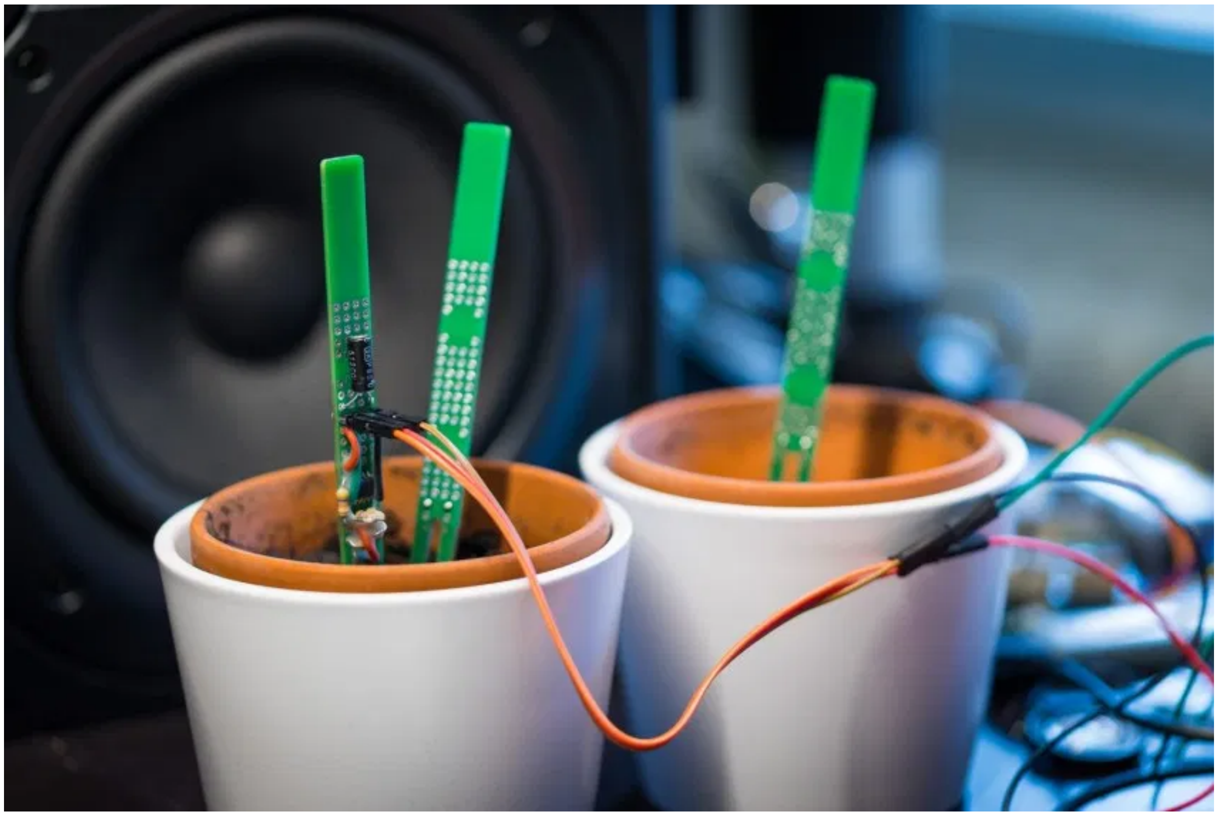
LUCKY RESISTOR

Menu

HOME LEARN PROJECTS LIBRARIES APPLICATIONS

ABOUT

Menu



How to Design a Cheap Plant

Watering Sensor (Part 1)

Posted on 2017-02-08

In this article I will talk about how I designed a cheap plant watering sensor. My goal is some kind of meta tutorial, where you can see the steps involved from the initial idea to the final sensor. If you ever planed to create a own device, I hope this article give you some inspiration to start your own project soon.

Why a Plant Watering Sensor?

I have a couple of plants in flowerpots and this plants not only like some light, they also need water from time to time. Watering this plants is something I often forget, with sad results. There are ready made solutions for this, but I have some objections with all of them. To be clear: There are really smart products out there – it is absolutely nothing wrong with them. It is just as I like to build my own fan controller, I like to build my own plant watering sensor in my very own fashion.

