Project SB3: Data Logger
Package Environment Monitoring
Final Report

Andrew Holt
ah635
Team 6
Emmanuel

06 June 2013

Abstract

An opportunity was noticed to develop a product for tracking the environmental conditions of a package in transit. A prototype system has been produced, requiring hardware, firmware and software development. The prototype has been found to show good potential, but requires some refining before it could become a commercially viable product. The project has been enjoyable and it has been rewarding to apply some of the knowledge gained in lecture courses to real product development.

Contents

1 Introduction **3**

2 Overview **3**
2.1 Summary of Design . . . 3
2.2 Project Management . . . 4

3 Design **4**
3.1 Hardware . . . 4
3.2 Software . . . 6
3.2.1 Serial Port Communications . . . 6
3.2.2 Communications . . . 7
3.2.3 User Interface . . . 7
3.3 Communications . . . 8

4 Problems Encountered and Technical Solutions **9**
4.1 RS232 Library . . . 9
4.2 Display of Data . . . 9
4.3 Linked mode streaming . . . 9
4.4 Problems with Sensors and Circuits . . . 10

5 Calibration and Test Procedures **10**

6 Conclusions and Next Steps **11**

A Source Code and Circuit Diagrams **12**
A.1 Thermistor Signal Conditioning Circuit . . . 12
A.2 datalogger.h . . . 12
A.3 datalogger.cpp . . . 13
A.4 comms.h . . . 19
A.5 comms.cpp . . . 20
A.6 rs232.h . . . 25
A.7 rs232.cpp . . . 26

B First Interim Report **34**

C Marketing Datasheet **34**

List of Figures

1 Overall system block diagram . . . 4
2 Completed data logger board. . . . 5
3 Communications stack showing interaction of each layer. . . . 7
4 Screenshot of user interface running under Ubuntu Linux. . . . 8
5 Data packet layout, showing positions of bytes and their contents. . . . 8
6 Temperature calibration using ice/water mixture. . . . 11
7 Temperature monitoring circuit. . . . 12

List of Tables

1 Commands sent from PC to data logger. . . . 9

1 Introduction

The express delivery industry is one of the fastest growing industries in the world today. In 2003, the industry made a direct contribution of US$64 billion to worldwide GDP, and has been growing by around 8% per year since [2]. However, customers have very little information of the environmental and handling conditions their package has been subject to during its transit. When shipping fragile or sensitive products, the conditions that the parcel is subject to may be critical in the integrity of the product. The two major issues here are:

1. Fragile goods which arrive broken: it is desirable to determine whether the package has been mishandled, and if so by whom.
2. Perishable goods: if they have been subjected to conditions outside of sensitive temperature and humidity limits they will have a shorted shelf life. This may not be obvious upon inspection of the goods, so detailed condition tracking is required.

While products have been around for a while to carry out this type of tracking on shipping containers, the same problem applies to smaller and domestic packages too. In the past six months, two large companies have announced products to achieve this, but it is clearly an emerging market and a good product launched now could have a great chance of success. One of these recent products, “SenseAware”[4] by FedEx Corporation, is already released and available however this is only available on a select number of carriers and services and is a business level product, not suited for everyday or relatively low value package monitoring. An alternative is the “DropTag” system recently announced by Cambridge Consultants [1], however this system is not yet at the commercial release stage.

With a problem identified and a solution required, as well as a clear business opportunity due to the interest in development from other companies, product development was begun.

2 Overview

In order to effectively track the condition of a parcel during transit, the final product was clearly required to be cheap, low power and physically small and robust. For this project, a simple prototype was to be developed, which if successful could be miniaturised and fabricated for the final product.

2.1 Summary of Design

The first stage of the design process was to evaluate which quantities should be measured and logged. The key environmental variables would be temperature and humidity. Vibration, orientation and shocks would also be required to understand the handling and transport conditions. GPS tracking would also be useful, but it was decided to focus on the other five initially as GPS could be relatively easily included with a dedicated GPS chip communicating over the I^{2}C protocol, and GPS reception may be limited at many points during transit.

The product would clearly be required to work without being plugged in to a computer, so a standalone mode would be developed. It would be useful for development, debugging and testing to operate with instant feedback from the computer system, so a linked mode was also developed.

To ensure long term operation, on-board memory would be required, so an EEPROM was used for data storage in the standalone mode. This would also require a battery supply.

Some kind of on-board display would also be useful on the board, for monitoring, debugging and status display, so an LCD screen was used along with five LEDs.

The platform choices were specified in the project requirements, so the project was to be developed for the STM32F100 ARM Cortex microcontroller and a PC running the Windows operating system. The key design areas would therefore be hardware, firmware and software. The budget was generous for the development board so cost was not really an issue, however to achieve a commercial product, the cost and size would need to be reduced. This could be achieved by miniaturising, as surface mount components tend to be cheaper; use of a cheaper microcontroller (which wouldn’t need the development features of the STM); and economies of scale in a full production product.

The block diagram for the system is shown in figure 1.

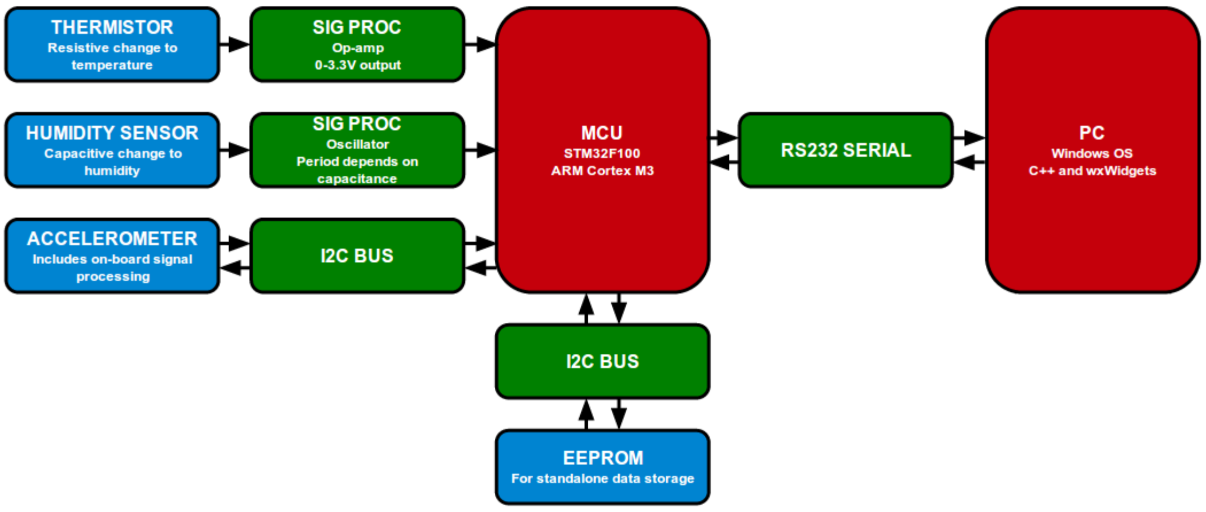

Figure 1: Overall system block diagram

2.2 Project Management

The project was divided between the two team members. It was decided that James would write the firmware for the microcontroller, Andrew would write the software for the computer and the hardware design was to be shared. The general development plan was to start by making a basic but working system and gradually adding features and complexity upon the existing system. This ensured that we would have a fully operational product, if not with every desired feature, by the end of the project, and would also allow the simpler parts to be used in testing more complex parts.

Initially basic hardware circuits were designed and built. The firmware and software were then developed. The hardware circuits were highly useful in testing the firmware, but in writing the firmware, flaws in the circuit designs became apparent, such as the initial plan to use a simple charge-discharge of the humidity sensor. For this reason, the humidity sensor circuit was updated during the project.

3 Design

A brief overview of the design has been given in section 2.1. This section gives technical detail about the parts of the system which I designed. The other parts are detailed in James' report.

A photo of the completed hardware is shown in figure 2.

3.1 Hardware

In order to measure temperature, it was decided to use a thermistor. Thermistors have some advantages over other temperature measurement devices such as thermocouples, resonant crystal thermometers and diode junction voltage measurement such as [3]:

1. The dependant quantity is very easy to measure.
2. Large change in measured quantity (thousands of ohms, compared to millivolt changes in thermocouples and diode junctions), meaning low demands are placed on the analogue circuitry.
3. Inexpensive.
4. Good stability.
5. Circuit can easily be designed to give very linear output, making the digital conversion to temperature straightforward.

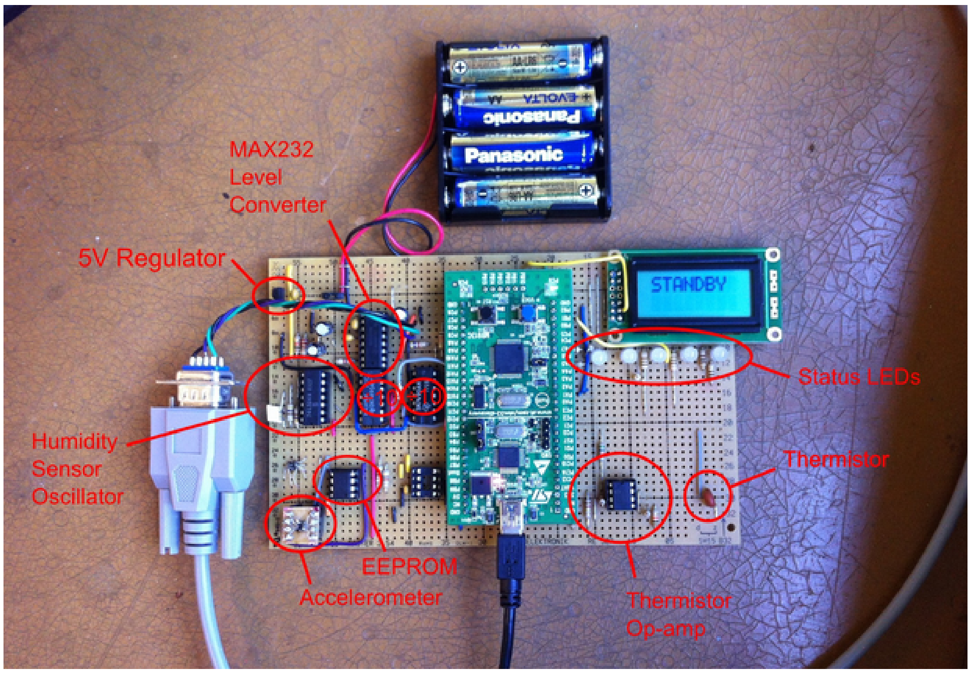

Figure 2: Completed data logger board.

