

We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

6,300

Open access books available

171,000

International authors and editors

190M

Downloads

Our authors are among the

154

Countries delivered to

TOP 1%

most cited scientists

12.2%

Contributors from top 500 universities



WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index
in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com



Chapter

IoT on an ESP32: Optimization Methods Regarding Battery Life and Write Speed to an SD-Card

Lukas M. Broell, Christian Hanshans and Dominik Kimmerle

Abstract

The ESP32 is a popular microcontroller for IoT use cases. For many IoT applications (e.g., environmental sensors or wearables), a continuous power supply is either not possible or too cumbersome, requiring battery operation. However, the ESP32 has a relatively high power consumption. This chapter focuses on battery life optimization methods for this family of microcontrollers. For scenarios where data logging is relevant, methods for increasing communication speed in relation to power consumption are examined in detail. Measurements of seven different commercially available development boards were evaluated in terms of sleep modes, reduced CPU frequencies, and serial communications with the goal of better power efficiency. Therefore, the common scenario of data logging was compared with the performance and power consumption when communicating with different SD cards and CPU frequencies via the SPI and SD bus. Our test results showed that peripheral components (such as voltage regulators) have a large impact on the power consumption of the ESP32 microcontrollers, especially in sleep mode. For data logging, higher clock rates combined with high-quality SD cards and using the SD bus in 4-bit mode resulted in the lowest battery discharge.

Keywords: ESP32, energy consumption, write speed, performance to energy, SD bus, SD-MMC, SDIO, CPU frequency, battery life, IoT, wearables

1. Introduction

For various research questions, comprehensive and objective data collection using appropriate sensor technology is essential. However, for some applications, there are no (affordable) devices available on the market or do not provide the needed data quality, form factor, or access to raw data. As a side effect, one might have special requirements regarding data privacy and protection. As a scientific institution in the biomedical field that has to deal with specific needs and research questions, financial restrictions, and sensitive data retrieved from the sensors, the above-mentioned aspects lead to many software and IoT-related hardware projects [1–4]. One of our medical projects intends to measure heart rate variability. The sensor has to exceed the precision of a clinical-grade ECG device, but at the same time has to be wireless, to be worn on the body, waterproof, heat resistant, and able to resist chemical disinfection [2].

The ECG-based measurement allows medical grade data quality for examining the activity or response of the autonomous nervous system, which is involved in many diseases such as chronic pain, addiction medicine, mental illnesses (e.g. depression) as well as in sports medicine and performance diagnostics. The device is used in several clinical trials and connected to an open-source IoT platform, that allows fleet management of many devices [1]. Another project that uses the same sensor deals with VR-based addiction diagnostics and treatment. In this use case, the main aspect is interoperability with the development environment and the flexibility of integrating further sensors. In both cases, it is particularly important that the battery life is as long as possible and that the raw data (ECG) is stored locally at the highest possible resolution, as correct wireless data transfer cannot always be ensured. There are many more applications in projects, where our custom build sensors came to use, for example, in environmental measurements (urban climate and fine dust measurements, spectro-radiometric measurements, or radiation sensors) or lab sensory, that is used as part of lab experiments or as a fundamental part of lab automation within microbial or cellular experiments [1].

Like us, many other research teams want to take data acquisition into their own hands and develop specially adapted instruments [5, 6]. It is also possible to describe exactly and transparently which algorithms were used to increase the reproducibility of the results. The widespread use and rapid development of easy-to-program microcontrollers such as the Raspberry Pi or microcontrollers based on the Arduino platform, as well as the many sensor modules, libraries and sample codes and projects available, make this easier, and once the basic system is developed, it is easy to add more sensors or adapt the system to different requirements. This allows data to be collected quickly under laboratory conditions. However, if the prototype is to be used in real-life scenarios or in field studies, additional obstacles need to be considered. Haghi et al. list the following points that should be considered when developing wearable IoT devices [7]:

- Easy and secure connectivity
- Low power consumption
- Wearability with small form factor
- Reduced risk of data loss through buffering

In order to limit the scope of this work, we will mainly focus on the aspects of reducing power consumption, as this is directly related to wearability and form factors. In addition, we will also look at data storage, as if large amounts of data are collected and need to be stored locally, this will also have a significant impact on battery life. The aim of this work is to list possible approaches for an optimal compromise between data write speed and energy efficiency in order to derive a best practice for custom development.

2. Background

This section provides background information for comparing different approaches to writing data to an SD card or improving the power efficiency of microcontrollers.

It also provides an initial comparison of commercially available microcontrollers in terms of functionality and power consumption.

2.1 Energy consumption of microcontrollers

To get a basic understanding of microcontroller power consumption, the following formula illustrates the factors that contribute to the microcontroller power consumption [8]:

$$P_d = fCU^2 \quad (1)$$

P_d —Dynamic part of power consumption

f —CPU Clock-frequency

C —Total capacitance of the field-effect transistors (FETs) in the circuit

U —Operating voltage

This relation shows that power consumption can be reduced by lowering the CPU clock-frequency f , the capacitance C or by decreasing the operating voltage U [8]. Since the power consumption is proportional to the operating voltage U squared, it seems to be obvious to initially reduce it as much as possible. Historically, early microprocessors used to run on a 5 V supply voltage. Since then, the voltage has been continuously lowered for that reason. In contrast, the maximum clock frequency has increased over the years to achieve a higher computing performance. However, this has also led to increased power consumption. Nowadays the strategy is to max out the clock frequency capability of a microcontroller while running on a significantly lower clock frequency [8].

The total capacitance is the sum of the capacitances of the individual field-effect transistors (FETs). Due to miniaturization, the FETs' individual capacitances have become smaller, but the number of FETs per processor's core continues to grow. The total capacitance of a given system, therefore, can only be reduced by switching off individual parts of the processor [8].

