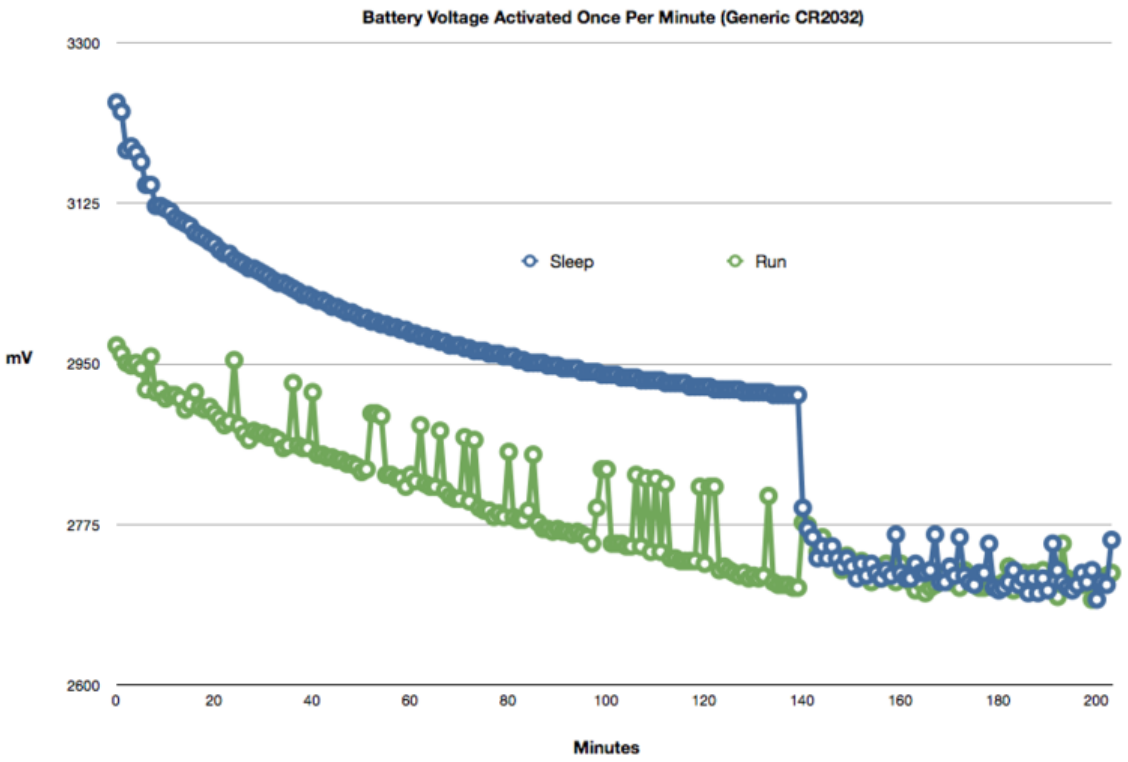
Low Power Coin Cell Voltage Logger

Created by Phillip Burgess

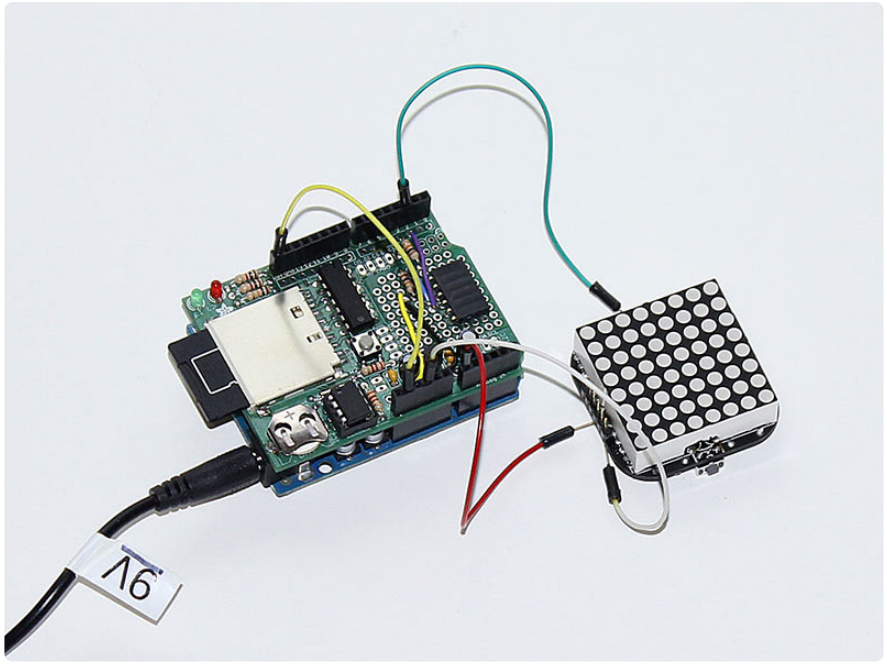


https://learn.adafruit.com/low-power-coin-cell-voltage-logger

Last updated on 2024-06-03 01:12:34 PM EDT

Table of Contents

Overview



In developing our new TIMESQUARE watch (http://adafru.it/1106), we knew that power use would be a hairy issue. The entire circuit, including an ATmega328P microcontroller and an 8x8 LED matrix, is powered from a single CR2032 lithium coin cell. We obsessed over different LED multiplexing arrangements and processor sleep modes, always trying to trim the power draw just a little bit more.

With the right tools such as the EEVblog µCurrent (http://adafru.it/882) and a good multimeter, measuring the most minute current changes is a simple task. But translating this into battery longevity isn't so cut-and-dried...the stated capacity in the battery datasheet assumes a small and constant load, while the watch current can vary greatly. What's more, the relationship between current draw and battery longevity isn't necessarily linear. This gets messy. Sometimes you just need to put math and theory aside, plug the thing in and observe the actual outcome.

