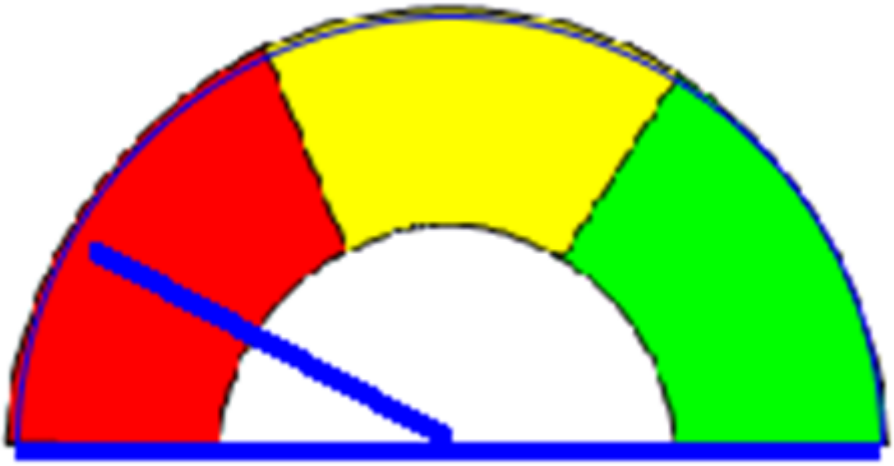
Energy Budgets

Created by mike stone

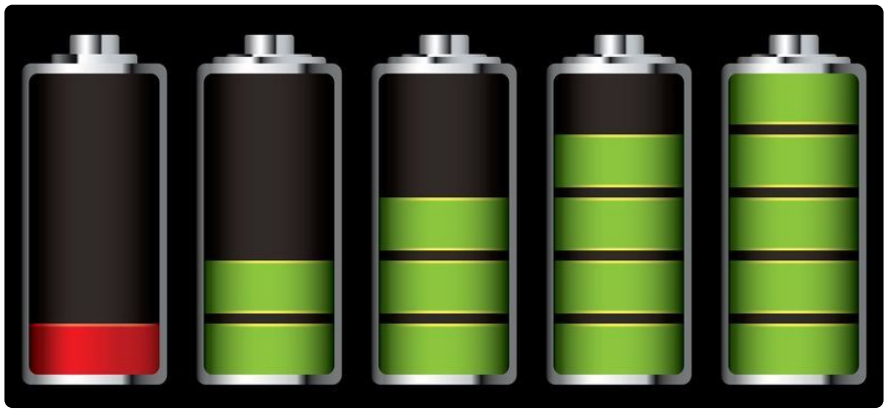


https://learn.adafruit.com/energy-budgets

Last updated on 2024-11-02 12:37:59 PM EDT

Table of Contents

Overview



When you're thinking about a project that will be battery powered, one of your first questions will be "how big a battery do I need?" or maybe, "I have a battery, how long will it last?"

This guide is an intermediate introduction to calculating the energy budget for battery-powered projects. There's math, but nothing hard: mostly multiplication and addition, with a little division here and there.

The bigger challenge is remembering what the numbers mean. That involves learning some vocabulary, and working with the ideas until they stop feeling new and strange.

Energy Storage

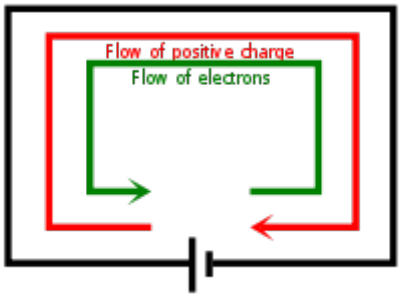
An energy budget starts with a specific battery's rated storage, which is measured in **milliamp-hours:** weird units that don't actually measure energy.

Milliamps measure current, and current is defined as the motion of charge (usually electrons) over time. 1 Ampere is 1 Coulomb (6.24e18) electrons per second, so multiplying current by time tells you how many electrons you sent from one place to another. 1 hour is 3600 seconds, so 1 milliamp-hour actually means 3.6 Coulombs of charge (about 22.5 billion-squared electrons).

Energy is measured in Joules, which don't have any direct relationship to numbers of electrons.

