value will always be smaller than 255 and therefore is stored in a single byte variable. The values are clipped to a minimum of 20 and a maximum of 89, these correspond to -20°C and 49°C. This will be further discussed in the data storage section.

6.2.4 Real Time Clock Interface

Reading the time from the RTC is a simple task. An I2C read is performed at the RTC address. The data is received in the BCD format. To simplify clock adjustment in later modules, this data is converted by a subroutine from BCD (Binary Coded Decimal) into binary data. The values of hours and minutes are stored in separate single byte variables of RAM within the PICAXE during each RTC read.

6.2.5 Measurement Time Determination

This section of the code determines when the next measurement and set-point adjustment needs to be made. As explained in previous sections, measurements are taken at fifteen minute intervals.

The RTC read is performed prior to this module of code. To determine when a period of fifteen minutes has elapsed, the minute data is divided by a modulus 15. The result is zero when the minutes are either 0, 15, 30 or 45. This value is used as a conditional branch when equal to zero. The main loop will complete in less than one minute. To avoid the branch from occurring more than once during the same minute, a further condition is stipulated. The non-volatile RAM available in

the RTC is used to store the minute value of the last measurement. If the minutes of the current measurement is not the same as the minutes of the last measurement, the program will branch to allow further action. Otherwise if the minute is the same as the current time then the measurements has already been taken dor the current minute. During the read from the RTC the circular buffer position and block of EEPROM currently in use is also read back. The usage of these values will be discussed in the next section. The memory map of the ram locations used within the RTC is displayed in the figure below.

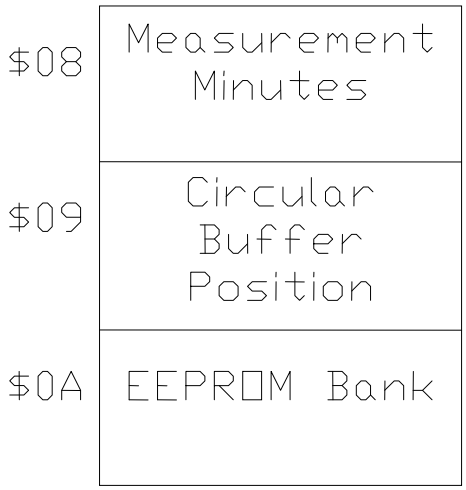


Figure 6.6: RTC RAM Locations Used

6.2.6 Storing Measurements

As previously discussed the temperature measurements are stored in a circular buffer of samples, containing three days of data. At this sampling rate the buffer requires a total of 288 samples. To store this information in EEPROM, two blocks are required, as each block contains 256 bytes. Block six is filled completely while block seven contains only 32 samples. This configuration is shown below.

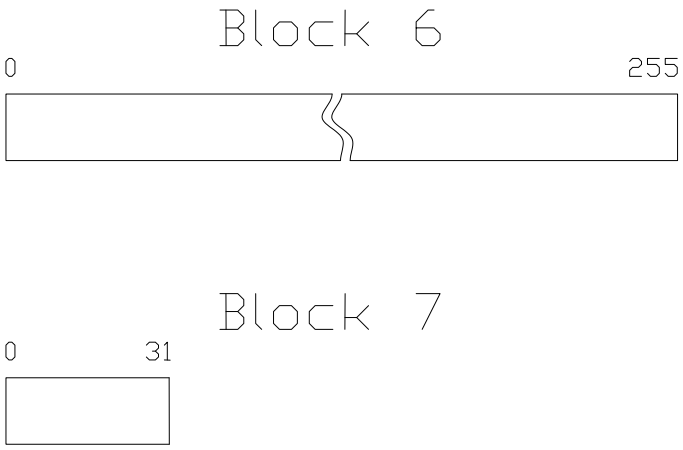


Figure 6.7: Sampling Buffer

The values read back form the RTC in the previous section are used here. On initial activation of the device the circular buffer position is set to zero, and the EEPROM set to block 6. The circular buffer position is incremented with each measurement until it reaches 255. At this point it reverts to zero and the block is changed to block 7. The position is once again incremented with each measurement until position 31 is reached. The block is then reset to block 6 and position zero, old data is overwritten.

The calculation of the set-points relies upon a complete set of samples. This means that the set-point can not be calculated until the buffer of 288 samples is complete. A bit flag is used to signify when the data set is complete. Upon a master reset the flag is cleared to a zero. When the block is set to block 6 after block 7 is complete, the flag is set to a one. This signifies that the samples are complete and the set-points can be calculated. While the sample buffer is incomplete and the outputs inactive, the display provides a notification. The message “INACTIVE” is flashed alternatively with the main screen.

6.2.7 Calculating Number of Occurrences

This code calculates the number of times each temperature value occurs within the buffer, in sequential order of lowest to highest temperature. The entire buffer of samples is stepped through sequentially. The temperature value stored within

each element is used as an index into the PICAXE RAM. Each time a value is encountered within the EEPROM, the count of that value in the RAM is incremented by one. This is continued until the entire buffer has been read. As a result the RAM contains the number of times each temperature has been recorded in the last 3 days, organized in ascending order from -20 to 49°C. The RAM is split into two blocks, as a result the first block contains the data from -20 to 27°C and the second block from 28 to 49°C. This is represented graphically in the figure below.

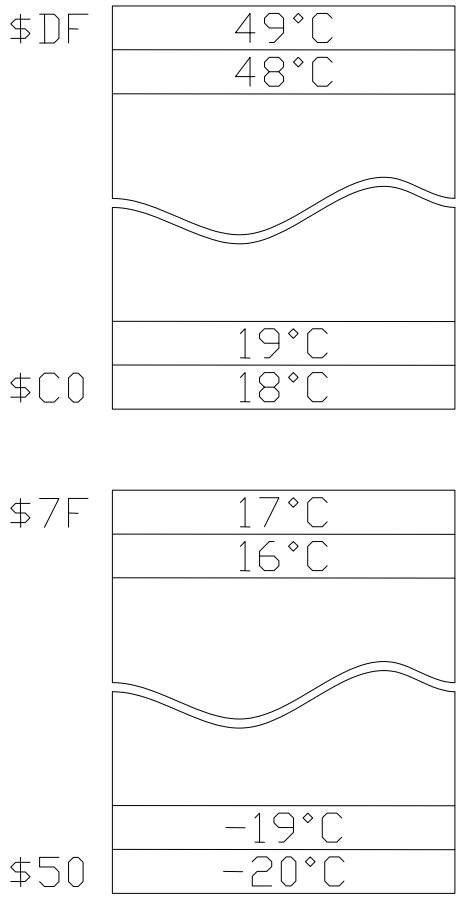


Figure 6.8: Temperature Histogram Locations

6.2.8 Calculating Cumulative Probability

To determine the set-points for the Normal and Rapid modes the cumulative probability is used. The sample buffer of 288 samples represents three days of

data. As discussed in previous sections the appropriate percentage running time for the Normal and Rapid modes is 14% and 50% respectively. This equates to 41 and 144 samples. The cumulative density is taken starting at the -20°C. For each temperature the number of occurrences is added to the total stored within a word variable. Once a total of 41 samples are reached this temperature is stored as the Normal set-point temperature. The process is repeated for the Rapid set-point, once 144 samples are reached this temperature corresponds to the Rapid set-point temperature and stored as such.

6.2.9 Checking Set-points

This subroutine checks the current temperature against the Normal and Rapid set-points and activates outputs accordingly. There are three output state conditions possible based on the temperature relative to the set-points, these are summarized in the following table.

Table 6.6: Temperature Output Conditions

Temperature Condition	Normal Output	Rapid Output
Normal < Rapid < Temp	Off	Off
Normal < Temp < Rapid	Off	On
Temp < Normal < Rapid	On	On

6.2.10 User Interfacing

The most challenging piece of code encountered in the development of this project was the user interface. This is split into two sections, a routine to display the main page of information and a routine to handle menu information. The main page contains the current relative humidity, dry-bulb temperature, wet-bulb temperature and the time. A layout was determined which would allow these

values to be displayed concurrently on the two sixteen character lines of the display. This layout is shown in the figure below.

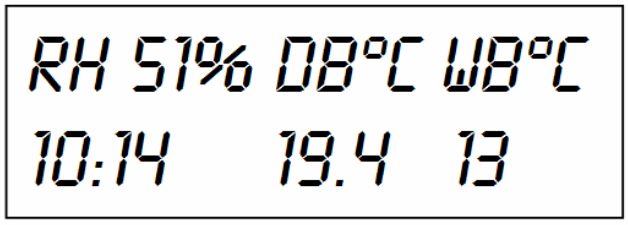


Figure 6.9: Display Main Page Layout

Due to the relatively quick execution of the main loop, an interrupt was deemed unnecessary for entering the menu. The buttons on the front of the display are assigned as mode and adjust. The mode button is used for moving through the menu, while the adjust button increments, selects or changes the value of the current parameter.

The mode button is checked at the beginning of the main loop. If active a branch is executed to the setting routine. This is a permanent branch and not a subroutine. This is due to the large number of possible exit points from within the menu code. The mode button must be held for three seconds to initiate the menu or the program returns to the main loop.

6.2.11 Switch Bounce

Any mechanical switch with contacts will exhibit switch bounce. This phenomenon will cause an oscillatory voltage to be present at the microcontroller input. Such an input may cause erratic behaviour of the system such as multiple key presses being registered as a result of a single key press. Switch bounce may be countered using hardware or software methods. Hardware methods would have required extra components, as such a software solution was chosen.

A subroutine within the code named Debounce was developed to eliminate switch bounce interference. On entry the DataLow variable is cleared to zero. The subroutine contains a counter; the counter is incremented at periods or 50ms. If the counter reaches 255 the routine is exited with a value of 0 in the temporary byte. A press of either button during this timing period will set a bit flag to 1. The subroutine will then exit at this point. The program uses these bits to determine program flow. The bit flags used to signify button presses are illustrated in the following figure.

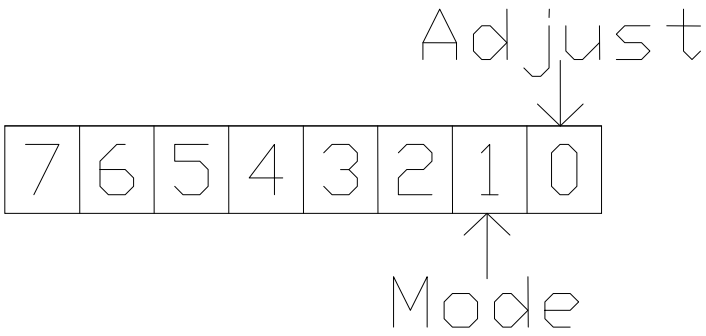


Figure 6.10: Debounce Bit Flags

6.2.12 Menus

The menus within the program allow data to be displayed on the screen. The time can be set in a fashion similar to a digital wrist watch. The menu items available are presented below in a flow chart form.

