Build your own

MOBILE WEATHER STATION

An open-source guide

All the information in this document is available in the public domain

CGIAR RESEARCH PROGRAM ON Water, Land and Ecosystems

IWMI International Water Management Institute 1985 2015 30

CGIAR RESEARCH PROGRAM ON Climate Change, Agriculture and Food Security CCAFS

Weather Station

Niroshan Sanjaya, nsanj88@gmail.com
Yann Chemin, ychemin@gmail.com

Abstract: Different types of weather stations are available online or through specialized shops, their cost can be high and their internals locked to the manufacturer's decision. From the wave of Open Design came the Open Source Hardware (OSHW) implementations. The aim of this manual is to leverage both OSHW and Open source software to guide, step-by-step, the creation of a weather station based on the Open Design, OSHW, OSS and commodities off the shelf (COTS) from local shops in Sri Lanka.

Keywords: Open Design, weather station, OSHW, FOSS, COTS, Lakduino, Arduino.

1 Equipment needed

To check the weather without yourself we can build Arduino powered weather station. There are few equipment we need to use for this weather station. Some of them are we can find locally, otherwise there are some web pages we can order via internet. In this study open source hardware are utilized. Therefore you can find the same hardware in different product/brand names. Arduino is a single-board microcontroller to make using electronics in multidisciplinary projects more accessible. Arduino can be found in different names such as Red Board, Lakduino, etc… but the usage and the features are same. Any of them can be utilized based the needs and design. This study is basically focused for local market. Thereforee Lakduino is used to develop this weather station. Lakduino is Sri Lankan product you can easily find in local market. Other equipment which are using for weather station is mention in Table 1.

Table 1 List of equipment and their prices (this prices are based on 2014 market price and 1$ = 130.50LKR)

Item	***Price***	***Quantity***
Wind Speed Meter	70$	1
Direction Meter		1
Rain Gauge		1
Mounting Hardware		1
Lakduino	2,000LKR	1
Weather Shield	40$	1
GPS Receiver - GP-635T	40$	1
Real Time Clock Module	15$	1
Open Log Data logger	25$	1
Cables / plugs	500LKR	
GSM module (further development)	60$	1
Power source	7,000LKR	1
Total	*42,225LKR*	

Apart from above mentioned equipment you may have to use some of other equipment or materials based one the design. When considering all the above equipment, the main detected object is Weather Meters. That is the one that stands out when looking at the weather station.

Weather Meters consist of a wind speed meter, a direction meter and a rain gauge. It can be consider as a one set. The rain gauge is a self-emptying bucket-type rain gauge which activates a momentary button closure for each 0.011" (0.2794mm) of rain that are collected. The anemometer (wind speed meter) encodes the wind speed by simply closing a switch which each rotation. A wind speed of 1.492 MPH produces a switch closure once per second. Finally, the wind vane reports wind direction as a voltage which is produced by the combination of resistors inside the sensor. When a voltage is supplied, the voltage returned can be translated to any of 16 possible positions. For more information on how this works, as well as a table of voltage and resistance values for each position. When you buying them, apart from those three items mounting hardware also included with them. Or else you can prepare your own design. In this study the given hardware is utilized.

Figure 1 Mounting the Wind speed, Direction meters and Rain Gauge

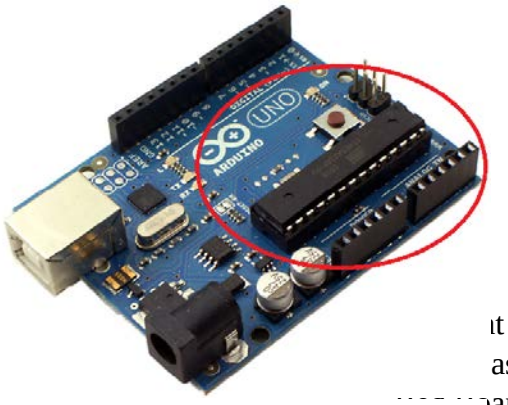


Figure 2 Arduino Uno

The main equipment we are using is Arduino. Lakduino has been operated in this study though different Arduino boards are available in the market. You can use any other types of Arduino boards based on the needs. Few physical differences can be found in different Arduino boards. When Red Board is considered, there is no such a big IC compare to the Arduino Uno Board. And the plugin cables are

[illegible]t in each. Mini USB
[illegible]as been used in the
[illegible]ard but Type B USB
[illegible]as been used in the

Arduino Uno Board although the system and capabilities are same.

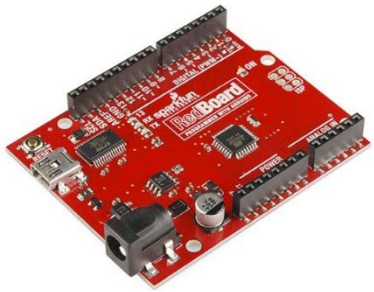
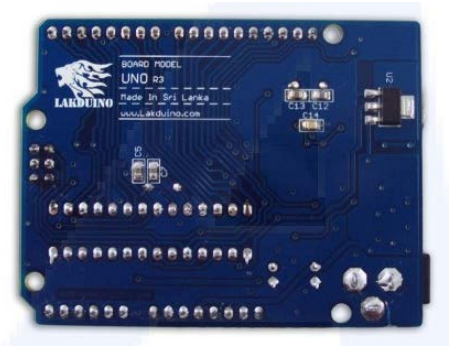


Figure 3 Red Board - Programmed with Arduino

The design of the Lakduino is also somewhat similar to Arduino Uno. Lakduino can be obtained via internet or from shops in locally (Appendix A).



When considering the input and output voltage of Arduino we can find Operating Voltage as 5V, Input Voltage (recommended) 7V-12V, Input Voltage (limits) 6.5V-15V and output DC Current per I/O Pin 40 mA and DC Current for 3.3V Pin 50 mA. Those figures are important when designing power supply for the system. And output Voltage figures are important when plugging shields on Arduino.

Another one of the main equipment used for this setup is a Weather Shield. When considering the Weather Shield design, it is an easy to use, Arduino shield that grants you access to barometric pressure, relative humidity, luminosity and temperature. There are also connections on this shield to optional sensors such as wind speed, direction, rain gauge and GPS for location and super accurate timing. You can collect all of above measures by using this device.

