Australian Meteorological Association Inc

Monana (PWS Edition)

THE OFFICIAL PUBLICATION OF THE AUSTRALIAN METEOROLOGICAL ASSOCIATION INC

March 2021 Edition

The next PWS Edition of the Monana magazine is due to be published on the 28th May 2021. Please submit all PWS items for publication by the 14th May 2021 to:

monana@ameta.org.au

But don't forget to also send non-PWS items for inclusion in the more general April edition of Monana.

"An Investment in knowledge always pays the best interest."

— Benjamin Franklin

The February 2021 AGM/membership meeting was noteworthy as it marks the departure from using the BOM Meeting Room. Work to select the most appropriate alternate venue is still on-going. Until a new "home-base" is determined, I would ask that members be tolerant as the Committee seeks out and tries a few different alternatives. Availability, cost and location are the primary factors being considered.

The presentation by Dr. Pushan Shah from the Environmental Protection Authority South Australia provided a good insight into the operation of the Environmental Science Branch in relation to air quality monitoring. Of particular interest was the development of a low cost air quality monitor using a Plantower sensor (see the article "Simplify Weather Station Building with a 3D Printer" in the March 2020 Monana) and an Arduino (see An Introduction to the Using the Arduino" in the same issue). More details of this sensor in this edition.

The professionally developed unit highlighted one of the oft overlooked aspects of home built weather sensors. Most of the effort is not getting the sensor to work, but rather providing communications, power and mounting hardware that will pass the test of continuous exposure to the weather (and insect wildlife).

The presentation also provided an interesting way of estimating where air pollution may be coming from using wind speed and direction, in conjunction with the air quality sensor. I had never thought about using the weather station for this, so I'm going to give it a shot when I have the time.

There were audience questions about whether it was worth people getting cheap air quality sensors when they weren't calibrated like the expensive units. As with other citizen measurements that may not be done with the highest quality instruments, the real value is readings over a wide area are combined together into a network. It is often the case that even uncalibrated instruments provide valuable information about change patterns, even if individual units are not within the desired accuracy levels.

In the end, whether it is worth getting an affordable, but less than perfect, instrument depends on how interested you are in what it is measuring.

Keep Happy, Keep Safe.

Mark Little
email: president@ameta.org.au
mobile: 0434 602 091

PWS Meetings

Discussions have been held with the Port Adelaide Enfield (PAE) Council about providing a meeting room for Personal Weather Station (PWS) group meetings. Although formal approval is yet to be approved, it looks likely that the PAE Council may be able to supply a meeting room and access to some of their STEM (Science, Technology, Engineering and Maths) facilities free of cost.

Obtaining council approvals is expected to take about 4 weeks which would put that at the end of March. As I will be away April and early May, the first meeting is expected to be in the latter half of May 2021. In a radical departure from the normal member's meetings, the PWS meetings will be held on the weekend during the day.

At this stage, they are likely to be held on a Saturday Afternoon. The venue is not fixed yet, but I am hoping that the offered venue will be the Enfield Library which is just off Main North Road just before the Gepps Cross intersection. This will make it easy for people to get to the library as the traffic should be light on a Saturday afternoon. Saturday afternoon gives a wider range of people an opportunity to attend. It also has relatively close access to three (3) electronic stores which will be convenient for project development.

One of the conditions required by the PAE Council for providing access to their facilities is that we provide free access to anyone who wishes to attend, which seems perfectly reasonable considering that they are providing their facilities free to us. This does not mean that these visitors will automatically become members of the AMetA, although until the next AGM, they will be able to join for zero dollars. After that, the membership fees will return and the magazine will return to being for members only, but access to the PWS meetings will remain open to all.

One of the PWS group objectives will be to look how to select, install, maintain and interpret the readings that are obtained from commercial personal weather stations. This sort of activity is suitable for people who have an interest in having a personal weather station, but may not have any meteorological or technical experience. After all, we all use phones and computers, but how many people REALLY understand how all these things work, or even care?

