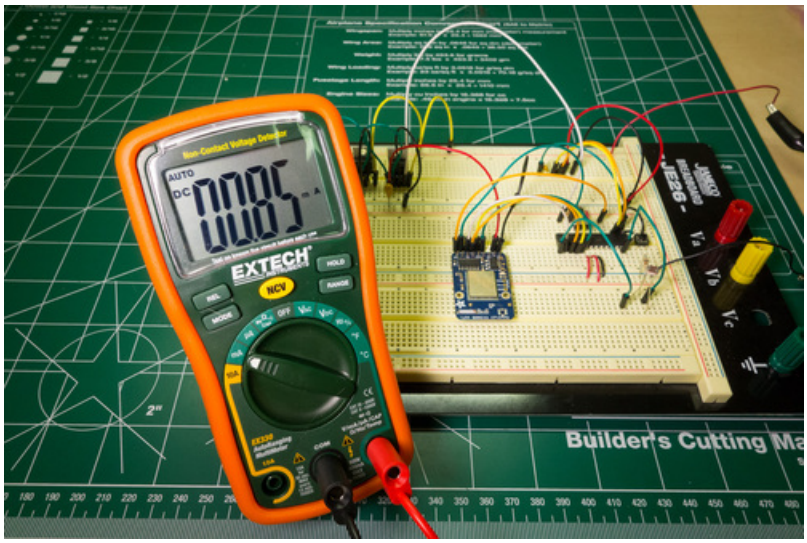
adafruit learning system

Low Power WiFi Datalogger

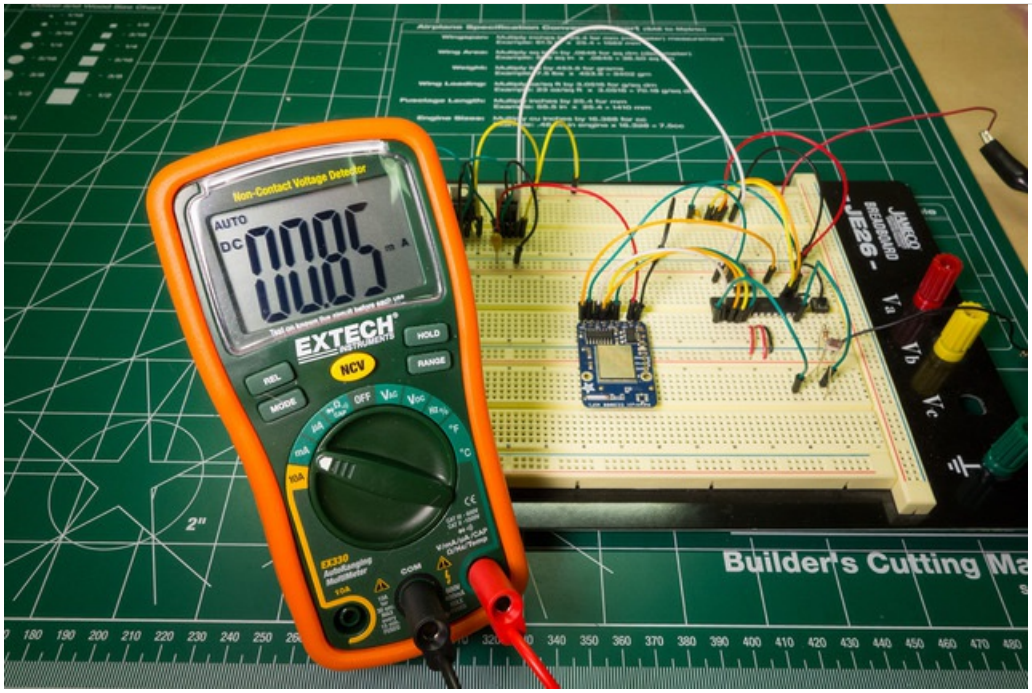
Created by Tony DiCola



Last updated on 2018-08-22 03:39:40 PM UTC

Guide Contents

Overview



Have you ever tried to run your Arduino project on batteries but been surprised to get only a few hours of battery life? In this guide I'll show you how I built a simple WiFi sensor datalogger and optimized it to run for days on batteries. You can learn some useful power saving tips to apply to your own Arduino project!

Before you get started it will help to familiarize yourself with these topics:

- Multimeters (https://adafru.it/aOy)
- Batteries (https://adafru.it/cl5)

Download and unzip the software that will be used in this project from the following link:

https://adafru.it/d8l

https://adafru.it/d8l

Battery Life & Current Consumption

Battery Life

How long will your project run on batteries? To answer that question you need to understand the capacity of your batteries and the amount of current consumed by your project.

Battery capacity is measured in milliamp-hours (abbreviated as mAh) which are a measure of how many hours a battery can sustain a constant draw of current. For example an 800 mAh battery can in theory provide 800 mA of current for an hour. However, in reality as current draw increases battery capacity decreases. Look for a datasheet from your battery manufacturer to see what the capacity of your batteries are at various current draws. For more information on batteries and battery capacity, check out this battery guide (https://adafru.it/c5x).

With the battery capacity and average current consumption, you can compute the expected runtime of your project by solving the equation:

Battery capacity (in mAh) / Average current consumption (in mA) = Hours of expected runtime

For example if your project consumes 150 mA and your batteries are rated at 800 mAh at that current draw, you can can compute:

800 mAh / 150 mA = 5.33 hours expected runtime

Current Consumption

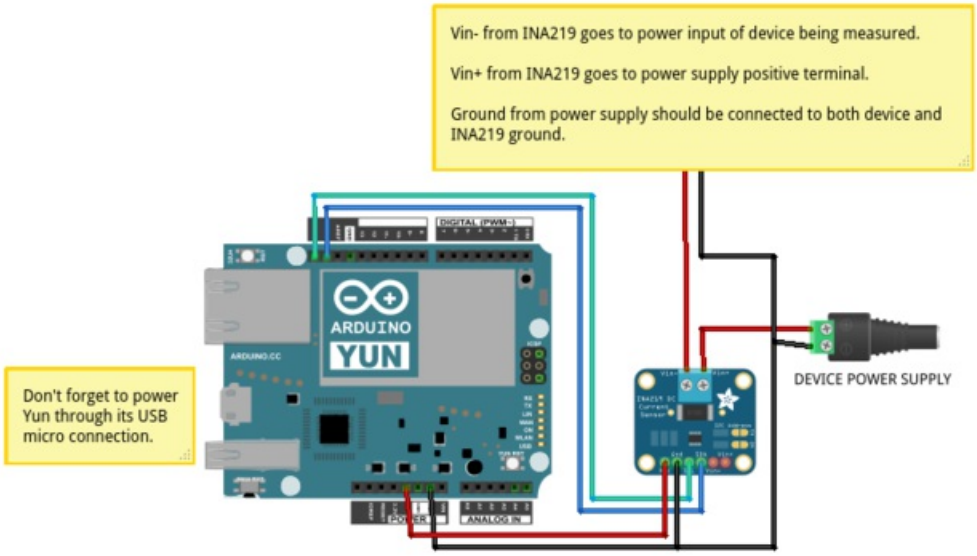
For simple projects with a fairly constant current draw, a multimeter might be all you need to measure average current consumption. If you aren't familiar with how to use a multimeter to measure current, take a look at this multimeter guide (https://adafru.it/d8J)for a great overview.