To that end, we built a test fixture to simulate a consistent use case: activating the watch display once per minute and monitoring the battery voltage as it declines, allowing us to objectively compare different versions of the watch software. The raw data is logged to an SD card for later review and conversion into nice graphs. So this is primarily a tutorial on using the Data Logging Shield for Arduino, but along the way there are some good ancillary tidbits on hardware and software.

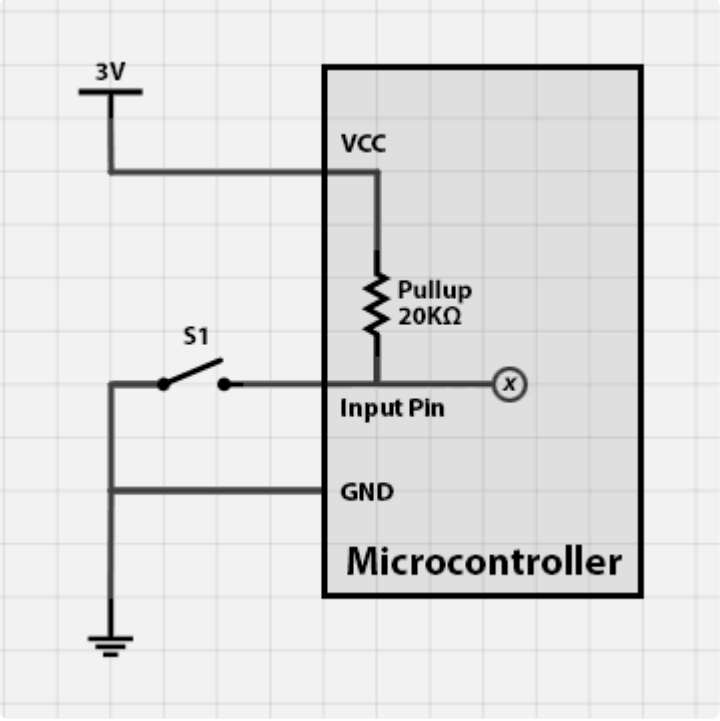
Hardware

Activating the watch...

The TIMESQUARE watch is activated by pressing either of the two side buttons. The time will be displayed for several seconds and the watch then turns off.

The ATmega microcontroller (MCU) spends most of its time in a very low-current power-down mode. The MCU pins to which the two buttons are connected were very carefully chosen — only these two pins support interrupt on change while asleep, which is used to revive the watch and enable the display.