Here a list of already existing projects and devices I own or checked out:

- Chirp! The plant watering alarm
  Very close to my idea, but I prefer a visual indicator.
- Koubachi Wi-Fi Plant Sensor
  Too expensive, wi-fi, now retired anyway.
- Plant Link
  Too expensive, wi-fi.

Step 1: Define the Expectations and Goals

After deciding to create a own plant watering sensor, I spend some days to think about my expectations and goals for this sensor.

For my personal case, I like to put a small sensor in each flowerpot. There will be five and more pots, therefore that number of sensors are required. A single sensor should be really cheap, so I can distribute as many of them as I like. Battery life should be at least one year, better two years. I collected all this thoughts into the following list:

- **Cheap**: Ideally less than €5 including the PCB.
- **Visual Signal**: A flashing LED, simple to notice but easy to ignore.
- **Simple**: Easy and simple to build with few components.
- **Beautiful**: It should ideally look like a decoration.
- **Long Battery Life**: The battery should last at least 1 year or longer.
- **Small**: The sensor should be almost invisible from far away.
- **Reliable**: The measurement should be reliable and the sensor must not corrode or degrade over years.
- **Safe**: The sensor must be safe for the plant and environment.
- **Low Battery Indicator**: The sensor should detect if the battery is at end of life and signal this.

Step 2: Do Research

The key part of the sensor is the part which detects how much water is in the soil of the flowerpot. This is something of great interest for whole industries and it is clear endless research was done on this topic. A very good start is searching for “soil moisture measurement” using Google Scholar.

“Soil Moisture Measurement” in Google Scholar

In most cases it is faster and simpler to read scientific papers for a given topic, especially if the talk about some very advanced brand new method to do some measurement. The scientist will explain all available “old” techniques in detail and talk about all the expected problems and disadvantages.

There are approximate three acknowledged methods, described in the following sections.

Measure the Resistance

Stick two electrodes into the soil and measure the resistance between the two.

- Good: Simplest and cheapest method available.
- Bad: The electrodes will corrode quickly because they have to touch the soil.
- Bad: Salt and minerals greatly influence the result.

Capacitive Measurement

Stick two flat plates into the soil, insulate them and measure how much electrons can be stored in the soil. The water and soil will act like the electrolyte in a capacitor. This is the same principle used in capacitive touch displays.

- Good: The electrodes/plates are completely insulated and do not corrode.
- Good: Salt and minerals will only very slightly influence the measurement.
- Bad: Close large water bags (humans) will influence the measurement. 🙂

High Frequency and Time Domain Reflectometry

Here you put one or two (insulated) electrodes into the soil and use a high frequency (>1MHz or even >100MHz) signal. Different elements will act like a filter on the signal. If you choose the correct frequency you can easily just check how much water filters the signals. The shift and reflection of the signal will let you measure other properties of the structure of the ground.

- Good: Best and most precise measurement of soil moisture.
- Good: With time domain reflectometry only one electrode required.
- Good: External influences can be easily detected and filtered.
- Bad: The high frequency requires fast and expensive components.
- Bad: The signal has to be shielded to protect other RF applications.

Conclusion

Measuring the resistance between two electrodes has almost only disadvantages, I can easily rule this out.

While the high frequency and time domain reflectometry sound very interesting, it will make the whole project expensive. Not very expensive, but I can not keep things under my targeted cost limit.

Therefore the **capacitive measurement** seems like a good compromise between reliability and cost.

Step 3: Do More Research

Capacitive measurement is the way to go. Now I need to know how to reliable measure a very small electrical capacity. Using two electrodes 60mm × 3mm the capacity of water soaked soil will be around 250pF. Dry soil will have less capacity, down to 50pF.

