Development and Test of Sensor-Aided Microcontroller Based Irrigation System with Web Browser Interface

Submitted to
Lance Fung
Dept of Electrical and
Computer Engineering
Curtin University
Perth, WA
October 25, 2002

By
Aaron Wills
Curtin University

Acknowledgements

There are a few thanks that have to go out to some people who have helped me along the way in order to produce this report.
I would like to thank Kevin most of all for his direction and guidance. The project would have been a lot more difficult without his help.
Thanks also to Antonio at Micromax (and JKMicro) for generosity in sponsoring me with a microcontroller and an I/O card which has been the heart of my system and something I couldn't have done without.
Thanks to my place of occupation, IT West, for help when I needed it and time off to do my project.
Last but not least, thanks to my family and my beautiful girlfriend Emma, for invaluable support throughout the last year.

Thank you!

Contents

Chapter 1

Introduction

It is well known that Australia is a dry continent. This stereotype has been reinforced in recent times with most of eastern Australia going without rain for the last 6 months and fears the dry spell could go on for another six months. Water restrictions have been in place in most places for several years now to try to limit water consumption. Keeping these facts in mind, I decided to tackle part of the problem by trying to improve the efficiency of water use in irrigation systems.

Common methods of water distribution can be enhanced or replaced by using recent technological advances. I hope to use it to improve the efficiency of water distribution, to automate the process of irrigation management, to provide an easy to use programming and reporting interface, and to provide a scalable, versatile base from which to expand or modify if needed.

One of the main drawbacks with the old fashioned auto timer system is that they do not cater for changing environmental conditions. Temperature, wind, rainfall and other elements can dramatically affect the amount of water needed to sustain a plants health. If these elements were monitored and used to influence the watering cycles, then the water used should be more effective.

Another aspect of regular irrigation systems that could be improved is its user interface. Not only can they be difficult to program and setup, they often don’t have the flexibility or scalability to do exactly what the user would like. With the

phenomenal growth of the Internet in the last decade, the use of a web browser interface has not only become the standard for viewing pages on the World Wide Web but has become more pervasive as a means of collecting user input or providing the output / status of control systems. In this situation the web interface provides an excellent platform from which to develop the system. The portability of this technology means that the system could be controlled from anywhere in the world (if required).

Once the basic requirements of the project had been established (sensor driven, web interface, high automation), the lengthy process of deciding what hardware to use and what software should tie it all together was undertaken. This took a quite some time as there were many alternatives on the market to choose from.

Eventually a microcontroller was chosen for the heart of the system. A PC based solution could have been designed easier but a microcontroller based solution meant that the system was more independent and hopefully more reliable, with cheaper running costs. Versatility was also a requirement of the design and as the controller is based on an Intel 386 processor running the DOS operating system, there is a wide range of software that can be run on it. There are also plans in the future to add extra components to the system to monitor security, control lighting etc.

Most of the sensors needed for the system could be have been cheaply produced using discrete components in conjunction with an analogue to digital converter, but in this case I used an all in one weather station which incorporated the following sensors: indoor / outdoor temperature, air pressure, rainfall, wind chill, wind speed, wind direction, indoor / outdoor humidity and indoor / outdoor dew point. Its serial port output meant it was able to send data directly to the controller for processing.

There is some question about the suitability of the Evapotranspiration calculation method (Hargreaves / Samani equation) used in this system. However, it was recommended on certain internet sites as a reasonably accurate method of calculating the Evapotranspiration which needed parameters that I could obtain with my sensor hardware.

Hopefully this project can show that automation in the area of irrigation can lead to efficient use of water and human resources.

I am writing this project as a requirement for my final unit of study of a Computer Technology degree, which required use of most of the knowledge acquired whilst at Curtin University.

Chapter 2

Theories, Models and Hypothesis

My main hypothesis in regards to this project is that using sensor based technology to automate irrigation can improve water usage efficiency. This is due to the fact that the sensors could provide information about the environment to an irrigation controller, and preset watering cycles could be adjusted to suit current weather conditions.

On rainy days, it would be expected of the system to detect the presence of rain and shut down cycles if necessary. On hotter, dryer days, the system should detect the high temperature and low humidity and subsequently increase the length of the next cycle. The Hargreaves / Samani equation only uses a subset of all the environmental elements gathered by the weather station, in calculation of evapotranspiration. This means not all of the sensors used in this project will have an affect on the watering cycle duration, but they may be used in the future if a more appropriate formulae for predicting evapotranspiration is found. All of the sensors are to have their data logged anyway, for reporting purposes, and for future examination.

Another feature of this system is to be its web browser user interface for gathering station programs, station information and displaying station status. This needs to be evaluated as an effective and advantageous method for user interaction. Ease of use, versatility and accessibility are hoped to be key elements of this design.

There are several irrigation technologies currently in widespread use around the world with some complimenting each other and some competing as the preferred implementation. Examples include:

1) Wireless technology to eliminate cables between controllers and stations. This adds the benefit of flexible sprinkler placement and saving of time and money as there are no cables to run or maintain (especially for larger sites). However, any money saved may be lost in the initial outlay for the wireless equipment.
2) The use of soil moisture sensors to provide feedback to the controllers. These devices would have been preferred over the use of the weather station (or perhaps in conjunction with it) in my project as this one sensor would be enough on its own to indicate to the controller what level of irrigation would be needed to compensate for the weather. This would greatly simplify the routines to dynamically adjust the cycles and could eliminate the need for an evapotranspiration equation. Again, a drawback is the fact that these devices are not cheap and that is why at this point in time I have chosen to use the weather station.
3) The most popular by far, is the regular 4 or 6 zone irrigation controller. These are relatively cheap, until you get to models with 8 or more zones. Overall they are quite inflexible, and quite often they can't do what the user would like. As well as this, they are 'static' devices without the ability to adjust to environmental conditions without user intervention.

In this project I hope to find the most suitable pieces of hardware and software to achieve the desired results of irrigation automation mentioned earlier. As I am on a budget, it isn't possible to purchase the more expensive parts that may be better suited for the task, and as such I've had to compromise on some parts. This means though, that if it can still be achieved successfully *and* relatively inexpensively, then it can have applications in more areas.

As far as I know there has not been any other studies into *how* effective, usage of sensor technology incorporated in an irrigation system, could be in delivering the least amount of water needed to sustain healthy plant or vegetative growth so I can't challenge or support the work of any others. There have been several studies into which evapotranspiration model is the best one to use, but no consensus as to which

one to use. The Hargreaves / Samani equation will be monitored for its effectiveness in this situation. As it makes some generalizations about the vegetation it may need to be adjusted to be accurate. Also it only uses the minimum and maximum daily temperature readings, and the location's latitude value to determine the current evapotranspiration. The state of the vegetation after several months may be the best indication of the suitability of using this formula.

Chapter 3

Materials and Methods

3.1 Hardware design

3.1.1 Microcontroller

The first step in the hardware design was to find some sources of equipment that I might need to complete the project. This was also an opportunity for me to see if there were any similar designs to the one that I proposed to build. If any products or devices of interest were found the manufacturer was contacted to obtain more information and the relevant costs involved.

It was known found from initial groundwork that a microcontroller based system was a desirable option. The main criteria in deciding on what to use were (in no particular order) size, performance, features and price. Initially I had considered using a serial interface in between the microcontroller and a PC for uploading of programs and retrieving data logged on the controller for reporting purposes. The PC was to perform all of the user input as well as gathering, processing and displaying the data. As well as performing these tasks locally, it should also be able to via a web page, to enable remote access of the system.

Tiger and Tiny Tiger computer modules were found to be of interest. They are computer modules that run BASIC (compiled on a PC and downloaded via an RS232 cable), and are about the size of a matchbox. They contained 2 serial ports, 10 bit analogue inputs, direct LCD connection, flash memory and battery backup.

I decided to try a different approach when I found microcontrollers at a couple of sites with embedded network chipsets including a RJ45 Ethernet network port. Some models were available as core modules to enable the user to design their own board, with the module at the heart of the system providing the processor and other main components. This approach could cut down the development time of a custom project by providing a base from which to develop. The Rabbit 2000 microcontroller core modules were available on a TCP/IP development board with flash RAM, Ethernet hardware, serial ports and digital I/O. The programs are created, compiled, and executed in Dynamic C and full TCP/IP source code was provided in the kit..

This seemed to fit better with my overall design as it simplified the exchange and management of data between the user and the system. It added a degree of robustness to the system as the controller could operate more independently. Being able to use the TCP/IP protocol meant the controller would be highly accessible and it should be relatively easy to set up communications with other devices. One drawback though was price, as most development kits were around the $650 price mark.

I was fortunate enough to receive a sponsor in Micromax; an eastern state based Electronics Company, who I had contacted previously via email to register some interest in some of their products. They supplied me with a full µFlashTCP development kit which had all the capabilities I was looking for. The µFlashTCP single board computer is based on the Intel 386Ex microcomputer which is software compatible with the Intel 80386 family of microprocessors. Onboard Ethernet provided a direct connection to 10BaseT networks. DOS compatibility allowed development in a familiar environment with a wide range of tools available. High endurance flash memory is used on the board in conjunction with onboard non-volatile memory. Applications are uploaded directly onto the flash disk. The kit also came with a simple web server program which could be modified to suit specific applications. See the appendix for more specification detail.

3.1.2 Sensors

After the processor had been chosen, the next decision to be made was what would be needed to sufficiently model the weather to allow for compensation of its effects. A number of ideas were considered, which included –

- Measuring the soil resistance to obtain a soil moisture level reading. As the moisture level decreases the resistance should increase. They could be placed at different depths to give a better overall indication of soil conditions.
- Using a 3 pin temperature sensor unit (possibly an NS ICLM35) to convert the current temperature to an appropriate voltage level. The three pins are ground, 5 volts and signal (1 – 4V). The signal would be sent to an ADC (Analogue to Digital Converter) which would send its output to the microcontrollers input port.
- Using a tipping bucket method of rainfall level reading. This involves having a small container on one side of a balance and a counterweight on the other side. The counterweight, in its starting position, would interrupt light between the two parts of an optocoupler (the emitter and the receiver). When it rained, the container would fill up, and at a certain point (most likely to be calibrated to be when 1 mm of rain had fallen) the container would reach its lowest point and the counterweight would reach its highest point, allowing light to pass through the optocoupler. Just after this, the container would tip over and empty its contents. The output of the optocoupler would be received by the microcontroller which would increment a rain counter on every transition of signal state.
- Using a 3 pin humidity sensor component (possibly a HIH3605A) to convert current humidity to an appropriate voltage level. The three pins are ground, 5 volts and signal (1 – 4V). The signal would be sent to an ADC (Analogue to Digital Converter) which would send its output to the microcontrollers input port.
- An alternative to using a more expensive humidity sensor could be to use two temperature sensors instead. Using the web bulb / dry bulb method to measure humidity, one sensor is wrapped in damp cotton or other material and one sensor is operated normally. The two temperature readings obtained are used with the web bulb / dry bulb equation to calculate the humidity.

- Evaporation could possibly be measured by filling a tube with water that had a level sensor inside it. As the water in the tube evaporated the water level would drop and at a certain point a switch could be triggered to send a signal to the microcontroller. This would let the controller know that enough evaporation has taken place for the system to need another cycle of irrigation.
- The wind speed could be measured if with an anemometer. Construction would involve using ping pong balls cut in half to catch the wind, attached to the top of a mast which would spin around. The mast would have a disc attached to it with holes in it around it lower down the mast. The two parts of an optocoupler would be attached on either side of the disc, registering a signal every time light was able to pass through one of the holes on the disc. The more holes the disc had in it, the more updates of wind speed would be possible. The time interval between signal pulses could be used to determine the wind speed.

Resources were again checked, targeting all major sensor manufacturers and irrigation suppliers to see if anything was available of the shelf. One requirement was the ability to be able to interface into a digital environment. After much research, most of the sensors found on the market weren't exactly suitable, being either the heavy duty industrial type which were too expensive to remain feasible, or simple sensors usually with an LCD display inbuilt, but no means of accessing any data signals from the device.

It was decided to design and build my own sensors using discrete components, and interface them into the microcontroller using an analogue to digital converter. This way, hopefully, I could produce all the sensors I needed relatively cheaply. That was the plan, until I managed to source a better alternative in an Oregon Scientific WM-918 weather station.

This instrument comes completely equipped with all necessary wiring and accessories: self-dumping electronic rain gauge, outdoor temperature/humidity sensor, and an anemometer/aerovane to measure wind speed and direction. Monitoring devices are wired to a compact desktop control panel that displays digital clock/calendar/alarm plus all weather readings (barometric pressure, dew point, wind speed and direction, temperature, wind chill, and rainfall). This is also the only one

with a 12-24 hour forecast system. The station operates on 8 AAA batteries or from a house current using an AC power adapter.

3.1.3 Relay Board Design

Relay output to the solenoid valves was selected over other alternatives (e.g. TRIAC) as providing the best isolation from possible transients induced in the solenoid wiring from lightning.

Power is obtained by rectifying the 24 volts AC required for the reticulation solenoid valves.

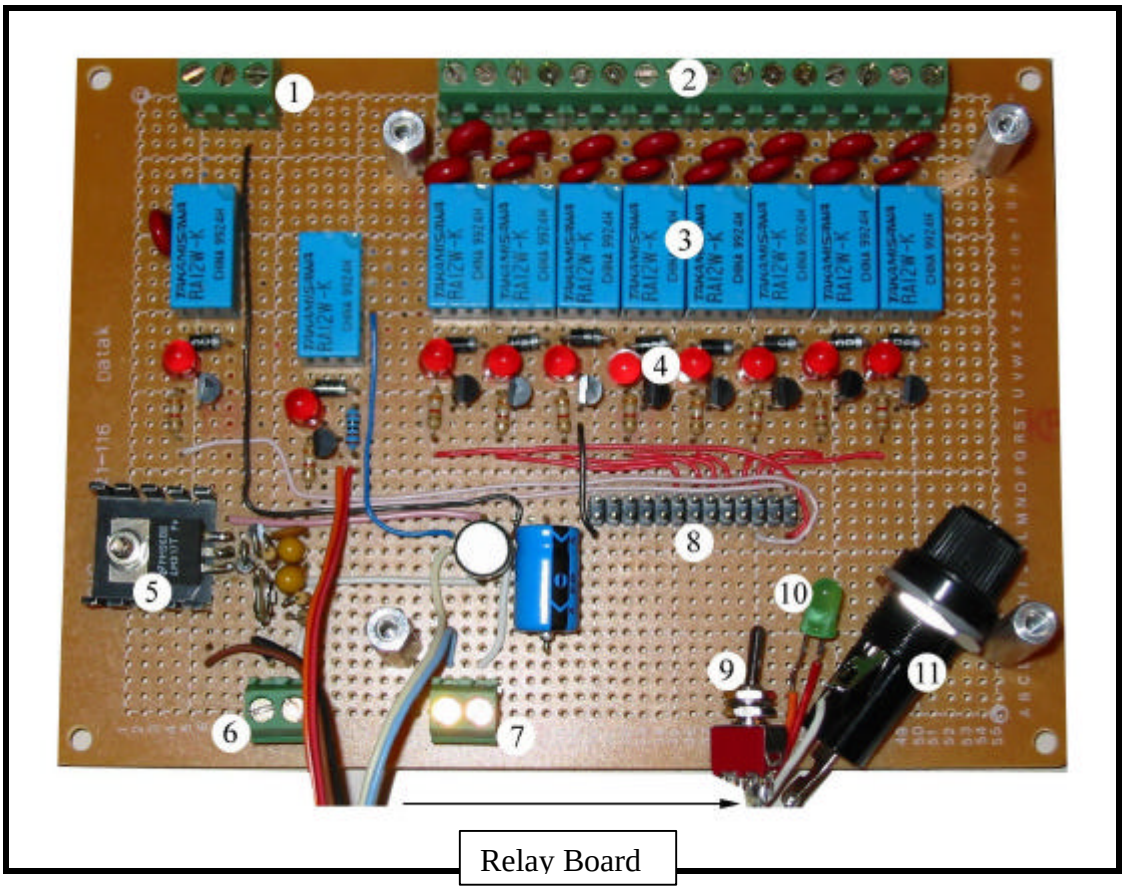
A LM317 was used as a regulator to provide the 12 volts for the relays – this was later modified to 13.8 volts to enable it to charge a standby 12 volt lead acid dry battery for mains failure power to the micro-controller. The output of this supply, besides powering the relays, is passed on to a 5 volt switch-mode supply used to power the micro-controller.

As double pole relays were used a saving of components was available by utilising each pole for a separate valve. Thus only 9 relays replaced the 16 that should have been required to control the reticulation 16 stations.

A led was included in each relay circuit for the prototype to enable ease of program testing during development. All relay coils have diodes across them to prevent back em spikes damaging other components. Capacitors were placed across the contacts to minimise contact wear from sparking.

See the following pages for an annotated photo and circuit diagram of the relay board.

