BEng(Hons) Electronic Engineering
Final Year Project

‘Web Based Weather Station Using a ‘TINI’ Java Machine’

By

David Pilch

04 June 2002

Module - **UEE034C3**

Supervisor - **Don Weston**

Acknowledgements

I would like to thank Don Weston for introducing me to this truly fascinating subject and for guiding me through the last few months. I would also like to say a special thanks to my family and friends for providing me with support and for putting up with me during this time.

Contents

1 Summary

This report looks at the design and development stages of building a web based weather station using a very small Java Machine called TINI. TINI is Ethernet equipped and boasts extensive I/O capability. This machine is used in this project to provide users with the ability to connect to it and retrieve the latest simulated weather data through a web browser. All of the weather station sensors have been designed and built within the timescale of this project. The whole project was tested fully on a Local Area Network (LAN) and views on getting TINI onto the internet were formed.

2 Introduction

2.1 Module Details

Module Title	Individual Project (Electronics)
Module Code	UEE034C3
Award	BEng (Hons) Electronic Engineering
Credit Rating/Level	30 Credits at Level 3
Module Leader	Ashley Longden
Supervisor	Don Weston

2.2 Project Overview

This project looks at the development of a web-based weather station using a relatively new technology called the Tiny InterNet Interface (TINI). The website for TINI[1] describes it as:

'A platform developed by Dallas Semiconductors to provide system designers and software developers with a simple, flexible and cost effective means to design a wide variety of hardware devices able to connect directly to corporate and home networks. The platform is a combination of a small but powerful chipset and a Java™-programmable runtime environment. The chipset provides processing, control, device-level communication and networking capabilities. The features of the underlying hardware are exposed to the software developer through a set of Java application programming interfaces.

The primary objective of the TINI platform is to give everything from small sensors and actuators to factory automation equipment and legacy hardware a voice on the network. This allows the devices to be monitored, controlled and managed remotely.'

The project investigates the capability of TINI by using an interface board developed by Taylec who are a Bristol based electronics company that specialise in hardware support for the TINI platform. For more detailed information about TINI and the Taylec interface board please see section 5.1.1.

For this project the TINI board is connected to some external electronic hardware which has been designed and developed as part of the project. This hardware consists of various electronic sensors that make up a weather station. The data from the sensors is collected and stored by the TINI machine. Users can log onto TINI via a webpage and retrieve the latest sensor data.

This project was found through the University of the West of England (UWE) in Bristol. The supervisor for this project (Don Weston) was already investigating the use of TINI as a teaching aide for foundation year students at UWE. Due to time restraints, he was looking for a student to take on the project and investigate the capabilities of the TINI machine. The idea of gearing the project

towards a weather station came from a contact at Taylec. They were looking for someone to experiment with TINI as the interface to a remote weather station. This provided an ideal scope for the project which sounded extremely interesting, potentially enjoyable and ideal for a final year honours project.

2.3 Project Aims

Below is a list of the main project aims:

- Develop software to retrieve data from the Analogue to Digital Converter (ADC) and display the values to the screen in a web browser environment.
- Create software to demonstrate the other functionality of TINI and the Taylec board.
- Design and develop any weather station sensors that are outside the budget for this project and develop suitable hardware to interface the sensors with the TINI board.
- Test the whole system over a LAN, making sure all of the information is being displayed on the web page as required.
- Consider what is required to set TINI up on the internet to allow users to access data across the world.

2.4 Learning Aims

Below is a list of the main learning aims for the project:

- Learn about the full potential of TINI and how it can be interfaced with the outside world. Also learn how to develop programs for TINI machines to manipulate the platforms different I/O functions.
- Learn how to develop web pages in HTML that can dynamically link to external hardware to retrieve information.
- Develop ideas for future uses of the TINI platform in the working environment.

3 Project Planning

3.1 Major Deliverables

- Project proposal presentation
- TINI configuration and successful running of test programs
- Successful development and testing of ADC applications on TINI anmd displaying data to a web page.
- Successful development and testing of the external hardware including weather station sensors.

3.2 Project Constraints

The project is mainly limited with cost of equipment (see equipment listing in 3.3.2). Many of the weather station sensors are actually available for applications like TINI. The sensors however can cost up to £500 altogether for an entire weather station. Therefore some of the sensors will have to be improvised. Another limitation is time. The project has to be finished by May 2002. Further development ideas can therefore only be investigated if time permits.

TINI is almost fully documented on the internet. Therefore without an internet connection this project would almost be impossible. All of the software and support for TINI is downloadable from various sites across the internet.

3.3 Measurement of Project Success

The level of project success has been measured by comparing the actual project deliverables and achievements throughout the project to the project objectives set out in the planning stages.

3.4 Resource Requirements

3.4.1 Materials & Equipment required

Below is a list of the equipment required for this project:

- Windows Equipped PC with network capabilities.
- TINI SIMM board
- Taylec interface board for TINI.
- Power supply
- Serial cable
- Network cabling
- Necessary components for the external hardware. Integrated Circuits (ICs), strip board and various connectors.
- Materials for development of weather station sensors

Figure 3.2.2 below shows the rough purchase costs of the equipment required for this project. This is assuming that a suitable PC is available for development puposes.

Description	Cost (£)
TINI SIMM board*	40
Taylec Interface Board*	80
Power Supply	10
Serial Cable	5
Networking Cable	5
Electronic Hardware Components	6
Weather Station Sensor Materials	Minimal
Total	**146**

*already purchased prior to the project start

Figure 3.3.2 – Table of Costs

The TINI SIMM board and the interface board were both purchased from Taylec who are a Bristol based electronics company. Both of these parts had already been ordered before the start of the project.

3.4.2 Required knowledge and learning

For this project there are two main areas for learning. This is making the assumption that the hardware development is not really learning but more of just applying previously learned methods.

Java for TINI

Applications for the TINI board are developed using the Java Application Programming Interface (API). Knowledge of programming in Java is required with special attention being paid towards the development of applications specifically for the TINI board. There is also a required understanding of the TINI applications that help with the development of TINI programs.

Webpage Development

The user interface for this project is a simple webpage. This webpage is responsible for connecting to the TINI board and retrieving data from the weather station sensors. Knowledge of designing web pages and how they communicate with external sources is required to produce this interface.

4 Planning & Research

4.1 TINI Information

4.1.1 What is TINI?

TINI is very small java machine produced by Dallas Semiconductors[2]. It consists of a standard 72 pin SIMM board assembly and is based on a DS80C390 microprocessor running at 36 MHz. This processor supports the following I/O capabilities which are exposed through Java APIs:

- Dual serial ports
- Dual 1-Wire net interfaces
- Dual CAN (Controller Area Network) controllers
- 2-wire synchronous serial bus
- General-purpose digital I/O
- Easy system expansion using a parallel bus interface
- Direct connection to 10/100Base-T Ethernet networks

It also has a real time clock for time stamping, flash ROM for execution of the runtime environment, 1 MByte of non-volatile SDRAM and a wide operating temperature range of -20°C to +70°C. All of this only requires a single polarity 5v power supply and has a maximum current rating of 250mA. Figure 4.1.1 shows the TINI board.

Figure 4.1.1 – The TINI board

TINI has been designed to provide remote sensors and hardware the ability to be part of a modern computer network. The TINI SIMM can be interfaced with a multitude of different I/O boards depending on the required application. See section 4.2 for more details.

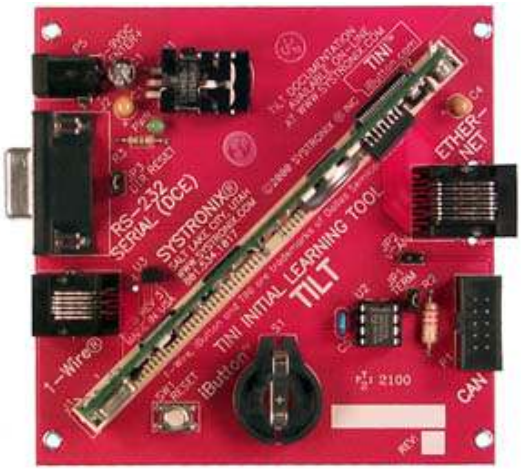
4.1.2 TINI Applications

TINI can be used for stand alone applications such as local monitoring and controlling of devices. However a large majority of TINI developers take advantage of the networking capabilities for remote monitoring and control of systems. Below are a few typical applications for TINI taken from the iButton[3] website:

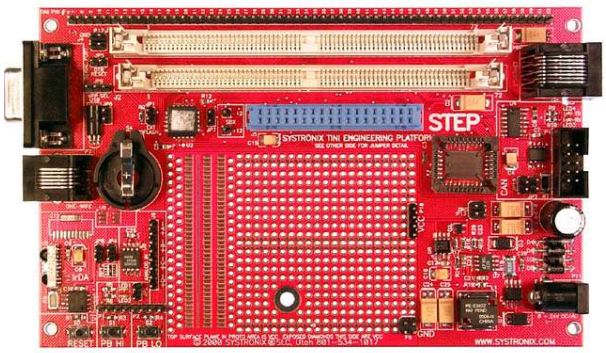
- ***Industrial controls*** *- TINI can easily be used for communicating with factory automation equipment, networked switches and actuators.*
- ***Web based equipment monitoring and control*** *- communicating with equipment to provide remote diagnostics and data collection for purposes such as monitoring device utilisation.*
- ***Protocol Conversion*** *- TINI based systems can be used to connect legacy devices to Ethernet networks. Depending on the I/O capabilities of the legacy system, this may be a job that can be done with a PC or workstation as well. However, TINI can do the job at a small fraction of the cost and size.*

4.2 Interface Boards

Since the release of TINI in June 2000 various companies have started producing interface boards for the TINI SIMM. These boards expose the I/O capabilities of the TINI SIMM to the user. Below are a few examples of different types of interface boards.



This is the TINI Initial Learning Tool (TILT) from Systronix[3]. It was designed as a low cost, basic sockets board for TINI. It Consists of a serial communications port, Ethernet port, CAN interface, 1 wire interface and a iButton socket. **Price - £35** (without TINI).



This is the Systronix TINI Engineering Platform (STEP) from Systronix[2]. This has extra functionality of a Infra Red and a 100x160mm prototyping board. **Price - £45** (without TINI).

This is the Mini-T board from Taylec. Has the most basic of functionality. It consists of a serial communications port, Ethernet port and one wire interface.
Price - £30 (without TINI).

This is the Tutor board from Taylec. It has extensive I/O capability with onboard ADC, DAC, digital I/O, CAN, 1-wire, serial communications, Ethernet and even an onboard LCD screen.
Price - £85 (without TINI).

The TINI Tutor board from Taylec has been used for the development of this project. This board provides extensive functionality for developing a whole range of applications. As the name suggests it is a great learning aide for new developers. One of the aims of this project is to provide an insight to the development of embedded Java applications. Information gained can then be used to produce a demonstration tool in embedded systems for foundation year students at UWE.

4.3 Tutor Board

The Tutor board is designed to allow TINI developers to get their own embedded Java programs up and running as quickly as possible. The board has the following functionality:

- Dual serial communications ports (RS232)
- Ethernet 10/100Base-T port
- 1-wire interface
- 2 line LCD screen
- 8-bit digital input via onboard dipswitches or header cable
- 8-bit digital output via onboard LEDs or header cable

- 8-bit Analogue to Digital Converter (ADC)
- 8-bit Digital to Analogue Converter (DAC)
- Interrupt and reset switches
- Onboard Light Dependant Resistor (LDR) for analogue input
- Control Area Network (CAN) interface.
- I2C bus (A two-wire serial bus used for control, diagnostic and power management).

The layout of the Taylec Tutor board can be seen on page 15 of the appendices.

4.4 Weather Station Components

Weather stations are becoming more complex as technology advances. We are now able to monitor certain aspects of weather systems that would not even have been thought of 20 years ago. Below is an extensive list of the different weather station sensors that are used today.

Basic Sensors

- Temperature
- Wind Speed (anemometer)
- Wind Direction
- Rain Fall
- Sunlight Index
- Air Pressure (barometer)
- Humidity

Other Sensors

- Pollen Count
- Air Quality (pollution)
- Ultra Violet intensity (sunburn factor)
- Speed and direction of cloud motion (nephoscope)
- Visibility
- Radar maps
- Infra Red Satellite pictures (measure heat of objects)
- Visible Satellite pictures
- Lightening detection

Due to budget constraints, this project will be limited to investigating the sensors that come under the 'Basic Sensor' section. A description of each sensor and how it can be implemented for TINI is discussed below.

Sensor	Description	Types	Cost (£)	Implement?
Temperature	Measures the air temperature normally within a vented box. Can be calibrated for centigrade, farenheit or kelvin.	Vary from a simple thermistor which varies in resistance depending on the temp to a fully electronic sensor	70p - £15	Yes
Wind Speed	Measures the wind speed by a rotating fan that usually consists of 3-4 plastic cups. The fan is propelled by the wind which in turn rotates a dc generator to output a voltage.	Wind speed and direction normally come in the same package called an anemometer. These can also be made using a dc motor or a stepper motor.	Basic: £60 Advanced: £400	Yes (make)
Wind Direction	Measures the direction of the wind by using a vain connected to a rotary electronic position sensor. This outputs a voltage depending on the position. This is normally combined with the Speed to give you the anemometer.	Same as above. They can be made using a simle position sensor or even a modified potentiometer.	As above	Yes (make)
Rain Gauge	This measures the amount of rain fall by outputting a digital count everytime a known mass of water passes through a funnel.	Generally quite complex for the electronic version but can be bought in a package. Ther are also methods for building your own.	Basic: £60 Advanced: £200	Yes (make)
Sunlight Index	Measures the intensity of the light. Normally retrieved from an electronic light sensor.	There are off the self version avalible but in general a LDR can be used to measure light levels.	50p - £5	Yes
Humidity	Measures the percentage of water within the air.	Humidity sensors can be bought or very crude versions can be made. Very difficult to make your won electronic version.	Basic: £25	No
Air Pressure	Measures the atmospheric pressure normally in millibar readings.	Special electronic pressure sensors are needed to measure pressure.	Basic: £20	No

The reason for not implementing the humidity and air pressure sensors is due to budget restraints of the project. This is also the reason why the wind speed, wind direction and rain gauge were made instead of bought.

4.5 Hardware

The standard hardware configuration of the Taylec board only provides one Analogue to Digital Converter (ADC) input. This means in theory only one analogue signal can be read in at a time. The weather station has at least 5 sensors that need to be read simultaneously. Some simple circuitry to switch between the sensors was required.

Each sensor also required some signal conditioning before being fed into the ADC. Signals into the ADC needed to range between 0 and 5v DC. Therefore the output from each signal needed to be scaled to the same range.

4.6 Web Page Development

The user interface for this project is a website that is accessible through a standard windows web browser. This allows the user to view all of the data from the weather station on the screen. Websites and the web pages within them are coded in Hypertext Mark-up Language (HTML). HTML lets you format text, add graphics, sound and video, and save it all to a text file that any computer can read. There are various applications such as Microsoft FrontPage or Macromedia Dreamweaver that simplify the making of web pages by actually writing the code for you. However these packages have limited capabilities and are only really useful for creating the page template quickly. With knowledge of HTML you are never limited to a particular program's features.

JavaScript is a programming language that can be embedded within web pages to add interactivity to a page. It lets you create an 'active' user interface that gives the user feedback as they navigate the web pages. For example you can create buttons that become highlighted as you move the mouse pointer over them. JavaScript can also be used to do mathematical calculations or to make sure that users enter correct information when filling out online forms. JavaScript is not really used in this project but could be used to improve the weather station website at a later date.

Macromedia Flash 5 is a program that allows you to create graphical effects for web pages. This can range from simple buttons that change state when selected, to full animated movies. This is a far more powerful tool than JavaScript for creating graphical effects for web pages and is being used more and more throughout the internet. Many companies are now using Flash movies as the introduction to their website, creating a unique user experience. This technology was integrated into the weather station website towards the end of the project to create the animated buttons in the menu bar.

4.7 TINI Software Development

As mentioned previously, applications for TINI are developed using the Java API. Java programs have to be converted into a format suitable for TINI before being executed on the board. For information on why this is required see section 5.3.1.

A series of online lectures were used as an introduction to programming embedded systems in Java. These were found at the following website link:

http://cwl.ssu.portsmouth.oh.us/classes/j1wire_class/

This series of lectures provided a good understanding of how Java applications are created and compiled. It also gave a brief introduction to writing programs for TINI applications.

One of the main issues with TINI is that it is almost entirely documented on the internet through websites and downloadable manuals. However the support that is there is extremely useful. All of the software that is required for TINI is downloadable from various sites. The main support site for TINI is www.iButton.com which has links to detailed instructions of how to setup TINI and where to get the required software. This was the main site used throughout this project for support.