Figure 4 Lakduino Board

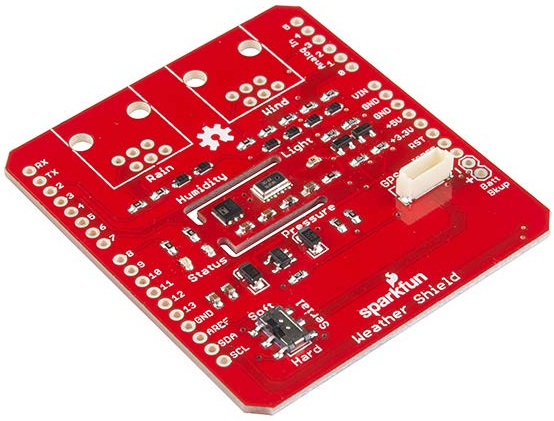


Figure 5 Weather Shield

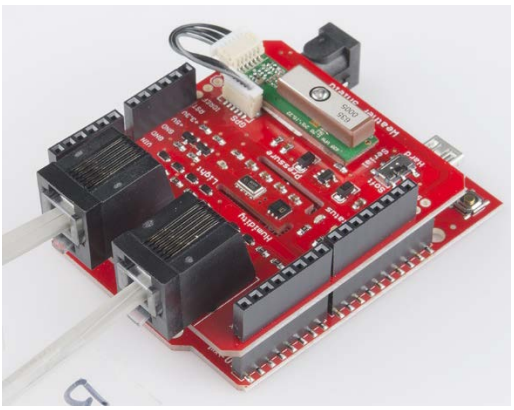
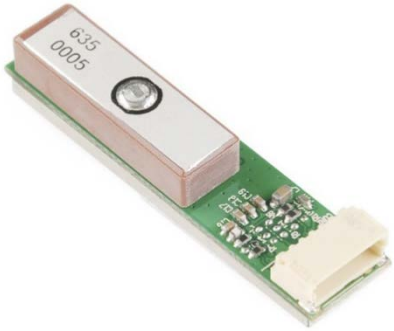


Figure 6 Weather Shield with Arduino

In this picture (Figure 5) you can see just the board with sensors. Thereforee you have to solder the wire/ plugs and phone cable plugs on the board. Then you can plug weather shield on the Arduino Board as shown in Figure 6.



To get the location here we are using a GPS module that help to give the coordinates of the

location. GPS Receiver - GP-635T has been used in this experiment. That can be easily plug on the weather shield. The GP-635T is a slim GPS receiver with -161dBm tracking sensitivity and only 27 second cold start time. The slim design makes it ideal for applications where you don't have a lot of space to work in. Really it's quite small. This 50-channel GPS module, based on the uBlox-6 chipset, has an antenna on board and connects to your system via TTL serial. The 1Hz update rate is fast enough for the majority of applications (and can be increased to 5Hz if you need).

Figure 7 GPS Module

GPS module also give GTM time of the location but for calibration and further purposes we are using a real time clock. That is giving the time stamps of each readings. This real time clock is a custom designed module for the DS1307 Real Time Clock. The module comes fully assembled and pre-programmed with the current time. The included Lithium coin cell battery (CR1225 41mAh) will run the module for a minimum of 9 years (17 years typical) without external 5V power. The DS1307 RTC is accessed via the I2C protocol.

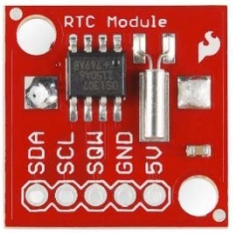


Figure 8 Real Time Clock

Now we have all the equipment but there is one question that is where this data is going to save, of course we can use GSM module to transfer data but in this stage there should be a method to save data. For that we are using an OpenLog data logger. OpenLog is an open source data logger. Simple to use, simple to change. We wanted to create a serial logger that just worked. Power up this device and it will start logging any received serial data at 9600bps. Sending Ctrl+z three times will drop out of logging and in to command mode. 'new' will create a new file. 'md' makes a directory. '?' brings up the list of commands. OpenLog doesn't do a lot, but it does log serial streams extremely well. OpenLog firmware is open source. OpenLog currently supports FAT16 and FAT32 microSD cards (it has been tested up to 16GB!). Thereforee all the data will be saved on MicroSD. That can be open by using Excel or Notepad because data will be saved as CSV file.

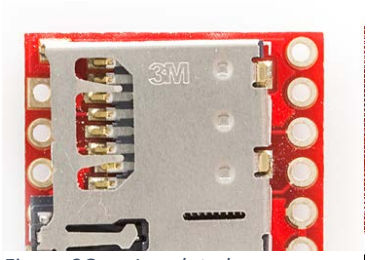


Figure 9OpenLog data logger

All the equipment is available now. Now we need to find a power source for that. There are several options for that. One is we can use 5V battery or we can give direct from computer but that is not possible because we cannot bring the computer to field and keep, Thereforee we are going to use a rechargeable battery with solar panel.

2 Power

Power supply is a key component of this system. Because there should be a continuous power supply otherwise the system will not work continuously. There can be many sustain way to give continuous power supply. Here rechargeable battery is used with a solar panel. Arduino with tolerate with 5V~15V. Therefore we can use any of the voltage inputs between that. Further if GSM module is going to be added it consume more power Therefore we need to ready for that.

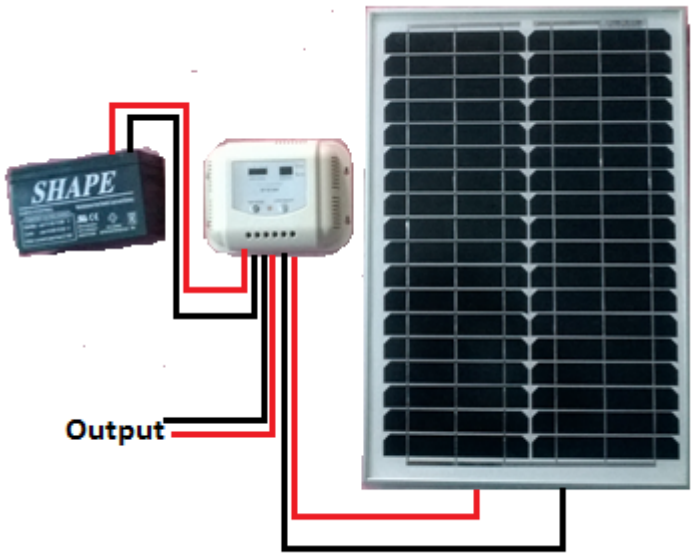


Figure 10 Component of the power supply and the connections

By considering all above facts, 12V 7AH battery is going to be used in this project. If a Solar panel is using for a source for recharge the battery. Then that should be suit for that. There can be many different solar panel we can find in the market. Normally you need to know how much energy your battery can store and then select a Solar panel that can replenish your 'stock' of energy in the battery in line with your pattern of use.

Battery capacity is measured in Amp Hours (7AH). You need to convert this to Watt Hours by multiplying the AH figure by the battery voltage (12V).

