With Great Power Comes Great Complexity

Powering Microcontroller Devices

~~Uncle Ben~~ ~~Voltaire~~ Me

Powering Microcontrollers

We just plug in a USB cable; problem solved!

Why does this work for most devices?

Most MCUs run between 1.8V and 5V

- ATmega 3.8P – ***1.8 - 5.5V***
- ESP32 – ***2.3 - 3.6V***
- PIC16/18 – ***4.2 - 5.5v***
- Raspberry Pi/Pico – ***5v (3.3v)***

- Speed Grade:
  - 0 - 4MHz @ 1.8 - 5.5V
  - 0 - 10MHz @ 2.7 - 5.5.V
  - 0 - 20MHz @ 4.5 - 5.5V

Why do we have different voltages?

Most modern circuitry uses low voltages as transistors are smaller, and spacing is tighter

Reduced voltage = reduced current draw = reduced heat

Base-emitter junctions in early TTL logic required at least 3V, but switching was faster at higher voltages. Given that 6V caused leakages, a value of 5V was chosen

Users of Power

When we design a circuit it's not generally just an MCU, but also peripherals

MCU bundled:

- Wired communications devices
- GPIO ports
- Data storage
- Wireless communications devices

External logic:

- Buttons
- Displays
- Sensors

Why do we care about power consumption?

There are many reasons to reduce power consumption

- Reduce heat dissipation
- Increase our component lifespan
- Reduce the cost of power components

If we intend using our devices in a mobile manner:

- Increase the time our device can operate on a single battery charge
- Make it truly mobile

Battery Powering our Devices

Considerations:

Battery capacity

- CR2032 (3v) – *225mAh*
- 603450 Li-Ion (3.7v) – *1200mAh*
- 18650 (3.7v) – *2000mAh*
- *26650 (3.2V) – 3000mAh*
- PP3 (9v) – *700mAh*
- AA (1.5v) – *2800mAh*

Technology:

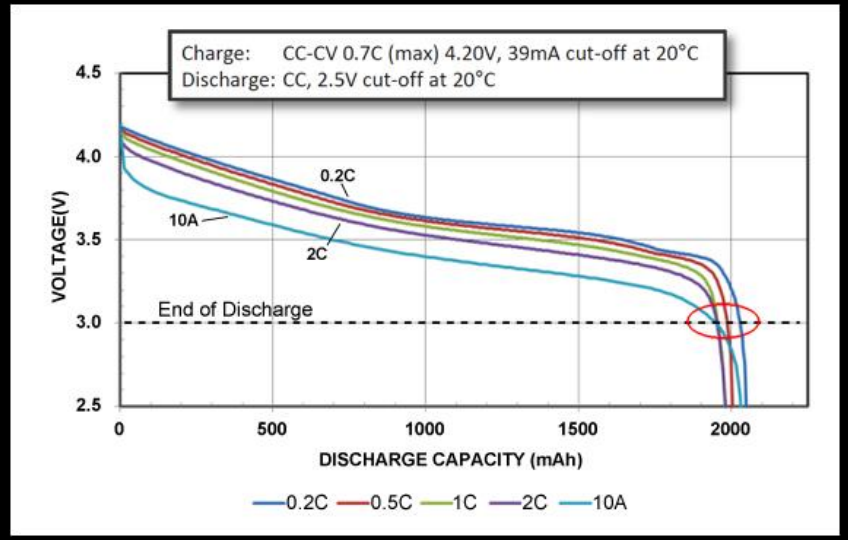
- Alkaline for cheap user replaceable applications
- Lithium-Polymer (LiFePO) for high current applications
- Lithium-ion
- Lithium Iron Phosphate ($LiFePO^4$)

Where is my 3.3V battery?

Most battery technologies don't provide the voltage we need for our circuits, and almost no power source can provide a constant voltage under changing loads and varying states of discharge.