5 Implementation

5.1 The Setup

Below is a list of the equipment required to setup TINI:

- TINI SIMM board
- TINI Tutor Board
- 9v Power Supply
- Ethernet Cabling
- Windows Equipped PC with Network Capabilities

For development purposes for this project TINI was setup in the following configuration:

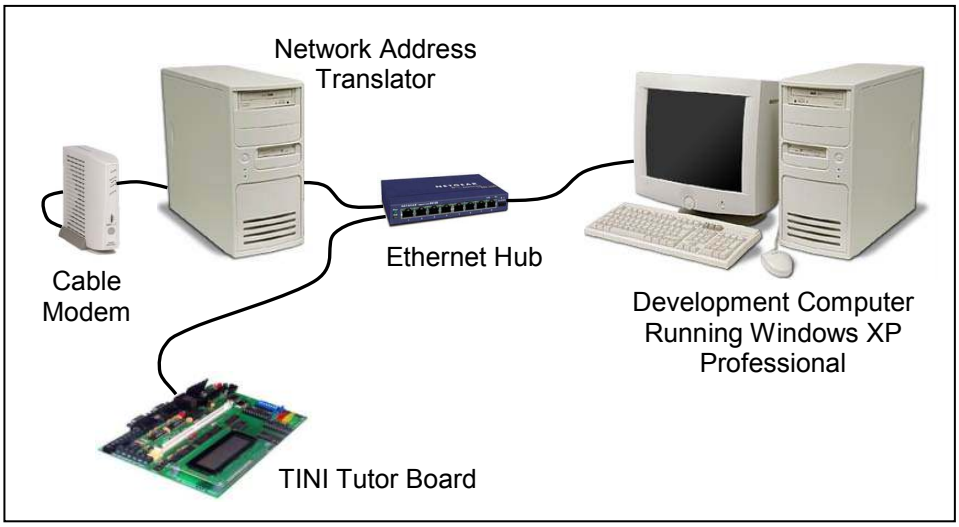


Figure 5.1 – Development Configuration

In this configuration TINI is available to anyone on the local network. TINI also has access to the internet via the network address translator/ gateway.

5.2 Configuring TINI

As previously mentioned in section 3.2.2 TINI is mainly documented on the internet. This includes all of the relevant software and support for initially configuring TINI. Setup instructions from the iButton[4] website were used. The following software was downloaded:

- Java SDK version 1.3.1 from Sun[5].
- Java Communication API[6]
- TINI SDK version 1.02d[7]

The Java SDK provides the relevant java compilers and class information. The java communications API allows Java to communicate with the communication ports on the computer. TINI SDK is all of the operating system and software

support needed for TINI. All of these were installed into their default directories according to the instructions.

A program called JavaKit (that is part of the Java SDK) is then used to connect to the TINI board through a serial communications port. This allows the TINI operating system to be transferred to the TINI board via the communications cable.

5.2.1 JavaKit Problems

One thing that the configuration instructions assumed was that the user had certain knowledge of the DOS commands 'set PATH' and 'set CLASSPATH'. Unfortunately this was not the case. The importance of these commands was about to become crystal clear.

The problem with Java compared to other modern programming compilers is that it is still based around DOS commands. Modern operating systems such as Windows XP are no longer based on DOS. They only have DOS simulators which can sometimes be a bit temperamental.

The DOS command 'set PATH' allows users to specify a path for an application. This allows the user to run applications from any directory in DOS without having to type the full path of the command. DOS will search all specified paths for an application, including all of the sub directories below these paths. If the following path is set for Java then all of the Java applications can be run from any directory in DOS:

```
set PATH C:\jdk1.3.1_02\bin %PATH%
```

The `%PATH%` part of this command simply includes any paths that have already been set. However, this method of setting the PATH variable only works for Windows 95 and 98. Windows XP requires that you add path variables through the system control panel. By clicking on the advanced tab then the 'Environment Variables' button you can edit the system variables. After the path variables have been added the machine needs rebooting before the changes take effect.

It was not mentioned anywhere in the TINI documentation that you had to set the PATH variables for Java, before using the given command line to run JavaKit. It also did not describe anywhere of how to set the path variables in Windows XP. This was found through the standard windows help files.

The 'set CLASSPATH' command is very similar to the 'set PATH' command but allows you to specify the default directory for classes that are used for compiling code. This was set in the same manner for Windows XP as described above for the PATH variables.

After having problems with the installation and running of JavaKit it made sense to create a program that installed all of the relevant files into the correct directories for the user. The main thing that was a hassle to get right was

setting up the communications API. This involved manually transferring files to specified directories. A program could easily be made to install Java SDK, the communications API and TINI SDK all at once. As long as all of the source files are in the same directory a batch file can be used to automatically copy files to the correct directories. A batch file is basically a list of dos commands within a text file that are run when the batch file is executed. The details of the batch file are shown below:

Run the Java SDK 1.3 setup program.
```
jdk\jdk1_3.exe
```

Copy the communication API files to the Java SDK directories.
```
copy commapi\win32com.dll c:\jdk1.3.1_02\jre\bin
copy commapi\comm.jar c:\jdk1.3.1_02\jre\lib\ext
copy commapi\javax.comm.properties c:\jdk1.3.1_02\jre\lib
```

Make a new directory for the TINI SDK
```
mkdir c:\tini1.02.
```

Copy all the TINI SDK files to the new directory
```
xcopy tinisdk c:\tini1.02 /e
```

Once this has been completed all you have to do is execute the other batch file provided called JavaKit.bat to run JavaKit. The JavaKit.bat file has the same command line as described in section 5.2.2. All of the setup files, batch files and a text file explaining the setup process can be zipped up into one file and posted on the TINI website to help developers setup their TINI machines.

5.2.2 Running JavaKit

The easiest way of running JavaKit (which was not mentioned in any of the TINI documentation) is by using the following command line in DOS:

```
C:\jdk1.3.1_02\bin\java –classpath
C:\tini1.02\bin\tini.jar; %CLASSPATH% JavaKit
```

This will still run even if *none* of the PATH and CLASSPATH variables have been set. If all of the environment variables have been set then all you need to type is the following:

```
java JavaKit
```

So setting the PATH and CLASSPATH variables can save a lot of typing for commands that are used regularly.

Once JavaKit is running, downloading the operating system (slush) to the TINI board is easy. JavaKit uses a serial cable to connect between the com port on the development computer and the serial 0 port on the Taylec board. The serial 0 port must be used at this stage, not the serial 1 port.

5.2.3 Slush

The iButton website describes slush as:

'A small system shell intended to provide a Unix-like interface with serial (TTY), Telnet, and FTP servers.

Slush is less than a full operating system, but more than a simple shell. It provides a way to view and manipulate the file system and to control system functions such as the watchdog timer and network configuration.

The system is designed to be a multi-threaded, multi-user system allowing simultaneous login sessions. Slush uses a username/password combination to authenticate a login request. Users can be added and removed from the system by a user with super user privileges.'

Once slush is installed the Ethernet needs to be configured so that TINI can be accessed over the network. This is just a case of using the 'ipconfig' command provided with slush. With this you can provide the TINI board with a fixed IP (Internet Protocol) address, subnet mask and gateway. TINI was configured as follows:

```
Hostname              : mytini.
Current IP            : 192.168.1.10
Default Gateway       : 192.168.1.250
Subnet Mask           : 255.255.255.0
Ethernet Address      : 00:60:35:00:9d:1d
Primary DNS           : 194.112.32.1
Secondary DNS         : 194.112.32.10
DNS Timeout           : 0 (ms)
DHCP Server           :
DHCP Enabled          : false
Mailhost              :
Restore From Flash    : Not Committed
```

Once the Ethernet has been configured you can log onto TINI using the standard windows Telnet program. Or transfer files to TINI with the standard File Transfer Protocol (FTP) program. With TINI now configured for network access you no longer need to use the serial cable or JavaKit for communication.

5.3 TINI Software Development

5.3.1 TINI Software Format & Tools

As mentioned previously, the functionality of TINI is exposed through the Java API. This means that even simple Java programs will run on TINI machines. Unfortunately it is not quite as simple as just transferring a class file to the TINI board and running it. Java programs have to be converted into a format suitable for TINI first. You need to do this because TINI is an 8-bit microcontroller and does not have the power to resolve everything during

execution with any reasonable speed. So converting to a special TINI file (*.tini) helps TINI's Java Virtual Machine (JVM) by resolving references at build time instead of at run time.

TINIConvertor is a program provided with the TINI SDK that converts Java (*.class) programs to the TINI (*.tini) format. Once in this format the files can be transferred to the TINI board by using FTP and executed.

5.3.2 First Program

A test program was provided with the Tutor board, which demonstrated all of the functionality of the board. This was already in the correct TINI format and was transferred to the TINI board using the FTP program. The program could then be executed by connecting to TINI using a Telnet session. Both FTP and Telnet sessions require login names and passwords. For development purposes during this project the following default login name and password was used:

Login Name: `root`
Password: `tini`

Once logged on with the Telnet session it is very simple to actually run applications. All you need to do is go to the directory where the *.tini was transferred to and type the following:

```
java test.tini
```

In this case ‘test.tini’ is the test program for the Tutor board. This program stepped through each function of the board, for example printing the date to the LCD display and turning the digital output LEDs on and off. All functions worked correctly.

Now it was time to write and test the first program. Staying with programming tradition the first application simply printed ‘Hello World’ to screen upon execution of the program. The reason for doing such a simple program was to check the method of compiling Java code and converting it into the TINI format. This method is described below:

1. Write the required program code in a text editor such as Notepad. Save the text file with the *exactly* the same name as the class that has been made (case sensitive) and give it a .java extension.
2. Open a DOS window and go to the directory where the *.java file was saved. Type the following to compile the code:

   ```
   javac myprog.java
   ```

3. If there are no errors a myprog.class file should now be in the same directory as the myprog.java file. This file now needs to be converted to the TINI format. Type the following:

```
java TINIConvertor -f myprog.class -o myprog.tini -d
c:\tini1.02\bin\tini.db
```

The `-f` specifies the file/zipfile/directory of the classes to be converted, `-o` specifies the output file name for the converted file and `-d` specifies the TINI database file required to do the conversion.

4. If the conversion was a success a myprog.tini file should be in the same directory as the myprog.class file. This file can be transferred to the TINI board using an FTP session and executed through a Telnet session.

5.3.3 Software Requirements

The main idea for this project is to use TINI as a remote monitoring device. The data that TINI monitors is provided by five different weather station sensors. Four of the five sensors will provide a varying analogue voltage within the range of 0-5v. The remaining sensor is the electronic rain gauge. This provides a TTL logic value of either 0 or 1 (5v) as the input.

The user needs to be able to connect to the TINI machine in real time via a web browser and retrieve data from all of the sensors simultaneously. This data needs to be displayed to the web page that the user is viewing.

The main problem with this is that the Tutor board only has one ADC, which means you can only read in one analogue signal at a time. This project needs to read in 4 simultaneously. One way of overcoming this problem is to use a 4 to 1 multiplexer that allows you to select which channel to read. All you have to do then is very quickly switch between the 4 channels, taking a reading for each channel. The user therefore thinks that the data is being read in at the same time when really it is actually being read in one after each other just very quickly. The digital output from the Tutor board can be used to control the channel select pins on the multiplexer.

5.3.4 Reading from the ADC

This program simply selects a channel on the multiplexer (applies a digital output) and reads the value from the ADC. It does this 4 times with a different channel select each time. This data is then displayed in real time to the TINI command window (Telnet session dos window). See Figure 5.3.4 for the layout of the multiplexer circuit.

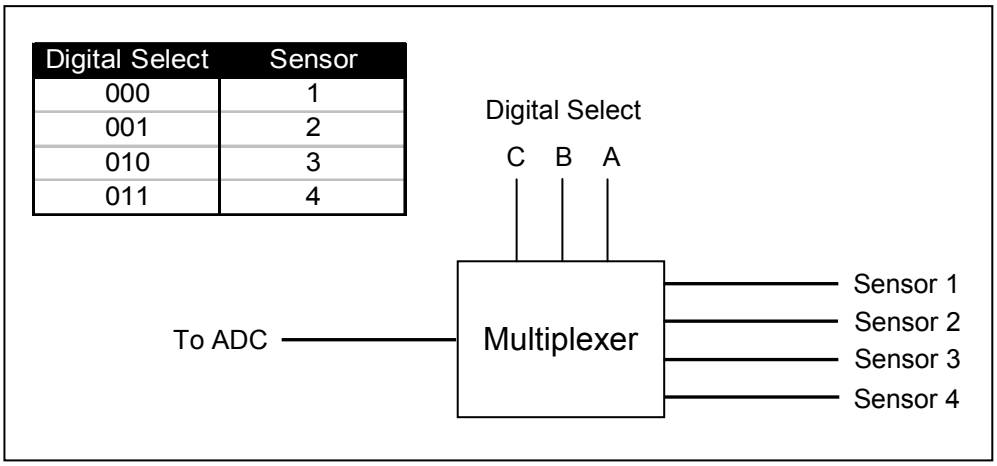


Figure 5.3.4 – Multiplexer Layout

The Java classes ADCManip.class and DigitalOutputManip.class are supplied with the Tutor board. The test.tini program supplied with the Tutor board uses these classes to manipulate the ADC and digital outputs. The methods were looked at and used to create the following program. Firstly to make the using of the DigitalOutputManip.class a little easier a MuxSelect.class was made. This simply allows you to refer to the channels as CH1, CH2 rather than the actual hexadecimal value. It is actually an extension of the DigitalOutputManip.class file. Figure 5.3.4.1 below shows the flow diagram for this class.

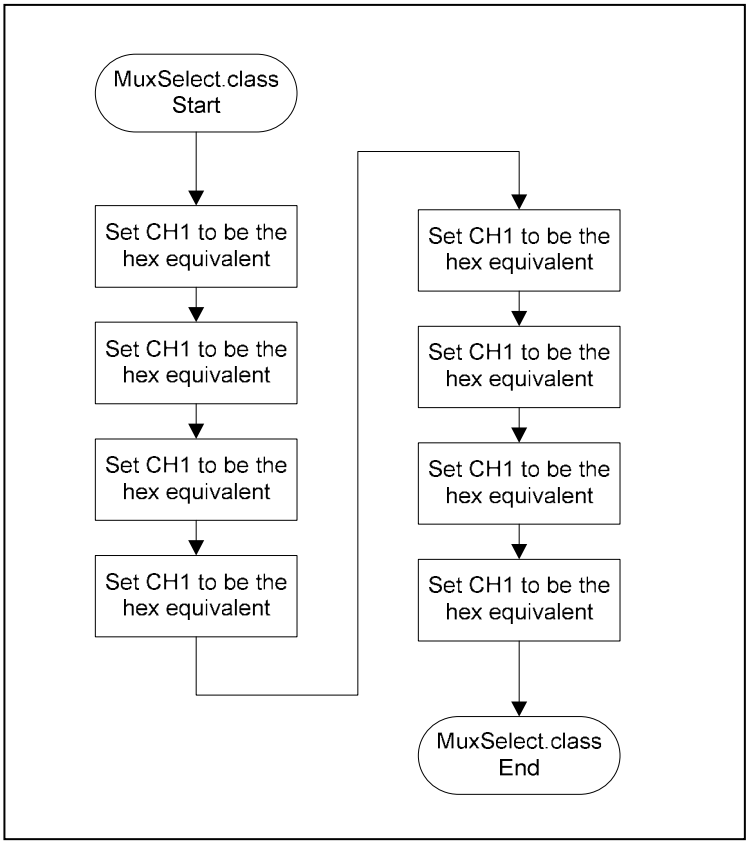


Figure 5.3.4.1 – Flowchart for MuxSelect.class

The MuxSelect class is used in all of the programs that manipulate the digital output of the Tutor board. Figure 5.3.4.2 show the flowchart for the basic ADC read program.