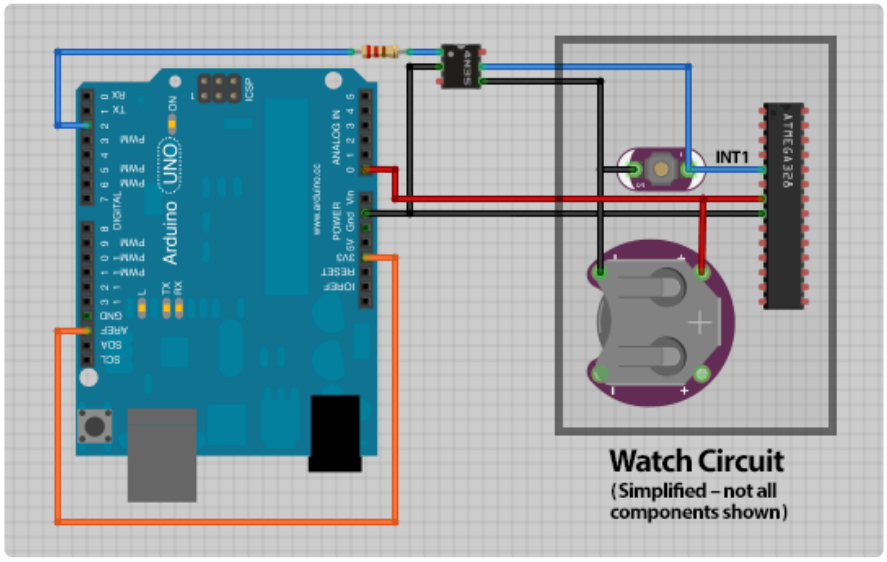
To minimize components, the internal pullup resistors are used on these pins. The buttons then simply connect between the two pins and ground. When the switch is open (button not pressed), what would normally be an unstable “floating” input is instead “pulled up” to VCC, which the MCU reads as a high logic level. Pressing a button creates a much lower resistance connection to ground, which is read as a low logic level.



Although the TIMESQUARE MCU could monitor its own voltage, we’d prefer to keep the device running only the actual watch code being tested, so as not to color the results. A separate microcontroller — an Arduino Uno — will record the measurements to an SD card. The + output from the 3V battery is connected to an analog input pin on the Arduino, and the Arduino’s **3.3V** output is routed to the **AREF** pin.

There are a couple of different ways this could be wired up. Given what we know from the pullup-vs-ground explanation above, we might be inclined to connect the watch input pin to one of the Arduino's digital output pins, then call the digitalWrite() function, passing HIGH or LOW to simulate an open or closed button state...but **this won't work correctly!** Most microcontrollers are opportunistic in finding usable voltage, and the watch will be perfectly happy to use power from the default HIGH logic state on that pin instead of the battery! This will skew the results horribly... it's worse than useless.

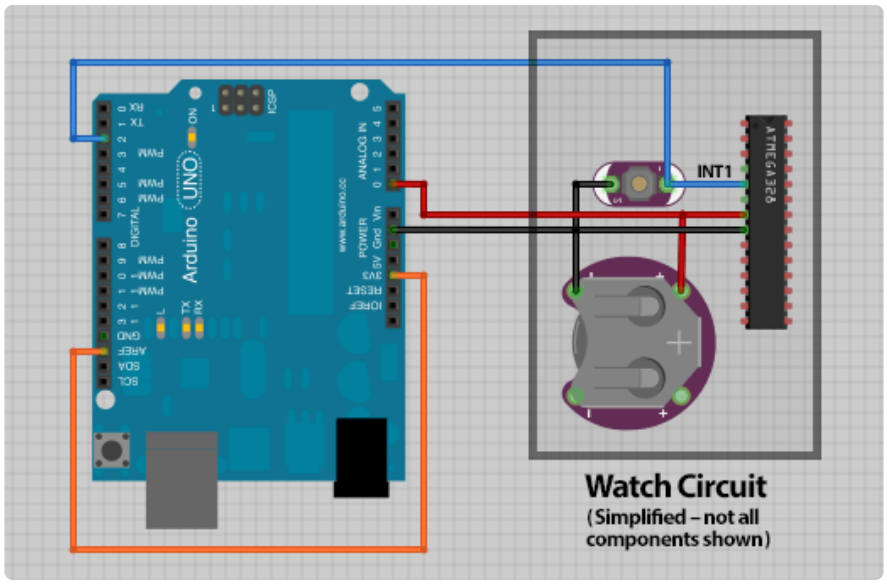
The traditional way around this is to use an optocoupler such as a 4N35 to isolate the voltages. The digital output from the Arduino drives an infrared LED inside the 4N35, which activates a phototransistor on the opposite side — wired to the watch — effectively pressing the button without injecting additional current into that circuit. Cool.



We could just do that and call it done, but:

- I had no optocouplers on-hand, and didn't want to wait for an order to arrive.
- There's a technique I've been wanting to use in a tutorial forever...