For more complex projects which have changing current demands, like this project, you'll want to measure current consumption over time. Using a series of current measurements you can compute an accurate estimate of average current consumption.

A great way to measure current over time is using this INA219-based current sensor breakout (http://adafru.it/904). If you aren't familiar with the INA219, be sure to read the guide on it's usage (https://adafru.it/d8K). The INA219 can be attached directly in series with the power input of your project and queried over an I2C interface to read current consumption. Using a device that can speak I2C, such as an Arduino, Raspberry Pi, or Beaglebone Black, you can record the current consumption of your project over time.

For this project I used an Arduino Yun (http://adafru.it/1498) connected to the INA219 breakout to log current consumption. A sketch running on the Yun talks to the INA219 and uses the Yun bridge library's Console class (https://adafru.it/d8L) to send data to my computer over my WiFi network. A simple Python script runs on my computer to periodically record current measurements from the Yun.

The Yun is not very good at being a low-power device since its running Linux! For that reason we're using it just for tracking the power usage of our CC3000 setup



Above is a picture of the Yun and INA219 datalogger hardware I built. The code for this hardware is provided in the software download:

- The **Yun_INA219_Datalogger** subdirectory is the Arduino sketch that should be loaded on the Yun hardware.
- The **Scripts/yun_ina219_datalogger.py** file is the Python code to record measurements from the INA219 datalogger.

If you have a different Arduino, such as an Uno or Mega, check the sketch code for tips on how to convert it to write to the serial output with a few small changes.

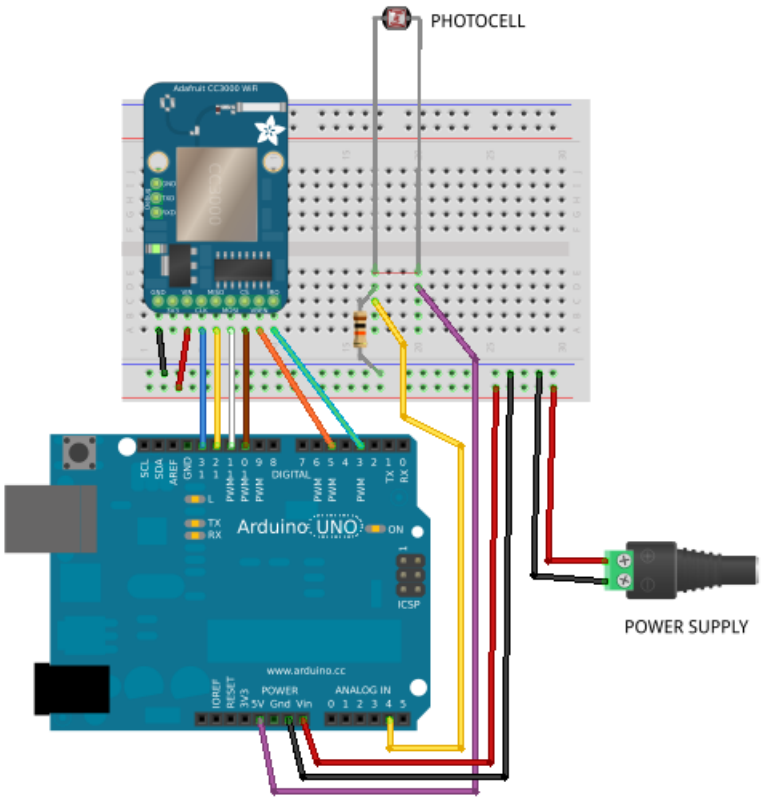
One thing to note about the INA219, by default the software library assumes you want to measure a wide range of voltage and current with an accuracy of about 1 mA. If you know the voltage and current demands of your project are relatively low (under 16 volts and 400 mA), you can configure the INA219 for better accuracy of about 0.1 mA. Add this function (https://adafru.it/d8M) to your INA219 library and call it instead of ina219SetCalibration_32V_2A() in the library's begin() function.

