HOW TO BUILD AN IOT DEVICE WITH LOW-POWER SLEEP

Posted on 27th November 2023 by Evette Ferrao in blog

Reading Time: 10 minutes
Recommended citation: Haßler, B. (2023). How to build an IoT device with low-power sleep. Open Development & Education. https://doi.org/10.53832/opendeved.1011

At OpenDevEd, we're committed to sharing our learning with the wider development community. This article is one such attempt. Learnings are often very focused insights, and are necessaril very specific. However, rather than trying to second guess what you, dear reader, may or may *find interesting, we share such posts nonetheless. If a post is too specific for your liking, you'll no doubt move on to another one. It has taken the authors of this blog several hours to reach the insights here, and we feel it's worth sharing. So, if you're interested in microcontrollers and low-power sleep, read on.*

Recent programme work at OpenDevEd has led us to focus on power considerations for 'Internet of Things' (IoT) devices used in the context of environmental research in East Africa, such as the Improving Learning Through Classroom Experience programme.

By low power, we mean devices that can operate continuously for long periods of time on battery power.

That in itself is perhaps not a difficult thing to do. However, we have the additional requirement for devices that can be produced, installed, and maintained at low cost, particularly in the context of low- and middle-income countries.

A good example of what we want is a television remote control: a device that is activated when something needs doing (i.e., when buttons are pressed to change a programme), but that otherwise 'sleeps', so that its batteries will last for several years. Our strategy is to build a device that allows for data monitoring and measurement (and potentially transmission) at particular times, not continuously. Typically, a measurement would only take 5–10 seconds, and measurements would be taken, for example, every 15 minutes. Between measurements, the device needs to enter a low-power 'deep sleep', consuming as little battery as possible. The idea of 'deep sleep' is that the power is minimised, but still allows the device to wake up later.

This article focuses on the importance of low-power sleep. Future articles will examine other aspects.

CONTEXT

OpenDevEd is part of a team of researchers implementing the Improving Learning Through Classroom Experience programme in Tanzania, funded by the UK Foreign, Commonwealth and Development Office. Our work focuses on practical approaches to improving students' learning experience through cost-effective and sustainable infrastructural development. In particular, we explore options for creating better classroom environments. We consider 'indoor environmental quality', with a particular focus on managing temperature, light, and sound.

Effectively monitoring factors pertinent to classroom environmental quality – such as temperature – could be done in-person, perhaps using a hand-held device; alternatively, autonomous sensors could be used. However, the availability of suitable sensors be a challenge: factors such as cost, ease of use, and power can limit the utility of sensor equipment.

In particular, it is important to ensure that the sensors have an effective deep sleep mode, saving battery while ensuring that timestamps on collected data are preserved to understand fluctuations in temperature over daily, seasonal, and annual cycles. Therefore, we have been experimenting with different low-cost, low-power devices for accurate and sustainable use within the context of the Improving Learning Through Classroom Experience programme.

DATA COLLECTION

As noted above, we're looking to develop a device that would take short measurements (for a duration of 5–10 seconds), and then go to sleep. This process is sometimes called 'strobing': just like a strobe light, there's a flash (the measurement), then 'darkness' for a period of time, i.e., 'low-power deep sleep'.

The following figure illustrates this process, by plotting the power consumption of a Raspberry Pi Pico in a setup that we will discuss below. In the graph, you can see that the Pico is active for a short period, consuming around 25 mA (milliamps). The consumption then drops to less than 5 mA during 'sleep'.