With these considerations, an ND06P00103K NTC 10 kΩ thermistor from AVX was selected, based on price and good electrical characteristics for the application requirements.

An analogue signal processing circuit was required to process the output from the thermistor to a signal usable by the microcontroller analogue-to-digital converter (ADC). This required a signal in the range of $0 - 3.3\,\mathrm{V}$.

The thermistor was placed in a voltage divider circuit with a series 10 kΩ resistor. This was selected to linearize the temperature curve. This combination gives a response linear to within 1 % from 0 to 40 °C and 3 % between −10 and 50 °C[3]. This voltage divider operates from a 3.3V rail, ensuring that the input to the next stage will always lie within its supply rails. At this supply level, the maximum power dissipation through the thermistor was calculated as 3.3 mW at 50 °C. This is well below the maximum dissipation for the thermistor of 0.71 mW, so will not have a noticeable affect on the temperature measurement.

The output from the voltage divider drives an op-amp non-inverting amplifier circuit. The voltage divider has relatively high output resistance, so use of a non-inverting op-amp configuration ensures a very large input resistance ($\sim 10^{13}\,\Omega$), avoiding loading on the measurement circuit.

The non-inverting amplifier is slightly non-standard due to a non-zero reference voltage on the inverting terminal. This is selected to give the same output as the thermistor setup at the low end of its response, which was selected as −20 °C as this will lie well below the temperature a package would expect to experience in transit and is at the limit of the linear response. This reference voltage was found to be 0.79 V, so is generated using a series combination of 22 kΩ and two 10 kΩ resistors, giving a very close voltage match with standard resistor values.

The gain was then calculated to produce a 3.3 V output swing over the input range to be received, and the gain required was found to be 1.47. This was achieved in the non-inverting amplifier using standard resistor values of 56 and 120 kΩ.

The output swing was chosen to span the full supply level of $0 - 3.3\,\mathrm{V}$ as the op-amp used provides very good rail-to-rail operation, and the extremes of the working temperature range are both unlikely to experienced and reaching the limits of the linear response from the thermistor. For the package tracking application, it is not critical to know how far beyond these limits the temperature has reached, so a simple saturating output is sufficient. This allows the ADC to operate with maximum precision.

The circuit diagram for the temperature monitoring circuit is shown in appendix A, figure 7.

A filter could easily have been implemented in the op-amp to avoid noise, since the signal is of very low frequency. However, the output was found to be very stable without filtering, so filtering was avoided for the sake of simplicity.

3.2 Software

It was suggested to use the National Instruments "LabWindows/CVI" package for developing the PC software and graphical user interface (gui), however it was decided to eschew this in favour of developing in C++ and the wxWidgets cross platform gui toolkit. The reasons for this were:

1. LabWindows is a highly specialist and proprietary development package. While it is simple to use and fast to setup, we found it unlikely that we would use it again in the future.
2. LabWindows is fairly limited in scope of what can be achieved, whereas wxWidgets and C++ allow pretty much anything to be achieved.
3. Our other project was using C++ and wxWidgets, so it was a simpler option to only learn one package, rather than one per project.
4. Past experience of coding in C++ could be utilised, whereas LabWindows relies on C which I have very little experience of.

There are three main components to the software development. The back end functionality required serial port communications and communications protocols with the microcontroller. The front end required a simple user interface to display the logged data and control the microcontroller functions. See appendix A for a full code listing.

The actual data transmission is handled by the serial port library.

The sending of packets to and from the serial port for the microcontroller are handled by the communications level, which provides an interface between the application and the actual data, so the user and user interface level do not depend on the communications system. This is advantageous as it means the top level application is independent of the communication medium, and could be converted to use USB or wifi without modifying the user application. It also simplifies the functions required in the user application as they do not directly handle the raw data, but instead use more user/programmer friendly data structures.

On top of the communications layer is the application and user interface layer. These allow easy and logical viewing and understanding of commands to be sent and data received by the computer.

This hierarchical structure is shown in figure 3. The layers map very roughly onto the Open Systems Interconnect (OSI) reference model. The user interface performs the duties of layer 7 (application). The communications layer is similar to layer 6 (presentation), while the rs232 library spans levels 4 and 5 (transport and session layers, respectively). The lower layers are defined by the RS232 protocol.

3.2.1 Serial Port Communications

The files `rs232.cpp` and `rs232.h` contain functions for sending and receiving data from the serial port. These were found online [5], released under the GNU General Public Licence. (See section 4 for more details.)

The important functions provided by this software is are:

`RS_232_OpenComport()` This sets up the connection with the serial port to be used, setting up a two way communications link to the microcontroller.

`RS_232_CloseComport()` Closes the connection to the serial port at the end of the operation.

`RS_232_PollComport()` Reads data that has been sent by the microcontroller to the computer.

`RS_232_SendBuf()` Writes to the serial port to send commands to the microcontroller.

These functions were all provided directly by the software found online, which greatly simplified the serial port reading.

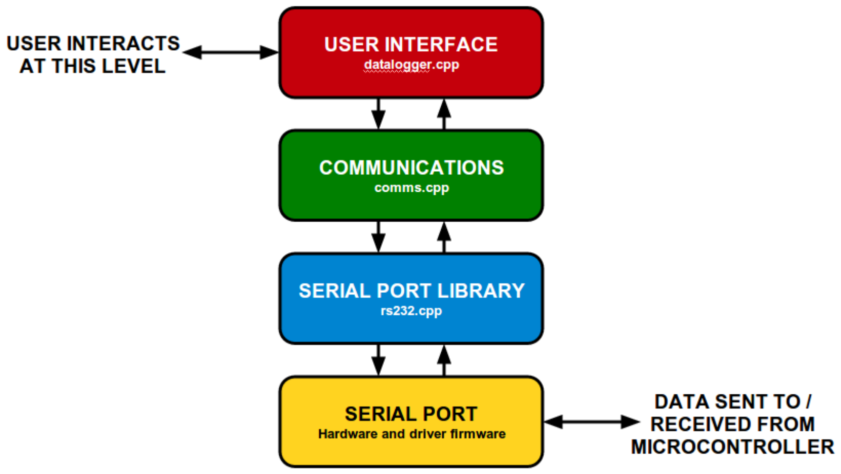
USER INTERACTS AT THIS LEVEL
USER INTERFACE
datalogger.cpp
COMMUNICATIONS
comms.cpp
SERIAL PORT LIBRARY
rs232.cpp
SERIAL PORT
Hardware and driver firmware
DATA SENT TO / RECEIVED FROM MICROCONTROLLER

Figure 3: Communications stack showing interaction of each layer.

3.2.2 Communications

The functions provided in the RS_232 library are very low level, and perform sending and receiving of individual bytes. For the user interface, it is useful to have a high level data structure, rather than dealing with individual bytes of data. The files `comms.cpp` and `comms.h` provide the means to do this. The functions here call the functions in `rs232.cpp`. It converts the user commands into the required bytes to send to the microcontroller, and organises the returned data into a `vector` data structure.

This level provides functions to be called by the data logger application:

`RS232_Init()` Opens the required comport to establish a link with the microcontroller.

`RS232_Close()` Closes the opened comport.

`send_command()` Sends a command to the microcontroller by writing to the serial port buffer (uses the `SendBuf` function).

`read_eeprom_data()` Reads data stored in the EEPROM into a `vector`.

`get_Readings()` Receives the data sent by the microcontroller in linked mode and stores it in a vector.

3.2.3 User Interface

The top level of the PC system is the user interface, which allows the user to interact with the system in an intuitive and non-technical way. The user interface[1] is shown in figure 4.[2]

The user can click buttons and select values from a drop down menu to operate the data logger. The output from the monitoring is displayed directly on the screen during operation in the linked mode. After uploading the data from standalone mode operation, or once a logging session has been completed in linked mode, the data may be exported to a CSV file.

Additionally, the user may set the sample rate of the data logger from the gui. This is stored on board the data logger, so may be used to set the standalone operation rate as well as linked mode.

Development was started on a feature to plot the data that had been logged (accessed with the "View All" button). This utilised a python plotting script, called by the user interface. This system worked on Linux based systems, but was not modified to run under Windows. This would be a useful feature, however it was not implemented to allow development of more critical features. As an interim replacement, the user may open a saved CSV file in their favourite application and plot it (very easy in Microsoft Excel, Matlab and other data analysis packages).

[1]Since this screenshot was taken, additional features have been added to allow setting of a threshold temperature and humidity. If these limits are crossed, a warning is displayed.

[2]It may be noted that James' report shows a user interface which looks quite different. This is due to running it on Windows 8, whereas I used Ubuntu. This shows the cross-platform capabilities of the software.

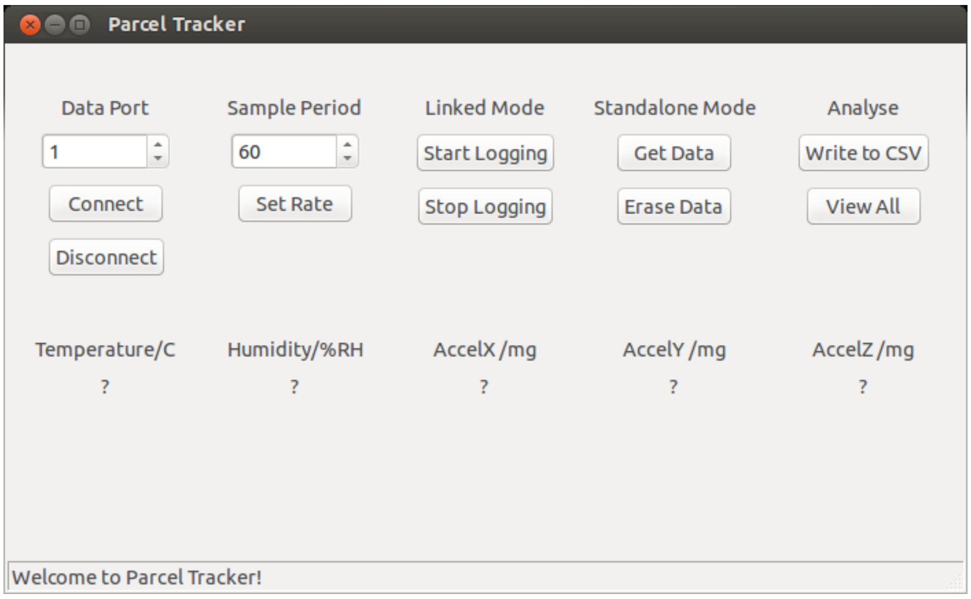

Figure 4: Screenshot of user interface running under Ubuntu Linux.