When we put the two ideas together -- Joules and Coulombs -- we get Volts (defined as Joules per Coulomb).

A battery (or any power source) is kind of like an escalator for electrons: the electrons enter through the positive end, the battery pumps Joules of energy into them, then they leave through the negative end. The energy leaves the battery with the electrons:



(yeah, that sounds backwards.. electrons have negative charge, so they move in the opposite direction from what we call 'conventional current'. It's a long story that involves Benjamin Franklin A: being very persuasive, and B: getting it backwards. To be fair, it was 100 years before anyone said, "hey...")

So a battery's milliamp-hour rating really means "this battery holds enough energy to raise X-many Coulombs of charge to a higher voltage".

That's a mouthful, and doesn't seem to answer "how long will my battery last?" any better than milliamp-hours, but stay tuned.

For simple circuits, like lighting an LED, morphing a battery's stored energy to milliamp-hours makes battery life calculations easier: a 100mAh coin cell can power a 10mA LED for 100mAh/10mA=10 hours.

Things get more complicated when you have an ESP8266 microcontroller that wakes up every ten minutes, takes some measurements (using 5mA), opens a WiFi connection to the Internet (80mA), posts data to Adafruit.IO, then goes back to sleep again.

Or say you're doing some cosplay, taking LiPo power to 5V through a PowerBoost, and using it to run a NeoPixel strip with the LEDs at a 60% duty cycle.

Milliamp-hours don't make those problems noticeably easier.

For those kinds of calculations, you need an **energy budget**.

By the way -

I'm not just throwing around units for the sheer joy of saying 'Coulombs' (at least, not entirely). The process I'm using is called **dimensional analysis**: a way to rearrange problems so they're easier to think about and to solve.

For the payoff, we need two more players:

When we multiply voltage by current, we get Joules/Coulomb x Coulombs/second. That simplifies to Joules/second, or **Power: the rate at which energy moves from one**

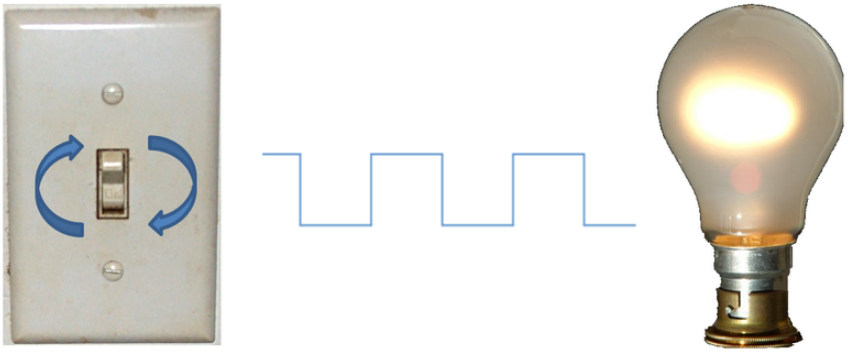
thing to another. The official unit of power is the Watt, named in honor of the man who put the steam in Steampunk.

If we multiply power by time, we get **Work: the amount of energy that did move from one thing to another**. There are an appalling number of units for work, including the Watt-hour, foot-pound, newton-meter, and erg, but I'm going to stick with Joules.

Energy budgets are easier to do when you arrange things in terms of power and work.

Guide image source: Microsoft PowerPoint clipart

Let's start with PWM



Pulse Width Modulation (PWM) is a convenient place to start thinking about energy budgets. The basic idea is simple, and it happens fast enough for the numbers to make sense.

When you pulse power to a device, you turn the power on and off. If you do that repeatedly, you get 'on' and 'off' pulses, each of which lasts a certain amount of time. We refer to those times as the **pulse width**. If we change the width of the pulses relative to each other, we **modulate** the width of the pulses.

It's traditional to make the on-time and off-time add up to a fixed **pulse period**. That gives us some number of uniform-sized pulses per second, called the **PWM frequency**. We also refer to the on-time and off-time as percentages of the pulse period, and we call the percentage width of the on-time the **duty cycle**.

For reference, let’s take something easy like using PWM to control the brightness of an LED. The operating principle is simple: when the LED has power, it lights up. When it doesn’t have power, the LED stays dark. Let’s say the LED uses 10mA when lit.