Connecting directly to an Arduino digital pin (as was "wrongly" proposed), there is a way to simulate the button press without feeding current into the watch circuit. It's not suited for every situation, but here it's a fortunate side-effect of the pullup-and-button design of the circuit. The trick is to never set the output pin HIGH. Yet we can still control the watch. Buhhh?



Mental models...

As a “software guy” who only stumbled into electronics much later, the implication of an MCU pin’s “tri-state output” and “high-impedance state” eluded me for the longest time...my eyes would glaze over at the jargon. If you’re in that same boat, hopefully this will clarify the situation, while also solving the problem at hand...

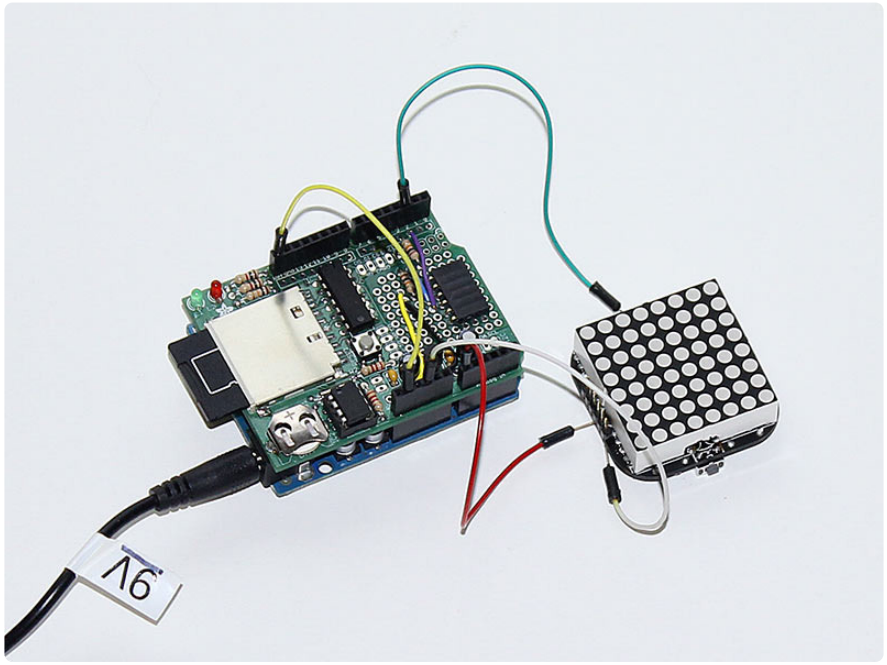
My naïve interpretation of HIGH and LOW pin states used to imagine HIGH as providing 5 Volts out, and LOW as being essentially an open circuit, passing no voltage. Such a model makes perfect sense when you’re blinking an LED (a favorite pastime!), but it’s **not accurate**. Rather, HIGH is a connection to VCC, LOW is a connection to ground...or, in jargon terms, they source and sink current, respectively. To block the flow of current, we have to set the pin as an INPUT. That’s the third state in “tri-state output.”

Or picture it this way: in school they made us watch dull fire safety films, where we learned not to throw doors open, but feel for heat first (indicating fire on the other side). That’s a fair model for voltage and MCU pins. As an INPUT — door closed — we can sense heat (or not) outside without letting fire pass through; the door impedes the fire’s progress. The high-impedance state. As an OUTPUT — door open — fire can now pass either way, whether flowing in (LOW) or out (HIGH). Three states. Tri-state.

So that’s how we control the watch. Setting and leaving the Arduino pin’s state LOW, then switching between INPUT (open circuit) and OUTPUT (closing the connection to ground), we’re performing the very same function as the button, without passing any voltage from the Arduino to the watch. Our battery measurements should be fairly accurate now.

Do keep that optocoupler technique in mind though...it's still a valid approach, and there are many other situations where they can be quite handy!

Here's the completed testing fixture. It's not much to look at...an Arduino, a Data Logging Shield, and wires soldered to the watch circuit's positive and negative connections, and one button:



(Ignore the extra components in the prototyping area on the shield...it was previously in use for another project, but those have no influence here.)

Software

Here's the complete sketch. This requires the SD card library, and a FAT-formatted SD card installed in the shield.

Every 60 seconds, the software takes two voltage measurements from the watch. The first is taken while the watch is still in the power-down state, indicating the resting voltage of the battery. The second reading is taken two seconds later, after the watch "marquee" display has been running (which drags down the voltage...so many LEDs!).

The minute counter (starting from zero) and two voltages are written to a line in a text file in CSV (comma-separated value) format, which can then be directly imported into most spreadsheet applications, or it's fairly easy for other software to parse.