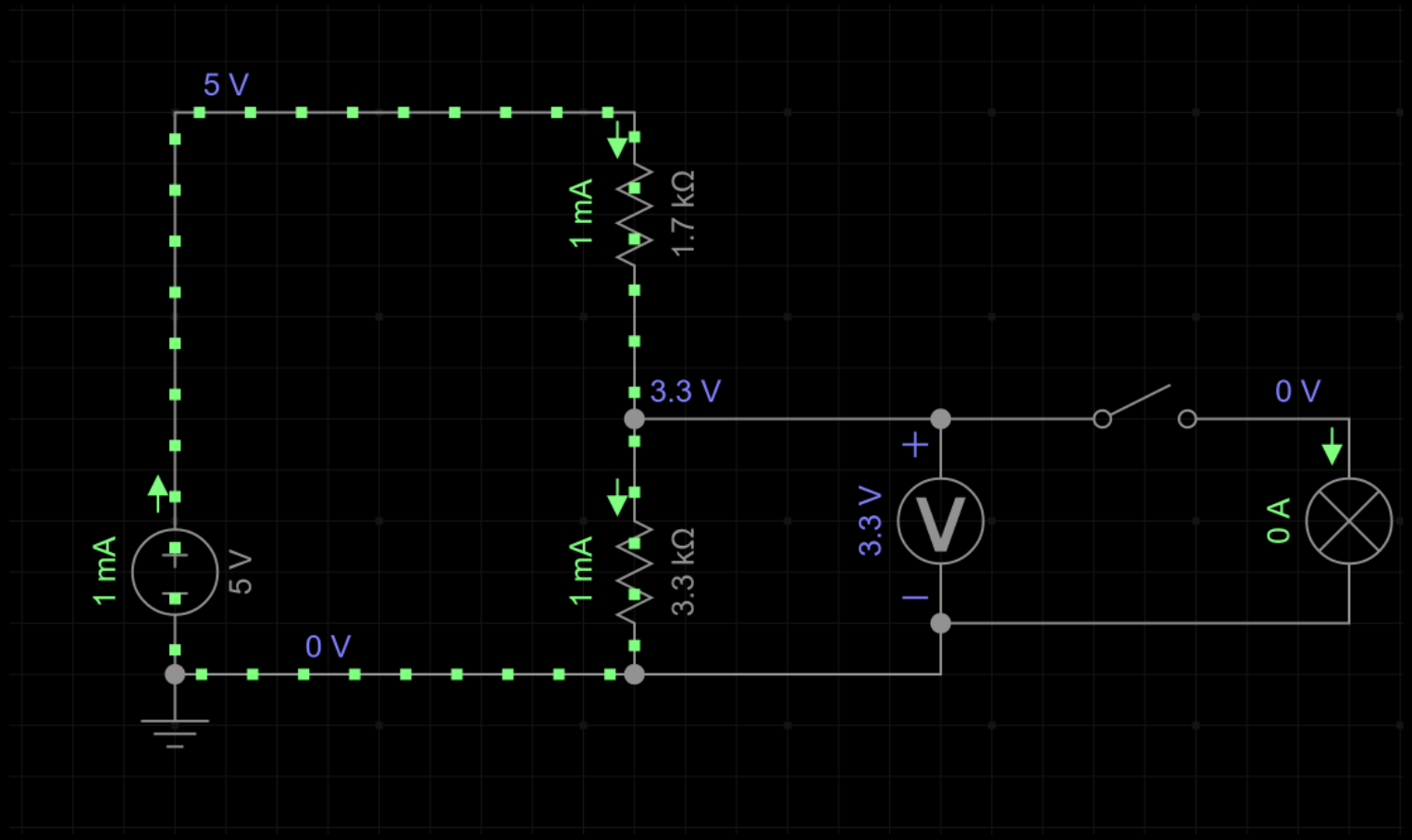
Regulation

- What do we need to do with our voltages?
  - Increase or decrease the voltage to achieve 3.3V
- Considerations:
  - Efficiency of the circuit at different loads
  - Complexity/cost



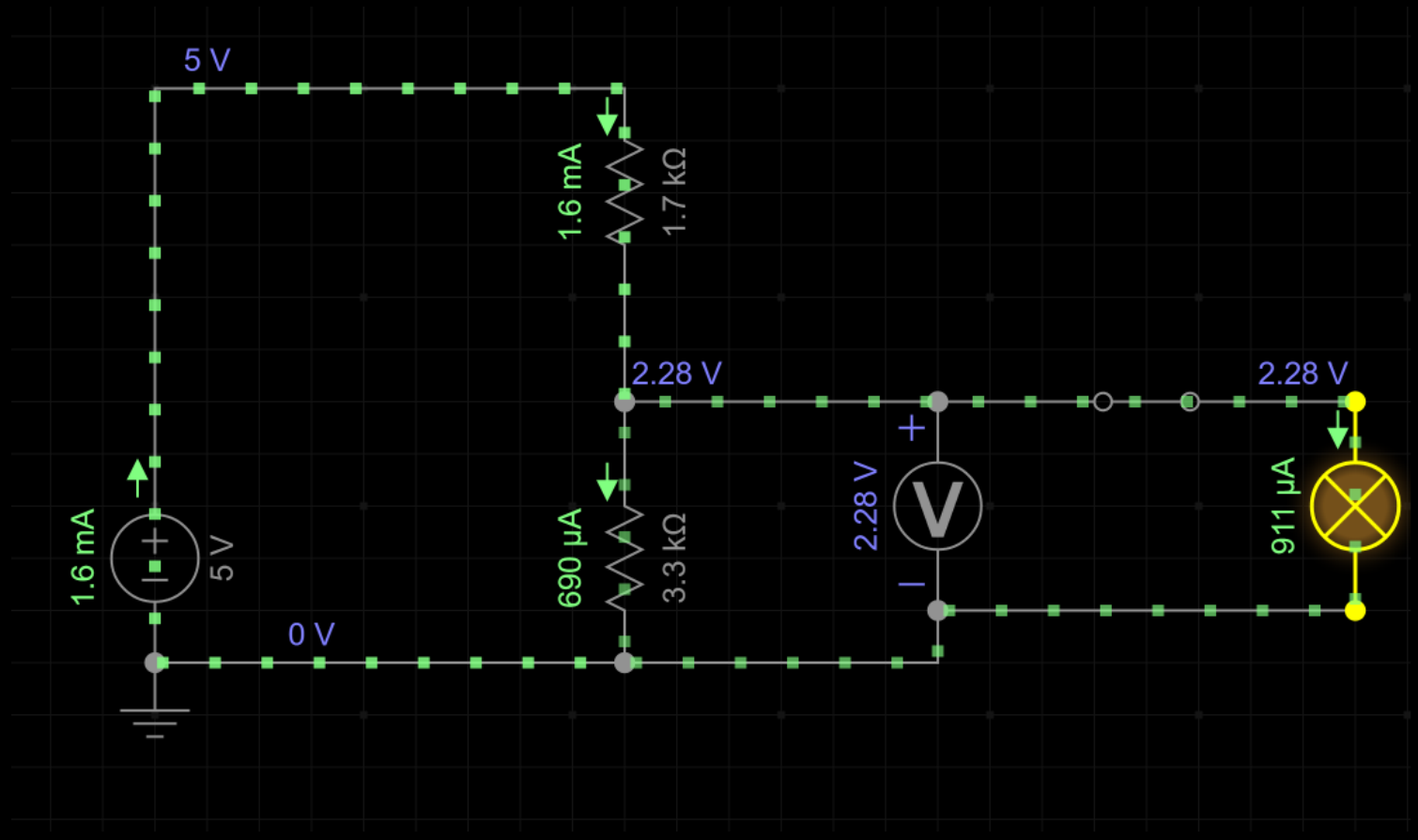
Regulation? Just use a couple of resistors!