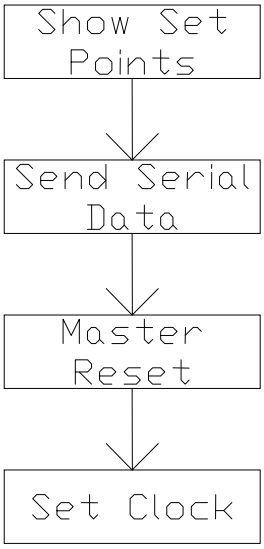


Figure 6.11: Menu items

6.2.13 Pre-defined Messages

To conserve program memory space the 256 bytes of EEPROM within the PICAXE are utilised to store strings. Display messages are pre-defined in blocks of sixteen bytes, each byte containing the ASCII code a character. Each sixteen block segment contains a string that fills one line of the LCD display. A sub-routine called PrintLine prints this string to the display at the position initialized prior to the function being called.

The EEPROM command is issued at the start of the program, followed by the ASCII characters to be used for the string. The programming editor loads these values into the PICAXE internal EEPROM during the program download. The strings stored within EEPROM are shown in the figure below.

SEND SERIAL DATA
YES NO
MASTER RESET?
SET CLOCK
SHOW SET-POINTS
RESET COMPLETE
INACTIVE

Figure 6.12: Pre-Defined Messages

6.2.14 Sample Downloading

The program allows for the data buffer to be transmitted serially to a PC. This is performed when selected from the controllers menu. The PICAXE 18X is capable of transmitting serial data via the programming cable. The "Sertxd" command is used to transmit the data. The data is transmitted at 4800 baud rate. The data from block 6 is sent followed by block 7. Each value is followed by $0D, a carriage return. The programming editor data terminal is capable of receiving the serial data. It can then be copied into a CSV (Comma Separated Variable) file, and used to form a plot of the temperature data. To simplify this process, a simple DOS command line program was found that was capable of receiving the data and storing it in a CSV file automatically. The program used is serialterm.exe, written by A. Schmidt, 2001. The serial port, baud rate and output file are set upon calling the function. The complete CSV file can be opened in MS Excel, the graphing facilities can be used to view the data graphically.

Chapter 7 – Analysis and Performance

Once the final prototype was completed and software modules written, testing was able to be carried out. The success of the tests were measured based on the operational requirements. This chapter outlines the tests and results obtained.

7.1 Initial System Test

The first test of the completed prototype was aimed at successfully recording a complete buffer of temperature samples. The prototype was placed outside with the main box undercover and the sensor box in an open environment. The power supply was connected to mains and the clock set to the correct time. A master reset was performed to clear the buffer entirely. As expected, the display began to flash the message "INACTIVE", signifying the sample buffer was not complete. The prototype was left uninterrupted for three days to log the ambient temperature data.

Upon recoding the 288th sample, the system screen returned to normal and the Rapid light illuminated. This signified that a complete buffer of samples had been stored. The programming cable was connected and the sample data read out. The figure below shows the recorded wet-bulb temperature, at a sampling interval of 15 minutes, over a period of three days.

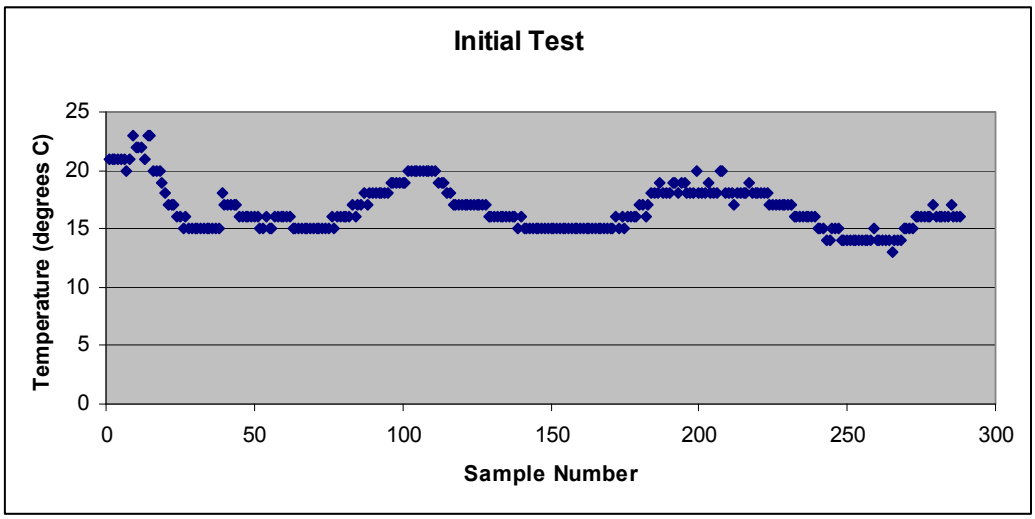


Figure 7.1: Initial Test Data

The graph clearly shows the temperature cycling over the three day test period. It is evident that the temperature can vary considerably in the short term; this is likely caused by atmospheric fluctuations such as varying cloud cover. It must also be noted that this is the wet-bulb temperature. The wet-bulb temperature is dependant on both the humidity and dry-bulb temperature components, for this reason the results are not intuitive. It is also apparent that the average daily temperature is falling over the course of the test.

The results of the test appear to be acceptable, the three days temperature fluctuations are clearly visible. No problems were observed with the device or its operation. With this test rated as successful, further testing was now possible to check the function of the set-point calculation.

7.2 Second System Test

The second system test was aimed specifically at checking that the set-points determined by the controller were providing correct operation. To save time, the data collected in the initial test was kept. This meant that the system operation could be monitored immediately, avoiding the need for another three days

sampling prior to the system functioning. The device was left mounted in the same position as during the first test.

The controller was allowed to operate uninterrupted until the buffer was again full. The sample log was downloaded to a PC prior to the circular buffer wrapping back to the first memory location and overwriting old samples. The data was formatted sequentially within MS Excel to create a graph of the entire test. This temperature data is included in Appendix H.

The results of the test are presented in Figure 7.2.

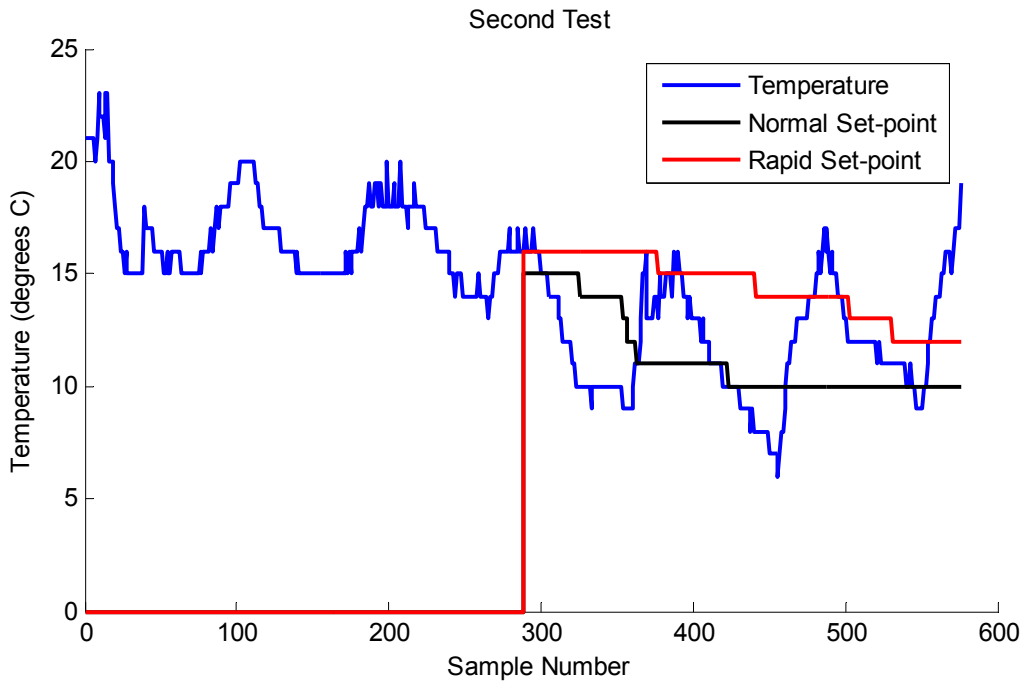


Figure 7.2: Second Test Results

The graph shows the temperature data collected over the six days. The set-points for the Normal and Rapid outputs are clearly visible. Each output is active when the temperature is equal to or below its respective set-point. The first three days data is required to initialise the system. Once this time has elapsed the set-points are calculated and the system is ready for operation. The average temperature observed over this six day test period appears to be falling, followed by a rise on the last day. The set-points reduce slowly to follow the temperature trend. It is

evident that the set-points are not reacting quickly enough to maintain the correct time percentages of 15 and 50%. These periods equate to sample lengths of 43 and 144 respectively.

7.3 System Improvements

Due to the slow response of the set-points to changing climate, an adjustment was deemed necessary to the system. To speed up the response, the buffer size was reduced to two days. This reduces the number of older samples stored and should speed up the response time.

To determine the effect of a reduced buffer size, the same data will be used. As it is impossible to reproduce the same temperature data the system was simulated in MATLAB. The code used to perform this function is included in Appendix I. The algorithm was reproduced with a buffer length of 192, equivalent to two days of samples. The data collected during the tests was used as the temperature and a sliding window represented the buffer. As the sample buffer required is only two days in length, the system will become operational one day earlier. The results of the simulation are presented in figure 7.3, along with the original three day buffer set-points for comparison.

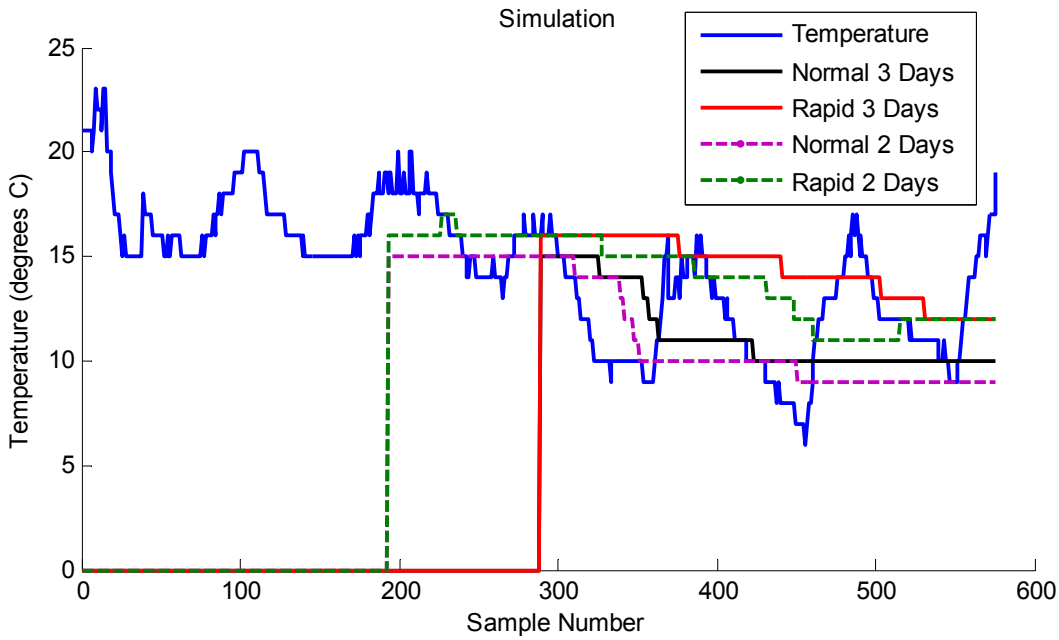


Figure 7.3: Buffer Size Simulation

It can be seen that the buffer of two days reacts quicker to the temperature average changing than the three day buffer. Analysis of the time percentages that the system would run in this case reveals that the two day buffer provides output periods significantly closer to the ideal percentages.