Figure 3.1 – Photo of Relay board



Relay Board

1. Common
2. Valve output
3. Relays
4. LED - valve status
5. Voltage regulator
6. 12 V DC input
7. 24 V AC input
8. Microcontroller interface
9. Override switch
10. LED – power status
11. Fuse

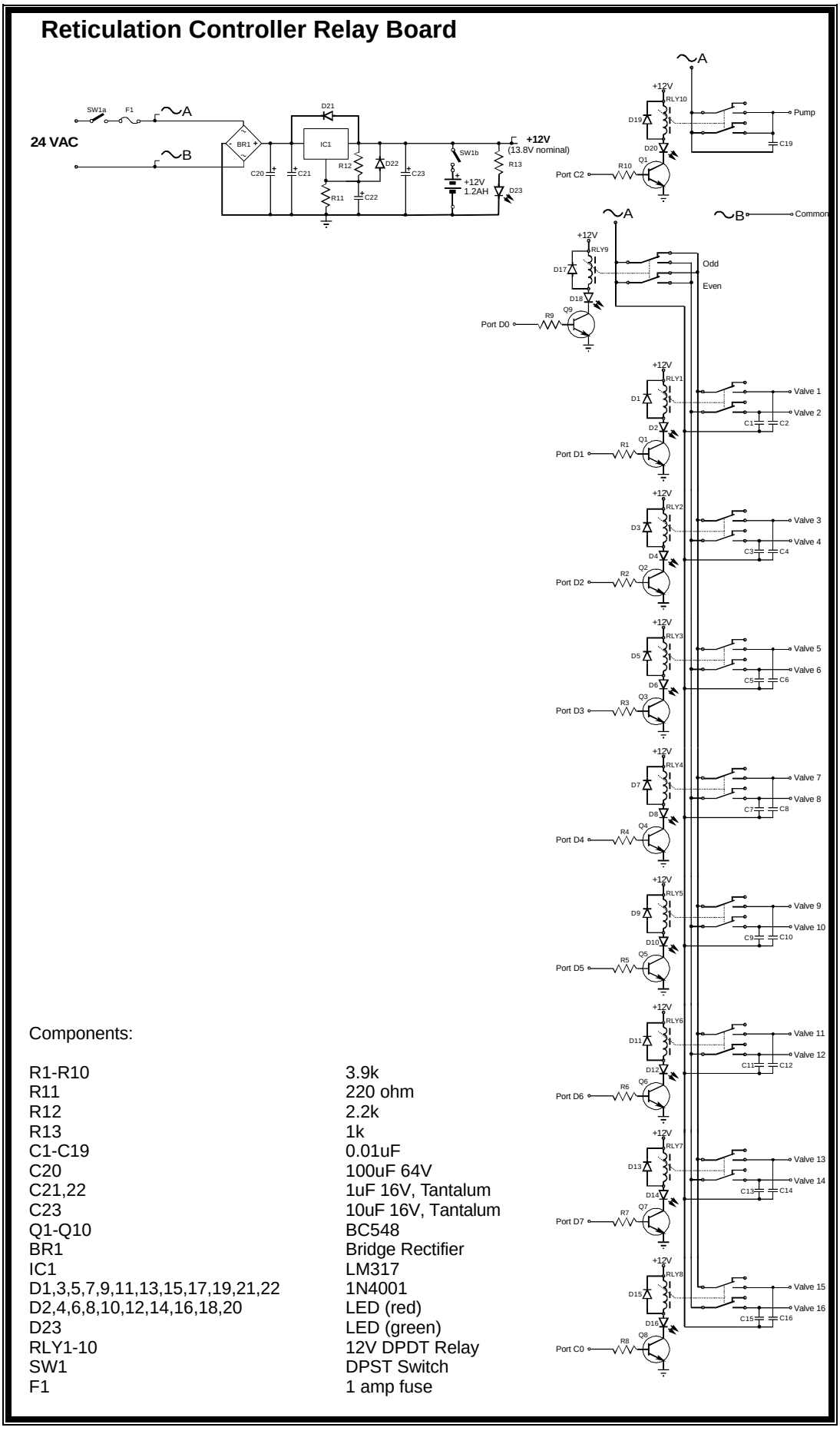


Figure 3.2 – Relay Board Schematic

3.2 Software Design

3.2.1 Browser interface

Before I could design the web interface for user input I needed to work out which controls would need to be accessible from the web page/s. It was decided that the following basic functions would be needed –

- Access to station data
- Ability to view and set cycles for each stations
- Ability to add or delete stations
- Ability to control level of automation for each cycle
- Access to weather data

These requirements were then used to form the web pages, sourced in html code, using a web page design program. They include –

- An index or home page from which there are hyperlinks to all other pages
- A station setup page where the user can enter information about a new station
- A cycle programming page for entry of each station's cycle parameters with a checkbox to enable or disable the Evapotranspiration adjustment routine.
- A current weather data web page.

Snapshots of these pages are contained in Appendix D – Web-based User Interface.

3.2.2 Web server

WebTCP is an embedded web server utility written in C++ that was included in the µFlashTCP software package. The standard program has been slightly modified to use CGI scripts for collecting user input. Because it is a DOS compatible program, it can run on the microcontroller (which runs in a DOS v3.3 compatible environment), turning it into a miniature web server. It is an executable program that is set to run in the startup batch file contained in the controller, so shortly after power is applied the web server is up and running.

3.2.3 File Transfer

Initial communications and uploading of programs into the controller was performed using a program called HyperTerminal which is shipped with Windows and is part of the default install. A serial cable was used to connect the controller's console port to the PC's serial port 1 or COM1. HyperTerminal was configured to connect "Direct to Com1" at 9600 bits per second, with 8 data bits, no parity bits, 1 stop bit, and no flow control. As soon as power is applied to the controller it should respond with a welcome message, and it is ready to begin developing applications.

To transfer a file to the controller the command UP was used with the following syntax:

UP *filename*

A line of 'C' characters will begin to appear, polling the terminal for the start of an Xmodem transfer. The file transfer is then started by selecting Transfer / Send. The 'Send File' window appears and you can then enter the required file. The Xmodem Protocol must be selected. A progress meter is then displayed showing the file transfer taking place. When the transfer has finished the DOS prompt will appear and the program or files can be run or viewed.

3.2.4 Weather station data collection

Extracting data from the weather station meant first discovering the protocol of its output data. Questions such as; how many bytes represented each sensor's value, what speed was the data transmitted at, do I use interrupts or not to alert the controller of data waiting from the weather station, - all needed to be answered before any programming could be started. The weather station's data protocol was found on the internet and has the following general specifications:

Group Number	Length (Bytes)	Report Interval	Contents Summary
8F	35	10 Sec	Time, Humidity
9F	34	10 Sec	Temperature
AF	31	10 Sec	Barometer, Dew Point
BF	14	10 Sec	Rain
CF	27	5 Sec	Wind, Wind Chill, General

Figure 3.3 – Weather station data protocol table

The full output data protocol from the weather station is listed in Appendix C. It shows what every byte represents and what data types all the variables are. I also managed to come across the baud rate, parity, etc for interfacing with the weather station. From this I wrote a program in c++ listed in Appendix F, to capture the data from the serial output of the weather station as it came in every 5 seconds. It then processes the stream of data using checksums for data validity. The data is broken up according to the protocol listing. The variables are then all copied to a log file on the microcontroller's drive for use by the other programs in the system and for reporting purposes.

There were some problems initially in receiving data from the WM 918, in that the whole data stream wasn't being received, so the checksums weren't allowing the main part of the program to run. However, when the data stream input routine was modified and fixed the whole stream was received without being truncated and the rest of the program could run.

3.2.5 Evaporation adjustment

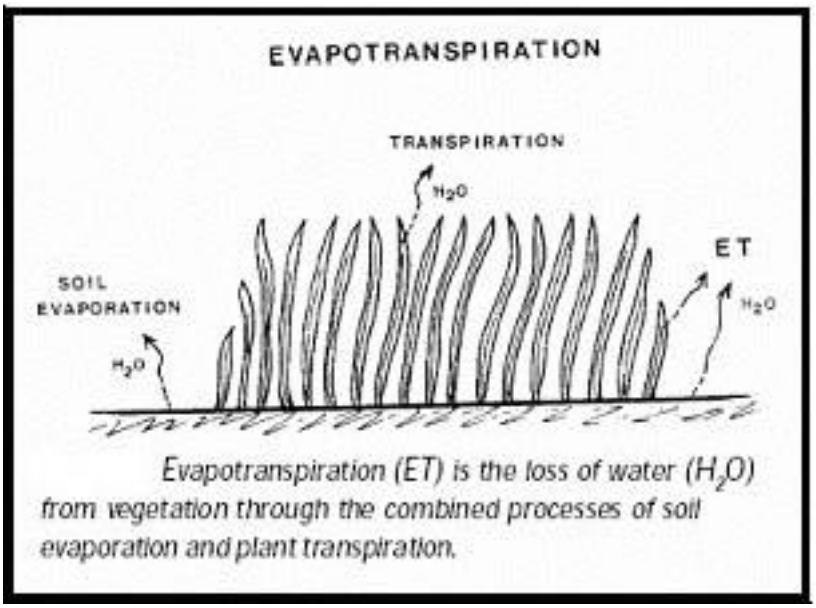


Figure 3.4 – Evapotranspiration definition

It was decided to use the Samani / Hargreaves method of evapotranspiration calculation due to its simplicity and minimum setup time and requirements. All it requires for inputs are minimum and maximum temperatures and the latitude of the

site. Another common method of calculating the evapotranspiration was the Penman Equation. This has found to be more accurate than the Samani / Hargreaves method over shorter periods but does require the measurement of solar radiation, an input not measurable with the current hardware. The actual function was written in c++ to accept the variable inputs and return the evaporation amount in millimeters to the main program.

Some theory behind the Samani / Hargreaves model and equation is given in Appendix E.

The function used to calculate evaporation is also listed in Appendix E.

The outline of the process which helped to calculate water requirement is shown on the following page.

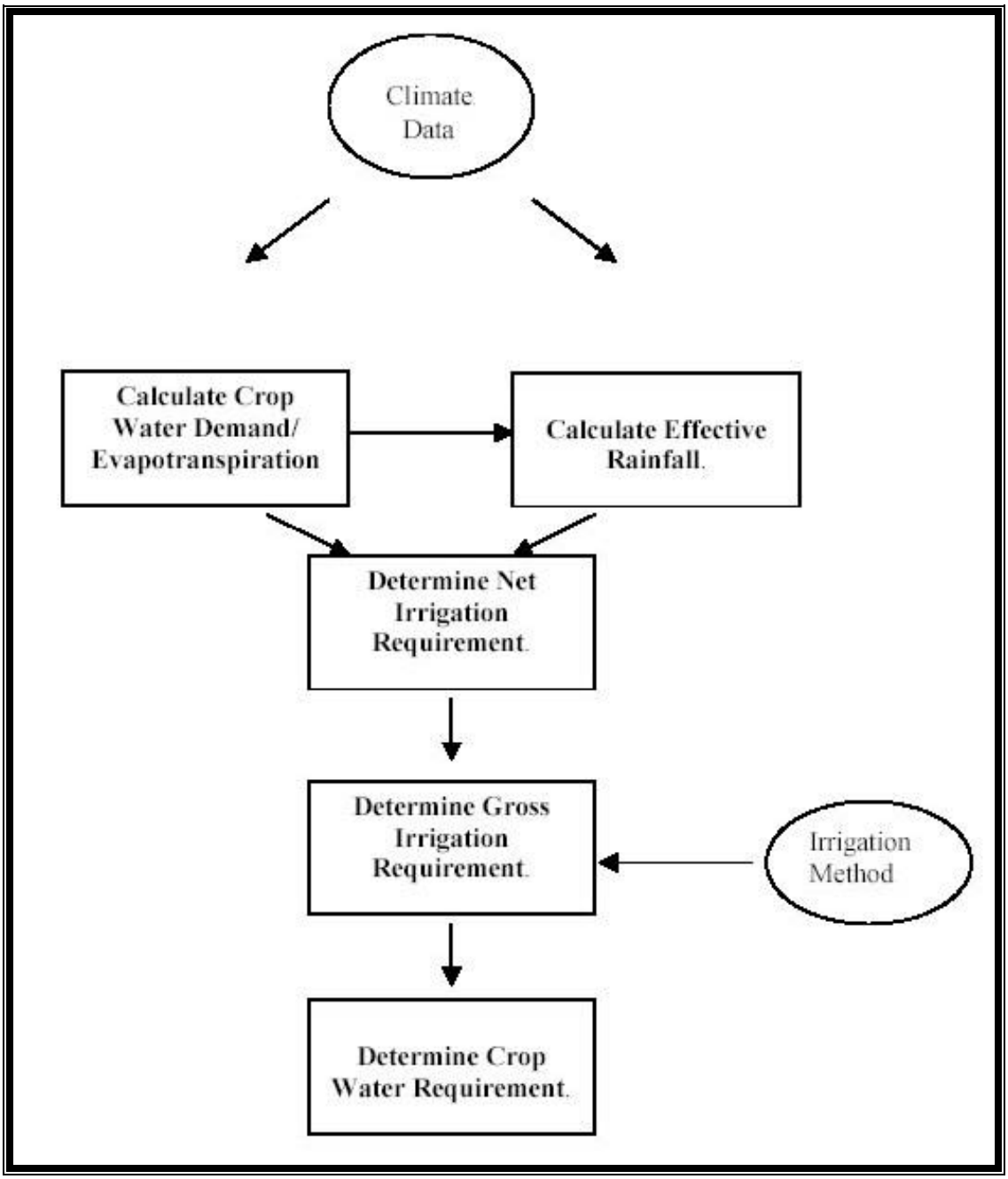


Figure 3.5 – Crop Water Requirement Determination Process

3.2.6 Valve Control

Listed in Appendix G are a few functions written in c++ by Kevin Taylor to control the irrigation valves. There is the function ValveOn which turns on the required valve 1-16. Function ValvesOff is called during initialization, and at the completion of a reticulation cycle - i.e. after all required valves have been operated. Function InitialisePort is called to initialize port directions and clear valves.

3.3 Implementation and Testing

3.3.1 Test Bench Setup

Most of the major components of the system were set up on a bench to establish that the various components were functioning as they should. The photo pictured below shows most of the parts during a test, without the relay board connected.
The controller has power applied (onto pins marked J8) to it from a transformer that plugs into the mains, which steps down to 5V after passing through another transformer / regulator box.

One thing not to do when powering on the microcontroller, is to reconnect to main transformer to the secondary transformer / regulator box when the main transformer is already turned on and plugged into the microcontroller. This procedure ruined my first transformer / regulator box so another unit had to be attained.

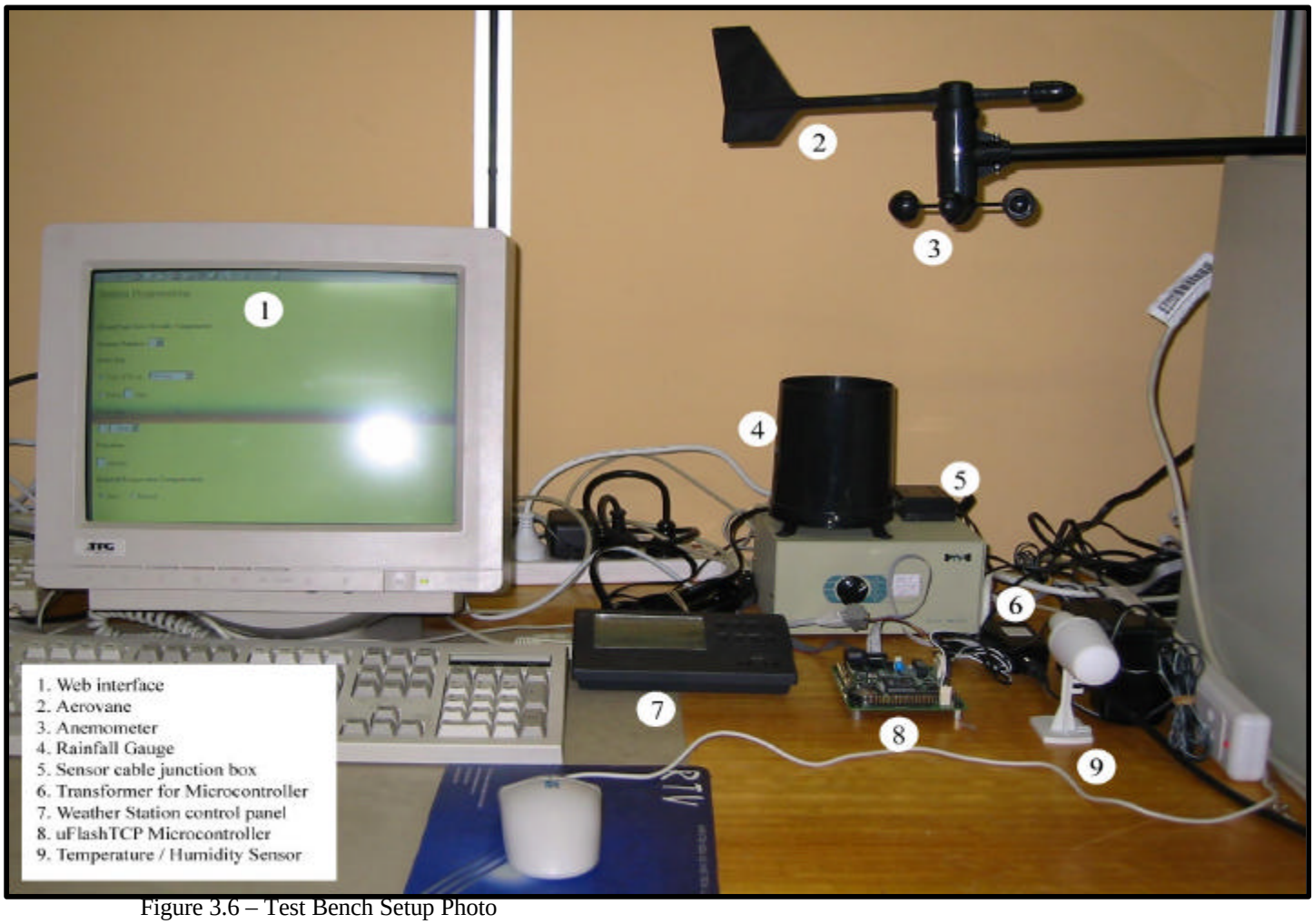


Figure 3.6 – Test Bench Setup Photo

A CAT 5 network cable is used to connect the controller to the network and test web page functionality. An RJ 45 adaptor is connected to the Ethernet pins on the controller allowing a normal network cable to be used. If you have a “crossover” cable (a CAT5 cable, usually red in colour, with the receive/transmit wires reversed) the other end of the cable can be plugged into another PC’s network card for direct communication with the controller. Otherwise a hub or switch is needed, with the controller and the pc connecting to the hub.