The PWS group will also be looking at a variety of more technical projects with different variants of those projects depending on individual needs. For example, the same sensor may be used in different projects that have different requirements that favour different data communication solutions.

1. Cabled system such as USB or Power-Over-Ethernet (POE). Provides communication and power by cable, especially in small backyard settings.
2. High Bandwidth Wireless such as Wi-Fi. Provides high bandwidth radio communication, but requires additional power source.
3. Long Distance, Low Power system such as LoRaWAN. Low data rates, but often battery powered.

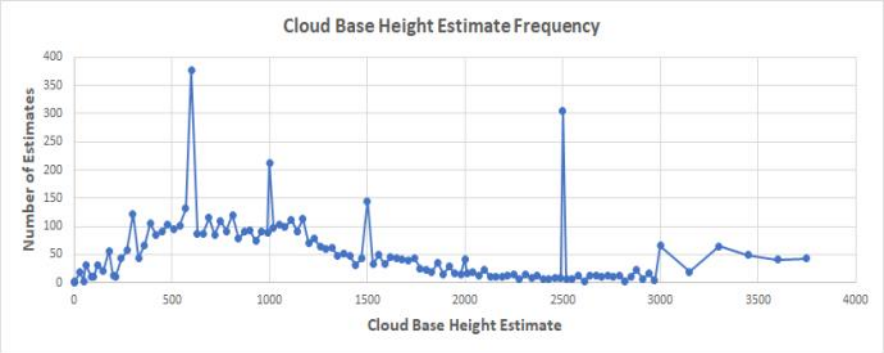
The Air Quality sensor systems provide an example how the same type of sensor may require different types of communication. For general distributed use, air quality sensors are often paired with a LoRaWAN communication system and run from battery power which may last over 12 months in some cases.

On the other hand, LoRaWAN is restricted to transmitting small messages every few minutes and this would not be suitable for a sensor that is gathering data nearly continuously for projects such as using the wind direction and speed with the air quality reading to estimate the location of pollution sources. In this case, a cabled system may be more suitable as it can provide more power to the sensor. If power is available, a Wi-Fi system obviates the need to run any communication cables (serial, USB or Ethernet).

Where possible, AMetA projects are created in a modular form so that people can more easily customise the hardware and software to meet their own objectives. **Please keep an eye out for further details about the start of the meetings in May.**

Cloud Base Investigation Conclusion

In the September 2020 edition of Monana there was some investigation into the observed cloud heights at the Parafield Airport to get an indication of the profile of the cloud base heights in the area. Plotting the frequency of observation against the cloud base heights provided a rather interesting graph (replicated below).



While it was expected that there would a curve from group level up to a few thousand metres (BOM archives cloud base height in metres, not feet as used in aviation). However, there were quite significant peaks at a few heights. The database queries that were used to generate this graph were double and triple checked to make sure that they weren't the cause, and the process was repeated for the Adelaide and Edinburgh airports. Similar results were obtained.

As discussed previously, it seemed that this sort of result was indicative of more than one type of observation being blended together. Initial enquiries to a person who used to work in the Bureau seemed to indicate that cloud base heights were done entirely using an inclinometer instrument. If this was the case, then this implied that there was a fault, but the fact that similar results were being obtained from all of the tested airports seemed to rule that out and that anomaly was somehow systematic.

To put this anomaly to bed, the Bureau Of Meteorology was contacted to see if they could provide an explanation of what was being seen. Not unexpectedly, given that COVID hit during this process, there was a bit of delay after the initial automated enquiry. However, one did come.

Just to let you know this is still being looked at. The team who look after the AWS and ceilometer equipment can not see any issues, so have forwarded it to on the teams who look after the downstream comms systems to see if there's an issue there. One thing that is worth mentioning is that ceilometer cloud heights are reported in feet but archived in meters, so be aware that there has been a conversion from feet to meters in those cloud heights.

Things went quiet from the BOM for awhile, but things were ticking away in the background at the BOM. Eventually, the mystery was solved when this email arrived.