For a 7AH, 12V battery the Watt Hours figure is 7 x 12 = 84WH

The power generation rating of a Solar panel is also given in Watts (here we are using 20W). To calculate the energy it can supply to the battery, multiply Watts by the hours exposed to sunshine, then multiply the result by 0.85 (this factor allows for natural system losses).For the Solar 20W panel in 4 hours* of sunshine, 20 x 4 x 0.85 = 68WH. This is the amount of energy the Solar panel can supply to the battery. Therefore, we need to think about that factors when selecting a solar panel to the battery. In this case we are using a 20W solar panel. Apart from that solar controller should be used to connect solar panel, battery and the Arduino. You can buy it from local market or you can design it by yourself or giving to electrician. The way solar panel and battery is plugging is shown in Figure 11.

3 Setup

Now we already identified all the equipment, their specification and capacities. Now we need to connect them as a one set. First you can plug weather shield on the Arduino, which can be consider as the first step. Once it plug you can see it as below Figure 11.

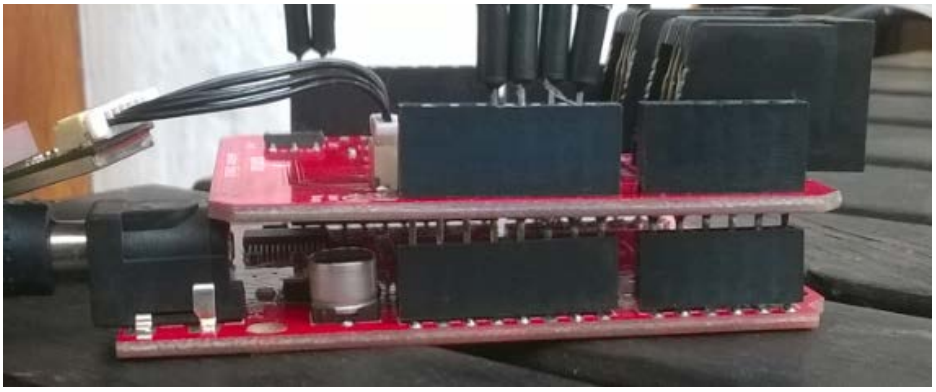
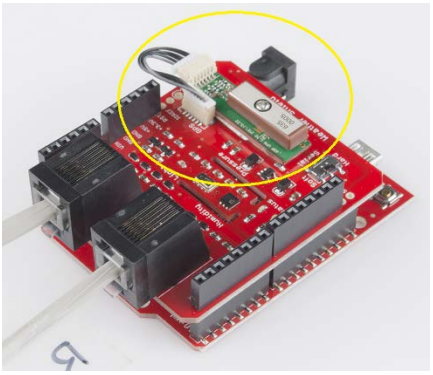


Figure 11 once after plug weather shield on the Red Board



Now you can see the same arrangement of pins on the weather shield as on the Arduino. Then GPS module can be easily plug on the weather shield.

Then you can chose either RTC or OpenLog, Here RTC has been selected. You need to connect RTC to the Arduino as shown in Figure 12. Otherwise there may be confusions and issues will arise when programming the Arduino.

Figure 12 After plugging the GPS module

Once you connect RTC to the Arduino as Figure 13. Then you can go for OpenLog.

You need to use only three wires when you are going to plug OpenLog as shown in Figure 14. You have to follow the given instruction. Basically connection to the proper end from proper start.

There can be many option when you are plugging modules to the board as the prototype plugging wires are used one end solder to module. That is up to designer. Because you can use any method to wiring these. Only thing you need to consider is the plugging number. That mean right start

should go to right end. Otherwise there may be lot of difficulties when running the Arduino. Finally you can plug Wind and Rain meters to weather shield that is very easy task you can easily plug. The inputs are marked on the board therefore you can plug it properly. Finally the system is looking like this (Figure 15).

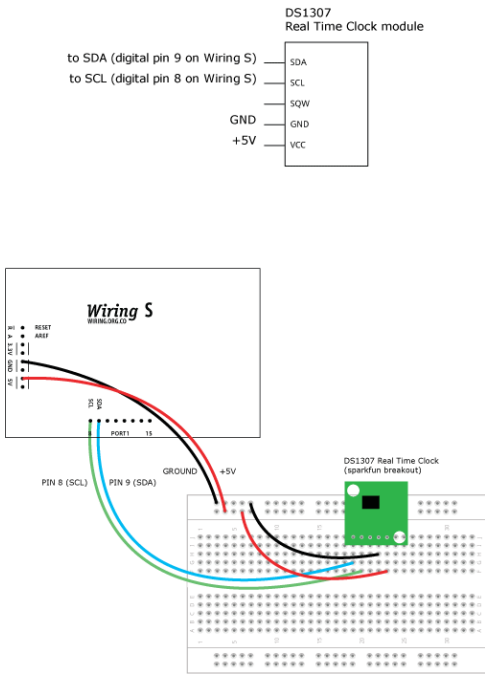


Figure 13 RTC wiring

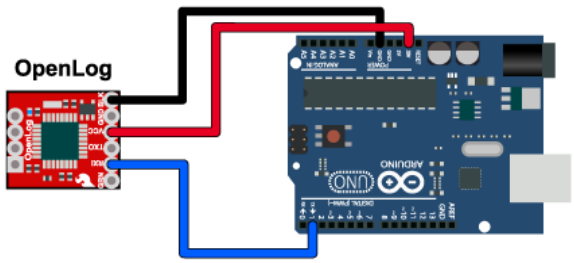
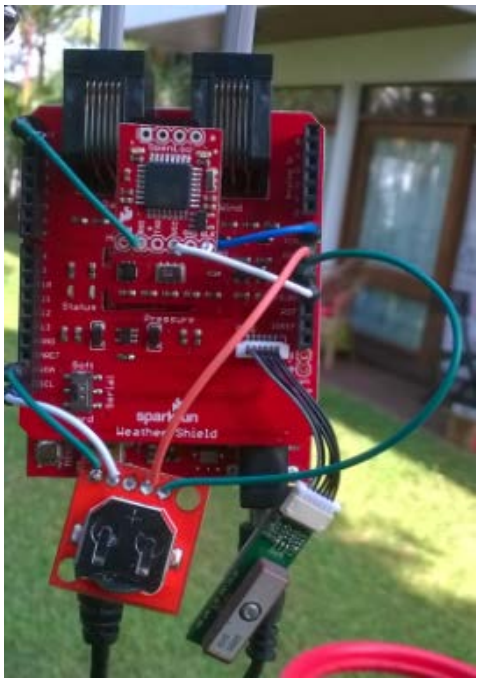


Figure 14 OpenLog wiring



Once Arduino is setup, it is necessary to cover the weather station with a radiation shield to correct temperature measurements. Another way is to plug a temperature probe and also protect it by a radiation shield is described in Appendix B.