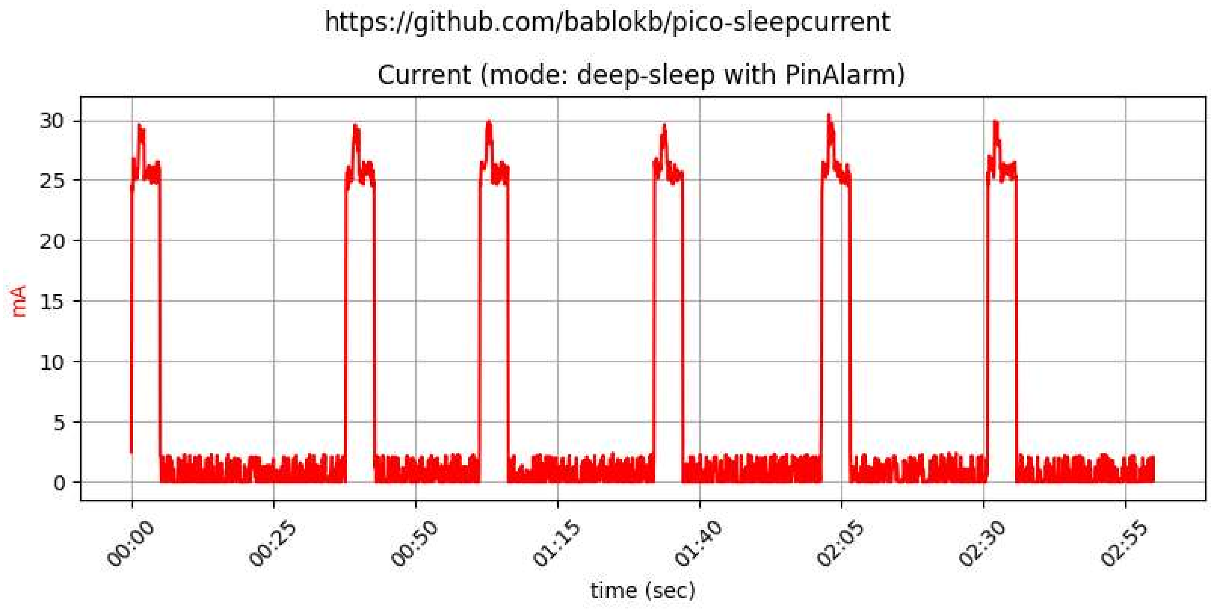

Figure 1. Power usage for a basic Raspberry Pi Pico device with an alarm function produced by GitHub user bablokb

DEVICE OPTIONS

With a vast increase in low-cost, 'do-it-yourself' computing devices, there are several different options that could be used as the basis of a low-power device.

It is important to realise that power consumption during sleep varies greatly depending on the facilities of the microcontroller used. There are various types of 'sleep' that are 'induced' in different ways. Broadly speaking, there are two strategies:

1. One strategy is to use the microcontrollers' own provision for sleeping ('internal sleep'); that is, the microcontroller unit at the heart of the device has a way of sleeping and waking up again.

However, this typically doesn't save much power.

2. Another strategy is to use an external 'clock' which wakes up the microcontrollers; for this, the microcontrollers can be programmed to wait for an external input to wake it up (via a physical pin connection).

Let's consider some devices.

RASPBERRY PI PICO

The Raspberry Pi Pico is a widely used low-power device, centred on a modular, open design circuit board (which, as of late 2022, has begun production in Kenya). The board uses around 50 mA during normal operation. It has internal sleep modes. Adafruit's Dan Halbert notes that *"when avoiding TimeAlarms, the Raspberry Pi Pico devboard can achieve under 2mA (1.4mA) while in deep sleep"*, but that *"when a Timealarm is added, the draw raises up to 7mA"*.

We'll now do a calculation that we'll repeat several times below, which puts this consumption into context. A standard alkaline AA battery has a capacity of around 2,700 mAh, which, at a consumption of 7mA, would last for about 16 days (2,700 mAh / 7 mA / 24 hours = 16 days). Clearly, this is not particularly long for an unattended device intended to provide continuous data, such as measuring temperature at planned intervals during both daily and weekly cycles.