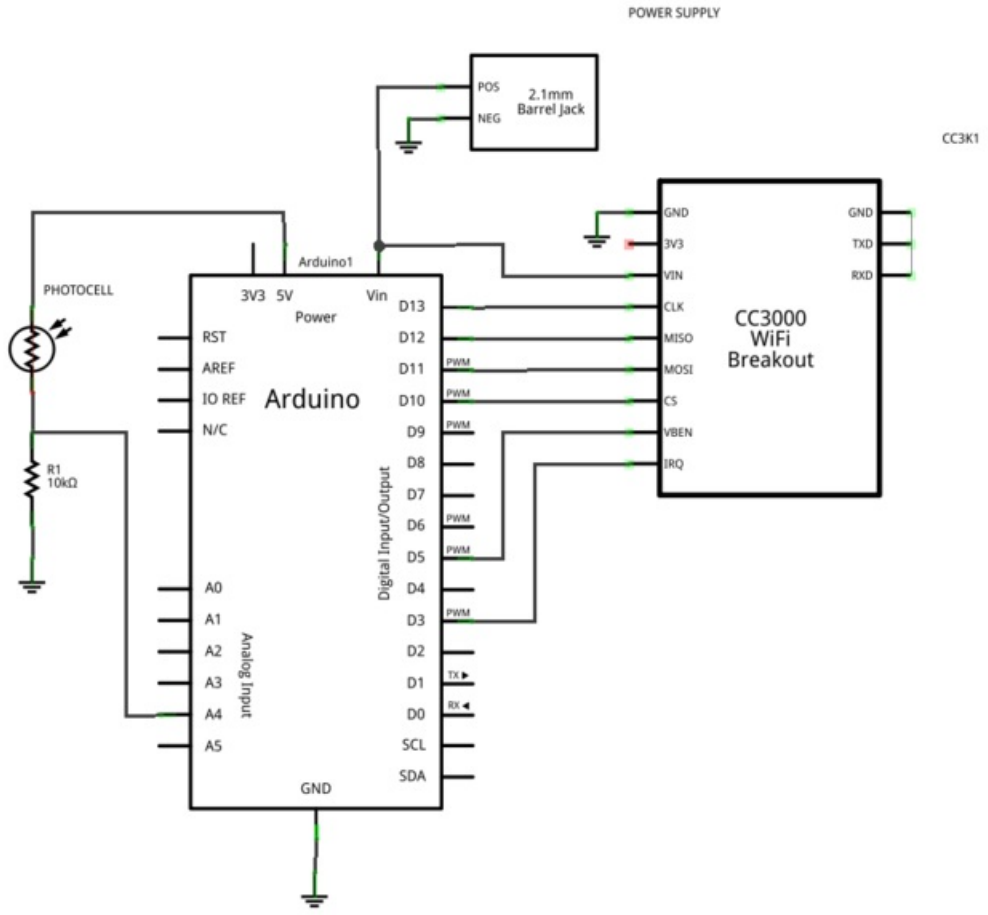
Example 1: No Optimizations

This example demonstrates the basic WiFi datalogging hardware. The hardware reads the value of a photocell every minute and sends it to a server listening on TCP port 8000. Later examples will investigate how to reduce power consumption and increase battery life of this simple example.

You will need the following parts to build this example:

- Arduino Uno (http://adafru.it/50) or Nano
- CC3000 breakout (http://adafru.it/1469) or shield (http://adafru.it/1491)
- Photocell (http://adafru.it/161)
- 10 kilo-ohm, 1/4 watt resistor
- 6x AA (http://adafru.it/248) or AAA battery holder as a power source
- Batteries, preferably rechargeable. Look for a name brand battery with a datasheet that provides the battery capacity.





Above you can see the hardware and schematic for the project. Hook up the CC3000 to the Arduino through the standard pins:

- CC3000 VIN to power supply positive.
- CC3000 ground to power supply ground.
- CC3000 CLK to Arduino digital pin 13.
- CC3000 MISO to Arduino digital pin 12.
- CC3000 MOSI to Arduino digital pin 11.
- CC3000 CS to Arduino digital pin 10.
- CC3000 VBEN to Arduino digital pin 5.
- CC3000 IRQ to digital pin 3.

Finally hook up the photocell:

- One photocell lead to Arduino 5V power.
- The other photocell lead through the 10 kilo-ohm resistor to ground.
- From the junction of the photocell and resistor, connect a wire back to an Arduino analog input such as analog pin 4.

Once the hardware is assembled, load in Arduino the **Example_1_No_Optimizations** sketch from the software download. There are a few configuration values you can change in the #define values at the top of the sketch:

- **ADAFRUIT_CC3000_IRQ**, **ADAFRUIT_CC3000_VBAT**, and **ADAFRUIT_CC3000_CS** should be set to the appropriate pins which connect the CC3000 to the Arduino.
- **WLAN_SSID**, **WLAN_PASS**, and **WLAN_SECURITY** should be set to the appropriate values for your wireless network.

- **SENSOR_PIN** should be set to the analog input which is connected to the photocell and resistor.
- **SERVER_IP** and **SERVER_PORT** should be set to the IP address and port of the computer which will receive photo sensor readings. The IP address should be written with commas instead of periods, for example if your computer has an IP address of 192.168.1.100 you would use a value of '**192, 168, 1, 100**' without quotes.

Once the configuration is updated, compile the sketch and load it on the hardware.

Listening Server

To receive measurements on your computer, you can either run a python script (good for Windows, Mac, or Linux with python installed (https://adafru.it/cFQ)) or use the netcat command (good for Mac or Linux out of the box, or Windows if installed separately (https://adafru.it/d8N)).

To run the python listener, in a terminal window navigate to the **Scripts** subdirectory of the software download and execute this command:

```
python listener.py
```

Alternatively, to use the netcat tool in a terminal window execute:

```
nc -lk 8000
```

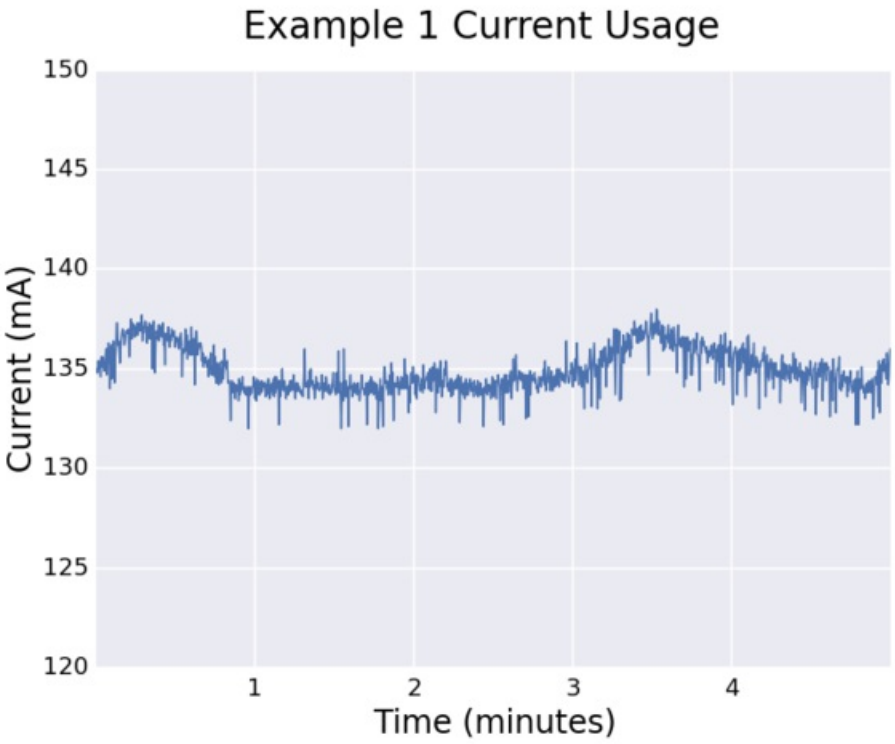
Either server will listen for connections on port 8000 and print out data received to the standard output. The python script will also append the time that each measurement occured, whereas the netcat command won't add anything. You can stop either server by pressing Ctrl-C in the terminal.

Once the server and hardware are running, wait a few minutes to confirm the server receives a measurement from the hardware about once a minute. If you don't see anything on the server after a few minutes, make sure the hardware is configured with the right server IP and there aren't any firewall or routing issues that might block the network traffic.