7.4 System Discussion

The system tests revealed the set-points adapting to follow the average temperature. The temperature data used for the tests is the worst case possible, this being a sudden drop in temperature, quickly followed by a rise. This results in the set-points adapting downwards and then being forced to adapt upwards. Tests using a continuous sine wave input showed that the set-points stabilise with the correct time percentages. This shows that the error encountered is caused solely by the temperature prediction component of the system. This problem of accurately predicting temperature minimums will be present for any method of prediction implemented. The two day buffer of samples provided the best results, increasing speed of reaction over the three day buffer.

Chapter 8 – Conclusions and Recommendations

8.1 Achievement of Objectives

The project specification in Appendix A contains the objectives of this project. The tests carried out have been analysed in regards to the project objectives and the success of the project has been judged in terms of the project specification.

As stated in the specification, the aim of the project was to "develop a controller that will automatically switch aeration fans based on ambient air state, to condition grain in storage during the optimal time." The general aim of the project has been completed. The controller provides automatically switched outputs, for interfacing with aeration motor controls. The optimal times of coolest air occur during the temperature troughs. The results show that the system does operate the outputs during these periods.

Research into control methods suitable for the control of aerators was successful and the design based on this. The air parameters required to be monitored were investigated and chosen as temperature and relative humidity. From these the wet-bulb temperature is derived.

The PICAXE 18X was selected as the microcontroller and interfaced successfully with other required components. A program capable of automatically controlling aeration fans, by operating them when appropriate, was completed. The final prototype was constructed and tested successfully.

The major outcomes of the project are outlined below:

- The system provides a user interface for the operator to allow monitoring of the device. The time can be set with the two keys on the device. The display provides a readout of the air parameters being monitored by the

device. The set-points temperatures for the Normal and Rapid modes can be checked.

- The system measures temperature and relative humidity. These are used to calculate the wet-bulb temperature.

- The system calculates Normal and Rapid mode set-points from previously stored temperature data. The Normal and Rapid outputs are switched on and off as necessary.

- Remote monitoring of the device was not investigated due to time limitations.

Overall the system has performed well. The system was designed and integrated into a fully functional prototype. The initial research was paramount to this success and the timeline followed to ensure completion.

8.2 Recommendations for Further Work

There are areas of further work that could enhance the operation of the system. Some recommendations are outlined below:

1) Noise Immunity

Adding a CRC check to the sensor interface would ensure that measurements were not corrupted during transmission. By improving code efficiency, memory could be released for this function. The noise immunity could also be improved by using shielded cable for connection to the sensor.

2) Manufacture the device on the designed PCB

The designed PCB could be constructed and the components fitted. This would then be tested to verify correct operation.

3) Add Power Management Algorithm

It was originally envisaged that a power optimisation algorithm would be present in the system. When active this would alter the running period to utilise the less expensive electricity tariff times. Daily cooling efficiency would be sacrificed as necessary to minimise running costs. The system could then catch up on lost hours during the weekend off-peak period.

4) Investigate remote monitoring

Remote monitoring of the system would allow its operation to be verified and any malfunctions recognised. This could also extend to the device remotely operating aerators from a distance. Such a feature would allow the device to operate fans in a remote location, rather than purchasing further aeration controllers, thus minimising cost.

These recommendations provide a platform for further work to be undertaken in this area.

References

Bureau of Meteorology 2006, *Climate Glossary*, Commonwealth of Australia, viewed Apr 2006, http://www.bom.gov.au/climate/glossary/wetbulb.shtml

CSIRO, Stored Grain Research Laboratory, *Time Proportioning Controller*, viewed 20 Apr 2006, http://sgrl.csiro.au/storage/moisture/time_proportioning.html

Dallas Semiconductor, *DS18B20 Programmable Resolution 1-Wire Digital Thermometer*, viewed 9 May 2006, www.maxim-ic.com

Dallas Semiconductor, *DS130764 X 8 Serial Real Time Clock*, viewed 20 May 2006, www.maxim-ic.com

Darby, J 1998, 'Putting grain aeration in order with generalized aeration categories', *Australian Postharvest Technical Conference*, pp 203-4

Darby, J 2000, 'Aeration control developments', *Australian Postharvest Technical Conference*, pp 39-46

Fusae, T 2004, *Grain Storage Aeration & Fumigation*, C RFM, Toowoomba

Kotzur, A 1998, 'Ambient air in-store grain storing: recent Australian experience', *Australian Postharvest Technical Conference*, pp 218-220

Leis, J 2002, *Digital Signal Processing: A MATLAB-based tutorial approach*, Research Studies Press LTD., Baldock

McPhee, J 1998, *Grain Aeration*, Farming Systems Institute, DPI Queensland, viewed 19 Apr 2006, http://agspsrv34.agric.wa.gov.au/ento/publications/p98156.html

Microchip Technology Inc. 2003, *24AA16/24LC16B Datasheet*, DS21703D, viewed 20 May 2006, ww1.microchip.com/downloads/en/devicedoc/21703d.pdf

Microchip Technology Inc. 2005, *PIC16F87/88 Data Sheet*, viewed 20 May 2006, http://ww1.microchip.com/downloads/en/DeviceDoc/30487c.pdf

Motorola, Inc. 1996, *ULN2803 ULN2804*, Revision 1, viewed 20May 2006, www.**datasheet**catalog.com/**datasheet**s_pdf/U/L/N/2/**ULN2803**.shtml

Newman, C 2002, 'Aeration - for preserving grain quality', *Farmnote*, Department of Agriculture, no. 24/2002

Revolution Education 2004, *PICAXE Manual*, viewed 11 May 2006, www.picaxe.co.uk

Revolution Education 2004, *AXE033.pmd,* v4.1, viewed 11 May 2006, www.picaxe.co.uk

SENSIRION AG 2003, *SHT1x / SHT7x Relative Humidity & Temperature Sensor System*, V2.01, viewed 10 June 2006, www.sensirion.com/en/download/humiditysensor/SHT11.htm

SENSIRION AG 2005, *SHTxx Application Note Non-Linearity Compensation*, Revision 1.32, viewed 10 June 2006, www.sensirion.com/humidity

SENSIRION AG 2006, *SHTxx Application Note CRC*, Revision 1.07, viewed 10 October 2006, www.sensirion.com/humidity

STMicroelectronics 2006, *L7800 SERIES POSITIVE VOLTAGE REGULATOR*, viewed 20 May 2006, http://www.st.com

Appendix A – Project Specification

University of Southern Queensland

Faculty of Engineering and Surveying

ENG 4111/4112 Research Project
PROJECT SPECIFICATION

FOR: ANDREW CHARLES

TOPIC: **AMBIENT AIR TEMPERATURE TREND AERATION CONTROLLER.**

SUPERVISOR: Mr. Mark Norman

ENROLMENT: ENG 4111 – S1, D, 2006
ENG 4112 – S2, D, 2006

PROJECT AIM: The project aims to develop a controller that will automatically switch aeration fans based on ambient air state, to condition grain in storage during the optimal time.

PROGRAMME: Issue A, 27 March 2006

1. Research information on aeration control methods in grain storage.
2. Research and select sensors appropriate for ambient air measurement.
3. Research and select microprocessor with other hardware components.
4. Design and simulate software required for control of aeration fans.
6. Construct prototype and evaluate.

As time permits

6. Investigate remote monitoring of controller operation.

AGREED: _____ (student) _____

(Supervisor)

(date)___/___/___

Appendix B – Wet-Bulb Temperature Lookup Table

RELATIVE HUMIDITY TABLES FOR WHIRLING PSYCHROMETER

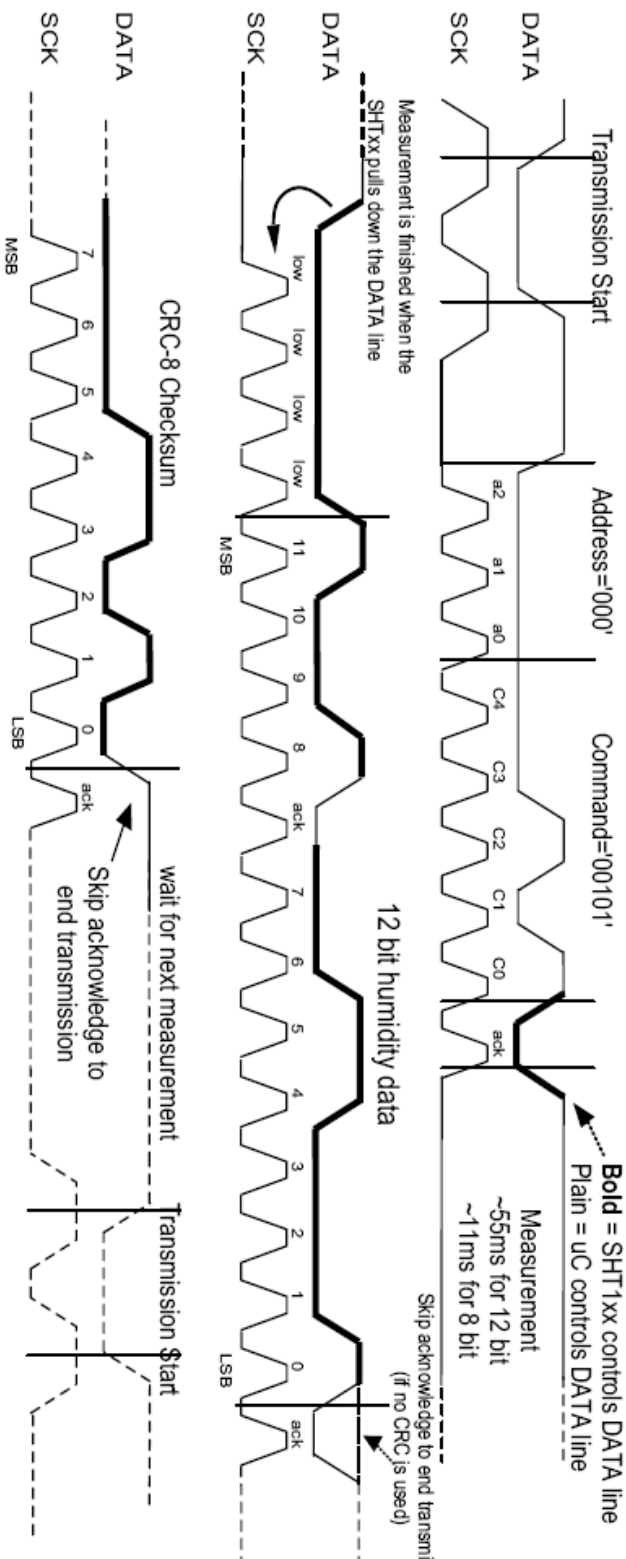
DEPRESSION OF WET BULB 0.5 —10 C.