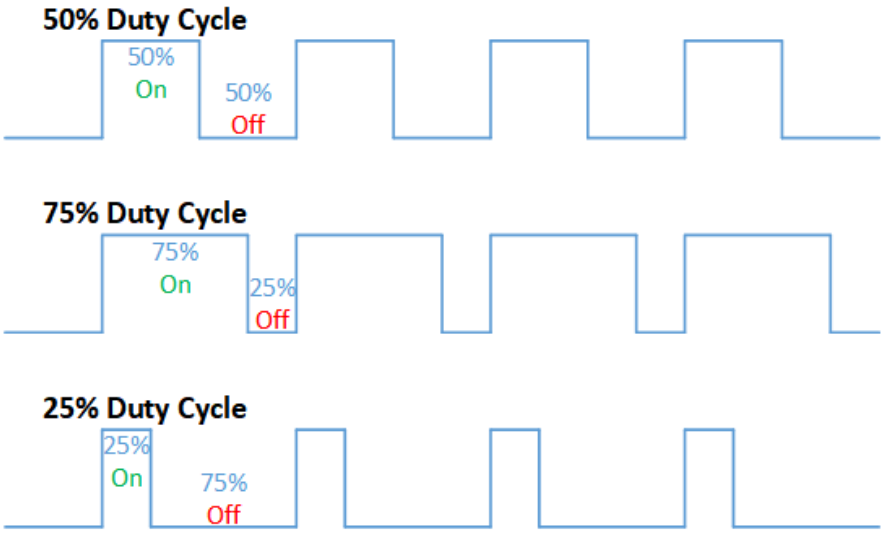
Let’s also say the PWM frequency is 1kHz (the circuit controlling the LED emits a pulse every millisecond) and that a pulse can have any width between 0ms (always off) and 1ms (always on).

It's easy to find the current for those two cases: if the LED is always off, the current is 0mA. If the LED is always on, the current is 10mA.

And to set the stage for what's coming, if 10 milliamps of current flows through an LED for 1 millisecond, it means 10 microcoulombs of charge have gone through the LED:

```
10e-3C/s x 1e-3s = 10e-6C
```

Flipping that around gives us the fundamental idea behind energy budgets: if 10 microcouloumbs of charge flow through a component in 1 millisecond, the average current through the component during that millisecond is 10mA.



Changing the duty cycle

Now let's make each pulse last 990 microseconds (a 99% duty cycle). The amount of charge that flows through the LED is only slightly less than it was for the always-on case:

```
10e-3C/s x 990e-3s = 9.9e-6C
```

Dividing the charge (9.9uC) by the total pulse width (1ms) give us the average charge during that time:

```
9.9e-6C x 1e-3s = 9.9e-3C/s = 9.9mA
```

So the average of 10mA at a 99% duty cycle is 9.9mA.

Flipping it around

If we reverse the output values so the LED only gets current for 10 microseconds (a 1% duty cycle), we could imagine that LED getting the power the first LED didn't use:

```
10mA - 9.9mA = 0.1mA
```

which matches the value we'd calculate the long way:

`10e-6C/s x 10e-3s = 100e-9C`

`100e-9C / 1e-3s = 100e-6C/s = 0.1mA`

Some other values

Now that you've seen the calculations a few times, we can apply them to other duty cycles:

It makes sense to think a 50% duty cycle (500us on, 500us off) would have half the average current of the always-on value:

`10e-3C/s x 500e-6s = 5e-6C`

`5e-6C / 1e-3s = 5e-3C/s = 5mA`

and that a 27.5% duty cycle would have that fraction of the always-on case:

`10e-3C/s x 275e-6s = 2.75e-6C`

`2.75e-6C / 1e-3 = 2.75C/s = 2.75mA`

Doing it the easy way

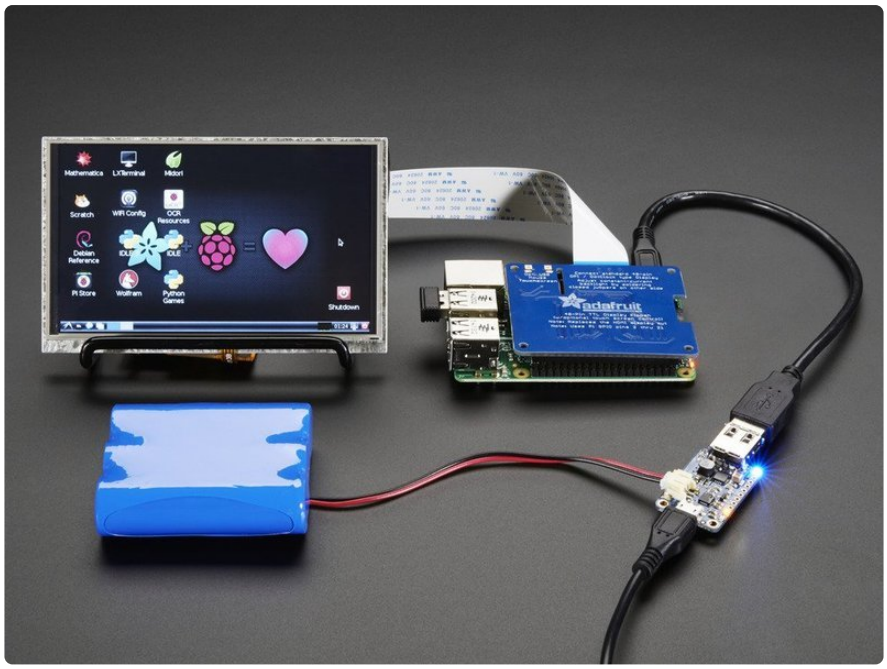
By now you've probably worked out that the average current for a duty cycle is just the total current times the duty cycle. You can rearrange the calculations to prove that's true, but seeing the pattern empirically is almost as good.

Knowing why it works, and being able to derive the calculations from the definitions of current and charge, is better.

Next, multiple voltages

Energy budgets are especially useful when you have circuits with parts running at different voltages: one that takes power from a 3.7V LiPo but has some pieces running at 3.3V while others take 5V from a PowerBoost, for instance.

As a matter of fact, battery life for that kind of circuit can change depending on how you make the power connections. We'll get to that in a bit, but first let's take a look at switching converters like the Adafruit PowerBoost (http://adafru.it/2465).



DC-to-DC converters

Switching converters get their name from the fact that they literally open and close switches to change one voltage to another. The switches are connected to an inductor, and the magic happens as energy moves into and out of the inductor's magnetic field.

Over the long term, all switching converters are constant-power devices: you have to send 1W into an ideal switching converter to get 1W out of it. The input can be 303mA @ 3.3V and the output can be 200mA @ 5V, or vice versa. As long as the voltage and current multiply to the same amount of power on both sides, the math works.

No real switching converter is perfect though, so you have to send more power in than you get out. The PowerBoost is at least 90% efficient, for instance, so it's safe to assume 1.11W of input will guarantee 1W of output.

Over the short term, switching converters are constant-energy devices that use PWM, doing the same amount of work in each part of the duty cycle.

To use the PowerBoost as an example, let's say we have a circuit that needs 100mA @ 5V.

The PowerBoost starts by closing the switch that connects the 3.7V LiPo to the inductor, and lets power flow in for 15 microseconds. During that time, the average current is about 250mA (trust me.. I've already run the numbers).

If we multiply the voltage, current, and time we get 13.9uJ of work, and about 95% of the energy from that work gets stored in the inductor's magnetic field:

Item	Voltage	Current	Time	Work
Input	3.7V	250mA	15us	13.9uJ
95% stored	-	-	-	13.2uJ

Then the PowerBoost opens the switch to the LiPo and closes the output switch for 10us. The same 250mA average current flows out, with about 95% of the energy stored in the inductor's magnetic field (12.5uJ) pushing it.

If we divide that 12.5uJ of energy by 10us of time and 250mA of current, we get an output voltage of 5V:

Item	Voltage	Current	Time	Work
Input	3.7V	250mA	15us	13.9uJ
95% stored	-	-	-	13.2uJ
95% released	-	-	-	12.5uJ
Output	5V	250mA	10us	12.5uJ

(the 95% stored and released energy values are sleight of hand, by the way. I said the PowerBoost is at least 90% efficient, and the square root of 0.90 is about 0.95. Applying the same scaling factor twice gives the correct result while helping to show where the energy losses occur)

The input and output currents are both examples of PWM, so we can use what we learned about that to calculate the average currents:

The 250mA input current only flows for 15us per cycle, for a total of 3.75uC of charge. Each PWM cycle lasts 25us, and if we divide 3.75uC by 25us, we get an amortized input current of 150mA.