A 9 pin serial cable should be attached from the console port or COM2 on the controller to a serial port of the pc to enable initial communications, file transfer and for troubleshooting purposes. A similar cable was used to interface the weather station to the COM1 port of the controller. This is the data cable that transfers all of the climactic data.

The sensors don’t connect directly to the weather station main console. The sensor cables converge at a junction box, which integrates them into one larger, flatter cable which then clips into the weather station. The sensors are provided with a cable attached resembling a telephone line, having an RJ 11 adaptor on the end.

Once HyperTerminal is configured correctly and power is applied, communication with the controller is possible. A display message from controller should scroll through, ending up at a command prompt. From here, the onboard network interface is configured assigning an IP address, net mask and gateway with the TCP/IP configuration utility provided.

The customized version of the web server, WebTCP, was uploaded to the controller, as well as the html pages that form the user interface, the weather station data processing routine, the valve control routine and the Evapotranspiration adjustment routine.

The system was run for a few weeks on the test bench, making sure that the sensor measurements were roughly calibrated, log files were being created and the web server was still running.

3.3.2 On-Site Setup

Once satisfied the system was properly functioning, everything was moved to the primary test site, located at a residence in Kalamunda, Western Australia. The site had 14 stations covering approximately 1000 m^2 of irrigated land. The total water usage is 9500 litres for 10 minutes at maximum capacity.

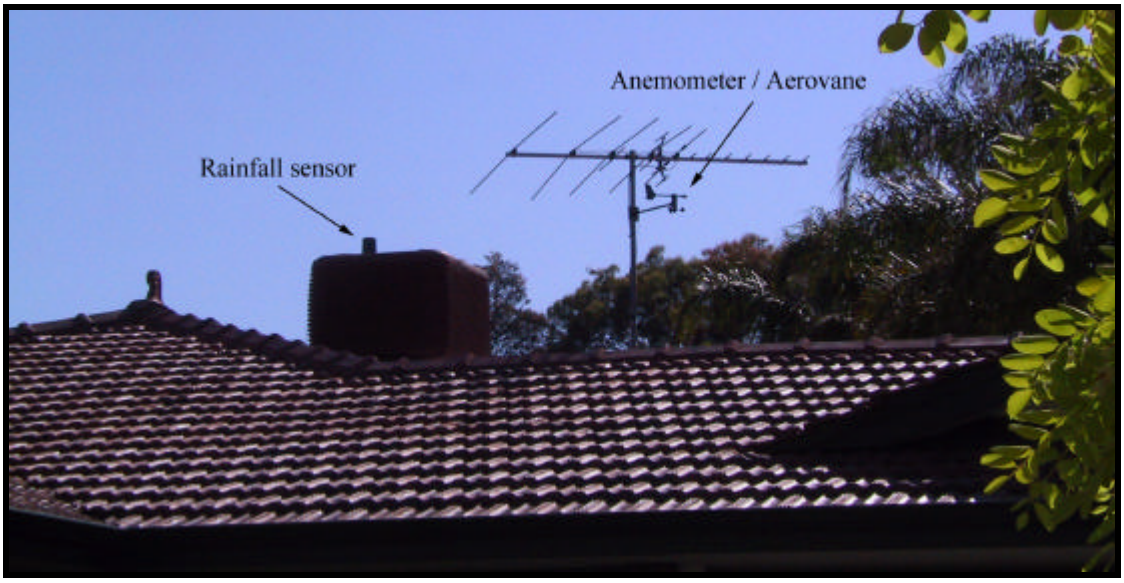
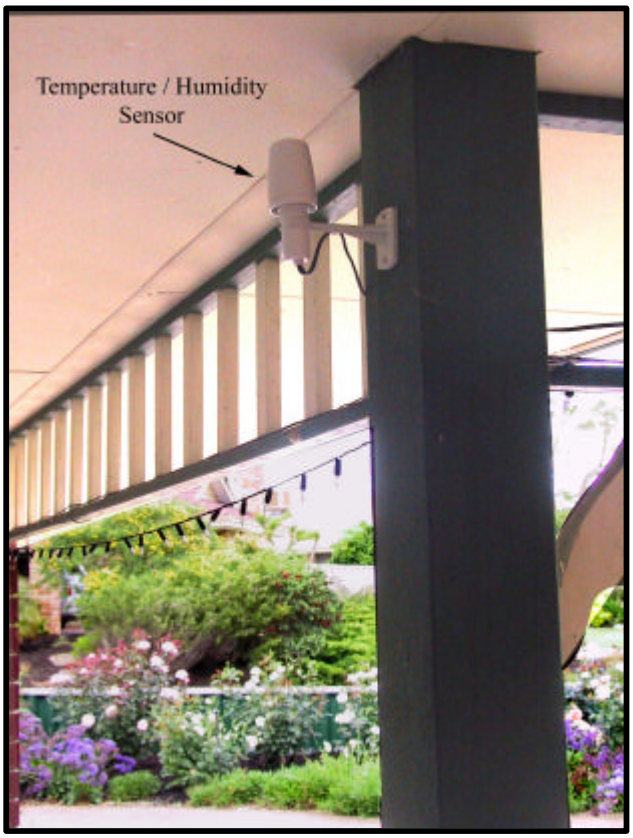


Figure 3.7 – On-site sensor placement.

Figure 3.8 – Temperature / Humidity Sensor Placement.



As seen above, the rainfall sensor was securely fixed on top of the air conditioning unit, so it would be up high and out of the way of any debris that could fall into the unit. The anemometer / aerovane sensor unit was fixed to the existing television antennae pole. The temperature / humidity sensor (shown on the next page) was mounted underneath the back verandah of the residence, ensuring it would be not be in sunlight but would be as ‘outside’ as possible. This sensor should be in a shaded spot so direct sunlight doesn't warm

up the sensor giving misleading readings for the temperature.

Avoiding running the wires to the out side sensor near any electrical lines or wires that contain AC voltages was advised. The LCD weather station console, the microcontroller and the relay boards were all placed in an undercover area at the rear of the house. The connector board, where the sensor cables meet, has to be indoors or inside a weather proof casing to avoid deterioration. When the weatherproof housings can be made, these parts would be placed outside as well.

Once all the cables were connected (shown as a guide on page 24) and power was applied, the weather station console needed to be reconfigured as their were no batteries installed for backup power at this time, so the time, date and unit types (Celsius or Fahrenheit etc.) were entered and set. The controller also started up, automatically executing the web server program. The Ethernet cable from the controller was plugged into a 10/100 Mb/s network switch inside the house which also had a couple of PC's linked to it. As the PC's were on the same TCP/IP subnet as the controller, they were able to browse to the web interface home page located at http://192.168.0.1/index.htm (on the controller, the file "b:\www\index.htm").

The station details were set up with the following parameters after clicking on the home page link to Setup New Station located at http://192.168.0.1/setupstn.htm (on the controller, the file "b:\www\setupstn.htm"):

Station No.	Location description	Water Usage (litres/min)	Location type
1	Front lawn 1	72	outdoor
2	Front lawn 2	72	outdoor
3	Front rose garden 1	72	outdoor
4	Front rose garden 2	72	outdoor
5	Front central garden	72	outdoor
6	Front garden by fence	72	outdoor
7	Rear lawn 1	72	outdoor
8	Rear lawn 2	72	outdoor
9	Rear lawn 3	72	outdoor
10	Rear lawn 4	72	outdoor
11	Rear lawn 5	72	outdoor
12	Rear fruit tree 1	72	outdoor

13	Rear fruit tree 2	72	outdoor
14	Rear rose garden	72	outdoor

Figure 3.9 – Station Details Table

After these were setup, the station cycles had to be entered via the link to Station Programming which is located at http://192.168.0.1/stnprog.htm (on the controller, the file "b:\www\stnprog.htm"). The details entered were; what station number the cycle applied to, whether the cycle was to run every constant number of days or certain days of the week, start time and duration, and whether the cycle was to be of automatic type or manual. Automatic mode means that sensor compensation will take place and the duration entered can be modified by the program. Manual mode means that whatever is entered by the user is applied no matter what the weather station readings.

Stations "Rear Lawn 1" and "Rear Lawn 2" were set on automatic and the rest of the stations were set to run on manual mode to check the functionality of the adjustment routine. Unfortunately there are no indoor or undercover stations, so the part of the program that adjusts the duration differently for these types of stations can't be tested long term. For example, an undercover (but outdoor) station will be affect by all of the same elements as an outdoor station except it wouldn't receive any rain. The program would check the station type, and if undercover it would know not to subtract any rainfall received from the net irrigation requirement of this station.

Initially, test cycles were run to check that the programmed valve was opening at the programmed time, and more importantly that it was shutting off the valve as well. Tests were also done to make sure the manual shutdown switch located on the relay board was functioning. Other tests included simulating rain to make sure the rainfall sensor was functioning and making an impact on any cycles that were set to run with the automatic setting.

The View Weather Data webpage which is located at http://192.168.0.1/wthdata.htm (on the controller, the file "b:\www\wthdata.htm") was tested to make sure the latest weather data was viewable via a web browser from a PC on the network.

The system was left in place in normal operating conditions for one month over the period August 1 to August 31 with the system functioning and logging data.

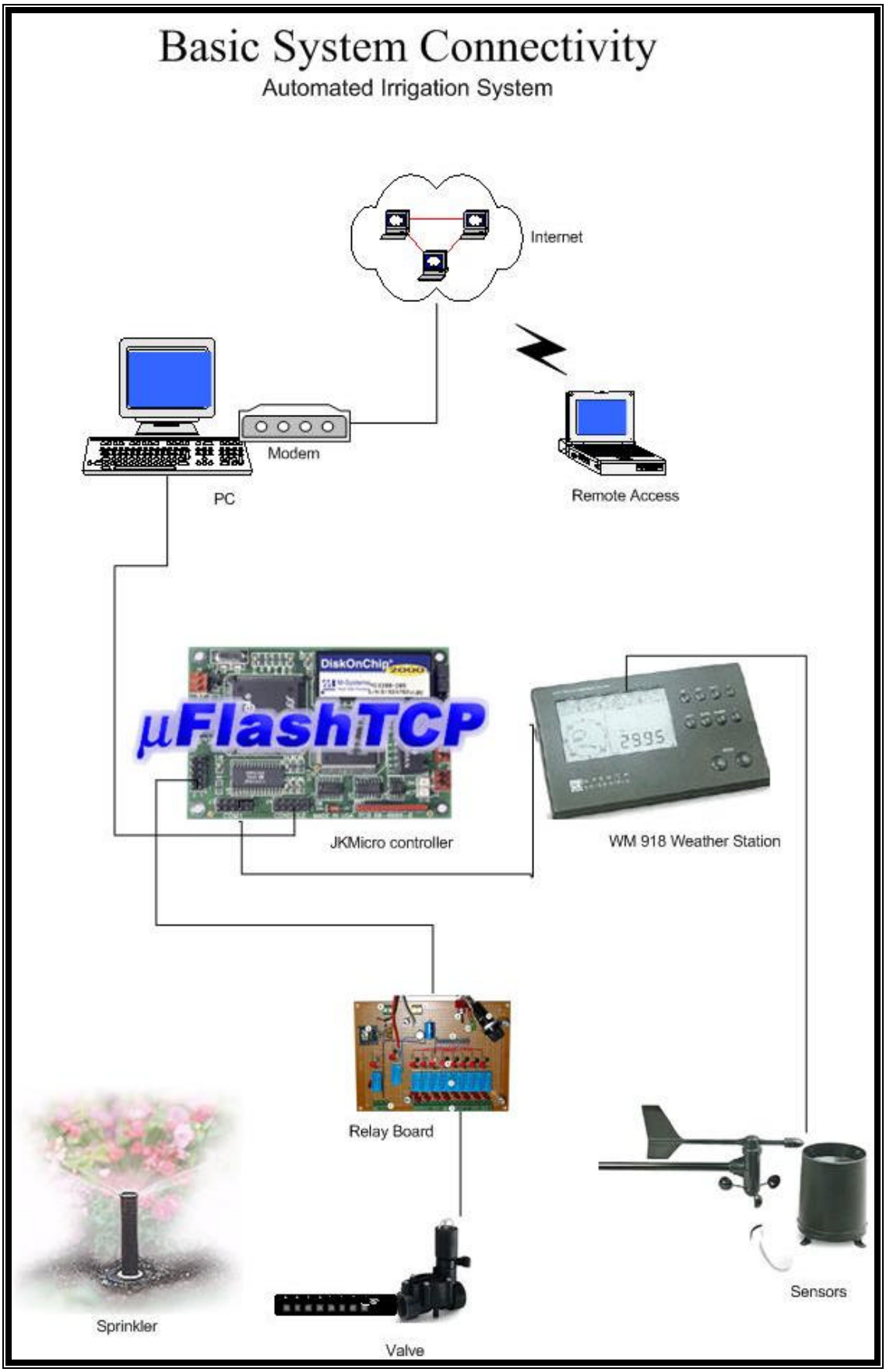


Figure 3.10 – Basic System Connectivity

Chapter 4

Results and Discussion

The graph and table below shows a portion of the climatic data logged for the time period August 1 to August 31 by the controller. The data is stored in a text file called log.txt located in the folder location “b:\www\” on the controller.

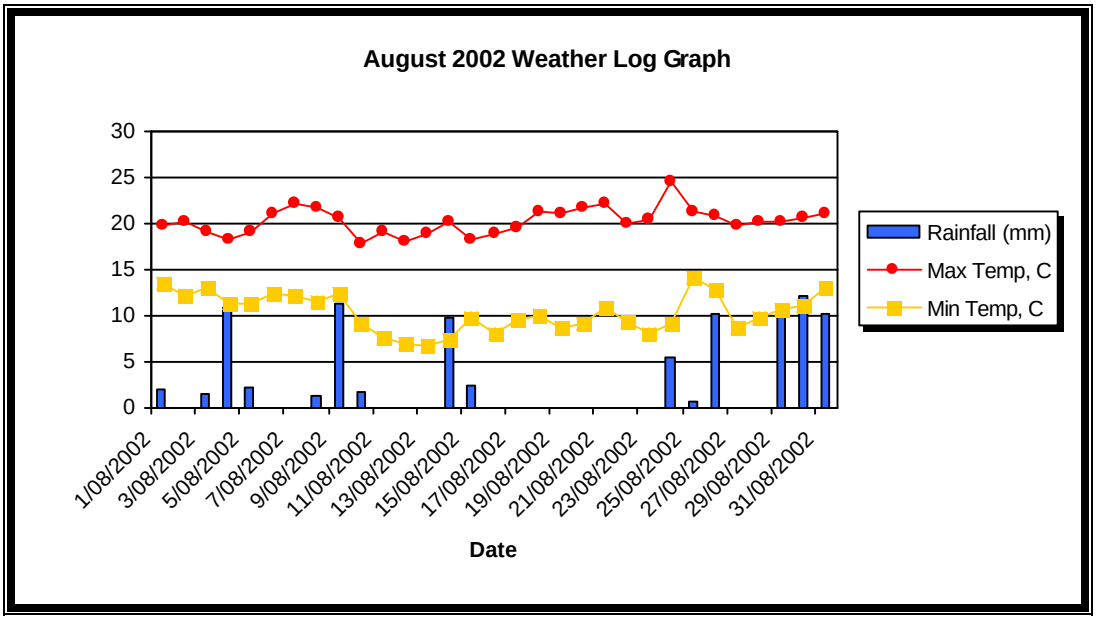


Figure 4.1 – August Weather Log Graphed

AUGUST, 2002							
Day	**MAX (°C)**	**MIN (°C)**	**RAIN (mm)**	**Day**	**MAX (°C)**	**MIN (°C)**	**RAIN(mm)**
1	19.7	13.5	2	17	19.6	9.6	0
2	20.2	12.1	0	18	21.3	9.9	0
3	19.2	13.0	2	19	21.0	8.6	0
4	18.3	11.2	11	20	21.7	9.1	0
5	19.2	11.2	2	21	22.1	10.9	0
6	21.0	12.4	0	22	20.1	9.3	0
7	22.2	12.1	0	23	20.4	8.1	0
8	21.7	11.5	1	24	24.6	9.2	5
9	20.7	12.5	11	25	21.3	14.2	1
10	17.8	9.2	2	26	20.9	12.8	10
11	19.1	7.6	0	27	19.8	8.7	0
12	18.1	7.0	0	28	20.3	9.7	0
13	18.9	6.8	0	29	20.3	10.6	10
14	20.2	7.5	10	30	20.7	11.1	12
15	18.3	9.8	2	31	21.0	13.0	10
16	19.0	8.1	0				

Figure 4.2 – August Weather Log Data Table

The following graph and table show the evapotranspiration calculated from the maximum and minimum temperatures and date, the rainfall recorded for each day, and the amount of water in mm that is needed to counteract the evaporative effect.