2.2 General energy saving measures for microcontrollers

Microcontrollers are usually optimized for an energy-efficient operation with a number of mechanisms to minimize energy consumption available. An energy-efficient operation of the microcontroller is usually implemented without the need of an operating system, which means that the programmer has to give appropriate instructions in the application program. Most microcontrollers offer a flexible adjustment of the clock frequency as well as low-power or sleep modes. Depending on the architecture of the microcontroller, certain processor parts or peripheral components are clocked down, the operating voltage is lowered or even disconnected from the power supply. This results in the following rules for energy-efficient programming [5, 6]:

- Complete tasks via hardware instead of software
- Use interrupts and low power mode instead of pin or flag polling
- Use precalculated tables instead of on-demand calculations

- Avoid frequent calls of subroutines, use procedural programming if possible
- Use the fastest possible sampling-rate, transmission-rate, and highest possible clock frequency for executing tasks [9]

Before describing methods for implementing these rules, it is important to consider the SD card communication options in order to select a microcontroller that offers a good trade-off between power efficiency and write performance.

2.3 SD-card communication

An SD-Card (Secure Digital Memory Card) is a digital storage medium that works on the principle of flash storage. This section deals exclusively with standard SD-Cards with 9 pins. All information in this section is taken from the SD specifications of the SD-Card technical committee [10].

2.3.1 Communication systems

The host (microcontroller, card reader, laptop, smartphone, etc.) can access the SD-Card using either the Serial Peripheral Interface (SPI) or the proprietary SD bus protocol.

SD bus: Communication via the SD bus is based on command and data bit streams that are initiated by a start bit and terminated by a stop bit. Each message consists of a command, response, and data block tokens.

- 1-Bit SD bus: Data transfer via a single transmission channel.
- 4-Bit SD bus: Four transmission channels used for higher data transmission rates.

SPI bus: The SPI bus is a bus system consisting of three channels for serial synchronous data transmission. Microcontrollers mostly communicate with SD-Cards via the SPI bus. The SPI protocol does not allow all functions of SD-Cards, like energy saving functionality (e.g., low voltage). Additionally, the maximum transfer speed of the bus speed does not correspond to the maximum read/write speed of the used SD-Card.

2.3.2 Write speed

The maximum supported clock rate is decisive for the highest data transfer rate that can be achieved. Clock frequencies available for standard SD-Cards at the respective communication protocols are listed in **Table 1**.

Unfortunately, most microcontrollers only support the SPI bus for storing data on an SD card. However, the ESP32, which is widely used for IoT applications, supports both SPI and SD bus. For this reason, the features and power saving options of the ESP32 will now be examined in more detail.

2.4 ESP32: Energy options

Like many other microcontrollers, the ESP32 offers a wide range of power saving options. Its processor core is divided into different modules (radio module, main

Communication protocol	Supported clock rates	Max. write speed
SPI	½ of CPU Clock	1.6 MB/s
SD bus 1-bit mode	Default Speed (25 MHz)	3.125 MB/s
	High Speed (50 MHz)	6.25 MB/s
	UHS-I (208 MHz)	26 MB/s
SD bus 4-bit mode	Default Speed (25 MHz)	12.5 MB/s
	High Speed (50 MHz)	25 MB/s
	UHS-I (208 MHz)	104 MB/s

Table 1.
 SD-card communication protocols and respective max write speeds [10].

Power Modi	Description	Current draw	
Modem-sleep	CPU is active	240 MHz	Dual-core 30 mA ~ 68 mA
			Single-core n.a.
	160 MHz	Dual-core 27 mA ~ 44 mA	
		Single-core 27 mA ~ 34 mA	
	80 MHz	Dual-core 20 mA ~ 31 mA	
		Single-core 20 mA ~ 25 mA	
Light-sleep	—	0.8 mA	
Deep-sleep	ULP active	150 µA	
	ULP sensor-monitored pattern	100 µA at 1% load	
	RTC timer + RTC memory	10 µA	
Hibernation	RTC timer only	5 µA	
Power off	CHIP_PU is set to low level, CPU switched off	1 µA	

Table 2.
 Power mode and energy consumption of the ESP32 [11].

processor core, and memory, cryptographic hardware acceleration, ultra-low-power co-processor with real-time clock and recovery memory), which can be switched off individually to save energy. **Table 2** lists the different power options of the ESP32 with the power consumption in the corresponding mode, according to the manufacturer's specifications.

2.5 Comparison of ESP32 development modules

Not only the processor contributes to the total energy consumption of a microcontroller but also all the peripheral modules like voltage regulators, sensors or external flash memory do so as well. Therefore, different developer modules using the ESP32 should be considered with regard to the total energy consumption in the respective power modes. **Table 3** shows the results of the measurements at the different power options.

The differences in power consumption between the different developer boards are considerable, as shown in **Table 3**. As a possible reason for the distinct deviation in power consumption, the built-in voltage regulators come into consideration since the

cDeveloper Modules	Reference (mA)	Light-Sleep (mA)	Deep-Sleep (mA)	Hibernation (mA)
Adafruit HUZZAH32	47	8.43	6.81	6.80
ESP32—DevKitC	51	10	9	9
Ai-Thinker NodeMCU 32S	55	15.05	6.18	6.18
Sparkfun ESP32 Thing	41	5.67	4.43	4.43
FireBeetle ESP32	39	1.94	0.011	0.008
WiPy 3.0	192	—	0.015	—

Table 3.
Comparison of the energy consumption of different ESP32 developer modules [12].

measured energy consumption exceeds the specifications from the data sheet by far. Voltage regulators are known to contribute significantly to the overall consumption of the system due to their quiescent current, which is especially noticeable in sleep states [13]. The voltage regulator of the FireBeetle ESP32, for example, has a maximum voltage drop of 0.31 V at 600 mA and a quiescent current of 4 μ A, while the voltage regulator of the Adafruit HUZZAH32 has a voltage drop of 0.4 V at 600 mA and a quiescent current of 80 μ A.

Nevertheless, the ESP32 itself is rather an energy-consuming. In consequence, the ESP32 family expanded by some more energy-efficient versions with the ESP32-S2 being the most interesting one for IoT applications. It provides almost all the known functionalities but comes as a single-core processor for less power consumption. The comparison of the ESP32 to the ESP32-S2 and its power consumption is summarized in **Table 4**.

The ESP32-S2 is an excellent choice for a variety of IoT applications, but the lack of an SD bus is a limiting factor in achieving an optimal combination of SD card write speed and power efficiency.

Now that the various options for writing to the SD card have been presented, as well as general and specific methods for reducing microcontroller power consumption, the next step is to review related work to evaluate the current best practice for power optimization in combination with high-resolution data acquisition.