DRY BULB READINGS — CELSIUS

°C	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	°C
0.5																																																									0.5
1.0																																																											1.0
1.5																																																										1.5
2.0																																																										2.0
2.5																																																										2.5
3.0																																																										3.0
3.5																																																										3.5
4.0																																																										4.0
4.5																																																										4.5
5.0																																																										5.0
5.5																																																										5.5
6.0																																																										6.0
6.5																																																										6.5
7.0																																																										7.0
7.5																																																										7.5
8.0																																																										8.0
8.5																																																										8.5
9.0																																																										9.0
9.5																																																										9.5
10.0																																																										10.0

G. H. ZEAL LTD., 8 DEER PARK ROAD, LONDON SW19 3UU

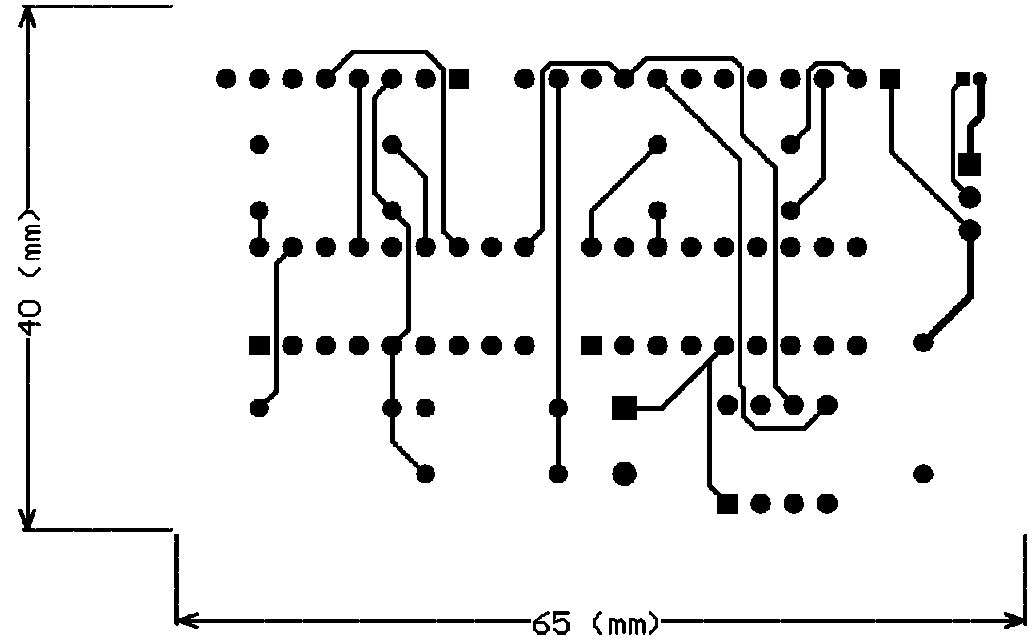
Appendix C – SHT15 Sensor Timing Diagram

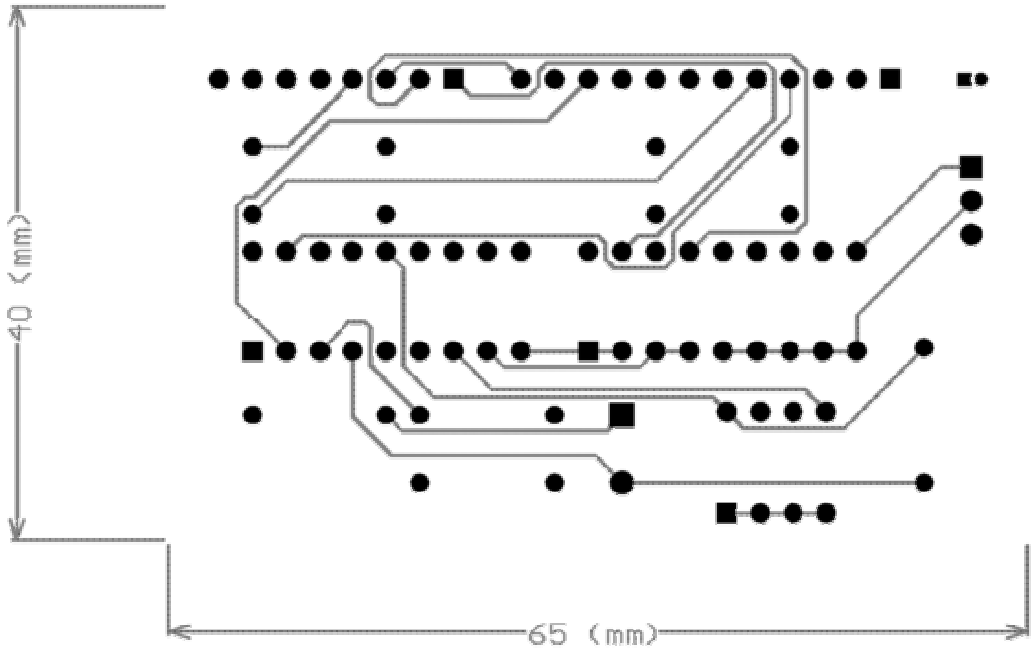

Transmission Start
Address='000'
Command='00101'
DATA
SCK
a2
a1
a0
C4
C3
C2
C1
C0
ack
Bold = SHT1xx controls DATA line
Plain = uC controls DATA line
Measurement
~55ms for 12 bit
~11ms for 8 bit
Skip acknowledge to end transmission
(if no CRC is used)
Measurement is finished when the
SHTxx pulls down the DATA line
low
low
low
low
11
10
9
8
ack
7
6
5
4
3
2
1
0
MSB
LSB
12 bit humidity data
CRC-8 Checksum
wait for next measurement
Transmission Start
Skip acknowledge to
end transmission

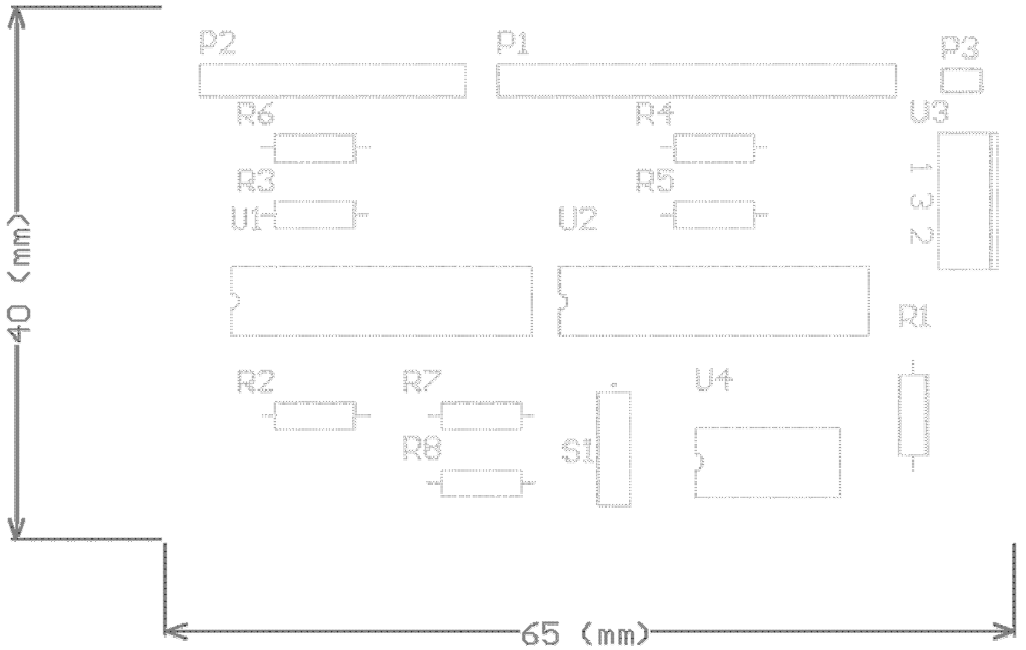
Appendix D – LCD Display Commands

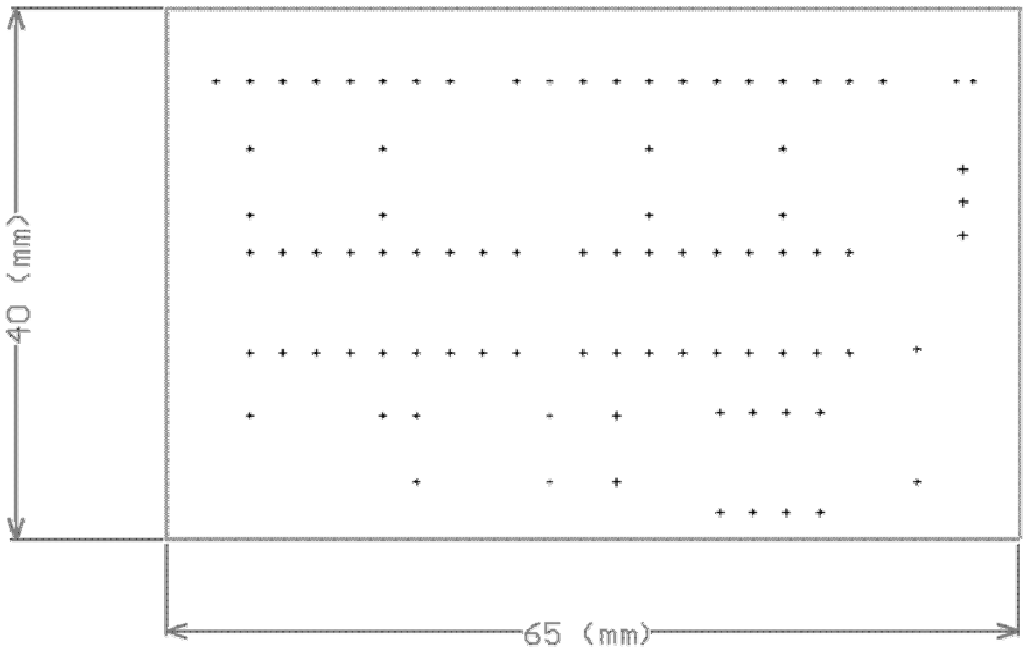
254,1	Clear Display (must be followed by a 'pause 30' command)
254,8	Hide Display
254,12	Restore Display
254,14	Turn on Cursor
254,16	Move Cursor Left
254,20	Move Cursor Right
254,128	Move to line 1, position 1
254, y	Move to line 1, position x (where y = 128 + x)
254,192	Move to line 2, position 1
254, y	Move to line 2, position x (where y = 192 + x)