3.3 Communications

It was required to define a communications protocol. This would ensure both the microcontroller and the PC knew what each byte and bit of a transmitted data packet meant. It was decided to avoid start-bits, stop-bits and check sums in the initial implementation, though room was left for these to be added later if necessary. In general, these weren't required as the PC could operate at sufficiently higher speed compared to the data logger that it faced no problems in reading the data.

In linked mode operation, with the microcontroller plugged in to the computer, the microcontroller was to send one packet of data for each sampling period. Each packet was to contain 10 bytes, with the first 2 bytes containing temperature data, in upper and lower bytes, then a byte of humidity data. The remaining 7 bytes were reserved for accelerometer data, as we were initially unsure exactly what information we could send from the accelerometer. These bytes could also be used for check sums to ensure the data was received correctly. Figure 5 shows the layout of a data packet.

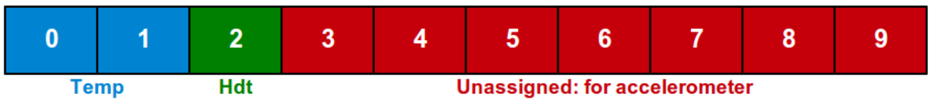

Figure 5: Data packet layout, showing positions of bytes and their contents.

When in standalone mode, the data logger would store all recorded data in the EEPROM. The first 16 bytes of EEPROM data were to be reserved for configuration information: the first two would store the length of the recorded data from the last log, while the third byte would store the current sample rate. The rest of the EEPROM (32,000 bytes) would be divided into 10 byte segments, each to contain one data packet, as specified for the standalone mode.

For sending commands from the PC to the microcontroller, it was decided that all commands sent should be 10 bytes long. Each would contain an ASCII character defining the desired command, as shown in table 1.

The communications protocols defined worked well for the application and did not need changed after definition. The accelerometer data was loaded with the maximum acceleration in each dimension during the sample period.

Character	Command
L	Begin logging in linked mode.
S	Stop logging (linked mode).
U	Upload data from EEPROM (standalone mode data).
E	Erase EEPROM contents (except 16 bytes of configuration data).
R [num]	Set sample period, in seconds (defined in num).

Table 1: Commands sent from PC to data logger.

4 Problems Encountered and Technical Solutions

In this section, some of the problems encountered during the problem will be discussed, along with the solutions we used to overcome these problems. I will focus on the problems in the areas I was focusing on but also give a mention to other problems in James's parts, see James's report for a fuller discussion of these.

4.1 RS232 Library

The decision to use C++ and wxWidgets instead of the LabWindows environment initially cased a problem of being unable to communicate with the RS232 port, since this was a feature built in to the LabWindows software.

Given the short time allowed for the project, it was not a viable option to write an RS232 library from scratch (interesting and challenging though this would doubtless have been!). A free and open-source library was found online[5] to provide this functionality and was found to work well.

Once this was found, it was relatively straightforward to write the communications layer functions. These were initially based on the similar functions in the supplied code, but became more heavily modified as features were added specific to the data logger, such as reading a packet, reading from EEPROM and writing according to the protocol we defined.

4.2 Display of Data

Displaying the data in the gui in a sensible and easily interpretable way turned out to be a significant problem. The original idea had been to create graphs on the gui using OpenGL as a graphics package. However, once the basic user interface had been created and the communication system implemented, there was insufficient time left to do this in time for the demonstration. There was already in place a function for outputting the stored data to a comma separated value (csv) file[3], so it was decided to utilise this in an outside program to do the plotting.

Some Python scripts were written to achieve this, which worked well under Linux. There were issues in running the same scripts on Windows, and since time was short, it was decided to focus efforts in other areas, such as the temperature and humidity boundaries on the gui display. The current recommended way to view the plotted data is to open the csv file in Excel and plot it from there.

4.3 Linked mode streaming

WxWidgets provides an event based graphical user interface. This operates by running an "event loop", and if something in this loop triggers, then the program runs the relevant "callback function", before returning to the loop. This procedure is ideal for the standard usage case where the user does one thing and waits for it to happen or complete the required function before requesting the next thing.

However, the linked mode data logging requires two things to happen at once. The user will first click the "Start Logging" button, which will start the program in a loop of reading data from the serial port, storing it, and updating the display. However, at the same time as this is going on, the program

[3]A common and widely used data storage format.

also needs to be checking the user interface to see if anything else is being requested, particularly the "Stop Logging" command. This causes a major difficulty as it requires running two different loops at the same time.

One approach may be to run a couple of iterations of one loop, and then a few of the other. However, this method is not adequate as if the program happens to be in the data collection loop when the button is pressed, the button press will simply be missed.

After some searching, it was found that wxWidgets provides a function called `onIdle` which runs a function only when the program has some idle time. This means that the program may run the data collection loop only when nothing else is happening, but as soon as another event is triggered, for example the stop button being pressed, the loop is interrupted and the program can deal with the new request.

The start and stop buttons can therefore simply set a flag (a `bool` type called `read_loop_on`), and then any idle time for which the loop is high will call the function to read the serial port.

This method was inspired by and borrows heavily from an example on the wxWidgets wiki[6]. There were initial concerns about whether the idle time would be enough to read the data successfully without dropping bytes, but it was found to be sufficient.

4.4 Problems with Sensors and Circuits

The changes to the hardware circuitry were mainly carried out by James, so I shall give only a brief description.

It was discovered that the initial idea for measuring the capacitance of the humidity sensor was not suitable as the charging and discharging was too fast to be accurately monitored by the microcontroller directly. Therefore the variable capacitance of the humidity sensor was used in a simple oscillator and after dividing the frequency by 100, this was slow enough for the microcontroller to detect the frequency with relative ease. This method proved to work well, though there were some issued with the final readings (see section 5).

Additionally, it was found that the accelerometer we used was not ideally suited to this application. The accelerometer contained an on board microcontroller to process the raw data, however this only gave selected output types, as desired in, say, a smartphone or hand-held gaming controller. It gave access to neither the raw data from which we could obtain more useful results, nor enough processing to give the sort of output we required. It was suitable for detecting a shake or, to an extent, an impact, but could not give detailed enough data to detect the acceleration profile of, for example, being dropped for a short time and landing hard, or falling off a shelf in a van and other critical scenarios.

5 Calibration and Test Procedures

In order to achieve accuracy in the data readings, it was necessary to calibrate the sensors.

Calibration of the thermistor was relatively straightforward, using an ice/water mixture at 0 °C and water heated to 50 °C, measured using an infared thermometer. Figure 7 shows the calibration process.

The humidity calibration was more complex. Two reference points were used, a freezer (which has 0 %RH) and the air above a saturated salt solution (75.3 %RH at 25 °C).

It was found in the final testing that the humidity readings were not accurate. Several possible contributing factors may be identified:

- The oscillator and frequency division circuitry may not have been working properly at the low temperatures of the freezer.
- The air above the saturated salt solution was not completely sealed from the surroundings, meaning the reading was not completely accurate.
- The reading from the humidity detection circuit was not linear, and may not have matched the supplied lookup table very closely.
- The humidity sensor was heavily handled during testing of the rest of the system, which almost certainly left residue salts and other contaminants to disturb the capacitance.

It is not possible to say which of these is/are the main contributor(s) to the inaccurate readings, but given further testing, these factors could be found and fixed.

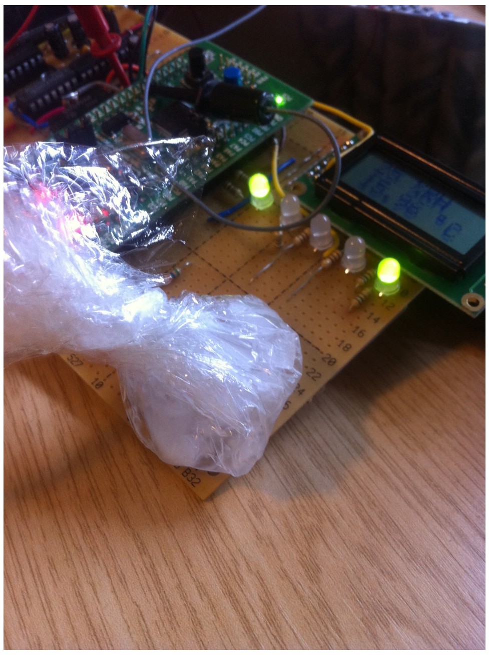
Figure 6: Temperature calibration using ice/water mixture.

6 Conclusions and Next Steps

This project has established that there is an opportunity for a commercial product in the package environment tracking field. A prototype data logger was developed for this application, and does show good potential. The current prototype would need some improvement and some major changes to become a commercial product, and the software would require some additional functionality and improvement, but it seems clear that this is possible and would not an unreasonable amount of extra work.

The suggested next steps for the improvement of the package environment tracking system would be:

1. Develop a system for graphical output of the logged data without the need of an external application.
2. Find the problems facing the humidity readings and fix these faults.
3. Find a more suitable accelerometer (that still matched the price requirements) to allow better analysis of the movement of the parcel.
4. Develop a system for notification of the sender/receiver of the package during shipping, providing real time tracking of the environmental conditions.
5. Add GPS functionality to also track the package location during transit.
6. Miniaturise the data logging system and greatly reduce both the physical size and the cost (likely to scale together) to make it commercially viable.

From a project perspective, it has been enjoyable to apply some of what has been learned in the electronics and software courses. It has also been enjoyable to work on a product and to see a real result at the end of the project.

One thing this project has reinforced is that the modern world relies on digital electronics, with only a small amount of simple analogue interfacing. Because of this change, it seems silly that the Cambridge Engineering course maintains such a focus on analogue electronic design. For students with a future in any field of engineering it would seem that the knowledge and skills to create a simple microcontroller based data logger would be far more useful than developing a purely analogue FM radio. To this same end, an understanding of the I^{2}C and other digital interfacing protocols would be a better education in the world of useful electronics than a detailed understanding of transistor or op amp design.