The cloud values provided in the Latest Weather Observations JSON files come from three different cloud observation sources. Most of the time the observations come from the ceilometer, but when there is a weather observer present (as often happens at Adelaide Airport) then the cloud values come from the manual 30-minute cloud observation that the observer makes, and not from the ceilometer. In addition, at synoptic times (0000, 0300, 0600, 0900 LCT etc) the cloud observations actually come from a synoptic observation which reports the cloud height as a height range. It is the synoptic observations which are generating the anomalies you found given those messages report standard height ranges.

So, apart from when the cloud base is reported as one of those anomalous standard values, the data will provide a cloud base that can be reliably used to check a local cloud base estimate.

The following part of the email indicated was also interesting (underlining added). The effort taken to look into this matter was taken seriously by the BOM, so never think that your personal weather interests don't really have any value and will never of interest to others.

The Latest Weather Observations products are a real-time product, with the cloud values only intended to provide an indication of the cloud height/amount at the time. They really shouldn't be used for official research/analysis, especially given they are a mash of different data sources with limitations as indicated above. <u>The above has been passed through as an improvement that should be made to the product however.</u>

Replacing My Sky Camera

By Mark Little

As quite a few members may know, I run a sky camera that looks south from my garden shed. It is often a source of comment, since it looks like the weather station has a wild tilt, but that is only because of the camera mounting and because the camera lens is a bit of a fisheye lens. The image to the left is a snapshot from that camera.

Unfortunately, although that camera lasted many years, it has gone to the electronic graveyard. It was time to find a replacement. Looking on the Internet, there is a myriad of different cameras, with a price range to match. The question is what performance should the camera have, and how much should it cost?

For sky watching purposes, the first choice is likely to be a security camera that is designed to work both in the daylight and at night. After all, on average, the sky is dark for about 50% of the time over a year. The other thing is that security cameras are often designed to be mounted outside in the weather which will usually indicate that it is built the IP66 standard. The next most important thing is to consider is what you want the camera to do.

There are essentially two main types of camera—an analogue camera that is usually designed to connect to a DVR (Digital Video Recorder). These cameras mostly don't have the same level of "smarts" as the digital cameras that use a Wi-Fi or Ethernet cable interface. The digital cameras can have a wide range of capabilities that allow the user to control the operation of the camera. Some cameras will even upload their images directly to the Internet, complete with added captions.

The new camera I selected this time wasn't a cylindrical "bullet" camera and used a different mounting system (but luckily the mounting holes were the same). I wasn't familiar with this model and it said that it required a smart phone to use, but given the (low) price, I thought it was worth taking a risk.

The camera can communicate via 2.4GHz & 5 GHz Wi-Fi as well as an Ethernet Cable. It also has the advantage of complying with the ONVIF interface standard. This standard is an open industry standard that provides and promotes standardised interfaces for IP-based physical security items, including cameras.

As I don't own a smart phone, I attempted to connect to the camera using the latest version of Firefox and Microsoft Edge. Much to my surprise, the camera returned a warning that the version of those browsers was too new to be compatible with the camera's firmware which is identified as being late 2020. A quick search of the Internet indicated that if I install the Apple Safari browser, the error message should go away.

The camera was connected to the router and I was able to now access the camera. Using the camera's menus the camera was set up to fit into my local network. The only issue that arose is that the camera gets its time from the Internet, but the camera cannot connect to the Internet. Fortunately, the computer that controls the camera can access the Internet, so the time is applied to the image by that computer, rather than by the camera itself.

In the end, it turns out that the smartphone application "ICSee" is only required if you want to easily set up the camera via Wi-Fi, because it sets up a Wi-Fi connection to the camera without the camera knowing the Wi-Fi router password. My tablet is an old Windows unit, so I could not load the app to test it out. However, one perk of being the acting Treasurer is that I have the AMetA's Android tablet. It worked a treat and just so that you know what is going on, the camera talks to you giving its status. In case I forgot to tell you, the camera can work as a two-way intercom as well as a camera.

The script that I use to control my sky camera needed two lines of code altered to control the new camera and so the sky camera is operating again. I don't think the night performance is not quite as good as my old camera, but that is probably understandable given that I purchased two new cameras for less than half the price of my old camera.