Power Mode	Description	ESP 32 nominal current	ESP32-S2 nominal current
Active	transmit 802.11b	240 mA	190 mA
	receive 802.11b	100 mA	68 mA
Modem-Sleep	240 MHz	30 mA ~ 68 mA	19 mA
	160 MHz	27 mA ~ 44 mA	16 mA
	80 MHz	20 mA ~ 31 mA	12 mA
Light-Sleep	—	0.8 mA	450 μ A
Deep-Sleep	ULP active	150 μ A	235 μ A
	ULP sensor-monitored	100 μ A at 1% load	22 μ A at 1% load
	RTC timer + memory	10 μ A	25 μ A
Hibernation	RTC timer only	5 μ A	20 μ A

Table 4.
Comparison of power consumption between ESP32 and ESP32-S2 [11, 14].

3. Related work

One work that optimizes the energy consumption when writing data to an SD card was done by Bradley and Wright in which the energy consumption of the Arduino Atmega328P was determined at 5 V and 16 MHz and 3.3 V and 8 MHz [15]. In each case, the SD card communication was implemented via SPI. It was found that the lower clock rate resulted in a lower discharge. At 16 MHz the transmission time was 9–10 ms, while at 8 MHz, 15 ms was measured. Also, at 8 MHz, a slight delay in SD card response was observed. In deep sleep, the microSD card adapter used for SPI communication contributed significantly to the total power consumption. Without the SD card, 120 and 96 μA were measured, while 800 and 750 μA were measured with the SD card connected. To reduce power consumption, a BS170 power control N-channel MOSFET was used. This reduced the current consumption during deep sleep to 21.1 and 18.6 μA , respectively, which is a reduction factor of 40. However, this MOSFET also led to a reduction in the transmission speed for SD communication from 20 to 150 ms. Further work with higher clock speed and larger SRAM is announced [15]. This work demonstrated a good option to minimize power consumption when using SD cards but only if a low sampling or transmission rate is required.

Regarding optimizing the write speed to an SD card, we could not find any comparable work in the scientific literature, but a blog post has been written demonstrating the performance increase when using the SD bus on the ESP32 compared to the SPI and how this was achieved [16]. A difference of about 230% for write operations and about 400% for read operations was shown using the SD bus in 4-bit mode compared to SPI.

Similarly, only one paper was found that addressed the energy-efficient operation of an ESP32 [17]. This paper gives a best practice for using an ESP32 in an industrial wireless sensor network. The different operating modes of the ESP32, as mentioned above, are listed with a recommendation to switch between operation modes over time to perform tasks with the suitable operation mode. It is also noted that in active mode, energy efficiency can be further improved by adjusting the processor clock speed.

As shown in the introduction and related work, there has been scarce work addressing the requirements of microcontrollers for wearable IoT applications and optimizing communication to a local storage medium. This is certainly a niche area, but the steady growth, ease of access, and the resulting variety of use cases have shown that the evaluation of further optimization methods is nevertheless useful and relevant.

Therefore, in the following, possible approaches to optimize the speed of writing data to an SD card while taking power consumption into account will be investigated. In addition, possible methods for further reducing power consumption by making various adjustments to the ESP32's operating modes will be investigated and different microcontrollers will be compared.

4. Material and methods

This section lists all the microcontrollers used, the different operating states, power-saving measures, and SD communication methods, as well as the measurement methods and evaluation methods of each test.

4.1 Optimization of the SD-card communication speed

The ESP32 features an SD bus interface that allows communication to SD-Cards in 1-bit and 4-bit modes. Read and write speed was compared in both modes with the performance of the SPI bus. In addition, a SanDisk Extreme 32GB (Speedclass 10, UHS Speed Class 3, max. Transfer speed 160 MB/s) was compared with a SanDisk Ultra 32GB (Speedclass 10, UHS Speed Class 1, max. Transfer speed 98 MB/s) during the three different communication scenarios. The Arduino sample programs “SD_Test” [18] for SPI and “SDMMC_Test” [19] for 1-bit and 4-bit data transfer were used to control the SD-Card, transferring data and measuring the transfer time. The sample programs are included when installing the ESP32 board manager into the Arduino IDE. All codes were executed on a DFRobot FireBeetle ESP32. **Table 5** shows the GPIO-Pin connections of the setup, respectively.

4.2 Optimization of power consumption

For power consumption measurements, the developer boards were operated with a 3.7 V LiPo battery. The following subsections describe the individual energy-saving options and experimental setups in more detail, measured with a digital multimeter (Testboy 313).

4.2.1 Determination of the most energy-efficient SD communication method

In all above-listed communication methods, the same Arduino sample programs for SD-Card communication were used. The current was measured during read and write operations. Over the elapsed time, the actual discharge was calculated according to the following formula, where C is the discharge in coulombs, A is the measured current in ampere, and t is the elapsed time in seconds.

$$C = A * t \quad (2)$$

Name		Pin		FireBeetle Pins SD bus	FireBeetle Pins SPI
SD bus	SPI	SD	Micro-SD		
DAT1	—	8	8	D0/IO4	—
DAT0	DO	7	7	D9/IO2	MISO/IO19
VSS	VSS	6	6	GND	GND
CLK	SCLK	5	5	BCLK/IO14	CLK/IO18
VCC	VCC	4	4	V3.3	V3.3
VSS2	VSS2	3	—	GND	GND
CMD	DI	2	3	A4/IO15	MOSI/IO23
DAT3	CS	1	2	D7/IO13	D8/IO5
DAT2	—	9	1	MCLK/IO12	—

Table 5.
SD-card connection to a DFRobot FireBeetle ESP32 in SD and SPI bus.

4.2.2 Comparison of different developer boards

Three frequently used ESP32 developer modules, one ESP32-S2 module, and also non-ESP boards were compared (see **Table 6**). Developer boards without an integrated battery voltage regulator were operated via an external battery voltage regulator (Adafruit Micro-Lipo Charger) and the same LiPo battery.

For a comparison between the developer boards, the current consumption was measured during two different operating states:

- Normal mode + LED blink
- deep sleep

4.2.3 CPU clock frequency reduction

To evaluate the impact of reduced CPU clock frequency on power consumption, the system was put into different operating states (see **Table 7**).