Since we're just counting the passage of minutes, the counter value is written to the file; it's not an absolute time stamp. If a project requires proper time/date stamps,

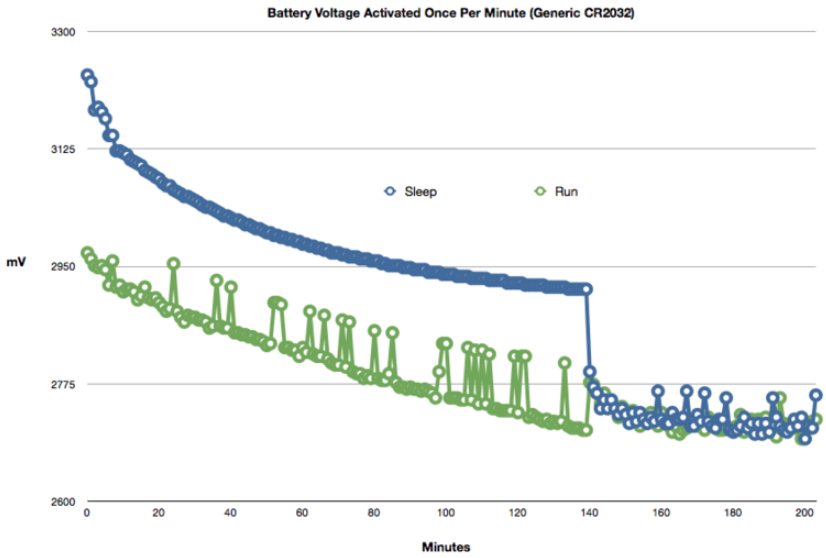
we'd want to tie into RTClib (an Arduino realtime clock library) for reading the shield's clock.

```
// Watch battery voltage logger.  Activates watch circuit every
// 60 seconds and records voltages (sleep and running) to SD card.
// Based on Tom Igoe's Datalogger example from the SD library.

// Connections:
// Arduino GND to watch battery -
// Arduino analog 0 to watch battery +
// Arduino digital 2 to pull-down switch on watch
// 3.3V to AREF (both on Arduino)

#include &lt;SD.h&gt;

long minutes = 0; // Elapsed time / line number

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

  // Use 3.3V analog reference for better resolution on 3V battery
  analogReference(EXTERNAL);

  // Watch is awakened with button tap which ties pin (w/internal
  // pullup) to ground.  Rather than write high/low levels to this
  // wire, a pin is set LOW and switched between input (high
  // impedance) and output states to approximate the button press
  // without introducing voltages into the watch circuit.
  pinMode(2, INPUT);
  digitalWrite(2, LOW);

  Serial.print("Initializing SD card...");
  Serial.println(SD.begin(10) ? // 10 = card sel. pin (4 on Arduino Ethernet)
    "card initialized." :
    "card failed, or not present.");

  // Sketch continues even if SD init fails...may just want to
  // read output in serial monitor for debugging.
}

void loop()
{
  long     mVsleep, mVrun;
  char     dataString[40];
  File     dataFile;
  uint32_t start = millis();

  mVsleep = 3300L * analogRead(A0) / 1023L; // Millivolts while asleep

  pinMode(2, OUTPUT); // Pull watch button LOW
  delay(100);         // Hold a moment
  pinMode(2, INPUT);  // and release

  delay(2000);  // Let the watch run for a couple seconds...

  mVrun = 3300L * analogRead(A0) / 1023L; // Millivolts while running

  sprintf(dataString, "%ld,%ld,%ld", minutes++, mVsleep, mVrun);

  if((dataFile = SD.open("datalog.csv", FILE_WRITE))) {
    dataFile.println(dataString);
    dataFile.close();
  } else {
    Serial.println("error opening datalog.csv");
  }
  Serial.println(dataString);
```

```
  // Serial and SD card access times may vary.  Rather than delay(),
  // monitor time to allow remainder of 1 minute to pass.
  while(millis() &lt; (start + 60000L));
}
```