Appendix E – PCB


40 (mm)
65 (mm)


40 (mm)
65 (mm)


P2
P1
P3
R6
R4
R3
R5
U1
U2
U3
1
3
2
R1
R2
R7
R8
S1
U4
40 (mm)
65 (mm)


40 (mm)
65 (mm)

Appendix F – Software Listing

```
'Written by Andrew Charles
'Student No. 0050009343
'Written with PICAXE Programming Editor

'Aeration Controller Project
'For use with PICAXE 18x
'Memory used: 1852 bytes

'--------------------Outputs--------------------------------
--

symbol Rapid = 2
symbol Normal = 3
symbol Sensclock = 6
symbol Sensout = 7

'--------------------Inputs---------------------------------
--

symbol Mode = pin0
symbol Adjust = pin1
symbol Sensin = pin6

'-------------------Variables-------------------------------
--

symbol DataLow = b0  'Working Byte Low 1
symbol DataHigh = b1 'Working Byte High 1
symbol DataWord = w0 'Working Word 1 = b0 & b1

symbol DataLow2 = b2 'Working Byte Low 2
symbol DataHigh2 = b3      'Working Byte High 2
symbol DataWord2 = w1      'Working Word 2 = b2 & b3

symbol NormalSet = b4      'Normal Set-point Temperature
symbol RapidSet = b5 'Rapid Set-Point Temperature

symbol Day = b6            'Current number of Day, 1-7
symbol Counter = b7  'Universal Loop Counter Variable
symbol WbTemp = b8   'Wet-bulb temperature
symbol Hum = b9            'Relative Humidity
symbol Temp = w5           'Dry-bulb temperature =b10 & b11
symbol Minutes = b12 'Current time minutes
symbol Hour = b13          'Current time hour, 24hr format

'------------------Constants-------------------------------
--

symbol TempMeas = %00000011            'Measure Temp
command
symbol HumMeas = %00000101       'Measure Humidity command
symbol StatRegWrite = %00000110  'Write to Status Register
command
symbol StatRegRead = %00000111   'Read from Status Register
command
symbol Setup = %00000001         'Status Register write,
set precision


symbol EEPROM1 = %10100000       'EEPROM BLOCK I2C
Addresses
symbol EEPROM2 = %10100010
```

```
symbol EEPROM3 = %10100100
symbol EEPROM4 = %10100110
symbol EEPROM5 = %10101000
symbol EEPROM6 = %10101010
symbol EEPROM7 = %10101100
symbol EEPROM8 = %10101110

symbol RTC = %11010000                'Real Time Clock I2C
Address
symbol LCD = $C6                      'LCD I2C Address

'----------Pre-defined Display Strings---------------------
--

EEPROM 0,  ("SEND SERIAL DATA")
EEPROM 16, ("YES           NO")
EEPROM 32, ("MASTER RESET?   ")
EEPROM 48, ("SET CLOCK       ")
EEPROM 64, ("SHOW SET-POINTS?")
EEPROM 80, ("RESET COMPLETE  ")
EEPROM 96, ("    INACTIVE    ")

'_____________________________________________________________
__


'-------------------MAIN PROGRAM---------------------------
--

Main:          'Initial Setup

         pause 2000              'Wait For Peripherals to
Initialize

         'Setup Temp/Hum Sensor
         Datalow = StatRegWrite  'Status Register adress
         gosub TransStart        'Initiate Transmission

         gosub WriteData         'Write data to sensor
         gosub Acklow            'Provide Low Acknowledge
         DataLow = Setup         'Precision data
         gosub WriteData         'Write setup 10/8 bit
accuracy
         gosub Acklow            'Provide Low Acknowledge

         gosub RTCRead           'Read Real Time Clock

         goto SetTrue            'Set initial Parameters
Upon Startup


         'Main Repetative Loop

Repeat:  if Mode = 1 then Settings     'Check Switch, jump
to settings if activated

After:   gosub TempRead    'Read Temperature

         gosub HumRead     'Read Humidity

         gosub RTCRead     'Read Time
```

```
          gosub WbCalc      'Calculate Wet Bulb Temperature


          gosub DispMain    'Display Main Page

          gosub CheckTime   'Check if time for Calculation

          pause 2000        'Pause for period to avoid
heating sensor

          goto Repeat       'Repeat Main Loop

'_____________________________________________________
__
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
'         Menu                                                    '
'User interface menu routines. This is a branch and not a
   '
'subroutine.  Options are View Set-points, Send Serial
Data,'
'Master Reset and Set Time                                        '
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''

Settings:      pause 3000                    'Pause to
elminate accidental press
          if Mode = 1 then SetTrue      'Continue if Mode
button still pressed
          goto repeat                   'Else Exit

SetTrue: i2cslave LCD,i2cslow,i2cbyte  'Set I2C Address to
LCD

          DataLow = 64                  'String 'Show Set-
Points?'
          gosub MenuItem                'Display Menu Item

          if bit0 = 1 then ShowSetPts   'If adjust key
pressed branch

          DataLow = 0                   'String 'Send Serial
Data'
          gosub MenuItem                'Display Menu Item

          if bit0 = 1 then SendData     'If adjust key
pressed branch

MasterRst:     DataLow = 32                   'String
'MASTER RESET?
          gosub MenuItem                'Display Menu Item

          if bit0 = 1 then Reset        'If adjust key
pressed branch

SetTime: gosub DispClear               'Clear LCD
          DataLow = 48                  'String 'Set Clock'
          gosub PrintLine               'Print
          pause 1000

HourSet: gosub TimePrint               'Print Time
```

```
          gosub DayPrint                'Print Day

          gosub Debounce                'Debounce

          if bit0 = 1 then IncHour      'If Adjust pressed
branch
          if bit1 = 1 then Minset       'If Mode pressed
branch
          goto Writetime                'Else Write time to
RTC

IncHour:  Hour = Hour + 1               'Increment Hour
          Hour = Hour//24               'If 24 then set to
zero
          pause 200
          goto Hourset                  'Update Display

MinSet:   pause 500

Minset2:  gosub TimePrint               'Print Time

          gosub Debounce                'Debounce

          if bit0 = 1 then IncMin       'If Adjust pressed
branch
          if bit1 = 1 then SetDay       'If Adjust pressed
branch
          goto Writetime                'Else Write time to
RTC

IncMin:   Minutes = Minutes + 1         'Increment Minute
          Minutes = Minutes//60         'If 60 then set to
zero
          pause 200
          goto MinSet2                  'Update Display

SetDay:   pause 500

SetDay2:  gosub DayPrint                'Print Day
          pause 10

          gosub Debounce                'Debounce

          if bit0 = 1 then IncDay       'If Adjust pressed
branch
          goto WriteTime                'Else Write time to
RTC

IncDay:   Day = Day + 1                 'Increment Day
          Day = Day//8                  'If 8 then set to
zero
          if Day = 0 then IncDay        'If 0 then set to 1
          pause 500
          goto SetDay2                  'Update Display

WriteTime:     i2cslave RTC,i2cslow,i2cbyte  'Set I2C
Address to RTC
          DataLow = Minutes
          gosub BinarytoBCD             'Convert minutes to
BCD
          writei2c 1,(DataLow)          'Write Minutes to
RTC
```

```
         DataLow = Hour
         gosub BinarytoBCD              'Convert hour to BCD
         writei2c 2,(DataLow,Day)       'Write Hour and Day
to RTC

         goto ExitToMain                'Branch to Exit


Reset:   'Master Reset, Clear Memory Buffer
         i2cslave EEPROM6, i2cslow, i2cbyte  'Set I2C
Address to EEPROM6
         DataLow = 15                        'Length of 16
bytes in block
         gosub Clearing                      'Clear buffer
         i2cslave EEPROM7, i2cslow, i2cbyte  'Set I2C
Address to EEPROM7
         DataLow = 1                         'Length of 16
bytes in block
         gosub Clearing                      'Clear buffer
         i2cslave RTC, i2cslow, i2cbyte      'Set I2C
Address to RTC
         writei2c $08, (1, 0, 0)             'Reset Flags
in RTC

         i2cslave LCD,i2cslow,i2cbyte  'Set I2C Address to
LCD
         gosub DispClear               'Clear Display
         DataLow = 80                  'String 'RESET
COMPLETE'
         gosub PrintLine               'Print
         pause 2000

ExitToMain:    goto After                    'Exit Menu,
return to Main loop


ShowSetPts:    'Display Set-Points on LCD
         writei2c 0,(254,1,255)        'Clear Display
         pause 30
         writei2c 0,(254,128,255)      'Position Top Left
         pause 10
         DataLow2 = NormalSet - 40     'Change format
         writei2c 0,("Normal = ",255)  'Print 'Normal = '
         pause 10
         if NormalSet >= 40 then skipneg3    'Account for
possible negative
         writei2c 0,("-",255)
         pause 10
         DataLow2 = 40 - NormalSet
skipneg3:      gosub calcnum
         writei2c 0,(DataHigh,DataLow,%11011111,"C",255)
'Print Set-Point
         pause 10

         writei2c 0,(254,192,255)      'Position Bottom
Left
         pause 10

         DataLow2 = RapidSet - 40      'Change format
         writei2c 0,("Rapid = ",255)   'Print 'Rapid = '
         pause 10
```

```
         if RapidSet >= 40 then skipneg4     'Account for
possible negative
         writei2c 0,("-",255)
         pause 10
         DataLow2 = 40 - RapidSet
skipneg4:      gosub calcnum
         writei2c 0,(DataHigh,DataLow,%11011111,"C",255)
   'Print Set-Point
         pause 4000                    'Pause to allow
viewing
         goto ExitToMain               'Branch to Exit

SendData:      'Send Serial Data via programming cable

         i2cslave EEPROM6, i2cslow, i2cbyte  'Block 6

         for Counter = 0 to 255
         readi2c Counter, (DataLow)
         sertxd(#DataLow,$0D)
         next Counter

         i2cslave EEPROM7, i2cslow, i2cbyte  'Block 7

         for Counter = 0 to 31
         readi2c Counter, (DataLow)
         sertxd(#DataLow,$0D)
         next Counter
         goto ExitToMain               'Branch to Exit

'_____________________________________________________________
__


'**********************Subroutines************************
**

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
'                    Clearing                                 '
'Clears the EEPROM memory in a block of 16 bytes              '
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''

Clearing:      for Counter = 0 to DataLow    'Loop for
Datlow blocks
         DataHigh = Counter * 16       'Start Address
16*DataLow
         'Write 255 into block, this signifies empty
         writei2c
DataHigh,(255,255,255,255,255,255,255,255,255,255,255,255,2
55,255,255,255)
         pause 10
         next Counter
         return

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
'                    TimePrint                                '
'Prints the Time stored in the PICAXE RAM to the LCD
   '
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
```

```
TimePrint:     writei2c 0,(254,192,255)      'Position Line
2
         pause 10
         DataLow2 = Hour
         gosub calcnum
         writei2c 0,(DataHigh,DataLow,":",255)     'Print
Hours
         pause 10
         DataLow2 = Minutes
         gosub calcnum
         writei2c 0,(DataHigh,DataLow,255)         'Print
Minutes
         pause 10
         return

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
'                    DayPrint                                      '
'Prints the Day stored in the PICAXE RAM to the LCD
   '
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''

DayPrint:      writei2c 0,(254,200,255)      'Position Line
2
         pause 10
         'Select day
         if Day = 1 then Mon
         if Day = 2 then Tue
         if Day = 3 then Wed
         if Day = 4 then Thu
         if Day = 5 then Fri
         if Day = 6 then Sat
         if Day = 7 then Sun

         'Print Day to LCD
Mon:           writei2c 0,("Mon",255)
         return

Tue:           writei2c 0,("Tue",255)
         return

Wed:           writei2c 0,("Wed",255)
         return

Thu:           writei2c 0,("Thu",255)
         return

Fri:           writei2c 0,("Fri",255)
         return

Sat:           writei2c 0,("Sat",255)
         return

Sun:           writei2c 0,("Sun",255)
         return

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
'                    MenuItem                                      '
'Prints the String within DataLow to first line of LCD.  '
```

```
'Prints Yes No on second line, checks for key presses,
exits'
'when pressed or time out                                    '
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''

MenuItem:      gosub DispClear              'Clear Display

         gosub PrintLine               'Print String in
DataLow
         pause 1000
         writei2c 0,(254,192,255)      'Shift Cursor to
Line 2
         DataLow = 16                  'Display String 'Yes
No'
         gosub PrintLine               'Print
         pause 10

         gosub Debounce                'Debounce
         return

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
'                    Debounce                                '
'Debounces the two input keys.  Sets flag bit0 for adjust,
   '
'bit1 for Mode.  Exist after approximately 12 seconds if no
   '
'key pressed.                                                '
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''

Debounce:      Datalow = 0                  'Clear flags
         for counter = 1 to 255
         if Adjust = 1 then Debounce1  'If adjust pressed
branch
         if Mode = 1 then Debounce2    'If Mode pressed
branch
         pause 50                      'Pause for timeout
         next counter
         goto ExitDbnce                'Exit if no key
pressed

Debounce1:     pause 10                     'Debounce
Adjust
         if Adjust = 1 then Debounce3
         goto Debounce

Debounce2:     pause 10                     'Debounce Mode
         if Mode = 1 then Debounce4
         goto Debounce                 'Branch to exit

Debounce3:     bit0 = 1                     'Set Adjust
key flag
         goto ExitDbnce

Debounce4:     bit1 = 1                     'Set Mode key
flag
ExitDbnce:     return

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
```

```
'                    PrintLine                                  '
'Print EEPROM string starting at index stored in datalow to
'
'LCD display                                                     '
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''

PrintLine:     DataHigh2 = DataLow + 15            'String
Length
         for Counter = DataLow to DataHigh2
         read Counter, DataHigh              'Read
character from EEPROM
         writei2c 0, (DataHigh,255)
         next Counter
         return

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
'                    Check Time                                 '
'Performs a read of RAM from RTC and determines if a new
'
'Measurement is required                                        '
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''

CheckTime:     i2cslave RTC,i2cslow,i2cbyte  'Set I2C
address to RTC

         'Read minutes of last measurement, Circular Buffer
Position and EEPROM Bank
         readi2c $08, (DataLow2, DataHigh, DataLow)

         'Modulus of Minutes = 0 if quarter hour interval
         DataHigh2 = Minutes // 15

         'If a quarter hour interval and not same interval
as last measurement then Measure
         if DataHigh2 = 0 and DataLow2 != Minutes then
Measure

         if bit1 = 1 then ExitCkTime   'If buffer complete
flag is active then exit
         i2cslave LCD,i2cslow,i2cbyte  'Set I2C address to
LCD
         pause 2000
         gosub DispClear               'Clear Display