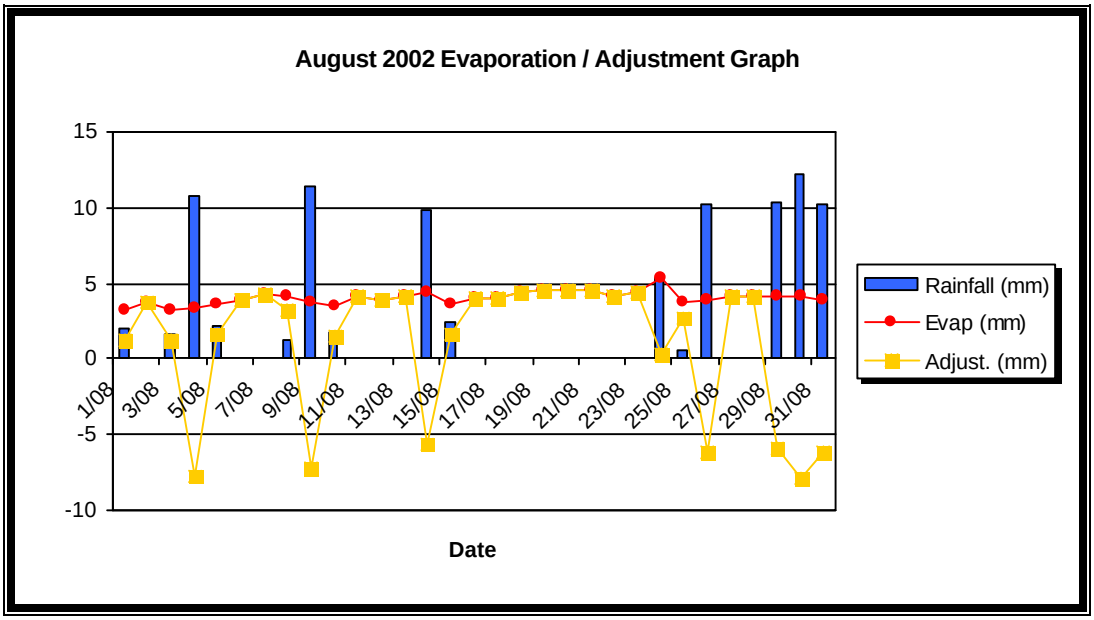


Figure 4.3 – August Evapotranspiration/ Adjustment Graph

AUGUST, 2002

Day	Evapotranspiration (mm)	Rainfall (mm)	Adjustment (mm)
1	3.2	2	1.2
2	3.7	0	3.7
3	3.2	2	1.2
4	3.3	11	-7.7
5	3.6	2	1.6
6	3.9	0	3.9
7	4.3	0	4.3
8	4.2	1	3.2
9	3.8	11	-7.2
10	3.5	2	1.5
11	4.1	0	4.1
12	3.9	0	3.9
13	4.1	0	4.1
14	4.4	10	-5.6
15	3.6	2	1.6
16	4.0	0	4.0

17	4.0	0	4.0
18	4.4	0	4.4
19	4.5	0	4.5
20	4.6	0	4.6
21	4.5	0	4.5
22	4.2	0	4.2
23	4.4	0	4.4
24	5.3	5	0.3
25	3.7	1	2.7
26	3.9	10	-6.1
27	4.2	0	4.2
28	4.2	0	4.2
29	4.1	10	-5.9
30	4.1	12	-7.9
31	3.9	10	-6.1
Total	122.5	91	31.5

Figure 4.4 – August Evapotranspiration/ Adjustment Graph Data Table

Every millimeter of water needed by the land equates to an extra 1 minute and 10 seconds of sprinkler time. This figure was achieved using the following method (as the method I found used imperial measurements, I converted the inputs to imperial and the results to metric):

- Find the approximate area covered by each zone in square feet.
- Calculate the gallons per minute (gpm) used by each zone.
- Use the following formula **PR (in/hr) = (96.3 x gpm) / Area (ft^2)** where PR stands for Precipitation Rate of the sprinklers in a station.
- Convert the PR in inches per hour to mm/min by multiplying by 0.423.
- Take the inverse of this result to obtain the number of minutes to deliver 1 millimetre of water to the zone.

The total area irrigated was known to be 1000 m^2, which divided amongst 14 stations this gives 71.42 m^2 of land per station. This converts to be 768 ft^2 of land per station. It was already known that each station could pump 72 litres per minute which converted to 16 gpm. This means the PR = 96.3 x 16 / 768 = 2.01 in/hr per station.

This converts to be 50.96 mm/hr or 0.85 mm/min. The inverse of this is 1.18 minutes which is 1 minute and 10 seconds to deliver 1mm.

The average adjustment needed for the August period per day was 3.7 mm which equates to 4 min 22 seconds time the valves would be open for. Taking a cumulative approach, all the adjustment time is added up until the next scheduled cycle. At this point all the cumulative time is used (up to a maximum period). If running the cycles every 3 days then the average duration of each cycle needed to counter evapotranspiration was be 13 minutes and 6 seconds. The first graph on the following page shows the duration and water usage of each cycle using the cumulative method running every 3 days. The second graph shows the duration and water usage of each cycle using the manual mode.

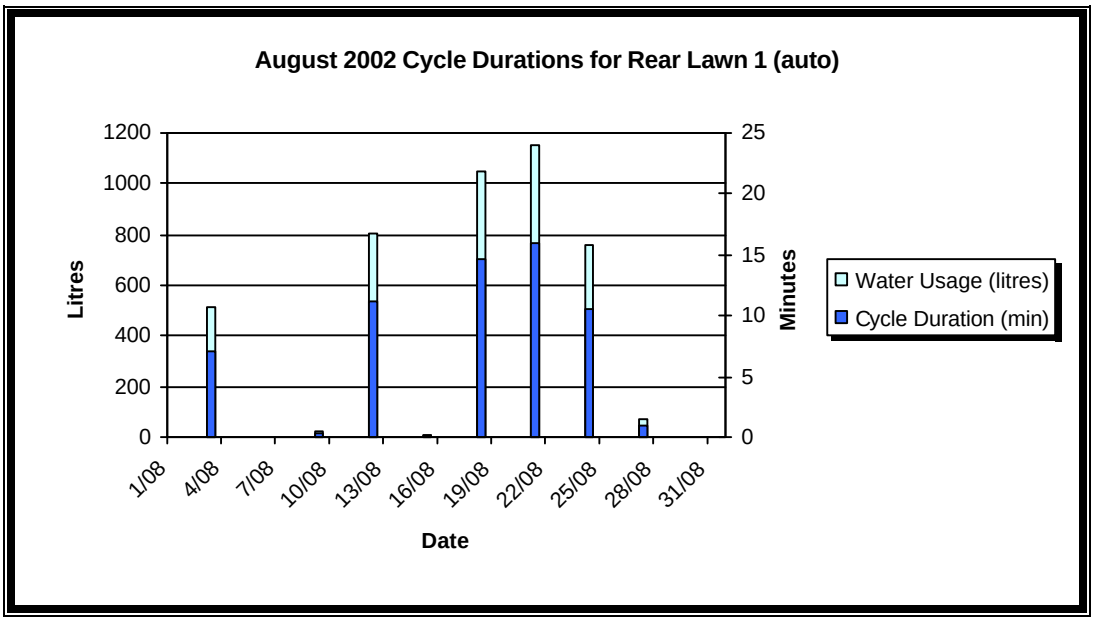


Figure 4.4 – August Cycle Durations for Rear Lawn 1

Note that automatic cycles that run for less than 2 minutes wouldn't be run due to the high overheads in starting up the system. The total amount of water used by the automatic station over the month was calculated to be 4376 litres. This compares to the manual station which was calculated to have used 7200 litres over the same period. This is about 65% more than the automatic station with an extra 2824 litres in one month.

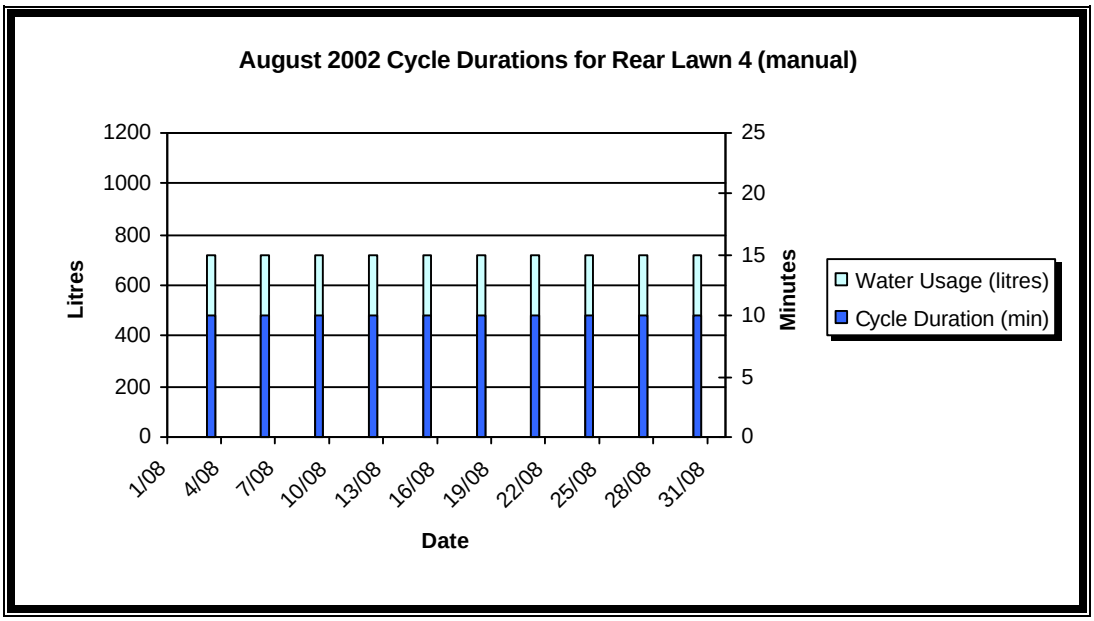


Figure 4.5 – August Cycle Durations for Rear Lawn 4

These results show that using the same programmed cycle times for two different stations can result in differing actual run times if one is running on automatic and the other on manual mode. It is also noticeable, on the auto station, that the cycle duration was shorter when rainfall was received in the 3 day period before the programmed cycle time. The actual cycle duration was slightly longer when the days leading up to the programmed cycle time were hotter. As expected the manual station had consistent run times which weather conditions did not affect. A bit more time would be needed to make sure the adjustments made by the algorithm are appropriate whilst maintaining vegetation health.

The browser interface was tested from a remote site by dialing into a computer on the same network subnet as the microcontroller. This host PC acted as a DHCP server giving the remote PC an IP address on the local network, and from that point the communication with the controller proceeded to behave as if connected directly to the network.

To simplify the cycle adjustment process, it would probably be more beneficial to use a soil moisture sensor instead of all the other climatic sensors. This way only one measurement would be needed and it would provide a more direct method of assessing the vegetation's water requirements. These devices, however, are not cheap at this stage so they were ruled out in the feasibility study.

Chapter 5

Conclusion

What I managed to discover in the process of design and implementation of my project is that by combining the technologies of automation in the area of reticulation and weather sensing equipment, more efficient water delivery can be made possible. This can be achieved while maintaining simplicity, ease of use and ease of implementation. The use of a browser interface meant it was relatively easy to work with from both the user point of view and the programmer's point of view. The initial cycle setup time is significantly less than the first attempt at setting up a cycle on most manual irrigation controllers. The lack of a hardware interface means that it isn't an expensive part that can be damaged and need replacing (although you will need a computer to modify details).

I found that the programmed and actual cycle times differed when a station was setup to make use of sensor based cycle time adjustments. Although I only had one month to gather results to be studied which isn't really enough time to make a definitive judgment, I could see that water saving of about 65% was possible for the month of August if using an automatically compensating system. This could have a significant effect on water consumption habits if results continued to show these sorts of differences. No visual indication of differences in vegetation health could be seen between the automatic station zone and the manual station zone as the testing time period was too short. The system has not yet been tested over the more extreme months which could provide problems not yet encountered, such as lightning or heat interference.

I believe this form of automation could be applied in a home environment well, due to its ability to have other applications developed for it such as home security and home lighting. The microcontroller has been reasonably easy to get up and developing as well as being very flexible. Occasionally the microcontroller would need resetting, which meant disconnecting the power source and reapplying it seconds later. This could have been caused by power supply fluctuations, faulty components, moisture in the controller or an undiscovered fault.

I found it a bit disappointing that soil moisture sensors were priced out of reach as they could have simplified the process of estimating water requirements for the vegetation. However, the weather station used in its place proved to be very reliable and multifunctional with applications not only in irrigation but perhaps other areas as well such as providing weather station data accessible to others on the internet. Some more evapotranspiration equations / algorithms may need to be assessed for better suitability if the cycle time adjustments seem to be having a detrimental effect on the vegetation.

Generally I feel the project was a success, with the system performing most of the functions that I wanted as I had expected. More time will obviously allow more information to be collected, with further refinements and improvements being inevitable.

Chapter 6

Recommendations

I would only recommend this model of irrigation control for medium to large areas of land (1000 m^2 or more) with a number of zones more than 6 - 8. This is due to the price and time involved in setup of the system. The system hasn't been tested with more than 14 stations, but with a bit more development in the pump control / cycle scheduling areas it seems the application will become more feasible as the number of stations increases. At the moment there is a 16 station limit but this could be increased by adding another relay board or making a new one with perhaps 32 relays and/or relays with different voltage levels for applications in other areas.

At the moment the web interface needs more refinement with more user controls. Functions that need work are:

- Pump control needs to be included on the web interface.
- Scheduling of cycles needs more development with more sophisticated user entry options available such as a calendar type graphical input.
- Ability to represent the site graphical showing zone boundaries, sensor locations and sprinkler locations.
- Each station's own web page including the ability to view station status and view information such as total hours in use and more.

The currently used evapotranspiration compensation needs qualification over a longer period to assess its suitability for our climate and circumstances. If it proves to be not suitable, then a number of other methods of estimating evapotranspiration, using a variety of inputs, are available. The Penman – Monteith method (Monteith 1981)

computes reference evapotranspiration from net radiation at the crop surface, soil heat flux, air temperature, wind speed and saturation vapour pressure deficit and although is much more complicated, it could prove to be more accurate than the Samani – Hargreaves method. This would probably be the next algorithm tried.

The soil moisture sensor needs to be evaluated as a replacement for the other sensors in the future when the price and availability factor is a bit better. This could simplify the system and lessen the risk of hardware failure due to the system having fewer physical components. Weather proof casings are still needed to enclose the microcontroller, weather station console, relay board and sensor junction box.

As mentioned earlier the core of the system – the µFlashTCP – is very flexible, and extra modules or applications are still to be developed for the base system. A burglar alarm / home security system with internet access should be attainable, perhaps with the ability to send an email or SMS when the alarm has been activated. Lighting, clothes lines, windows, window shutters and roller doors could all potentially be controlled and monitored via the web interface with a bit more development. With the recent introduction of the new Mini – ITX motherboard standard, it would now be considerably easier and cheaper to obtain a small, powerful platform from which to develop a similar system.

Testing is currently underway to integrate wireless technology into the system, with a 22 Mb/s 2.4 GHz Wireless Access Point running in wireless client mode that is connected directly to the microcontroller instead of connecting to a switch or another PC. Once operational this will allow the controller unit to be placed up to 130 meters from the receiver with the standard antennae. This would be more advantageous in larger sites where a large cable run may have been necessary.

Another technology of interest that could enhance the system is the use of electronic sprinkler solenoids that can have individually assigned ID numbers. Packets of data could run through the sprinkler data network having the sprinklers ID number in its header, which is similar to the TCP/IP protocol when applied to computers. This would mean individual sprinkler heads could be controlled providing the ultimate flexibility in cycle control.

Chapter 7

References and Bibliography

Keir, Andy
WM-918 Weather Station Communication Protocol
2001

Kruse, Matt
Guide to CGI Scripting
www.mattkruse.com/info/cgi/

Waterfield, Bob
WX200 / WM918 Modifications and Fixes
www.qsl.net/zllvfo/wx200/mods.htm

Z. A. Samani and M. Pessarakli,
Estimating Potential Crop Evapotranspiration with Minimum Data
Transactions of the ASAE Vol. 29, No. 2, pp. 522-524
1986

Georgie Mitchell, Ray H Griggs, Verel Benson, and Jimmy Williams
The Penman Potential Evaporation Equation
1997

Liberty, Jesse
Teach Yourself C++ in 21 Days, Fourth Edition
Sams Publishing, 2001

Pennells, Steve
"WA faces 100 years of hot, dry weather"
West Australian 2002

Bogart, Theodore F
"Electronic Devices and Circuits"
Fourth Edition
Prentice Hall, 1997

Brey, Barry B.
"The Intel Microprocessors"
8086/8088, 80186, 80286, 80386, AND 80486
Architecture, Programming, and Interfacing
Third Edition
Prentice Hall, 1994

Allen R. G., Pereira L. S., Raes D, and Smith M.
Crop Evapotranspiration - Guidelines for computing crop water requirements.
FAO Irrigation and Drainage Paper No. 56. Rome, Italy
1998.

Doorenbos, J. and W.O. Pruitt.
"Crop water requirements"
FAO Irrigation and Drainage Paper No. 24. Rome, Italy.
1977.

Hargreaves, G.H. and Z.A. Samani.
"Estimating potential Evapotranspiration."
Journal of Irrigation and Drainage Division.
1982

Geoff Kite and Peter Droogers

“Comparing Estimates of Actual Evapotranspiration from Satellites, Hydrological Models, and Field Data: A Case Study from Western Turkey”.

IWMI

2000

Appendices

Appendix A Microcontroller specifications

Courtesy JKMicro Website

Error! Unknown switch argument.

Error! Unknown switch argument.