Nevertheless, this project has been a positive experience and greatly enjoyable. It has given insight into the design of real world electronic and software systems as well as the importance of product development alongside technical development.

A Source Code and Circuit Diagrams

A.1 Thermistor Signal Conditioning Circuit

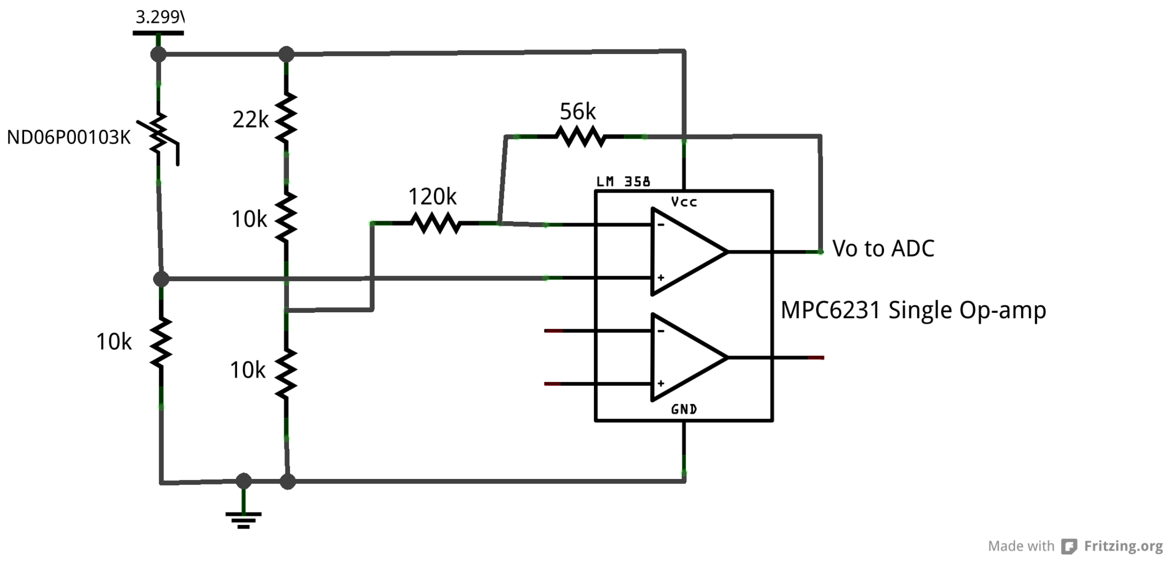

Figure 7: Temperature monitoring circuit.

A.2 datalogger.h

```
/*
File name: gui/datalogger.h
Description: header file for gui framework
Author: Andy Holt
Date: Mon 20 May 2013 16:52
*/

#ifndef datalogger_h
#define datalogger_h

#include <wx/wx.h>
#include <wx/spinctrl.h>
#include <wx/textctrl.h>
#include <iostream>
#include <fstream>

enum
  {
    PORT_SELECT = wxID_HIGHEST + 1,
    SAMPLE_SELECT,
    PORT_CONNECT,
    PORT_DISCONNECT,
    SAMPLE_SEND,
    LOG_START,
    LOG_STOP,
    DATA_GET,
    DATA_ERASE,
    SAVE_CSV,
    FIND_EVENTS,
    GRAPH_DATA,
    TEMP_LABEL,
    HUMID_LABEL,
    ACCX_LABEL,
    ACCY_LABEL,
    ACCZ_LABEL,
    TEMP,
    HUMID,
    ACCX,
    ACCY,
    ACCZ,
  };

//class definition from wxApp;

class MyApp: public wxApp
{
public:
  bool read_loop_on;
  virtual bool OnInit();
  void onIdle(wxIdleEvent& evt);
};

// class definition from wxFrame

class MyFrame: public wxFrame
{
public:
  MyFrame (const wxString& title, const wxPoint& pos, const wxSize& size);
  wxSpinCtrl *spin_port;
  wxSpinCtrl *spin_sample;
  wxString currentdocpath;

  //private:
  void OnExit(wxCommandEvent& event);
  void OnAbout(wxCommandEvent& event);
  void OnPortSelect(wxSpinEvent& event);
  void OnPortConnect(wxCommandEvent& event);
  void OnPortDisconnect(wxCommandEvent& event);
  void OnSampleSelect(wxSpinEvent& event);
  void OnSampleSend(wxCommandEvent& event);
  void OnLogStart(wxCommandEvent& event);
  void OnLogStop(wxCommandEvent& event);
  void OnDataGet(wxCommandEvent& event);
  void OnDataErase(wxCommandEvent& event);
  void OnCSVWrite(wxCommandEvent& event);
  void OnFind_Events(wxCommandEvent& event);
  void OnGraph_Data(wxCommandEvent& event);
  DECLARE_EVENT_TABLE();

};

#endif  // datalogger_h
```

