

## How to recreate “Geocache on the Rocks” (GC3NEYY)

By: fogg



One of the ways of designing fun geocaches is to make people interact with the geocache. For example, you could use microcontrollers, digital sensors and actuators to create little gadgets that the geocachers have to play with before they get the next clue (coordinates or code). One of the more popular ways of creating these gadgets is by using the [Arduino](#) computing platform. A basic knowledge of computing, hardware and programming is helpful to get started in this project.

These instructions will show you how to create a geocache similar to [Geocache on the Rocks](#). This is a multcache, where at the first stage you have to lower the temperature of the geocache to a certain degree in order to get the coordinates for the final stage.



## **Supplies:**

Basic knowledge of C++ programming and hardware building  
Arduino board  
Temperature sensor DS18B20  
ATtiny84 or similar  
LED display  
Preform Tube or similar  
Various wiring  
Soldering iron  
Battery

\*You may need other parts depending on your geocache's specific configuration.

## **How to Build:**

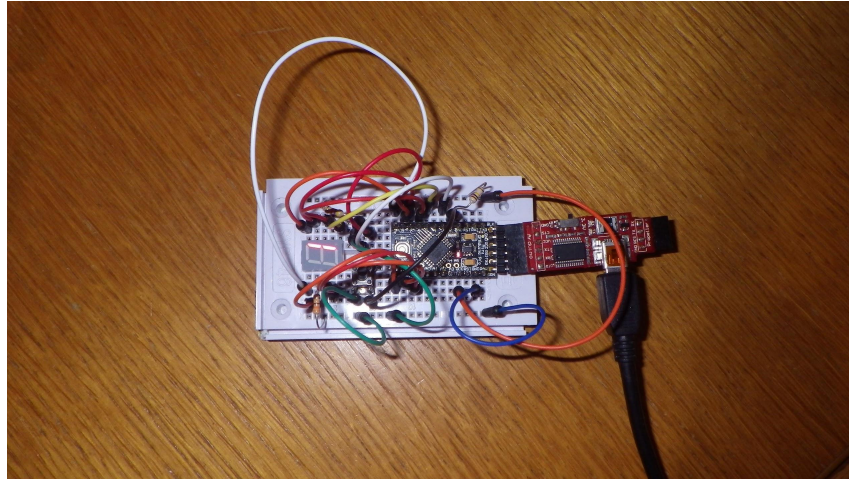
### **1. Begin With a Crazy Idea**

It always starts with a crazy idea, such as: "Let us send the geocachers to the top of the hill before they get the coordinates" or "the geocachers have to cool down the gadget to the freezing point before they get the clue." Then you have to research to see how you can make that idea come to life. If Arduino is a method to realize your idea, research whether there are sensors that you can use to detect whether the geocacher actually did what you asked for. In our cases, temperature sensors would do the trick.

### **2. But will it work?**

The first step is prototyping. This will help you get a basic idea of whether or not your idea will actually work. A good choice is to begin with the *Open Source/Open Hardware* Arduino platform. There are many different types of Arduinos, but an Arduino Pro Mini is a good choice since it fits easily on a prototyping breadboard. However, all versions of the Arduino will do. The Arduino comes with a programming environment that supports the programming language C++ and makes programming it a piece of cake. In fact, you can easily find a variety of code libraries that will help interface with many different sensors.

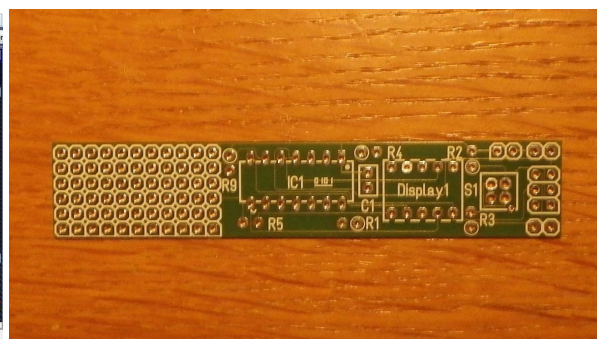
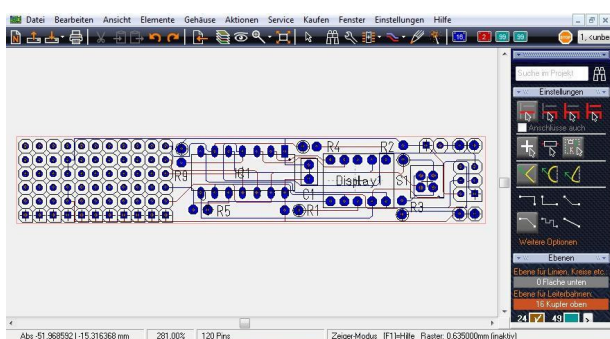
For our uses, we have chosen the temperature sensor DS18B20. There is an OneWire Arduino library which helps you with the nitty-gritty details of talking with this sensor. Using all this, you can set up a first prototype on a breadboard in less than 30 minutes and check it out. With another 30 minutes or so, you may be able to add a seven-segment LED display as a means to output the clue.



### 3. Honey, I shrunk the Arduino

While the Arduino is a nice tool for prototyping, it is not the ideal platform for creating a physical computing geocache. It's too large and wastes too much energy. In effect, it can't be used in a battery-based system that is supposed to run on the same battery for years. If you use a large enough enclosure for your gadget and ask the geocacher to bring a battery, then things might work out. In our case, though, we wanted to squeeze everything in a small tube. In order to accomplish this, a smaller, cheaper and less powerful microcontroller (ATtiny84) was used. This is also [supported by the Arduino programming environment](#).