Wi-Fi network scan and modem sleep were run on all three developer boards. The effects of different CPU clock frequencies on SD communication speed were run exclusively on the FireBeetle ESP32 as it has the lowest energy consumption among the ESP32 boards with little quiescent current (according to **Table 2**). Hence, it reflects the actual power consumption of the processor best, and the Adafruit Feather S2 has no built-in SD bus interface. The same “SDMMC_Test” sample program was used as before. For a more accurate current measurement on read and write

Development module	CPU	CPU-Clock (MHz)	Nominal current	Voltage regulator	Quiescent current
ESP32-DevKitC-32D	ESP 32	240	40 mA	AMS1117	5 mA
Adafruit HUZZAH32	ESP 32	240	40 mA	AP2112-3.3	80 μ A
FireBeetle ESP32	ESP 32	240	40 mA	RT9080-33GJ5	4 μ A
Adafruit Feather S2	ESP32-S2	240	19 mA	NCP167BMX3 0TBG	18 μ A
Arduino Nano 33 BLE	Cortex M4F	64	6.4 mA	MP2322GQH-Z	5 μ A
Feather M0 Bluefruit	Cortex M0+	48	5.0 mA	AP2112-3.3	80 μ A
nRF52840 MDK	Cortex M4F	64	6.4 mA	n.a.	n.a.

Table 6.
 Development modules' specifications for energy efficiency comparison.

Operation state	Development modules	CPU-Clock rates (MHz)
Wi-Fi-Scan	HUZZAH32, FireBeetle ESP32, Feather S2	240, 160, 80
Modem sleep + LED blink	HUZZAH32, FireBeetle ESP32, Feather S2	240, 160, 80, 40, 20
Modem sleep + SD communication	FireBeetle ESP32	240, 160, 80, 60

Table 7.
 Tested operating states and clock frequencies of different ESP32 boards for energy efficiency comparison.

operations, a data transfer of 8 MB was set instead of the default 1 MB. The Wi-Fi network scan was only performed up to a minimum clock frequency of 80 MHz as Wi-Fi connectivity is only guaranteed by the ESP32 up to this clock frequency [3].

5. Results

In this section, the results of the compared optimization methods for the communication speed to the SD-Card as well as for the energy efficiency of the ESP32 in different operating states are given.

5.1 SD-card communication speed

Figure 1 shows the comparison of transfer speed between SD bus in 1-bit and 4-bit mode as well as communication via SPI bus on a SanDisk Ultra 32 GB and a SanDisk Extreme 32 GB in milliseconds.

The SD bus is clearly superior to the SPI bus, but there is only a little difference between the 1-bit and 4-bit modes. In 4-bit mode, the write processes are about 10% and read processes are about 25% faster in comparison to the 1-bit mode. The differences in reading operations between SanDisk Ultra and SanDisk Extreme are neglectable, however, while writing, the SanDisk Extreme 32GB performed the task 20% faster.

5.2 Energy consumption

5.2.1 Comparison of SD communication methods

To further evaluate the best SD-Card communication method for mobile devices, the results of the current measurements during read and write operations are listed

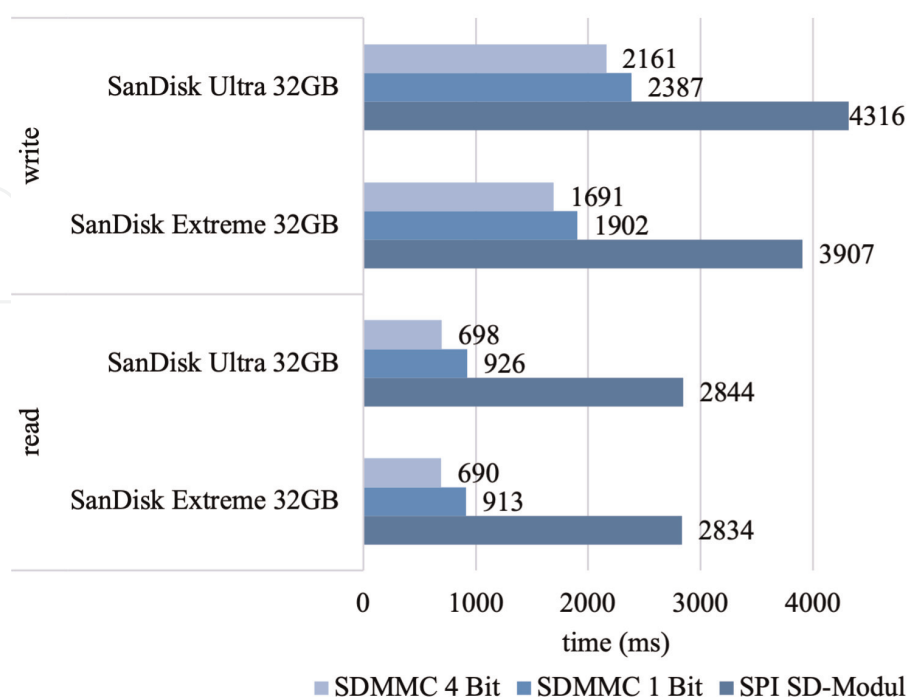


Figure 1. Read and write speeds of different SD-cards in SD bus 1-bit mode, SD bus 4-bit mode, and SPI bus.

below. **Table 8** shows the durations and measured currents of the SPI bus with three different commercially available SD-Card modules as well as SD bus communication (1- and 4-Bit mode) using a wired SD-adapter.

No great difference was observed between the various SD-Card modules in the SPI bus mode, neither in read or write speed nor in current consumption. However, there is a considerable difference among both SD bus protocols: comparing SD bus to SPI during reading, the current consumption is increased by around 13% in 1-bit SD bus mode and by around 15% in 4-bit mode, however, while writing, the current draw is only increased by around 5% in 1-bit mode and by around 8% in 4-bit mode. Generally, reading processes showed a higher power consumption than slower writing processes.

To evaluate whether a faster write speed via the SD bus results in a less overall discharge of the battery (despite its higher current draw) each actual discharge was calculated and is shown in **Figure 2**.

As can be seen, the higher write speed causes less discharge of the battery despite a higher current flow during the operation. Therefore, the SD bus in the 4-bit mode has the lowest overall energy consumption in this scenario.