Current Usage

Looking at the code for this example you'll see it's a fairly simple sketch. The setup() function connects to the WiFi network, and the loop() function waits a minute before sending a new reading to the server using the CC3000 board.

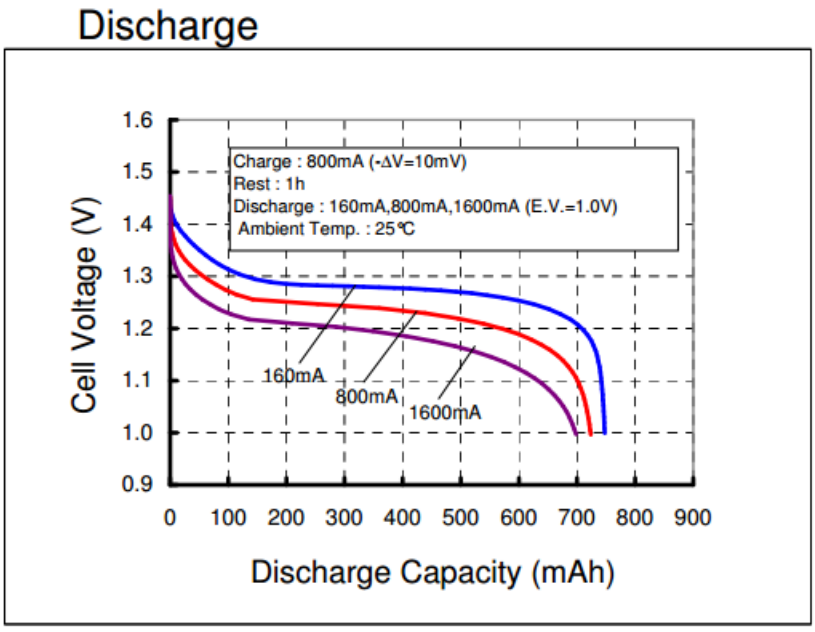
The current consumption of this hardware should be fairly high because both the Arduino and CC3000 constantly run at full power, even when sitting idle between measurements. To confirm this assumption I connected my hardware (an Arduino Nano and CC3000 breakout) to the INA219 current datalogger (https://adafru.it/d8O) and measured the current consumption for 5 minutes.



Above you can see the current consumption I measured for this example hardware. The current usage is quite steady at a value of about 135 milliamps. This measurement confirms that there's no difference between waiting to send and sending a measurement, in both states the Arduino & CC3000 are drawing a steady amount of current.

Expected Battery Life

With a measure of the average current consumption, I can estimate and test how long the hardware will run on batteries.



For this test I'm using eneloop AAA Ni-MH rechargeable batteries (https://adafru.it/d8P). Above is a picture of the battery's rated milliamp-hour capacity at different discharge currents (taken from this datasheet (https://adafru.it/d8Q)).

The 160mA discharge current curve is pretty close to the current consumption of this example, 135mA, so the battery capacity should be 750 mAh.

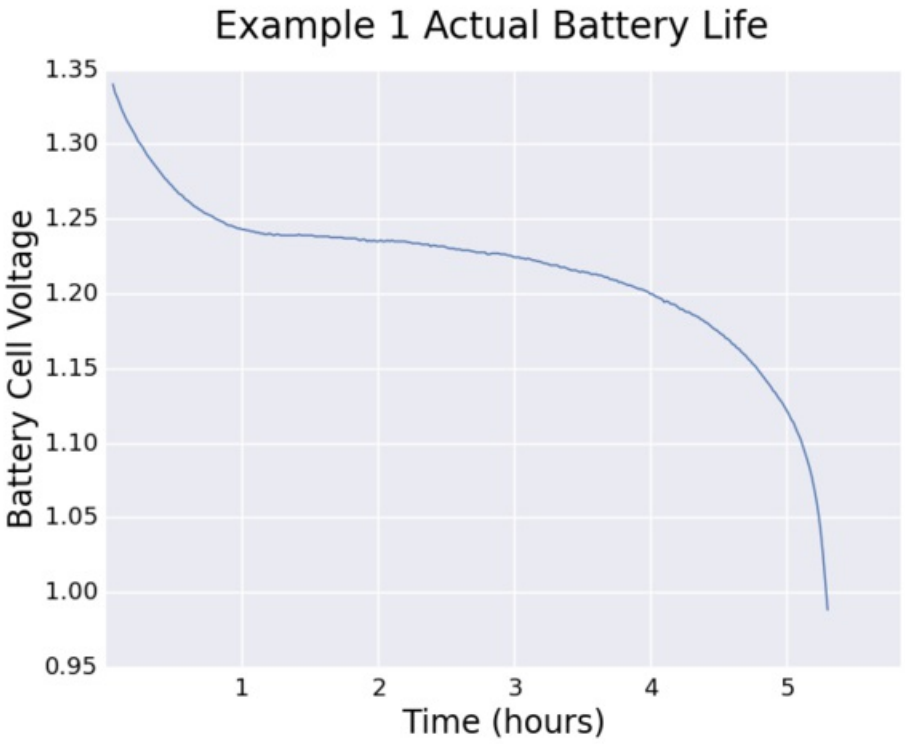
If the hardware's average current consumption was much higher, for example around 800mA or more, I would want to use a lower milliamp-hour rating in battery life calculations. Remember, pulling more current from a battery results in a lower battery capacity.

Dividing the milliamp hour capacity of the batteries by the average current consumption will yield the number of hours I can expect the hardware to run.

> **Expected battery life:**
>
> 750 mAh / 135 mA = **5.6 hours**

Actual Battery Life

To measure the actual battery life I ran the hardware on batteries and used the INA219 datalogger to measure battery pack voltage over time.



Above is a graph of the battery cell voltage over time. The cell voltage was computed by dividing the total battery pack voltage by the number of batteries in series, in this case 6.

At a full charge the battery cell voltage starts around 1.35 volts. Over time the voltage drops following a curve that's similar to what's seen in the battery datasheet.

Once the battery cell voltage drops below 1 volt the batteries are exhausted and the hardware will quickly stop running. Be careful not to run rechargeable batteries too far below 1 volt or else they might be damaged.

The actual battery life of the hardware was quite close to the expected battery life. Factors such as age of the batteries, ambient temperature, and number of charge/discharge cycles can all affect capacity and might explain some of the difference in expected vs. actual capacity.