A.3 datalogger.cpp

```
/*
File name: gui/datalogger.cpp
Description: main file for gui framework
Author: Andy Holt
Date: Mon 20 May 2013 15:30
*/

#include "datalogger.h"
#include "comms.h"

extern int com_port_no;
extern bool com_port_open;
extern config_data cfgdata;
extern std::vector <packet> data;

wxStaticText  *temperature_label;
wxStaticText  *temperature;
wxStaticText  *humidity;
wxStaticText  *accelerationx;
wxStaticText  *accelerationy;
wxStaticText  *accelerationz;

extern BYTE rx_buff[RX_BUFF_LEN];
extern BYTE tx_buff[10];

int sample_period = 60;

 BEGIN_EVENT_TABLE(MyFrame, wxFrame)
  EVT_MENU(wxID_EXIT, MyFrame::OnExit)
  EVT_MENU(wxID_ABOUT, MyFrame::OnAbout)
  EVT_MENU(LOG_START, MyFrame::OnLogStart)
  EVT_MENU(LOG_STOP, MyFrame::OnLogStop)
  EVT_MENU(DATA_GET, MyFrame::OnDataGet)
  EVT_MENU(DATA_ERASE, MyFrame::OnDataErase)
  EVT_SPINCTRL(PORT_SELECT, MyFrame::OnPortSelect)
  EVT_SPINCTRL(SAMPLE_SELECT, MyFrame::OnSampleSelect)
  EVT_BUTTON(PORT_CONNECT, MyFrame::OnPortConnect)
  EVT_BUTTON(PORT_DISCONNECT, MyFrame::OnPortDisconnect)
  EVT_BUTTON(SAMPLE_SEND, MyFrame::OnSampleSend)
  EVT_BUTTON(LOG_START, MyFrame::OnLogStart)
  EVT_BUTTON(LOG_STOP, MyFrame::OnLogStop)
  EVT_BUTTON(DATA_GET, MyFrame::OnDataGet)
  EVT_BUTTON(DATA_ERASE, MyFrame::OnDataErase)
  EVT_BUTTON(SAVE_CSV, MyFrame::OnCSVWrite)
//  EVT_BUTTON(FIND_EVENTS, MyFrame::OnFind_Events)
  EVT_BUTTON(GRAPH_DATA, MyFrame::OnGraph_Data)
END_EVENT_TABLE()


IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{
  read_loop_on=false;
  MyFrame *frame = new MyFrame ( wxT("Parcel␣Tracker"), wxPoint(50, 50),
         wxSize(450, 340));
  frame->Show(true);
  return true;
}


// idea from http://wiki.wxwidgets.org/Making_a_render_loop
void MyApp::onIdle(wxIdleEvent& evt)
{
  if(read_loop_on)
    {
      get_Readings();
      evt.RequestMore();
    }
}

MyFrame::MyFrame(const wxString& title, const wxPoint& pos, const
     wxSize& size)
  : wxFrame(NULL, wxID_ANY, title, pos, wxSize(700,400), wxDEFAULT_FRAME_STYLE | wxFULL_REPAINT_ON_RESI
{
  wxMenu *menuFile = new wxMenu;
  menuFile->Append(wxID_EXIT);
  wxMenu *menuData = new wxMenu;
  menuData->Append(LOG_START, wxT("&Start␣Logging\tCtrl-L"),
       wxT("Help␣string:␣Begin␣logging␣with␣board␣plugged␣in."));
  menuData->Append(LOG_STOP, wxT("&Stop␣Logging\tCtrl-K"),
       wxT("Help␣string:␣Stop␣logging␣with␣board␣plugged␣in."));
  menuData->Append(DATA_GET, wxT("&Get␣Data\tCtrl-D"),
       wxT("Help␣string:␣Get␣data␣from␣board."));
  menuData->Append(DATA_ERASE, wxT("&Erase␣Data\tCtrl-E"),
       wxT("Help␣string:␣Erase␣chip␣memory."));
  wxMenu *menuHelp = new wxMenu;
  menuHelp->Append(wxID_ABOUT);

  wxMenuBar *menuBar = new wxMenuBar;
  menuBar->Append(menuFile, wxT("&File"));
  menuBar->Append(menuData, wxT("&Data"));
  menuBar->Append(menuHelp, wxT("&Help"));

  SetMenuBar(menuBar);

  wxBoxSizer *topsizer = new wxBoxSizer(wxVERTICAL);

  wxBoxSizer *ctrlsizer = new wxBoxSizer(wxHORIZONTAL);

  wxBoxSizer *connect_sizer = new wxBoxSizer(wxVERTICAL);



  connect_sizer->Add(new wxStaticText(this, wxID_ANY, wxT("Data␣Port")),
        0, wxALL | wxALIGN_CENTER_HORIZONTAL, 5);
  spin_port = new wxSpinCtrl(this, PORT_SELECT, wxString(wxT("1")));
  connect_sizer->Add(spin_port, 0, wxALL | wxALIGN_CENTER_HORIZONTAL, 5);
  connect_sizer->Add(new wxButton(this, PORT_CONNECT, wxT("Connect")),
        0, wxALL | wxALIGN_CENTER_HORIZONTAL, 5);
  connect_sizer->Add(new wxButton(this, PORT_DISCONNECT, wxT("Disconnect")),
        0, wxALL | wxALIGN_CENTER_HORIZONTAL, 5);
  ctrlsizer->Add(connect_sizer, 1, wxEXPAND | wxALL, 5);

  wxBoxSizer *samplerate_sizer = new wxBoxSizer(wxVERTICAL);
  samplerate_sizer->Add(new wxStaticText(this, wxID_ANY, wxT("Sample␣Period")),
      0, wxALL | wxALIGN_CENTER_HORIZONTAL, 5);
  spin_sample = new wxSpinCtrl(this, SAMPLE_SELECT, wxString(wxT("60")));
  samplerate_sizer->Add(spin_sample, 0, wxALL | wxALIGN_CENTER_HORIZONTAL, 5);
  samplerate_sizer->Add(new wxButton(this, SAMPLE_SEND, wxT("Set␣Rate")),
      0, wxALL | wxALIGN_CENTER_HORIZONTAL, 5);
  ctrlsizer->Add(samplerate_sizer, 1, wxEXPAND | wxALL, 5);

  wxBoxSizer *datalink_sizer = new wxBoxSizer(wxVERTICAL);
  datalink_sizer->Add(new wxStaticText(this, wxID_ANY, wxT("Linked␣Mode")),
          0, wxALL | wxALIGN_CENTER_HORIZONTAL, 5);
  datalink_sizer->Add(new wxButton(this, LOG_START, wxT("Start␣Logging")),
          0, wxALL | wxALIGN_CENTER_HORIZONTAL, 5);
  datalink_sizer->Add(new wxButton(this, LOG_STOP, wxT("Stop␣Logging")),
          0, wxALL | wxALIGN_CENTER_HORIZONTAL, 5);
  ctrlsizer->Add(datalink_sizer, 1, wxEXPAND | wxALL, 5);

  wxBoxSizer *upload_sizer = new wxBoxSizer(wxVERTICAL);
  upload_sizer->Add(new wxStaticText(this, wxID_ANY, wxT("Standalone␣Mode")),
        0, wxALL | wxALIGN_CENTER_HORIZONTAL, 5);
  upload_sizer->Add(new wxButton(this, DATA_GET, wxT("Get␣Data")),
          0, wxALL | wxALIGN_CENTER_HORIZONTAL, 5);
  upload_sizer->Add(new wxButton(this, DATA_ERASE, wxT("Erase␣Data")),
          0, wxALL | wxALIGN_CENTER_HORIZONTAL, 5);
  ctrlsizer->Add(upload_sizer, 1, wxEXPAND | wxALL, 5);

  wxBoxSizer *analysis_sizer = new wxBoxSizer(wxVERTICAL);
  analysis_sizer->Add(new wxStaticText(this, wxID_ANY, wxT("Analyse")),
          0, wxALL | wxALIGN_CENTER_HORIZONTAL, 5);
  analysis_sizer->Add(new wxButton(this, SAVE_CSV, wxT("Write␣to␣CSV")),
          0, wxALL | wxALIGN_CENTER_HORIZONTAL, 5);
//  analysis_sizer->Add(new wxButton(this, FIND_EVENTS, wxT("Find Events")),
//          0, wxALL | wxALIGN_CENTER_HORIZONTAL, 5);
  analysis_sizer->Add(new wxButton(this, GRAPH_DATA, wxT("View␣All")),
          0, wxALL | wxALIGN_CENTER_HORIZONTAL, 5);
  ctrlsizer->Add(analysis_sizer, 1, wxEXPAND | wxALL, 5);
  topsizer->Add(ctrlsizer, 1, wxALL | wxEXPAND, 5);


  wxBoxSizer *outputsizer = new wxBoxSizer(wxHORIZONTAL);

  wxBoxSizer *tempsizer = new wxBoxSizer(wxVERTICAL);
  tempsizer->Add(new wxStaticText(this,TEMP_LABEL,wxT("Temperature/C")),
     0, wxALL | wxALIGN_CENTER_HORIZONTAL, 5);
  temperature = new wxStaticText(this,TEMP,wxT("?"));
  tempsizer->Add(temperature, 0, wxALL | wxALIGN_CENTER_HORIZONTAL, 5);
  outputsizer->Add(tempsizer, 1, wxALL | wxEXPAND, 5);

  wxBoxSizer *humsizer = new wxBoxSizer(wxVERTICAL);
  humsizer->Add(new wxStaticText(this,HUMID_LABEL,wxT("Humidity/%RH")),
      0,wxALL|wxALIGN_CENTER_HORIZONTAL, 5);
  humidity = new wxStaticText(this,HUMID,wxT("?"));
  humsizer->Add(humidity, 0, wxALL | wxALIGN_CENTER_HORIZONTAL, 5);
  outputsizer->Add(humsizer, 1, wxALL | wxEXPAND, 5);

  wxBoxSizer *xaccelsizer = new wxBoxSizer(wxVERTICAL);
  xaccelsizer->Add(new wxStaticText(this,ACCX_LABEL,wxT("AccelX␣/mg")),
          0,wxALL|wxALIGN_CENTER_HORIZONTAL, 5);
  accelerationx = new wxStaticText(this,ACCX,wxT("?"));
  xaccelsizer->Add(accelerationx, 0, wxALL | wxALIGN_CENTER_HORIZONTAL, 5);
  outputsizer->Add(xaccelsizer, 1, wxALL | wxEXPAND, 5);

  wxBoxSizer *yaccelsizer = new wxBoxSizer(wxVERTICAL);
  yaccelsizer->Add(new wxStaticText(this,ACCY_LABEL,wxT("AccelY␣/mg")),
        0,wxALL|wxALIGN_CENTER_HORIZONTAL, 5);
  accelerationy = new wxStaticText(this,ACCY,wxT("?"));
  yaccelsizer->Add(accelerationy, 0, wxALL | wxALIGN_CENTER_HORIZONTAL, 5);
  outputsizer->Add(yaccelsizer, 1, wxALL | wxEXPAND, 5);

  wxBoxSizer *zaccelsizer = new wxBoxSizer(wxVERTICAL);
  zaccelsizer->Add(new wxStaticText(this,ACCZ_LABEL,wxT("AccelZ␣/mg")),
       0,wxALL|wxALIGN_CENTER_HORIZONTAL, 5);
  accelerationz = new wxStaticText(this,ACCZ,wxT("?"));
  zaccelsizer->Add(accelerationz, 0, wxALL | wxALIGN_CENTER_HORIZONTAL, 5);
  outputsizer->Add(zaccelsizer, 1, wxALL | wxEXPAND, 5);

  topsizer->Add(outputsizer, 1, wxALL | wxEXPAND, 5);

  SetSizeHints(400,400);
  SetSizer(topsizer);

  CreateStatusBar();
  SetStatusText(wxT("Welcome␣to␣Parcel␣Tracker!"));
}

/********************** Callback Functions *****************************/

void MyFrame::OnExit(wxCommandEvent& event)
{
  Close(true);
}

void MyFrame::OnAbout(wxCommandEvent& event)
{
  wxMessageBox(wxT("This␣is␣the␣interface␣for␣Parcel␣Tracker.\n\
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣By␣James␣Glanville␣and␣Andy␣Holt.\n\
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Created␣using␣wxWidgets."),
         wxT("About␣Parcel␣Tracker"), wxOK | wxICON_INFORMATION);
}

void MyFrame::OnPortSelect(wxSpinEvent& event)
{
  com_port_no = event.GetPosition()-1;
}
```

```
void MyFrame::OnPortConnect(wxCommandEvent& event)
{
  if (com_port_open)
    {
      wxLogMessage(wxT("Already␣connected!"));
      return;
    }

  RS232_Init(com_port_no);

  if (!com_port_open)
    {
      wxLogMessage(wxT("Error:␣com␣port␣was␣unable␣to␣open"));
    }

  return;
}

void MyFrame::OnPortDisconnect(wxCommandEvent& event)
{
  if (!com_port_open)
    {
      wxLogMessage(wxT("No␣com␣port␣to␣close!"));
      return;
    }

  RS232_Close();

  if (!com_port_open)
    {
      wxLogMessage(wxT("Com␣port␣closed."));
    }
  else
    {
      wxLogMessage(wxT("Error,␣com␣port␣has␣not␣closed␣properly"));
    }
  return;
}

void MyFrame::OnSampleSelect(wxSpinEvent& event)
{
  sample_period = event.GetPosition();
}

void MyFrame::OnSampleSend(wxCommandEvent& event)
{
  if (!com_port_open)
    {
      wxLogMessage(wxT("Error:␣no␣com␣port␣open!"));
      return;
    }

  // set first char in array to R and second to value of sample_period
  for (int i = 0; i < 10; i++)
    {
      tx_buff[i] = '\0';
    }
  tx_buff[0] = 'R';
  tx_buff[1] = sample_period;

  send_command(2);
}

void MyFrame::OnLogStart(wxCommandEvent& event)
{
  if (!com_port_open)
    {
      wxLogMessage(wxT("Error:␣no␣com␣port␣open!"));
      return;
    }
  // set first char in array to L for beginning logging
   for (int i = 0; i < 10; i++)
  {
    tx_buff[i] = '\0';
    }
  tx_buff[0] = 'L';
    data.clear();

  // send via serial port

  send_command(10);

  Connect( wxID_ANY, wxEVT_IDLE, wxIdleEventHandler(MyApp::onIdle) );
  wxGetApp().read_loop_on = true;
}

void MyFrame::OnLogStop(wxCommandEvent& event)
{
  // set first char in array to S to stop logging
  for (int i = 0; i < 10; i++)
  {
    tx_buff[i] = '\0';
    }
  tx_buff[0] = 'S';

  // send via serial port

  send_command(10);

  Disconnect( wxEVT_IDLE, wxIdleEventHandler(MyApp::onIdle) );
  wxGetApp().read_loop_on = false;
}

void MyFrame::OnDataGet(wxCommandEvent& event)
{
  if (!com_port_open)
    {
      wxLogMessage(wxT("Error:␣no␣com␣port␣open!"));
      return;
    }

  // set first char in array to U to upload data
  for (int i = 0; i < 10; i++)
    {
      tx_buff[i] = '\0';
    }
  tx_buff[0] = 'U';
    data.clear();

  // send via serial port

  send_command(1);
  read_eeprom_data();


}

void MyFrame::OnDataErase(wxCommandEvent& event)
{
  if (!com_port_open)
    {
      wxLogMessage(wxT("Error:␣no␣com␣port␣open!"));
      return;
    }

  // set first char in array to E to erase eeprom data
  for (int i = 0; i < 10; i++)
  {
    tx_buff[i] = '\0';
    }
  tx_buff[0] = 'E';

  // send via serial port

  send_command(10);
}

void MyFrame::OnCSVWrite(wxCommandEvent& event)
{
  int maccelx,maccely,maccelz;
  wxFileDialog *writetocsv = new wxFileDialog(this,
                 wxT("Choose␣file␣to␣save␣in"),
                 wxEmptyString,
                 wxEmptyString,
                 _("CSV␣files␣(*.csv)|*.csv"),
                 wxFD_SAVE,
                 wxDefaultPosition);

  if (writetocsv->ShowModal() == wxID_OK) // if the user click "Open" instead of "Cancel"
    {
      currentdocpath = writetocsv->GetPath();
    }

  writetocsv->Destroy();

  std::ofstream csvfile;
  csvfile.open(currentdocpath.fn_str());
  csvfile << "Time/s,Temperature/C,Humidity/%RH,Accx/mg,Accy/mg,Accz/mg,Shaken?\n";
  int datalen = cfgdata.datalen_u * 256 + cfgdata.datalen_l;
  for (int k = 0; k < data.size(); k += 1)
    {
    maccelx = data[k].accel_0 & 0x0F;
    if (data[k].accel_0 & 0x10) {maccelx=-maccelx;}
    maccelx = (maccelx*1000)/21.33;
    maccely = data[k].accel_1 & 0x0F;
    if (data[k].accel_1 & 0x10) {maccely=-maccely;}
    maccely = (maccely*1000)/21.33;
    maccelz = data[k].accel_1 & 0x0F;
    if (data[k].accel_2 & 0x10) {maccelz=-maccelz;}
    maccelz = (maccelz*1000)/21.33;
    csvfile<< k*sample_period<<","     <<float(data[k].temp_u*256+data[k].temp_l)/100 << "," << (int)da
    }
  csvfile.close();

}

/*void  MyFrame::OnFind_Events(wxCommandEvent& event)
{
  wxLogMessage(wxT("Finding interest points"));
}*/

void  MyFrame::OnGraph_Data(wxCommandEvent& event)
{
  wxString command = wxT("./plotdata.py␣&");
  //  command.Append(currentdocpath);

  // this is better as it returns control to the gui before the plots
  // are closed.
  // wxExecute(command, wxEXEC_ASYNC);

  // but this one works...
  wxShell(command);
}
```