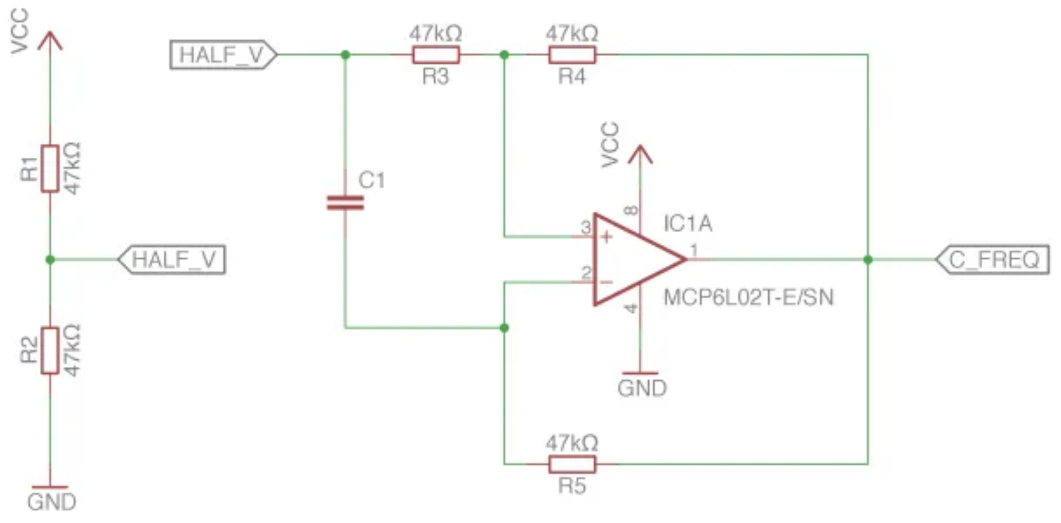
This is also a very well researched topic, and there are many methods to measure the capacity of something. For the sensor, I have some important constraints for the method of measurement:

- Low Power: Should require minimal power.
- Simple Input: The measurement should generate a simple value which can be read by any cheap microcontroller.
- Fast: The measurement has to be fast – faster measurements will need less power.

Find a Method of Measrement

After checking various methods I found a great one in the book The Art of Electronics. It is a simple oscillator using two op-amps. The shown circuit uses the first op-amp to generate a saw tooth signal charging and discharging the capacitor and a second op-amp as Schmitt trigger to create a rectangle signal.

I simplified this oscillator, because every microcontroller will automatically rectify the input signal, therefore this part is not required. A single op-amp as oscillator is sufficient for the task.



For the values of the resistors I assumed 250pF capacity and tried to minimise the quiescient current of the circuit. The voltage divider always draws current, so I had to use large resistor values here which still provide enough current to load the capacitor C1, which is actually the sensor part which is stuck in the soil of the flower pot.

Calculate some Key Values

Using the Electronic Toolbox app I did some calculations:

This is a good value, the power should only last until the measurement is done and the LED flashes. So this should ideally never last longer than 500ms.

To get an impression of the expected frequency of the oscillator, I just calculate the change/discharge time of the assumed capacitor of 250pF.

These are just approximations, the real values will be probably much lower, because each wire will have resistance and capacitance which will add to these values. Also the op-amp will slow down the frequency because of the switching time. The only important information I get from these values is: The resulting frequency will be in the kHz range, which will be perfect for slow microcontrollers to process.

Step 4: Decisions

To continue with the project I had to make some decisions. They are not final at this point, but they will define the direction for further tests. Ideally I can live with these decisions, but if I get stuck I always can rethink these constraints.

- Method of Measurement: Capacitive measurement.
- Used Circuit: Use a op-amp based oscillator.
- Battery: Try to work with the 2-3V from a CR2032 lithium ion coin cell.
- Size: Use hand solderable SMD components.

Step 5: Physically Verify the Method of Measurement

The Op-Amps

To do some initial tests I checked the endless list of op-amps. The cheaper ones are the slower ones so I ordered a range of op-amp chips:

- Microchip MCP6L02T-E/SN (Dual, 1MHz)
- Microchip MC33202DG (Dual, 2.2MHz)
- Microchip MCP6282-E/SN (Dual, 5MHz)

I ordered the dual versions to have a second op-amp to do further tests if required. For all these op-amps there are various package options, including single op-amp versions.

The Electrodes

Basically you need two insulated plates with the electrolyte (soil) between them. The plates itself need to be properly insulated against each other as well. I quickly created a board in EAGLE with a small prototyping area where I can solder the op-amp chip and some wires and resistors. I did this completely in the board editor, without connected schema.