If you are interested in installing your own sky camera, but don't know where to start, contact me and I will see if I can assist you.

I'm Setting Up A Sandbox!

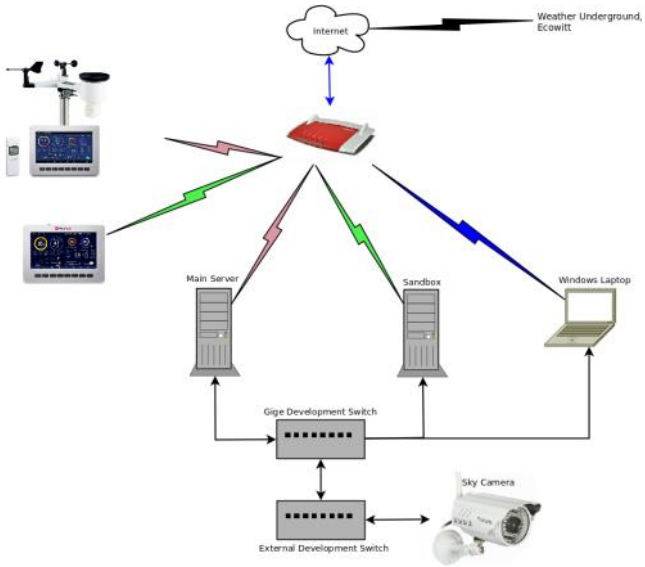
by Mark Little

No, I don't mean this nifty boat shaped sand box (click image to see more about that sandbox). I mean a software testing environment that allows me to test software without affecting the operation of my normal systems. I could have set up a Virtual Machine (VM) on my main computer, but I wanted to be able to power down my main computer, which would not have been possible if the sandbox was on a virtual machine on that computer.

The system that I have set up is probably too much for most people, but fear not, there may be free on-line alternatives that you can use to test your code and its functionality, as long as you aren't trying to control hardware. They are also excellent ways to look at a wide variety of computer languages without having to install the language on your own computer.

For example, the CodePad site lets you enter code for a dozen languages including C, C++, D, PHP, Perl, Plan text, Python, Ruby, Scheme, TCL, O Caml Hashkell & Lua. Not enough languages? Well, ideone goes further with over 60 languages that can be used. No, that's wasn't a typo! There really are (at least) that many computer languages. What?!? You've never heard of Scala, Unlamda, Elixir or Fantom? Well, that's okay, because neither had I. If you want to look at a selection of on-line sandboxes, I suggest that you go to "Top 10 Best Websites To Test Your Code Online – Reliable Sandbox Tools" to look at some of the available options.

Those sandbox sites are designed for general use by an indeterminately large number of users, while my sandbox is designed to suit my specific interests, which is Personal Weather Stations (and generally playing with computers). So what do I want my sandbox to do? Firstly, I want to be able to develop software that uses the data from my PWS without affecting the current operation of my system. Because I want to deal with interconnected programs running in real time, the general sandbox sites aren't suitable for testing those interactions.



In the partial network diagram to the left, the pink, green and blue flashes represent domestic Wi-Fi connections, while the black, one is the connection to the WOW, Weather Underground and the (password-protected) Ecowitt weather sites.

On the development side, there is a Gigabit Ethernet Local Area Network (LAN) which allows for much faster data exchanges than via Wi-Fi.

However, what initially appears odd is that there appears to be one and a half weather stations. That is, the second weather station has no sensors. Just like the 60+ programming languages, this is not a mistake. Our houseboat has a weather station, but we do not leave it running while we are away. The photo below should give you an indication of why the unattended operation of that unit is not considered reliable.

So what I do is take home the console and turn it on. Because the weather station console only reads the sensors, both consoles will lock onto the same sensors. As you can see from the diagram, the weather station talks to the main server, while the console from the boat talks to the sandbox. This means that I can play with the weather station software on the sandbox without losing any readings which are being received by the main server.