> **Actual Battery Life:**
>
> **5.3 hours**

Example 2: Power-Down Sleep

In the previous example you saw the hardware pulled a constant amount of current even while sitting idle between measurements. Is there a way to reduce the power consumption during those idle periods? Indeed, using the sleep modes of the Arduino and CC3000 you can greatly reduce current consumption.

Sleep is a special mode of the processor which halts normal program execution and shuts down internal components to reduce power consumption. Once a processor is asleep it can only be awakened by specific events such as an external interrupt from a button press, or a timer counting a period of time.

One thing to note is that sleep modes aren't currently exposed by Arduino's programming library. However, because Arduino is built on the AVR Libc library (https://adafru.it/d98) you can access that library's sleep (https://adafru.it/d99) and power management (https://adafru.it/d9a) functions in an Arduino sketch. This example is a little more complex than a typical Arduino sketch because it makes use of these lower level library functions to access sleep modes.

Table 9-1. Active Clock Domains and Wake-up Sources in the Different Sleep Modes.

	Active Clock Domains					Oscillators		Wake-up Sources							
Sleep Mode	clk_{CPU}	clk_{FLASH}	clk_{IO}	clk_{ADC}	clk_{ASY}	Main Clock Source Enabled	Timer Oscillator Enabled	INT1, INT0 and Pin Change	TWI Address Match	Timer2	SPM/EEPROM Ready	ADC	WDT	Other I/O	Software BOD Disable
Idle			X	X	X	X	X(2)	X	X	X	X	X	X	X	
ADC Noise Reduction				X	X	X	X(2)	X(3)	X	X(2)	X	X	X		
Power-down								X(3)	X				X		X
Power-save					X		X(2)	X(3)	X	X			X		X
Standby(1)						X		X(3)	X				X		X
Extended Standby					X(2)	X	X(2)	X(3)	X	X			X		X

Notes: 1. Only recommended with external crystal or resonator selected as clock source.
2. If Timer/Counter2 is running in asynchronous mode.
3. For INT1 and INT0, only level interrupt.

Above is a table of the sleep modes from section 9 of the ATmega328P processor datasheet (https://adafru.it/d9b) (the processor that powers the Arduino Uno R3 and Nano v3). Notice that different sleep modes disable various internal clocks and components to reduce power consumption. Also of interest is what can wake the processor from each sleep mode, such as an interrupt or timer. I recommend skimming the sleep section of the datasheet for an overview of each sleep mode.

For this example I chose to use the lowest power consumption sleep mode, power-down sleep. In this mode almost all components of the processor are disabled and only an external interrupt, two-wire interface signal, or watchdog timer can wake the processor. Because I need the hardware to wake from sleep after a period of time, I use the watchdog timer to wake from power-down sleep.

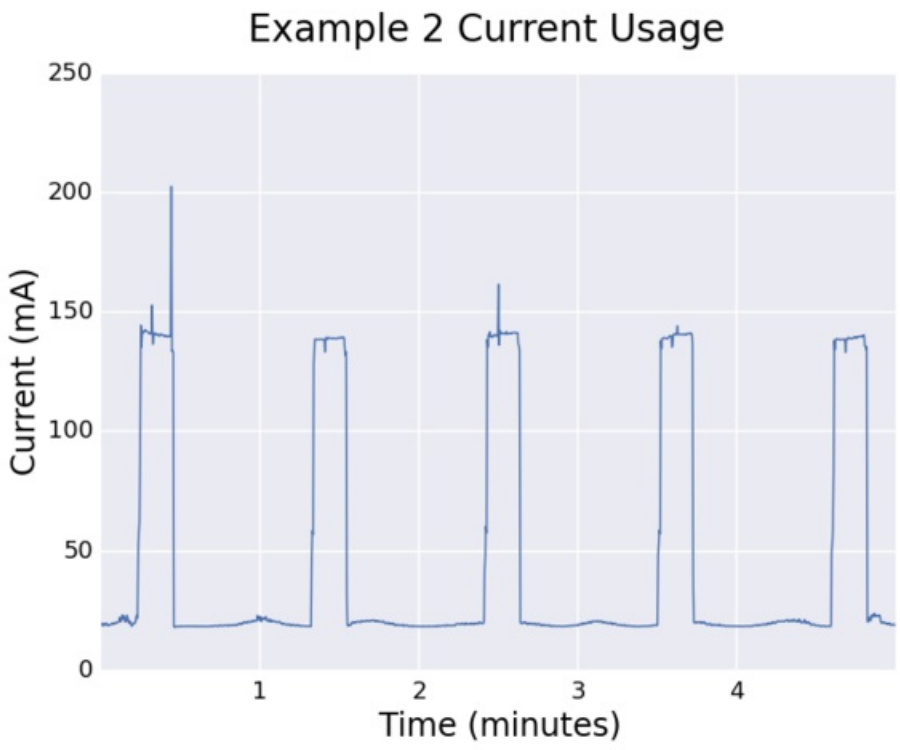
The watchdog is a component inside the AVR processor which resets the processor if it detects normal program execution has stopped. Typically the watchdog is used to make an unreliable or buggy program run with more stability. However by changing the watchdog control registers, it's possible to use the watchdog as a timer which wakes the processor from power-down sleep instead of reseting it. See section 10.8 of the ATmega328P datasheet for more information on the watchdog.

The hardware for this example is exactly the same as the hardware from the first example. The only changes made are in the software. Load the **Example_2_Power_Down_Sleep** sketch from the software download in Arduino. Update the configuration in the same way as example 1, and upload the sketch to your hardware.

Looking at the sketch code, the differences from the previous example are:

> **The watchdog is configured as a timer to wake from sleep.**
>
> Inside the setup() function the watchdog control register is changed so the watchdog functions as a timer that fires interrupts instead of resetting the processor. The watchdog timer has a limited configuration of periods, so the maximum period of about 8 seconds is used for this example.
>
> **The Arduino is put into power-down sleep mode while sitting idle between measurements.**
>
> When the watchdog timer fires every 8 seconds the Arduino will be awakened from sleep. Once awake the Arduino increments a count and after it reaches 7 sleep iterations (roughly 56 seconds) a sensor measurement is logged. Finally, the Arduino is put back into sleep mode while it waits for the watchdog to wake it again.
>
> **The CC3000 is disabled while the Arduino is asleep.**
>
> In the setup() function communication with the CC3000 is initialized, and then the wlan_stop() (https://adafru.it/d9c) function is called to put the CC3000 into a low power disabled state. When the Arduino awakes to take a measurement the CC3000 is enabled by calling wlan_start() (https://adafru.it/d9c), connects to the wireless network to send the measurement, and finally shuts down by calling wlan_stop() (https://adafru.it/d9c) again.