A.4 comms.h

```
/*
File name: gui/comms.h
Description: header file for communications stuff
```

```
Author: Andy Holt
Date: Mon 20 May 2013 17:15
*/


#ifndef comms_h
#define comms_h
#include <wx/wx.h>
#include <vector>

typedef unsigned char BYTE;

#define RX_BUFF_LEN 64

struct packet {
  BYTE temp_u;
  BYTE temp_l;
  BYTE humid;
  BYTE accel_0;
  BYTE accel_1;
  BYTE accel_2;
  BYTE accel_3;
  BYTE accel_4;
  BYTE accel_5;
  BYTE accel_6;
};

struct config_data {
  BYTE datalen_u;
  BYTE datalen_l;
  BYTE samp_period;
  BYTE config_3;
  BYTE config_4;
  BYTE config_5;
  BYTE config_6;
  BYTE config_7;
  BYTE config_8;
  BYTE config_9;
  BYTE config_10;
  BYTE config_11;
  BYTE config_12;
  BYTE config_13;
  BYTE config_14;
  BYTE config_15;
};

void RS232_Init(int port_no);
void RS232_Close(void);
BYTE* Read_Data_Block(void);
void send_command(int n);

int read_sensor_data(int *value);
void read_eeprom_data(void);
void get_Readings(void);

#endif  // datalogger_h
```

A.5 comms.cpp

```
/*
File name: gui/comms.cpp
Description: source file for communications stuff
Author: Andy Holt
Date: Mon 20 May 2013 17:21
*/

#include "rs232.h"
#include "comms.h"

using namespace std;
```

```
#define eeprom_size 32000

extern wxStaticText *temperature;
extern wxStaticText *humidity;
extern wxStaticText *accelerationx;
extern wxStaticText *accelerationy;
extern wxStaticText *accelerationz;

int com_port_no = 0;
bool com_port_open = false;
BYTE rx_buff[RX_BUFF_LEN];
BYTE tx_buff[10];

std::vector <packet> data;
std::vector <BYTE> read_buff;
config_data cfgdata;


// RS232 functions
void RS232_Init(int port_no)
{
  // Open COM port given by port_no parameters: COM port,
  // device name, baud rate, parity, data bits, stop bits,
  // input queue size, output queue size
  //  OpenComConfig(port_no, "", 115200, 0, 8, 1, 1024, 16);

  int comportopen = 0;
  comportopen = RS232_OpenComport(port_no, 115200);

  if (comportopen == 1)
  {
  wxLogMessage(wxT("Comport␣opening␣has␣failed,␣returned␣1"));
  }
  else
  {
      // remember COM port number and state
      wxLogMessage(wxT("Successful␣comport␣opening"));
      com_port_no = port_no;
      com_port_open = true;
  }
}

// Close the open COM port
void RS232_Close(void)
{
  if (com_port_open)
    {
      RS232_CloseComport(com_port_no);
    }

  com_port_open = false;
}

// Read in a block of data from the COM port's input buffer
/*
BYTE* Read_Data_Block(void)
{
  // wait for enough data (4 bytes)
  while(GetInQLen(4) < RX_BUFF_LEN);

  // perform read of the COM port buffer, placing result in
  // rx_buff
  //  n = ComRd(4, rx_buff, RX_BUFF_LEN - 1);
  int n = RS232_PollComport(com_port_no, rx_buff, RX_BUFF_LEN - 1);

  rx_buff[n] = 0;   // end string with null

  // return pointer to rx_buff
  return rx_buff;
```

```
}
*/

// Read in a block of data from the COM port's input buffer
BYTE* Read_Data_Block(void)
{
  int blocksize = 4;     // no of bytes to be read
  int i = 0;
  int n = 0;
  BYTE tmpbuff[RX_BUFF_LEN];
  while (i < blocksize)
    {
      n = RS232_PollComport(com_port_no, tmpbuff, RX_BUFF_LEN - 1);
      for(int j = i; j < i+n; j++)
  {
    rx_buff[j] = tmpbuff[j-i]; // load up rx_buff, preserving
             // whats already there
  }
      i += n;
    }
  rx_buff[n] = 0;    // end string with null
  return rx_buff;
}

// send n byte command
void send_command(int n)
{
  if(com_port_open)
    {
      // ComWrt(com_port_no, tx_buff, n);
      int pass = RS232_SendBuf(com_port_no, tx_buff, n);
    }
}

/******** User comms functions ********/

int read_sensor_data(int *value)
{
  BYTE buffer[3];
  int timeout = 100000;
  int blocksize = 4;     // no of bytes to be read
  int i = 0;
  int n = 0;
  BYTE tmpbuff[RX_BUFF_LEN];

  if (! com_port_open)
    {
      wxLogMessage(wxT("Error:\n␣No␣open␣COM␣port."));
      return 0;
    }

  // flush RS232 input buffer
  //  FlushInQ(com_port_no);

  // send ADC read command
  tx_buff[0] = 1;
  send_command(1);

  // wait for returned ADC value
  /*  while((GetInQLen(com_port_no) < 3) && (timeout > 0))
    {
      timeout--;
    }
  */
  while (i < blocksize && timeout > 0)
    {
      n = RS232_PollComport(com_port_no, tmpbuff, RX_BUFF_LEN - 1);
      for(int j = i; j < i+n; j++)
  {
    rx_buff[j] = tmpbuff[j-i]; // load up rx_buff, preserving
```

```
              // whats already there
  }
      i += n;
      timeout--;
    }


  // if timeout, something's wrong
  if (timeout == 0)
    {
      wxLogMessage(wxT("Error:\n␣No␣received␣Data"));
      exit(-1);
    }

  // read 3 bytes from input buffer
  //  ComRd(com_port_no, buffer, 3);
  RS232_PollComport(com_port_no, buffer, 3);

    // assign received value
  *value = buffer[1] * 256 + buffer[2];

  return 0;
}


void read_eeprom_data(void)
{
  int i=0;
  int n = 0;
  BYTE data_a[eeprom_size];
  BYTE tmpdata[eeprom_size];
  packet tmppacket;

  if(!com_port_open)
    {
      wxLogMessage(wxT("Error:␣no␣com␣port␣open"));
      return;
    }

  while (i < eeprom_size)
    {
      // put the n bytes availiable into tmpdata
      n = RS232_PollComport(com_port_no, tmpdata, eeprom_size - 1);
      for (int j = i; j < i+n; j++)
  {
    // load tmpdata contents into next n elements of data
    data_a[j] = tmpdata[j-i];
  }
      i += n;
    }

  // put data from data array into more structured vector
  cfgdata.datalen_u = data_a[0];
  cfgdata.datalen_l = data_a[1];
  cfgdata.samp_period = data_a[2];
  cfgdata.config_3 = data_a[3];
  cfgdata.config_4 = data_a[4];
  cfgdata.config_5 = data_a[5];
  cfgdata.config_6 = data_a[6];
  cfgdata.config_7 = data_a[7];
  cfgdata.config_8 = data_a[8];
  cfgdata.config_9 = data_a[9];
  cfgdata.config_10 = data_a[10];
  cfgdata.config_11 = data_a[11];
  cfgdata.config_12 = data_a[12];
  cfgdata.config_13 = data_a[13];
  cfgdata.config_14 = data_a[14];
  cfgdata.config_15 = data_a[15];

  int datalen = cfgdata.datalen_u * 256 + cfgdata.datalen_l;
  for (int k = 16; k < datalen*10; k += 10)
    {
      tmppacket.temp_u  = data_a[k + 0];
      tmppacket.temp_l  = data_a[k + 1];
      tmppacket.humid   = data_a[k + 2];
      tmppacket.accel_0 = data_a[k + 3];
      tmppacket.accel_1 = data_a[k + 4];
      tmppacket.accel_2 = data_a[k + 5];
      tmppacket.accel_3 = data_a[k + 6];
      tmppacket.accel_4 = data_a[k + 7];
      tmppacket.accel_5 = data_a[k + 8];
      tmppacket.accel_6 = data_a[k + 9];

    data.push_back(tmppacket);

  }

  return;
}


void get_Readings(void)
{
  int n = 0;
  int i = 0;
  int maccelx,maccely,maccelz;
  packet tmppacket;
  BYTE tmpdata[eeprom_size];
  int datalen = 0;

  n = RS232_PollComport(com_port_no, tmpdata, eeprom_size - 1);

  for (i=0; i<n; i++)
    {
      read_buff.push_back(tmpdata[i]);
    }

  while (read_buff.size() >= 10)
    {
    datalen++;
    cfgdata.datalen_u=(datalen>>8)&0xFF;
   cfgdata.datalen_l=datalen&0xFF;

      tmppacket.temp_u  = read_buff.front();
      read_buff.erase(read_buff.begin());
      tmppacket.temp_l  = read_buff.front();
      read_buff.erase(read_buff.begin());
      tmppacket.humid   = read_buff.front();
      read_buff.erase(read_buff.begin());
      tmppacket.accel_0 = read_buff.front();
      read_buff.erase(read_buff.begin());
      tmppacket.accel_1 = read_buff.front();
      read_buff.erase(read_buff.begin());
      tmppacket.accel_2 = read_buff.front();
      read_buff.erase(read_buff.begin());
      tmppacket.accel_3 = read_buff.front();
      read_buff.erase(read_buff.begin());
      tmppacket.accel_4 = read_buff.front();
      read_buff.erase(read_buff.begin());
      tmppacket.accel_5 = read_buff.front();
      read_buff.erase(read_buff.begin());
      tmppacket.accel_6 = read_buff.front();
      read_buff.erase(read_buff.begin());
      data.push_back(tmppacket);
        maccelx = tmppacket.accel_0 & 0x0F;
    if (tmppacket.accel_0 & 0x10) {maccelx=-maccelx;}
    maccelx = (maccelx*1000)/21.33;
    maccely = tmppacket.accel_1 & 0x0F;
    if (tmppacket.accel_1 & 0x10) {maccely=-maccely;}
    maccely = (maccely*1000)/21.33;
    maccelz = tmppacket.accel_1 & 0x0F;
    if (tmppacket.accel_2 & 0x10) {maccelz=-maccelz;}
    maccelz = (maccelz*1000)/21.33;

  temperature->SetLabel(wxString::Format(wxT("%i.%i"),(tmppacket.temp_u*256+tmppacket.temp_l)/100,(tmpp
  humidity->SetLabel(wxString::Format(wxT("%i"),tmppacket.humid));
  accelerationx->SetLabel(wxString::Format(wxT("%i"),maccelx));
  accelerationy->SetLabel(wxString::Format(wxT("%i"),maccely));
  accelerationz->SetLabel(wxString::Format(wxT("%i"),maccelz));
    }

}
```