The 250mA output current only flows for 10us per cycle, for a total of 2.5uC of charge. Dividing that by the 25us cycle time gives us an amortized output current of 100mA.

If we amortize the work done during the input and output phases, we get 556mW of input power and 500mW of output power:

Item	Voltage	Current	Time	Charge	Work	Am. current	Am. power
Input	3.7V	250mA	15us	3.75uC	13.9uJ	150mA	556mW
95% stored	-	-	-	-	13.2uJ	-	-
95% released	-	-	-	-	12.5uJ	-	-
Output	5V	250mA	10us	2.5uC	12.5uJ	100mA	500mW

The actual conversion is done by moving the same amount of energy in different amounts of time. The amortized values of the duty cycles make the same average current through the inductor, and slightly decreasing energy, look like different voltages and currents at the input and output.

You can use the same calculations in circuits that use step-up or step-down converters of any value.

Now about those connections..

If you're building a hybrid 3.3V/5V circuit that gets its power from a PowerBoost, it's natural to ask whether you should connect the 3.3V side directly to the LiPo or to use the 5V supply.

Let's run the numbers assuming the 3.3V circuit wants 100mA:

If we take power directly from a 3.7V LiPo, the 3.3V circuit burns off an average of 0.4V in its voltage regulator. The power loss is 0.4V x 0.1A = 40mW.

If we use the 5V supply, the PowerBoost has to draw an average of 150mA from the LiPo, and will lose up to 10% of that during the boost process. Using the numbers above, 1.4uJ/25us = 56mW of loss getting the 3.7V up to 5V.

Then the 3.3V circuit's regulator has to burn off the excess 1.7V, which produces 1.7V x 0.1A = 170mW of loss. Adding that to the 56mW lost in the PowerBoost gives us a total of 226mW loss.

Using the PowerBoost's 5V output to run a 3.3V circuit wastes more than 4x as much energy as running the 3.3V circuit directly from the LiPo.

Now a sample energy budget

We've already shown that 1 milliamp-hour is 3.6 Coulombs. We also know than 1 Volt applies 1 Joule of energy to each Coulomb of charge. Putting those together, we can calculate the actual energy storage for some of the common battery sizes:

Battery type	Joules per mAh	Normal rating	Stored energy
1.5V AAA cell	5.4J	1,200mAh	6,480J
1.5V AA cell	5.4J	2,700mAh	14,580J
1.5V C cell	5.4J	8,000mAh	42,300J
1.5V D cell	5.4J	12,000mAh	64,800J
3V CR1220 coin cell	10.8J	40mAh	432J
3V CR1620 coin cell	10.8J	80mAh	864J
3V CR2032 coin cell	10.8J	240mAh	2,592J
3.7V LiPo pack	13.7J	100mAh	1,370J
3.7V 18650 LiPo	13.7J	2,200mAh	30,140J

When you put cells in series, the Joules per millamp-hour add together. Each cell in a 4.5V 3xAA pack adds 5.4J, for a total of 16.2J per milliamp-hour. The total energy stored in the pack is 43,740J.

A 4.5V circuit that uses 10mA consumes 45mW of power, or 0.045J per second. A quick comparison with the pack's stored energy shows that the pack should last a little less than a million seconds, or about 270 hours.

It's okay to put alkaline cells or coin cells in series, but DO NOT PUT LIPO CELLS IN SERIES OR PARALLEL. LiPos are much more likely to damage each other, and their common failure modes include 'catching on fire' and 'exploding'. Only carefully selected and tested LiPo cells can work together safely.

Putting the units to work

Now let's go back to that example of an ESP8266 that wakes up once every ten minutes, uses 5mA to read a sensor for maybe 100ms, then makes a Wifi connection that uses 80mA for an average of, say, 12 seconds before shutting down again.