Current Usage



Above is a graph of the current consumption over time for this example. There are periodic peaks of ~135 mA when a measurement is being logged, and then valleys of ~19 mA when the Arduino is asleep. This is quite an improvement

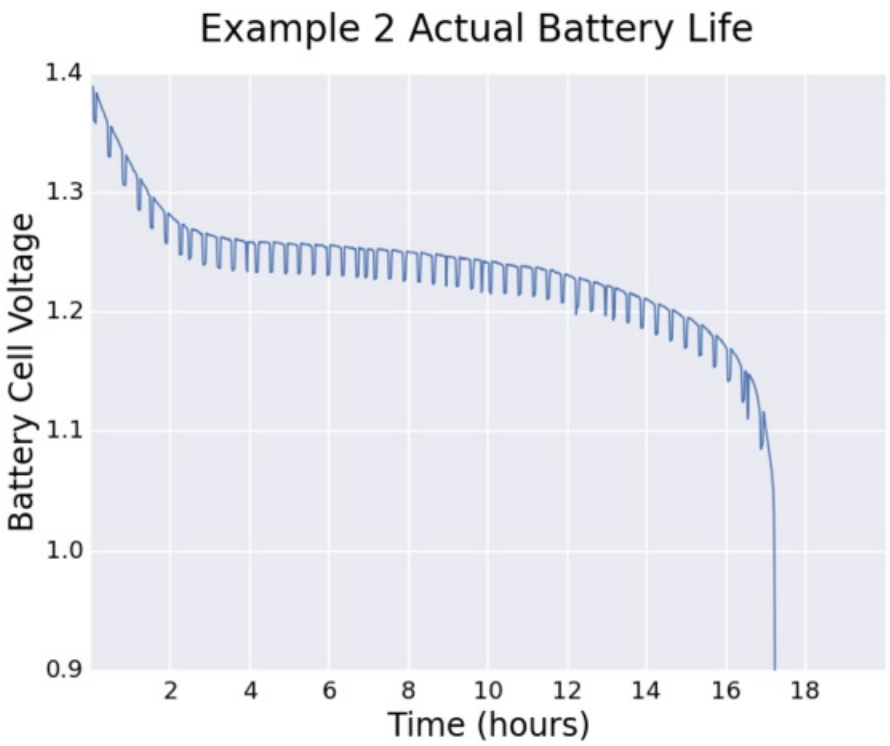
from the first example as the average current consumption has fallen from 135 mA to about 45 mA!

Expected Battery Life

For this test I'll use the same AAA Ni-MH batteries as the first example (https://adafru.it/d9d). Looking at the chart of battery capacity at different discharge rates, I'll extrapolate a bit and assume at 45 mA of average current consumption the batteries will have a capacity of 775 mAh.

> **Expected battery life:**
>
> 775 mAh / 45 mA = **17.2 hours**

Actual Battery Life



Looking at the chart of battery voltage over time above, there's an interesting pattern that appears. When the hardware is consuming more current to send a measurement the voltage drops slightly. There's some aliasing in this chart because both the voltage sampling and hardware measurements happen at the same 1 minute frequency. However you can see how the changing current demands affect the battery voltage over time.

In this test the hardware ran for exactly the amount of time expected!

> **Actual battery life:**
>
> **17.2 hours**

Example 3: Upgraded Hardware

The drop from 135 mA to 45 mA of current consumption in the previous example was significant, but are there more opportunities to lower the power usage? In this example you'll see how to improve the hardware of the datalogger to reduce the power consumption even further.

There are some components on the datalogger that can pull a significant amount of current even while the Arduino is asleep:

Power LEDs

Both the Arduino Nano and CC3000 have small LEDs that constantly draw current through resistors. These LEDs are useful indicators when the hardware is connected to wall power, but when running on batteries they consume precious current. Looking at the CC3000 (https://adafru.it/d9e) and Arduino Nano schematics (https://adafru.it/d9f), each LED pulls about 8-10 mA of current alone.

Voltage Regulators

The linear voltage regulators on board the Arduino and CC3000 do their job well, but aren't particularly power efficient. Power regulators have a minimum current draw, their quiescent current, and the regulators on the Arduino and CC3000 have a quiescent current of about 5-10 mA each.

Photocell and Pull-up Resistors

The photocell and resistor in series with it, like the LEDs, are another source of constant power consumption. Even the pull-up resistor on the Arduino Nano's reset pin consumes a non-trivial amount of current, about 5 mA.

To reduce the power consumption I made the following changes to the hardware:

Switch to barebones Arduino using the ATmega328P processor on a breadboard.

By building my own Arduino on a breadboard I can upgrade and remove components compared to the stock Arduino board. With my custom built Arduino I removed the power LED, serial to USB communication chip, and switched to a larger 1 mega-ohm pull-up resistor on the reset pin to reduce current consumption.

Upgrade the voltage regulators.

I added much more power efficient LT1529 voltage regulators (https://adafru.it/d9g) to provide the 5 volts and 3.3 volts required by the Arduino and CC3000 respectively. These regulators have a very low quiescent current of 0.05 mA. The regulators can also be shut down completely by pulling a shutdown pin low. This means the Arduino can shut off power to the CC3000 while it's asleep and ensure there's no unnecessary current draw from the CC3000 power LED.

One down side of these regulators are their cost; in single quantities they cost about $4-5. However their high current rating (3 amps) will make them useful to have for future projects. You don't need to use these exact regulators too--look for any 3.3 and 5 volt regulators that have low quiescent current and ideally a shut down pin.

Power the photocell/sensor only during measurements.

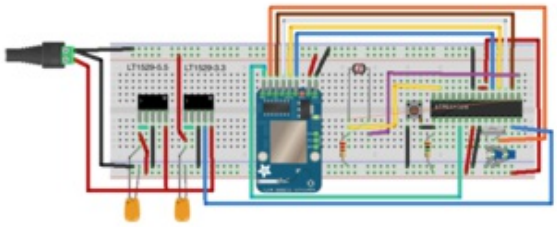
> I made a small change to supply power to the photocell using a digital output of the Arduino instead of the 5V power regulator. This means I can control when power is applied to the photocell rather than it constantly drawing power. Small optimizations like this to control when sensors or other components in your hardware have power can add up to significant power savings.