5.2.2 Comparison of various developer boards

The first comparison of the developer boards was carried out during a standard test program, the flashing of the built-in LED. **Figure 3** shows the current consumption of the various developer boards during this test program.

In this comparison, the ESP32 modules show the highest power consumption during the LED blink program, with the DFRobot FireBeetle ESP32 performing best among them. As expected, the newer single-core ESP32-S2 of the Adafruit Feather S2 shows a lower power consumption. The lowest power consumption is shown by the non-ESP32 boards, of which the Arduino Nano 33 BLE has the highest power consumption among the non-ESP32 boards. Even though having the same CPU as the Arduino, the nRF52840 MDK had the lowest power consumption. The Adafruit Feather M0 Bluefruit shows a slightly higher power consumption than the nRF52840 MDK despite the efficient ARM Cortex M0+ CPU. The comparison of the developer boards in deep sleep mode is shown in **Figure 4**.

SD-Card modules & communication methods	read		write	
	Current draw (mA)	Time (ms)	Current draw (mA)	Time (ms)
Standard SD-SPI	100	2851	86	3886
Adafruit μ SD-Module	102	1738	89	2768
NoName SPI SD-Module	100	1736	86	2791
SD-Adapter SPI	103	1779	90	2766
SD bus 1-Bit mode	115	913	92	1902
SD bus 4-Bit mode	117	761	95	1744

Table 8. Comparison of current consumption during read and write operations on the SD-card of different modules and communication methods.

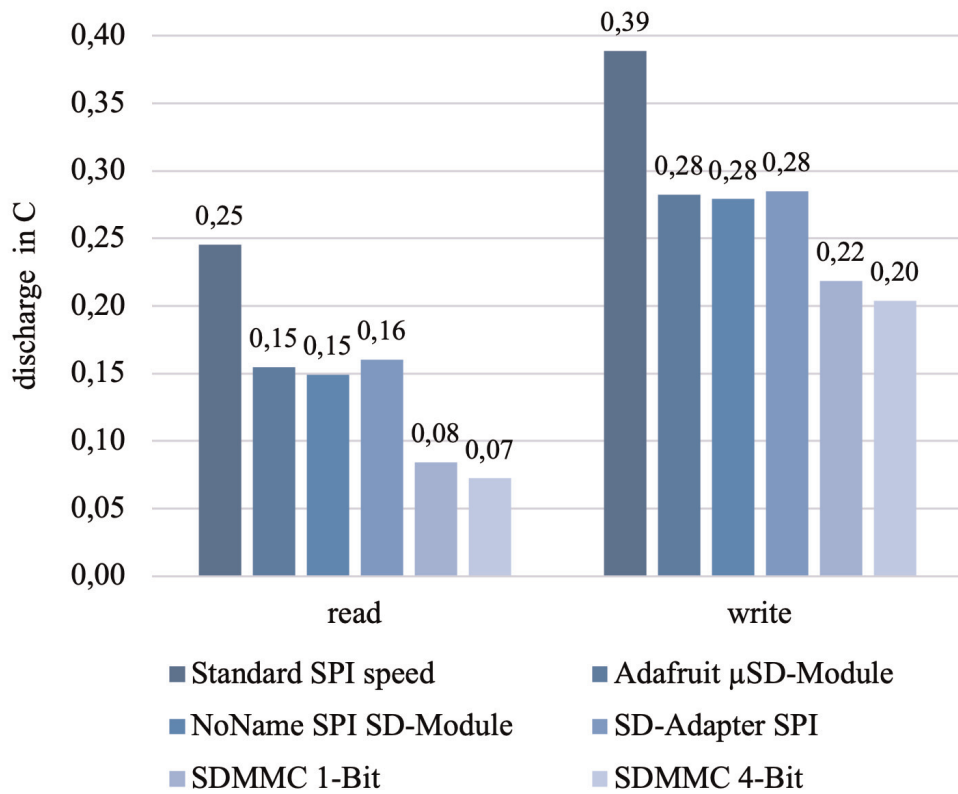


Figure 2. Calculated battery discharge when reading and writing from and to an SD-card with different methods and modules.

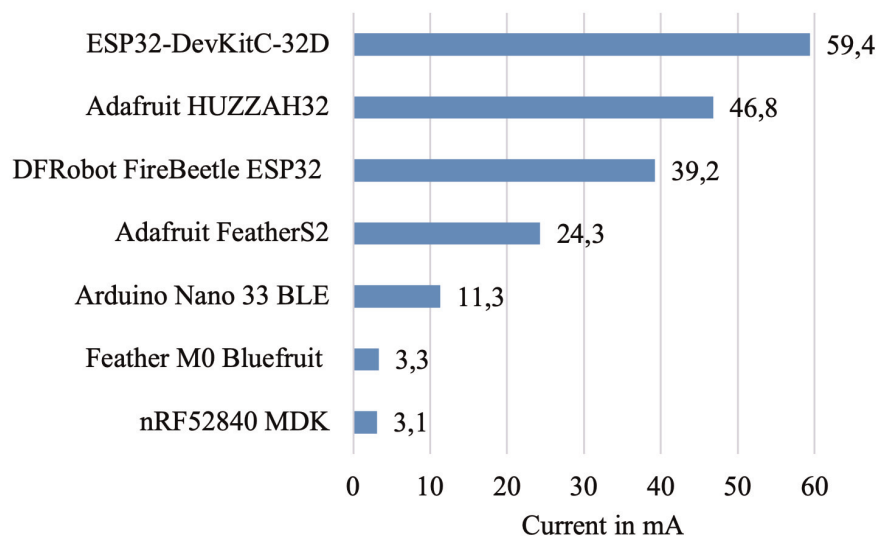


Figure 3. Current measurement of different developer boards during a specific program sequence (LED blink).

The Adafruit HUZZAH32 (according to **Figure 4**) has the highest power consumption in the sleep state. This is consistent with **Table 2**. Correspondingly, the FireBeetle ESP32 shows a low power usage in the sleep state, only beaten by the Adafruit Feather S2, although the ESP32-S2 processor itself has a higher current requirement in deep sleep (see **Table 3**) and uses a less economical voltage regulator (see **Table 6**).