To test the sandbox properly, it had to be made available on the Internet, but I only have one Internet address (Brigadoon.power.on.net) which is already used by my website. The internet address for my website is "http://brigadoon.power.on.net". The address consists of two main parts. The first "http://" tells the browser that it should use the Hypertext Transfer Protocol, although "https://" (Hypertext Transfer Protocol Secure) is also used to provide a more secure connection using encrypted transmissions.

The browser uses HTTP or HTTPS to connect to the server at the location "Brigadoon.power.on.net". What is not obvious from the addresses that we enter is that a connection also requires a "port" number. If one takes the analogy of a house, the "brigadoon.power.on.net" is the address of the house, and the port is the door that is unlocked and ready to use. In the case of HTTP that number is usually port 80 and for HTTPS it is port 443. In fact, the browser assumes that if it sees HTTP, it will use port 80, and port 443 when it sees HTTPS, so these port numbers do not need to be included each time a website is accessed.

However, it is possible to (1) set a web server to look at a port other than 80 or 443; and (2) tell the web browser to attempt to connect to another port. A common alternate port for HTTP is 8080, and 8443 for HTTPS.

So, my router that sits on the Internet is configured so that my normal server receives connection requests that go to port 80 and port 443, but the sandbox webserver is configured to respond to port 8080. So when I want to connect to the sandbox from the Internet, the address is "http://brigadoon.power.on.net:8080". Before you think that you might see what is happening on that sandbox, it doesn't normally operate, so the chances that you will be able to connect to it are quite slim. Not only that, I might be using a different port than the 8080 port, because I can. ☺

The sandbox is overkill for many small home projects, but it allows a copy of larger and more complex new developments to undergo full regression testing. To show that the new software did not introduce or re-introduce any faults the sandbox has an up-to-date copy of all of the main server's software and data. This allows for easy comparisons to see that the unaltered parts of the system function as it did before the new software was installed.

It also means that the testing of the new software will take place in a system with the exactly (or nearly exactly) same configuration as the main server. To be honest, I don't tend to do regression testing for most changes, because I'm just running my system for my own amusement and if it doesn't work properly until I fix it, too bad! If I was running a system where people were paying for a service, then full testing on the sandbox before installing on the live system would be essential for keeping the customers happy.

In earlier days, upgrading the Linux operating system could be a bit hit and miss, so I tended to try out the upgrades on a sandbox. In the early days, the worst case scenario of where the operating system would not start or became unstable was not uncommon. Fortunately, this is no longer the case, and I just jump in. Although I'm not completely foolish, I do a backup first.

Raspberry Pi Pico

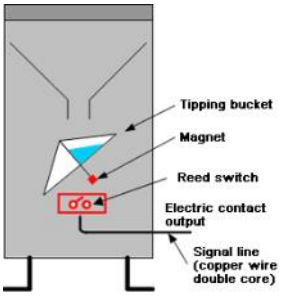
The new Raspberry Pi Pi costs less than $6 from local suppliers has 2MBytes of flash memory and 26 multi-function ports, with three ports being Analogue to Digital Converter (ADC) enabled. Compared to the Arduino Uno which is much less capable and costing between $10.60 (on-line) and up to $36 locally, it is a device that may shake up the market.

Currently, it is lacking the depth of ready-to-use software and hardware hats and bonnets available for the Arduino family, but I think that will be remedied in a fairly short period of time.

Because the circuit board has a similar footprint as the 40pin Dual In Line (DIL) integrated circuit, it will be easy to integrate into custom projects just using 0.1" pitch development boards, some thing would require a prototyping hat for the Arduino. I must get one of these to play with!

An Introduction to Interrupts

This article is designed for people with a basic understanding of programming an Arduino using the Arduino IDE and the objective is to show how to use an interrupt on the Arduino. Most meteorological sensors look after themselves and all you have to do is read the sensor to get the temperature, humidity, pressure or whatever. You can read the sensor at any time and you will get a reading.