All the modules and components are now plugged properly to the Arduino. Now you can power up the Arduino by giving the power supply. It will not work properly without programming the Arduino properly. In the next step will explain how to program the Arduino for weather station.

Figure 15 once after plug all the modules

4 Arduino box

There is no any fixed box for Arduino. Any box can be utilized based on the design. Basically Weather station is going to locate somewhere outside of home or office Therefore we need to think about the durability vs rain and radiation of the Arduino box. It should be covered from rain and sun radiation. If it is get wet due to rain there can be short circuit and it can be harm to the board. If id direct to the sun radiation the temperature of the devices can be increase Therefore temperature can be harm to the Arduino Board. There can be many solution for that. In this case we are going to use an Aluminum box (Figure 16). It will be covered by a radiation resistant layer. Arduino with weather shield is going to fix in that particular box.

Figure 16 Aluminum Box for Arduino

5 Mounting

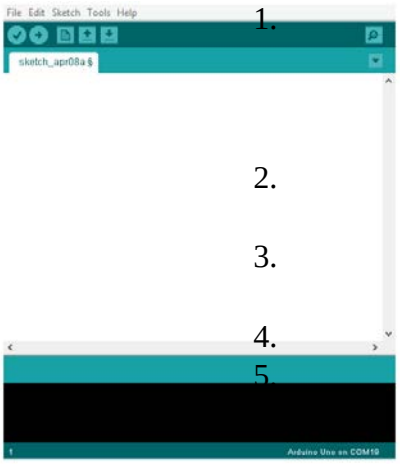
There can be different method and technique when you are mounting the weather station at a particular location. There is no any rules to follow when you are mounting the weather station. Although there is a one rule you need to follow. That is when the weather station is fixing some location you need to check about the North Direction. On the wind direction meter there is a mark to show the North, Therefore the wind direction meter should be locate based on that. It should be turned and adjusted toward to the North Direction. Another thing is when you are locating the Weather Station keep mind not to fix somewhere in shelter or where water directly come to the Rain Gauge from a roof or any other water flow. Other than that weather station can be fixed based on the needs.

Figure 17 Fixed in Anuradhapura Irrigation department (Test Running)

6 Programming the Arduino

Now you already plug the modules to the Arduino. When you are programming the Arduino that is not important to keep all the modules plugged. You can just unplug and program the Arduino, then you can plug the modules. Once you power up the Arduino it will run the uploaded code. You can power it up by using USB or direct power source. No need to keep it plugged on the PC. The way of giving power is described in earlier steps.

Before programming, you have to follow few steps,



1. You need to download Arduino IDE and it should be installed to your computer. You can download it by using following link (http://arduino.cc/en/main/software).
2. Then you can connect Arduino to the computer and run the Arduino software.
3. You need to select the right port, you can follow the instruction from above link.
4. Now you can type the code and verify them
5. Finally you can upload them to the board, when it is uploading LED which is on the Arduino will be blinks.

Figure 18 User interface of the software

For the weather station code is shown in below Appendix C. Once you follow the above steps you can simply copy (from Appendix C) and paste on Arduino sketch and you can simply uploaded it in to the Arduino.

7 Output and Data

The reading are saved as CSV (Comma-separated values) file as menstioned erliar. Once the SD card pluged to the computer and open it, files are shown inside the SD card. It would be shown as .txt extention file. If the file is open by using note pad it whould be shown as below.

“ $,lon=80.418937,lat=8.321109,altitude=0.00,sats=5,date=2014-04-09,time=08:46:17,RTCdate=20000101,RTCtime=00:00:00,winddir=0,windspeedms=nan,windgustms=0.0,windgustdir=0,windspdms_avg2m=0.3,winddir_avg2m=253,windgustms_10m=3.8,windgustdir_10m=225,humidity=48.2,tempc=44.6,rainhourmm=0.56,raindailymm=0.56,pressure=99763.50,batt_lvl=5.16,light_lvl=3.16,# “

You can see in the example all the data are separated by a comma. Now extention can be changed as .txt to .csv or you can open with LibreOffice Calc or MSExel.

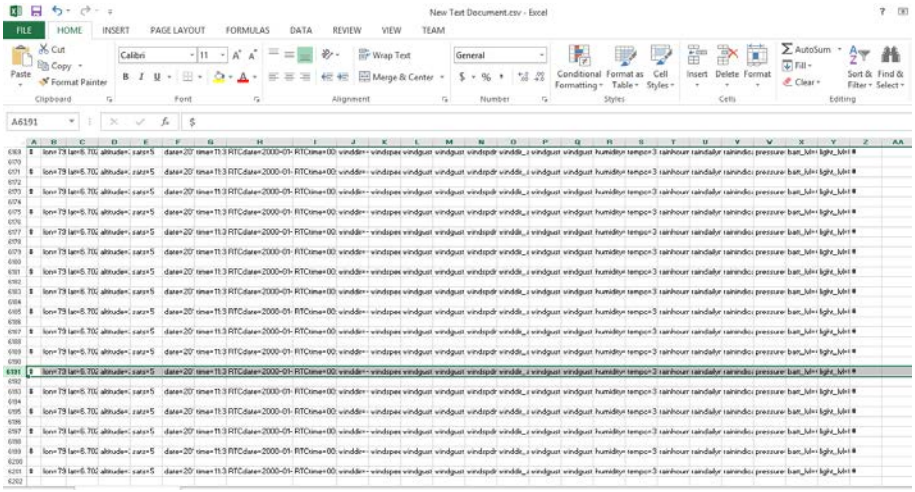


Figure 19 Sample MSExel datasheet

Sample dataline

$ lon=79.912277 lat=6.702870 altitude=26.50 sats=5
date=2014-05-09 time=11:39:34 RTCdate=2014-05-09 RTCtime=17:09:00
winddir=-1 windspeedms=nan windgustms=0.0 windgustdir=0
windspdms_avg2m=0.0 winddir_avg2m=-1 windgustms_10m=0.0 windgustdir_10m=0
humidity=70.8 tempc=32.2 rainhourmm=1.40 rain5mmm=1.40
raindailymm=3.91 rainindicate=1 pressure=100810.75 batt_lvl=4.27
light_lvl=0.02 #