A.6 rs232.h (not written by me!)

```
/*
***************************************************************************
*
* Author: Teunis van Beelen
*
* Copyright (C) 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013 Teunis van Beelen
*
* teuniz@gmail.com
*
***************************************************************************
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation version 2 of the License.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License along
* with this program; if not, write to the Free Software Foundation, Inc.,
* 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
*
***************************************************************************
*
* This version of GPL is at http://www.gnu.org/licenses/old-licenses/gpl-2.0.txt
*
***************************************************************************
*/

/* last revision: February 1, 2013 */

/* For more info and how to use this libray, visit: http://www.teuniz.net/RS-232/ */


#ifndef rs232_INCLUDED
#define rs232_INCLUDED

#ifdef __cplusplus
extern "C" {
#endif

#include <stdio.h>
#include <string.h>



#ifdef __linux__

#include <termios.h>
#include <sys/ioctl.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <limits.h>

#else

#include <windows.h>

#endif

int RS232_OpenComport(int, int);
int RS232_PollComport(int, unsigned char *, int);
int RS232_SendByte(int, unsigned char);
int RS232_SendBuf(int, unsigned char *, int);
void RS232_CloseComport(int);
void RS232_cputs(int, const char *);
int RS232_IsCTSEnabled(int);
int RS232_IsDSREnabled(int);
void RS232_enableDTR(int);
void RS232_disableDTR(int);
void RS232_enableRTS(int);
void RS232_disableRTS(int);


#ifdef __cplusplus
} /* extern "C" */
#endif

#endif
```