As you can see, I added a plate on the front and backside on each leg of the sensor. Both are connected with a via hole. The final board will look like this:

Ordering the Prototype

I ordered a couple of this prototype boards to do initial tests. For this prototype I chose Eurocircuits, principally because of their great PCB visualisation and check tools. These tools are really helpful if you really quickly create some board. It will warn you about anything which could be a problem and you get a very nice preview of the final result.

Create the First Test Setup

After receiving the boards, I created the first test setup using two flowerpots. I filled one flowerpot with water soaked soil and the other one with dry soil.

First I soldered the cheapest and slowest (1MHz) op-amp on a SOIC/DIL adapter and built the circuit on a breadboard. Then I soldered two wires to two of the prototype boards and tested if the oscillator works as expected.

Surprisingly it worked perfectly with the initial resistor values I used – no change was required. The frequency was in the expected kHz range.

Now I soldered the oscillator circuit on one board, again with the cheapest and slowest (1MHz) op-amp I bought for tests. I added also a capacitor to stabilise the power supply and a header to connect the test circuit with power and the oscilloscope to see the results.

Verify the Results

The signal is easily visible using the oscilloscope:

This is the measurement in really wet soil. The frequency is ~28kHz which is in the lower range of the expected frequencies. If I stick the probe into dry soil, the signal I get this result:

The frequency will go up to ~125kHz and it is clear, this is the upper end of the possible frequency with the used circuit and op-amp. It is a perfect range to use this as input for a microcontroller.

If the probe is removed from the soil, oscillation stops completely. This is a nice side effect, this way the MCU can easily detect this state as well.

Step 6: Create a Rough Design of the Final Device

Create a Block Diagram

Now, after successfully verified the key component of the final device, it is time to create a very rough design of the final device. The best way to do this is a simple block diagram:

This is the second version of the diagram. In the first version I tried to simple use a low-power MCU for power control and timing, but this is not as power efficient as thought. It also requires the MCU to keep an internal timer running to wake from sleep – this further increases the power consumption.

To solve the power problem, I added a low-power timer. This is a chip which is optimised to use as few as possible current and enable a power regulator or MOSFET to power the actual circuit for a short time. There are plenty different chips of this kind on the marked, and they are cheap as well.

A button is required because each plant is different and there should be an easy way to select the level where the sensor signals an alarm.

Step 7: Search for the Right Components

Searching the right components for the device is not a simple task – each component has its own constraints. The first step therefore is to make a list of these constraints for all components:

- Constraints for all components:
  - Powered from 3V CR2032 coin-cell, therefore all components have to work in a voltage range between 3V down to 2.1V. The cut-off voltage of the coin-cell is 2V, but better to stop at 2.1V.
  - The components should use very low power, ideally way below 1mA.
  - A easy solderable SMD package should be available, like SOIC or SOT-23 and similar.
  - The components should have a good quality but should be cheap to keep everything below the total cost limit.
- Battery Holder
  - Small/flat
  - Works with CR2032
- Low-Power System Timer
  - Support the timer interval of 10 seconds.
  - Can drive a P-Channel MOSFET to control the power of the device.
  - Can be manually triggered using a push button.
- MOSFET
  - P-Channel
  - Can handle the required current for the oscillator, MCU and LED.
- Button
  - Easy to press and feel the press because there is likely no other feedback.

- Op-Amp for Oscillator
  - Should work in the expected frequency range. We already verified this before.
- LED
  - Very bright at a low current to create a visible flash.
- Microcontroller
  - 2 digital inputs: One from the button and one from the oscillator.
  - 1 digital output for the signal LED – ideally it can drive the LED directly.
  - 1 ADC input to monitor the battery voltage, ideally with an internal voltage reference.
  - Internal counter which can be driven from the oscillator input to measure the frequency.
  - Clock minimum 1MHz to be able to process the oscillator input.
  - Internal clock source to keep things simple.
  - Reliable works in the voltage range from 3V down to 2.1V.
  - Minimum 1Kbyte flash memory for the firmware.
  - Some kind of internal EEPROM wich can be programmed down to 2.1V. Flash usually can not be programmed at this low voltage levels.
  - Ideally can be reprogrammed in assembled state – so later firmware updates are possible without unsoldering the MCU.
  - Fast start-up time to keep power consumption as low as possible.