In the example data show a one line of readings. In the line you can see the different informations. First two values are the GPS reading. It shows as the longitude and lattitue, and the next value is the altitude. First five values are the readings of the GPS module. In the exel sheet the third row you can see the number of satellite which connected with the GPS module. Dates and time are in GMT. Next column you can see the time stamps that is the reading of RTC clock. Next Columns for the wind meters, which shows the wind speed, direction and the average. You can customize the average according to your study when programming the Arduino. Relative humidity is shown in the next column. And the next column for the temperature, in this system value is indicated in Celsius if you want you can change it as Fahrenheit. Next column are for the rain guage readings, in this study wee need to identify the rain event with 5mnts accuracy. Therefor here the rain guage readings are given in the hourly, daily and the 5mnts millimeters. And the rain event indicate wih 5mnts accuracy. When there is rain it shows as “1” and if there is no any rain it shows as “0”. By using that we can customize the rainfall events with 5 mnts accuracy. Pressure would be indicated in the next columns. Final columns for the battry level and the light level. Those are the values which are given by the system. You can use those values based on the study and the rqurement. Is the parmeters should be changed you can do it when programming the Arduino.

8 Discussion

Basically when we are fixing modules with Arduino we need to do it very correctly and otherwise the modules will not give the accurate or expected results. When programming also we need to program it with our needed. As an example in this case Rain Gauge is programed to give hourly rain fall. With the requirement it has been changed to 5 minutes. Then we can identify the rain duration with 5mnts accuracy. Lot of issues and troubles will arise when programming the Arduino. Therefore it should be considered as a key step of the process. And the codes should be properly use specially when declaring variables. When designing the box you need to keep remember about the sensors which are located on the weather shield. There are light, temperature and pressure sensors. Those sensors should be exposure to the environment. To measure the pressure it is enough to have a hole. But for the light sensor it should be open to environment. But that is based on the study or research which you are going to conduct. Sometime values which is given by the light or thermal sensor will not be useful, in that case you can ignore those sensor and design a box. When you are using RTC (Real Time Clock) before utilize it you need to program it to the local time. Otherwise it will not give the accurate information. That should be keep in mind before you are going to plug it.

Designing the mounting hardware and the fixing some location based on the particular location. Some time it is better to fixed on the iron or aluminum tube and put concrete and fixed it or it can be fixed somewhere on a slab or roof top. It is basically depend on the location. Other fact we need to consider is security of the devices. If it is going to locate in remote area and collect data through GSM module, security will be an issue. Those factors need to be considered when the system is going to locate.

Weather stations which are available in the market are basically fixed to some particular function in that case this can be customize based on the need on the study or research. And all the component can be buy from local market. Thereforee anybody can be able to build their own weather stations. That will be good advantage for the researchers.

Appendix A

Online Resources

Lakduino and other small accessories can be found at www.lankatronics.com:

http://www.lankatronics.com/lakduino.html

The weather shield is made by Sparkfun.com:

https://www.sparkfun.com/

Appendix B

Radiation Shield DIY

Abstract: There can be different methods and materials to build radiation shield DIY. Doing that may confuse you when following different steps as a beginner. Those are explained in various contexts. The aim of this manual is to describe the way, how to create a radiation shield with commodities off the shelf (COTS) from local shops in Sri Lanka.

B.1. Equipment needed

Different equipment can be used according to your interest. These are the equipment which we found in local shops and it seems those are very much suitable for this work. Here is the list of items and the existing market prices. Those prices are the maximum prices and you may find some equipment less than these mentioned prices.

Table 2: List of items and current market prices

Item	***rice***	***antity***
Plastic cup	70	4
hreaded bar	210	1
Steel hook	50	1
Bottom cup	20	1
Nuts	10	34
Pen	15	1
Lunch box		1
Total		

Apart from above list you may have to use few drops of silicon gum. That will be very useful for water proofing and fix items to the box. Silicon gum tube can be bought around 400LKR. Then you need to buy a gun which is also around 300LKR. You have to use small volume for one set.

Figure 20: Radiation Shield DIY parts

All the items can be found from Sri Lankan grocery shops or from Sunday fair. You are able to build a radiation shield by using only these equipment. You can chose any other objects more or less similar to these. There can be plenty of items you can find from any kind of shops.

If all the items are with you now let's start to make the radiation shield.

B.2. Making holes in plastic cups

In this image too much holes as the testing one. But you may have to make only three small holes in the cups and one big hole in the middle of them you have to keep one for the top without making a middle hole. Many different method cane be used to make holes such as using sharp objects, drill machine or solder iron. The material which we use is strong than plastics Thereforee it is difficult to use sharp object to make holes. Solder iron is one of the best option to make holes without any crack. Or else you can use high speed drilling machine if you have that. When you are making holes on the bottom cup that is easier. Because, that is made by soft plastics. But the cups are stronger than bottom cup. When you are making holes there can be certain issues you will face. One is holes are not parallel to each cup. If so you cannot insert threaded rod directly. It is better to follow below steps to make it easier.

Steps

1. Put all the cups together as layers
2. Mark a line or any mark that is easy to overlap the cups again as the first step.
3. Make three holes on the top cups
4. Follow the top cup when you making holes in other cups.
5. Remove the first cup
6. Make a hole in the middle on the second cup and make holes on others as the second cup
7. Use a sharp object and resize the holes as you want

Figure B.21 making holes on the bottom cup

Figure B.3 Making holes on the cups

Figure B.4 Making second and third holes

When you are making holes you need to do them as same in every cups. For that, small trick can be used, what you have to do is once you make a first hole you can insert a threaded rod in to it and make the second hole, once you make the second hole use the second thread bar and go for third.

B.3. Central axis

Once we finished the bottom cup of the radiation shield we need to find a method to put a thermistor prob. Central axis can be used to fix the thermistor prob. To make a central axis we need to find an equipment. There can be different solutions for that. One method is we can use another threaded rod but it is important to note that, when it is getting heated due to radiation of sun and that can be an impact to change the readings of thermistor. Therefore it is better to use a material which is not absorbing heat, to overcome that issue.

Another equipment that can be used for that: is a pen tube. You can use the tube of a used pen or you can buy a new pen and take the tube (figure B.5).

Figure B.5 Tube of a used pen

It can be fixed easily because it already has thread and the size of hole on the bottom cup matches with it. Thereforee it can be easily tied. You can use any other object as the center axis. But make sure not use a metal one since it can be an effect to readings of thermistor. Thereforee when you are selecting the central axis keep in mind about the material that is made of.

B.4. Building up the cups on the threaded bar

All the preliminary preparations are done now. Now the time is to start building up the cups on threaded bar.

When you buy a threaded rod, length of it should be more than 1m long. You need to cut three parts from threaded rod: length should be not more that 25cm.