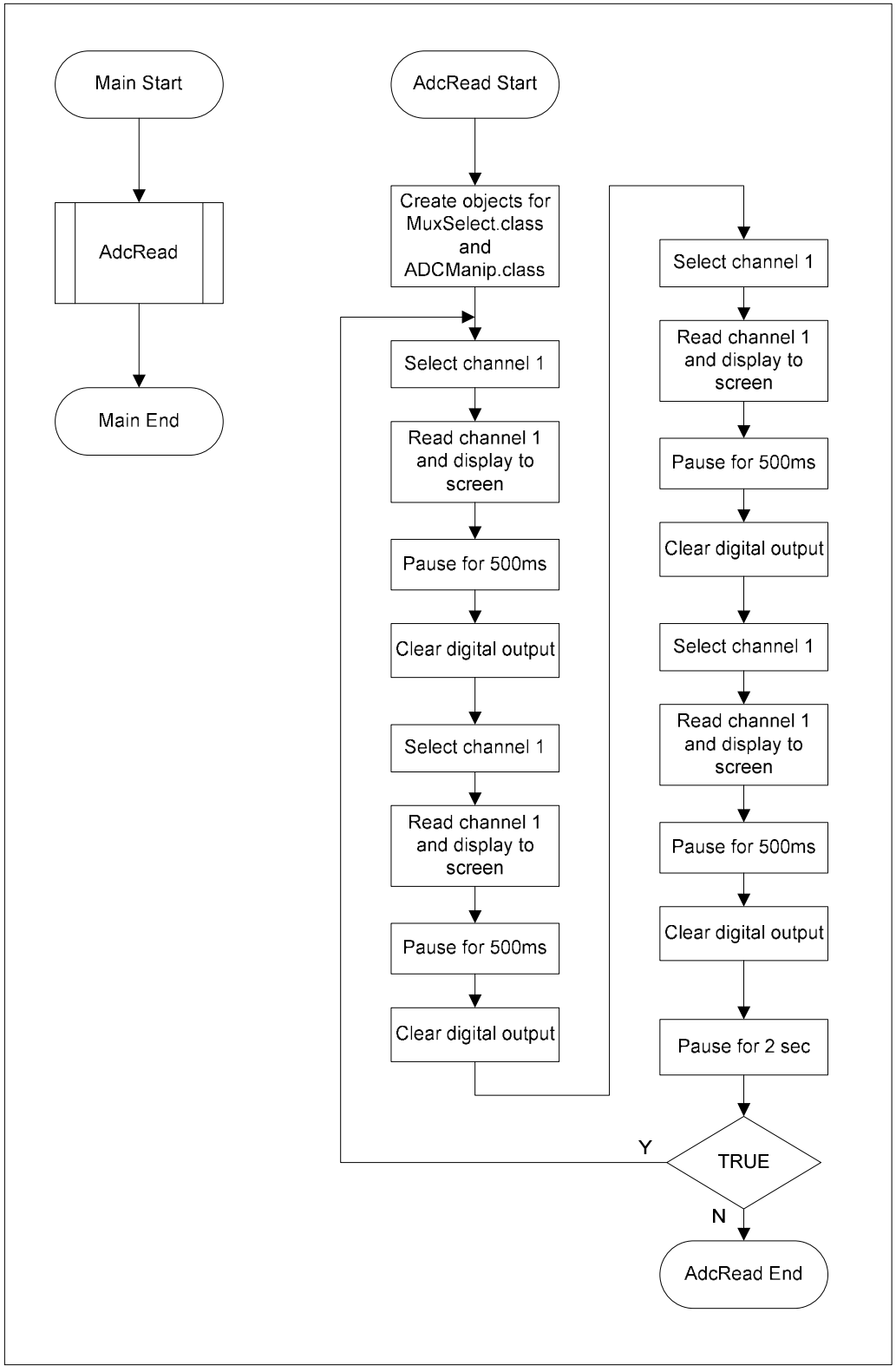


Figure 5.3.4.1 – Flowchart for Adc.class progam

This is a very simple program that simply selects a channel (sets the digital output to a certain value) then reads the ADC value from that channel and displays it to screen. It reads all 4 channels then pauses for 2 seconds before repeating. The Java code for this program can be seen on page 1 of the appendices. This program was compiled and converted to TINI format and transferred to the TINI board.

The Tutor board has three options as the input for the ADC. Either and external input, an onboard potentiometer or an onboard Light Dependant Resistor (LDR). This can be selected by changing a jumper setting. For the purpose of the software testing the LDR option was selected. This is the easiest thing to change in terms of value. By simply turning a light on and off you can change the ADC input value dramatically.

The simple ADC program was run on the TINI board and worked as expected. The onboard LEDs showed the digital output changing and the ADC values being displayed to screen changed according to the light level being applied to the LDR.

5.3.5 Webpage Integration Ideas

Now that the simple ADC program was working, manipulating the ADC was no longer a problem. The main hurdle to overcome now was how to retrieve this data remotely without having to log on to TINI with a Telnet session.

The first idea was to still use a Telnet session but to carry out an automatic login. Telnet commands can be issued from a webpage in HTML code. The command line shown below can be used from the address line in a web browser to log onto some Telnet servers automatically.

```
telnet://loginname:password@telnetaddress
```

so for this TINI it would be:

```
telnet://root:tini@192.168.1.10
```

Unfortunately TINI does not support this method. The only way to log onto TINI through Telnet (without manually reconfiguring the operating system) is by entering the login name and password. Even if the auto login did work the data would still need some major manipulation to make it compatible with a webpage.

However this auto login does work for FTP through a web browser. This makes transferring files to TINI a lot simpler and quicker. The screen shot in Figure 5.3.5 shows the TINI file system from internet explorer.

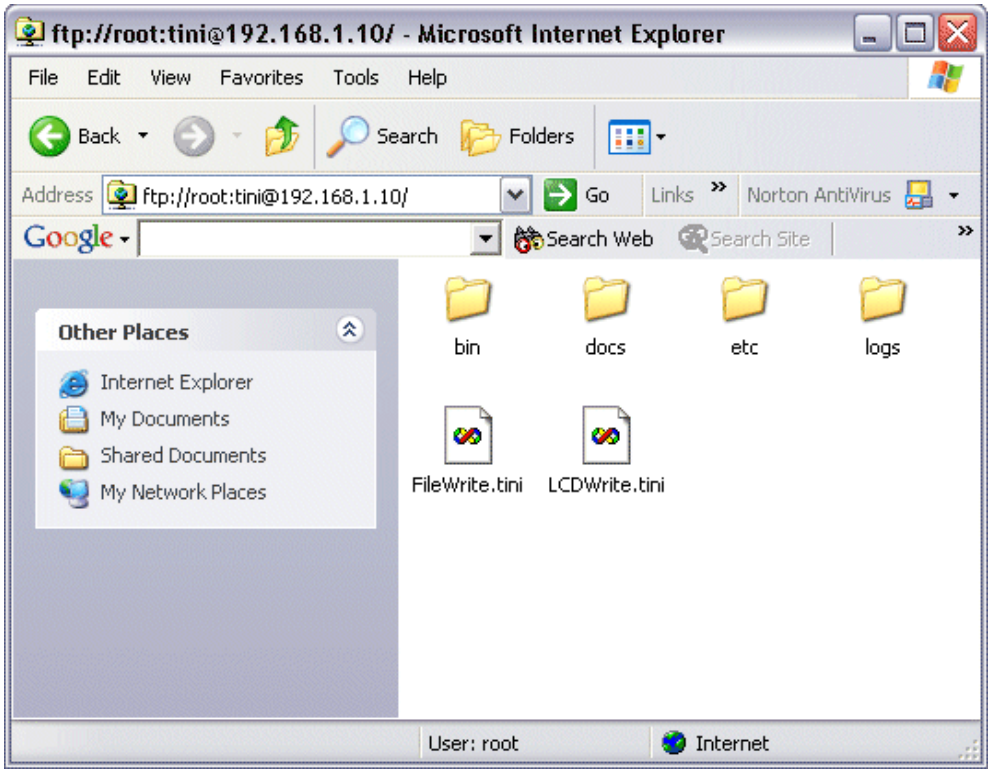


Figure 5.3.5 – Accessing TINI through a web browser

At this stage there was still no successful method of remotely accessing data from the TINI ADC through a web browser. An e-mail was sent to the TINI interest group[8] asking about the problem of remote access through a web browser. The reply to this e-mail can be seen on page 16 of the appendices. The feedback was extremely helpful and suggested using a program developed by Smart Software Consulting[9] (SmartSC) called TiniHttpServer. This program converts TINI into a very small web server allowing it to serve its own web pages. This would allow TINI to embed the data read from the ADC in a web page stored locally. This webpage could then be called up as a frame within a main webpage on another server.

5.3.6 TINIHttpServer

TiniHttpServer is described by SmartSC as:

'A multi-threaded HTTP server for TINI that supports Java Servlets. TiniHttpServer turns your TINI into a web server with server-side programming capabilities. As distributed, TiniHttpServer is ready to serve up your Java applets, HTML documents, and other files directly from your TINI, plus it is bundled with nine demonstration servlets.'

A servlet is a Java class used to extend the capabilities of a server that hosts request-response applications. Servlets can be invoked and executed on a server, usually on behalf of a client, while Java applets work on the client side. Servlets are most commonly used to extend the applications hosted by Web

servers. So for the purposes of this TINI project servlets can be used to manipulate the ADC on the Tutor board and display the data to a web page.

To allow the TiniHttpServer to be loaded to the TINI board the following software is required:

- The TiniHttpServer software from SmartSC[7].
- The class files from the Java Servlet Specification 2.2[10]
- Ant version 1.4[11] - an open source cross platform build tool written in Java. TiniHttpServer build environment is based on Ant.
- TiniAnt 1.1.1[12] - a free plug-in for Ant that integrates the TINI build process into Ant.

This was all downloaded from the relevant sources and installed as instructed. After configuring some of the files to be specific to the TINI machine being used, TiniHttpServer could be loaded to TINI by simply going to the TiniHttpServer directory in DOS and typing `ant`. This automatically installed all of the required files onto the TINI machine. After resetting TINI the HTTP server was ready to use.

By entering the IP address (192.168.1.10) into the address bar of the web browser, the default webpage for TiniHttpServer was displayed. This provided links to all of the demonstration servlets provided with the server software. They all worked fine. Now it was just a case of writing a servlet specifically for this project and getting it to work. Within the TiniHttpServer documentation there is a good guide to compiling the server software with your own servlets and web pages.

5.3.7 ADC Servlet

To get an idea of how servlets are written, the example servlets provided with the TiniHttpServer software were looked at. When connected to TINI these servlets simply printed to file HTTP code which had data from TINI embedded within it. So the user would see a webpage with real-time information. So now the previous ADC program needed to be converted into a servlet format.

The servlet could retrieve the ADC data directly every time a user accesses the web page. This would display the data at that exact point in time but could cause a few problems as soon as multiple users try to access the same page all at the same time. The successful retrieval of data is relying on the multiplexer scrolling through all 4 channels in the correct order. However if more than one user attempts to retrieve data at times that are only milliseconds apart from one another, the multiplexer could become unsynchronised. Therefore the users could receive completely wrong data.

This problem can easily be overcome by running a separate program to read the data from the ADC periodically and saving the values to a file. Each time the data is read from the ADC it overrides the current data in the file. All the servlet does then is to read the data from the file rather than directly from the

ADC. This means that the multiplexer always stays synchronised because it is no longer controlled by user input. Now multiple users can read the same file at the same time and all access data which is only a matter of seconds old, thus still in relative real time.

There are now two sections to reading the ADC data, the program for actually reading and storing the data to file and the servlet that users use to access the data. The flow chart in Figure 5.3.7 explains how the reading and storing of the ADC data works. See page 2 in the appendices for the actual code.

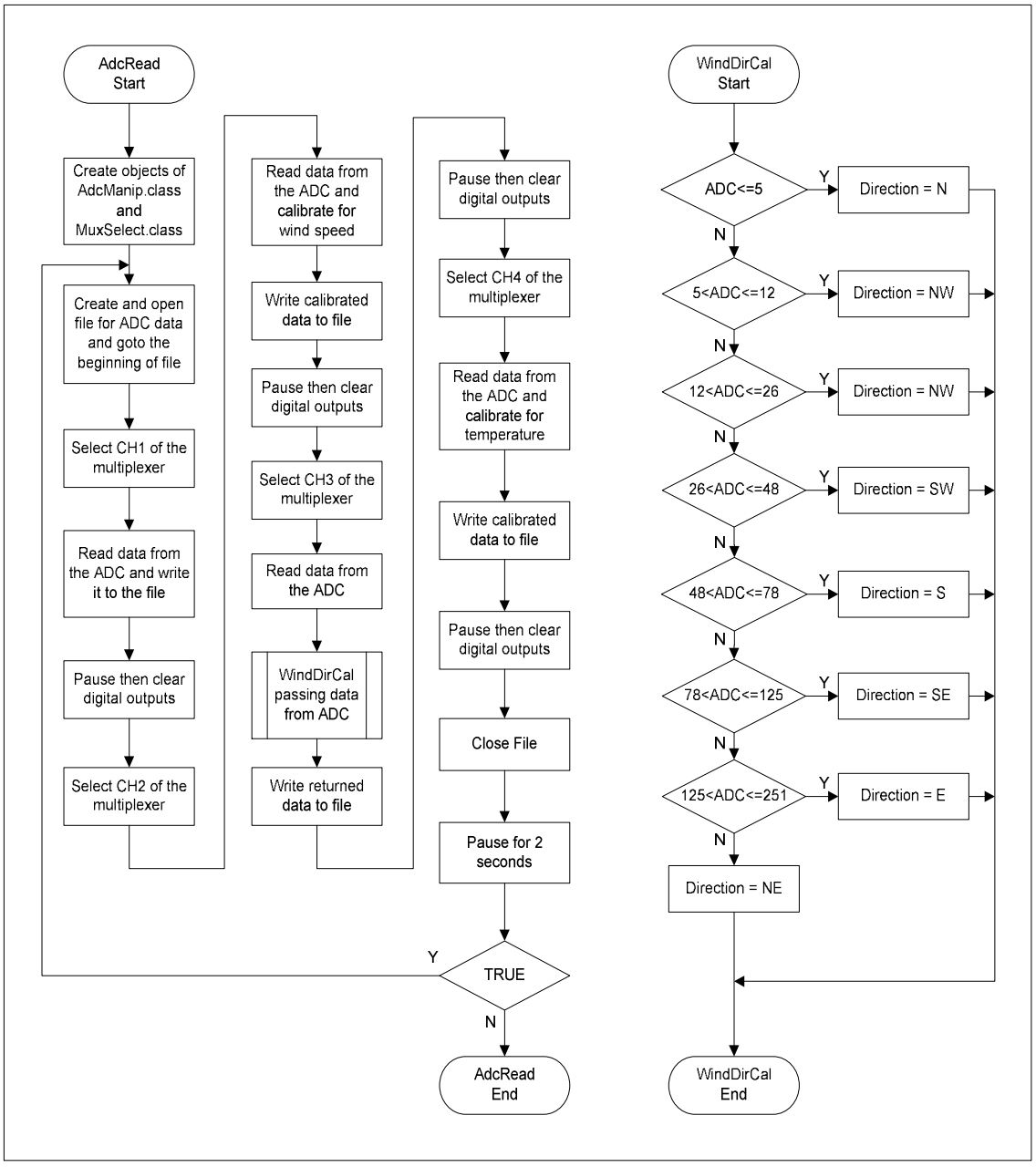


Figure 5.3.7 – Flowchart for AdcRead.class program

This program needs to run continuously in the background whenever TINI is turned on. By editing the start file (192.168.1.10/etc/.startup) for slush you can make programs automatically start when TINI is powered up or reset. The easiest way of doing this is to copy it from the /etc directory from TINI and edit it in notepad, then simply override the existing file on TINI with the modified version. The start up file now looks like the one below:

```
########
#Autogen'd slush startup file
setenv FTPServer enable
setenv TelnetServer enable
setenv SerialServer enable
##
#Add user calls to setenv here:
##
initializeNetwork
########
#Add other user additions here:
java /bin/TiniHttpServer.tini /etc/server.props &
java /docs/adc/Adc.tini
```

The last two lines have been added to automatically start the TiniHttpServer and the Adc.tini program.

Now that the ADC data is being updated to a file, the servlet has to open up the file and include the data within the HTTP code. Figure 5.3.7.1 shows the structure of the AdcServlet. This servlet is an extension of the standard Java HTTPServlet.