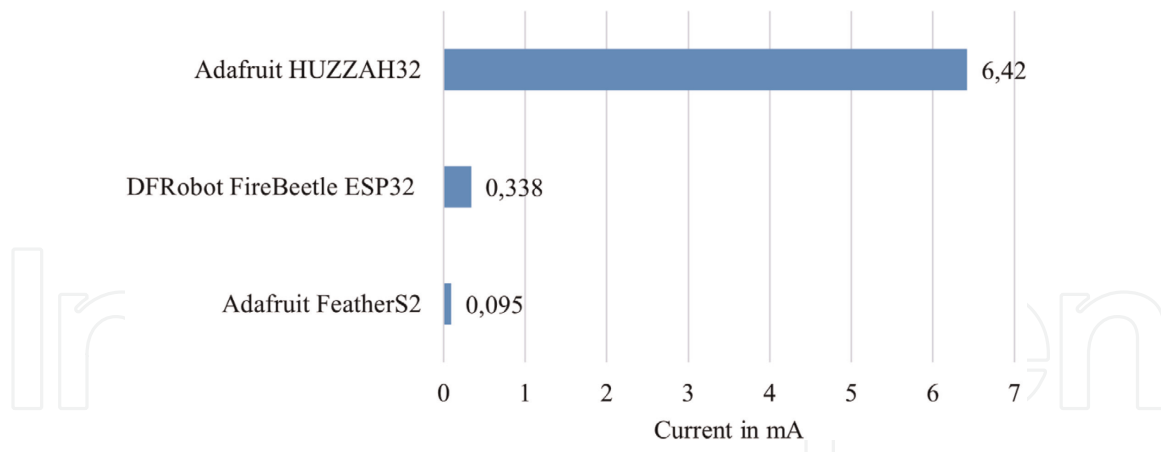


Figure 4. Current measurement of different ESP32 developer boards in deep sleep. The ESP32 and ESP32-S2 boards were used due to their similar deep sleep functionality.

5.2.3 CPU clock frequency reduction

Since only the Adafruit Feather S2 among the non-ESP32 developer boards has Wi-Fi connectivity, for the comparison of current consumption at different CPU clock frequencies and operating states, it was compared with the two ESP32 boards. **Figure 5** visualizes the results of the current measurements. The Wi-Fi scan was performed up to a minimum clock frequency of 80 MHz, since the Wi-Fi module of the ESP32 is not supported at a lower clock frequency [3].

Figure 5 shows that the Adafruit HUZZAH32 consistently has the highest power consumption in different operating states and at reduced CPU clock frequencies, whereas the Adafruit FeatherS2 consistently shows the lowest power consumption in this comparison. It is noticeable that the DFRobot FireBeetle ESP32 only has a slightly higher power requirement than the FeatherS2 in Wi-Fi Scan and at a clock frequency

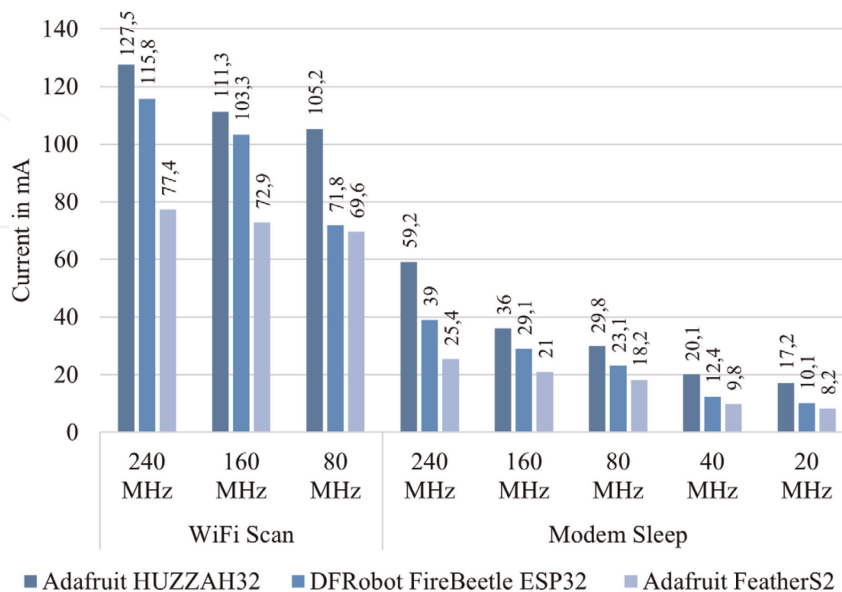


Figure 5. Measurement of current at different CPU clock frequencies during Wi-Fi network scan and modem sleep of different ESP32 developer board.

	CPU-Clock (MHz)	read		write			
		Time (ms)		Current (mA)	Time (ms)		Current (mA)
		M	SD		M	SD	
4-Bit	240	1221.3	16.95	120.3	3163.8	5.82	98.8
	160	1421.8	12.69	98.7	3622	3.74	78.5
	80	1842.5	20.27	80.7	4729	3.32	63.8
	60	1210.3	6.83	120.2	3330.3	276.9	99.9
1-Bit	240	2563.7	28.06	102.3	4376	4.08	92.8
	160	2762.8	12.65	86.2	4807.5	5.22	75.3

Table 9. Comparison of current consumption and process duration during read and write operations at different CPU clock frequencies in 4-bit and 1-bit SD bus protocol.

of 80 MHz, as well as a constant approximation of current draw to the Feather S2 in modem sleep with reduced clock frequency.

To evaluate the functionality at reduced CPU clock frequencies in modem sleep, **Table 9** lists the performance when reading from and writing to the SD-Card at different clock frequencies as well as the simultaneously measured current flow. As with the Wi-Fi module, the ESP32 does not seem to support the SD bus protocol at clock frequencies lower than 80 MHz. 60 MHz could still be executed, but lower clock frequencies generated error messages and the data transfer was not executed.