One sensor where this is not so easy is with the tipping bucket (or tipping spoon) rain gauge. As shown by the diagram to the left, the tipping bucket/spoon rain gauge works by a container filling and tipping over to empty. As it tips, a magnet passes a reed switch which sends a pulse to the computer.

The problem is that the pulse is short and can come at any time, even when the computer is busy doing something else. What needs to happen is that the computer responds to the pulse whenever it happens. This is where an interrupt comes into play. It causes the computer (an Arduino, in this example) to interrupt what it is doing and to respond to the pulse. Once the pulse has been attended to, the program will continue with what it was doing before.

While this seems complicated, the C/C++ language handles much of the work for you. On the other hand, you still have to do your bit to keep the code that occurs during an interrupt small, to prevent it from slowing down or causing other problems with your main program.

First we need to look at the hardware. One side of the rain gauge switch is normally connected to earth and the other side of the switch is connected to a pin on the Arduino. The pin we need to select is a pin that is capable of handling interrupt processing. Looking at the Arduino documentation, pin 3 can be used with interrupts, so that is the pin that will be used.

```
#define RAINFALL_PIN        3
```

To detect the state of the rain gauge switch connected to RAINFALL_PIN (pin 3), the default state of the pin must be set to an input pin with the Pull-Up enable. The pull-up in the chip forces the pin to HIGH (Logic 1). As a result, the pin will be dragged to LOW (Logic 0) when the gauge switch closes and connects the pin to Earth (LOW, Logic 0).

```
pinMode( RAINFALL_PIN, INPUT_PULLUP );
```

We also need a variable that will hold the count of the bucket tips. It needs to be a big number to catch a lot of bucket/spoon tips, so we will use a 32 bit (4-byte) number to hold the count. Since there will be no negative counts, the number can be unsigned, effectively doubling the number of counts that it can hold. Using a 32 bit unsigned number can be identified by the identifier “uint32_t”.

Since this number will be used in the interrupt, it can change without the main routine knowing it. We need to identify this so the main program will always read a fresh copy when it uses that value. This is identified by the identifier “volatile”. Putting those two items together gives:

```
volatile uint32_t      rainfall_total;
```

Because the rainfall_total can change at any time, we need to make a different variable where we can catch the rainfall_total and use it in our normal program flow. It should be the same size as rainfall total, but not volatile.

```
uint32_t       total_rainfall;
```

We also need to identify a function to call when the interrupt occurs. It is identified in the normal way.

```
void IncrementRainFallInterrupt();
```

Now that we have defined the name of the interrupt processing routine, we need to tell the Arduino that we are going to use an interrupt on that pin, and that we want the interrupt to be triggered when the switch closes and pulls the pin low.

```
attachInterrupt( digitalPinToInterrupt(RAINFALL_PIN), IncrementRainfallInterrupt, FALLING );
```

Now we need to decide what the interrupt function should do when triggered. For simplicity in this example, we assume that the mechanical reed switch contacts do not bounce when they contact. If they did, we would need to worry about “debounce” hardware or software to ensure that we only got one pulse when the switch operated.

As discussed before we need to keep the amount and complexity of the code in an interrupt as small as possible, so all we do is increment the rainfall counter.

```
void IncrementRainfallCounter()
{
 rainfall_total++;
}
```

Also discussed previously, we need to be able to read the rainfall_total during a period when it can't be changed via an interrupt so that we aren't halfway through reading it when it changes. To ensure that this does not happen, we first turn off the interrupts, then read the value and after that enable interrupts again. If we do that fast enough, we are unlikely to lose pulses from the rain gauge.

```
cli();    // Turn off Interrupts
rainfall_total = total_rainfall;       // Safely read the value
sei();   // Turn On Interrupts
```