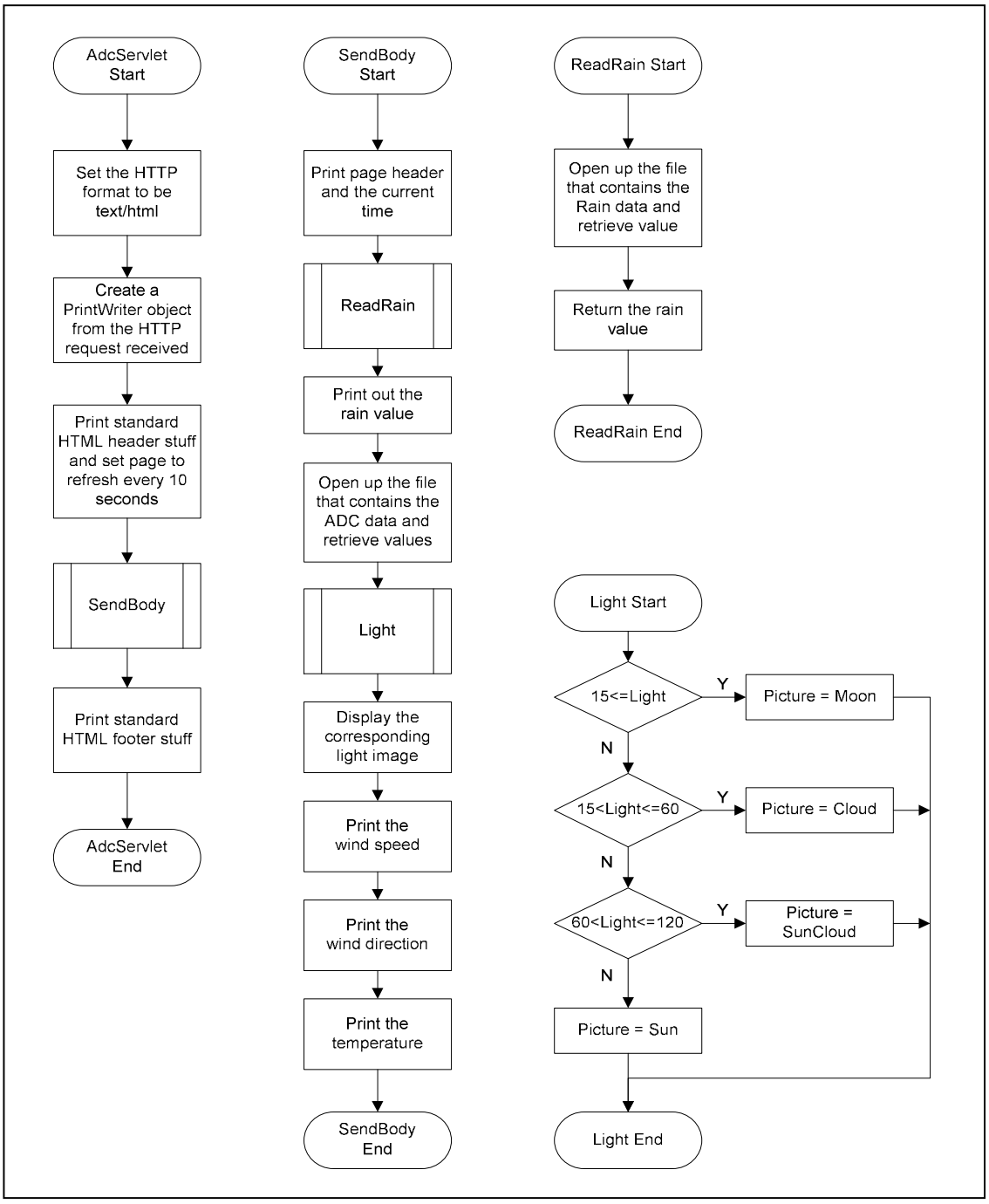


Figure 5.3.7.1 – Flowchart of the AdcServlet.class program

The first time of running this code for real did not work. This was simply because the paths for the files were wrong for the light level pictures and data files. This was due to the ADCServet being in a different directory to the other files. Once the file path was changed the light level of the room was displayed in real time to the web browser screen. This was a major milestone for the project. To test this fully the multiplexer hardware needed to be built with multiple sensors attached. Figure 5.3.7.2 is a screen shot of the webpage. For more details about the rainfall function in figure 5.3.7.1 see section 5.3.8.

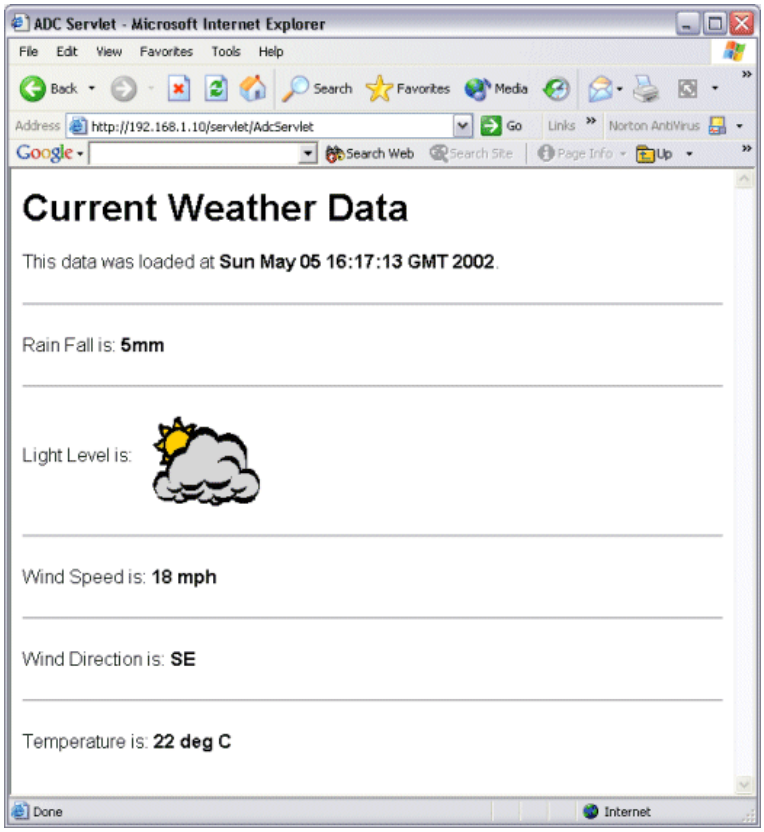


Figure 5.3.7.2 – A screen capture of the AdcServlet

As you can see instead of printing a number to screen for the light level a picture is displayed representing the level of light. There are 4 images; moon, cloud, sun/cloud and sun. See page ?? of the appendices for the actual code.

5.3.8 Checking Rain Fall

The rainfall value is slightly different to the other signals because it only changes between logic 1 and logic 0. A logic 1 pulse signifies that the rainfall has increased by 1mm (see section 5.5.5 for more details). The other sensors have a varying voltage as the input therefore they are fed into the ADC through the multiplexer. The fact that the rainfall sensor only changes logic states means that it can be used as a digital input rather than an analogue. A program was made to continuously check the state of the digital input port. Every time the logic state changes from a 0 to a 1 a value in a file gets updated. The AdcServlet program simply reads the value of the rain data file and displays it to the screen at the same time as it reads the ADC data file. This appears to the user that all data is being read from the sensors and displayed at the same time. The value of this file gets reset to 0 at the end of each day. Figure 5.3.8 shows the structure of the Rain.class program.

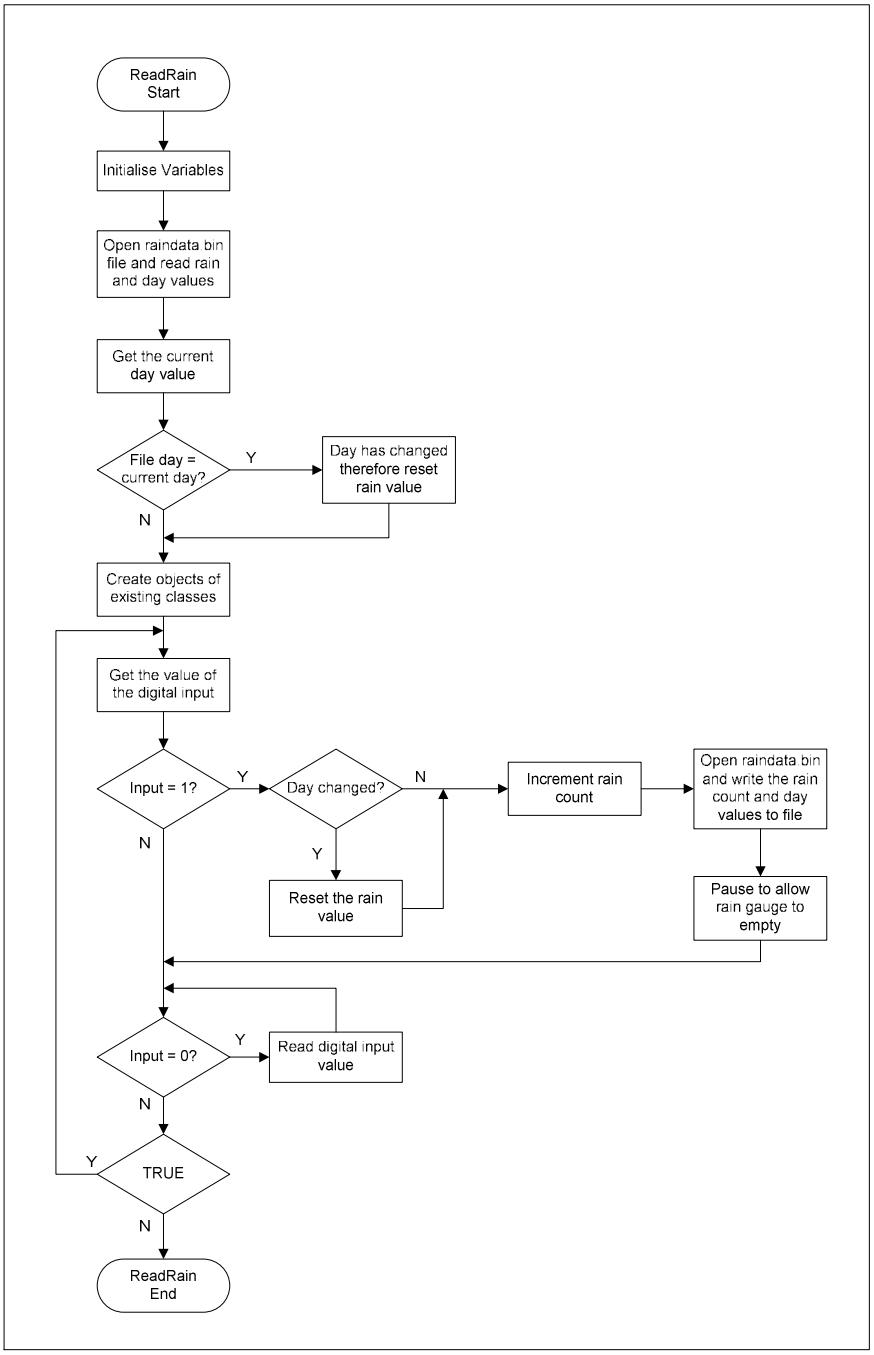


Figure 5.3.8 – Flowchart of the Rain.class program

See page 12 of the appendices for the actual code.

5.3.9 Displaying Data to the LCD

Now that data is successfully being displayed to a webpage (pending hardware development) it is time to look at the other functionality that is available on the Tutor board. What if you want to use TINI to monitor the weather sensors as a stand alone system? For example you want to make TINI and the sensors portable to take on field work. Currently you need a computer with a web browser to access the information. You could use the laptop computer to make it a portable system, or you could use the Liquid Crystal Display (LCD) that is

built onto the Tutor board. By displaying data to the LCD screen you could also use it during setting up of the system if you can not gain access to web browsing facilities. This would allow the user to check that data is being read correctly.

A program was developed to display the sensor data to the LCD screen. Users can select which sensor data is displayed on the LCD by changing the state of the onboard dip switches. Only dip switches 1 to 5 are used. If a dip switch is on then the data for that sensor will be displayed in turn. If none of the dip switches are on then the message “Web Server Mode” is displayed to indicate that the data can still be viewed through a web browser. This program simply uses the existing LCDManipFormat class to manipulate the LCD screen and the existing DipManip class to manipulate the dip switches. Figure 5.3.9 shows the structure for the LCDWrite program.

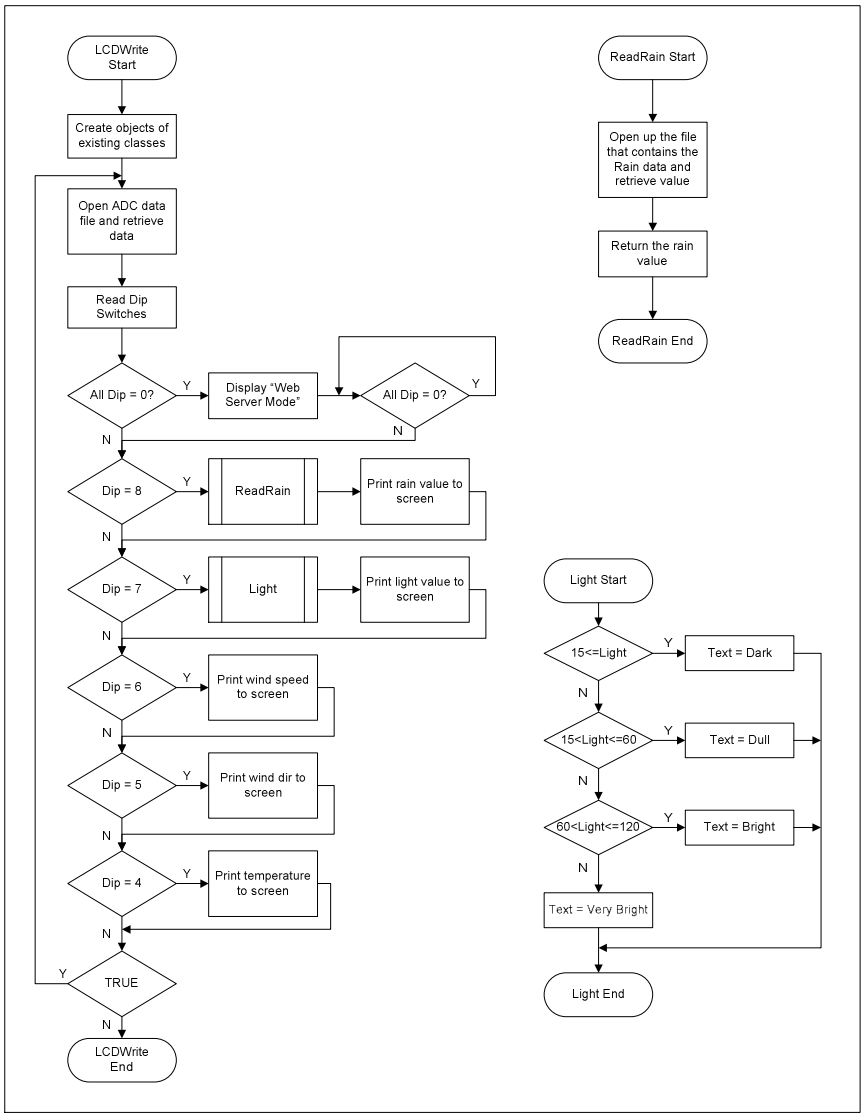


Figure 5.3.9 – Flowchart of the LCDWrite.class Program

As you can see from figure 5.3.9 the dip values used to check were actually 8,7,6,5,4, instead of 1,2,3,4,5. This is because the dip switch package has been put on the wrong way round on the Tutor board. The number 1 dip switch is normally the Least Significant Bit (LSB) of value 1 but in this case it is the

Most Significant Bit (MSB) of value 128. For this reason the opposite dip switch values are used so that when the user selects dip switch 1 the 1st sensor data is displayed. This program was also added to the start up file for TINI.

5.3.10 Temperature Graph

A program was developed towards the end of the project to take a temperature reading every hour for 24 hours then create a graph of the data recorded. This allows any user to see the temperature range over the last day. Each day the temperature graph gets updated with the previous day's results. This is really to demonstrate how to draw basic graphs in HTML code. It also has quite a practical side to it. There are two stages to this program, one takes a temperature reading every hour and stores the value to a file, the other converts the recorded values into a graph at the end of the 24hrs. Figure 5.3.10 shows the structure for the temperature graph program.

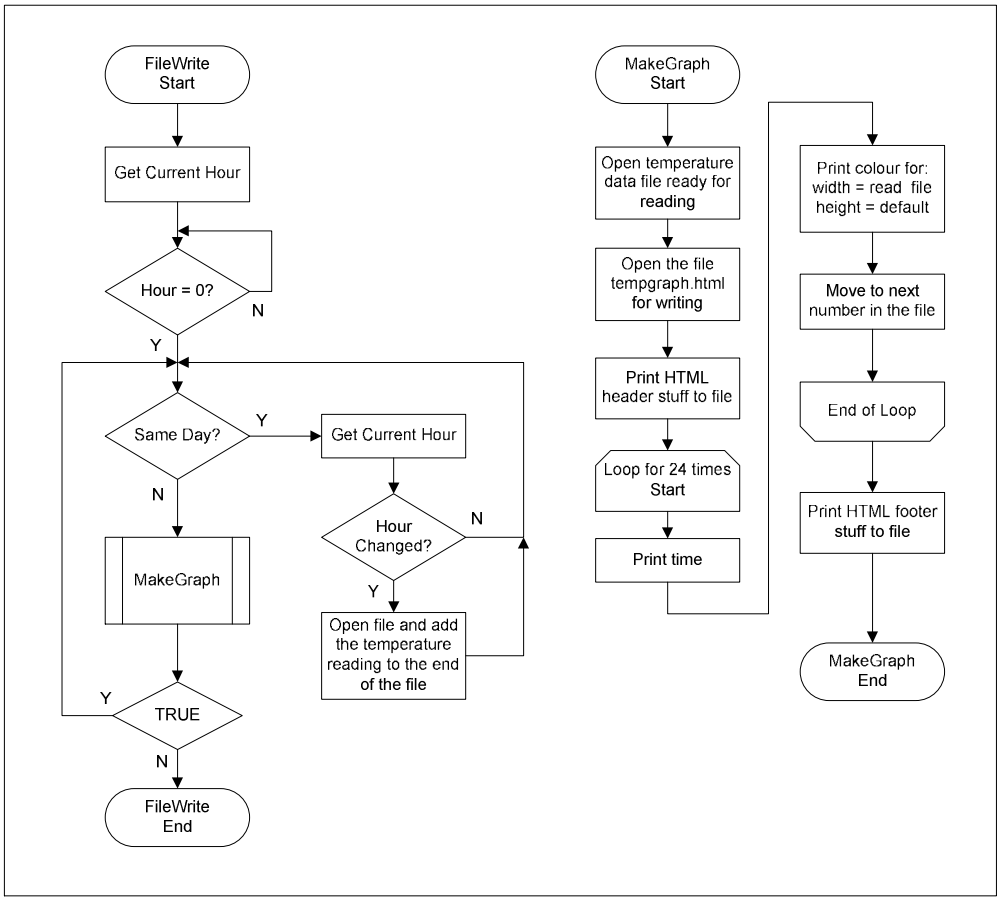


Figure 5.3.10 – Flowchart of the FileWrite.class program

For purposes of testing this program was run using minutes rather than hours. So a graph was produced every 24 minutes. This saved waiting lots of time. The program has also been tested over several days sampling every hour. See page 7 of the appendices for the actual code.