The ESP8266 runs at 3.3V, and let's assume that the sensor does too. 5mA @ 3.3V consumes 16.5mW or 0.016.5 Joules per second. The sensor is active for 100ms, so it uses a total of 0.00165 Joules during that time:

Item	Voltage	Current	Power	Time	Work	Amortized
Sensor	3.3V	5mA	16.5mW	0.1s	0.00165J	-

The Wifi connection uses 80mA @ 3.3V, which comes to 264mW or 0.264J/s. Running the connection for 12 seconds consumes 3.168 Joules.

Adding the two together, let's call the average work 3.17 Joules each time the ESP8266 wakes up:

Item	Voltage	Current	Power	Time	Work	Amortized
Sensor	3.3V	5mA	16.5mW	0.1s	0.00167J	-
Wifi	3.3V	80mA	264mW	12s	3.168J	-
Awake	-	-	-	-	3.17J	3.17J/cycle

The ESP8266 spends the rest of its time sleeping, and let's say it draws 70uA during that time.

70uA @ 3.3V is 231uW, or 231 millionths of a Joule per second.

The ESP8266 spends 12.1 seconds awake per 10-minute cycle, and sleeps the remaining 587.9 seconds. That means the ESP8266 uses 0.136 Joules per cycle sleeping.

Its total energy consumption per cycle is about 3.3 Joules:

Item	Voltage	Current	Power	Time	Work	Amortized
Sensor	3.3V	5mA	16.5mW	0.1s	0.00165J	-
Wifi	3.3V	80mA	264mW	12s	3.168J	-
Awake	-	-	-	-	3.17J	3.17J/cycle
Asleep	3.3V	70uA	231uW	587.9s	0.136J	0.136J/cycle
Sum	-	-	-	-	3.3J	3.3J/cycle

Now it's time to convert what we have into values that are more convenient to work with.

Each cycle is 600 seconds long and consumes 3.3 Joules of energy. If we just divide the energy by the time we get 0.0055 Joules per second, or an average power of 5.5mW.

If we divide 5.5mW by 3.3V, we get an average current of 1.67mA.

(running that through Ohm's Law, the system consumes about the same amount of energy as a 2k resistor that's always connected between 3.3V and GND)

We can also scale the 5.5mW up to an average rate of 19.8 Joules per hour:

Item	Voltage	Current	Power	Time	Work	Amortized
Sensor	3.3V	5mA	16.5mW	0.1s	0.00165J	-
Wifi	3.3V	80mA	264mW	12s	3.168J	-
Awake	-	-	-	-	3.17J	3.17J/cycle
Asleep	3.3V	70uA	231uW	587.9s	0.136J	0.136J/cycle
Sum	-	-	-	-	3.3J	3.3J/cycle
Equivalent	3.3V	1.67mA	5.5mW	3600s	19.8J	19.8J/hr

If we want to power the ESP8266 from the 3xAA battery pack that holds 43,740 Joules of energy, it's tempting to use that value of 19.8J/hr and think we can get more than 2,000 hours of battery life.

That's wrong though: the battery pack's voltage is 4.5V, while we've done all of our other calculations for a circuit running at 3.3V.

The difference in voltage has to be burned off as heat in the 3.3V regulator, and that's another kind of work.

That's where calculating the average current comes in:

The difference between 4.5V and 3.3V is 1.2V, and if we multiply that by the 1.67mA calculated above, we get 2mW of power consumption in the regulator.

Multiplying 0.002J/s by 3600 seconds tells us the voltage regulator uses about 7.2J per hour.

Adding that to the 19.8J/hr the ESP8266 uses gives us a total power consumption of 27J/hr from a 4.5V power source:

Item	Voltage	Current	Power	Time	Work	Amortized
Sensor	3.3V	5mA	16.5mW	0.1s	0.00165J	-
Wifi	3.3V	80mA	264mW	12s	3.168J	-
Awake	-	-	-	-	3.17J	3.17J/cycle
Asleep	3.3V	70uA	231uW	587.9s	0.136J	0.136J/cycle
Sum	-	-	-	-	3.3J	3.3J/cycle
Equivalent	3.3V	1.67mA	5.5mW	3600s	19.8J	19.8J/hr
Regulator	1.2V	1.67mA	2mW	3600s	7.2J	7.2J/hr
Sum	4.5V	1.67mA	7.5mW	3600s	27J	27J/hr