First start to build with bottom cup.

Figure B.6
Start with the bottom cup

Figure B.7
Fix thermistor and other cups

Figure B.8
Building up with other cups

It is better to start with central axis that should be fixed first. Then threaded bar can be fixed one by one. There should be two screw nuts on both side of cups and tie it well. (Figure B.6)

And once completed the bottom cup, then thermistor probe can be inserted through the middle hole. It can be tied up with aluminum or nylon thread. To ensure the knot you can put silicon drop on it. (Figure B.7)

Then one by one, you can fix the rest of the cups. Keep in mind to put two screw nuts on both sides of each cup and the keep the distance between two cups similar to each one of them. The top cup should not be touched with the thermistor; to ensure that, there should be a significant distance between the top cup and the thermistor. (Figure B.8)

B.5. Mounting Frame

Once after completing the previous step mounting frame should be fixed on it. There can be different ways to fix it. Considering weight and balance it is enough to use two threaded bar to fix it. Otherwise we can fix it with three threaded bar by using a metal board or something similar on top of the cups. Distance between handle and the cup can be changed by cutting the thread bar (Figure B.9). You can use some innovative ideas to make it looks nice.

Mounting frame consist with few holes or otherwise we can make holes as we need. By using that we can mount the radiation shield to a wall or a pole. Here we have mounted it to a pole. To mount it you can use screw nut or bolt nut; easily you can mount it to a pole (Figure B.10).

Figure B.9 Radiation Shield with the mounting frame

Figure B.10 mounted radiation shield

B.6. Discussion

Buying a radiation shield is very expensive. Instead of buying such an expensive item you can make it by yourself using local material. To make a radiation shield you do not need to worry about the required equipment and material. You can simply find them from local shops either grocery shop or Sunday fair. When you are selecting cups, you do not need to think about shapes and sizes; you can choose whatever you prefer. There are few things you have to keep in mind when you are building up the radiation shield. Most importantly, the distance between two cups. When you decide the distance, which should not be allowed to reach the sun radiation to thermistor and the other thing is the distance between thermistor and the top cup. Thermistor should not touch the top cup and there should be a significant distance between them. Other than that there are no any difficulties when you are making a radiation shield.