5.4 Website Development

The user interface for this project is a webpage that was designed to incorporate the AdcServlet show in figure 5.3.7.2. This web page is part of a larger website designed to demonstrate the flexibility and power of the TINI machine. There are various pictures and details of the TINI hardware with links to other useful sites. A description of this specific project takes up most of the website. It also has support section to help new TINI users setup their systems easily. It outlines the problems that were encountered during the early stages of this project and offers a step by step guide of how to setup TINI on a network. This is using the setup program discussed earlier in section 5.2.1.

Most of the web pages within this website were designed initially by hand using HTML. However Microsoft FrontPage was used later on to adjust the layout for some of the pages and to add in extra features. Navigation around the website is controlled by a menu bar down the left hand side of the screen. This menu bar was designed using Macromedia Flash 5 to enable the buttons to change state with the user input. See section 3.3.2 for more details about Flash 5.

The main page used to display the weather station sensor data, actually consists of 4 different web pages. The menu bar down the left hand side is one page, the main page to the right is another. Then embedded within this main page there are two pages served from TINI. The first being the weather station data (AdcServlet) and the other being the previous days Temperature graph. These were included in the main page by using inline frames. An inline frame is basically a webpage within a webpage. The AdcServlet has been programmed to refresh itself automatically every 10 seconds. Figure 5.4 show a screen shot of the main webpage with the AdcServlet embedded within it.

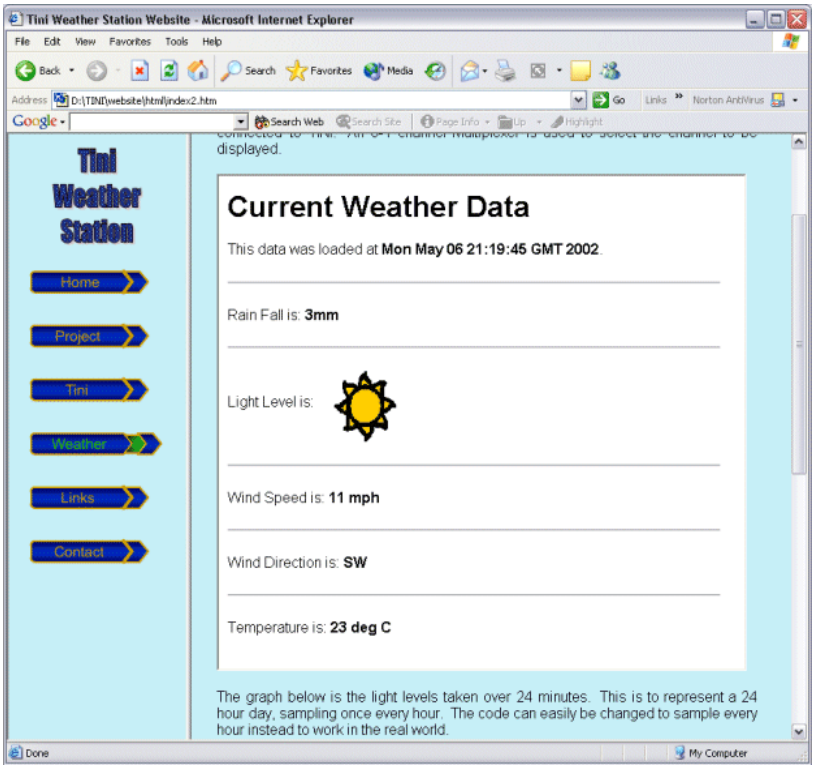


Figure 5.4 – Screen shot of the main weather webpage

5.5 Sensor Development

The weather station sensors that were used for this project are as follows:

- Sun Light Level
- Temperature
- Wind Direction
- Wind Speed
- Rain Gauge

These 5 sensors were designed and developed for this project to save costs. Most of these sensors would not be suitable to operate in the real world but are perfectly suitable for the purposes of this project. The sensors are described below:

5.5.1 Sun Light Level

The sensor to measure the level of light for this project is a simple Light Dependant Resistor (LDR). As the name suggests the resistance across the terminals varies depending of the amount of light it receives. It has a resistance of 2MΩ when dark (0 Lux) and a resistance of 33KΩ at 10 Lux. Lux is the measurement of the intensity of light. Originally one Lux was the light falling on a surface one foot away from a "candle". Now it is a carefully calibrated light source. With very bright light the resistance of the LDR becomes insignificant. Figure 5.5.1 shows the LDR.

Figure 5.5.1

The LDR was purchased from Maplin Electronics for £0.99. To see how this is integrated with the main circuit please see section 5.6.1.

5.5.2 Temperature

The temperature sensor is another very simple device. It is a LM335 precision temperature sensor package. This is a 2 terminal Zener diode which has a breakdown voltage directly proportional to the absolute temperature at 10mV/^{o}K. So at 0^{o}C (273^{o}K) the output voltage is 2.73v using the circuit shown in figure 5.5.2.2.

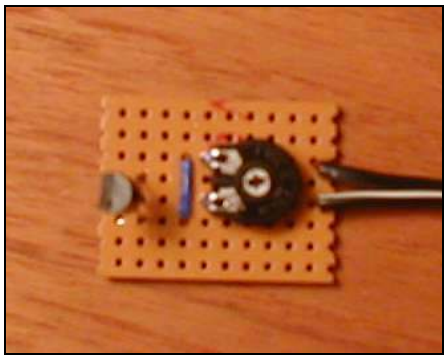


Figure 5.5.2.1 – Sensor Hardware

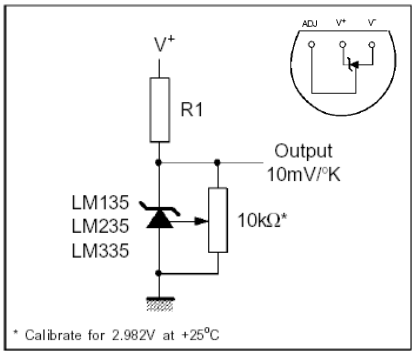


Figure 5.5.2.2 – Circuit from data spec

The LM335 sensor is calibrated according to figure 5.5.2.2 using a 10KΩ potentiometer. The actual temperature sensor hardware is show in figure 5.5.2.1. The LM335 was purchased from Maplin Electronics for £1.49. To see how this is integrated with the main circuit please see section 5.6.4.

5.5.3 Wind Direction

The wind direction sensor for this project is quite a crude solution. This is only suitable for demonstration purposes. A standard 100KΩ rotary potentiometer was purchased and taken apart to remove the stop inside. This allows the spindle to rotate all the way round. Before the stop was removed the degree of rotation was limited to about 350^{o}. However, removing the stop does mean that there is about a 10^{o} section that has zero resistance. This section was simply set to be the value for North. There are rotary position sensors that are designed for this type of problem but unfortunately even the cheapest cost about £15. Due to budget restraints the potentiometer option was chosen. Figure 5.5.3 shows the wind direction sensor.

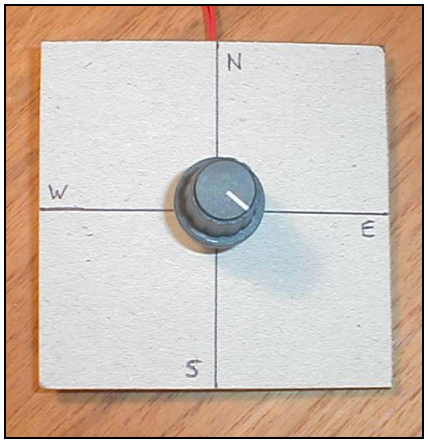


Figure 5.5.3 – Wind direction sensor

In this current state it could never read the wind direction because it doesn't even have a vain to be moved with the wind. The main thing that this sensor shows is the concept. Users can demonstrate the functionality by simply turning the potentiometer. To see how this is integrated with the main circuit please see section 5.6.3.

5.5.4 Wind Speed

This sensor was a bit more complicated to create than the previous ones. An electronic signal was needed as the output from a rotating set of plastic cups. These cups needed to rotate when placed in a wind stream. A search on the internet for 'home made anemometer' was carried out and a link to the www.otherpower.com website was found. This website explained in detail how to make your own anemometer.

There were two different options both very similar in approach. The first was to use a simple dc motor as a voltage generator. This would produce an output voltage linear to the speed that the spindle was rotating at. However, a dc motor looses its linearity at higher speeds and the brushes inside a dc motor eventually wear out. The other alternative (which is a far better solution) is to use a brushless stepper motor. This retains linearity at high speeds because the voltage can also be compared to the rotational frequency. A stepper motor could typically have 24 windings to provide a step size of 15^{o}. When being turned manually, it will produce 24 pulses for every rotation. This pulse count can be read as a frequency and converted into speed. This makes the stepper motor far more accurate and linear. Also a stepper motor does not use brushes like dc motors, therefore they are far more durable and less likely to wear out.

For this project it was decided to use a simple 6v dc motor. Although this was the less accurate and less durable option it worked out to be the cheaper one. As mentioned before, the main aim of this project is to provide a demonstration tool, so the accuracy of the equipment is not really the main priority. Equipment accuracy and reliability can be looked at as a future development. Figure 5.5.4 shows the plastic cup construction, mounted on top of a 6v dc motor.

Figure 5.5.4 – Wind speed sensor

All of the parts for this sensor were sourced from various different resources, so this sensor did not actually cost anything (other than time) to build. To see how this is integrated with the main circuit please see section 5.6.2.

5.5.5 Rain Gauge

This was the most complex and time consuming sensor to design and build. A method was needed to measure electronically the amount of rainfall over a period of time. The most common way to measure the amount of rainfall without electronics is by using a funnel and a container with a measurement scale up the side. This can be converted to an electronic scale by using the natural conduction characteristics of water. See figure 5.5.5.1 for details.

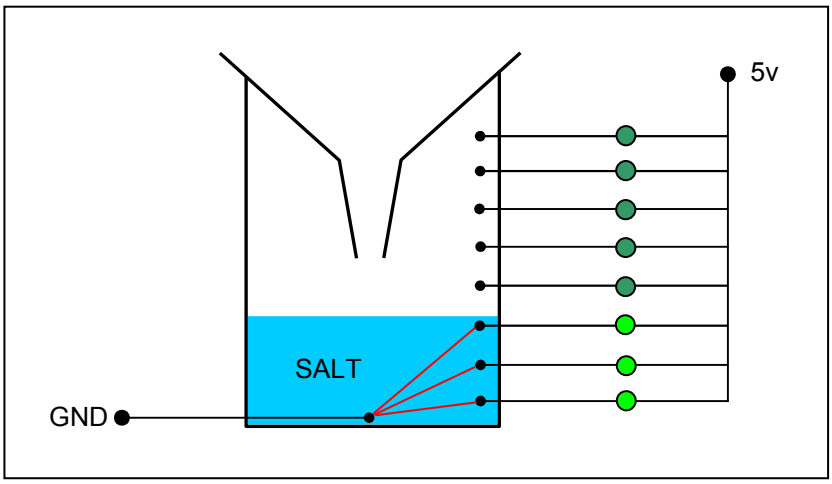


Figure 5.5.5.1 – Basic electronic rain gauge

For this rain gauge you have one open ended wire on the bottom of the container that is linked to ground. You then need several more open ended wires that are positioned up the side of the container. All of these are linked to a low voltage dc supply (5v) through Light Emitting Diodes (LEDs). All you do then is add a teaspoon of salt to the bottom of the glass and wait for rain. As the container fills up and reaches the first wire the water conducts electricity and switches the first LED on. As the water gets deeper more LEDs turn on giving you an indication of how full the container is. The salt in the bottom dissolves in the water and helps to conduct the electricity. These outputs could be fed into a controller instead of using LEDs for the display.

However there are some major draw backs to this approach. The main problem is that at the end of every day the water has to be emptied from the container by hand and more salt added for the next day. This is hardly a practical solution.

Another approach is a self emptying method designed around a balanced spoon. The idea is to guide the rain water onto the spoon using a funnel. When the spoon gets full enough the spoon tips off balance and empties the spoon onto the ground. Therefore if you know the volume of the spoon and the size of the capture area (the funnel) you can work out the amount of rainfall by the number of times the spoon tips. To record this value a small magnet was attached to the back of the spoon. When in its resting position the magnet holds a reed switch closed and every time the spoon tips the circuit gets broken. All you have to do is record the amount of times the circuit gets broken and you

have a rainfall value. Figure 5.5.5.2 shows a photo of the developed rain gauge for this project.

Figure 5.5.5.2 – Complete rain gauge

This rain gauge has a two tiered effect because for the purpose of demonstration, the water can not simply drop onto the floor. The emptied water is guided into a funnel and the water can be collected in a container placed beneath. It took several attempts before the correct shape for the spoon was found. The cup of the spoon used here is quite long; this length helps to completely empty the spoon as it tips. The water collects towards the back of the spoon to start with and gradually creeps forward as it fills. At the point that the spoon becomes unbalanced all of the water rushes forward to the front of the spoon which off balances the spoon even more, thus in one swift action completely emptying the spoon. To see how this is integrated with the main circuit please see section 5.6.5.

5.6 Hardware Development

This section looks at the electronic hardware that was produced to integrate all of the sensors described in section 5.5 with the TINI board. As mentioned in section 5.3.4 the TINI board only has one ADC which means that the inputs from the sensors need to be passed through a multiplexer controlled by TINI. The multiplexer channels can then be quickly scrolled through to read in all of the sensor data within a spilt second. This means that TINI can still have visibility of all the sensors.

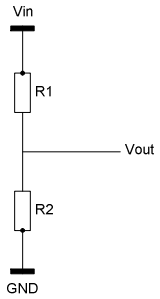
The ADC on the tutor board can read in values between 0 and 5 volts. This means that all of the sensors have to be calibrated to produce values within this range. Each sensor has a signal conditioning circuit where the data is calibrated to range between 0 and 5 volts. The output from each of these

circuits is fed in to the multiplexer. All of the circuits are powered from the 5v supply of the Tutor board via ribbon cables. The next few sections describe the circuits for each sensor.

5.6.1 Light Sensor

The LDRs resistance can vary between a few hundred ohms when light and 2MΩ when dark. The easiest way of converting this resistance into a voltage is by using simple voltage divider methods. We ideally want 5 volts into the multiplexer when it's very bright and nothing in when it's completely dark. The voltage divider rule is a follows:

$$Vout = \frac{R2}{R1 + R2} Vin \quad \text{where} \quad R1 = LDR$$



If the value of R2 is too small (say 100Ω) then even with the LDR at only 100Ω the maximum voltage is only 2.5 volts. If the value of R2 is too big then the minimum value of Vout never really reaches 0. A compromise of 1KΩ was decided. This provides a minimum voltage of 2.5mV (LDR = 2MΩ) and a maximum voltage of 4.54v (LDR = 100Ω). The circuit can be seen in figure 5.6.1.