Connectors and Jumpers

J1
General I/O & Synchronous Serial

J2
RS-485

J3
COM1

J4
COM2 (Console)

J5
Multi-I/O Bus (General I/O)

J8
Power

J8
Ethernet

JP1
Watchdog NMI Enable

JP2
Socket Memory Type

JP3
Boot Memory Location

Specifications

Processor	Intel 386Ex, 25MHz
Operating System	XDOS(MS/PC DOS 3.3 compatible)
Memory	512K SRAM, 512K Flash
Ethernet	10BASE-T, NE2000 compatible Link status and Activity LEDs
Serial Port 1	RS-232 with 5 handshake lines COM1, address 0x3F8, IRQ4 115200 baud maximum
Serial Port 2	RS-232 no handshaking or RS-485 half duplex, COM2, address 0x2F8, IRQ 3 115200 baud maximum
Digital I/O	10 Bits: P3.0-P3.5 & P1.4-P1.7, Pin configurable as input or output P3.3 & P3.4 configurable as interrupts 8mA source/sink.
Watchdog	Programmable Timeout, Generates processor NMI
Sync. Serial	Full duplex Independent Rx and Tx clocks, Master or Slave operating mode
Supply Voltage	5V DC ±5%
Supply Current	400mA(nominal)
Humidity	5 - 90%, non-condensing
Temperature	-20 to +85 °C
Weight	1.6oz (45 gm)
Dimensions	3.75" x 2.50" x 0.63" (95mm x 63.5mm x 16mm)

I/O Port DC Characteristics

Symbol	Parameter	MIN	MAX	Units	Condition
V_{IL}	Input Low	-0.3	0.8	V	
V_{IH}	Input High	2.0	Vcc+0.3	V	
V_{OL}	Output Low		0.45	V	I_{OL}= 8mA
V_{OH}	Output High	Vcc-0.5		V	I_{OH}= -8mA

Mating Connectors

Connector	Mfg	MFG P/N	JK micro P/N
J1,J3,J4,J5 (2x5)	Molex	22-55-2101	
	Oupiin	4072-2X05H	28-0030
Pins (for 2x5)	Molex	16-02-0096	
	Oupiin	4072-PIN-T	28-0033
J2,J8 (1x3)	Molex	22-01-2031	
	Oupiin	4071-3H	28-0012
J10 (1x8)	Molex	22-01-2081	
	Oupiin	4071-08H	28-0037
Pins (for 1x3, 1x8)	Molex	08-50-0114	
	Oupiin	4071-PIN-T	28-0013

Expansion Options

- 32 pin DIP socket to accept an additional
  512K Flash or
  512K SRAM or
  M-Systems DiskOnChip or
  Battery backed clock/calendar w/ 128K SRAM upgrade (20-0074)
- Serial bus (J5) for use with Multi-I/O or µI/O peripheral board

Appendix B Weather Station Specifications

Unit Dimensions 178 x 108 x 43 mm.
Uses 8 batteries 'AAA' 1.5v.
Including main adaptor 12v DC.
Weight: 315g.

Error! Unknown switch argument.

Temperature / Humidity
Thermometer
Temperature for in and Out
Measuring range In: O°C to +50°C, Out: -40°C to +60°C
Temperature alarm
Resolution: 0.1'C

Hygrometer
Relative humidity for In and Out
Measuring range In/Out: 10% to 97%
Humidity alarm
Resolution: 1%

Error! Unknown switch argument.

Wind Speed and Direction
Wind meter
Measuring range for wind speed: 0 to 56 m/s

Wind speed alarm
Resolution for wind speed: 0.2 m/s
Measuring range for wind direction: 0° to 359° * Wind speed measurement accuracy: ± 8°
Resolution for wind direction: 1°
Wind Chill
Measuring range -85°C to +60°C
Wind chill alarm
Resolution: 1°C

Rainfall
Rainfall counter
Measuring range: O to 9.999 mm
Resolution: 1 mm
Measuring time - accumulative: 24 hours
Measuring time: 1 impulse per 1mm rainfall (equivalent to 1 litre/m)

Barometer
Absolute barometric pressure or relative sea level barometric pressure
Measuring range: 795 mbar to 1050 mbar
Weather forecast indication by symbols
Pressure trend bar display: rising, steady, falling
Pressure alarm
Resolution: 1 mbar
Pressure measuring time: 15 minutes

Dew Point
Dew point for In and Out
Measuring range In/Out: 0°C to + 59°C
In/Out measurement accuracy: 25% to 90%: ±9°C
Dew point alarm
Resolution: 1°C

Appendix C Weather Station Data Protocol

Protocol of Oregon Scientific WX-918 Electronic Weather Station

See references for credits

Blank entries indicate undefined or unknown data.

Byte	Nibble	Bit(s)	Datum	Description 'part' of lo<format<hi unit @ resolution
8F. 0	HH	all	Group	8F ----------
8F. 1	DD	all	Time	Second 0<ab<59 @ 1
8F. 2	DD	all	Time	Minute 0<ab<59 @ 1
8F. 3	DD	all	Time	Hour 0<ab<23 @ 1
8F. 4	DD	all	Time	Day 1<ab<31 @ 1
8F. 5	Bx	0	Time	Format: 0=12 Hour, 1=24 Hour
8F. 5	Bx	1	Time	Format: 0=Day-Month, 1=Month-Day
8F. 5	Bx	2,3		
8F. 5	xH	all	Time	Month: 1=Jan, 2=Feb, ... B=Nov, C=Dec
8F. 6	DD	all	Time	Alarm: Minute
8F. 7	DD	all	Time	Alarm: Hour
8F. 8	DD	all	Humid	Indoor: 10<ab<97 % @ 1
8F. 9	DD	all	Humid	Indoor Hi: <ab> %
8F.10	DD	all	Humid	Indoor Hi: Minute
8F.11	DD	all	Humid	Indoor Hi: Hour
8F.12	DD	all	Humid	Indoor Hi: Day
8F.13	Dx	all	Humid	Indoor Lo: 'b' <ab> %
8F.13	xH	all	Humid	Indoor Hi: Month
8F.14	Dx	all	Humid	Indoor Lo: Minute 'b' of <ab>
8F.14	xD	all	Humid	Indoor Lo: 'a' of <ab> %
8F.15	Dx	all	Humid	Indoor Lo: Hour 'b' of <ab>
8F.15	xD	all	Humid	Indoor Lo: Minute 'a' of <ab>
8F.16	Dx	all	Humid	Indoor Lo: Day 'b' of <ab>
8F.16	xD	all	Humid	Indoor Lo: Hour 'a' of <ab>
8F.17	Hx	all	Humid	Indoor Lo: Month
8F.17	xD	all	Humid	Indoor Lo: Day 'a' of <ab>
8F.18	DD	all	Humid	Indoor Alarm Hi: <ab> %
8F.19	DD	all	Humid	Indoor Alarm Lo: <ab> %
8F.20	DD	all	Humid	Outdoor: 10<ab<97 % @ 1
8F.21	DD	all	Humid	Outdoor Hi: <ab> %
8F.22	DD	all	Humid	Outdoor Hi: Minute
8F.23	DD	all	Humid	Outdoor Hi: Hour
8F.24	DD	all	Humid	Outdoor Hi: Day
8F.25	Dx	all	Humid	Outdoor Lo: 'b' of <ab> %
8F.25	xH	all	Humid	Outdoor Hi: Month
8F.26	Dx	all	Humid	Outdoor Lo: Minute 'b' of <ab>
8F.26	xD	all	Humid	Outdoor Lo: 'a' of <ab> %
8F.27	Dx	all	Humid	Outdoor Lo: Hour 'b' of <ab>
8F.27	xD	all	Humid	Outdoor Lo: Minute 'a' of <ab>
8F.28	Dx	all	Humid	Outdoor Lo: Day 'b' of <ab>
8F.28	xD	all	Humid	Outdoor Lo: Hour 'a' of <ab>

8F.29	Hx	all	Humid	Outdoor Lo: Month
8F.29	xD	all	Humid	Outdoor Lo: Day 'a' of <ab>
8F.30	DD	all	Humid	Outdoor Alarm Hi: <ab> %
8F.31	DD	all	Humid	Outdoor Alarm Lo: <ab> %
8F.32	Bx	0	Humid	Outdoor: O.R. (out of range) = 1
8F.32	Bx	1		
8F.32	Bx	2	Humid	Indoor Hi: O.R. = 1
8F.32	Bx	3	Humid	Indoor: O.R. = 1
8F.32	xB	0-2		
8F.32	xB	3	Humid	Outdoor Hi: O.R. = 1
8F.33	Bx	0,1	Humid	Humidity Outdoors Alarm Set when both bits = 1
8F.33	Bx	2,3	Humid	Humidity Indoors Alarm Set when both bits = 1
8F.33	xB	0-2		
8F.33	xB	3	Time	Alarm Set = 1
8F.34	HH	all	Cksum	Unsigned sum of first 34 bytes
9F. 0	HH	all	Group	9F ----------
9F. 1	DD	all	Temp	Indoor: 'bc' of 0<ab.c<50 degrees C @ 0.1
9F. 2	Dx	all	Temp	Indoor Hi: 'c' of <ab.c> C
9F. 2	xB	0-2	Temp	Indoor: 'a' of <ab.c> C
9F. 2	xB	3	Temp	Indoor: Sign 0=+, 1=-
9F. 3	BD	0-2,all	Temp	Indoor Hi: 'ab' of <ab.c> C
9F. 3	Bx	3	Temp	Indoor Hi: Sign 0=+, 1=-
9F. 4	DD	all	Temp	Indoor Hi: Minute
9F. 5	DD	all	Temp	Indoor Hi: Hour
9F. 6	DD	all	Temp	Indoor Hi: Day
9F. 7	Dx	all	Temp	Indoor Lo: 'c' of <ab.c> C
9F. 7	xH	all	Temp	Indoor Hi: Month
9F. 8	BD	0-2,all	Temp	Indoor Lo: 'ab' of <ab.c> C
9F. 8	Bx	3	Temp	Indoor Lo: Sign 0=+, 1=-
9F. 9	DD	all	Temp	Indoor Lo: Minute
9F.10	DD	all	Temp	Indoor Lo: Hour
9F.11	DD	all	Temp	Indoor Lo: Day
9F.12	Dx	all	Temp	Indoor Alarm Hi: 'c' of 32<abc<122 deg F @ 1
9F.12	xH	all	Temp	Indoor Lo: Month
9F.13	BD	0,all	Temp	Indoor Alarm Hi: 'ab' of <abc> F
9F.13	Bx	1-3		
9F.14	DD	all	Temp	Indoor Alarm Lo: 'bc' of 32<abc<122 deg F @ 1
9F.15	Bx	0,1		
9F.15	Bx	2	Temp	Format: 0=degrees C, 1=degrees F
9F.15	Bx	3		
9F.15	xB	0	Temp	Indoor Alarm Lo: 'a' of <abc> F
9F.15	xB	1-3		
9F.16	DD	all	Temp	Outdoor: 'bc' of -40<ab.c<60 degrees C @ 0.1
9F.17	Dx	all	Temp	Outdoor Hi: 'c' of <ab.c> C
9F.17	xB	0-2	Temp	Outdoor: 'a' of <ab.c> C
9F.17	xB	3	Temp	Outdoor: Sign 0=+, 1=-
9F.18	BD	0-2,all	Temp	Outdoor Hi: 'ab' of <ab.c> C
9F.18	Bx	3	Temp	Outdoor Hi: Sign 0=+, 1=-
9F.19	DD	all	Temp	Outdoor Hi: Minute
9F.20	DD	all	Temp	Outdoor Hi: Hour
9F.21	DD	all	Temp	Outdoor Hi: Day
9F.22	Dx	all	Temp	Outdoor Lo: 'c' of <ab.c> C
9F.22	xH	all	Temp	Outdoor Hi: Month
9F.23	BD	0-2,all	Temp	Outdoor Lo: 'ab' of <ab.c> C
9F.23	Bx	3	Temp	Outdoor Lo: Sign 0=+, 1=-
9F.24	DD	all	Temp	Outdoor Lo: Minute
9F.25	DD	all	Temp	Outdoor Lo: Hour
9F.26	DD	all	Temp	Outdoor Lo: Day
9F.27	Dx	all	Temp	Outdoor Alarm Hi: 'c' of -40<abc<140 deg F @ 1
9F.27	xH	all	Temp	Outdoor Lo: Month

9F.28	BD	0,all	Temp	Outdoor Alarm Hi: 'ab' of <abc> F
9F.28	Bx	1,2		
9F.28	Bx	3	Temp	Outdoor Alarm Hi: Sign 0=+, 1=-
9F.29	DD	all	Temp	Outdoor Alarm Lo: 'bc' of <abc> F
9F.30	Bx	all		
9F.30	xB	0	Temp	Outdoor Alarm Lo: 'a' of <abc> F
9F.30	xB	1,2		
9F.30	xB	3	Temp	Outdoor Alarm Lo: Sign 0=+ 1=-
9F.31	BB	all		
9F.32	Bx	0,1	Temp	Temp Outdoors Alarm Set when both bits=1
9F.32	Bx	2,3	Temp	Temp Indoors Alarm Set when both bits=1
9F.32	xB	all		
9F.33	HH	all	Cksum	unsigned sum of first 33 bytes
AF. 0	HH	all	Group	AF ----------
AF. 1	DD	all	Barom	Local: 'cd' of 795<abcd<1050 mb @ 1
AF. 2	DD	all	Barom	Local: 'ab' of <abcd> mb
AF. 3	DD	all	Barom	SeaLevel: 'de' of 795<abcd.e<1050 mb @ 1
AF. 4	DD	all	Barom	SeaLevel: 'bc' of <abcd.e> mb
AF. 5	Bx	0,1	Barom	Format: 0=inches, 1=mm, 2=mb, 3=hpa
AF. 5	Bx	2,3		
AF. 5	xD	all	Barom	SeaLevel: 'a' of <abcd.e> mb
AF. 6	Bx	0-2	Barom	Trend: 1=Raising, 2=Steady, 4=Falling
AF. 6	Bx	3		
AF. 6	xB	all	Barom	Prediction: 1=Sunny, 2=Cloudy, 4=Partly, 8=Rain
AF. 7	DD	all	Dewpt	Indoor: 0<ab<47 degrees C @ 1
AF. 8	DD	all	Dewpt	Indoor Hi: <ab> C
AF. 9	DD	all	Dewpt	Indoor Hi: Minute
AF.10	DD	all	Dewpt	Indoor Hi: Hour
AF.11	DD	all	Dewpt	Indoor Hi: Day
AF.12	Dx	all	Dewpt	Indoor Lo: 'b' of <ab> C
AF.12	xH	all	Dewpt	Indoor Hi: Month
AF.13	Dx	all	Dewpt	Indoor Lo: Minute 'b' of <ab>
AF.13	xD	all	Dewpt	Indoor Lo: 'a' of <ab> C
AF.14	Dx	all	Dewpt	Indoor Lo: Hour 'b' of <ab>
AF.14	xD	all	Dewpt	Indoor Lo: Minute 'a' of <ab>
AF.15	Dx	all	Dewpt	Indoor Lo: Day 'b' of <ab>
AF.15	xD	all	Dewpt	Indoor Lo: Hour 'a' of <ab>
AF.16	Hx	all	Dewpt	Indoor Lo: Month
AF.16	xD	all	Dewpt	Indoor Lo: Day 'a' of <ab>
AF.17	Hx	all	Dewpt	Outdoor Alarm: 0=1 C ... F=16 C
AF.17	xH	all	Dewpt	Indoor Alarm: 0=1 C ... F=16 C
AF.18	DD	all	Dewpt	Outdoor: 0<ab<56 degrees C @ 1
AF.19	DD	all	Dewpt	Outdoor Hi: <ab> C
AF.20	DD	all	Dewpt	Outdoor Hi: Minute
AF.21	DD	all	Dewpt	Outdoor Hi: Hour
AF.22	DD	all	Dewpt	Outdoor Hi: Day
AF.23	Dx	all	Dewpt	Outdoor Lo: 'b' of <ab> C
AF.23	xH	all	Dewpt	Outdoor Hi: Month
AF.24	Dx	all	Dewpt	Outdoor Lo: Minute 'b' of <ab>
AF.24	xD	all	Dewpt	Outdoor Lo: 'a' of <ab> C
AF.25	Dx	all	Dewpt	Outdoor Lo: Hour 'b' of <ab>
AF.25	xD	all	Dewpt	Outdoor Lo: Minute 'a' of <ab>
AF.26	Dx	all	Dewpt	Outdoor Lo: Day 'b' of <ab>
AF.26	xD	all	Dewpt	Outdoor Lo: Hour 'a' of <ab>
AF.27	Hx	all	Dewpt	Outdoor Lo: Month
AF.27	xD	all	Dewpt	Outdoor Lo: Day 'a' of <ab>
AF.28	Bx	0	Dewpt	Outdoor Lo: O.R. = 1
AF.28	Bx	1		
AF.28	Bx	2	Dewpt	Outdoor: O.R. = 1
AF.28	Bx	3	Dewpt	Indoor Lo: O.R. = 1