To build this example you will want to be familiar with how to build a breadboard Arduino (https://adafru.it/d9h). Be warned this is an advanced project that could be difficult for a beginner to build and troubleshoot. You will need the following parts:

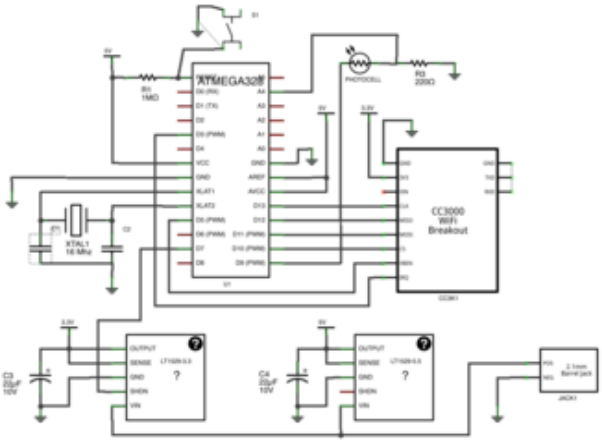
- **Breadboard Arduino parts:**
  - ATmega328P processor (http://adafru.it/123).
    - Make sure to get the P version because it supports more power efficient sleep modes than the standard ATmega328 processor.
  - 16 mhz crystal or resonator.
  - 2x 22 picofarad ceramic capacitors.
  - Small momentary pushbutton (http://adafru.it/367) for the reset switch.
  - 1 mega-ohm resistor to pull the processor reset pin high.
  - Arduino programmer (http://adafru.it/46) to load the Arduino bootloader on the ATmega chip.
    - You won't need this programmer if you buy an ATmega328P that is already programmed (http://adafru.it/123) with the Arduino bootloader.
  - FTDI serial to USB cable (http://adafru.it/70) to communicate with and load sketches on the Arduino.
- **Upgraded voltage regulator parts:**
  - An LT1529-3.3 (https://adafru.it/Cc-) and LT1529-5 (https://adafru.it/Cd0) voltage regulator, in the TO-220 5 pin package.
    - Unfortunately these chips won't directly plug in to the breadboard so you'll need to carefully bend the pins to fit. Use pliers and bend as few times as possible to reduce the chance of breaking them.
  - 2x 22 microfarad capacitors.
- **Datalogger parts:**
  - CC3000 breakout board (http://adafru.it/1469). Unfortunately you can't use the CC3000 shield easily with the breadboard Arduino.
  - Photocell (http://adafru.it/161) and 10 kilo-ohm resistor.

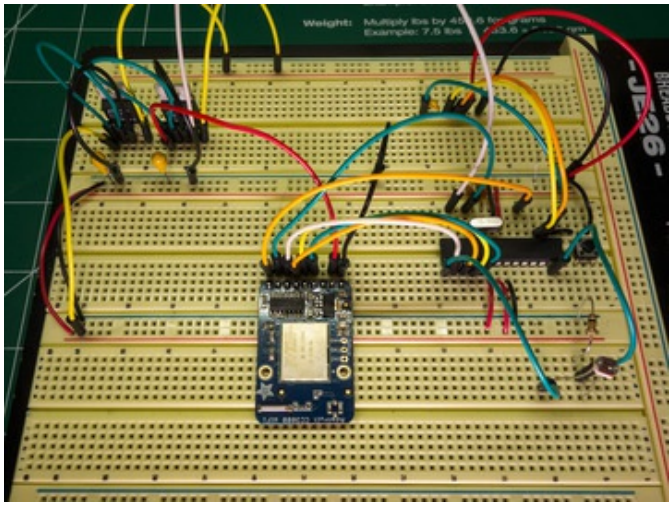
Assemble the hardware as shown in the pictures and schematic to the left. Follow this guide on building and programming a breadboard Arduino (https://adafru.it/d9h) for help. Build the Arduino first and verify it works with a blinking LED or other test program before moving on to add the CC3000 and its 3.3 volt regulator.

One important change to note is that the CC3000 is no



longer powered on its VIN pin. The external 3.3 volt regulator will instead feed power into the 3.3V pin on the CC3000. **Be very careful not to mix up wires and send 5 volts into this pin or you could damage the CC3000!** Check voltages with a multimeter before hooking up the outputs of the voltage regulators to be sure they are what you expect.





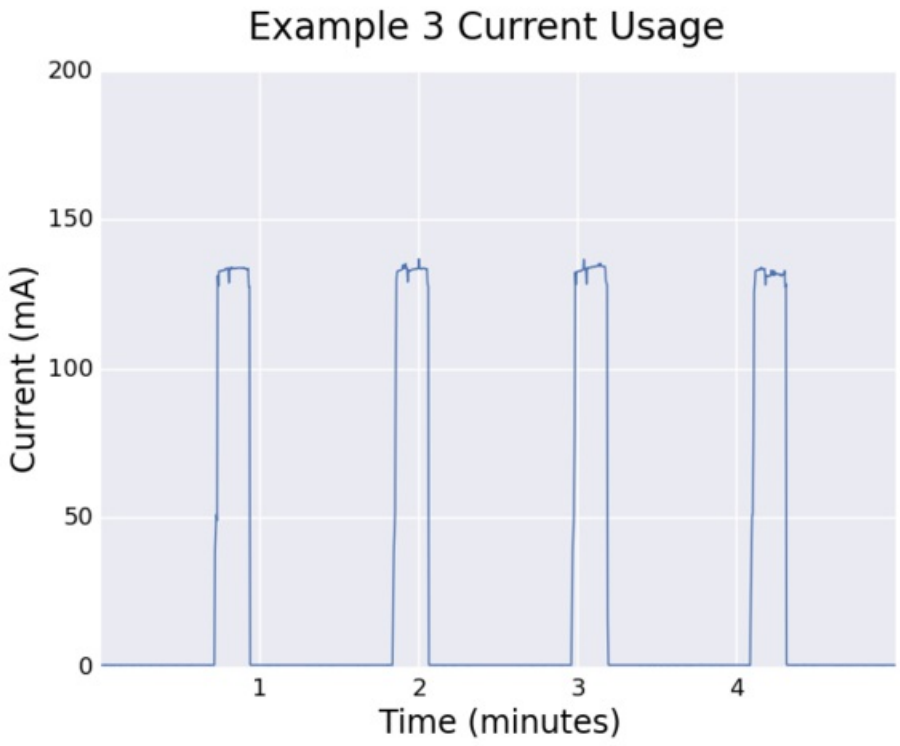
Once the hardware is assembled, load the **Example_3_Upgraded_Hardware** sketch in Arduino. You will need to change a few new configuration values at the top of the sketch:

- **REG_SHUTDOWN_PIN** should be set to the digital pin which is connected to the LT1529-3.3 regulator shutdown pin.
- **SENSOR_POWER_PIN** should be set to the digital pin which is connected to the photocell for power.