THE ADAFRUIT PICOWBELL ADALOGGER

Building on the modularity of the Raspberry Pi ecosystem, Adafruit's PiCowbell Adalogger is an add-on piece of hardware incorporating a real-time clock. This allows the power source for the Pico itself to be changed without losing accurate timestamps on collected sensor data. However, the PiCowbell board uses the 'EN (3V3_EN)' pin:

EN (3V3_EN) – This connects to the enable pin on the Raspberry Pi Pico, and is pulled high (to VSYS) via a 100kΩ resistor.

This means that the power consumption is not very different from the Pico's power consumption (https://forums.adafruit.com/viewtopic.php?t=201034, https://forums.adafruit.com/viewtopic.php?p=970723#p970723, https://learn.adafruit.com/adafruit-picowbell-adalogger-for-pico). In other words, while the Picobell adds an 'external clock', there isn't much gain in power consumption over what the Pico already offers.

However, the PiCowbell does also provide an SD card slot, addressing the Pico's problem of limited available data storage space: with the MicroPython programming system installed, only 848 kB of space remains for programmes and data. (Note that the SD card is supported in the firmware image provided on the CircuitPython website. In MicroPython, this will require an alternative code.)

ESPRESSIF ESP32

Although modular and inexpensive, the Raspberry Pi Pico does seem to be inefficient when it comes to sleep, particularly when compared to lower power microcontrollers, such as the Espressif Systems ESP32 microcontroller. In standby mode, the ESP32 consumes only 0.25 mA, or 250 μA (microamps). With a consumption of 0.25 mA, an alkaline battery would last for 10,800 hours, or well over a year (2,700 mAh / 0.25 mA / 24 hours = 450 days).

With standby power consumption that low, the device becomes suitable for our application. However, while the Espressif ESP32 chip is very widely used in industrial low-power applications, the ecosystem felt too specialised for our purposes. By contrast, the Raspberry Pi Pico ecosystem is very maker-friendly. This is important because we want to keep development costs to a minimum. Therefore, we decided to find a solution using the Raspberry Pi platform.

Moreover, as we'll see below, it can lower the stand-by power consumption even further than what the ESP32 offers.

CIRCUITS FOR LOW-POWER SLEEP

Clearly, the crux of extending battery life in a low-power device, used for strobe-type measurements, centres on the efficiency of its sleep cycle. So, how can we reduce power consumption during sleep? The Raspberry Pi Pico cannot run much below a current of 2 mA on its own internal sleep functions. However, the Pico is a simple device, which can turn on and off quickly; therefore, one solution is to simply turn off the Pico when not in use. That is to say, rather than relying on the Pico's own sleep function, can we design a circuit that would turn the Pico off when it's not needed?

Pimoroni offers some devices that support low-power operation. Our tests of Pimoroni's Enviro Indoor (Pico W Aboard) (a Pico-based environmental monitoring board) and the Badger 2040 display system showed currents of around 30 µA. That consumption is over 46 times lower than the standard Pico system, and 8 times lower than the ESP32. This low-power consumption is achieved by cutting power to the Pico altogether, using an external circuit. The circuit uses a real-time clock circuit that can wake the Pico up as needed.