         DataLow = 96                  'String 'INACTIVE'
         gosub PrintLine               'Print

ExitCkTime:    return

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
'                    Measure                                    '
'Stores Measurement in EEPROM Bank, may call
   '
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''

Measure: if bit0 = 1 then UseBlock7          'Check flag,
choose EEPROM Block
```

```
         i2cslave EEPROM6,i2cslow,i2cbyte    'Set I2C
address to EEPROM Block 6

BlckChosen:    DataHigh2 = WbTemp - 20             'Adjust
WbTemp to start at -20 degrees C

         writei2c DataHigh,(DataHigh2)       'Store Current
Measurement in EEPROM
         pause 10

         i2cslave RTC, i2cslow, i2cbyte      'Set I2C
address to RTC

         'If next measurement is in second block change
block
         if DataHigh = 255 and bit0 = 0 then BlockChng1

         'If next measurement is in first block change
block
         if DataHigh = 31 and bit0 = 1 then BlockChng2

         DataHigh = DataHigh + 1             'Else
increment location within current block

         goto WriteLoc                       'Branch to
Write index location to LCD RAM

UseBlock7:     i2cslave EEPROM7,i2cslow,i2cbyte    'Set I2C
address to EEPROM Block 7
         goto BlckChosen                     'Branch to
block operations

BlockChng1:    bit0 = 1                            'Set
flag to Block 6
         goto Changed

BlockChng2:    bit0 = 0                            'Set
flag to block 7
         bit1 = 1                            'Set buffer
complete flag

Changed: DataHigh = 0                        'Location zero
in block

WriteLoc:      'Write current measurement minutes, next
measurement location and block
         writei2c $08, (Minutes, DataHigh, DataLow)

         if bit1 = 0 then ExitMeas           'If Buffer not
complete then exit

         'Else

         gosub CalcOccur1  'Calculate Occurance of
Temperatures
         gosub CumProb     'Calculate Cumulative
Probability
         gosub ChkSetPts   'Activate outputs if necessary

ExitMeas:      return

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
```

```
''
'                    CalcProbs                              '
'Calculates number of occurences for each temperature in
the'
'range of -20 to 49 degrees C, stores histogram in RAM   '
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''

CalcOccur1:    for Counter = 80 to 127
         poke Counter, 0                      'Clear Ram
block 1
         next Counter

         for Counter = 192 to 239
         poke Counter, 0                      'Clear Ram
block 2
         next Counter

         i2cslave EEPROM6,i2cslow,i2cbyte     'Set I2C
address to EEPROM6
         for Counter = 0 to 255               'Step through
block
         gosub RamProb                        'Increment
histogram

         next Counter

CalcOccur2:    i2cslave EEPROM7, i2cslow, i2cbyte  'Set I2C
address to EEPROM7
         for Counter = 0 to 31                'Step through
block
         gosub RamProb                        'Increment
histogram

         next Counter
         return

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
'                    RamProb                                '
'Reads data from EEPROM and increments number of occurances
'
'of that temperature in the range of -20 to 49 degrees C '
                          '
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''

RamProb: readi2c Counter, (DataLow)           'Read in
Measurement Data
         if DataLow > 47 then Ram2            'Store in 2nd
block of Ram
         DataHigh = 80 + DataLow              'Adjust to
suit Ram location
         peek DataHigh, DataLow2              'Peek location
DataHigh into DataLow2, current number of occurances
         DataLow2 = DataLow2 + 1              'Increment
value
         poke DataHigh, DataLow2              'Poke
incremented value back into ram
         goto ExitRamp

Ram2:          DataHigh = 192 + DataLow -48        'Adjust
```

```
to suit Ram location
         peek DataHigh, DataLow2             'Peek location
DataHigh into DataLow2, current number of occurances
         DataLow2 = DataLow2 + 1             'Increment
value
         poke DataHigh, DataLow2             'Poke
incremented value back into ra
ExitRamP:      return

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
'        Calc Cumulative Probability                    '
'Finds cumulative probability of data to determine Normal
   '
'and Rapid set-point temperatures
   '
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''

CumProb: DataWord2 = 0                       'Clear 16 bit
word

         'Calculate Normal Set-Point

         for Counter = 80 to 127             'Step through
first block of Ram
         peek Counter, DataLow               'Peek location
Counter into DataLow, number of occurances
         DataWord2 = DataWord2 + DataLow     'Increment
cumulative density to include current temperature
         if DataWord2 >= 41 then SetNormal   'If time
percentage reached branch
         next Counter

         for Counter = 192 to 204            'Step through
second block of Ram
         peek Counter, DataLow               'Peek location
Counter into DataLow, number of occurances
         DataWord2 = DataWord2 + DataLow     'Increment
cumulative density to include current temperature
         if DataWord2 >= 41 then SetNormal   'If time
percentage reached branch
         next Counter

         end                                 'Error if this
point reached

SetNormal:     if Counter >128 then SetNorm1       'Check
which block of ram value was in use

         NormalSet = Counter - 80 + 20       'Adjust to
range
         goto nextpt

SetNorm1:      NormalSet = Counter - 192 + 20 + 48 'Adjust
to range

nextpt:  DataWord2 = 0                       'Clear 16 bit
word

         'Calculate Rapid Set-Point
```

```
         for Counter = 80 to 127             'Step through
first block of Ram
         peek Counter, DataLow               'Peek location
Counter into DataLow, number of occurances
         DataWord2 = DataWord2 + DataLow     'Increment
cumulative density to include current temperature
         if DataWord2 > 144 then SetRapid    'If time
percentage reached branch
         next Counter

         for Counter = 192 to 204            'Step through
second block of Ram
         peek Counter, DataLow               'Peek location
Counter into DataLow, number of occurances
         DataWord2 = DataWord2 + DataLow     'Increment
cumulative density to include current temperature
         if DataWord2 > 144 then SetRapid    'If time
percentage reached branch
         next Counter

         end                                 'Error if this
point reached

SetRapid:      if Counter >128 then SetRap1        'Check
which block of ram value was in use

         RapidSet = Counter - 80 + 20        'Adjust to
range
         goto ExitCumP                       'Branch to
exit

SetRap1: RapidSet = Counter - 192 + 20 + 48  'Adjust to
range
ExitCumP:      return

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
'                    CheckSetPts                          '
'Checks current temperature against set-points and
activates'
'outputs as required                                      '
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''

ChkSetPts:
         if WbTemp > RapidSet then Case1     'If
Temperature is above Rapid
         if WbTemp > NormalSet then Case2    'If
Temperature is above Normal but below Rapid

         'Else temperature is below Normal and Rapid
         high Rapid
         high Normal                         'Activate
Normal and Rapid
         goto FinChk


Case1:   low Rapid                           'Activate None
         low Normal
         goto FinChk

Case2:   high Rapid                          'Activate
```

```
Rapid
         low Normal

FinChk:  return

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
'                    DispClear                                          '
'Clears both lines of LCD Display and returns cursor to top
   '
'left
   '
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''

DispClear:     writei2c 0,(254,1,255)        'Clear Display
         pause 30
         writei2c 0,(254,128,255)      'Position Top Left
         pause 10
         return

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
'              Display Main Page                                        '
'Displays the main idle page.  Contains Relative Humidity,
   '
'Wet-bulb temp, Dry-bulb temp and time                                   '
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''

DispMain:      i2cslave LCD,i2cslow,i2cbyte  'Set I2C
address to LCD
         writei2c 0,(254,128,255)      'Position Top left
         pause 10

         DataLow2 = Hum                'Load Relative
Humidity
         gosub Calcnum                 'Convert to Ascii
values

         'Write static data
         writei2c 0,("RH ",DataHigh,DataLow,"%
DB",%11011111,"C WB",%11011111,"C",255)
         pause 10
         writei2c 0,(254,192,255)      'Next Line
         pause 10

         DataLow2 = Hour               'Load Hours
         gosub Calcnum                 'Convert to Ascii
values

         'Write Hours
         writei2c 0,(DataHigh,DataLow,":",255)
         pause 10

         DataLow2 = Minutes            'Load Minutes
         gosub Calcnum                 'Convert to Ascii
values

         'Write minutes
         writei2c 0,(DataHigh,DataLow,"  ",255)
         pause 10
```

```
          'Prepare Dry-bulb temp for display
          DataWord = Temp - 400          'Change form
          if Temp >= 400 then SkipNeg    'Check negative
          writei2c 0,("-",255)           'Display negative
          pause 10

          DataWord = 400 - Temp          'Change because
negative

          'Convert into places for display
SkipNeg:  DataHigh2 = DataWord /100 + $30
          DataLow2 = DataWord //100
          DataHigh = Datalow2//10 + $30
          Datalow2 = Datalow2/10 + $30

          'Write dry-bulb temp to LCD
          writei2c 0,(DataHigh2,DataLow2,".",DataHigh,"
",255)
          pause 10

          'Prepare wet-bulb temp for display
          DataLow2 = WbTemp - 40         'Change form
          if WbTemp >= 40 then SkipNeg2 'Check negative
          writei2c 0,("-",255)pause 10   'Display negative

          DataLow2 = 40 - WbTemp         'Change because
negative

          'Convert into places for display
SkipNeg2:       gosub Calcnum                 'Convert to
Ascii values

          'Write wet-bulb temp to LCD
          writei2c 0,(DataHigh,DataLow,"  ",255)
          pause 10
          return

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
'                    Calcnum                                          '
'Converts binary value to charater to display on LCD, value
   '
'to convert is in DataLow2 and outputs in DataHigh and   '
'DataLow                                                             '
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''

Calcnum:  DataHigh = DataLow2/10 + $30         'Find tens
place

          DataLow = DataLow2//10 + $30         'Find ones
place
          return

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
'         Perform Temprature Measurement                              '
'Read temperature from sensor and store in Temp variable '
'12 bit value is multiplied by 10 and offset by +400
   '
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
```

```
''

TempRead:      DataLow = TempMeas      'Load temperature
measure instruction
         gosub Transstart        'Transmission start
sequence
         gosub WriteData         'Write instruction
         gosub Acklow            'Provide low acknowledge
         pause 100               'Wait for Measurement
         DataWord = 0            'Clear variable for
measurement
         gosub ReadData          'Read first byte
         gosub Acklow            'Provide low acknowledge
         gosub ReadData          'Read second byte
         gosub Ackhigh           'Provide high acknowledge

         'Convert word into temperature value
         'Temp * 10, offset by +400
         DataWord = DataWord * 10 /25
         Temp = DataWord         'Store value
         return

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
'        Perform Humidity Measurement                        '
'Read humidity from sensor and store in Hum variable, 8 bit
   '
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''

HumRead: DataLow = HumMeas       'Load humidity measure
instruction
         gosub TransStart        'Transmission start
sequence
         gosub WriteData         'Write instruction
         gosub Acklow            'Provide low acknowledge
         pause 50                'Wait for Measurement
         DataWord = 0            'Clear variable for
measurement
         gosub ReadData          'Read first byte, empty
         gosub Acklow            'Provide low acknowledge
         gosub ReadData          'Read second byte
         gosub Ackhigh           'Provide high acknowledge

         'Convert into humidity value
         if DataLow <= 107 then less
         DataWord = DataLow*111 + 2893       'First linear
range
         goto finish
less:          DataWord = DataLow*143              'Second
linear range
         if DataWord >= 512 then subtract    'Check for
underflow
         DataWord = 512
subtract:      DataWord = DataWord - 512
finish:  DataWord = DataWord/256
         Hum = DataLow           'Store Value
         return

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
'        Calculate Wb Temp                                   '
```

```
'Use lookup table to find wet bulb temperature from dry-
bulb'
'and relative humidity, offset by +40                    '
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
WbCalc:  'Determine block of table

         if Temp < 470 then Bank1      'up to 7 degrees
         if Temp < 590 then Bank2      'up to 19 degrees
         if Temp < 710 then Bank3      'up to 31 degrees
         if Temp < 830 then Bank4      'up to 43 degrees
         goto Bank5                    'else up to 50
degrees

         'Adjust for index into table
Bank1:   i2cslave EEPROM1, i2cslow, i2cbyte
         DataLow = Temp - 350
         goto WbWorking

Bank2:   i2cslave EEPROM2, i2cslow, i2cbyte
         DataLow = Temp - 470
         goto WbWorking

Bank3:   i2cslave EEPROM3, i2cslow, i2cbyte
         DataLow = Temp - 590
         goto WbWorking

Bank4:   i2cslave EEPROM4, i2cslow, i2cbyte
         DataLow = Temp - 710
         goto WbWorking

Bank5:   i2cslave EEPROM5, i2cslow, i2cbyte
         DataLow = Temp - 830
         goto WbWorking

         'Find index into section of table
WbWorking:     DataLow = DataLow / 10 * 20   'Table index
         DataHigh = DataLow + 19       'Find end of column
         for Counter = DataLow to DataHigh
         readi2c Counter,(DataLow2)    'Read value feom
table
         if DataLow2 < Hum then WbDone 'If value is less
than humidty then branch
         next Counter

         'Calculate depression from table index when exited
WbDone:  WbTemp = Counter - DataLow    'Find index
         DataWord2 = WbTemp * 5        'Temp difference in
half degree times 10
         WbTemp = Temp - DataWord2 /10 'Subtract depression

         'Clip range from -20 to 49 degrees C
         if WbTemp < 20 then TooLow
         if WbTemp > 89 then TooHigh
         goto ExitWbCalc

TooLow:  WbTemp = 20
         goto ExitWbCalc

TooHigh: WbTemp = 89
ExitWbCalc:    return
```