Based on that estimate, we can expect the 3xAA pack to last around 1620 hours (about 67 days):

Item	Voltage	Current	Power	Time	Work	Amortized
Sensor	3.3V	5mA	16.5mW	0.1s	0.00165J	-
Wifi	3.3V	80mA	264mW	12s	3.168J	-
Awake	-	-	-	-	3.17J	3.17J/cycle
Asleep	3.3V	70uA	231uW	587.9s	0.136J	0.136J/cycle
Sum	-	-	-	-	3.3J	3.3J/cycle
Equivalent	3.3V	1.67mA	5.5mW	3600s	19.8J	19.8J/hr
Regulator	1.2V	1.67mA	2mW	3600s	7.2J	7.2J/hr
Sum	4.5V	1.67mA	7.5mW	3600s	27J	27J/hr
Project	4.5V	-	-	1620hr	43,740J	27J/hr
3 x AA pack	4.5V	-	-	-	43,740J	-

Learning from the results

The most interesting thing all that tells us is that we're wasting a lot of energy in the voltage regulator.. almost 40% of the total.

If we switch to a 2200mAh 3.7V 18650 LiPo, the 3.3V voltage regulator will only have to drop 0.4V.

Using the same 1.67mA estimate for the average current, that reduces the regulator's power consumption to 670uW, which scales up to about 2.4J/hr.

Adding that to the ESP8266's consumption of 19.8J/hr gives us a total of 22.2J/hr:

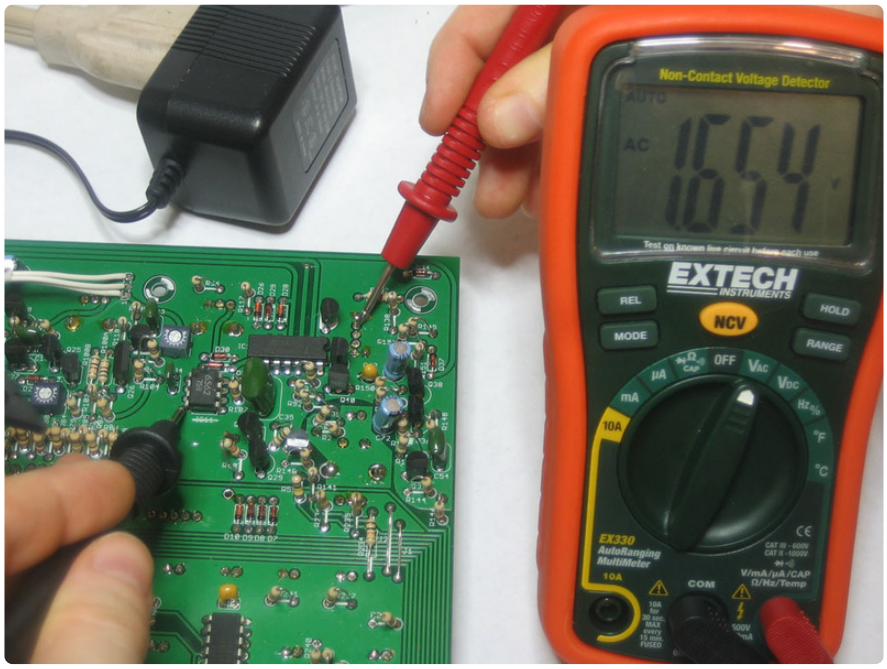
Item	Voltage	Current	Power	Time	Work	Amortized
Sensor	3.3V	5mA	16.5mW	0.1s	0.00165J	-
Wifi	3.3V	80mA	264mW	12s	3.168J	-
Awake	-	-	-	-	3.17J	3.17J/cycle
Asleep	3.3V	70uA	231uW	587.9s	0.136J	0.136J/cycle
Sum	-	-	-	-	3.3J	3.3J/cycle
Equivalent	3.3V	1.67mA	5.5mW	3600s	19.8J	19.8J/hr
Regulator	0.4V	1.67mA	668uW	3600s	2.4J	2.4J/hr
Sum	3.7V	1.67mA	6.168mW	3600s	22.2J	22.2J/hr
Project	3.7V	-	-	1357hr	30,125J	22.2J/hr
18650 LiPo	3.7V	-	-	-	30,135J	-

The 18650 holds 30,140J, so we can expect it to last around 1357 hours (about 56 days)

We still lose energy in the voltage regulator, but only about 11% of the total.