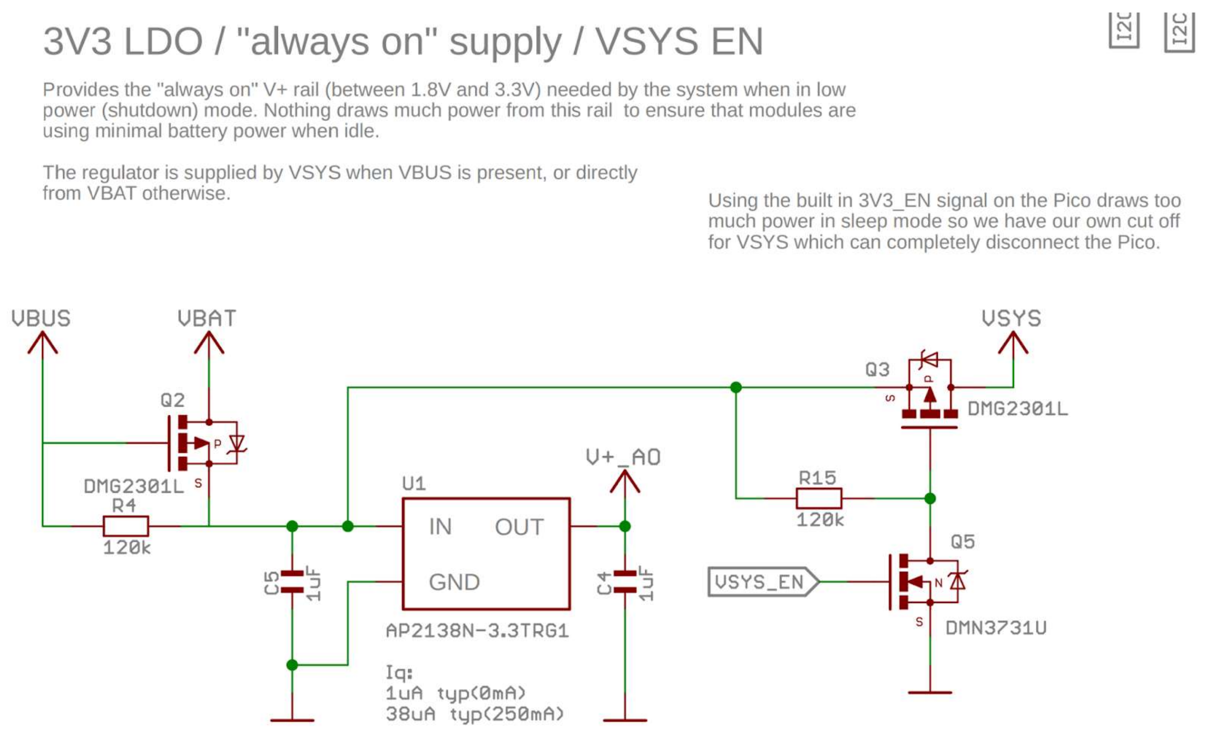
3V3 LDO / "always on" supply / VSYS EN
Provides the "always on" V+ rail (between 1.8V and 3.3V) needed by the system when in low power (shutdown) mode. Nothing draws much power from this rail to ensure that modules are using minimal battery power when idle.
The regulator is supplied by VSYS when VBUS is present, or directly from VBAT otherwise.
Using the built in 3V3_EN signal on the Pico draws too much power in sleep mode so we have our own cut off for VSYS which can completely disconnect the Pico.
I2C
I2C
VBUS
VBAT
VSYS
Q2
DMG2301L
R4
120k
Q3
DMG2301L
V+_AO
U1
IN
OUT
GND
AP2138N-3.3TRG1
Iq:
1uA typ(0mA)
38uA typ(250mA)
C5
1uF
C4
1uF
R15
120k
Q5
VSYS_EN
DMN3731U

Figure 2. Excerpt from the schematic of the Enviro Indoor showing the arrangement of the device's power supply control.

Therefore, just for standby, using an AA battery, the calculation for battery life would give about 10 years of operation (2,700 mAh / 0.03 mA (or 30 µA) / 24 hours = 3,750 days, or just over 10 years).

An example for a separate add-on, using the same principle, is here (again courtesy of GitHub user bablokb). Compared to Figure 1, we now get a figure as shown below, where the standby current drops much lower.