Appendix C

Code uploaded to the Arduino Weather Station

```
#include <Wire.h> //I2C needed for sensors
#include "MPL3115A2.h" //Pressure sensor
#include "HTU21D.h" //Humidity sensor
#include <SoftwareSerial.h> //Needed for GPS
#include <TinyGPS++.h> //GPS parsing

//this is for RTC
int clockAddress = 0x68;  // This is the I2C address
int command = 0; // This is the command char, in ascii form, sent from the serial port long
previousMillis = 0; // will store last time Temp was updated
byte second, minute, hour, dayOfWeek, dayOfMonth, month, year;

byte decToBcd(byte val)
{
return ( (val/10*16) + (val%10) );
}

//         Convert binary coded decimal to normal decimal numbers
byte bcdToDec(byte val)
{
return ( (val/16*10) + (val%16) );
}

//         Gets the date and time from the ds1307 and prints result
char* getDateDs1307(int flag) {
//if flag == 0 : date output
//if flag == 1 : time output
//Reset the register pointer
Wire.beginTransmission(clockAddress);
Wire.write(byte(0x00));
Wire.endTransmission();

Wire.requestFrom(clockAddress, 7);

// A few of these need masks because certain bits are control bits
second = bcdToDec(Wire.read() & 0x7f);
minute          = bcdToDec(Wire.read());

// Need to change this if 12 hour am/pm
hour            = bcdToDec(Wire.read() & 0x3f);
dayOfWeek  = bcdToDec(Wire.read());
dayOfMonth = bcdToDec(Wire.read());
month           = bcdToDec(Wire.read());
year            = bcdToDec(Wire.read());


char sza[32];
```

```
if (flag==0)
sprintf(sza, "%02d-%02d-%02d",year,month,dayOfMonth);
if (flag==1)
sprintf(sza, "%02d:%02d:%02d",hour,minute,second);
return(sza);
}
//end of RTC


TinyGPSPlus gps;

static const int RXPin = 5, TXPin = 4; //GPS is attached to pin 4(TX from GPS) and pin 5(RX
SoftwareSerial ss(RXPin, TXPin);

MPL3115A2 myPressure; //Create an instance of the pressure sensor
HTU21D myHumidity; //Create an instance of the humidity sensor

//Hardware pin definitions
//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
//digital I/O pins
const byte WSPEED = 3;
const byte RAIN = 2;
const byte STAT1 = 7;
const byte STAT2 = 8;
const byte GPS_PWRCTL = 6; //Pulling this pin low puts GPS to sleep but maintains RTC and R

//analog I/O pins
const byte REFERENCE_3V3 =A3;
const byte LIGHT = A1;
const byte BATT = A2;
const byte WDIR = A0;

//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
//Global Variables
//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
long lastSecond; //The millis counter to see when a second rolls by
byte seconds; //When it hits 60, increase the current minute
byte seconds_2m; //Keeps track of the "wind speed/dir avg" over last 2 minutes array of dat
byte minutes; //Keeps track of where we are in various arrays of data
byte minutes_10m; //Keeps track of where we are in wind gust/dir over last 10 minutes array
byte minutes_5m; // Niroshan this is for 5 mnts rain

long lastWindCheck = 0; volatile
long lastWindIRQ = 0; volatile
byte windClicks = 0;
```

```
//We need to keep track of the following variables:
//Wind speed/dir each update (no storage)
//Wind gust/dir over the day (no storage)
//Wind speed/dir, avg over 2 minutes (store 1 per second)
//Wind gust/dir over last 10 minutes (store 1 per minute)
//Rain over the past hour (store 1 per minute)
//Total rain over date (store one per day)

byte windspdavg[120]; //120 bytes to keep track of 2 minute average
int winddiravg[120]; //120 ints to keep track of 2 minute average
float windgust_10m[10]; //10 floats to keep track of 10 minute max //Yann: to check if OK
volatile float rain5m[5]; //Niroshan 5 float to keep rain data of 5 mnts //Yann: added vol
int windgustdirection_10m[10]; //10 ints to keep track of 10 minute max
volatile float rainHour[60]; //60 floating numbers to keep track of 60 minutes of rain

//These are all the weather values that wunderground expects:
int winddir = 0; // [0-360 instantaneous wind direction]
float windspeedms = 0; // [mph instantaneous wind speed]
float windgustms = 0; // [mph current wind gust, using software specific time period]
int windgustdir = 0; // [0-360 using software specific time period]
float windspdms_avg2m = 0; // [mph 2 minute average wind speed mph]
int winddir_avg2m = 0; // [0-360 2 minute average wind direction]
float windgustms_10m = 0; // [mph past 10 minutes wind gust mph ]
int windgustdir_10m = 0; // [0-360 past 10 minutes wind gust direction]
float humidity = 0; // [%]
float tempf = 0; // [temperature F]
float rainin = 0; // [rain inches over the past hour)] -- the accumulated rainfall in the p
float rainin_5m = 0; // [rain inches over the past hour)] -- the accumulated rainfall in th
volatile float dailyrainin = 0; // [rain inches so far today in local time]
//float baromin = 30.03;// [barom in] - It's hard to calculate baromin locally, do this in
float pressure = 0;

//float dewptf; // [dewpoint F] - It's hard to calculate dewpoint locally, do this in the a

float batt_lvl = 11.8; //[analog value from 0 to 1023]
float light_lvl = 455; //[analog value from 0 to 1023]
//Rain time stamp Niroshan
int Rainindi=0;
//Variables used for GPS
//float flat, flon; // 39.015024 -102.283608686
//unsigned long age;
//int year;
//byte month, day, hour, minute, second, hundredths;

// volatiles are subject to modification by IRQs
volatile unsigned long raintime, rainlast, raininterval, rain;
```

```
//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
//Interrupt routines (these are called by the hardware interrupts, not by the main code)
//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
void rainIRQ()
//   Count rain gauge bucket tips as they occur
//   Activated by the magnet and reed switch in the rain gauge, attached to input D2
{
raintime = millis(); // grab current time
raininterval = raintime - rainlast; // calculate interval between this and last event

if (raininterval > 10) // ignore switch-bounce glitches less than 10mS after initial ed
{
dailyrainin += 0.011*25.4; //Each dump is 0.011" of water
rainHour[minutes] += 0.011*25.4; //Increase this minute's amount of rain
rain5m[minutes_5m] +=0.011*25.4; // increase this 5 mnts amout of rain
rainlast = raintime; // set up for next event
}
//Nirosha
//Rain or not (1 or 0)
if(rainin_5m > 0)
{
Rainindi=1;
}
if(rainin_5m == 0)
{
Rainindi=0;
}
//Niroshan
}

void wspeedIRQ()
// Activated by the magnet in the anemometer (2 ticks per rotation), attached to input D3
{
if (millis() - lastWindIRQ > 10) // Ignore switch-bounce glitches less than 10ms (142MPH
{
lastWindIRQ = millis(); //Grab the current time
windClicks++; //There is 1.492MPH for each click per second.
}

}

void setup()
{
Serial.begin(9600);
```

```
ss.begin(9600); //Begin listening to GPS over software serial at 9600. This should be the

pinMode(STAT1, OUTPUT); //Status LED Blue
pinMode(STAT2, OUTPUT); //Status LED Green

pinMode(GPS_PWRCTL, OUTPUT);
digitalWrite(GPS_PWRCTL, HIGH); //Pulling this pin low puts GPS to sleep but maintains RT

pinMode(WSPEED, INPUT_PULLUP); // input from wind meters windspeed sensor
pinMode(RAIN, INPUT_PULLUP); // input from wind meters rain gauge sensor

pinMode(REFERENCE_3V3,INPUT);
pinMode(LIGHT, INPUT);

//Configure the pressure sensor
myPressure.begin(); // Get sensor online
myPressure.setModeBarometer(); // Measure pressure in Pascals from 20 to 110 kPa
myPressure.setOversampleRate(7); // Set Oversample to the recommended 128
myPressure.enableEventFlags(); // Enable all three pressure and temp event flags

//Configure the humidity sensor
myHumidity.begin();

seconds = 0;
lastSecond = millis();

//attach external interrupt pins to IRQ functions
attachInterrupt(0, rainIRQ, FALLING);
attachInterrupt(1, wspeedIRQ, FALLING);

//turn on interrupts
interrupts();

//Serial.println("Weather Shield online!");

}

void loop()
{
//Keep track of which minute it is
if(millis() - lastSecond >= 1000)
{