The quiescent current draw here is only 1mA:



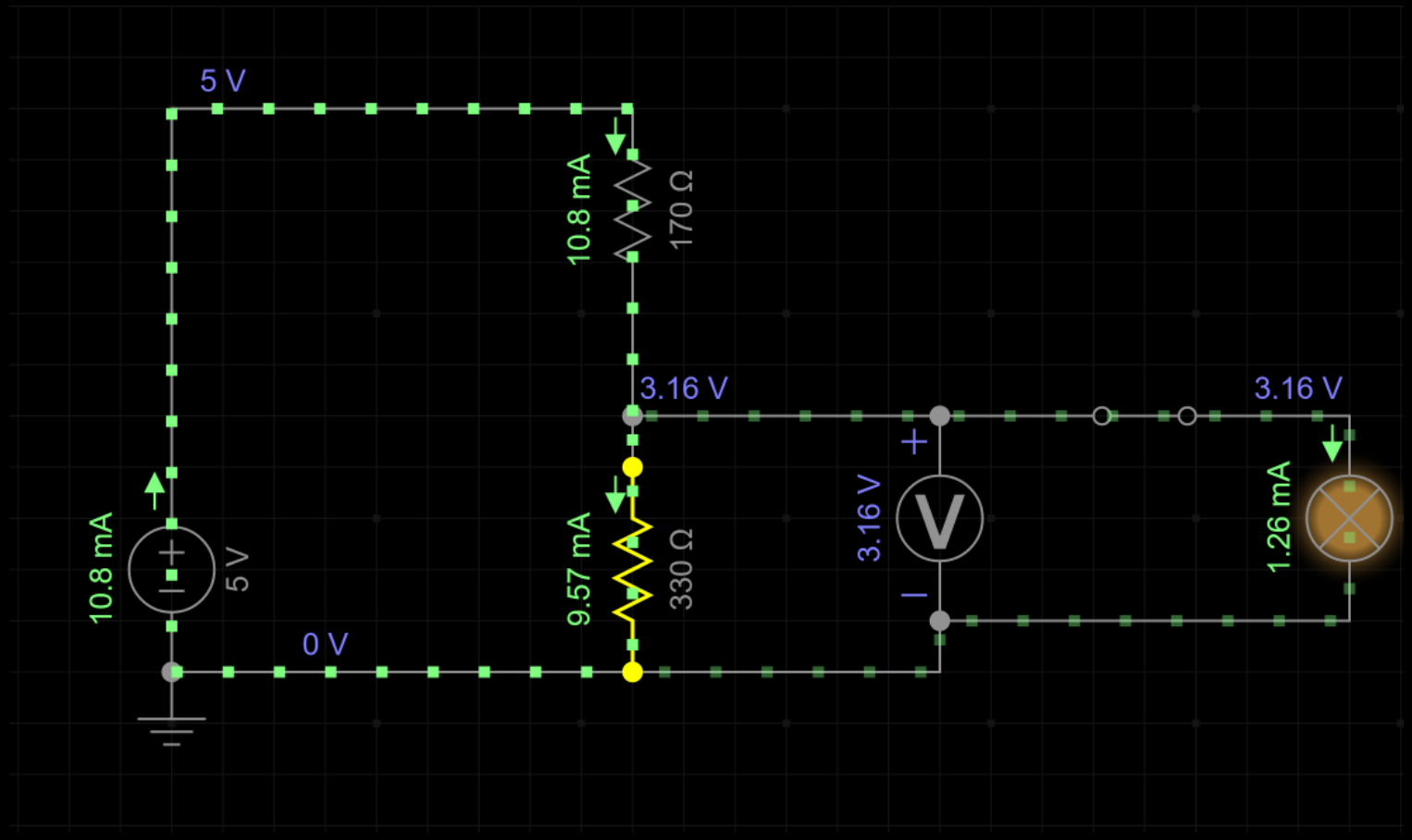
Regulation? Just use a couple of resistors!