I usually start with the most complex part of the device, which is the microcontroller in this case. The most complex part usually influences many other elements of the circuit. Defining this part will therefore influence the search for the other components as well.

The Microcontroller

A great help are the filters on the websites of the various suppliers. For example on the Mouser website I just selected the 8-bit Microcontroller category:

8-bit Microcontrollers at Mouser Europe

First I change the sorting to lowest price first. Next I select all supply voltages which match my requirements and filter to SMD components. I also select all with some defined ADC resolution and matching package sizes. The final list has only 337 matches left.

I can now sequentially check the microcontrollers in the list, starting with the cheapest ones. In this case the list starts with some ATtiny10 in SOT-23-5 packages – this would actually be nice, but they have no built-in EEPROM.

Next in the list are the ATtiny13A MCUs in SOIC-8 packages, this component is a perfect match. There are a number of subtle package differences and variants, but the SSH variant is perfect for our case. At the time I write this, it is 0.47€ for 100 units – this is a very reasonable price.

The Low-Power System Timer

I use the same principle for the system timer. First I select the category "All Products > Semiconductors > Clock & Timer ICs > Timers & Support Products". Here I change the sorting to lowest price first. Next I filter for a minimum voltage below 2V, SMT and the suitable package sizes.

The first match is the Texas Instruments TPL5110DDCR. This system timer perfectly matches our requirements, and with 0.58€ for 100 units it has a reasonable price.

Key Components

I repeat this steps with all key components and get this list of components:

- Battery Holder: Linx BAT-HLD-001
- System Timer: Texas Instruments TPL5110DDCR
- Microcontroller: Microchip ATTINY13A-SSHR
- Op-Amp: Microchip MCP6L01T-E/OT
- LED: Avago HLMP-K155
- Power MOSFET: Fairchild Semiconductor BSS84
- Button: ALPS SKQGAKE010

To manage the component list, I use the BOM tool of the Octopart service. It will easily let you collect components for a project and you always have the availability of these components in view.

Step 8: Design the Complete Circuit Diagram

Before I order the components, I first create a complete circuit diagram to verify if all components will work together. This way I also can check if I have all required resistors, capacitors and similar available to create the prototype on a breadboard.

I get the following circuit diagram (click to enlarge):

Step 9: Order Components for a Working Prototype

To verify if the device will work as expected, I order a small number of all components to create a simple prototype on a breadboard. I also check if I have the right SMD adapters for all components. Here I found Adafruit has a wide range of adapters for all different kind of SMD packages.

Usually I order about five pieces from each component. While testing, components can die, so it is nice to have some spares left. Also SMD components do not need much space and it is simple to keep all of them organised in binders.

Step 10: Build the Prototype

After the components arrive I solder them on adapter boards and build a complete working prototype on a breadboard:

Before I build the whole circuit as shown in the image, I test each component individually. It is easier to test the individual components first and you are sure they work as expected if there are any problems in the complete circuit.

Fir this and later tests an oscilloscope is a must. You can monitor individual signals and compare this measurements with values you get in the microcontroller. An oscilloscope is also very important to check the timing of the signals in your circuit.

Step 11: Write a Test Firmware

With the prototype in place, it is time to write a test firmware. Here I use the Atmel Studio IDE and the Atmel ICE debugger. The firmware for the chip is relatively simple. Every time the MCU is powered on, it will execute these steps:

1. Initialise the hardware/registers
2. Check the battery level.
    - If the battery voltage is lower than 2.1V, double flash and exit.
3. Check if the button is pressed.
    - >1s: Set the target value to the lowest value.
    - >3s: Set the target value to the current value.
    - Write the new value to EEPROM.
    - Triple flash to as feedback for the write.
4. Read the current oscillator frequency.
    - Faster than stored value: Exit.
    - Slower than stored value: Single flash and exit.

This is a very simple firmware and it fits easily into the 1kByte flash memory.

