Method for Auto-Restarting GoTenna Mesh Units

Disclaimer:
I am not an electronics engineer, but rather a basic hobbyist who has been messing with electronics for a long time (since the days of using discrete transistors and well-before Arduino). This project involves some rather fine soldering and general tearing-apart of electronic devices. And since at least one of these devices, the GoTenna Mesh (GTM) unit, has a lithium-polymer battery in it, be very careful and learn about what you can and can't do with lithium cells. They don't like being mistreated. I'd suggest CAREFULLY removing it before working on the power-button part of the unit. And when/if you do so, be very careful about making sure the battery wires don't touch anything conductive, like each other, or anything else metal. Directly shorting-out a lithium battery can cause it to burn/puff/smoke/explode. And that's NOT good.

Aside from that, I am not responsible if you fry anything, including your nice GTM unit.

Our goal here is to make a mechanism to self-restart our GTM unit if it ever loses power. We can't always avoid the power-loss situation, but we CAN make it come on again when power is restored.

This is what we need to accomplish this feat...

1. a mechanism to electronically push the power button on the GTM (GoTenna Mesh) device
2. a sequence of button-pushes that will get it to do what we want, and
3. we need to figure out what we want it to do.

Item 3 is easy... we want to turn the GTM unit on, and put it into relay mode. Easy, right?

The basic problem we're trying to solve is that the device, when deployed in an unmanned location, can sometimes lose power and turn itself off. This occurs when, for example, solar power is lost and the internal battery has lost its charge. Once the device turns itself off, there's no easy way to turn it back on. And when we do, it won't be in the desired relay-mode state.

We can solve this through electronic means by cutting into the power button circuitry and connecting an opto-isolator/optocoupler across the button pins. This chip takes a simple logic signal and essentially connects it's two output pins together like a switch would. We will supply the logic signal from an Arduino microcontroller, because they are cheap, small, easy to obtain, and easy to program.

It's also simpler if we can read the blinking light signals on the GTM to determine if it's on or what state it is in. So I have devised a simple photocell circuit that will let the Arduino sense these light signals to make intelligent decisions.

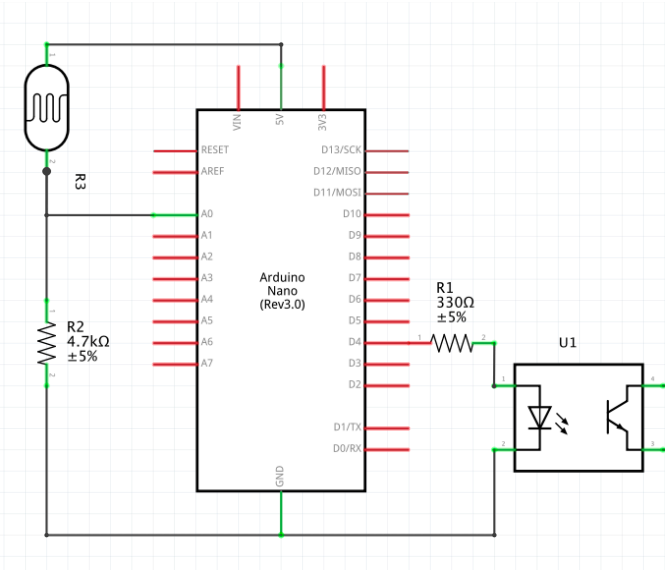
Parts:

- Arduino microcontroller... preferably an Arduino nano or pro-micro, due to small size (and get a 5V model... the 3.3V models will probably work, but 5V is more standard)… if you're bold, use a pro-mini
- a 330 ohm resistor – this will act as a basic current limiting resistor for the input side of our optocoupler, which is basically just an LED encased in plastic... and LED's need current limiting
- optocoupler chip... I used an A817V that I had lying around, but most any will probably work… some have 4 pins, some have 6… they'll probably all work, as long as you pay attention to the pins you need to use. In a second built, I used a 4N35 optocoupler, because I got a pile of them cheap on Amazon.
- small photocell
- a 4.7K ohm resistor to pair with the photocell (to make a voltage divider that the Arduino can read)

For a quick test, you might also need a little LED circuit, which means you'll need an LED, another 330ohm resistor, and a battery. It's explained more below.