A.7 rs232.cpp (not written by me!)

```
/*
***************************************************************************
*
* Author: Teunis van Beelen
*
* Copyright (C) 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013 Teunis van Beelen
*
* teuniz@gmail.com
*
***************************************************************************
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation version 2 of the License.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License along
* with this program; if not, write to the Free Software Foundation, Inc.,
* 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
*
***************************************************************************
*
* This version of GPL is at http://www.gnu.org/licenses/old-licenses/gpl-2.0.txt
*
***************************************************************************
*/

/* last revision: February 1, 2013 */

/* For more info and how to use this libray, visit: http://www.teuniz.net/RS-232/ */


#include "rs232.h"
```

```
extern "C"{

#ifdef __linux__   /* Linux */


int Cport[30],
    error;

struct termios new_port_settings,
       old_port_settings[30];

char comports[30][16]={"/dev/ttyS0","/dev/ttyS1","/dev/ttyS2","/dev/ttyS3","/dev/ttyS4","/dev/ttyS5",
                       "/dev/ttyS6","/dev/ttyS7","/dev/ttyS8","/dev/ttyS9","/dev/ttyS10","/dev/ttyS11",
                       "/dev/ttyS12","/dev/ttyS13","/dev/ttyS14","/dev/ttyS15","/dev/ttyUSB0",
                       "/dev/ttyUSB1","/dev/ttyUSB2","/dev/ttyUSB3","/dev/ttyUSB4","/dev/ttyUSB5",
                       "/dev/ttyAMA0","/dev/ttyAMA1","/dev/ttyACM0","/dev/ttyACM1",
                       "/dev/rfcomm0","/dev/rfcomm1","/dev/ircomm0","/dev/ircomm1"};



int RS232_OpenComport(int comport_number, int baudrate)
{
  int baudr, status;

  if((comport_number>29)||(comport_number<0))
  {
    printf("illegal␣comport␣number\n");
    return(1);
  }

  switch(baudrate)
  {
    case      50 : baudr = B50;
                   break;
    case      75 : baudr = B75;
                   break;
    case     110 : baudr = B110;
                   break;
    case     134 : baudr = B134;
                   break;
    case     150 : baudr = B150;
                   break;
    case     200 : baudr = B200;
                   break;
    case     300 : baudr = B300;
                   break;
    case     600 : baudr = B600;
                   break;
    case    1200 : baudr = B1200;
                   break;
    case    1800 : baudr = B1800;
                   break;
    case    2400 : baudr = B2400;
                   break;
    case    4800 : baudr = B4800;
                   break;
    case    9600 : baudr = B9600;
                   break;
    case   19200 : baudr = B19200;
                   break;
    case   38400 : baudr = B38400;
                   break;
    case   57600 : baudr = B57600;
                   break;
    case  115200 : baudr = B115200;
                   break;
    case  230400 : baudr = B230400;
                   break;
    case  460800 : baudr = B460800;
                   break;
```

```c
  case  500000 : baudr = B500000;
                 break;
  case  576000 : baudr = B576000;
                 break;
  case  921600 : baudr = B921600;
                 break;
  case 1000000 : baudr = B1000000;
                 break;
  default      : printf("invalid␣baudrate\n");
                 return(1);
                 break;
}

printf("got␣to␣here!");

Cport[comport_number] = open(comports[comport_number], O_RDWR | O_NOCTTY | O_NDELAY);
if(Cport[comport_number]==-1)
{
      while(1);

  perror("unable␣to␣open␣comport␣");

  return(1);
}

error = tcgetattr(Cport[comport_number], old_port_settings + comport_number);
if(error==-1)
{
      while(1);

  close(Cport[comport_number]);
  perror("unable␣to␣read␣portsettings␣");
  return(1);
}
memset(&new_port_settings, 0, sizeof(new_port_settings));  /* clear the new struct */

new_port_settings.c_cflag = baudr | CS8 | CLOCAL | CREAD;
new_port_settings.c_iflag = IGNPAR;
new_port_settings.c_oflag = 0;
new_port_settings.c_lflag = 0;
new_port_settings.c_cc[VMIN] = 0;      /* block untill n bytes are received */
new_port_settings.c_cc[VTIME] = 0;     /* block untill a timer expires (n * 100 mSec.) */
error = tcsetattr(Cport[comport_number], TCSANOW, &new_port_settings);
if(error==-1)
{
      while(1);

  close(Cport[comport_number]);
  perror("unable␣to␣adjust␣portsettings␣");
  return(1);
}

if(ioctl(Cport[comport_number], TIOCMGET, &status) == -1)
{
      while(1);

  perror("unable␣to␣get␣portstatus");
  return(1);
}

status |= TIOCM_DTR;    /* turn on DTR */
status |= TIOCM_RTS;    /* turn on RTS */

if(ioctl(Cport[comport_number], TIOCMSET, &status) == -1)
{   while(1);

  perror("unable␣to␣set␣portstatus");
  return(1);
}
```

```
  return(0);
}


int RS232_PollComport(int comport_number, unsigned char *buf, int size)
{
  int n;

#ifndef __STRICT_ANSI__                       /* __STRICT_ANSI__ is defined when the -ansi option is us
  if(size>SSIZE_MAX)  size = (int)SSIZE_MAX;  /* SSIZE_MAX is defined in limits.h */
#else
  if(size>4096)  size = 4096;
#endif

  n = read(Cport[comport_number], buf, size);

  return(n);
}


int RS232_SendByte(int comport_number, unsigned char byte)
{
  int n;

  n = write(Cport[comport_number], &byte, 1);
  if(n<0)  return(1);

  return(0);
}


int RS232_SendBuf(int comport_number, unsigned char *buf, int size)
{
  return(write(Cport[comport_number], buf, size));
}


void RS232_CloseComport(int comport_number)
{
  int status;

  if(ioctl(Cport[comport_number], TIOCMGET, &status) == -1)
  {
    perror("unable␣to␣get␣portstatus");
  }

  status &= ~TIOCM_DTR;    /* turn off DTR */
  status &= ~TIOCM_RTS;    /* turn off RTS */

  if(ioctl(Cport[comport_number], TIOCMSET, &status) == -1)
  {
    perror("unable␣to␣set␣portstatus");
  }

  close(Cport[comport_number]);
  tcsetattr(Cport[comport_number], TCSANOW, old_port_settings + comport_number);
}

/*
Constant  Description
TIOCM_LE  DSR (data set ready/line enable)
TIOCM_DTR DTR (data terminal ready)
TIOCM_RTS RTS (request to send)
TIOCM_ST  Secondary TXD (transmit)
TIOCM_SR  Secondary RXD (receive)
TIOCM_CTS CTS (clear to send)
TIOCM_CAR DCD (data carrier detect)
TIOCM_CD  Synonym for TIOCM_CAR
TIOCM_RNG RNG (ring)
TIOCM_RI  Synonym for TIOCM_RNG
```

```
TIOCM_DSR DSR (data set ready)
*/

int RS232_IsCTSEnabled(int comport_number)
{
  int status;

  ioctl(Cport[comport_number], TIOCMGET, &status);

  if(status&TIOCM_CTS) return(1);
  else return(0);
}

int RS232_IsDSREnabled(int comport_number)
{
  int status;

  ioctl(Cport[comport_number], TIOCMGET, &status);

  if(status&TIOCM_DSR) return(1);
  else return(0);
}

void RS232_enableDTR(int comport_number)
{
  int status;

  if(ioctl(Cport[comport_number], TIOCMGET, &status) == -1)
  {
    perror("unable␣to␣get␣portstatus");
  }

  status |= TIOCM_DTR;    /* turn on DTR */

  if(ioctl(Cport[comport_number], TIOCMSET, &status) == -1)
  {
    perror("unable␣to␣set␣portstatus");
  }
}

void RS232_disableDTR(int comport_number)
{
  int status;

  if(ioctl(Cport[comport_number], TIOCMGET, &status) == -1)
  {
    perror("unable␣to␣get␣portstatus");
  }

  status &= ~TIOCM_DTR;    /* turn off DTR */

  if(ioctl(Cport[comport_number], TIOCMSET, &status) == -1)
  {
    perror("unable␣to␣set␣portstatus");
  }
}

void RS232_enableRTS(int comport_number)
{
  int status;

  if(ioctl(Cport[comport_number], TIOCMGET, &status) == -1)
  {
    perror("unable␣to␣get␣portstatus");
  }

  status |= TIOCM_RTS;    /* turn on RTS */

  if(ioctl(Cport[comport_number], TIOCMSET, &status) == -1)
  {
    perror("unable␣to␣set␣portstatus");
  }
}

void RS232_disableRTS(int comport_number)
{
  int status;

  if(ioctl(Cport[comport_number], TIOCMGET, &status) == -1)
  {
    perror("unable␣to␣get␣portstatus");
  }

  status &= ~TIOCM_RTS;    /* turn off RTS */

  if(ioctl(Cport[comport_number], TIOCMSET, &status) == -1)
  {
    perror("unable␣to␣set␣portstatus");
  }
}


#else         /* windows */


HANDLE Cport[16];


char comports[16][10]={"\\\\.\\COM1",  "\\\\.\\COM2",  "\\\\.\\COM3",  "\\\\.\\COM4",
                       "\\\\.\\COM5",  "\\\\.\\COM6",  "\\\\.\\COM7",  "\\\\.\\COM8",
                       "\\\\.\\COM9",  "\\\\.\\COM10", "\\\\.\\COM11", "\\\\.\\COM12",
                       "\\\\.\\COM13", "\\\\.\\COM14", "\\\\.\\COM15", "\\\\.\\COM16"};

char baudr[64];


int RS232_OpenComport(int comport_number, int baudrate)
{
  DCB port_settings;
  COMMTIMEOUTS Cptimeouts;


  if((comport_number>15)||(comport_number<0))
  {
    printf("illegal␣comport␣number\n");
    return(1);
  }

  switch(baudrate)
  {
    case     110 : strcpy(baudr, "baud=110␣data=8␣parity=N␣stop=1␣dtr=on␣rts=on");
                   break;
    case     300 : strcpy(baudr, "baud=300␣data=8␣parity=N␣stop=1␣dtr=on␣rts=on");
                   break;
    case     600 : strcpy(baudr, "baud=600␣data=8␣parity=N␣stop=1␣dtr=on␣rts=on");
                   break;
    case    1200 : strcpy(baudr, "baud=1200␣data=8␣parity=N␣stop=1␣dtr=on␣rts=on");
                   break;
    case    2400 : strcpy(baudr, "baud=2400␣data=8␣parity=N␣stop=1␣dtr=on␣rts=on");
                   break;
    case    4800 : strcpy(baudr, "baud=4800␣data=8␣parity=N␣stop=1␣dtr=on␣rts=on");
                   break;
    case    9600 : strcpy(baudr, "baud=9600␣data=8␣parity=N␣stop=1␣dtr=on␣rts=on");
                   break;
    case   19200 : strcpy(baudr, "baud=19200␣data=8␣parity=N␣stop=1␣dtr=on␣rts=on");
                   break;
    case   38400 : strcpy(baudr, "baud=38400␣data=8␣parity=N␣stop=1␣dtr=on␣rts=on");
                   break;
    case   57600 : strcpy(baudr, "baud=57600␣data=8␣parity=N␣stop=1␣dtr=on␣rts=on");
                   break;
    case  115200 : strcpy(baudr, "baud=115200␣data=8␣parity=N␣stop=1␣dtr=on␣rts=on");
                   break;
    case  128000 : strcpy(baudr, "baud=128000␣data=8␣parity=N␣stop=1␣dtr=on␣rts=on");
                   break;
    case  256000 : strcpy(baudr, "baud=256000␣data=8␣parity=N␣stop=1␣dtr=on␣rts=on");
                   break;
    case  500000 : strcpy(baudr, "baud=500000␣data=8␣parity=N␣stop=1␣dtr=on␣rts=on");
                   break;
    case 1000000 : strcpy(baudr, "baud=1000000␣data=8␣parity=N␣stop=1␣dtr=on␣rts=on");
                   break;
    default      : printf("invalid␣baudrate\n");
                   return(1);
                   break;
  }

  Cport[comport_number] = CreateFileA(comports[comport_number],
                      GENERIC_READ|GENERIC_WRITE,
                      0,                            /* no share   */
                      NULL,                         /* no security */
                      OPEN_EXISTING,
                      0,                            /* no threads */
                      NULL);                        /* no templates */

  if(Cport[comport_number]==INVALID_HANDLE_VALUE)
  {
    printf("unable␣to␣open␣comport\n");
    return(1);
  }

  memset(&port_settings, 0, sizeof(port_settings));  /* clear the new struct  */
  port_settings.DCBlength = sizeof(port_settings);

  if(!BuildCommDCBA(baudr, &port_settings))
  {
    printf("unable␣to␣set␣comport␣dcb␣settings\n");
    CloseHandle(Cport[comport_number]);
    return(1);
  }

  if(!SetCommState(Cport[comport_number], &port_settings))
  {
    printf("unable␣to␣set␣comport␣cfg␣settings\n");
    CloseHandle(Cport[comport_number]);
    return(1);
  }

  Cptimeouts.ReadIntervalTimeout         = MAXDWORD;
  Cptimeouts.ReadTotalTimeoutMultiplier  = 0;
  Cptimeouts.ReadTotalTimeoutConstant    = 0;
  Cptimeouts.WriteTotalTimeoutMultiplier = 0;
  Cptimeouts.WriteTotalTimeoutConstant   = 0;

  if(!SetCommTimeouts(Cport[comport_number], &Cptimeouts))
  {
    printf("unable␣to␣set␣comport␣time-out␣settings\n");
    CloseHandle(Cport[comport_number]);
    return(1);
  }

  return(0);
}


int RS232_PollComport(int comport_number, unsigned char *buf, int size)
{
  int n;

  if(size>4096)  size = 4096;

/* added the void pointer cast, otherwise gcc will complain about */
```

```c
/* "warning: dereferencing type-punned pointer will break strict aliasing rules" */

  ReadFile(Cport[comport_number], buf, size, (LPDWORD)((void *)&n), NULL);

  return(n);
}


int RS232_SendByte(int comport_number, unsigned char byte)
{
  int n;

  WriteFile(Cport[comport_number], &byte, 1, (LPDWORD)((void *)&n), NULL);

  if(n<0)  return(1);

  return(0);
}


int RS232_SendBuf(int comport_number, unsigned char *buf, int size)
{
  int n;

  if(WriteFile(Cport[comport_number], buf, size, (LPDWORD)((void *)&n), NULL))
  {
    return(n);
  }

  return(-1);
}


void RS232_CloseComport(int comport_number)
{
  CloseHandle(Cport[comport_number]);
}


int RS232_IsCTSEnabled(int comport_number)
{
  int status;

  GetCommModemStatus(Cport[comport_number], (LPDWORD)((void *)&status));

  if(status&MS_CTS_ON) return(1);
  else return(0);
}


int RS232_IsDSREnabled(int comport_number)
{
  int status;

  GetCommModemStatus(Cport[comport_number], (LPDWORD)((void *)&status));

  if(status&MS_DSR_ON) return(1);
  else return(0);
}


void RS232_enableDTR(int comport_number)
{
  EscapeCommFunction(Cport[comport_number], SETDTR);
}


void RS232_disableDTR(int comport_number)
{
  EscapeCommFunction(Cport[comport_number], CLRDTR);
```

```
}

void RS232_enableRTS(int comport_number)
{
  EscapeCommFunction(Cport[comport_number], SETRTS);
}


void RS232_disableRTS(int comport_number)
{
  EscapeCommFunction(Cport[comport_number], CLRRTS);
}


#endif


void RS232_cputs(int comport_number, const char *text)  /* sends a string to serial port */
{
  while(*text != 0)   RS232_SendByte(comport_number, *(text++));
}


}
```

B First Interim Report

C Marketing Datasheet

References

[1] http://www.cambridgeconsultants.com/news/pr/release/116/en. Feb 2013. Accessed on 3/6/13.

[2] Oxford Economic Forecasting. The impact of the express delivery industry on the global economy. http://eiciindia.org/frontsite/An%20Oxford%20Economic%20Forecasting%20-%20The%20impact%20of%20the%20Express%20Delivery%20Industry%20on%20the%20global%20economy.pdf, 2005. Accessed on 3/6/13.

[3] P. Horowitz and W. Hill. *The Art of Electronics*, chapter 15: measurements and signal processing, pages 992–993. Cambridge University Press, second edition, 1989.

[4] http://news.van.fedex.com/SenseAware. Nov 2012. Accessed on 3/6/13.

[5] Teunis van Beelen. Rs-232 for linux and windows. http://www.teuniz.net/RS-232/.

[6] http://wiki.wxwidgets.org/Making_a_render_loop.