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
'        Convert BCD to Binary                                          '
'Uses BCD value in DataLow and converts to binary, returns
'
'result in DataLow                                                      '
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''

BCDtoBinary:
         DataHigh = DataLow & %11110000      'Remove lower
4 bits
         DataHigh = DataHigh/16 * 10         'Tens place
         DataLow = DataLow & %00001111       'Remove upper
4 bits, ones place
         DataLow = DataLow + DataHigh        'Add together
         return

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
'        Convert Binary to BCD                                          '
'Uses Binary value in DataLow and converts to BCD, returns
'
'result in DataLow                                                      '
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''

BinarytoBCD:
         DataHigh = DataLow/10               'Tens place
         DataHigh = DataHigh * 16            'Shift bits up
         DataLow = DataLow//10               'Ones place
         Datalow = DataLow + DataHigh        'Combine
         return

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
'                    RTCread                                            '
'Read Minutes and Hours from Real Time Clock, uses                      '
'BCDtoBinary to convert to binary values
   '
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''

RTCread: i2cslave RTC, i2cslow, i2cbyte      'RTC i2cslave
setup
         readi2c 1, (Minutes, Hour, Day)     'Read min,
hour
         DataLow = Minutes                   'Convert
Minutes
         gosub BCDtoBinary
         Minutes = DataLow                   'Store Minutes
         DataLow = Hour                      'Convert Hour
         gosub BCDtoBinary
         Hour = DataLow                      'Store Hour
         return

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
'                    TransStart                                         '
'Sensor Transmission Start Sequence, used to begin
   '
'communication                                                          '
```

```
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
TransStart:    low sensclock           'START sequence
         high sensout
         high sensclock
         low sensout
         low sensclock
         high sensclock
         high sensout
         low sensclock
         return

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
'                    WriteData                                  '
'Write Data byte in DataLow to Sensor                           '
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''

WriteData:     for Counter = 1 to 8
         if bit7 = 1 then Write1
         low sensout                          'Output a low
         goto Writing
Write1:  high sensout                         'Output a high

Writing: pulsout sensclock,10                 'Pulse clock
         DataLow = DataLow*2                  'Shift to next
bit
         next Counter
         return

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
'                    AckHigh                                    '
'High Acknowledge to Sensor
   '
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
Ackhigh: high sensout                         'Output High
         pulsout sensclock,10                 'Pulse clock
         return

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
'                    AckLow                                     '
'Low Acknowledge to Sensor                                      '
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
Acklow:  low sensout
         pulsout sensclock,10
         return

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
'                    ReadData
'Read Byte from Sensor into DataLow
   '
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
ReadData:      high sensout                        'Output
pullup high
         for Counter = 1 to 8
```

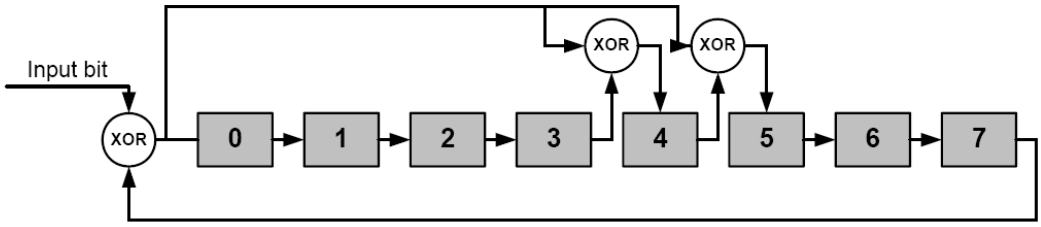
```
        high sensclock                      'Clock high
        DataWord = DataWord*2
        bit0 = sensin
        low sensclock                       'Clock low
        next Counter
        return