Instructions:

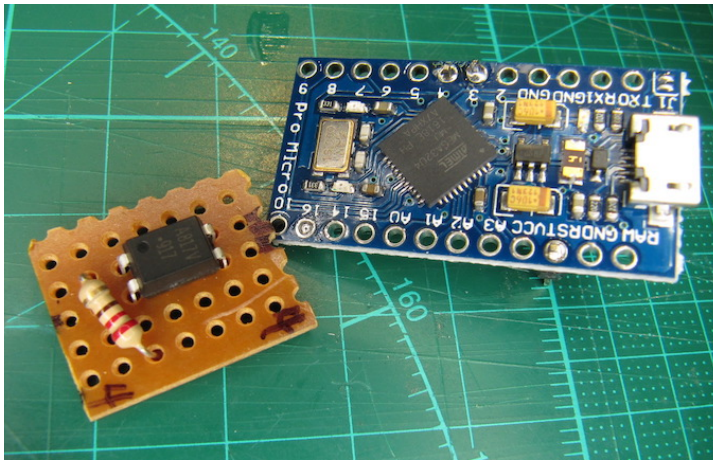
Use your electronics ninja skills to connect the components together per the basic schematic shown here:



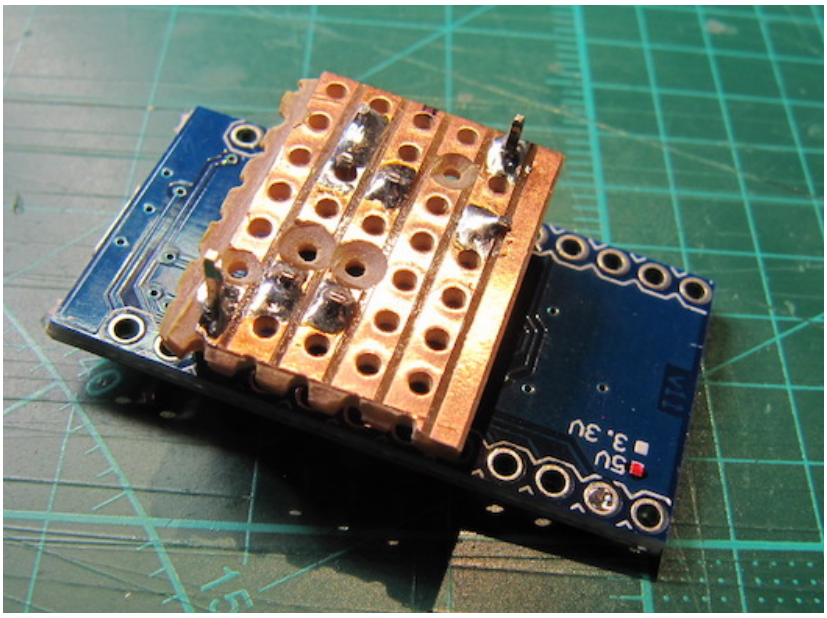
This diagram does not show the incoming power connections, which would come from a battery connected to Vin and GND on the Arduino. Use a 5 or 6 V battery (i.e. 4 aa cells or a 5V USB power bank).

The left side of the photocell circuit that detects light from the LED's on the GTM unit, and the right side of the diagram is the optocoupler circuit that will do the virtual button-pushing for us. Please note the Arduino pins they connect to. If you connect them to different pins than shown above, you will need to change the Arduino code to compensate.