AF.28	Bx	0		
AF.28	Bx	1	Dewpt	Indoor: O.R. = 1
AF.28	Bx	2,3		
AF.29	Bx	0		
AF.29	Bx	1,2	Dewpt	In and Out Alarm Set when both bits=1
AF.29	Bx	3	Barom	Alarm Set = 1
AF.29	xH	all	Barom	Alarm: 0=1mb ... F=16mb
AF.30	HH	all	Cksum	unsigned sum of first 30 bytes
BF. 0	HH	all	Group	BF ----------
BF. 1	DD	all	Rain	Rate: 'bc' of 0<abc<998 mm/hr @ 1
BF. 2	Bx	all		
BF. 2	xD	all	Rain	Rate: 'a' of <abc> mm/hr
BF. 3	DD	all	Rain	Yesterday: 'cd' of 0<abcd<9999 mm @ 1
BF. 4	DD	all	Rain	Yesterday: 'ab' of <abcd> mm
BF. 5	DD	all	Rain	Total: 'cd' of <abcd> mm
BF. 6	DD	all	Rain	Total: 'ab' of <abcd> mm
BF. 7	DD	all	Rain	Reset: Minute
BF. 8	DD	all	Rain	Reset: Hour
BF. 9	DD	all	Rain	Reset: Day
BF.10	Bx	0		
BF.10	Bx	1	Rain	Format: 0=mm, 1=inches
BF.10	Bx	2,3		
BF.10	xH	all	Rain	Reset: Month
BF.11	DD	all	Rain	Alarm: 'bc' of 0<ab.c<39.3 in/hr @ 0.1
BF.12	Bx	0	Rain	Alarm Set = 1
BF.12	Bx	1-2		
BF.12	Bx	3	Rain	Rate: O.R. = 1
BF.12	xD	all	Rain	Rate Alarm: 'a' of <ab.c> in/hr
BF.13	HH	all	Cksum	Unsigned sum of first 13 bytes
CF. 0	HH	all	Group	CF ----------
CF. 1	DD	all	Wind	Gust Speed: 'bc' of 0<ab.c<56 m/s @ 0.2
CF. 2	Dx	all	Wind	Gust Dir: 'c' of 0<abc<359 degrees @ 1
CF. 2	xD	all	Wind	Gust Speed: 'a' of <ab.c> m/s
CF. 3	DD	all	Wind	Gust Dir: 'ab' of <abc>
CF. 4	DD	all	Wind	Avg Speed: 'bc' of 0<ab.c<56 m/s @ 0.1
CF. 5	Dx	all	Wind	Avg Dir: 'c' of <abc>
CF. 5	xD	all	Wind	Avg Speed: 'a' of <ab.c> m/s
CF. 6	DD	all	Wind	Avg Dir: 'ab' of <abc>
CF. 7	DD	all	Wind	Hi Speed: 'bc' of <ab.c> m/s
CF. 8	Dx	all	Wind	Hi Dir: 'c' of <abc>
CF. 8	xD	all	Wind	Hi Speed: 'a' of <ab.c> m/s
CF. 9	DD	all	Wind	Hi Dir: 'ab' of <abc>
CF.10	DD	all	Wind	Hi: Minute
CF.11	DD	all	Wind	Hi: Hour
CF.12	DD	all	Wind	Hi: Day
CF.13	Dx	all	Wind	Alarm: 'c' of 0<abc<125 mph @ 1
CF.13	xH	all	Wind	Hi: Month
CF.14	Bx	1-3		
CF.14	BD	0,all	Wind	Alarm: 'ab' of <abc> mph
CF.15	Bx	1		
CF.15	Bx	2,3	Wind	Format: 0=mph, 1=knots, 2=m/s, 3=kph
CF.15	xB	all		
CF.16	DD	all	Chill	Temp: -85<ab<60 degrees C @ 1
CF.17	DD	all	Chill	Lo: <ab> C
CF.18	DD	all	Chill	Lo: Minute
CF.19	DD	all	Chill	Lo: Hour
CF.20	DD	all	Chill	Lo: Day
CF.21	Bx	0	Chill	Lo: Sign 0=+, 1=-
CF.21	Bx	1	Chill	Temp: Sign 0=+, 1=-
CF.21	Bx	2,3		

CF.21	xH	all	Chill	Lo: Month
CF.22	DD	all	Chill	Alarm: 'bc' of -121<abc<140 degrees F @ 1
CF.23	Bx	0	Chill	Alarm: 'a' of <abc> F
CF.23	Bx	1		
CF.23	Bx	2	General	Power Source 0=AC, 1=DC
CF.23	Bx	3	General	Low Battery Indicator = 1
CF.23	xB	0-2		
CF.23	xB	3	Chill	Alarm: Sign 0=+, 1=-
CF.24	Bx	0-2	General	Display Selected: 0=Time, 1=Temp ... 7=Rain
CF.24	Bx	3		
CF.24	xB	0,1	General	Display Subscreen: 0=first ... 3=fourth
CF.24	xB	2,3	General	Display: 0=main, 1=mem, 2=alarm.in, 3=alarm.out
CF.25	Bx	0		
CF.25	Bx	1	Wind	Hi Speed: O.R. = 1
CF.25	Bx	2	Wind	Avg Speed: O.R. = 1
CF.25	Bx	3	Wind	Gust Speed: O.R. = 1
CF.25	xB	0		
CF.25	xB	1	Chill	Alarm Set = 1
CF.25	xB	2	Wind	Alarm Set = 1
CF.25	xB	3		
CF.26	HH	all	Cksum	Unsigned sum of first 26 bytes

==========

Nibble Column:
D-> 4 bit decimal number Range: 0-9
H-> 4 bit hex number Range: 0-15
B-> Bit encoded value Range: Variable
x-> Not defined in this entry

Bits Column:
Bits within defined Nibbles
0 - Lo order
3 - Hi order

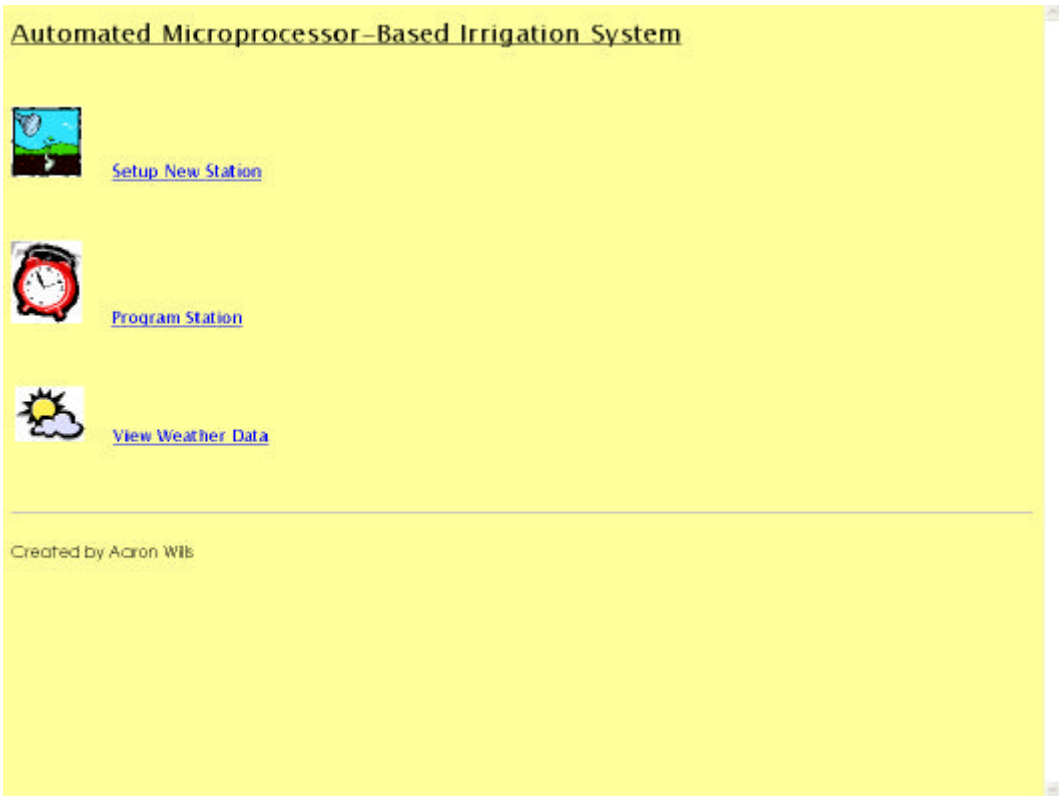
General:
All data is sent in the units shown and is independent of the units selected.
Data is sent 9600 baud 8n1.
Sensors that are not functioning or are out of range will return 'EE' as the measurement.

Cksum:
The last byte in each group is a checksum of that group. It is calculated by performing an unsigned add of all the bytes in the group, including the group number but not the checksum byte. The checksum is the lo-order byte of the sum.

Group Info

Group Number	Length (Bytes)	Report Interval	Contents Summary
8F	35	10 Sec	Time, Humidity
9F	34	10 Sec	Temperature
AF	31	10 Sec	Barometer, Dew Point
BF	14	10 Sec	Rain
CF	27	5 Sec	Wind, Wind Chill, General

Appendix D Web-based User Interface




Station Programming
Station Number 1
Start day :
Day of Week Monday
Every days
Start time :
AM
Duration :
minutes
Rainfall/Evaporation Compensation
Auto Manual
Send New Program
Back To Home


Conditions as of: 03:59:00, 01 September
24 Hour Trend
Current Min Max Prev Min Prev Max Units
Humidity: 69 49 70 10 97 %
Temperature: 15.8 15.7 20.5 11.6 28.5 Deg C
Wind Chill: 16 16 21 12 28.5 Deg C
Dew Point: 10 Deg C
Pressure: 1013.0 and Steady mB
Rain Rate: 0 mm/hr Today: 0 Prev Total: 0 mm
Total: 86 Since 14:29, 05 March mm
Wind Gust: Calm @ Calm Deg Hi Gust: 1.8 Prev Hi Gust: 3.0 m/s
Avg Speed: Calm @ Calm Deg m/s
Weather data is collected from an Oregon Scientific WMR918 weather station by a
µFlashTCP that produces and serves the web pages.

Appendix E Hargreaves / Samani Evapotranspiration Model

Evapotranspiration (ET) from a well-watered grass has long been used as a reference value for estimating crop consumptive use.

The Hargreaves model is empirical in nature and with some recent modifications (Hargreaves and Samani, 1982) takes the form:

Error! Unknown switch argument. (E.1)

where PET is the potential evapotranspiration rate (mm d^{-1}), R_a is the total incoming extraterrestrial solar radiation in the same units as evaporation), C_t is a temperature reduction coefficient which is a function of relative humidity (w_a), δ_t is the difference between the mean monthly maximum and mean monthly minimum temperatures (^oF), and $T_{avg.d}$ is the mean temperature (^oF) in the time step. A relationship between the temperature reduction coefficient and the relative humidity has been regressed from measurements made at 18 locations in the United States to account for the reduction in PET with increased relative humidity:

Error! Unknown switch argument. (E.2)

The following empirical simplifications permit the use of the formula with the sole input of temperature data, latitude (ϕ in degrees), and the Julian day (J) to estimate incoming solar energy (Duffie and Beckman, 1980):

Error! Unknown switch argument. (E.3)

where d_r is the relative distance between the earth and the sun given by:

Error! Unknown switch argument. (E.4)

δ is the solar declination (radians) defined by:

Error! Unknown switch argument. (E.5)

and w_s is the sunset hour angle (radians) given by:

Error! Unknown switch argument. (E.6)

With these modifications, the Hargreaves equation is more universally applicable, as it does not require the observed solar input.

A number of independent investigations have compared the estimates of evapotranspiration from different models. The Hargreaves equation consistently produces accurate estimates of potential evapotranspiration (as measured using energy

balance techniques, the Penman combination equation, or lysimetric observations), and in some cases, much better than estimates made using other methods (Hargreaves and Samani, 1982; Mohan, 1991; Saeed, 1986). Mohan (1991) found the Hargreaves equation to have a high correlation with the Penman combination equation for estimates of average weekly evapotranspiration in humid regions.

The reason for the success with such an empirical model is because of the theory which it reflects. In a comparison with the Penman combination equation, the model considers the following: the incoming solar energy (R_a), the average amount of energy removed in the form of sensible heat from the amount available for evaporation (δ_t), an approximation of the ratio of $s(T_a)$ to the sum of $s(T_a)$ and γ by using the temperature (T), and a reduction in the driving gradient when the vapour pressure deficit is small (C_t).

```
function Samani(maxtemp, mintemp, date, radian){

        var delim_num, j=0;


        // Move strings over to numbers using eval
        kt = eval(document.retrive.kt.value);


  //Get date from var
  delim_num = date.indexOf('/');
  month = date.substring(0,delim_num);
  date = date.substring(delim_num+1, date.length);
  delim_num = date.indexOf('/');
  day = date.substring(0,delim_num);
  date = date.substring(delim_num+1, date.length);
  year = date.substring(0,date.length);


        // Convert strings to numbers
        month = eval(month);
        day = eval(day);
        year = eval(year);


//   Convert deg F to deg C if nessary
      if(document.retrive.temp_type[0].checked){
            maxtemp = FtoC(maxtemp);
            mintemp = FtoC(mintemp);
      }

//   Samani's equation

                  with(Math){

        j    = month -.5 + (day/30.5);

        tc   = (maxtemp + mintemp)/2;

        td   = maxtemp - mintemp;

        dec1 = -0.00117 - 0.40117 * cos(PI * j/6);
```

```
        dec2 = -0.042185 * sin(PI * j/6) + 0.00163 * cos(PI * j/3);
        dec  = dec1 + dec2 + 0.00208 * sin(PI * j/3);
        es   = 1.00016 - 0.032126 * cos(PI * j/6) - 0.003354 * sin(PI * j/6);
        tmpvr1= -tan(radian);
        tmpvr2= tan(dec);
        tmpvr3= tmpvr1 * tmpvr2;
        om   = acos(tmpvr3);
        ra1  = 916.732 * (om * sin(radian) * sin(dec) + cos(radian) * cos(dec) * sin(om));
        ra   = ra1/es * 10/(596-0.55 * tc);

        if(td < 0) { alert("Data error for the following line: Mintemp is greater than the Maxtemp\n");}
        else{
                                pet   = (0.0162 * kt) * ra * sqrt(td) * (tc + 17.8);
                        }
                }
                // pet is in mm convert to in
                pet = (pet/10) * .39;
    return pet;
}
```