Let’s add a load…



Regulation? Just use a couple of resistors!

Let’s reduce the resistance of the divider, so the load has less impact…



What regulation do we need?

Do we need to reduce the voltage or raise the voltage?

Reduction appears simple. For example, a 78xx family regulator is cheap and simple, and there are dozens of alternative components.

- Current limits
  - What is the max sustained current we need to supply?
  - The ESP32 spikes up to 250mA with all GPIO and all peripherals active, but we can use capacitors to handle transient spikes
- Drop out voltage
  - How much more than our desired output voltage do we have to provide the regulator with? The 78xx series recommends 3V!
- Thermals
  - To regulate a 12V supply down to 5V, how much energy is lost as heat?

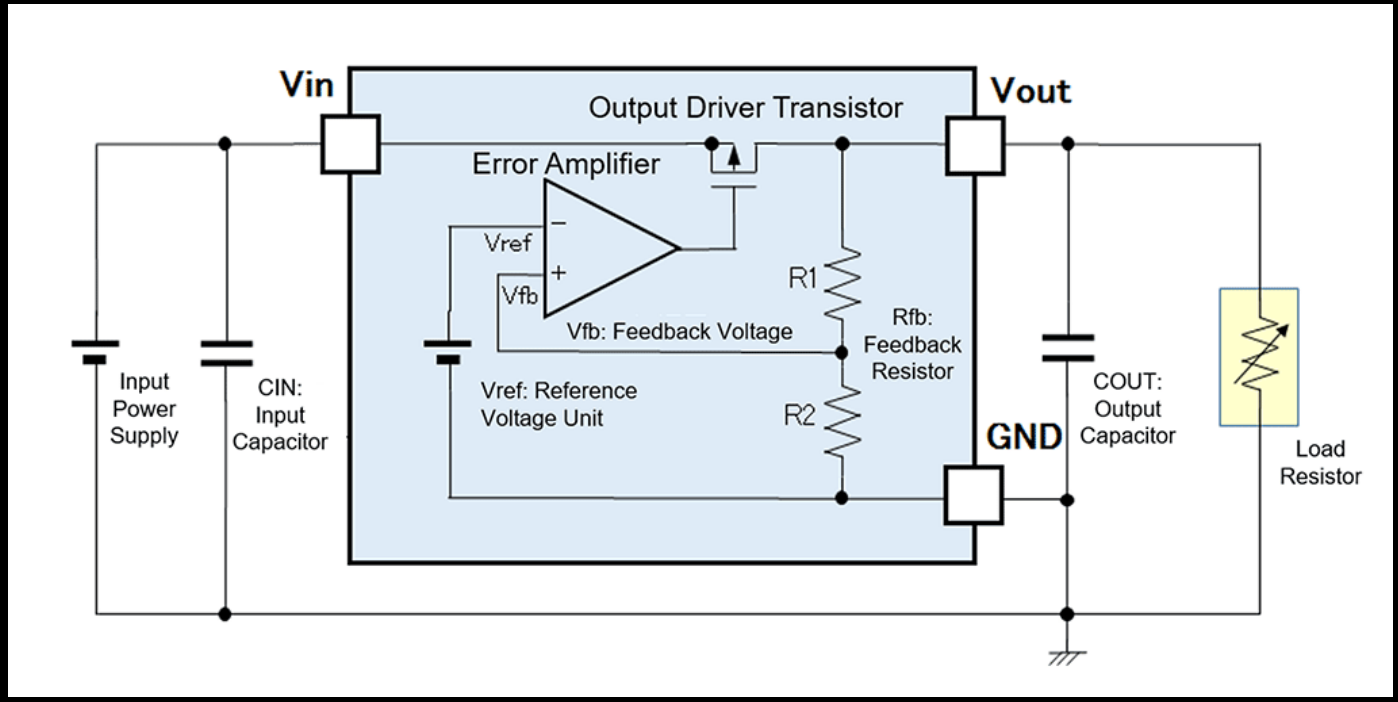
Sample 3.3V Linear Regulator

LD1117V33 - 3.3V 800mA TO-220 Low Drop Out Voltage Regulator

Specifications

Input Voltage MAX: 15 V
Output Voltage: 3.3 V
Dropout Voltage - Max: 1.1 V
Output Current: 800mA Min, 1.3 A Max
Load Regulation: 10 mV
Number of Outputs: 1 Output
Output Type: Fixed
Maximum Operating Temperature: + 150 C
Mounting Style: Through Hole
Package / Case: TO-220-3
Input Bias Current - Max: 5 mA
Line Regulation: 6 mV
Maximum Power Dissipation: 12 W
Minimum Operating Temperature: 0 C
Packaging: Tube
Series: LD1117V
Voltage Regulation Accuracy: 1 %