Results

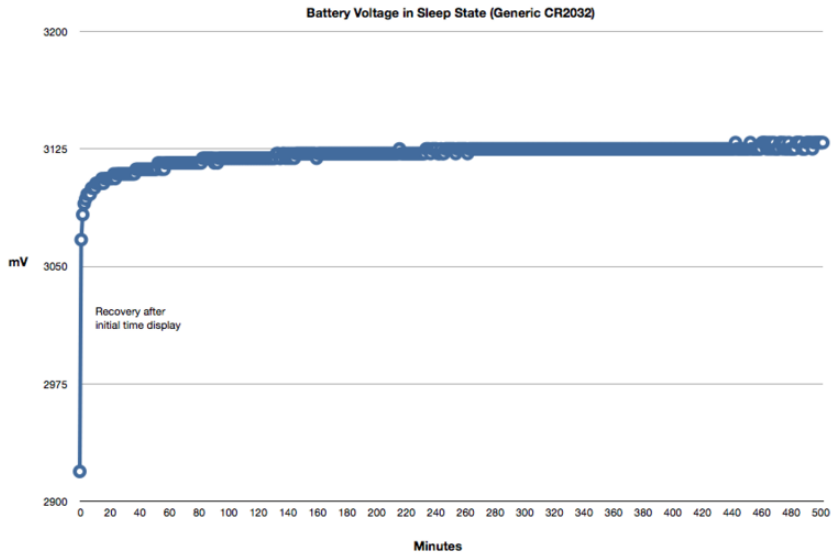
Here are some results after importing the resulting CSV files into a Numbers spreadsheet, and producing a chart:



The jumps in the green line are due to the content of the watch display at the moment the reading was taken; some digit combinations simply scroll by quicker than others. A blank display draws less current, and the voltage reading will be slightly higher. It's surprisingly sensitive...in another test, the resting battery voltage was seen rising slightly right when the thermostat kicks on in the morning. (Batteries are mildly sensitive to temperature. This is why your smoke alarm waits until you're sleeping to start that "low battery" beep...the cooler nighttime air brings the already weak battery's voltage just below the warning threshold.)

Notice at 140 minutes the two lines merge. This is what we dubbed the "death spiral." The battery voltage has dropped low enough that the watch executes a brown-out reset. This restarts the code, which restarts the display, causing another brown-out... repeating until the battery gives out fully.

(For the record, battery life has been improved substantially since this early graph was taken, so you should get well more than 140 viewings from your watch!)



Here you can see the battery voltage gradually recover following an initial time display. This is chemistry, and is why there isn't a linear relationship between current draw and battery longevity.

The stuttering parts of the line are due to the limited resolution of the Arduino's analog-to-digital converter. Those could be filtered out by taking multiple voltage readings and averaging the results, but it wasn't necessary to go to that level of detail here; seeing the trend was sufficient.

Other Lessons

The key to making TIMESQUARE practical was to trim the power-down current as much as possible. Certainly, the running current is important too, but the power-down state is where the watch will spend most of its time by far. There may be idle times when it's left in a drawer for days or weeks...maybe even months, though we hope not...and it has only a single coin cell to draw from. As you can imagine, considerable effort was spent testing and measuring sleep modes and disabling every possible peripheral to reduce the idle current.

One of the more power-hungry peripherals on the ATmega 328P is the brown-out detect circuit, which senses a low voltage condition and calls an interrupt function, the brown-out reset (BOR) handler. This feature is used in products for such things as storing state information in EEPROM before gracefully shutting down. The BOR circuit is enabled by default on the Arduino...and this is very important.

Certain Atmel chips...the 328P among them...can disable the brownout circuit in software (rather than configuration fuses), potentially saving many microamps of current. If you're programming a "raw" chip via the ISP header, that's fantastic...if you need to save every last bit of power, and if you don't need the brownout detection, have at it. But if you're using a bootloader-based programming system like Arduino, disabling BOR can have disastrous results!

As the supply voltage dips below the brownout threshold, without BOR the chip will start to behave erratically, and may spontaneously jump to any random memory location. And if that code eventually leads into any bootloader function that erases or writes a flash page, the application — or much worse, the bootloader itself — can become corrupted, leaving no easy way to re-flash the watch.

This is NOT the unlikely one-in-a-million change you might think! First, the watch WILL repeatedly brown out any time the battery runs low. Second, keep in mind that it doesn't have to jump exactly to the start of a block-erasing function, just to any code that may eventually lead there. The odds of this happening during an unprotected brownout seem to be about 1 percent...the phenomenon has been observed in the wild with other projects and even while developing this code...it's a real thing! So BOR is left enabled to provide a proper safety net. If you're programming for an Arduino bootloader-based board, you should too.

Really, resist the allure of the nano-amps, DO NOT go blindly adding BOR-disabling code to your project, you'll regret it later. Just don't. Okay? Don't. Thanks.