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
''
```

Appendix G – CRC Data

2.1 Bitwise

With the bitwise method, the receiver copies the structure of the CRC generator in hard- or software.
An algorithm to calculate this could look like this:

1) Initialise CRC Register to low nibble of status register (reversed ($s_0s_1s_2s_3$'0000))
2) Compare each (transmitted and received) bit with bit 7
3) If the same: shift CRC register, bit0='0'
   else: shift CRC register and then invert bit4 and bit5, bit0='1' (see figure 1)
4) receive new bit and go to 2)
5) The CRC value retrieved from the SHTxx must be reversed (bit 0 = bit 7, bit 1=bit 6 … bit 7 = bit 0) and can then be compared to the final CRC value.(2)



2.2 Bytewise

With this implementation the CRC data is stored in a 256 byte lookup table.
Perform the following operations:

1. Initialize the CRC register with the value of the lower nibble of the value of the status register (reversed ($s_0s_1s_2s_3$'0000)). (default '00000000' = 0)
2. XOR each (transmitted and received) byte with the previous CRC value.
   The result is the new byte that you need to calculate the CRC value from.
3. Use this value as the index to the table to obtain the new CRC value.
4. Repeat from 2.) until you have passed all bytes through the process.
5. The last byte retrieved from the table is the final CRC value.
6. The CRC value retrieved from the SHTxx must be reversed (bit 0 = bit 7, bit 1=bit 6 … bit 7 = bit 0) and can then be compared to the final CRC value.(2)

2.2.1 256 byte CRC Lookup table

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
0	49	98	83	196	245	166	151	185	136	219	234	125	76	31	46	67	114	33	16	135	182	229	212	250	203	152	169	62	15	92	109
32	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**	**48**	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**	**61**	**62**	**63**
134	183	228	213	66	115	32	17	63	14	93	108	251	202	153	168	197	244	167	150	1	48	99	82	124	77	30	47	184	137	218	235
64	**65**	**66**	**67**	**68**	**69**	**70**	**71**	**72**	**73**	**74**	**75**	**76**	**77**	**78**	**79**	**80**	**81**	**82**	**83**	**84**	**85**	**86**	**87**	**88**	**89**	**90**	**91**	**92**	**93**	**94**	**95**
61	12	95	110	249	200	155	170	132	181	230	215	64	113	34	19	126	79	28	45	186	139	216	233	199	246	165	148	3	50	97	80
96	**97**	**98**	**99**	**100**	**101**	**102**	**103**	**104**	**105**	**106**	**107**	**108**	**109**	**110**	**111**	**112**	**113**	**114**	**115**	**116**	**117**	**118**	**119**	**120**	**121**	**122**	**123**	**124**	**125**	**126**	**127**
187	138	217	232	127	78	29	44	2	51	96	81	198	247	164	149	248	201	154	171	60	13	94	111	65	112	35	18	133	180	231	214
128	**129**	**130**	**131**	**132**	**133**	**134**	**135**	**136**	**137**	**138**	**139**	**140**	**141**	**142**	**143**	**144**	**145**	**146**	**147**	**148**	**149**	**150**	**151**	**152**	**153**	**154**	**155**	**156**	**157**	**158**	**159**
122	75	24	41	190	143	220	237	195	242	161	144	7	54	101	84	57	8	91	106	253	204	159	174	128	177	226	211	68	117	38	23
160	**161**	**162**	**163**	**164**	**165**	**166**	**167**	**168**	**169**	**170**	**171**	**172**	**173**	**174**	**175**	**176**	**177**	**178**	**179**	**180**	**181**	**182**	**183**	**184**	**185**	**186**	**187**	**188**	**189**	**190**	**191**
252	205	158	175	56	9	90	107	69	116	39	22	129	176	227	210	191	142	221	236	123	74	25	40	6	55	100	85	194	243	160	145
192	**193**	**194**	**195**	**196**	**197**	**198**	**199**	**200**	**201**	**202**	**203**	**204**	**205**	**206**	**207**	**208**	**209**	**210**	**211**	**212**	**213**	**214**	**215**	**216**	**217**	**218**	**219**	**220**	**221**	**222**	**223**
71	118	37	20	131	178	225	208	254	207	156	173	58	11	88	105	4	53	102	87	192	241	162	147	189	140	223	238	121	72	27	42
224	**225**	**226**	**227**	**228**	**229**	**230**	**231**	**232**	**233**	**234**	**235**	**236**	**237**	**238**	**239**	**240**	**241**	**242**	**243**	**244**	**245**	**246**	**247**	**248**	**249**	**250**	**251**	**252**	**253**	**254**	**255**
193	240	163	146	5	52	103	86	120	73	26	43	188	141	222	239	130	179	224	209	70	119	36	21	59	10	89	104	255	206	157	172

Appendix H – Raw Temperature Sample Data

41	35	40	35	39	35	30	34
41	35	40	35	38	35	30	34
41	36	40	35	38	36	30	34
41	36	39	35	38	36	30	35
41	36	39	35	38	36	30	35
41	36	39	35	38	36	30	35
40	36	38	35	38	36	30	35
41	36	38	35	37	36	30	34
43	35	37	35	37	37	29	36
42	35	37	36	37	36	30	36
42	35	37	35	37	36	30	35
42	35	37	36	37	36	30	36
41	35	37	35	37	36	30	35
43	35	37	36	37	36	30	35
43	35	37	36	37	37	30	35
40	35	37	36	36	36	30	34
40	35	37	36	36	36	30	34
40	35	37	37	36	36	30	34
39	35	37	37	36	36	30	33
38	35	37	36	36	37	30	34
37	35	36	37	36	36	30	33
37	36	36	38	36	36	30	33
37	35	36	38	36	36	30	33
36	36	36	38	35	36	30	33
36	36	36	39	35	37	30	33
35	36	36	38	35	36	30	33
36	36	36	38	34	36	30	32
35	36	36	38	34	36	30	33
35	37	36	39	35	36	30	32
35	36	36	39	35	35	29	32
35	37	35	38	35	35	29	32
35	37	36	39	34	35	29	32
35	38	35	39	34	35	29	31
35	37	35	38	34	35	29	31
35	38	35	38	34	34	29	31
35	38	35	38	34	34	29	31
35	38	35	40	34	34	30	31
35	38	35	38	34	34	31	31
38	38	35	38	34	34	31	31
37	38	35	38	34	34	31	31
37	38	35	39	34	34	32	30
37	39	35	38	34	33	33	30
37	39	35	38	35	33	35	30
37	39	35	38	34	32	35	30
36	39	35	40	34	32	36	30
36	39	35	40	34	32	33	30
36	39	35	38	34	32	33	30
36	40	35	38	34	32	33	30
36	40	35	38	33	32	33	30
36	40	35	37	34	31	34	30
36	40	35	38	34	31	34	30
35	40	35	38	34	31	34	30
35	40	35	38	35	30	33	29
36	40	35	38	35	30	35	29

29	37	31	36	33	36	37	36
29	36	30	35	33	36	38	36
29	35	30	35	33	36	37	36
29	36	30	34	33	36	37	36
29	35	29	33	33	36	37	35
28	35	29	33	33	36	38	35
29	35	29	32	34	35	39	35
28	34	29	31	35	35	39	35
28	34	29	32	34	35	40	35
28	34	30	31	34	35	40	35
28	34	30	31	34	35	38	35
28	33	31	32	35	36	39	35
28	33	32	33	36	35	39	34
28	33	32	34	36	35	38	34
28	32	33	34	36	35	38	34
28	32	33	34	37	35	37	34
28	32	34	34	39	35	37	34
27	32	34	34	41	35	37	34
27	32	34	34	39	35	37	34
27	32	34	33	39	35	37	34
27	32	35	34	39	36	37	34
27	32	35	33	40	35	37	34
27	32	35	33	39	36	37	34
26	32	36	33	39	36	37	34
27	32	36	33	40	36	37	34
28	32	36	33	41	36	37	35
28	32	36	33	40	36	36	35
29	32	35	33	40	36	36	36
30	32	36	33	42	36	37	36
31	32	37	33	41	35	36	36
31	32	37	33	40	35	36	37
32	32	37	33	40	35	36	37
32	32	37	33	40	35	36	37
32	31	39	33	40	35	36	37
32	32	39	33	40	35	36	37
33	31	39	33	38	35	37	38
33	31	39	33	40	35	37	38
33	31	38	33	41	35	37	39
33	31	40	33	39	35	37	40
33	31	38	32	38	35	37	40
33	31	39	32	39	35	37	40
33	31	39	32	38	35	37	41
33	31	39	32	37	35	37	41
34	31	39	32	36	35	37	42
34	31	39	32	36	35	37	42
34	31	38	32	35	36	37	42
34	31	38	32	35	36	36	42
34	31	38	32	35	36	36	
35	31	39	32	35	37	36	
35	31	38	32	35	36	36	
36	31	38	33	34	36	36	
35	31	38	32	35	37	36	
36	30	37	33	35	37	36	
37	30	37	33	36	37	36	
36	30	37	33	36	37	36	

Appendix I – Simulation Code

```
%MATALAB Simulation of Set-points
%Written by Andrew Charles
figure(1)
buffer = 288; %Buffer Length
length = 576; %Length of sample data
val = csvread('test.csv'); %Read Data from CSV file
val = val(1:576); %Select only Length Required
range = 0:1:30; %Temperature Range

title('Temperature')
xlabel('Sample Number')
ylabel('Temperature')

normal = zeros(1,length); %Set up empty array
rapid = zeros(1,length); %Set up empty array

for c = buffer+1:length %Sliding window
data = val((c-buffer):c); %Data in sliding window
x = histc(data,[range]); %Take Histogram
a = 0;  %Clear variable
b = 0;  %Clear variable

while a < (0.15*buffer) %Normal Percentage
    a = a + x(b+1);
    b = b+1;
end
pt = range(b); %Find index
normal(c) = pt; %Store

a = 0;
b = 0;
while a < (0.5*buffer) %Rapid Percentage
    a = a + x(b+1);
    b = b+1;
end
pt = range(b); %Find index
rapid(c) = pt; %Store

end
plot((1:length),val)
hold on
plot((1:length),normal,'k')
plot((1:length),rapid,'r')
title('Second Test')
xlabel('Sample Number')
ylabel('Temperature (degrees C)')
legend('Temperature','Normal Set-point','Rapid Set-
point')
```

Appendix J – Resource Planning

Requirement	Purpose	Cost	Solution
Workshop	To construct and test design	Nil	University workshop facilities are available and my own facilities exist
Computer	Research, project write up, Protel software and design software	Nil	On campus and my personal computer available for use, Pic software not yet known as PIC has not been chosen
Small consumables	Electronic components such as resistors and capacitors	<$20	Most common components can be obtained from electronic stores and University store
Electronic components- PIC, sensors, programmers	These components are critical to the design	$50 to $250	Budget limited to approximately $250 Online suppliers: Microzed, Futurelec etc

The budget for the components has been deemed around $250. Expenditure beyond this budget should be avoided but could be facilitated.

Appendix K – Timeline