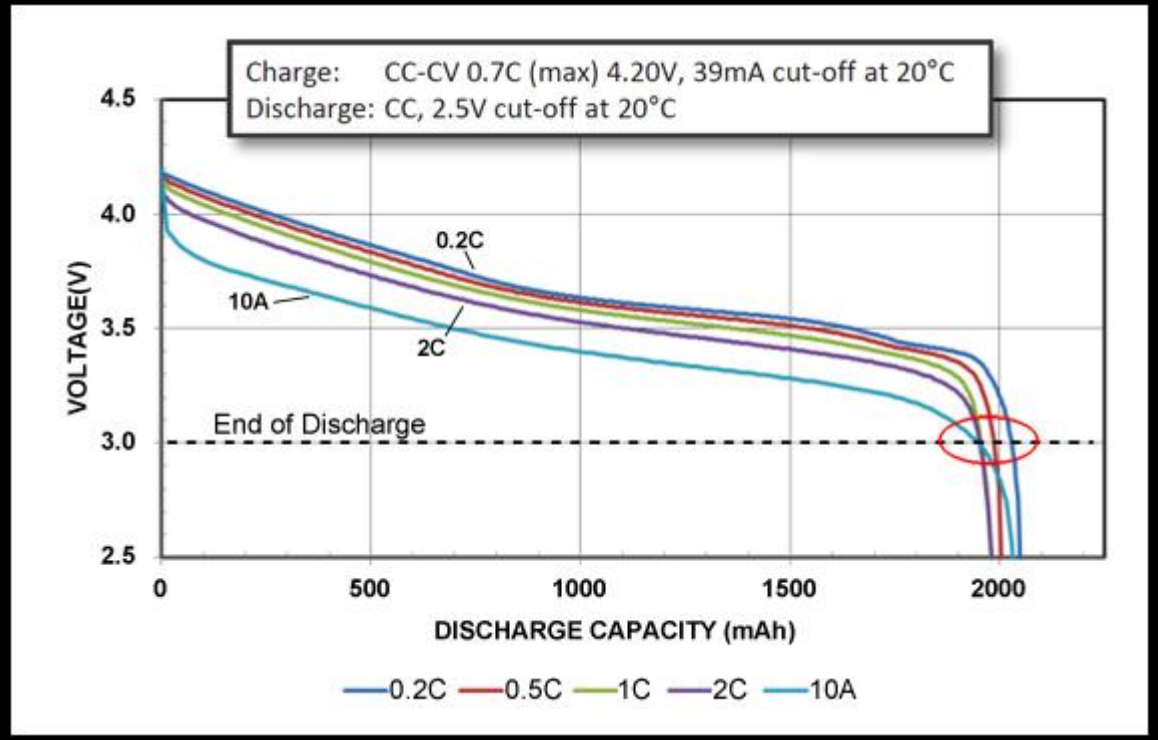
*Specifications are subject to change without notice.



What if I have a 3.7V battery?

At lower currents we get over 3.3V for over 80% of the battery's discharge range.

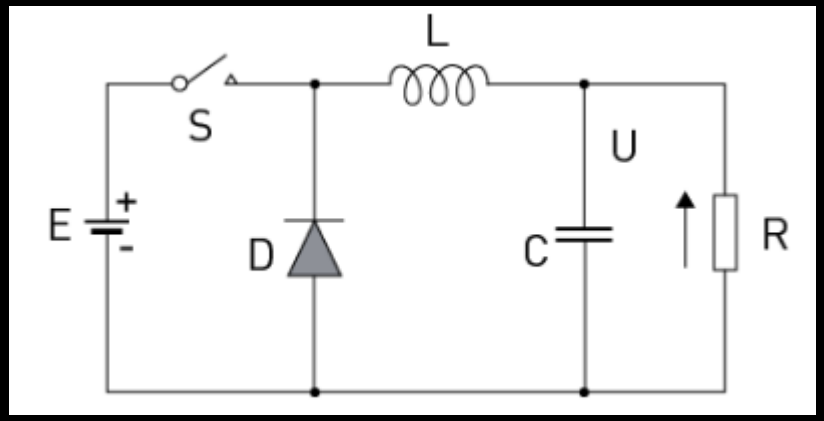
With a 1V dropout regulator, we can see from this chart that we will never be able to generate 3.3V, so we need another option.