It is now to put together a program where you can try this out for yourself.

```
#define RAINFALL_PIN  3                  // Arduino Pin to use for Interrupt
volatile uint32_t rainfall_total;        // Volatile rainfall total used in Interrupt
uint32_t          total_rainfall;        // rainfall total used outside of the Interrupt
uint32_t          comparison_value;      // Comparison value to ensure output only prints when rainfall changes

// Interrupt Routine
void IncrementRainfallInterrupt()
{
 rainfall_total++;                       // Increment the Rainfall total because the bucket/spoon has tipped
}

void setup()
{
 rainfall_total = 0;                     // Initialise Rainfall Total

 total_rainfall = 0;                     // And the Non-Interrupt Value

 comparison_value = 999;                 // Set to anything but Initial Value of total_rainfall;

 Serial.begin(9600);                     // Set Serial Port at 9600 baud

 pinMode( RAINFALL_PIN, INPUT_PULLUP);      // Set Rainfall pin as an unput that is pulled HIGH

 attachInterrupt( digitalPinToInterrupt(RAINFALL_PIN), IncrementRainfallInterrupt, FALLING); // Attach Interrupt

 Serial.println("Starting Rain Gauge Program");         // Print a Start Up Message

}

void loop()
{
 // Get the Current Rainfall Value
 cli();                                  // Prevent Interrupts from messing up read
 total_rainfall = rainfall_total;        // Make a non-volatile copy
 sei();                                  // Enable Interrupts again

 if (total_rainfall != comparison_value)  // Only output the rainfall if it has changed since last time
 {
  Serial.print("Rainfall Total is "); Serial.print(total_rainfall); Serial.println(" tip(s).");      // Print the message
  comparison_value = total_rainfall;      // Set the Comparison valier to the current rainfall
 }
 delay(5000);                            //  Delay so that you can test that the interrupt catches the pulses while asleep
}
```

After you have entered this code onto an Arduino, connect a tipping buck rain gauge (or just a push button switch) between pins 3 (interrupt pin) and the GND (ground) pin.

When the program starts, it should print two lines. The first “Starting the Rain Gauge Program””, followed by a second line “Rainfall total is 0 tip(s)”. Then tip the bucket/spoon or press the switch once and wait. Because of the delay it may be up to five (5) seconds before the program outputs “Rainfall total is 1 tip(s)”. Please remember that contact bounce may result in more than one pulse being generated.

Once you have satisfied yourself that program is detecting a pulse, tip the bucket/spoon or operate the switch multiple times. The message output by the program should show that the program has counted each pulse.

If you are getting contact bounce, the simplest solution is to apply a filter to reduce the effects of contact bounce. Rather than going through the mechanisms involved and what circuit to apply to reduce its effects, I suggest that you visit The Lab Book Pages for additional information. When looking at the explanation of Switch Bounce, it is useful to remember that in the “simple switch pull-up circuit”, the pull-up resistor R1 is provided by the PULLUP setting of the Arduino.

If you are looking for a more practical implementation of a rain gauge, the GroundWeather project will provide an example of implementing a tipping bucket rain gauge in conjunction with a range of other sensors. Although you may not be looking for a solution that uses a USB connection to the Arduino, it should give you some ideas of how to read a variety of sensors. My site is currently under redevelopment, so expect to be some bits to be missing at the moment, but the source code will be there.

An Arduino Clock

While the previous example attached an interrupt to an external event, this interrupt example uses an internal timer which runs off the Arduino’s internal Counter Timer to produce an interrupt at 1 second intervals. The interrupt routine then updates the seconds, minutes, hours of the day, the date, the month and the year. While the clock will continue to run on its own, the accuracy of the Arduino’s clock means that it will periodically need to be synced to a higher quality time source.

To get the Counter Timer Circuit to generate 1 second pulses from the clock, a series of internal hardware dividers need to be set up. While you can work these settings out from the data sheets, there is a web site which does the calculations for you and provides the codes to set the values, giving the following:

```
cli(); // Stop Interrupts
TCCR1A = 0; // Set Entire TCCR1A Register to 0
TCCR1B = 0; // Same for TCCR1B
TCNT1 = 0; // Initialize Counter to 0

// TIMER 1 for interrupt frequency 1 Hz: set compare match register for 1 Hz increments
OCR1A = 62499;      // = 16000000 / (256 * 1) - 1 (must be <65536)

// turn on CTC mode
TCCR1B |= (1 << WGM12); // Do Not Alter Other Bits

// Set CS12, CS11 and CS10 bits for 256 prescaler
TCCR1B |= (1 << CS12) | (0 << CS11) | (0 << CS10); // (CS11 and CS10 shown for clarity only)
// enable timer compare interrupt
TIMSK1 |= (1 << OCIE1A); // Do Not Alter Other Bits

sei(); // Allow Interrupts Again
```

If you don’t understand what all that means, it doesn’t really matter at this stage, because the person who set up the webpage that calculated the values does. You can look up the datasheets later is you need to know.