If you want to power your gadget with a 3.6V Lithium battery with an AA form factor (which may last up to 10 years!), the printed circuit board fitting into the larger preform tubes should not be larger than 3.35"x0.62". Fitting all the components on such a board means that you should design your own printed circuit board. This is, in fact, is less of a problem that one might think. There are a number of free hobbyist versions of programs around (e.g., Eagle or Target 3000) which one can use for small projects.



You can then send the design files to a printer circuit board (PCB) producer. The PCBs

are usually created within a few weeks.

After receiving them, you can just solder the electronic components to the board. The whole step 3 will probably take a couple weeks, mainly because you have to wait for the PCBs. However, in the meantime you can work on the software.

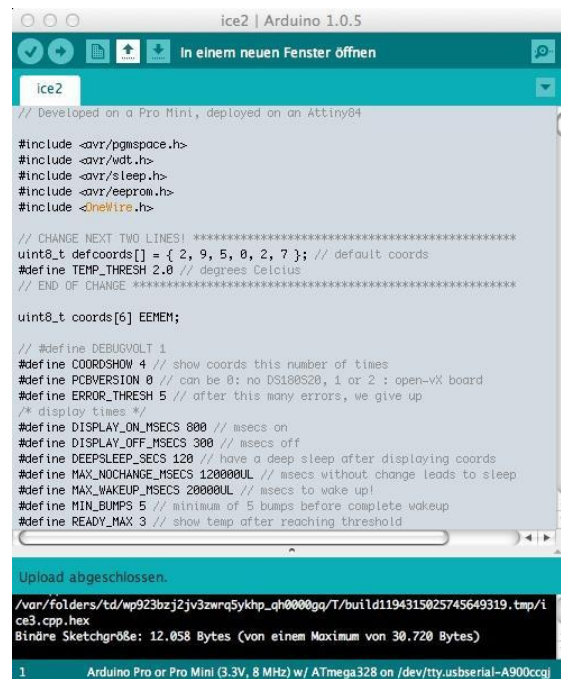


#### 4. The Hard(ware) part is over

With the hardware ready, the rest should be easy—one would think. However, the software developed in step 2 is probably not ready to control the gadget without a problem. The next step is debugging and refining. Be sure to carefully analyze what situations can come up in the field and how to save as much energy as possible.

The main tool for the design here is the notion of a *finite state machine*. That means that you enumerate the possible *states* your gadget might be in, such as *sleeping*, *waking up*, *measuring*, *showing the coordinates* or *low battery state*. As a next step, you describe what the gadget should do while in each particular state and how it changes states. With that, you have a pretty good idea of what the gadget will do and how to translate this into a C++ program. I prefer to test the program initially using the original Arduino platform because you can get debugging messages through the serial interface, which you do not have on the ATtinys. And if everything works, I switch to the ATtiny, making small changes to the program if necessary.





```
ice2 | Arduino 1.0.5
In einem neuen Fenster öffnen

ice2
// Developed on a Pro Mini, deployed on an Attiny84

#include <avr/pgmspace.h>
#include <avr/wdt.h>
#include <avr/sleep.h>
#include <avr/eeprom.h>
#include <OneWire.h>

// CHANGE NEXT TWO LINES! *****
uint8_t defcoords[] = { 2, 9, 5, 0, 2, 7 }; // default coords
#define TEMP_THRESH 2.0 // degrees Celcius
// END OF CHANGE *****

uint8_t coords[6] EEMEM;

// #define DEBUGVOLT 1
#define COORDSHOW 4 // show coords this number of times
#define PCBVERSION 0 // can be 0: no DS18B20, 1 or 2 : open-vX board
#define ERROR_THRESH 5 // after this many errors, we give up
/* display times */
#define DISPLAY_ON_MSECS 800 // msec on
#define DISPLAY_OFF_MSECS 300 // msec off
#define DEEPSLEEP_SECS 120 // have a deep sleep after displaying coords
#define MAX_NOCHANGE_MSECS 120000UL // msec without change leads to sleep
#define MAX_WAKEUP_MSECS 20000UL // msec to wake up!
#define MIN_BUMPS 5 // minimum of 5 bumps before complete wakeup
#define READY_MAX 3 // show temp after reaching threshold

Upload abgeschlossen.
/var/folders/td/wp923bzj2jv3zwrq5ykhq_gh00000gq/T/build1194315025745649319.tmp/ice3.cpp.hex
Binäre Sketchgröße: 12.058 Bytes (von einem Maximum von 30.720 Bytes)
1 Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega328 on /dev/tty.usbserial-A900ccqj
```

## 5. Made for eternity

Geocaches are definitely not made for eternity, but they should last for a long time with proper maintenance. In order for a physical computing geocache to be sustainable, electrical power and energy consumption are key factors. With the described design, i.e. Lithium batteries, very low-energy processors with a quiescent current of around 2  $\mu$ A and a maximum current of 2mAh per visit, the cache can easily last for 10 years and 500 visits.

Another key factor is the robustness of the software. Good software will ensure that the gadget runs without software updates for 10 years without a glitch. To do this, test your software extensively and make sure your software can recover from unforeseen circumstances. It's a good idea to ask others to play around with the geocache before you deploy it. You can't imagine what they will do to your little gadget.

However, even with a good design and testing, it may be necessary to correct errors in the field. That's why it's a good idea to design them in a way that even after deployment you can change the software.