Buck Convertors

A Buck Converter lowers the input voltage to a regulated output value by rapidly turning on and off the input voltage through a switching transistor (using an internal reference to adjust the ratio of on and off), then feeding the output through and inductor, diode, and capacitor (flywheel circuit) to provide a smooth output.

The switching in the convertor can be running at over 20KHz.

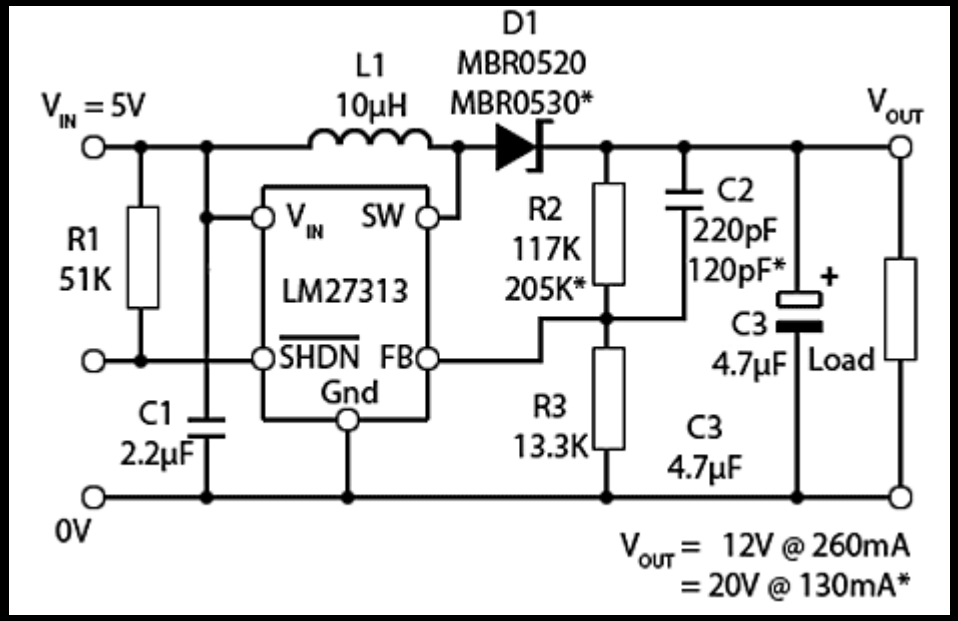


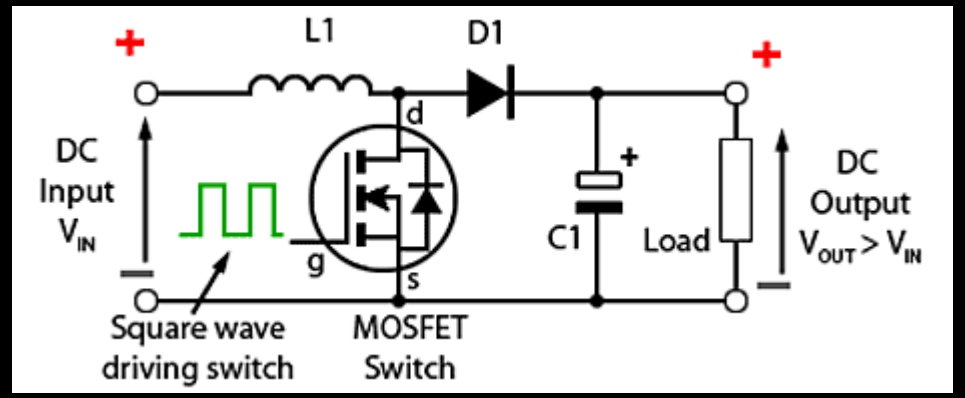
Boost Convertors

A Boost Converter raises the input voltage to a regulated output value by rapidly turning on and off a power MOSFET at the input. This is used to store energy in an inductor, then release it as back-EMF, charging up the output capacitor and providing a higher voltage than at the input.

Boost converters can have internal oscillators running at 1.6MHz

Here's the internal schematic for an LM27313:



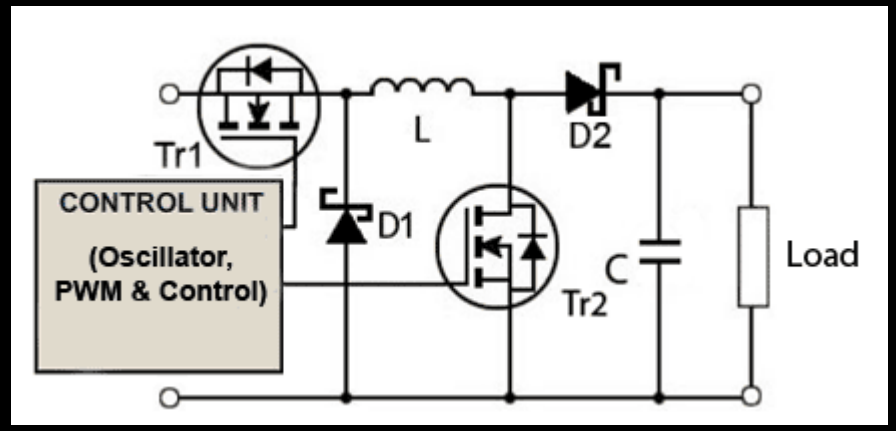


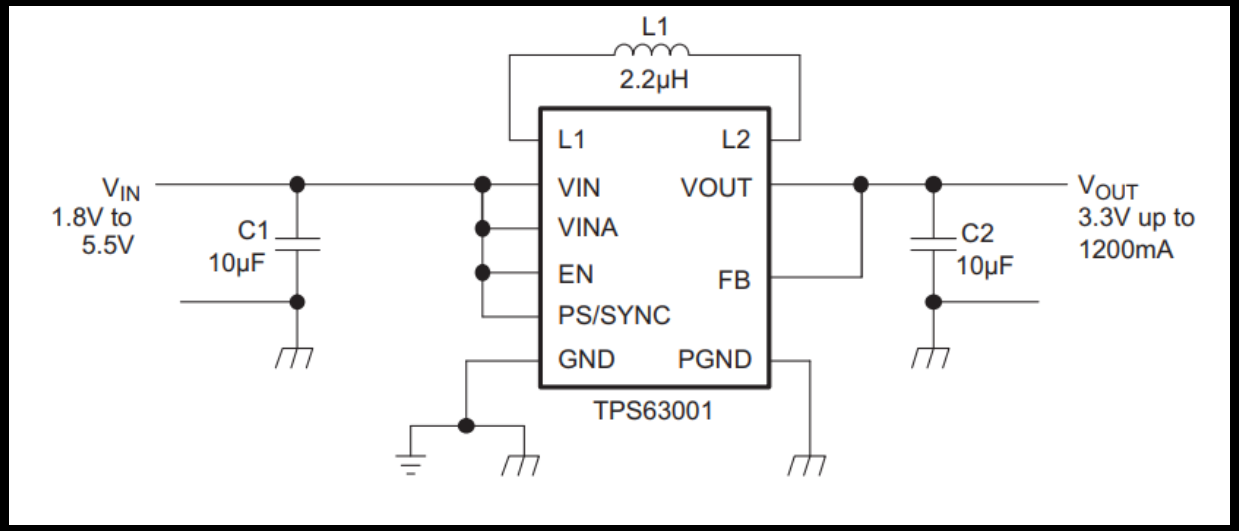
Buck-Boost Convertors

A Buck-Boost Converter combines the previous two principles so allow raising or reducing the input voltage.

Boost converters can have internal oscillators running at 1.6MHz

Here's a schematic for the application of the TI TPS63000, which can deliver 1.8A from an input voltage as low as 2.5V with up to 96% efficiency.





Issues and Considerations

- Efficiency
  - Battery life
  - Heat dissipation
- RF interference
  - EMI generation
- Cost
- Circuit real-estate
  - Support components
- Complexity of design
  - There’s nothing simpler than a 3-pin linear regulator!

Reducing power consumption

Once you have your hardware capable of running efficiently, the best place to make power savings is in your software design.

This requires significant attention to your data management processes, and the workflows that your device will follow in different situations.

- Do we sit in a loop checking for button presses?
- How often do we need to communicate externally?
- Do we perform actions periodically? Every second? Every hour?
- Can external devices trigger behaviour in our system?

We'll use the ESP32 as an example, but you'll find most of these features (or similar) in all manufacturers' models of MCU.

Sleep Mode

- Light Sleep
  - Peripherals are powered off
  - CPU RAM content is retained
  - CPU is placed into a halted state
  - Waking the device continues execution from where it was halted
- Deep Sleep
  - Peripherals are powered off
  - CPU RAM content is lost
  - CPU is placed into a halted state
  - Waking the device causes a standard boot process

Most MCU's have some form of Low Power CPU to act as an orchestrator, and other low power peripherals such as counters and timers that run independently of the main CPU, and that consume microamps instead of milliamps.

In the ESP32 this is called the ULP coprocessor and it contains the RTC.

Peripherals

When you're not using a peripheral, turn it off!

When in sleep mode, the ESP32 will disable most communications peripherals, and will also no longer be managing the pull-up and pull-down state of GPIOs*.

When you awake from light or deep sleep it's your responsibility to reinitialise the peripherals. Wake up your WiFi, re-establish network functionality, setup UARTs etc.

You may also want to initialise your own external peripherals, which is why it's important to provide power switching on any external circuitry. Use a GPIO line to activate an external display, enable a stepper motor controller, or provide power to a level sensor.

Timing

If we need to wake up once per day, use the built-in countdown timers or real-time clocks in our device.

Most devices have a single countdown timer for periodic awakening, so careful management is needed if we want to simulate multiple timers.