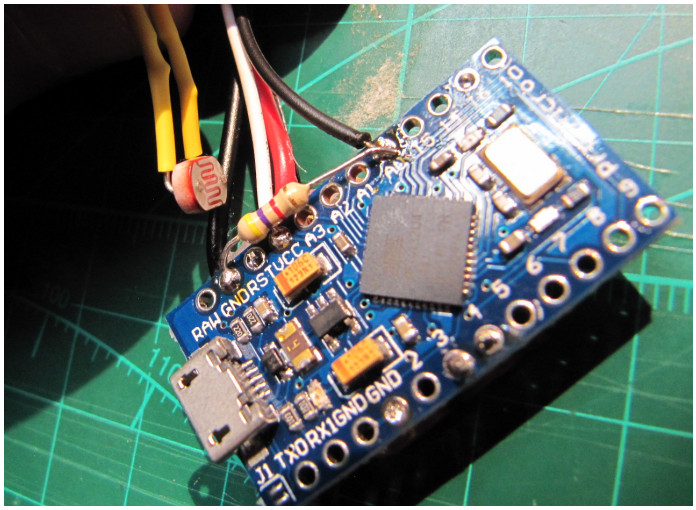
You can use a breadboard for building this circuit, but for a more permanent installation, you can make a custom printed circuit board (pcb), use a perfboard-style pcb, stripboard, or any other technique you're familiar with. I used a stripboard for the resistor and optocoupler, and ended up with this (and I used an Arduino pro micro as seen in the photo... a cheap Chinese clone model):



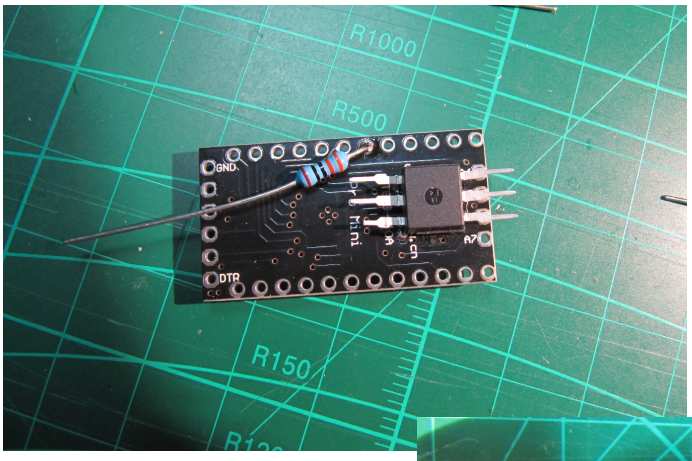
… then I mounted it on the back of the Arduino with a few connector pins and got this:

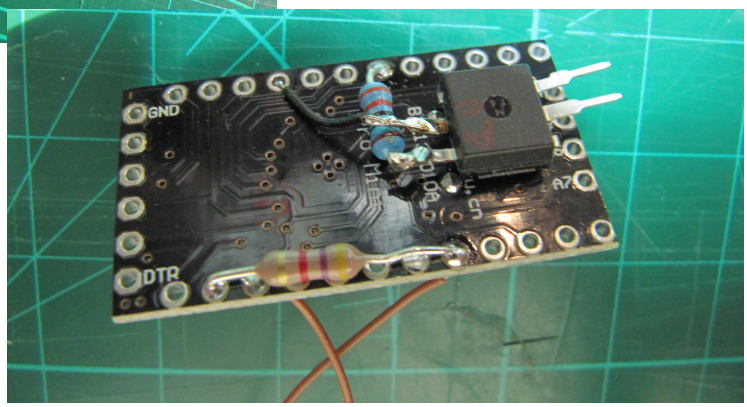


Later, after I realized I needed the photocell mechanism as well, I just added the resistor and extended photocell wires directly onto the Arduino board, since it fit nicely:

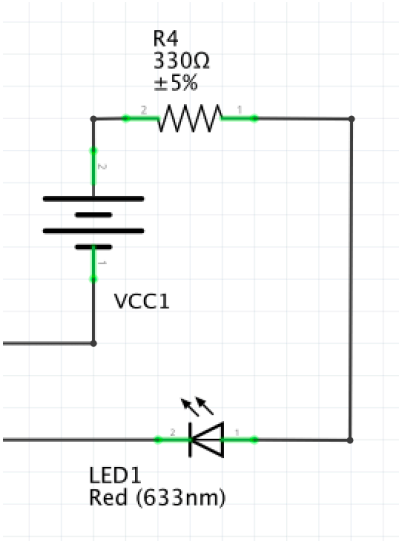


It was then that a realized that *all* of the parts could easily go onto the back of the Arduino board easily, so in a second build, I superglued the optocoupler upside down on the bottom of the Arduino and connected the other parts to it as needed (this is also when I changed to using a 4N35 optocoupler, which has 6 pins instead of 4… but you only use 4 of them.). Here are a few pics to hint at how well that worked:

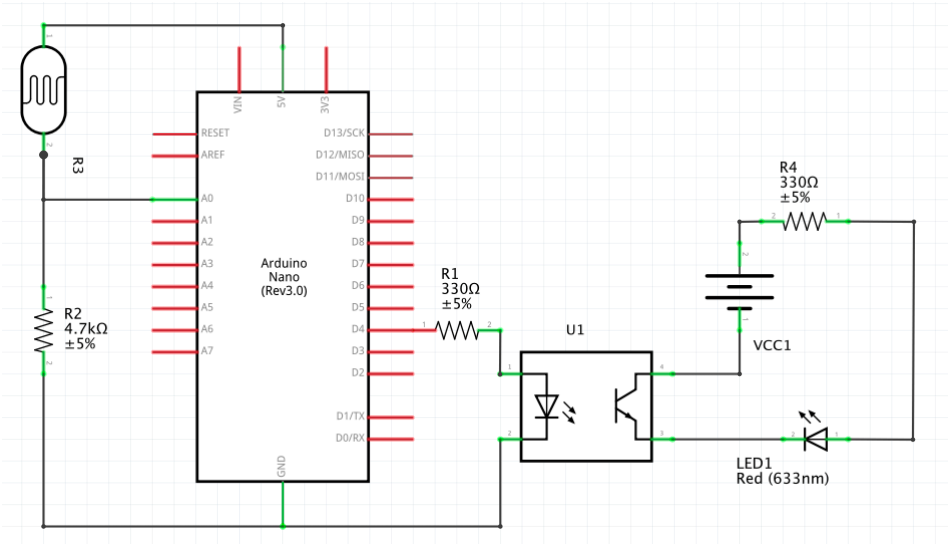




However you decide to construct it, once the circuit is together, you'll need to test it before connecting it to your GTM unit. This is best done with a simple LED circuit like the following:



To test it, load the standard Arduino BLINK program into the Arduino and connect the LED test circuit above to the output pins of your optocoupler. You're basically putting together a test rig that looks like this schematic:



The Arduino standard BLINK program (modified to use digital pin 4):

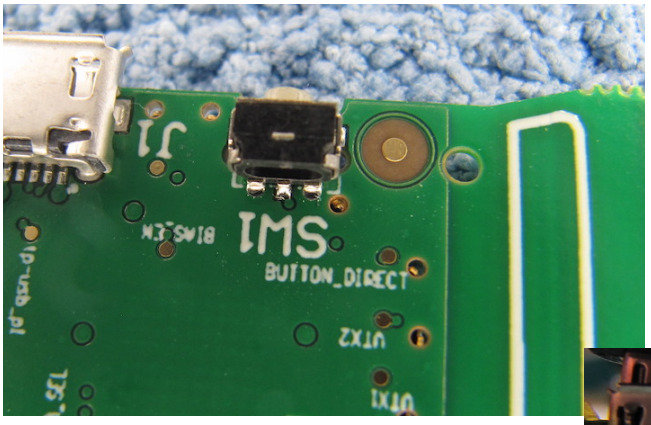
```
void setup() {
// initialize digital pin 4 as an output.
pinMode(4, OUTPUT);
}

void loop() {
digitalWrite(4, HIGH);
delay(1000);   // wait for a second
digitalWrite(4, LOW);
delay(1000);          // wait for a second
}
```

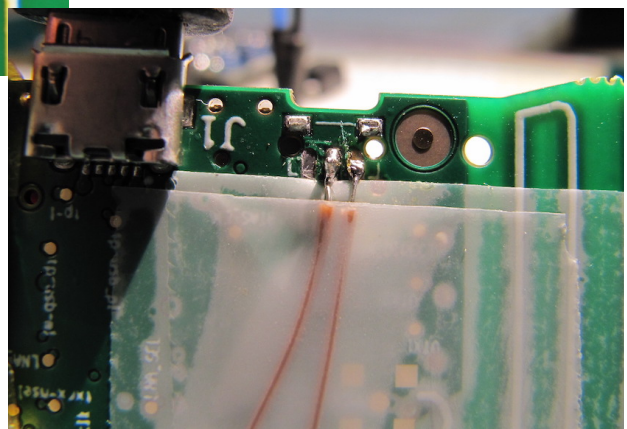
When you power up the Arduino, you should see the test-circuit LED blinking as if it were directly connected to the Arduino itself. Please note that the LED circuit is a completely separate circuit and is not electrically connected to the Arduino circuit at all... that's what an optocoupler does for us. The input and output sides (see the schematic to see what I mean) are in the same little chip case, but they are not connected to each other. The only thing moving between them is light.