Like the previous interrupt example, we need to have a routine that will be called when the interrupt is triggered. However, in this case, we don’t attach the interrupt to a pin, but rather we attach it to an address that the timer will call when it is triggered. We do this by having an ISR (Interrupt Service Routine) for the internal timer device. In this case we are using TImer1, so the name of the routine is:

```
ISR( TIMER1_COMPA_vect )
{
 // Insert Clock routine here
}
```

Each time a second passes, TIMER1 will generate an interrupt. This means that whenever the routine is called it knows to increment the number of seconds. From then on, it is simply a chain. For example, if the seconds are greater than 59, reset the seconds to zero and increment the minutes. If the minutes are greater than 59, set the minutes to 0 and increment the hours. If the hours are greater than 23, set the hours to 0 and increment the days and so on.

Determining when the days over-flow is a bit more tricky as months may have a different number of days, February being double trouble because of leap years. The old rhyme "*Thirty days hath September, April, June and November. All the rest have thirty-one, Excepting February alone, And that has twenty-eight days clear And twenty-nine in each leap year*." comes in handy when testing whether is the end of the month, but the leap years do complicate it a bit.

However this also helps to make it easy to know if it is a leap year "*If the year is divisible by 100, but not 400, then it is a leap year, otherwise if it is divisible by 4, it is a leap year*."

It is also possible to calculate the day of the week from the date. This is not something that I knew how to do, so I looked it up on the Internet. The following formula is named Zeller's Rule after a Reverend Zeller who developed it. One of the important things to note is that this process shifts the start of the year to March (3rd month), so that the troublesome month of February (with its potential leap year day) is the last month of the calculation year and doesn't affect the calculation of other days.

Assuming that the routine will only be used in a Real Time Clock, it can also be assumed that the dates being used will only be in the 21st Century, so the computer program can be optimised for that period. Below is the code used to calculate the day of the week for a date. All of the values are integers, so all calculations will be truncated to whole numbers.

```
if ( ( Month == 1 ) || ( Month == 2 ) )
{
  Month += 10;
  Year--;
}
else
{
  Month -= 2;
}
Year -= 2000;

DayOfWeek = ((Day + ((13 * Month  - 1) / 5) + Year + (Year/4) - 35) % 7 );
```

The first part in orange adjusts the start of the year to March as required by the calculation, while the last part calculates the day of the week as a number. The green line gets the year in the current century for the routine optimised for the 21st century.

Let's confirm this routine actually works by picking a date—"04-November-2021". Using January as 1 and December as 12, we adjust the month to 9 and the year to 21. Then plugging that into the formula, we get:

```
DayOfWeek = ((4 + ((13*9 - 1)/5) + 21 + (21/4) - 35)%7) ;
```

Simplifying, we get DayOfWeek = (4 + 23 + 21 + 5 - 35)%7 giving DayOfWeek = 4. So, if Sunday = 0 and Saturday = 6, then my birthday will be on a Thursday this year. Luckily, the calendar on the computer agrees with this calculation. Now let's try "01-January-2022" to confirm that this works for a date in January.

```
DayOfWeek = ((1 + (13*11 - 1)/5) + 21 + 5 – 35) % 7 ) ;
```

We get DayOfWeek = 6, and therefore a Saturday. Again, my calendar agrees with this calculation.

P.S. If you try this calculation, don't forget that integer arithmetic drops any faction part of a division and "%" returns the remainder of an integer division. That is, 43/6 gives 7 as it answer; and 43%6 gives 1 as its answer.