Consider how long we need to be awake. Can we send out some data, then sleep, or do we have to wait for a response? How long do we wait?

Battery consumption is directly proportional to the time we are using our devices.

If we poll an external WiFi device (assume 100mA) every second, waiting for a response for 250ms, we are drawing the equivalent of a constant 25mA, giving us less than 80 *theoretical* hours (3 days) from a 18650 battery.
If we reduce the polling time to every 5 seconds, and expect a response within 100ms, we are drawing 2mA, resulting in 1000 *theoretical* hours (40 days) of battery life.

In theory…

Interrupts

Interrupts are your friend!!

- Interrupts are External or Internal
- Internal (within each core):
  - Timer comparators (core specific)
  - Performance monitors (watchdog)
  - Software
- External:
  - GPIO – low, high, change, falling, rising
  - UART data
  - RTC timer events
  - Touch inputs

External circuits

Did you turn off the bedroom lights?

If you add circuitry, make sure you can turn it off!

Amplifier circuits

External displays

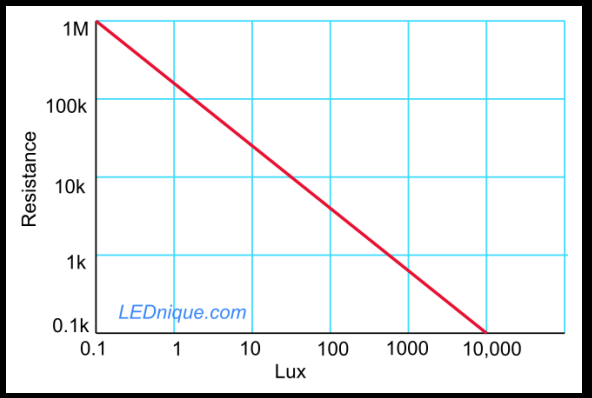
Sensor chips

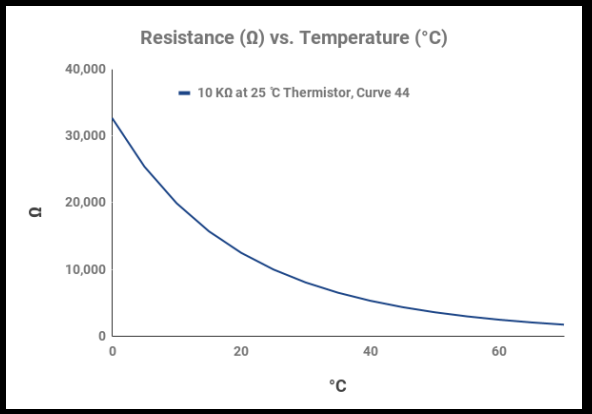
Potential dividers

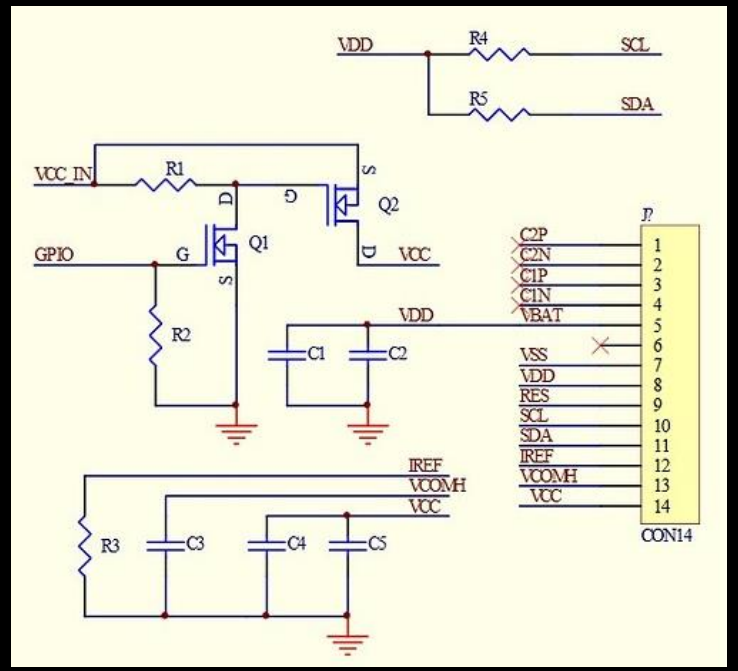
MOSFETs are your friend!

Make sure V_{DS} is minimal

$V_{DS(min)}=I_D \times R_{DS(on)}$







Reducing power consumption

(A.k.a. Avoiding the need for big batteries)

- Sleep modes
- Peripherals
- Timing
- Interrupts
- External circuits
- Pull-up resistors
- GPIO sleep-mode states

Design Scenarios

- I have designed a remote keyboard, and when a button is pressed, I want to send it over WiFi
- I add a display to the keyboard that will indicate whether CAPS LOCK is on
- I have a device to measure my water meter usage. The meter contains a rotating magnet for every 100L of water used
- How much water is in my water-tank?