digitalWrite(STAT1, HIGH); //Blink stat LED

lastSecond += 1000;
```

```
//Take a speed and direction reading every second for 2 minute average
if(++seconds_2m > 119) seconds_2m = 0;

//Calc the wind speed and direction every second for 120 second to get 2 minute average
float currentSpeed = get_wind_speed();
//float currentSpeed = random(5); //For testing
int currentDirection = get_wind_direction();
windspdavg[seconds_2m] = (int)currentSpeed;
winddiravg[seconds_2m] = currentDirection;
//if(seconds_2m % 10 == 0) displayArrays(); //For testing

//Check to see if this is a gust for the minute
if(currentSpeed > windgust_10m[minutes_10m])
{
windgust_10m[minutes_10m] = currentSpeed;
windgustdirection_10m[minutes_10m] = currentDirection;
}

//Check to see if this is a gust for the day
if(currentSpeed > windgustms)
{
windgustms = currentSpeed;
windgustdir = currentDirection;
}

if(++seconds > 59)
{
seconds = 0;

if(++minutes > 59) minutes = 0;
if(++minutes_10m > 9) minutes_10m = 0;
if(++minutes_5m > 4) minutes_5m = 0;


rainHour[minutes] = 0; //Zero out this minute's rainfall amount
rain5m[minutes_5m] = 0; //Zero out this 5 minutes' rain Niroshan
windgust_10m[minutes_10m] = 0; //Zero out this minute's gust
}

//Report all readings every second
printWeather();

digitalWrite(STAT1, LOW); //Turn off stat LED
}
```

```
smartdelay(800); //Wait 1 second, and gather GPS data
}

//While we delay for a given amount of time, gather GPS data static
void smartdelay(unsigned long ms)
{
unsigned long start = millis();
do
{
while (ss.available())
gps.encode(ss.read());
}
while (millis() - start < ms);
}


//Calculates each of the variables that wunderground is expecting
void calcWeather()
{
//Calc winddir
winddir = get_wind_direction();

//Calc windspeed
windspeedms = get_wind_speed();

//Calc windgustms
//Calc windgustdir
//Report the largest windgust today
windgustms = 0;
windgustdir = 0;

//Calc windspdms_avg2m
float temp = 0;
for(int i = 0 ; i < 120 ; i++)
temp += windspdavg[i];

temp /= 120.0;
windspdms_avg2m = temp;

//Calc winddir_avg2m
temp = 0; //Can't use winddir_avg2m because it's an int
for(int i = 0 ; i < 120 ; i++)
temp += winddiravg[i];

temp /= 120;
winddir_avg2m = temp;

//Calc windgustms_10m
```

```
//Calc windgustdir_10m
//Find the largest windgust in the last 10 minutes
windgustms_10m = 0;
windgustdir_10m = 0;
//Step through the 10 minutes
for(int i = 0; i < 10 ; i++)
{
if(windgust_10m[i] > windgustms_10m)
{
windgustms_10m = windgust_10m[i];
windgustdir_10m = windgustdirection_10m[i];
}
}

//Calc humidity
humidity = myHumidity.readHumidity();
//float temp_h = myHumidity.readTemperature();
//Serial.print(" TempH:"); //Serial.print(temp_h, 2);


//Calc tempf from pressure sensor
tempf = myPressure.readTemp();
//Serial.print(" TempP:");
//Serial.print(tempf, 2);

//Total rainfall for the day is calculated within the interrupt //Calculate
amount of rainfall for the last 60 minutes //Niroshan Stop the gathering
hourly rain fall data
rainin = 0;
for(int i = 0 ; i < 60 ; i++) //change to 60 mnts
rainin += rainHour[i];

rainin_5m = 0;
for(int i = 0 ; i < 5 ; i++) //change to 5 mnts
rainin_5m += rain5m[i];


//Calc pressure
pressure = myPressure.readPressure();

//Calc dewptf

//Calc light level
light_lvl = get_light_level();

//Calc battery level
```

```
batt_lvl = get_battery_level();

}

//Returns the voltage of the light sensor based on the 3.3V rail
//This allows us to ignore what VCC might be (an Arduino plugged into USB has VCC of 4.5 to
float get_light_level()
{
float operatingVoltage = analogRead(REFERENCE_3V3);

float lightSensor = analogRead(LIGHT);

operatingVoltage = 3.3 / operatingVoltage; //The reference voltage is 3.3V

lightSensor = operatingVoltage * lightSensor;

return(lightSensor);
}

//Returns the voltage of the raw pin based on the 3.3V rail
//This allows us to ignore what VCC might be (an Arduino plugged into USB has VCC of 4.5 to //Battery level is connected to the RAW pin on Arduino and is fed through two 5% resistors: //3.9K on the high side (R1), and 1K on the low side (R2)
float get_battery_level()
{
float operatingVoltage = analogRead(REFERENCE_3V3);

float rawVoltage = analogRead(BATT);

operatingVoltage = 3.30 / operatingVoltage; //The reference voltage is 3.3V

rawVoltage = operatingVoltage * rawVoltage; //Convert the 0 to 1023 int to actual voltage

rawVoltage *= 4.90; //(3.9k+1k)/1k - multiple BATT voltage by the voltage divider to get

return(rawVoltage);
}

//Returns the instataneous wind speed float
get_wind_speed()
{
float deltaTime = millis() - lastWindCheck; //750ms

deltaTime /= 1000.0; //Convert to seconds

float windSpeed = (float)windClicks / deltaTime; //3 / 0.750s = 4
```

```
windClicks = 0; //Reset and start watching for new wind
lastWindCheck = millis();

//windSpeed *= 1.492; //4 * 1.492 = 5.968MPH

/* Serial.println();
Serial.print("Windspeed:");
Serial.println(windSpeed);*/

return(windSpeed);
}

//Read the wind direction sensor, return heading in degrees int
get_wind_direction()
{
unsigned int adc;

adc = analogRead(WDIR); // get the current reading from the sensor

   //   The following table is ADC readings for the wind direction sensor output, sorted from
   //   Each threshold is the midpoint between adjacent headings. The output is degrees for th
   //   Note that these are not in compass degree order! See Weather Meters datasheet for more

if (adc < 380) return (113);
if (adc < 393) return (68);
if (adc < 414) return (90);
if (adc < 456) return (158);
if (adc < 508) return (135);
if (adc < 551) return (203);
if (adc < 615) return (180);
if (adc < 680) return (23);
if (adc < 746) return (45);
if (adc < 801) return (248);
if (adc < 833) return (225);
if (adc < 878) return (338);
if (adc < 913) return (0);
if (adc < 940) return (293);
if (adc < 967) return (315);
if (adc < 990) return (270);
return (-1); // error, disconnected?
}


//Prints the various variables directly to the port
//I don't like the way this function is written but Arduino doesn't support floats under sp
```

```
void printWeather()
{
calcWeather(); //Go calc all the various sensors

Serial.println();
Serial.print("$,lon=");
Serial.print(gps.location.lng(), 6);
Serial.print(",lat=");
Serial.print(gps.location.lat(), 6);
Serial.print(",altitude=");
Serial.print(gps.altitude.meters());
Serial.print(",sats=");
Serial.print(gps.satellites.value());

char sz[32];
Serial.print(",date=");
sprintf(sz, "%02d-%02d-%02d", gps.date.year(), gps.date.month(), gps.date.day());
Serial.print(sz);

Serial.print(",time=");
sprintf(sz, "%02d:%02d:%02d", gps.time.hour(), gps.time.minute(), gps.time.second()); Serial.print(sz);

Serial.print(",RTCdate=20"); sprintf(sz, "%s",
getDateDs1307(0)); Serial.print(sz);
Serial.print(",RTCtime="); sprintf(sz, "%s",
getDateDs1307(1)); Serial.print(sz);


Serial.print(",winddir=");
Serial.print(winddir);
Serial.print(",windspeedms=");
Serial.print(windspeedms, 1);
Serial.print(",windgustms=");
Serial.print(windgustms, 1);
Serial.print(",windgustdir=");
Serial.print(windgustdir);
Serial.print(",windspdms_avg2m=");
Serial.print(windspdms_avg2m, 1);
Serial.print(",winddir_avg2m=");
Serial.print(winddir_avg2m);
Serial.print(",windgustms_10m=");
Serial.print(windgustms_10m, 1);
Serial.print(",windgustdir_10m=");
Serial.print(windgustdir_10m);
Serial.print(",humidity=");
Serial.print(humidity, 1);
Serial.print(",tempc=");
Serial.print(tempf, 1);
Serial.print(",rainhourmm=");
Serial.print(rainin, 2);
Serial.print(",rain5mmm=");
Serial.print(rainin_5m);
```

```
Serial.print(",raindailymm=");
Serial.print(dailyrainin, 2);
Serial.print(",rainindicate=");
Serial.print(Rainindi,1);
Serial.print(",pressure=");
Serial.print(pressure, 2);
Serial.print(",batt_lvl=");
Serial.print(batt_lvl, 2);
Serial.print(",light_lvl=");
Serial.print(light_lvl, 2);


Serial.print(",");
Serial.println("#");

}
```