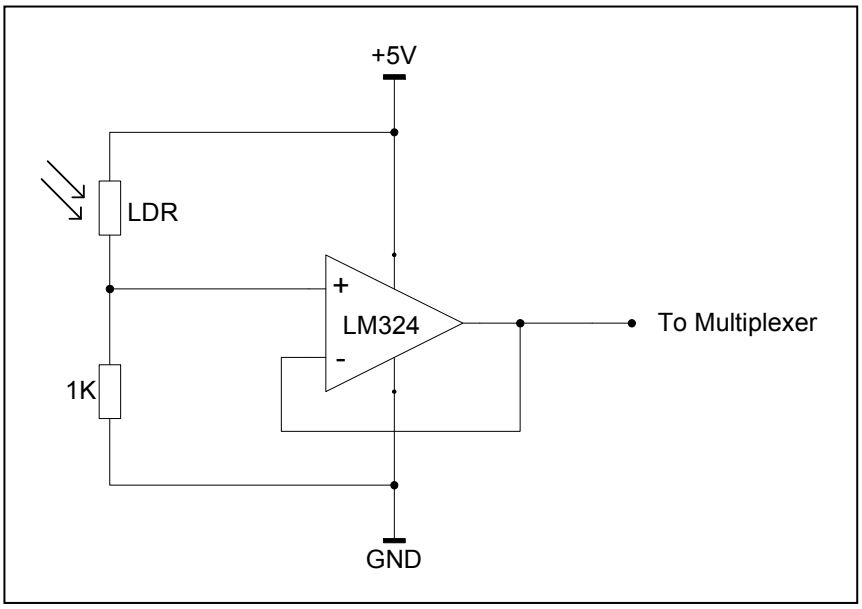


Figure 5.6.1 – Light sensor circuit diagram

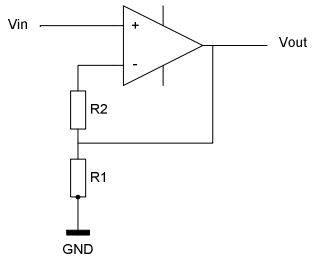
A non-amplifying buffer has been placed between the output from the voltage divider and the input to the multiplexer. This is to stop the large varying resistance of the LDR having an unwanted effect on the multiplexer input impedance.

5.6.2 Wind Speed

The dc motor is rated a 6v for 2400rpm. Therefore if the cups were rotating at 2400rpm the voltage would be too high for the ADC input. Some calibration values were needed from the wind speed sensor to get an idea of the true input range for different wind speeds. The simplest way of calibrating the motor was to attach a volt meter to the trailing leads from the motor. The wind speed sensor was then taken for a drive in a motor car. The sensor was then held out of the side window of the car by one person whilst another viewed the voltmeter reading and made notes of voltage and the cars speed. This was done for speeds up to 60mph. It was found that even at 60mph the voltage reading did not get much higher than 2.2v. This was a bit lower than expected.

Although this calibration method was very crude it did provide some useful results. The output from the motor would only have a chance of reaching 6v if the wind was nearly 140mph! This is never likely to happen. It was decided that the output from the motor could be amplified by 2 and it still would require a 75mph wind to reach 5 volts. Passing the signal through an operational amplifier would also naturally limit the voltage on the input because the output voltage is restricted to the supply voltage minus 1.5v. Therefore the voltage into the ADC will never be over 3.5v. The resistor values required for a gain of two can be worked out as follows:

$$Vgain = \frac{R2 + R1}{R1}$$



To give a gain of 2 the values of R1 and R2 have to be the same. The resistor value was chosen to be 1.5K because it is a medium value resistor and there were plenty lying around. Figure 5.6.2 shows the circuit diagram for this sensor.

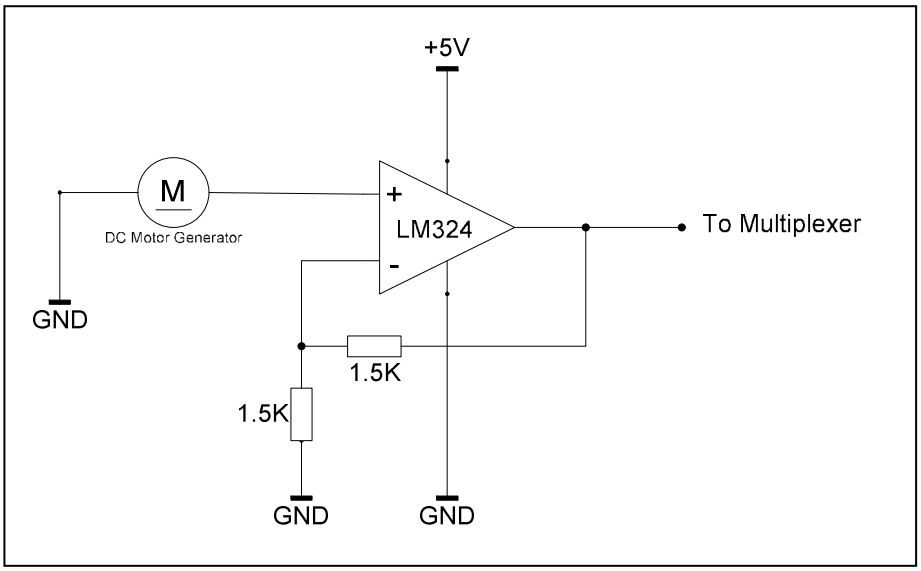
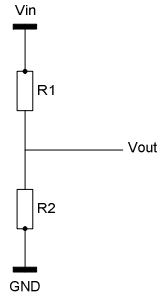


Figure 5.6.2 – Wind Speed sensor circuit diagram

5.6.3 Wind Direction

This circuit is very similar to the Light sensor. The voltage divider rule is used to calculate the value of R2 with the potentiometer being a 100k logarithmic. When carrying out testing with a linear pot the voltage was responding in a logarithmic fashion. To make the voltage output more linear a logarithmic pot was used.

$$Vout = \frac{R2}{R1 + R2} Vin \quad \text{where} \quad R1 = Pot$$



To provide a suitable voltage swing on the output a resistor value of 4.7KΩ was selected. This provides a minimum voltage of 0.2v and a maximum voltage of 5v (when the pot is in its dead zone at 0Ω). The circuit for this sensor is show in figure 5.6.3 below.

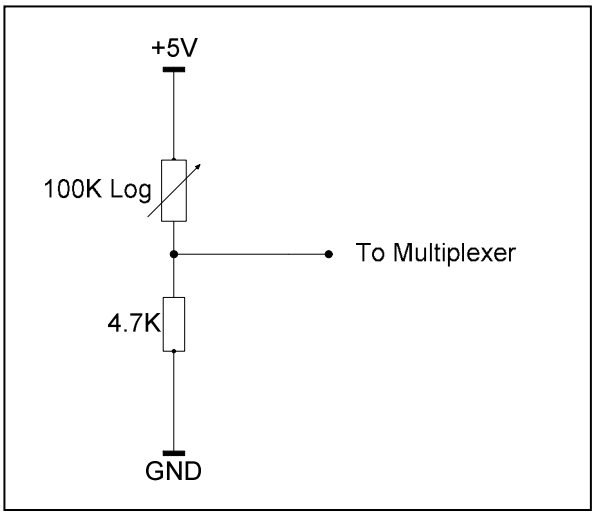


Figure 5.6.3 – Wind direction sensor circuit diagram

5.6.4 Temperature

The LM335 temperature sensor works between -40°C and 100°C. For this project the temperature range will be limited to -10°C to 90°C. At -10°C the voltage out from the LM335 is 2.63v. To calibrate this to the ADC specification the output needs to be 0v at -10°C. Therefore a -2.63v offset needs to be applied to the input temperature sensor output. This is done by using a voltage inverter package (ICL7660S) to invert the 5v supply to a -5v. Then by using a 4.7KΩ potentiometer this value can be tweaked to give -2.63v. This is summed with the output from the LM335 temperature sensor to give 0v at -10°C. The maximum output will now be 1.1v at 100°C.

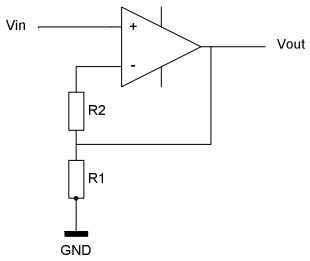
To calibrate this new range to the ADC specification the new temperature sensor output can be multiplied by 5. This will produce a voltage of 5 volts when the temperature is 90°C. The voltage amplification is carried out by an op-amp. The resistor values for a gain of 5 are as follows:

$$Vgain = \frac{R2 + R1}{R1}$$

$$5 = \frac{R2 + 10K}{10K}$$

$$(5 \times 10K) - 10K = R2$$

$$R2 = 40K$$



The nearest value for this is 39KΩ. The circuit diagram for this is shown in figure 5.6.4.

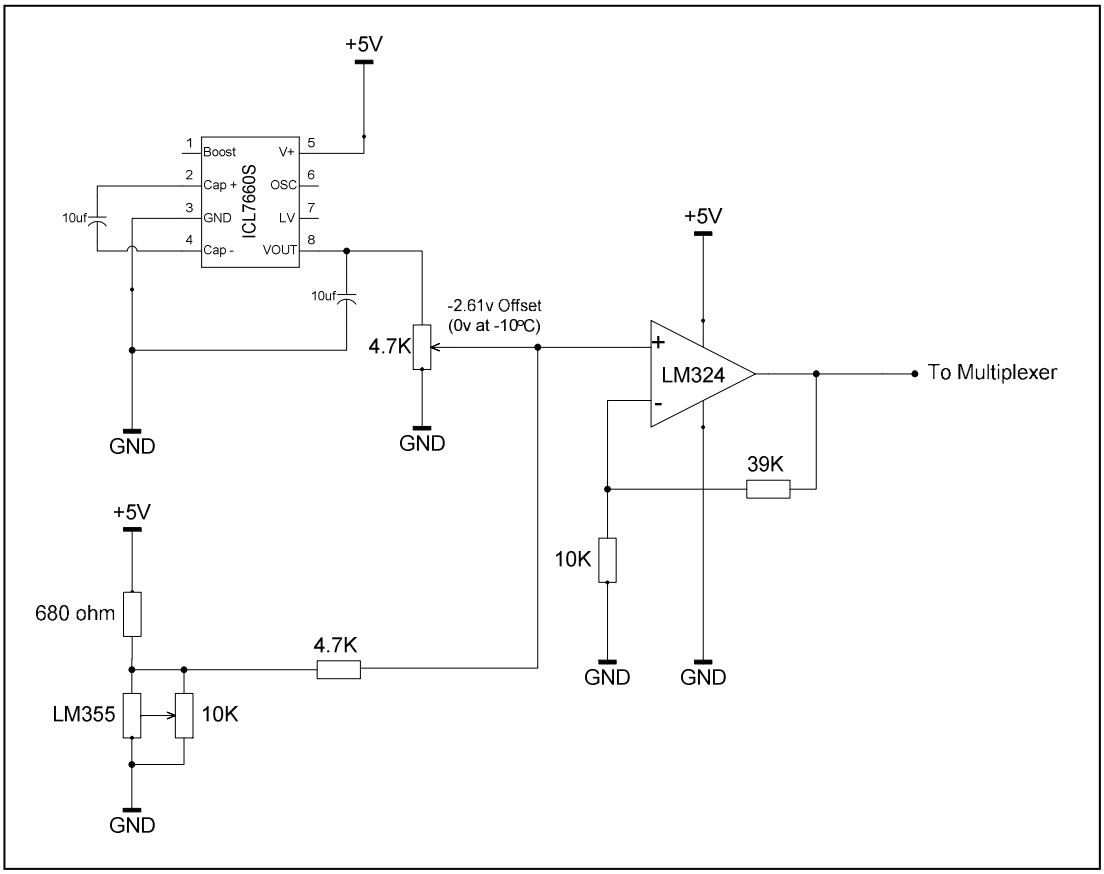


Figure 5.6.4 – Temperature sensor circuit diagram

The actual maximum temperature that will be available from this circuit is about 70°C. This is because the maximum output from the op-amp is actually 5-1.5v which is 3.5v. Therefore 5/100 = 0.05, then 3.5/0.05 = 70°C. This is still far hotter than the atmosphere will ever get.

5.6.5 Rain

This is the simplest circuit of all the sensors. Every time the rain gauge tips the circuit is broken by a reed switch being released. To make this more logical for the digital input an inverter has been used to turn a break in the circuit into a positive pulse. A 1KΩ resistor ties the input to ground when there is a break in the circuit. The inverter then outputs logic 1. This gets fed directly to the digital input for the TINI board. The circuit is shown in figure 5.6.5 below.

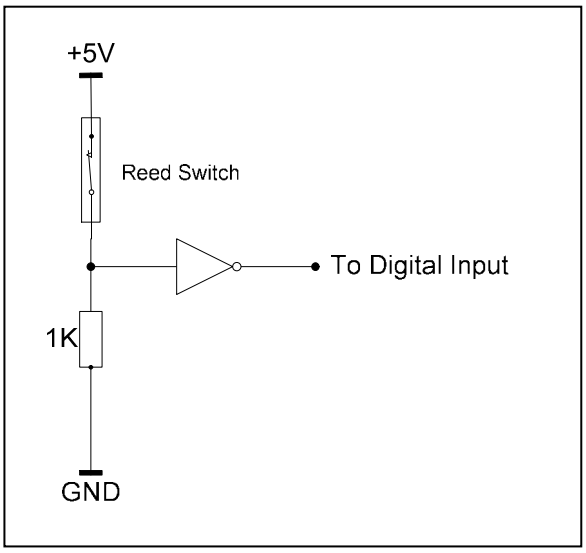


Figure 5.6.5 – Rain sensor circuit diagram

5.6.6 Complete Circuit

All of the outputs from the sensors (apart from the rain sensor) are connected directly to an 8-1 multiplexer. An 8 input multiplexer was not needed (a 4 input would have been sufficient) but this at least leaves room for expansion at a later date. The channel select pins for the multiplexer are controlled by the digital outputs from the TINI board. These are joined to the multiplexer board via ribbon cables. These cables also provide power to the board. The rain sensor circuit links directly to the digital input port on the TINI board via ribbon cables. The complete circuit diagram is shown in figure 5.6.6.

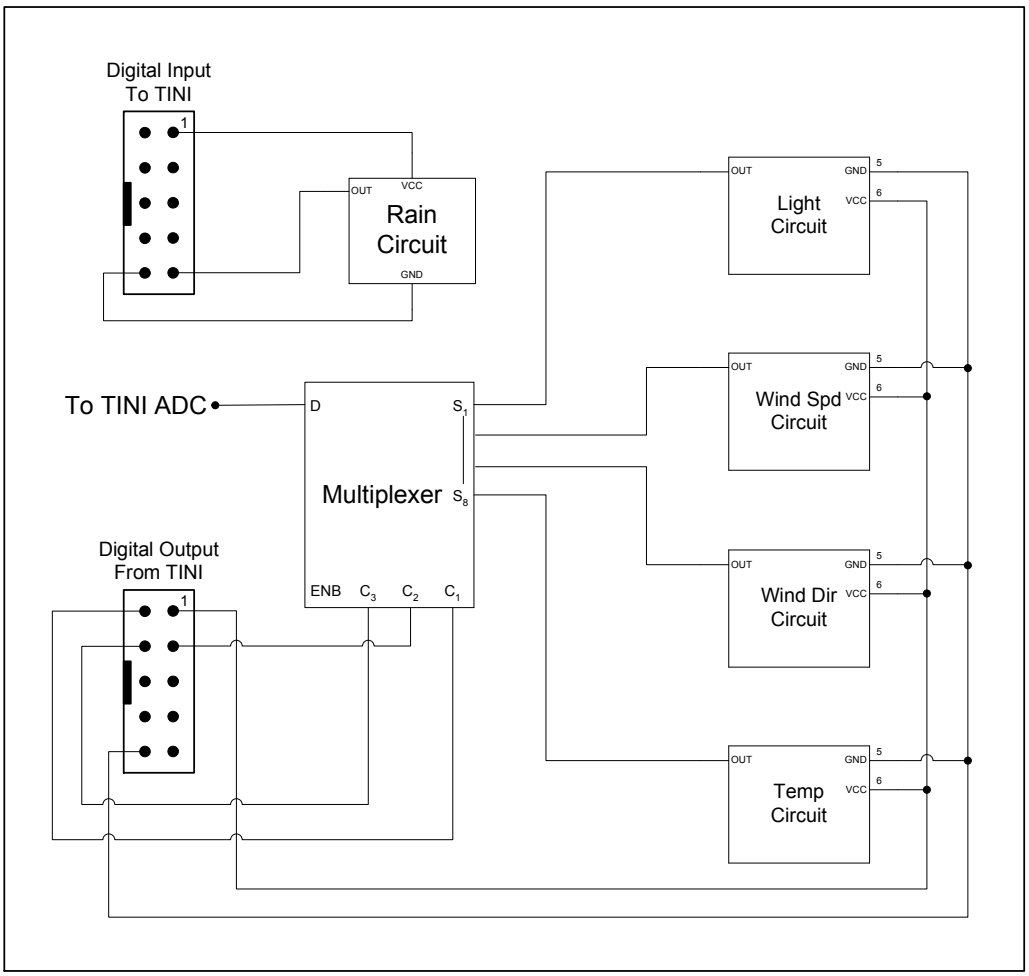


Figure 5.6.6 – Multiplexer circuit diagram

6 Testing

6.1 Software

The software was tested during the development stage by using the onboard LDR sensor as the main input to the system. If the data was being received from the LDR correctly then there was a high chance that it would run ok from the external hardware. This did mean that for a majority of the development process there was only ever one input rather than all 5. This was until the hardware had been developed. When combining the external hardware there were a few things that needed to be adjusted.