Appendix F Weather Station Data Extraction Routine

```
/* using _bios_serialcom example (for COM1) */
#include <stdlib.h>
#include <bios.h>
#include <stdio.h>
#include <dos.h>
#include <string.h>
#include <math.h>

#define DTR        0x01    // Data Terminal Ready
#define RTS        0x02    // Ready To Send
#define COM1PORT    0x0000  // Pointer to Location of COM1 port
#define COM2PORT    0x0002  // Pointer to Location of COM2 port
#define COM1        0
#define COM2        1
#define DATA_READY  0x100
#define FALSE       0
#define TRUE            !FALSE  //Had to add an underscore to TRUE, it may already be defined

#define SETTINGS ( 0xE0 | 0x00 | 0x02 | 0x00)  // 9600,N,7,1

int intNNA, intNNB, intBufferCount, intChkSum, i;
int intChkSum8F, intChkSum9F, intChkSumAF, intChkSumBF, intChkSumCF;
int intChkSumWG, intWDay, intWMonth, intHumMonth, intTsign, intTOutlook;
int intWindGustDir, intWindAvDirection, intWindHiDirection;
int intRainYesterday, intRainTotal;
int NextInt;

char * strNchar, * strStartChar, * strEventMsg, * strErrorMsg;
char * strG8, * strG9, * strGA, * strGB, * strGC;
char * strCsum8F, * strCsum9F, * strCsumAF, * strCsumBF, * strCsumCF;
char * strTempChkSum;
char * strMainString, * strWindString;
char * strWgroup, * strWgCsum, * strTime;
char * strOutSign, * strTHMonth, * strTLMonth, * strOmaxSign;
char * strOminSign, * strInVsign, * strBarTrend, * strDPSign;
char * strRmonth, * strWCSign, * strOutHumTime,* strInHumidity, * strOsTempHiTime;
char * strOutHumidity,* strOutHumHi,* strOutHumDay, * strOsTempHiDate;
char * strOsTempLoTime, * strOsTempLoDate, * strBarometer, * strOutlook;
char * strTotalRainTime, * strTotalRainDate, * strWindHiTime, * strWindHiDate;

float sngDewPoint, sngRainRate, sngTmpRain, sngWindHiGust, sngOsTempHi;
float sngWindGustSpeed, sngWindAvSpeed, sngOutsideTemp, sngOsTempLo;
float sngInsideTemp, sngWindChill;

int blnGoodData;      /* boolean value  */

/* arrays for serial data  */
char NextChar;
unsigned short int byteRawDataArray[142];

/* arrays for the data groups  */
char WindGroupArray[54];
char Group8FArray[70];
char Group9FArray[68];
char GroupAFArray[62];
char GroupBFArray[28];
char GroupCFArray[54];

/* Procedures to store the serial data into arrays  */
void ChopFullData();
void ChopWindData();
```

```
// Procedure to convert an integer into a hexadecimal string
// char * Hex(int);

// Procedures to decode the data groups
void Decode8FGroup();
void Decode9FGroup();
void DecodeAFGroup();
void DecodeBFGroup();
void DecodeCFGroup();
void DecodeWindOnlyGroup();

// Procedure to store the weather data into a file w_full.txt
int StoreFullData(void);

// Procedure to store the wind data into a file w_wind.txt
int StoreWindData(void);

// Function to turn a hexadecimal string into an integer
int axtoi(char *hexStg, int numofchars);

// Function to convert an integer string into a value
int aistoi(char *hexStg, int numchars);

// ****************************************************************
int main( void )

{

int intBufferCount, status, DONE = FALSE;
                // in,
                // out,
int far *RS232_Addr;
/*  Determine port location of COM1.
                 0x40:0x00 = COM1 I/O port address
                 0x40:0x02 = COM2 I/O port address
*/
 RS232_Addr = (int far *) MK_FP( 0x0040, COM1PORT );
 if( !*RS232_Addr )
                  return -1;

 _bios_serialcom( 0, COM1, SETTINGS );

while( !DONE )
 {
                /*  Reset DTR and RTS to prepare for send/receive of
 next character.
                */
                outportb( *RS232_Addr + 4, DTR | RTS );

                /*  Get status of com port.
                */
                status =  _bios_serialcom( 3, COM1, 0 );
                intBufferCount = 0;
                if( status & DATA_READY )

                //  There's a character on the port.  Get it and put into string.

                /* Either we receive 27 bytes or 141 bytes and the first character must be
                either CF or 8F  */
                while(( NextInt =_bios_serialcom( 2, COM1, 0 ) & 0x7F) != 0)
                                // putch( NextInt ); /*- displays to screen characters read */
                                {
                                byteRawDataArray[intBufferCount] = NextInt;
                                intBufferCount++;
                                }
                 /* If there are 27 characters in the buffer then put them into a
                 byte array */
                 itoa(byteRawDataArray[0], strStartChar, 16);
                 if( intBufferCount == 27 )
                                itoa(byteRawDataArray[0], strStartChar, 16);
                                else
                 /* Must have got more than 27 bytes so should have 141 bytes   */
                                {
                                if( intBufferCount == 140 )
                                                {
                                                /* for (i=0; i<=intBufferCount; i++)
```

```
                                        {
                                        (*byteRawDataArray)[i] = (unsigned short int) strInputBuffer[i];
                                        }     *strStartChar    */
                                itoa(byteRawDataArray[0], strStartChar, 16);
                                }
                        }
                /* Call the appropriate routine depending on whether the first character
                of the data was a CF or an 8F */
                if(strStartChar == "cf")
                        {
                        ChopWindData();
                        StoreWindData();
                        }
                        else if(strStartChar == "8f")
                                        {
                                        ChopFullData();
                                        StoreFullData();
        }
  }
;

return 0;
}

// ****************************************************************
void ChopFullData()

/* This routine chops up the full (141 bytes) data and puts it into arrays for
each of the data groups  */

/* These are strings which hold raw data extracted from the byteRawDataArray
array  */
{

strG8 = "";
strG9 = "";
strGA = "";
strGB = "";
strGC = "";

// intChkSum equals the decimal value of strStartChar
intChkSum = atoi(strStartChar);

/* We've already got the first character (8F) so get the next 34 for a total
of 35 for the 8F group  */
for (intNNA = 1; intNNA < 35 ; intNNA++)
                {
                intNNB = byteRawDataArray[intNNA];
                intChkSum = intChkSum + intNNB;
                if(intNNA == 34)
                        intChkSum8F = intChkSum - intNNB;
                itoa(intNNB, strNchar, 16);
                if(strlen(strNchar) == 1)
                        strNchar = strcat("0", strNchar);
                strcat(strG8, strNchar);
                }
strcat(strG8, "8f");

// Get the next 34 characters for the 9F group
intChkSum = 0;
for (intNNA = 35; intNNA < 69 ; intNNA++)
                {
                intNNB = byteRawDataArray[intNNA];
                intChkSum = intChkSum + intNNB;
                if(intNNA == 68)
                        intChkSum9F = intChkSum - intNNB;
                itoa(intNNB, strNchar, 16);
                if(strlen(strNchar) == 1)
                        strNchar = strcat("0", strNchar);
                strcat(strG9, strNchar);
                }

// Get the next 31 characters for the AF group
intChkSum = 0;
for (intNNA = 69; intNNA < 100 ; intNNA++)
                {
```

```
		intNNB = byteRawDataArray[intNNA];
		intChkSum = intChkSum + intNNB;
		if(intNNA == 99)
			intChkSumAF = intChkSum - intNNB;
		itoa(intNNB, strNchar, 16);
		if(strlen(strNchar) == 1)
			strNchar = strcat("0", strNchar);
		strcat(strGA, strNchar);
		}

// Get the next 14 characters for the BF group
intChkSum = 0;
for (intNNA = 100; intNNA < 114 ; intNNA++)
		{
		intNNB = byteRawDataArray[intNNA];
		intChkSum = intChkSum + intNNB;
		if(intNNA == 113)
			intChkSumBF = intChkSum - intNNB;
		itoa(intNNB, strNchar, 16);
		if(strlen(strNchar) == 1)
			strNchar = strcat("0", strNchar);
		strcat(strGB, strNchar);
		}

// Get the next 27 characters for the CF group
intChkSum = 0;
for (intNNA = 114; intNNA < 141 ; intNNA++)
		{
		intNNB = byteRawDataArray[intNNA];
		intChkSum = intChkSum + intNNB;
		if(intNNA == 140)
			intChkSumCF = intChkSum - intNNB;
		itoa(intNNB, strNchar, 16);
		if(strlen(strNchar) == 1)
			strNchar = strcat("0", strNchar);
		strcat(strGC, strNchar);
		}

/* Now extract the checksums from the groups (the last two
characters in each */

itoa(intChkSum8F, strTempChkSum, 16);
strCsum8F[0] = strTempChkSum[strlen(strTempChkSum) - 2];
strCsum8F[1] = strTempChkSum[strlen(strTempChkSum) - 1];
itoa(intChkSum9F, strTempChkSum, 16);
strCsum9F[0] = strTempChkSum[strlen(strTempChkSum) - 2];
strCsum9F[1] = strTempChkSum[strlen(strTempChkSum) - 1];
itoa(intChkSumAF, strTempChkSum, 16);
strCsumAF[0] = strTempChkSum[strlen(strTempChkSum) - 2];
strCsumAF[1] = strTempChkSum[strlen(strTempChkSum) - 1];
itoa(intChkSumBF, strTempChkSum, 16);
strCsumBF[0] = strTempChkSum[strlen(strTempChkSum) - 2];
strCsumBF[1] = strTempChkSum[strlen(strTempChkSum) - 1];
itoa(intChkSumCF, strTempChkSum, 16);
strCsumCF[0] = strTempChkSum[strlen(strTempChkSum) - 2];
strCsumCF[1] = strTempChkSum[strlen(strTempChkSum) - 1];

strMainString = "";
// Check the checksum for the 8F group
strTempChkSum[0] = strG8[strlen(strG8) - 2];
strTempChkSum[1] = strG8[strlen(strG8) - 1];
if(strTempChkSum != strCsum8F)
		{
		blnGoodData = FALSE;
		goto NoFurtherAction;
		}

// Check the checksum for the 9F group
strTempChkSum[0] = strG9[strlen(strG9) - 2];
strTempChkSum[1] = strG9[strlen(strG9) - 1];
if(strTempChkSum != strCsum9F)
		{
		blnGoodData = FALSE;
		goto NoFurtherAction;
		}
```

```
// Check the checksum for the AF group
strTempChkSum[0] = strGA[strlen(strGA) - 2];
strTempChkSum[1] = strGA[strlen(strGA) - 1];
if(strTempChkSum != strCsumAF)
        {
        blnGoodData = FALSE;
        goto NoFurtherAction;
        }

// Check the checksum for the BF group
strTempChkSum[0] = strGB[strlen(strGB) - 2];
strTempChkSum[1] = strGB[strlen(strGB) - 1];
if(strTempChkSum != strCsumBF)
        {
        blnGoodData = FALSE;
        goto NoFurtherAction;
        }

// Check the checksum for the CF group
strTempChkSum[0] = strGC[strlen(strGC) - 2];
strTempChkSum[1] = strGC[strlen(strGC) - 1];
if(strTempChkSum != strCsumCF)
        {
        blnGoodData = FALSE;
        goto NoFurtherAction;
        }

// If we get to this point then all the checksums must be okay so
blnGoodData = TRUE;

// Put the data for each group into an array for each group
for (intNNA = 1; intNNA <= strlen(strG8); intNNA++)
        Group8FArray[intNNA] = strG8[intNNA];
for (intNNA = 1; intNNA <= strlen(strG9); intNNA++)
        Group9FArray[intNNA] = strG9[intNNA];
for (intNNA = 1; intNNA <= strlen(strGA); intNNA++)
        GroupAFArray[intNNA] = strGA[intNNA];
for (intNNA = 1; intNNA <= strlen(strGB); intNNA++)
        GroupBFArray[intNNA] = strGB[intNNA];
for (intNNA = 1; intNNA <= strlen(strGC); intNNA++)
        GroupCFArray[intNNA] = strGC[intNNA];

NoFurtherAction:
/* The arrays Group8FArray to GroupCFArray contain the data for
the respective groups.  We have 5 seconds to process these arrays
before more data arrives.        If blnGoodData is True then we call the
extracting and handling routines */

if(blnGoodData == TRUE)
        {
        Decode8FGroup;
        Decode9FGroup;
        DecodeAFGroup;
        DecodeBFGroup;
        DecodeCFGroup;
        }

// End of ChopFullData routine
}

// *********************************************************************
void ChopWindData()

/* This routine chops up the wind only (27 bytes) data and puts it
in an array.  Note that the array is 0 to 26 which equals 27 bytes */
{
intChkSum = axtoi(strStartChar, 1);

// If the array is the wrong size then abandon this routine
int intArraySize = sizeof(byteRawDataArray);
if(intArraySize != 27)
        {
        blnGoodData = FALSE;
        goto NoAction;
        }
```

```
for (intNNA = 1; intNNA < 27 ; intNNA++)
        {
        intNNB = byteRawDataArray[intNNA];
        intChkSum = intChkSum + intNNB;
        itoa(intNNB, strNchar, 16);
        if(strlen(strNchar) == 1)
                strNchar = strcat("0", strNchar);
        strcat(strWindString, strNchar);
        }
intChkSumWG = intChkSum - intNNB;
strWgroup = "cf";
strcat(strWgroup, strWindString);
itoa(intChkSumWG, strTempChkSum, 16);
strWgCsum[0] = strTempChkSum[(strlen(strTempChkSum)) - 2];
strWgCsum[1] = strTempChkSum[(strlen(strTempChkSum)) - 1];
strWindString = "";
// Check the checksums
strTempChkSum[0] = strWgroup[(strlen(strWgroup)) - 2];
strTempChkSum[1] = strWgroup[(strlen(strWgroup)) - 1];
if(strTempChkSum != strWgCsum)
        {
        blnGoodData = FALSE;
        goto NoAction;
        }
        else
                blnGoodData = TRUE;

// Put the data into an array
for(intNNA = 1; intNNA < strlen(strWgroup); intNNA++)
        WindGroupArray[intNNA] = strWgroup[intNNA];

NoAction:
/* The array WindGroupArray[] holds the data for the wind only data
group. We have 5 seconds before more data arrives so process the
current data unless it is corrupt */
if(blnGoodData == TRUE)
        DecodeWindOnlyGroup;

// End of ChopWindData routine
}

// ********************************************************************
void Decode8FGroup()

{
char * strTempString = "";

// Minute, hour, day and month from the WM-918
strTime = "";
strTime[0] = Group8FArray[7];
strTime[1] = Group8FArray[8];
strTime[2] = ':';
strTime[3] = Group8FArray[5];
strTime[4] = Group8FArray[6];
strTime[5] = ':';
strTime[6] = Group8FArray[3];
strTime[7] = Group8FArray[4];
/* The following line converts the numerical characters stored in
the array into a value, with the second variable being the number
of characters read.   */
strTempString = "";
strTempString[0] = Group8FArray[9];
strTempString[1] = Group8FArray[10];
intWDay = atoi(strTempString);
strTempString = "";
strTempString[0] = Group8FArray[12];
intWMonth = axtoi(strTempString, 1);

// Get the current outside humidity value
strOutHumidity = "";
strOutHumidity[0] = Group8FArray[41];
strOutHumidity[1] = Group8FArray[42];

// Get the outside humidity highest recorded value
strOutHumHi ="";
strOutHumHi[0] = Group8FArray[43];
```

```
strOutHumHi[1] = Group8FArray[44];

// Get the outside humidity highest recorded time hour and minute
strOutHumTime = "";
strOutHumTime[0] = Group8FArray[47];
strOutHumTime[1] = Group8FArray[48];
strOutHumTime[2] = ':';
strOutHumTime[3] = Group8FArray[45];
strOutHumTime[4] = Group8FArray[46];

// Get the outside humidity highest recorded day
strOutHumDay = "";
strOutHumDay[0] = Group8FArray[49];
strOutHumDay[1] = Group8FArray[50];

// Get the outside humidity highest recorded month
strTempString = "";
strTempString[0] = Group8FArray[52];
intHumMonth = axtoi(strTempString, 1);

// Indoor humidity
strInHumidity = "";
strInHumidity[0] = Group8FArray[17];
strInHumidity[1] = Group8FArray[18];

// End of Decode8FArray

}

// *********************************************************************
void Decode9FGroup()

{
int intTempInt = 0;
float sngTempFloat = 0;
char * strTempString = "";

// Get the current temperature and sign
strTempString = "";
strTempString[0] = Group9FArray[36];
intTsign = axtoi(strTempString, 1);
intTsign = intTsign & 0x8;
if(intTsign == 8)
        strOutSign = "-";
        else
                strOutSign = "+";

// Get temperature
strTempString = "";
strTempString[0] = Group9FArray[36];
intTempInt = axtoi(strTempString, 1);
intTempInt = intTempInt & 0x7;
intTempInt = intTempInt * 10;
sngOutsideTemp = intTempInt;
strTempString = "";
strTempString[0] = Group9FArray[33];
sngOutsideTemp += axtoi(strTempString, 1);
strTempString = "";
strTempString[0] = Group9FArray[34];
intTempInt = axtoi(strTempString, 1);
sngTempFloat = (float)intTempInt / 10.0;
sngOutsideTemp += sngTempFloat;

// Get outdoor temperature highest recorded sign
strTempString = "";
strTempString[0] = Group9FArray[37];
intTsign = axtoi(strTempString, 1);
intTsign = intTempInt & 0x8;

if(intTsign == 8)
        strOmaxSign = "-";
        else
                strOmaxSign = "+";

// Get outdoor temperature highest recorded value
strTempString = "";
```

```
strTempString[0] = Group9FArray[37];
intTempInt = axtoi(strTempString, 1);
intTempInt = intTempInt & 0x7;
intTempInt = intTempInt * 10;
sngOsTempHi = intTempInt;
strTempString = "";
strTempString[0] = Group9FArray[38];
sngOsTempHi += axtoi(strTempString, 1);
strTempString = "";
strTempString[0] = Group9FArray[35];
intTempInt = axtoi(strTempString, 1);
sngTempFloat = (float)intTempInt / 10.0;
sngOsTempHi += sngTempFloat;

// Get outdoor temperature highest recorded time
strOsTempHiTime = "";
strOsTempHiTime[0] = Group9FArray[41];
strOsTempHiTime[1] = Group9FArray[42];
strOsTempHiTime[2] = ':';
strOsTempHiTime[3] = Group9FArray[39];
strOsTempHiTime[4] = Group9FArray[40];

// Get outdoor temperature highest recorded date
strOsTempHiDate = "";
strOsTempHiDate[0] = Group9FArray[43];
strOsTempHiDate[1] = Group9FArray[44];

// Get outdoor temperature highest recorded month
strTempString = "";
strTempString[0] = Group9FArray[46];
intTempInt = axtoi(strTempString,1);
itoa(intTempInt, strTHMonth, 10);

// Get the outdoor temperature lowest recorded sign
strTempString = "";
strTempString[0] = Group9FArray[47];
intTsign = axtoi(strTempString, 1);
intTsign = intTempInt & 0x8;
if(intTsign == 8)
        strOminSign = "-";
        else
                strOminSign = "+";

// Get outdoor temperature lowest recorded value
strTempString = "";
strTempString[0] = Group9FArray[47];
intTempInt = axtoi(strTempString, 1);
intTempInt = intTempInt & 0x7;
intTempInt = intTempInt * 10;
sngOsTempLo = intTempInt;
strTempString = "";
strTempString[0] = Group9FArray[48];
sngOsTempLo += axtoi(strTempString, 1);
strTempString = "";
strTempString[0] = Group9FArray[45];
intTempInt = axtoi(strTempString, 1);
sngTempFloat = (float)intTempInt / 10.0;
sngOsTempLo += sngTempFloat;

// Get outdoor temperature lowest recorded time
strOsTempLoTime = "";
strOsTempLoTime[0] = Group9FArray[51];
strOsTempLoTime[1] = Group9FArray[52];
strOsTempLoTime[2] = ':';
strOsTempLoTime[3] = Group9FArray[49];
strOsTempLoTime[4] = Group9FArray[50];

// Get outdoor temperature lowest recorded date
strOsTempLoDate = "";
strOsTempLoDate[0] = Group9FArray[53];
strOsTempLoDate[1] = Group9FArray[54];

// Get outdoor temperature lowest recorded month
strTempString = "";
strTempString[0] = Group9FArray[56];
intTempInt = axtoi(strTempString,1);
```

```
itoa(intTempInt, strTLMonth, 10);

// Get the current indoor temperature and sign
strTempString = "";
strTempString[0] = Group9FArray[6];
intTsign = axtoi(strTempString, 1);
intTsign = intTempInt & 0x8;
if(intTsign == 8)
        strInVsign = "-";
        else
                strInVsign = "+";

// get indoor temp
strTempString = "";
strTempString[0] = Group9FArray[6];
intTempInt = axtoi(strTempString, 1);
intTempInt = intTempInt & 0x7;
intTempInt = intTempInt * 10;
sngInsideTemp = intTempInt;
strTempString = "";
strTempString[0] = Group9FArray[3];
sngInsideTemp += axtoi(strTempString, 1);
strTempString = "";
strTempString[0] = Group9FArray[4];
intTempInt = axtoi(strTempString, 1);
sngTempFloat = (float)intTempInt / 10.0;
sngInsideTemp += sngTempFloat;

// End of Decode9FGroup
}

// ****************************************************************
void DecodeAFGroup()

{
char * strTempString = "";

// Get the current barometer value
strBarometer = "";
strBarometer[0] = GroupAFArray[12];
strBarometer[1] = GroupAFArray[9];
strBarometer[2] = GroupAFArray[10];
strBarometer[3] = GroupAFArray[7];
strBarometer[4] = '.';
strBarometer[5] = GroupAFArray[8];

// Get the barometer trend
switch (GroupAFArray[13])
        {
        case 1 : strBarTrend = "Rising"; break;
        case 2 : strBarTrend = "Steady"; break;
        case 4 : strBarTrend = "Falling"; break;
        }

// Get the current dewpoint in Celcius
strTempString = "";
strTempString[0] = GroupAFArray[37];
strTempString[1] = GroupAFArray[38];
sngDewPoint = atoi(strTempString);

// Get the Dewpoint sign
if(sngDewPoint >= 0)
        strDPSign = "+";
        else
                strDPSign = "-";

// Get the current outlook
// ********** Might have to create new routine for binary conversion
strTempString = "";
strTempString[0] = GroupAFArray[14];
switch (axtoi(strTempString, 1))   // Evaluate the forecast
        {
        case 1 : strOutlook = "Fine"; break;
        case 2 : strOutlook = "Cloudy"; break;
        case 4 : strOutlook = "Part Cloudy"; break;
        case 8 : strOutlook = "Rain"; break;
```

```
            }

// End of DecodeAFGroup
}

// ****************************************************************
void DecodeBFGroup()

{
int intTempInt = 0;
char * strTempString = "";
char chTempChar;

// Get the current rainfall rate
chTempChar = GroupBFArray[6] & 0xf;
sngRainRate = axtoi(&chTempChar, 1) * 100;
sngRainRate += axtoi(&GroupBFArray[3],2);

// Get the rainfall yesterday (from the weather station)
strTempString = "";
strTempString[0] = GroupBFArray[9];
strTempString[1] = GroupBFArray[10];
strTempString[2] = GroupBFArray[7];
strTempString[3] = GroupBFArray[8];
intRainYesterday = atoi(strTempString);

// Get the rainfall total (since last WM -918 reset)
strTempString = "";
strTempString[0] = GroupBFArray[13];
strTempString[1] = GroupBFArray[14];
strTempString[2] = GroupBFArray[11];
strTempString[3] = GroupBFArray[12];
intRainTotal = atoi(strTempString);

// Get the rainfall total of last reset hours and minutes
strTotalRainTime = "";
strTotalRainTime[0] = GroupBFArray[17];
strTotalRainTime[1] = GroupBFArray[18];
strTotalRainTime[2] = ':';
strTotalRainTime[3] = GroupBFArray[15];
strTotalRainTime[4] = GroupBFArray[16];

// Get the rainfall total of last reset day
strTotalRainDate = "";
strTotalRainDate[0] = GroupBFArray[19];
strTotalRainDate[1] = GroupBFArray[20];

// Get the rainfall total time of last reset month
strTempString = "";
strTempString[0] = GroupBFArray[22];
intTempInt = axtoi(strTempString,1);
itoa(intTempInt, strRmonth, 10);

// End of DecodeBFGroup routine
}

// ****************************************************************
void DecodeCFGroup()

{
int intTempInt = 0;
float sngTempFloat = 0;
char * strTempString = "";

// Get the current wind speed (gust)
strTempString = "";
strTempString[0] = GroupCFArray[6];
intTempInt = axtoi(strTempString, 1);
intTempInt = intTempInt * 10;
sngWindGustSpeed = intTempInt;
strTempString = "";
strTempString[0] = GroupCFArray[3];
sngWindGustSpeed += axtoi(strTempString, 1);
strTempString = "";
strTempString[0] = GroupCFArray[4];
intTempInt = axtoi(strTempString, 1);
```

```
sngTempFloat = (float)intTempInt / 10.0;
sngWindGustSpeed += sngTempFloat;

// Get the current wind direction (gust)
strTempString = "";
strTempString[0]= GroupCFArray[7];
strTempString[1]= GroupCFArray[8];
strTempString[2]= GroupCFArray[5];
intWindGustDir = atoi(strTempString);

// Get the current wind speed (average)
strTempString = "";
strTempString[0] = GroupCFArray[12];
intTempInt = axtoi(strTempString, 1);
intTempInt = intTempInt * 10;
sngWindAvSpeed = intTempInt;
strTempString = "";
strTempString[0] = GroupCFArray[9];
sngWindAvSpeed += axtoi(strTempString, 1);
strTempString = "";
strTempString[0] = GroupCFArray[10];
intTempInt = axtoi(strTempString, 1);
sngTempFloat = (float)intTempInt / 10.0;
sngWindAvSpeed += sngTempFloat;
// **********************Need to check how many decimal places

// Get the current wind direction (average)
strTempString = "";
strTempString[0] = GroupCFArray[13];
strTempString[1] = GroupCFArray[14];
strTempString[2] = GroupCFArray[11];
intWindAvDirection = atoi(strTempString);

// Get the wind highest recorded gust
strTempString = "";
strTempString[0] = GroupCFArray[18];
intTempInt = axtoi(strTempString, 1);
intTempInt = intTempInt * 10;
sngWindHiGust = intTempInt;
strTempString = "";
strTempString[0] = GroupCFArray[15];
sngWindHiGust += axtoi(strTempString, 1);
strTempString = "";
strTempString[0] = GroupCFArray[16];
intTempInt = axtoi(strTempString, 1);
sngTempFloat = (float)intTempInt / 10.0;
sngWindHiGust += sngTempFloat;

// Get the wind highest recorded gust direction
strTempString = "";
strTempString[0] = GroupCFArray[19];
strTempString[1] = GroupCFArray[20];
strTempString[2] = GroupCFArray[17];
intWindHiDirection = atoi(strTempString);

// Get the wind highest recorded gust hours and minutes
strWindHiTime = "";
strWindHiTime[0] = GroupCFArray[23];
strWindHiTime[1] = GroupCFArray[24];
strWindHiTime[2] = ':';
strWindHiTime[3] = GroupCFArray[21];
strWindHiTime[4] = GroupCFArray[22];

// Get the wind highest recorded gust date
strWindHiDate = "";
strWindHiDate[0] = GroupCFArray[25];
strWindHiDate[1] = GroupCFArray[26];

// Get the wind highest recorded gust month
strTempString = "";
strTempString[0] = GroupCFArray[28];
intHumMonth = axtoi(strTempString,1);

// Get the wind chill sign
strTempString = "";
strTempString[0] = GroupCFArray[43];
```

```
intTsign = axtoi(strTempString, 1);
intTsign = intTempInt & 0x2;
if(intTsign == 8)
		strWCSign = "-";
		else
			strWCSign = "+";

// Get the wind chill value
strTempString = "";
strTempString[0] = GroupCFArray[33];
strTempString[1] = GroupCFArray[34];
sngWindChill = atoi(strTempString);

/* The low battery indicator
char temphex = GroupCFArray[47] & 0x8;
if (axtoi(&temphex,1) == 8)
		;// Batterys are flat
		else
		;		// Batterys are OK
*/
// End of DecodeCFGroup routine
}

// ****************************************************************
void DecodeWindOnlyGroup()

{
// This is the wind only group which is mainly CF
int intTempInt = 0;
float sngTempFloat = 0;
char * strTempString = "";

// Get the current wind speed (gust)
strTempString = "";
strTempString[0] = WindGroupArray[6];
intTempInt = axtoi(strTempString, 1);
intTempInt = intTempInt * 10;
sngWindGustSpeed = intTempInt;
strTempString = "";
strTempString[0] = WindGroupArray[3];
sngWindGustSpeed += axtoi(strTempString, 1);
strTempString = "";
strTempString[0] = WindGroupArray[4];
intTempInt = axtoi(strTempString, 1);
sngTempFloat = (float)intTempInt / 10.0;
sngWindGustSpeed += sngTempFloat;

// Get the current wind direction (gust)
strTempString = "";
strTempString[0] = WindGroupArray[7];
strTempString[1] = WindGroupArray[8];
strTempString[2] = WindGroupArray[5];
intWindGustDir = atoi(strTempString);

// Get the current wind speed (average)
strTempString = "";
strTempString[0] = WindGroupArray[12];
intTempInt = axtoi(strTempString, 1);
intTempInt = intTempInt * 10;
sngWindAvSpeed = intTempInt;
strTempString = "";
strTempString[0] = WindGroupArray[9];
sngWindAvSpeed += axtoi(strTempString, 1);
strTempString = "";
strTempString[0] = WindGroupArray[10];
intTempInt = axtoi(strTempString, 1);
sngTempFloat = (float)intTempInt / 10.0;
sngWindAvSpeed += sngTempFloat;
// **********************Need to check how many decimal places
// Need to check how many decimal places

// Get the current wind direction (average)
strTempString = "";
strTempString[0] = WindGroupArray[13];
strTempString[1] = WindGroupArray[14];
strTempString[2] = WindGroupArray[11];
```

```
intWindAvDirection = atoi(strTempString);

// Get the wind highest recorded gust
strTempString = "";
strTempString[0] = WindGroupArray[18];
intTempInt = axtoi(strTempString, 1);
intTempInt = intTempInt * 10;
sngWindHiGust = intTempInt;
strTempString = "";
strTempString[0] = WindGroupArray[15];
sngWindHiGust += axtoi(strTempString, 1);
strTempString = "";
strTempString[0] = WindGroupArray[16];
intTempInt = axtoi(strTempString, 1);
sngTempFloat = (float)intTempInt / 10.0;
sngWindHiGust += sngTempFloat;

// Get the wind highest recorded gust direction
strTempString = "";
strTempString[0] = WindGroupArray[19];
strTempString[1] = WindGroupArray[20];
strTempString[2] = WindGroupArray[17];
intWindHiDirection = atoi(strTempString);

// Get the wind highest recorded gust hours and minutes
strWindHiTime = "";
strWindHiTime[0] = WindGroupArray[23];
strWindHiTime[1] = WindGroupArray[24];
strWindHiTime[2] = ':';
strWindHiTime[3] = WindGroupArray[21];
strWindHiTime[4] = WindGroupArray[22];

// Get the wind highest recorded gust date
strWindHiDate = "";
strWindHiDate[0] = WindGroupArray[25];
strWindHiDate[1] = WindGroupArray[26];

// Get the wind highest recorded gust month
strTempString = "";
strTempString[0] = WindGroupArray[28];
intHumMonth = axtoi(strTempString,1);

// Get the wind chill sign
strTempString = "";
strTempString[0] = WindGroupArray[43];
intTsign = axtoi(strTempString, 1);
intTsign = intTempInt & 0x2;
if(intTsign == 8)
        strWCSign = "-";
        else
                strWCSign = "+";

// Get the wind chill value
strTempString = "";
strTempString[0] = WindGroupArray[33];
strTempString[1] = WindGroupArray[34];
sngWindChill = atoi(strTempString);

// End of DecodeWindOnlyGroup routine
}

// ***********************************************************************
// ***********************************************************************


// ***********************************************************************
// ***********************************************************************
/* Description:
This function provides a method to read hexadecimal numbers.
The atoi() function ignores the A-F digits in a hexadecimal
number.  In fact, the first non-digit character in the string
passed to atoi() ends the conversion.
The following example converts a four-character text string that
represents a hexadecimal number into an integer.  It can be used
as a template to create other ASCII-to-number conversions.
The example code defines a C\C++ function called axtoi() that
```

```
does the conversion.  It includes a short main() that to test the
function.   numofchars was originally set at a constant 4 */

int axtoi(char *hexStg, int numofchars)
 {
  int n = 0;         // position in string
  int m = 0;         // position in digit[] to shift
  int count;         // loop index
  int intValue = 0;  // integer value of hex string
  int digit[5] = {0,0,0,0,0};      // hold values to convert

  while (n < numofchars) {
                if (hexStg[n]=='\0')
                        break;
                if (hexStg[n] > 0x29 && hexStg[n] < 0x40 ) //if 0 to 9
                        digit[n] = hexStg[n] & 0x0f;            //convert to int
                else if (hexStg[n] >='a' && hexStg[n] <= 'f') //if a to f
                        digit[n] = (hexStg[n] & 0x0f) + 9;      //convert to int
                else if (hexStg[n] >='A' && hexStg[n] <= 'F') //if A to F
                        digit[n] = (hexStg[n] & 0x0f) + 9;      //convert to int
                else break;
               n++;
  }
  count = n;
  m = n - 1;
  n = 0;
  while(n < count) {
                // digit[n] is value of hex digit at position n
                // (m << 2) is the number of positions to shift
                // OR the bits into return value
                intValue = intValue | (digit[n] << (m << 2));
                m--;   // adjust the position to set
                n++;   // next digit to process
  }
  return (intValue);
}

/* ********************************************************************

int aistoi(char *strInt, int numchars) {
  int n = 0;         // position in string
  int intValue = 0;  // integer value of hex string
  int digit[5];      // hold values to convert
  while (n < numchars) {
                if (strInt[n]=='\0')
                        break;
                if (strInt[n] > 0x29 && strInt[n] < 0x40 ) //if 0 to 9
                        digit[n] = strInt[n] & 0x0f;            //convert to int
                        else break;
               n++;
  }
  n = 0;
  while(n < numchars) {
                // digit[n] is value of the digit at position n
                intValue = intValue + (digit[n] * pow10(numchars - n -1));
                n++;   // next digit to process
  }
  return (intValue);
}

*********************************************************************/
int StoreFullData(void)
{
FILE *stream;
/* open a file for update */
stream = fopen("W_FULL.TXT", "w+");

/* write a string into the file */
// 8F Group
fprintf(stream, "%s,%i,%i,%s,%s,%s,%s,%i,%s,",
               strTime, intWDay, intWMonth, strOutHumidity, strOutHumHi, strOutHumTime,
               strOutHumDay, intHumMonth, strInHumidity);
// 9F Group
fprintf(stream, "%s,%.1f,%s,%.1f,%s,%s,%s,%s,%.1f,%s,%s,%s,%s,%.1f,",
               strOutSign, sngOutsideTemp, strOmaxSign, sngOsTempHi, strOsTempHiTime,
               strOsTempHiDate, strTHMonth, strOminSign, sngOsTempLo, strOsTempLoTime,
```

```
            strOsTempLoDate, strTLMonth, strInVsign, sngInsideTemp);
// AF Group
fprintf(stream, "%s,%s,%.1f,%s,%s,",
            strBarometer, strBarTrend, sngDewPoint, strDPSign, strOutlook);
// BF Group
fprintf(stream, "%.1f,%i,%i,%s,%s,",
            sngRainRate, intRainYesterday, intRainTotal, strTotalRainDate, strRmonth);
// CF Group
fprintf(stream, "%.1f,%i,%.1f,%i,%.1f,%i,%s,%i,%s,%.1f",
            sngWindGustSpeed, intWindGustDir, sngWindAvSpeed, intWindAvDirection,
            sngWindHiGust, intWindHiDirection, strWindHiTime, intHumMonth,
            strWCSign, sngWindChill);

fclose(stream);
return 0;
}


// ***********************************************************************
int StoreWindData(void)

{
FILE *stream;
/* open a file for update */
stream = fopen("W_WIND.TXT", "w+");

/* write a string into the file */
// WindOnly Group
fprintf(stream, "%.1f,%i,%.1f,%i,%.1f,%i,%s,%i,%s,%.1f",
            sngWindGustSpeed, intWindGustDir, sngWindAvSpeed, intWindAvDirection,
            sngWindHiGust, intWindHiDirection, strWindHiTime, intHumMonth,
            strWCSign, sngWindChill);

fclose(stream);
return 0;
}
```