Table 9 also shows that the current decreases with lower clock frequencies and a lower current requirement in 1-bit mode, the communication speed is also reduced in both manners. At 60 MHz in 4-bit mode, the measurement is comparable to 240 MHz, which indicates that the reduction of the clock frequency below 80 MHz leads to malfunctions. As before, the calculation of the actual battery discharge should provide information about which mode at which clock frequency means the lowest power

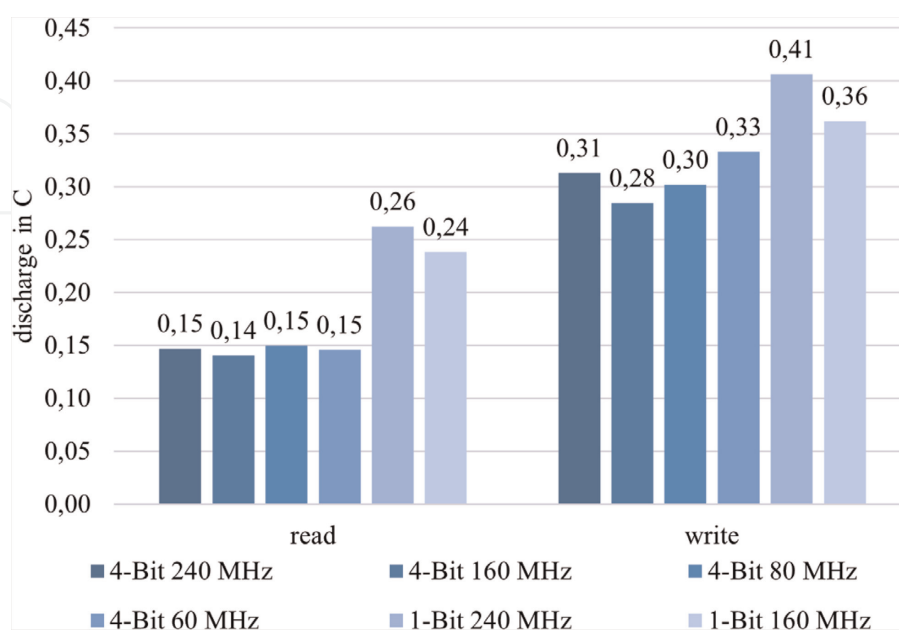


Figure 6. Calculated actual battery discharge when reading and writing the SD-card at different CPU clock frequencies.

consumption and therefore longest battery lifetime. **Figure 6** shows the discharge calculated from **Table 9** during the read and write operations of the two modes at the different clock frequencies.

The direct comparison shows that despite the higher current requirement, the 4-bit mode causes less battery discharge than the 1-bit mode due to the faster communication speed. The comparison of the power consumption of the 4-bit mode at different clock frequencies shows only minor differences, of which the best efficiency was observed at a clock frequency of 160 MHz in 4-bit mode.

6. Discussion

In this section, the results presented above are discussed and possible reasons for controversial results are mentioned.

6.1 SD-card communication speed

As assumed, the SD bus showed higher data rates than SPI, but the difference was much smaller than expected. Data rates of only 1.45 MB/s in 4-bit mode while reading and 0.59 MB/s while writing were shown, even though the ESP32 supports the high frequency (50 MHz) 4-bit SD bus mode, which according to the datasheet should allow a transfer rate of up to 25 MB/s. Additionally, the 1-bit mode was not 4 times slower than the 4-bit mode. Only a 25% slower transfer rate for reading and 11% slower for writing data was measured. Lower transfer rates than those reported by the data sheets were also measured for the SPI bus, with 0.35 MB/s achieved while reading and 0.26 MB/s while writing, they are slightly less than a quarter of the nominal value of up to 1.6 MB/s. Possible causes for the deviations could be explained by inefficient register allocation of the libraries provided for the ESP32 SD bus interface, which is also shown by the fact that other colleagues could also not achieve higher transfer rates when using the ESP32 [16].

Nevertheless, **Figure 2** shows that the method with the highest transfer rate and thus a shorter transfer duration also causes the lowest battery discharge, which is why the SD-bus in 4-bit mode on the ESP32 is recommended even if its performance potential cannot be achieved. Likewise, the usage of a high-quality SD-Card is also recommended. The data can be transferred at three different CPU clock rates, whereby the differences in discharge turned out to be comparatively small, in this case, the rule faster data transfer equals lower discharge does not seem to apply. In our test, the clock rate of 160 MHz showed the lowest discharge.

6.2 Power consumption

Measuring the current with a multimeter while operating the microcontroller on a battery is not the most precise method, but sufficient for the extent of the differences between the respective compared developer boards and processes. In terms of energy consumption, it was basically found that the ESP32 has the highest energy consumption consistent with the data sheets, followed by the ESP32-S2, then the Cortex M0+ of the Adafruit Bluefruit M0, and with slightly less consumption of the nRF52840. Although the ESP32 has the highest power consumption, this processor also offers some advantages such as SD bus connectivity and high processor clock frequency for

more computational performance and faster data rates, so further research was done with the ESP32.

Consistent with **Table 2**, in our series of measurements the FireBeetle ESP32 also showed the lowest quiescent current and power consumption in deep sleep among the ESP32 boards. However, the Adafruit Feather ESP32-S2 had an even lower power consumption, although the ESP32-S2 itself has a higher power consumption in deep sleep and a less efficient voltage regulator. The lower power consumption that we observed can be explained by the intelligent circuit layout: all peripheral modules are connected in a second circuit, which is switched off in deep sleep.

The reduction of the clock frequency showed a large effect up to a clock rate of 40 MHz. Below that, only a small reduction in energy consumption was detectable. At 20 MHz the ESP32 had a similar power consumption as the Arduino Nano 33 BLE at 64 MHz. But the Arduino Nano 33 BLE has several peripheral modules like the NINA Bluetooth module and a 9-axis IMU which is responsible for the higher power consumption compared to the nRF52840 MDK which uses the same CPU. Without the peripheral consumers, the Cortex M0+ as well as the nRF52840 showed a significantly lower power requirement than the throttled ESP32.

It could be demonstrated that the choice of a low-power developer board has a great impact on the overall power consumption of the system. If an economical module with low quiescent current is used, the power requirements can be throttled down to almost the level of the more economical microcontrollers such as the Adafruit Feather S2 or the Arduino Nano 33 BLE with still sufficient performance reserve. But if less performance is sufficient and small amounts of data have to be transferred, the Cortex M0+ or the nRF52840 are clearly recommended. As long as fast Wi-Fi data transfer and better performance are required but no fast data storage on SD-Cards is necessary, the ESP32-S2 would be the best choice.