If that works, we can now connect the optocoupler pins to our GTM power button connections. Of course, this means breaking open your GTM unit and delicately desoldering the pushbutton. I just yanked it off with some needle-nose pliers, but that sloppy approach peels off the copper from the board, so soldering the new wires on was a hassle. And these solder joints are tiny enough that it's a bit of a hassle anyway. It's hard to solder them without creating a solder bridge between the connections, so use care and very small amounts of solder.

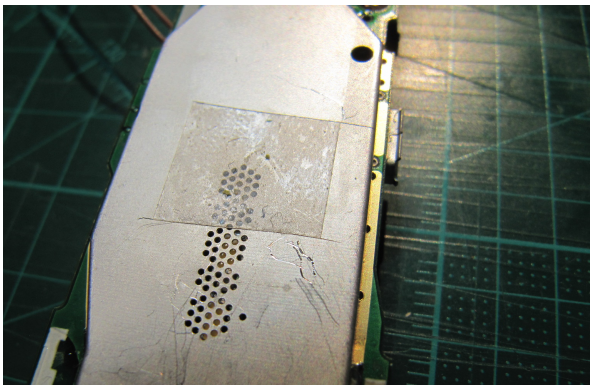
The pushbutton on the GTM connects with 3 pins/pads on the pcb.

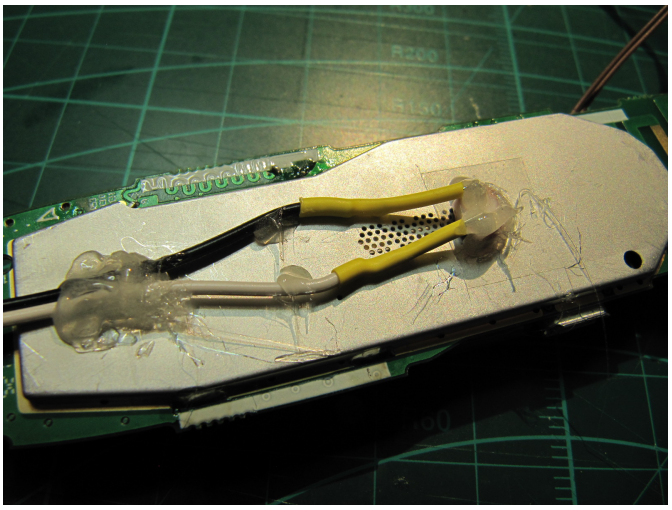


It appears that the two outer pins are both connected to ground, so we really only need to connect our optocoupler to the center pin and one outer pin... it doesn't matter which outer pin we use. I soldered very small gauge wire-wrap type of wires to these pads and connected them to the optocoupler.



Next, you'll need to plop your photocell onto the LED section on the GTM unit and glue it there somehow. I used hot glue, but we should probably consider if that will hold up outdoors if we put this whole mess into the hot sun somewhere. Use whatever you need, but also beware that the surface of the photocell is conductive, so put a piece of scotch tape over the metal case on the GTM in the small area where the photocell will attach. But leave as many little holes on the GTM shield open as you can... they are likely there for ventilation purposes.





Once that is complete... you're done! Now, you can load the Arduino program at the end of this listing and it should turn your GTM device on and off for you.

The code is on the last page of this document. You should be able to cut and paste it into your Arduino editor, but I will provide it as a separate text file as well on the GTM forums.