The rest of the sketch code is almost exactly the same as the previous sleep example, with a few small differences:

- When the Arduino wakes up or sleeps the CC3000 it also turns on and off the 3.3 volt regulator by pulling its shutdown pin low or high (low is shutdown, high is enabled).
- The sensor power pin is set high right before reading a measurement and then pulled low again. This powers the photocell only when necessary to take a reading.
- Before entering sleep mode a few registors are updated to disable the ATmega processor brown-out detection circuitry while asleep. This step slightly reduces the power consumption of the Arduino while it's asleep.

Current Usage



The graph of current consumption over time above shows a significant drop in current usage at idle. With the Arduino asleep the hardware only consumes 0.9 mA of current, a savings of almost 20 mA from the previous example!

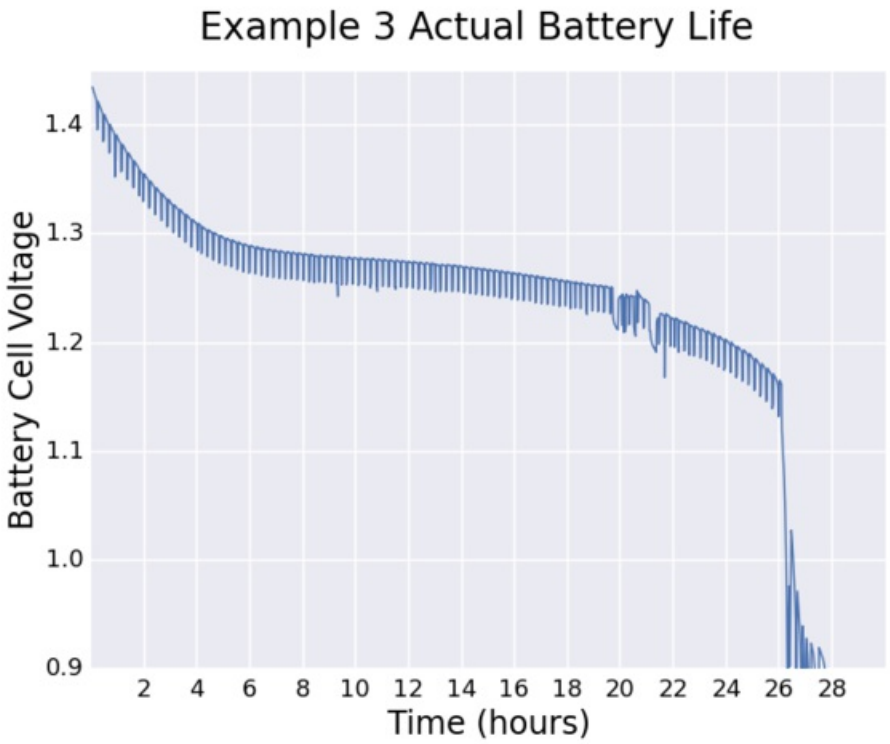
At full power during measurement send the current consumption is slightly reduced to 132 mA. In total the average current consumption of this example is only about 23.3 mA.

Expected Battery Life

For this test I will again use the same AAA Ni-MH batteries as previous tests, and assume they have a 775 mAh capacity like in the last example.

> **Expected battery life:**
>
> 775 mAh / 23.3 mA = **33.3 hours**

Actual Battery Life



Above is the graph of battery cell voltage over time. Unfortunately after 20 hours my wireless router started refusing to let the CC3000 connect to my network and caused the code to get stuck in a loop while fully powered. Resetting the router and the hardware helped fix the problem, but later in the night it happened again around hour 26. Getting stuck at the higher current consumption very quickly drained the batteries and stopped the test prematurely.

This is a good case where the watchdog's normal reset functionality could be useful. If the code gets stuck in a loop the watchdog can reset the unresponsive hardware and prevent it from staying in a constant high power state. However because the watchdog is also used as a timer to wake from sleep, care would need to be taken to use the watchdog as a reset when awake and as a wake up timer when asleep.

Without the connection issues the curve of battery life looks like it was on a path to hit at least 30 hours. However with the issues encountered, the actual battery life in this test was **28 hours**.

Extending Battery Life Further

What do you do when you want to run even longer on batteries? Some options to increase the runtime of the project are:

> **Use higher capacity batteries.**
>
> Switching to AA rechargeable batteries with 1900 mAh of capacity would easily let the hardware run for over 3 days on a single charge. Lithium ion batteries with 4000 mAh capacity could run for about a week. A small lead acid battery with 10 amp-hours of capacity could run for over half a month! Check out this battery guide (https://adafru.it/cl5) for more information on the types and capacities of different batteries.
>
> **Reduce measurement frequency.**

Instead of recording a measurement every minute, increase the measurement frequency to every 10 minutes or more. The longer the measurement frequency, the more time the Arduino will stay in its low power sleep mode and reduce the average current consumption.

Using the measured current usage data you can estimate the new current consumption by solving the equation:

Ia = Current while awake = 133 mA
Is = Current while asleep = 0.9 mA
Sa = Seconds spent awake = 10 seconds
F = Measurement frequency = 600 seconds (10 minutes)

(Ia * Sa + Is * (F - Sa)) / F = average current consumption at frequency F

(133 * 10 + 0.9 * (600 - 10)) / 600 = 3.1 mA average current consumption

With a 3.1 mA average current consumption the hardware could run for almost a month on 1900 mAh AA batteries!

Scavenge power with a solar cell.

Consider adding a solar cell to the hardware to charge the batteries while the Arduino is asleep. Check out this guide on solar cells and battery charging (https://adafru.it/d9k) for more information. If there's enough light available your hardware could run indefinitely!

Summary

In summary, optimizing your project for low power usage and long battery life is a matter of measuring and reducing current consumption. A multimeter (http://adafru.it/308) or INA219 breakout (http://adafru.it/904) are great tools for measuring current consumption. Using processor sleep modes and removing unnecessary current drains from hardware can help greatly reduce current consumption. It's possible to build projects which run for days or even months on batteries!

For more information on reducing power consumption, check out the following links:

- Power saving techniques for microprocessors (https://adafru.it/d9l). This is a great guide to reducing power consumption on an Arduino project that covers many of the power saving techniques used in this guide.
- Sleeping Arduino (https://adafru.it/d9m). This is a good series of blog posts that dive into the various sleep modes of the Arduino's processor.
- ATmega328P datasheet (https://adafru.it/d9b). Go straight to the source for information how low power sleep, watchdog, and other power saving functions of the Arduino processor work. Check out section 9 in particular for low power features.