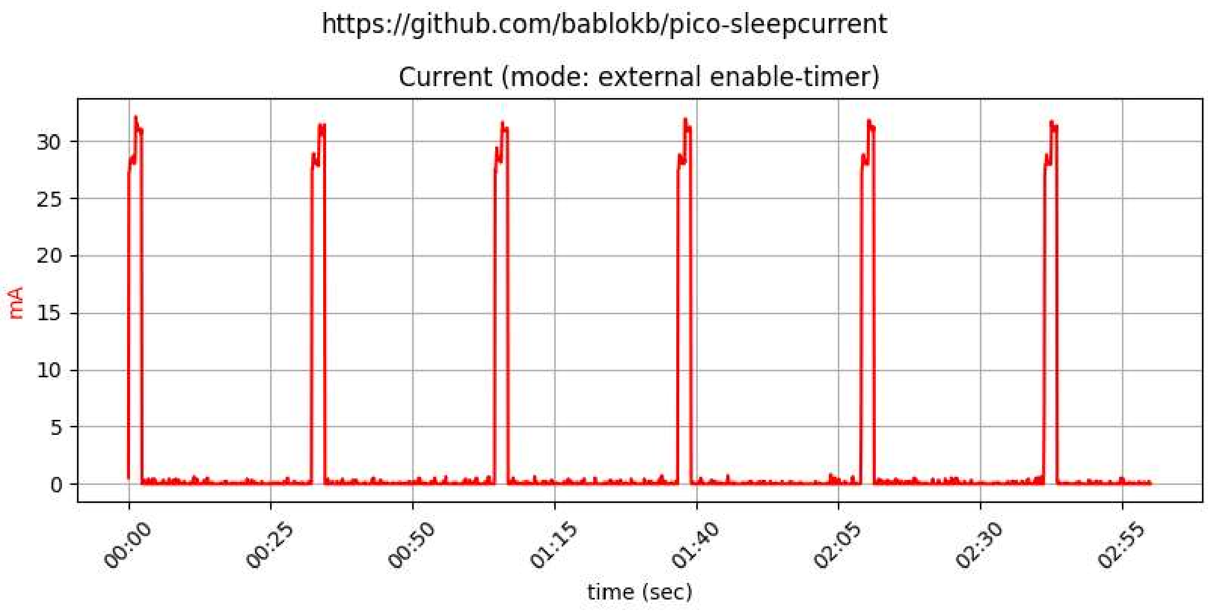

Figure 3. Power usage for an alarm function on a Raspberry Pi Pico device equipped with a 3V3 pin power control, produced by Github user bablokb

Given that the Pico turns off completely, we now need to find another way of keeping time. One option for accurate timing would be incorporating a real-time clock (RTC), which typically has very low-power consumption. For example, Pimoroni's RV3028 Real-Time Clock Breakout is an ultra-low-power RTC, boasting a supposed ~100 nA typical current draw. Using the same high-end AA battery, with a current of 100 nA, this is equivalent to some 3,000 years (2,700 mAh / 100 nA / 24 hours / 365.25 days = over 3,213 years). In other words, the power use added by the RTC (0.1 μA) is insignificant, compared to the remaining stand-by current of 30 μA.

RESISTOR-BASED TIMERS – NANOAMPS?

It may be possible to reduce sleep current to the nanoamp range, using alternative boards that use resistor-based timers, such as the ⇡TPL5110 made by Texas Instruments. Several vendors have produced breakouts based on that chip, and state power consumption in the nanoamp range. For example, Sparkfun says that their TPL5110 Nano Power Timer,

"is ideal for applications that require low power, and especially those projects that are running off of a LiPo [lithium-ion polymer rechargeable] battery. The Nano Power Timer will turn on your project, most likely a microcontroller, after the set amount of time, continuously. When your microcontroller has completed whatever needs doing, sampling air quality for example, it can then signal back to the Nano Power Timer to turn it off. "

The power consumption is very low:

"While the project is off, the Nano Power Timer will only consume 35nA of power until the timer turns the project back on again."

However, working with these very small currents requires a lot of care, and there's always the risk of unwanted leakage. For example, another breakout for the ⇡TPL5110 device is the Adafruit TPL5110 Power Timer Breakout. However, you need to take care to cut the solder jumper to get to the nanoamp range. We reached out to Adafruit customer support, who helpfully advised this:

"That's why we designed the board with a solder jumper so you can cut the trace to the LED. If you cut the traces, the current consumption drops back into the TPL5110's nanoamp range.