Appendix G Valve Control Routine

```
/*****************************************
* Needs the dos.h library called earlier
* for the inportb function. Needs
* PORT_C and PORT_D set in the main
* function
*****************************************/
/*****************************************
* Fuction ValvesOff
*
* Call during initialisation and at he completion
* of a reticulation cycle - ie after all required
* valves have been operated.
*****************************************/
void ValvesOff()
{
  outportb(PORT_C, 0x00); /* turn off master valve */
  outportb(PORT_D, 0x00); /* turn off other valves */
}

/*****************************************
* Function InitialisePort
*
* Call to initialise port directions and clear
* valves.
*****************************************/
void InitialisePort()
{
  unsigned char portDir;

        portDir = inportb(PORT_DIR); /* get value of port direction register */
        portDir |= 0x0d; /* set ports C and D for output */
        outportb(PORT_DIR, portDir); /* write back to direction register */
        ValvesOff; /* turn all valves off */
}

/*****************************************
* Function ValveOn
*
* Turns on required valve 1-16
*****************************************/
void ValveOn(int v)
{
```

```
  unsigned char portD;
  unsigned char portC;
  portC = 0x4;

  switch (v)
    {           case 1: portD = 0x02;
                case 2: portD = 0x03;
                case 3: portD = 0x04;
                case 4: portD = 0x05;
                case 5: portD = 0x08;
                case 6: portD = 0x09;
                case 7: portD = 0x10;
                case 8: portD = 0x11;
                case 9: portD = 0x20;
                case 10:portD = 0x21;
                case 11:portD = 0x40;
                case 12:portD = 0x41;
                case 13:portD = 0x80;
                case 14:portD = 0x81;
                case 15:{
                              portD = 0x00;
                              portC = 0x05;  }
                case 16: {
                              portD = 0x01;
                              portC = 0x05;  }
                }

        outportb(PORT_D, portD); /* turn on valve (1-14) */
        outportb(PORT_C, portC); /* turn on master valve (if not on) */
}
```

List of Figures