Use-Case Scenario:

You have a nice solar-powered GTM setup in a remote location. You have both the internal GTM battery and an external battery available to provide power. The external battery is charged by the solar panel and, in turn, keeps the GTM's internal battery charged. If it gets cloudy for awhile, the external battery provides a period of charge for the whole system.

BUT, it gets cloudy for a few days in a row, and the external battery's charge drains off. Eventually, the internal battery charge also drains and your GTM turns itself off.

THEN, the sun comes out again, charges the external battery, and in turn, the internal battery. But the GTM remains in an OFF state.

With this Arduino circuit in place, also connected to the external battery for power, when the power is available again, the Arduino will come on and start its program. The program simply issues the necessary button presses to start the GTM unit and put it into relay mode. If the GTM does not yet have enough charge to start, nothing happens... no problem.

The Arduino circuit will sleep for an hour (configurable) and then issue the same series of button presses if it appears the GTM is still off. Eventually, the GTM will have enough power to come on again, and subsequent button presses issued by the Arduino will put it into relay mode.

The downsides:

- the Arduino uses some power... not much, but more than not having it. As such, it will drain your power slightly faster than not having it. I will figure out the power-saving code needed to make it use as little as possible while sleeping.
- extra complexity
- if you haven't used Arduinos or done electronics before, it will be a hassle… so find a friend who does this stuff to help

The upside:

- you can stay at home, instead of running out to your remote location and pushing the power-button when you need to turn your GTM back on.

Button Logic in the GTM:

It appears that the GTM is aware of 3 kinds of button presses...

- momentary press … typically under 1 sec
- 1 second press … use for powering the unit on, but does nothing if it's already on
- long-press... approx 4 seconds... used to turn the unit off

In the Arduino code, I call these quickpress, secpress, and longpress, respectively. I also made a threepress function which does 3 quickpresses with short delays in between.

So, the algorithm to start the GTM unit goes like this:

quickpress to test whether unit is on or not. If it is, lights will blink and the photocell will detect the blinking. If it's not powered-on, it won't blink.

If unit is on, the nothing else will happen, and the Arduino will sleep for a period of time before checking again (default time is 1 hour).

If unit is off, the Arduino will fire-off the startup sequence, which is this:

1.secpress to start unit
2.wait 3 seconds for it to come on fully and settle
3.3 quickpress to get it to go to relay mode
4.delay 2 seconds
5.3 quickpress to verify going into relay mode
6.go to sleep and try it again later