Firstly there were several delays that needed tweaking. These were mainly to do with the time between the multiplex switching, to make sure there was enough time to get the correct data. When the hardware was added each element needed calibrating in code, so some careful calculations were carried out to do this (see AdcServlet in the appendices for more details).

6.2 Hardware

The hardware was first tested as individual elements, then once satisfied with the functionality of each, they were combined in stages and tested. Eventually the system was complete and tested as a whole.

6.2.1 Individual

The conditioning circuits for each sensor were built and tested at first on bread board. This allowed component values to be easily changed if required. Once satisfied with the performance of a circuit it would get transferred to the main strip board. The multiplex circuit was tested first because this is the building block of the system. As each circuit was added, the main circuit would get tested again to make sure that everything still worked alright. The first main test of the multiplexer was when the first conditioning circuit was added. This was the sun light level. The reason for this being the first is that it should react pretty much the same as the onboard LDR. Once this circuit was successfully being read through the multiplexer and into the ADC it was then a simple case of adding in the remaining circuits.

No real problems encountered other than accidentally using a dual polarity op-amp instead of a single polarity one for the amplifying circuit. This was soon realised and quickly resolved. The board quickly got quite cramped with cabling but this is really expected with strip board. It would be nice to put the multiplex circuit onto PCB in the future.

6.2.2 Whole System

When all of the external hardware was complete and the software had been calibrated, testing on the full system took place. The powering up operation of TINI was checked to make sure that all the required programs were executing

on start up. The web page was tested to make sure it was connecting to the correct TINI servlet. All of the sensors were individually tested to see of they were displaying the correct data to screen with the correct units being displayed. The temperature sensor was compared to a digital thermometer to make sure that both increased by the same amount as they were heated up or cooled down.

6.3 LAN

All of the testing for TINI was carried out over a Local Area Network (LAN). The web pages were stored locally rather than on an internet server. This was because the files were continuously being updated and access times are far quicker from a local storage.

The system was checked for multi-user capabilities. 3 or 4 computers accessed the weather data from TINI at the same time without having any problems.

6.4 WAN

Implementing TINI on the actual internet proved slightly harder. In order for a TINI machine to actually be a node on the internet (in the same way it is on the LAN) in requires a fixed IP address. These are not that easy to come across a potentially can cost a great deal of money. Due to this, other approaches were considered.

- The TINI machine remains on the LAN but has access to the internet via a gateway. TINI can regularly (say every ten minutes or so) FTP the latest weather update to an internet server that hosts a main website. All that would happen is a file containing the weather information would get updated so as you log onto the website a new weather.html page gets displayed every 10 minutes. This means that users can not directly link to the web pages on TINI but can still see the data.
- TINI could send out e-mails (say every few hour) to subscribers of a site that wish to receive constant weather updates. This is perfectly possible through standard slush commands.
- Another option is to implement a Point to Point Protocol (PPP). This is where the user wishing to receive data dials up TINI directly from a computer instead of going via the internet. This could be a possibility if TINI was being used to monitor something specific for a company. The technicians for the company could have the access number and link up through a phone line.

7 Outcomes & Problems

7.1 Finished State

The TINI weather station is fully accessible over a LAN via a windows web browser. TINI can monitor temperature, wind speed, wind direction, level of sun light and the amount of rain fall. This data gets displayed to the user in an easy to read format which updates every 10 seconds. This data is displayed as part of a larger website that provides support to other TINI users.

TINI can also display selected weather data to its LCD screen at the users request. There have also been experiments with the recording of temperature data and displaying this in a graph format on the webpage.

7.2 Problems

The main problems encountered were during the setting up and configuration stages of the project. There were initial problems with getting to grips with TINI and running programs like JavaKit.

One of the major problems was with accessing TINI in real time via a web browser. This was actually quite easy in the end as soon as TiniHttpServer had been discovered. This turned TINI into a web server which could then serve its own web pages and servlets.

7.3 Critical Analysis

The main element that had an effect on this project is time. There are so many angles left to take with this project and unfortunately time has run out. Time management (especially towards the start of the project) could have been better which would have eased the workload towards the end.

7.4 Future Development

There are many uses for the TINI machine that have come to light during this project. It would be nice to look at using TINI for control rather than just monitoring data. If TINI could be used to control equipment through a web browser it could become very interesting indeed.

8 Conclusion

The web based weather station using TINI has been a great success overall. A majority of the objectives that were set out initially have been met or exceeded.

TINI was successfully configured to serve its own web pages over a Local Area Network (LAN). Users are able to connect to the TINI machine through a windows based web browser and retrieve the simulated weather data in real time.

Other functionality of the Taylec Tutor board was explored. The real time weather data can also be displayed to the onboard LCD screen. This means that TINI and the weather sensors can be used as a stand alone machine to view the simulated weather data. This is completely separate from the network or any PC. The user can select the data to be viewed by setting dip switches.

The sensors used for this simulation were designed and made during this project. None of these sensors would really be suitable in the real world but they provide good enough simulation data for the purpose of this project.

The whole weather station system was tested thoroughly on a LAN with 6 other computers present. Multiple user access was tested on the LAN with 3 or 4 different computers all accessing the weather data at the same time. This was successful.

The scope for using TINI directly on the internet was investigated but found to be not possible at this point in time due to internet access problems. The best solution would be to have TINI on a LAN with access to the internet. TINI could then update an internet server with information at regular intervals.

During this project all of the learning aims were achieved. The power of TINI is incredible and learning all about it has been a joy.

9 Bibliography

9.1 Websites

http://www.ibutton.com/TINI/index.html - This is the main website for TINI

http://www.dalsemi.com – Manufactures of TINI

http://www.systronix.com – Interface board manufacturer

http://www.ibutton.com – Home of iButton

http://java.sun.com/j2se/1.3/ - Java SDK download

http://java.sun.com/products/javacomm/index.html - Communications API

http://www.ibutton.com/TINI/software/soft_order.html - TINI SDK download

The TINI interest group is an e-mail forum where you can network with other TINI developers. This is a major source of useful information about TINI.

www.smartsc.com – TiniHttpServer developers

http://java.sun.com/products/servlet/download.html#specs – Servlet classes

http://jakarta.apache.org/builds/jakarta-ant/release/v1.4/bin/ - Ant source

http://tiniant.sourceforge.net/ - TiniAnt source

9.2 Manuals & Data Sheets

1. The TINI Specification and Aser Guide – Dom Loomis
2. TINI Data Sheet – Dallas Semiconductors
3. LM324 Data Sheet – Fairchild
4. LM335 Data Sheet – SGS Thompson
5. HCF4051 Data Sheet – SGS Thompson
6. ICL7660S Data Sheet – Harris
7. MC10409UB Data Sheet – On

9.3 Books

1. Electronic Principles – Marvino 6th Edition
2. HTML 4 for the world wide web – Elizabeth Castro
3. JavaScript for the world wide web – Tom Negrino & Dori Smith

10 Appendices

10.1 Code

10.1.1 Adc.class – DOS

```
import java.lang.*;
import java.util.*;
import java.io.*;
import ADCManip;
import Interrupt;
import MuxSelect;
import com.dalsemi.comm.*;
import com.dalsemi.system.*;


// ADC Read Routine

public class Adc
{
  private static final void pause(int delay_)
  {
    try
    {
      Thread.sleep(delay_);
    }
    catch (InterruptedException e_)
    {
    }
  }

  public static void readADC()
  {
    try
    {
      System.out.println("ADC Tester");

      ADCManip adc=new ADCManip();
      MuxSelect select=new MuxSelect();

      while(1==1)
      {
     select.add(MuxSelect.CH1);
     System.out.println("Channel 1: " + adc.readLinearized());
        pause(500);
     select.clear();

     select.add(MuxSelect.CH2);
     System.out.println("Channel 2: " + adc.readLinearized());
        pause(500);
     select.clear();

     select.add(MuxSelect.CH3);
     System.out.println("Channel 3: " + adc.readLinearized());
        pause(500);
     select.clear();

     select.add(MuxSelect.CH4);
     System.out.println("Channel 4: " + adc.readLinearized());
        pause(500);
     select.clear();

     pause(2000);
```

```
      }
    }
    catch(IllegalAddressException err_)
    {
      System.out.println(err_.toString());
    }
  }

  public static void main(String[] args)
  {
      readADC();
  }

}
```

10.1.2 Adc.class

```
////////////////////////////////////////////////////////////////////////
//Adc.class source file
////////////////////////////////////////////////////////////////////////

import java.lang.*;
import java.util.*;
import java.io.*;
import com.dalsemi.comm.*;
import com.dalsemi.system.*;

import com.taylec.tini.ADCManip;
import MuxSelect;



// ADC Read Routine

public class Adc
{
  private static final void pause(int delay_)
  {
    try
    {
      Thread.sleep(delay_);
    }
    catch (InterruptedException e_)
    {
    }
  }

  public static void readADC()
  {

    try
    {

      ADCManip adc=new ADCManip();
      MuxSelect select=new MuxSelect();

      while(1==1)
      {

           RandomAccessFile file = new
           RandomAccessFile("./docs/adc/adcdata.bin", "rw");

           file.seek(0);

           select.add(MuxSelect.CH1);
```

```
          file.writeDouble(adc.readLinearized());
          pause(100);
          select.clear();

          select.add(MuxSelect.CH2);
          double windSpeed = adc.readLinearized();
          int Speed = (int)(windSpeed/3.4);
          file.writeInt(Speed);
          pause(100);
          select.clear();

          select.add(MuxSelect.CH3);
          double windDir = adc.readLinearized();
          String Dir = WindDirCal(windDir);
          file.writeUTF(Dir);
          pause(100);
          select.clear();

          select.add(MuxSelect.CH4);
          double temp = adc.readLinearized();
          int newTemp = (int)(temp/2.57);
          file.writeInt(newTemp);
          pause(100);
          select.clear();

          file.close();

          pause(2000);

      }
    }
    catch(IllegalAddressException err_)
    {
      System.out.println(err_.toString());
    }
    catch (FileNotFoundException e)
    {
      System.err.println("This shouldn't happen: " + e);
    }
    catch (IOException e)
    {
      System.err.println("Writing error: " + e);
    }
  }

  public static String WindDirCal(double wind)
  {
     String dir = "";

     if(wind<=5)
     {
       dir = "N";
     }
     else if(5<wind&&wind<=12)
     {
       dir = "NW";
     }
     else if(12<wind&&wind<=26)
     {
       dir = "W";
     }
     else if(26<wind&&wind<=48)
     {
       dir = "SW";
     }
     else if(48<wind&&wind<=78)
     {
       dir = "S";
```

```
      }
      else if(78<wind&&wind<=125)
      {
        dir = "SE";
      }
      else if(125<wind&&wind<=215)
      {
        dir = "E";
      }
      else
      {
        dir = "NE";
      }


      return dir;
  }

  public static void main(String[] args)
  {
      readADC();
  }

}
```

10.1.3 AdcServlet.class

```
//AdcServlet.java
//
//This Servlet reads data from the ADC and displays it as text on a web page.

import java.io.IOException;
import java.io.PrintWriter;

import java.util.Date;
import java.io.*;

import javax.servlet.*;
import javax.servlet.http.*;

import com.dalsemi.onewire.OneWireException;
import com.dalsemi.system.*;



public class AdcServlet
extends HttpServlet
implements SingleThreadModel
{
      private static final void pause(int delay_)
      {
            try
            {
               Thread.sleep(delay_);
            }
            catch (InterruptedException e_)
            {
            }
      }

      public void doPost( HttpServletRequest req, HttpServletResponse res)
      throws ServletException, IOException
      {
            doGet( req, res);
      }

      public void doGet( HttpServletRequest req, HttpServletResponse res)
```

```
throws ServletException, IOException
{

      // Do HTTP header stuff
      res.setContentType( "text/html");

      // Get Writer
      PrintWriter pw = res.getWriter();

      // Start HTML
      pw.println( "<html>");
      pw.println( "<head>");
      pw.println( "<meta http-equiv = 'refresh' Content = '10;
      192.168.1.10/servlet/AdcServlet'>");
      pw.println( "<title>Weather</title></head>");
      pw.println( "<BODY>");
      pw.println( "<body>");

      // Send HTML body
      try
      {
            sendBody( pw);
      }
      catch( OneWireException ibe)
      {
            pw.println( "<H1>Got a ADCException!</H1>");
            pw.println( ibe);
      }

      // Finish HTML
      pw.println( "</body></html>");
}

public void sendBody( PrintWriter pw)
throws OneWireException
{
      try
      {
            pw.println( "<H1><FONT FACE='Arial'>Current Weather
            Data</FONT></H1>");
            pw.print  ( "<FONT FACE='Arial'>This data was loaded at <b>");
            pw.print  ( getCurrentTime());
            pw.println( "</b>.<p>");

            pw.println( "<hr>");
            pw.println( "<br>");
            pw.print  ( "Rain Fall is:  <b>");
            pw.print  ( ReadRain());
            pw.println( "mm</b><br>");
            pw.println( "<br>");
            pause(50);


            //get data from file

            RandomAccessFile inputFile = new
            RandomAccessFile("./docs/adc/adcdata.bin", "r");

            inputFile.seek(0);

            double light = inputFile.readDouble();
            int windSpeed = inputFile.readInt();
            String windDir = inputFile.readUTF();
            int temp = inputFile.readInt();

            inputFile.close();
```

```
            pw.println( "<hr>");
            pw.println( "<br>");
            pw.print  ( "Light Level is:     ");
            pw.print  ( "<img src=\"../pics/");
            String pic = Light(light);
            pw.print  ( pic );
            pw.print  ( "\" align=\"middle\" width=\"105\"
            height=\"77\">");
            pw.println( "<br>");
            pw.println( "<br>");

            pw.println( "<hr>");
            pw.println( "<br>");
            pw.print ( "Wind Speed is:  <b>");
            pw.print  ( windSpeed);
            pw.println( " mph</b><br>");
            pw.println( "<br>");

            pw.println( "<hr>");
            pw.println( "<br>");
            pw.print  ( "Wind Direction is:  <b>");
            pw.print  ( windDir);
            pw.println( "</b><br>");
            pw.println( "<br>");

            pw.println( "<hr>");
            pw.println( "<br>");
            pw.print  ( "Temperature is:  <b>");
            pw.print  ( temp);
            pw.println( " deg C</b><br>");
            pw.println( "<br>");


      }
      catch (FileNotFoundException e)
      {
            System.err.println("This shouldn't happen: " + e);
      }
      catch (IOException e)
      {
            System.err.println("Writing error: " + e);
      }

}

public static int ReadRain()
{
    int data = 0;

    try
    {
           RandomAccessFile file = new
           RandomAccessFile("./docs/rain/raindata.bin", "r");

           file.seek(0);

           data = file.readInt();

           file.close();

    }
    catch (FileNotFoundException e)
    {
           System.err.println("This shouldn't happen: " + e);
                data = 998;
    }
    catch (IOException e)
    {
```

```
                System.err.println("Writing error: " + e);
                    data = 999;
            }

            return data;
        }



        public static String Light(double light)
        {
            String pic = "";

            if(light<=15)
            {
                pic = "moon.gif";
            }
            else if(15<light&&light<=60)
            {
                pic = "cloud.gif";
            }
            else if(60<light&&light<=120)
            {
                pic = "suncloud.gif";
            }
            else
            {
                pic = "sun.gif";
            }

            return pic;
        }

        public static String getCurrentTime()
        {
            return new Date().toString();
        }

}
```

10.1.4 FileWrite.class

```
import java.io.*;

public class FileRead
{
    public static void Read()
    {
     PrintStream outputFile = null;

     try
     {
           double data = 0;

           RandomAccessFile inputFile = new
           RandomAccessFile("./docs/graf/tempdata.bin", "r");

           //Go to that position.
           inputFile.seek(0);


           outputFile = new PrintStream(new
           FileOutputStream("./docs/graf/tempgraph.html"));

           outputFile.println("<HTML>");
           outputFile.println("<HEAD><TITLE>Temperature Graph</TITLE></HEAD>");
```

```
            outputFile.println("<BODY>");
            outputFile.println("<H1><FONT FACE='Arial'>Temperature Graph</H1>");

            outputFile.println("<SCRIPT LANGUAGE='JavaScript'>");

            int num = 0;
            String time = "";



             for(int i=0;i<24;i++)
            {
                  if(num<10)
                  {
                    time = "0"+num;
                  }
                  else
                  {
                    time = ""+num;
                  }

                  data = inputFile.readDouble();

                  outputFile.println("document.write('<font face=\"arial\"
                  size=3>' + \"" + time + ":00Hrs\");");
                  outputFile.println("document.write('<img src=\"clear.gif\"
                  width=10 height=10>');");
                  outputFile.println("document.write('<img src=\"red.gif\"
                  width=" + data + " height=10>');");
                  outputFile.println("document.write('<br>');");

                  num++;
            }

            outputFile.println("</SCRIPT>");

            outputFile.println("</BODY>");
            outputFile.println("</HTML>");

            outputFile.close();


              inputFile.close();


      }
      catch (FileNotFoundException e)
      {
             System.err.println("This shouldn't happen: " + e);
      }
      catch (IOException e)
      {
             System.err.println("Writing error: " + e);
      }
   }

}
```