The LiPo doesn't last quite as long as the 3xAA pack, but the 3xAA pack wastes almost four times as much energy as the LiPo.

And now the process



So what the heck just happened?

You've seen me do some energy calculations and a sample energy budget, throwing numbers all over the place as I went. Even if you were able to follow along step by step, it's reasonable to wonder how I got from the beginning to the end.

There’s a method to the madness, but it can get lost behind all the details.

In fact, the basic idea of doing an energy budget is to get rid of details that make it hard to guess how much energy a circuit needs over time.

The main offenders are duty cycles and pieces operating at different voltages. All the number shuffling serves the purpose of making those values line up better.

Here’s a high-level description of the process:

Start with what you have, and calculate power.

Circuits are defined in terms of their voltage and current, and those values are usually easy to find. If they aren't in a datasheet or some other documentation, you can hook up a multimeter and measure them directly.

They’re what you have to start with, so do the only calculation you can: power.. the amount of energy that moves from one place to another while the circuit is in operation.

Use power and time to calculate work.

If parts of your circuit only consume power some of the time, use those times for the next calculation available to you: work.. the amount of energy the circuit uses in a given amount of time.

The time values are almost entirely under your control. There are some operational constraints.. it takes a certain amount of time to establish a Wifi connection and transmit data to the internet.. but you have enormous freedom to make tradeoffs.

Many times, you'll want to do energy budgets to compare different timing options for the project you're building. The usual question is, "do I want this feature enough to accept what it does to overall battery life?"

Use work to calculate average power.

This is where we get rid of the details related to time and duty cycles. Reducing "on for this long, off for that long" to average power during the whole period condenses two currents and two time values into a single, more general number.

The details aren't really gone of course, they're just all mushed together. You'll have to calculate a new average power value any time you change a duty cycle.

Use average power to calculate average current.

Once you know the average power, you can use it to calculate the average current.

Average power and average current are intermediate products in an energy budget.. values we calculate on the way to finding some value we really want.. but they're also handy on their own.

Circuit designers like to use values called 'figures of merit' that serve as a quick shorthand for comparing one circuit to another. Average power and average current are useful figures of merit. You'll probably find yourself using them automatically once you get used to the math.

Use average current to match voltages.

Once you have an average current, you can use it to fill in any differences in voltage.

This step is mostly about finding missing pieces: if you have a 3.3V circuit running from a 5V power supply, there has to be a 1.7V piece somewhere.

That missing piece is usually the voltage regulator: an important but easily ignored part of any circuit.

Most circuits use either linear regulators or switching regulators.

Linear regulators are basically smart resistors that adjust their own resistance to keep the output voltage where it should be. A linear regulator's power consumption is also the same as a resistor's: the average current times the voltage between its input and output terminals.

Switching regulators work the same way as a PowerBoost, with duty cycles that turn a higher input voltage to a lower output voltage. And like the PowerBoost, they're constant-power devices with some efficiency loss. The loss is the only energy the the regulator consumes itself, and that's just some fraction of the average power.

It doesn't hurt to run the numbers on a switching regulator's duty cycle and instantaneous current though. Those will tell you what kind of demands the regulator will make on the battery.

Find the average power for the whole system

Once you've found all the missing voltages and worked out the power used by the voltage regulators, you can add the pieces together and get a single figure of merit for the whole system: the average power it consumes when operating at a given supply voltage.

Use the average power to find battery life

That sequence of twists and turns produces a value you can compare to a battery's stored energy: power (Joules per second) calculated for the battery's output voltage.

There are two ways you can make the final comparison to a battery's milliamp-hour rating:

The first is to use the average power and the voltage to calculate the average current. That will give you a value whose units are milliamps, which divide into milliamp-hours to give you an estimate for battery life.

The second is to multiply the battery's voltage (Joules per Coulomb) by its milliamp-hour rating (a complicated way to say Joules) to get the battery's stored energy in Joules. Divide that by the average power to get the estimated battery life.

Each is about as easy as the other. I happen to prefer working with standard units, but use whatever feels most natural to you.