I will talk more about writing the firmware in the fourth part of the tutorial. The code I wrote at this point was very simple, all done in the `main.cpp` file.

Epilogue

Thank you for reading this article! I hope it was insightful and will help you with your own projects.

Continue reading part 2 of the article.

While I enumerated the steps, this is obviously no simple linear process. If you get stuck at some point, or some component does not meet your expectations, you will have to iterate over some of the steps until you get the desired result.

In a few weeks I will publish the second part, which will focus on the work from the prototype to the final device.

Have fun!

Share with:

5 More

Like this:

Loading...

10 thoughts on "How to Design a Cheap Plant Watering Sensor (Part 1)"

daryyan says:
2017-02-12 at 09:13

Hi! This is a great project! I was looking (and even purchased) Chirp! but would also prefer visual indication with LED or something rather than having a sound. Can't wait for the part two. I was wondering, is this project open source?

Loading...

Reply

Lucky Resistor says:
2017-02-12 at 10:57

I did not decide yet, but most likely I will publish the design as open hardware. The work on this project is still in progress, so first I will have to do extensive testing before I see if everything works as expected.

Loading...

Reply

---

anand s says:
2017-02-19 at 02:21

I really like your step by step tutorial by calculated number anyway what simulator program do you use ? Its better to include some picture when you measure using oscilloscope just my suggest

Loading...

Reply

Lucky Resistor says:
2017-02-19 at 12:16

Thank you very much for your feedback!

For this project, I did not use any simulator. I just calculated key values to get some approximations – then built the prototype to verify this values. I tried many times to create a simulation of circuits in advance (with different simulators), but honestly I failed every time. The difference between the simulation and reality was always too great to be of any use. I assume, I just lack the proper mathematical knowledge to setup everything properly. It would be interesting to see, if someone can simulate the oscillator circuit and compare the results of the simulation with the actual measured values from the oscilloscope.

What kind of pictures using the oscilloscope would you like to see? Where the probes are placed, or what part of the measurement process would you like to see?

Loading...

Reply

Pingback: Plant Watering Sensor, from Prototype to Project Walkthrough « Adafruit Industries – Makers, hackers, artists, designers and engineers!

Alan Lord says:
2017-03-07 at 16:33

Really well written piece. Thanks! I have read up to part 4 and enjoyed them very much. I am building a somewhat similar project but my soil detectors will be in a Polytunnel and will send data over Wifi (ESP8266).

Loading...

Reply

Lucky Resistor says:
2017-03-07 at 16:49

Thank you for the compliment! Glad to hear you liked the article series.

Loading...

Reply

Brendan says:
2017-05-01 at 07:11

Hello Alan,

Do you have any write up of your project so far?

Loading...

Reply

Brendan says:
2017-05-01 at 07:25

Disregard my last post.

Loading...

Reply

Pingback: Using an oscillator to measure soil moisture.... - Brendan's Bits

Leave a Reply

Enter your comment here...

This site uses Akismet to reduce spam. Learn how your comment data is processed.

TOP POSTS & PAGES

Plastic Cup Meltdown - Working with Epoxy

Accessing the SD Card (2)

How and Why to use Namespaces

How and Why to Avoid Preprocessor Macros

Accessing the SD Card (Part 3)

Event-based Firmware (Part 1/2)

Units of Measurements for Safe C++ Variables

Accessing the SD Card

Data Logger

Learn C++

FOLLOW ON TWITTER

Follow

SUBSCRIBE TO BLOG VIA EMAIL

Enter your email address to subscribe to this blog and receive notifications of new posts by email.

Join 82 other subscribers

Email Address

Subscribe

RECENT POSTS

Excellent Cases from Distrelec's RND Brand

Always-On Project

Stencils from OSH Stencils Arrived

Small Boards for the Swiss-Precision Timer

How to Write Custom Snowflake Patterns

Programming the Snowflake Decoration

Candlelight Effect

How to Wire the Snowflake Decoration

CATEGORIES

Common (29)

Fun (5)

How and Why (10)

Improve your Code (17)

Learn (19)

Projects (59)

Recommendations (5)

Review (2)

Software (5)