10.1.5 FileRead.class

```
import java.io.*;

public class FileRead
{
    public static void Read()
```

```
{
 PrintStream outputFile = null;

 try
 {
        double data = 0;

        RandomAccessFile inputFile = new
        RandomAccessFile("./docs/graf/tempdata.bin", "r");

       //Go to that position.
       inputFile.seek(0);


       outputFile = new PrintStream(new
       FileOutputStream("./docs/graf/tempgraph.html"));

       outputFile.println("<HTML>");
       outputFile.println("<HEAD><TITLE>Temperature Graph</TITLE></HEAD>");
       outputFile.println("<BODY>");
       outputFile.println("<H1><FONT FACE='Arial'>Temperature Graph</H1>");

       outputFile.println("<SCRIPT LANGUAGE='JavaScript'>");

       int num = 0;
       String time = "";

       for(int i=0;i<24;i++)
       {
            if(num<10)
            {
              time = "0"+num;
            }
            else
            {
              time = ""+num;
            }

                 data = inputFile.readDouble();

            outputFile.println("document.write('<font face=\"arial\"
            size=3>' + \"" + time + ":00Hrs\");");
            outputFile.println("document.write('<img src=\"clear.gif\"
            width=10 height=10>');");
            outputFile.println("document.write('<img src=\"red.gif\"
            width=" + data + " height=10>');");
            outputFile.println("document.write('<br>');");

            num++;
       }



       outputFile.println("</SCRIPT>");

       outputFile.println("</BODY>");
       outputFile.println("</HTML>");

       outputFile.close();


        inputFile.close();


 }
 catch (FileNotFoundException e)
 {
        System.err.println("This shouldn't happen: " + e);
```

```
      }
      catch (IOException e)
      {
             System.err.println("Writing error: " + e);
      }
    }

}
```

10.1.6 LCDWrite.class

```
import java.lang.*;
import java.util.*;
import java.io.*;

import com.dalsemi.comm.*;
import com.dalsemi.system.*;

import com.taylec.tini.LCDManipFormat;
import com.taylec.tini.DIPManip;
import MuxSelect;

public class LCDWrite
{
  private static final void pause(int delay_)
  {
    try
    {
      Thread.sleep(delay_);
    }
    catch (InterruptedException e_)
    {
    }
  }

  public static void LCDOut()
  {


     try
     {
          PrintStream os = System.out;
          LCDManipFormat manip = new LCDManipFormat();
          DIPManip dip=new DIPManip();


       while(1==1)
       {

          DipValue=dip.read();

          if((DipValue&DIPManip.S6) == DIPManip.S6)
          {
               manip.out.print("Wind Spd: " + windSpeed + " mph");
               manip.out.flush();
               pause(2000);
               manip.clear();
          }

          DipValue=dip.read();

          if((DipValue&DIPManip.S5) == DIPManip.S5)
          {
               manip.out.print("Wind Dir: " + windDir);
               manip.out.flush();
               pause(2000);
               manip.clear();
```

```
            }

            DipValue=dip.read();

            if((DipValue&DIPManip.S4) == DIPManip.S4)
            {
                  manip.out.print("Temperature: " + temp);
                  manip.out.flush();
                  pause(2000);
                  manip.clear();
            }
          }

      }
      catch (IllegalAddressException err_)
      {
         System.out.println(err_.toString());
      }
      catch (FileNotFoundException e)
      {
         System.err.println("This shouldn't happen: " + e);
      }
      catch (IOException e)
      {
          System.err.println("Writing error: " + e);
      }


  }

  public static int ReadRain()
  {
      int data = 0;

      try
      {
             RandomAccessFile file = new
             RandomAccessFile("./docs/rain/raindata.bin", "r");

             file.seek(0);

             data = file.readInt();

             file.close();

      }
      catch (FileNotFoundException e)
      {
         System.err.println("This shouldn't happen: " + e);
      }
      catch (IOException e)
      {
          System.err.println("Writing error: " + e);
      }

      return data;
  }

  public static String Light(double light)
  {
      String text = "";

      if(light<=15)
      {
          text = "Dark";
      }
      else if(15<light&&light<=60)
      {
```

```
            text = "Dull";
        }
        else if(60<light&&light<=120)
        {
            text = "Bright";
        }
        else
        {
            text = "Very Bright";
        }

        return text;
    }


    public static void main(String[] args)
    {
        LCDOut();
    }
}
```

10.1.7 Rain.class

```
import java.lang.*;
import java.util.*;
import java.io.*;

import com.dalsemi.comm.*;
import com.dalsemi.system.*;

import com.taylec.tini.DIPManip;


public class Rain
{
  private static final void pause(int delay_)
  {
    try
    {
      Thread.sleep(delay_);
    }
    catch (InterruptedException e_)
    {
    }
  }

  public static void ReadRain()
  {
    int val = 0;
    int oldDay = 0;

    //check existing file value
    try
    {
     RandomAccessFile file = new RandomAccessFile("./docs/rain/raindata.bin",
     "rw");

     file.seek(0);
     val = file.readInt();
     oldDay = file.readInt();

     file.close();

    }
    catch(FileNotFoundException e)
    {
        System.err.println("This shouldn't happen: " + e);
```

```
}
catch (IOException e)
{
    System.err.println("Reading error: " + e);
}


int day = getCurrentTime();

if(oldDay!=day)
{
 val = 0;
}


while(1==1)
{
 try
 {
      PrintStream os = System.out;
      DIPManip dip=new DIPManip();

      int DipValue=dip.read();


      if((DipValue&DIPManip.S1) == DIPManip.S1)
      {
           if(getCurrentTime()!=day)
           {
              day = getCurrentTime();
              val = 0;
           }

           val++;

           RandomAccessFile file = new
           RandomAccessFile("./docs/rain/raindata.bin", "rw");

           file.seek(0);

           file.writeInt(val);
           file.writeInt(getCurrentTime());

                file.close();

           pause(600);

      }

      while((DipValue&DIPManip.S1) == 0)
      {
           DipValue=dip.read();

           pause(100);
      }

 }
 catch (IllegalAddressException err_)
 {
           System.out.println(err_.toString());
 }
 catch (FileNotFoundException e)
 {
       System.err.println("This shouldn't happen: " + e);
 }
 catch (IOException e)
 {
       System.err.println("Writing error: " + e);
```

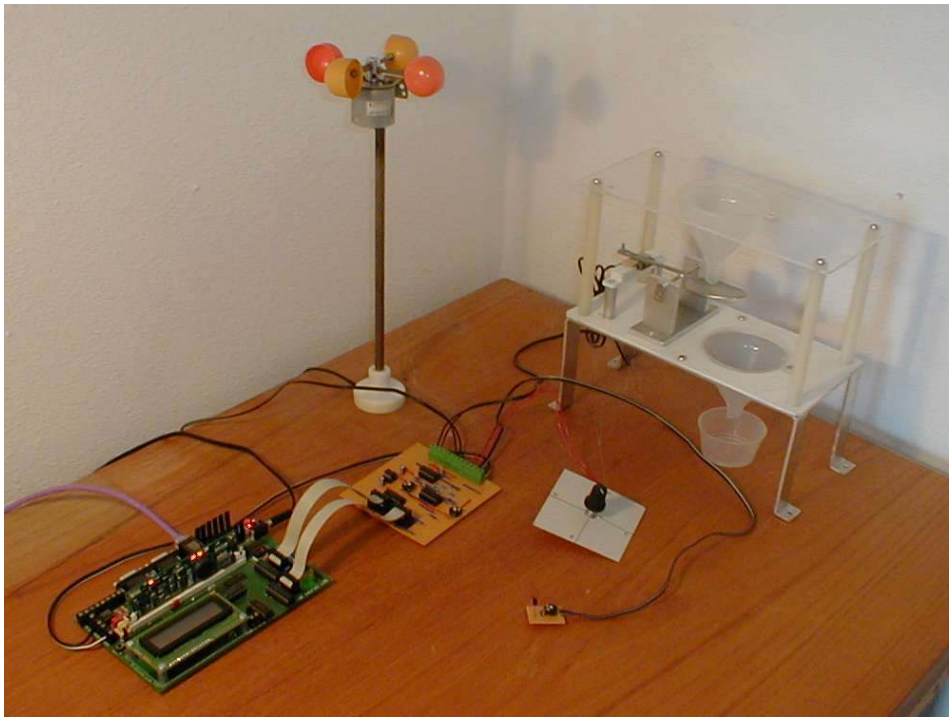
```
      }
     }

  }

  public static int getCurrentTime()
  {
     Clock c = new Clock();
     c.getRTC();
     return c.getDay() ;
  }

  public static void main(String[] args)
  {
    ReadRain();
  }
}
```

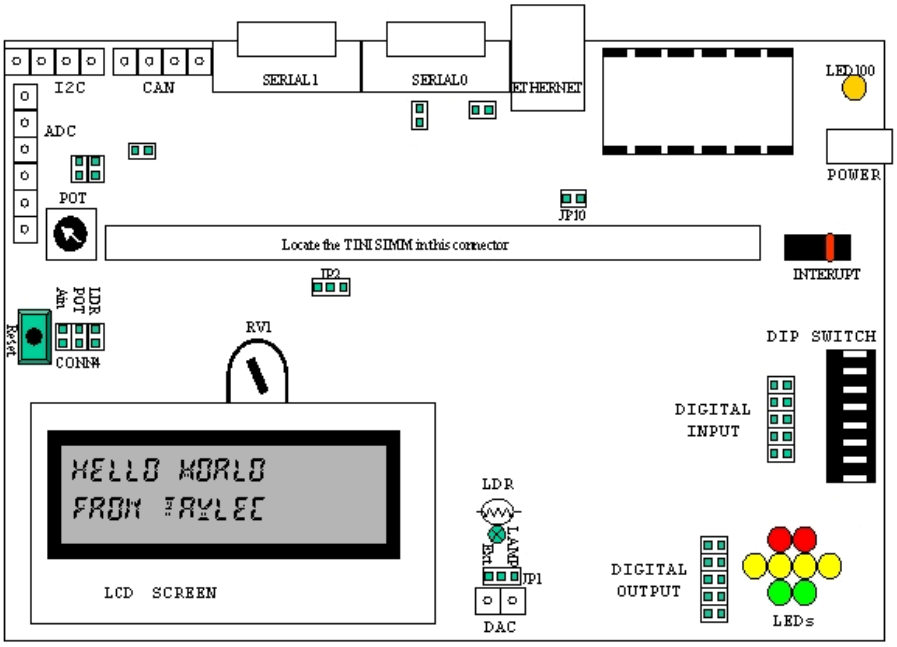
10.1.8 MuxSelect.class

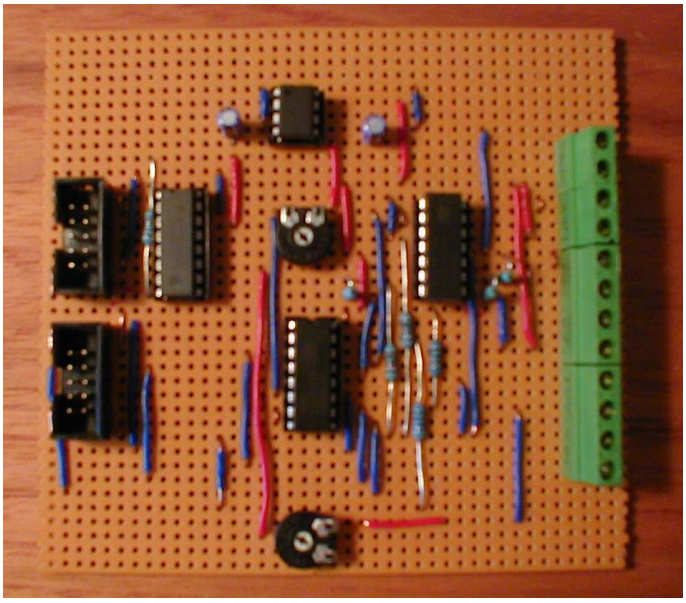
```
import com.dalsemi.system.*;

//Selects the Mux Channel

public class MuxSelect extends com.taylec.tini.DigitalOutputManip
{

  //Channel 1
  public static final int CH1=0x00;

  //Channel 2
  public static final int CH2=0x01;

  //Channel 3
  public static final int CH3=0x02;

  //Channel 4
  public static final int CH4=0x03;

  //Channel 5
  public static final int CH5=0x04;

  //Channel 6
  public static final int CH6=0x05;

  //Channel 7
  public static final int CH7=0x06;

  //Channel 8
  public static final int CH8=0x07;


  //Construct with default Dataport address and turn all LEDs off
  //@see com.dalsemi.system.DataPort

  public MuxSelect() throws IllegalAddressException
  {
    super(0x380000);
    clear();
  }


  //Construct with a given Dataport address and turn all LEDs off
  //@param address_ Dataport address
  //@see com.dalsemi.system.DataPort
```

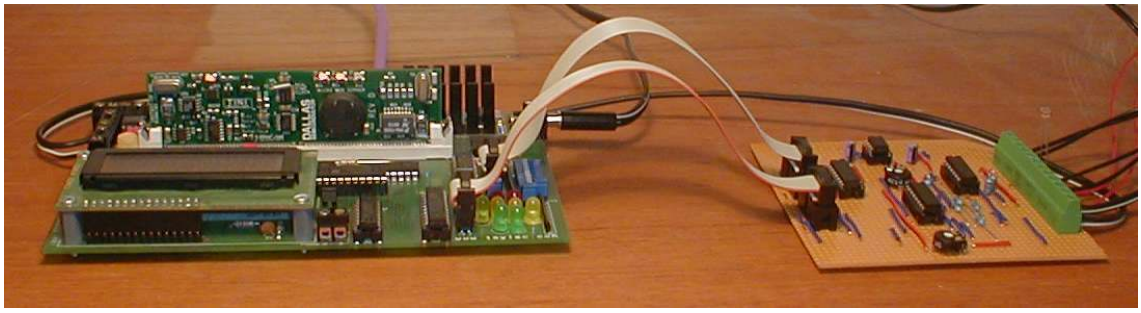
```
  public MuxSelect(int address_) throws IllegalAddressException
  {
    super(address_);
    clear();
  }

}
```

10.2 Pictures









10.3 Emails

Email to TINI interest group – 30/01/2002

Hi,

If you want to generate web pages with dynamic content then have you looked at Smartsc's TiniHttpServer :-

http://www.smartsc.com/tini/TiniHttpServer/

Bob

In message <001d01c1a9bc$f817a710$fb01a8c0@dave>, David Pilch <bristol@d avidpilch.freeserve.co.uk> wrote

I'm quite new to TINI so please bear with me. I have a www.taylec.com Tutor board and 1Mb TINI Board. I have installed TINI OS and Slush and programs run quite happily using either JavaKit or Telnet. However, I want to be able to connect to TINI and run *.tini programs with a single command line, so it can be incorporated into a web page. Is there any way of achieving this? I keep on having problems with the login prompt popping up. Telnet does support auto login using the URL telnet://root:tini@hostname but TINI does not appear to support this because the login prompt still appears.

Is there any way of removing the login request altogether or getting round the problem another way?

Any suggestions would be appreciated.

Regards,

David Pilch.

Bob Hinton

11 References

[1] http://www.ibutton.com/TINI/index.html - This is the main website for TINI

[2] http://www.dalsemi.com – Manufactures of TINI

[3] http://www.systronix.com – Interface board manufacturer

[4] http://www.ibutton.com – Home of iButton

[5] http://java.sun.com/j2se/1.3/ - Java SDK download

[6] http://java.sun.com/products/javacomm/index.html - Communications API

[7] http://www.ibutton.com/TINI/software/soft_order.html - TINI SDK download

[8] The TINI interest group is an e-mail forum where you can network with other TINI developers. This is a major source of useful information about TINI.

[9] www.smartsc.com – TiniHttpServer developers

[10] http://java.sun.com/products/servlet/download.html#specs – Servlet classes

[11] http://jakarta.apache.org/builds/jakarta-ant/release/v1.4/bin/ - Ant source

[12] http://tiniant.sourceforge.net/ - TiniAnt source