Based on measurements, the TPL5110 only reaches the 35nA banner spec when the supply voltage is 2.5V. For supply voltages up to 5V, the TPL5110 uses 80 nA to 90 nA. We usually say 'less than 100 nA', which still comes to about 1 mAh of energy consumption per year."

Obviously, a reduction from the standby afford by the real-time clock (30 μA) down to below 0.1 μA = 100 nA (for operating voltage around 3V) would be significant (300 times lower power consumption). However, we would lose the ability to flexibly time our measurements. This means that the device would have to wake up the microcontroller (say) every 15 minutes; if we needed regular measurements, this would be feasible.

Overall, the flexibility of measurements is significant in our scenario. We may want to measure more frequently during the day and less frequently (or not at all) during the evenings and weekends.

For argument's sake, consider this possible work-around: The microcontroller is woken every 15 mins, and once awake, the microcontroller decides whether to take a measurement or save power by not taking a measurement and going straight back to sleep. Overall, turning on the microcontroller is quite power intensive, and it is better to stick with the RTC-based approach: It consumes more power in standby than the resistor-based approach, but gives us flexibility over when to take measurements (which saves power).

BATTERY LIFE OF ZIGBEE SENSORS

In our project, we will also use ZigBee sensors. These typically have low battery consumption. While some Zigbee devices use AA batteries, most of them use button cells, which have capacity of the order of 100 mAh, and some last around two years, which would be equivalent to a current of 11 μA. Typically, these devices also do not report time, but just send instantaneous readings via the Zigbee protocol. While a consumption of 11 μA is not in the nanoamp range, we would not be surprised if these devices use resister-based timing mechanisms.

REALITY CHECK

It is important to note that the above calculations are idealised. The voltage of the battery will decrease as it discharges. At some point, the voltage will be too low to power the Pico (or other microcontroller). That means that the full charge is not usable, and the actual duration of operation is lower than the above calculation suggests. For the purposes of this article, we assume that the full charge is available. We'll return to this issue in a future blog post, when we provide actual measurements of power consumption.

For a sneak preview, see https://github.com/bablokb/pcb-pico-datalogger, where power use measurements are available. Note that our first design had slightly higher consumption (around 80 μA) while the current design matches the Pimoroni range (around 30 μA).

CONCLUSIONS

With burgeoning maker-friendly computing, there are several options for developing 'low-power devices' suitable for 'strobe measurements'. For the purposes of our use case, our device will need to measure environmental properties during the school day, when the school is being used by students and staff. Preventing unnecessary powering on can reduce battery wastage. Therefore, taking 'strobe measurements' during key hours can help save battery, instead of simply taking regular measurements.

However, power consumption is just one of many considerations around implementing our device in the context of the Improving Learning Through Classroom Experience programme. Future technology-focused articles will describe selecting and testing sensors for an environmental sensing project, as well as design choices for an indoor environmental quality-sensing network.

REFERENCES AND LINKS

- https://forums.adafruit.com/viewtopic.php?t=201034
- https://forums.adafruit.com/viewtopic.php?p=970723#p970723
- https://learn.adafruit.com/deep-sleep-with-circuitpython/rp2040-sleep
- https://github.com/bablokb/pico-sleepcurrent
- https://forums.adafruit.com/viewtopic.php?t=201047
- https://forums.pimoroni.com/t/low-power-consumption-on-raspberry-pi-pico-badger-picowbell/22086

- https://forums.pimoroni.com/t/raspberry-pi-pico-why-does-free-space-show-as-848-kb-and-advice-on-how-to-add-an-sd-card-to-an-enviro-indoor/22038/13

CITATION

Haßler, B. (2023). *How to build an IoT device with low-power sleep* (Improving Learning Through Classroom Experience in East Africa). OpenDevEd. https://doi.org/10.53832/opendeved.1011

PREVIOUS POST
Benefits of using ISSB in school buildings