Arduino code for this solution (you can cut and paste it into your Arduino IDE, but it might require some cleanup since PDF cut-and-paste is a bit messy at times):

```
#include <LowPower.h>

const int sleeptime=2;      // change this value to the number of minutes you want it
to sleep
                            // I use 2 minutes for testing and proving

// the setup function runs once when you press reset or power the board

const int LIGHT_PIN = A0; // Pin connected to photocell voltage divider output
const int OPTO_PIN = 4;    // d4 is what we're using to operate pushbutton with
optocoupler
const int INT_LED=13;     // this is internal onboard LED

void setup() {
  pinMode(INT_LED, OUTPUT);
  pinMode(OPTO_PIN, OUTPUT);
  pinMode(LIGHT_PIN, INPUT);

  Serial.begin(9600);     // use for debug... will turn off later
  digitalWrite(INT_LED,LOW);  // turn off internal LED to save power
}

// the loop function runs over and over again forever

void loop() {
  // in this loop, we will first do quickpress to see if unit is on, by checking light
output immediately after

  quickpress(OPTO_PIN);
  int v = getlightreading();

  Serial.println("light value from main program = " + String(v));

  if (v < 800)  {    //this means the device is off and no led response occurred... so
we need to turn it on
                  //if reading is over 800, we assume the device responded with some
blinky lights
                  // light value will be between 0 and 1023... testing showed that
LEDs on GTM drive my sensor to 990 +-20

    Serial.println("no light signal... starting ON sequence in 5 sec");
    delay(5000);     // this is a delay for testing only... will remove

    secpress(OPTO_PIN);   // turns unit on
    delay(3000);
    threepress(OPTO_PIN);   // first sequence of 3 presses to put it into relay mode
    delay(3000);          // wait a few seconds, then
    threepress(OPTO_PIN);     // second sequence of 3 presses to put it into relay
mode

    Serial.println("device should now be in relay mode");
  }
  waitfortime(sleeptime);    // sleep time in minutes
}


int getlightreading(void)  {
```

```
  // this module takes the highest of 10 photocell readings over roughly 200ms to
determine if the GTM LED came on
  // this multiple-read approach accomodates timing issues since the LED flashes
instead of shining continuously

  int highval=0;
  int curval=10;

  for (int i=0; i<=10; i++) {
    curval = analogRead(LIGHT_PIN);

    Serial.println(String(curval));

    if (curval > highval)
      highval=curval;
    delay(20);
  }
  Serial.println("light reading = " + String(highval));
  Serial.println("");

  return highval;
}


void waitfortime(int min)  {
  // this function is not fully functional yet.. it just performs a nominal wait
period for testing at the moment
  // the low-power library permits the device to be shutdown using a watchdog timer
for restart... this can only do 8 seconds
  // so I'll try doing it repeatedly with a loop... first trying 15 loop cycles at 8
seconds each, for a 2 minute sleep
  int sec;
  int i=0;
  int cycles;

  sec=min*60;

  cycles=int(sec/8);    // this will approximate the time... may be off by a few
seconds, but who cares?

  for (i=0; i<cycles; i++) {
    LowPower.powerDown(SLEEP_8S, ADC_OFF, BOD_OFF);
  }

}

void quickpress(int pinnum)  {  // short keypress for checking unit status (get it to
blink)
  digitalWrite(pinnum, HIGH);
  delay(100);                    // keypress is 100ms in duration
  digitalWrite(pinnum, LOW);
}

void secpress(int pinnum)  {  // 1-second keypress for starting the unit
  digitalWrite(pinnum, HIGH);
  delay(1000);                // keypress is 1 second in duration
  digitalWrite(pinnum, LOW);
}

void longpress(int pinnum)  {  // long press to turn unit off (4 seconds works for
shutoff)
  digitalWrite(pinnum, HIGH);
  delay(4000);                // keypress is 4 seconds in duration
  digitalWrite(pinnum, LOW);
```

```
}

void threepress(int pinnum)  {  // does 3 quick presses to put into relay mode
  quickpress(pinnum);
  delay(200);               // delay 200ms between presses
  quickpress(pinnum);
  delay(200);
  quickpress(pinnum);
}
```

This code also provides the low-power capability, available using the low-power Arduino library from Rocketscream. Google it, get it, and install it into your Arduino IDE first.

See the addendum below about power saving features and approaches.

-RandyR

Addendum: Power Management

I added some power management capabilities into the code. After the pushbutton sequences are applied, the Arduino will go into sleep mode for a predetermined amount of time. This can be changed by altering the sleep time constant at the top of the code block before to compile and load it into the Arduino. I set it for 2 minutes for testing purposes, during which time I measure the current draw. Here are some results I have noticed so far:

Arduino Pro-Micro
This one is moderately power hungry, drawing 60-70 ma of current while running. When it fell into the low power sleep mode, it dropped to 24ma or so, which is still high. I suspect this is caused by the 3 LED's on the unit which are on all the time when it is powered. The onboard voltage regulator also sucks up some power.

Arduino Pro Mini
This one only used around 40 ma while running and active, and dropped to 14ma or so while sleeping. It has one LED onboard, which I removed to save power. After that, it drops to 8 ma while sleeping. The voltage regulator is likely to blame.

Most folks doing low-power applications with these devices suggest removing the regulator and the LED's from the board, in most cases. If you have a stable battery power supply (like a 5V USB battery), it probably won't hurt to remove the regulator.

Also, removing these tiny components is a hassle. Damn, they're small!!
I'll likely continue to research and experiment on power saving techniques to get the power down even further. Meanwhile, you can do the math yourself to figure out how much this thing is really chewing up, and assess what it means for your installation. If your external battery pack is 4000 mah, and this thing draws an average of 10ma (40 ma for a few seconds, followed by long periods of 8 ma sleep), it could supply the Arduino for 400 hrs. Of course, it will hopefully also be getting charged by solar panels or some other power source as well, so it may tend to be no problem at all.