7. Conclusion

In summary, the combination of high write speed and low power consumption is difficult to reconcile. It is recommended to write to the SD-Card as infrequently as possible and at 160 MHz using an ESP32 in SD bus 4-bit mode. In ordinary program cycles, the ESP32 should be operated in the state with the lowest power consumption. If a Wi-Fi connection is not necessary at any time or there is no stable Wi-Fi connection, the ESP32 should be operated in modem sleep mode and the clock rate should be reduced to the minimum necessary to function, although a reduction below 40 MHz has little effect. Tasks with high computational effort should be performed at the highest possible clock rate to reduce the duration of the increased power consumption. For our application, the FireBeetle ESP32 is best suited and is recommended for comparable applications. If similar computational performance but no high-speed data logging is required, we suggest using the ESP32-S2 instead. If less processing power is sufficient, either the ARM Cortex M0+ with an additional Wi-Fi module for Wi-Fi applications or the nRF52840 for Bluetooth Low Energy applications are the best choices.

Conflict of interest

The authors declare no conflict of interest.


IntechOpen

Author details

Lukas M. Broell*, Christian Hanshans and Dominik Kimmerle
Department of Applied Sciences and Mechatronics, University of Applied Sciences
Munich, Munich, Germany

*Address all correspondence to: lbroell@hm.edu

IntechOpen

© 2023 The Author(s). Licensee IntechOpen. This chapter is distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/3.0>), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited. 

References

- [1] Stadler S, Borrero ER, Zauner J, Hanshans C. Development and implementation of an OpenSource IoT platform, network and data warehouse for privacy-compliant applications in research and industry: A fast-growing, multi-user, hardware and cloud agnostic IoT data solution. Easy to deploy, privacy compliant, cost-effective, and OpenSource. *Current Directions in Biomedical Engineering*. 2021;7(2): 507-510
- [2] Hanshans C, Broell LM, Plischke H, Offenbaecher M, Zauner J, Faust MMR, et al. Movement filtered heart rate variability (HRV) data from a chest-worn sensor. *Current Directions in Biomedical Engineering*. 2021;7(2):57-60
- [3] Hanshans C, Maisch B, Zauner J, Faust MMR, Bröll LM, Karch S. Virtual therapeutics—Requirements to deliver value with virtual reality and biofeedback applications for alcohol addiction therapy. *Current Directions in Biomedical Engineering*. 2021;7(2):81-84
- [4] Stadler S, Plischke H, Hanshans C. Development of bioimpedance sensors and measurement system for biomedical In-vitro applications: Versatile and cost-effective biosensing system for light-induced apoptosis and cell growth studies of epithelial and endothelial tissue. *Current Directions in Biomedical Engineering*. 2021;7(2):496-499
- [5] Iribarren AP, Luján JP, Azócar G, Mazzorana B, Medina K, Durán G, et al. Arduino data loggers: A helping hand in physical geography. *The Geographical Journal*. p. 1-15. DOI: 10.1111/geoj.12480
- [6] Kondaveeti HK, Kumaravelu NK, Vanambathina SD, Mathe SE, Vappangi S. A systematic literature review on prototyping with Arduino: Applications, challenges, advantages, and limitations. *Computer Science Review*. 2021;40:100364
- [7] Haghi M, Thurow K, Stoll R. Wearable devices in medical internet of things: Scientific research and commercially available devices. *Healthcare Informatics Research*. 2017;23(1):4
- [8] Wüst K. Energieeffizienz von Mikroprozessoren. In: *Mikroprozessortechnik* [Internet]. Wiesbaden: Vieweg+Teubner; 2011. pp. 237-248. DOI: 10.1007/978-3-8348-9881-4_12
- [9] Al-Kofahi MM, Al-Shorman MY, Al-Kofahi OM. Toward energy efficient microcontrollers and internet-of-things systems. *Computers and Electrical Engineering*. 2019;79:106457
- [10] SD Specifications Part 1 Physical Layer Simplified Specification Version 8.00. The technical Committee SD Card Association; 2020
- [11] Espressif Systems. ESP32 Series Datasheet Version 3.8 [Internet]. 2021. Available from: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf
- [12] Christopher David. Guide to Reduce the ESP32 Power Consumption by 95% [Internet]. 2020. Available from: <https://diyIoT.com/reduce-the-esp32-power-consumption/> [Accessed: July 11, 2022]
- [13] Bui T. Why Low Quiescent Current Matters for Longer Battery Life [Internet]. maxim integrated; 2018. Available from: <https://www.maximintegrated.com/content/dam/files/design/technical-documents/white-papers/why-low-quiescent-current->

matters-for-longer-battery-life.pdf
[Accessed: January 7, 2023]

[14] Espressif Systems. ESP32-S2 Technical Reference Manual (Version 1.0) [Internet]. Espressif Systems; 2021. Available from: https://www.espressif.com/sites/default/files/documentation/esp32-s2_technical_reference_manual_en.pdf

[15] Bradley LJ, Wright NG. Optimising SD saving events to maximise battery lifetime for Arduino™/Atmega328P data loggers. *IEEE Access*. 2020;**8**: 214832-214841

[16] A breakdown of my experience trying to talk to an SD card REALLY fast with the ESP32 using SDMMC [Internet]. *r/esp32*. 2019. Available from: www.reddit.com/r/esp32/comments/d71es9/a_breakdown_of_my_experience_trying_to_talk_to_an/ [Accessed: January 5, 2023]

[17] Gatial E, Balogh Z, Hluchy L. Concept of energy efficient ESP32 Chip for industrial wireless sensor network. In: 2020 IEEE 24th International Conference on Intelligent Engineering Systems (INES) [Internet]. Reykjavík, Iceland: IEEE; 2020. pp. 179-184. Available from: <https://ieeexplore.ieee.org/document/9147189/> Accessed: February 18, 2023

[18] Arduino core for the ESP32, ESP32-S2, ESP32-S3 and ESP32-C3—Example Programm “SD_Test.ino” [Internet]. Espressif Systems; 2022. Available from: https://github.com/espressif/arduino-esp32/blob/3e65a5721ae17b578136db1c55accb549a4d0204/libraries/SD/examples/SD_Test/SD_Test.ino [Accessed: August 11, 2022]

[19] Arduino core for the ESP32, ESP32-S2, ESP32-S3 and ESP32-C3—Example Programm “SDMMC_Test.ino”

[Internet]. Espressif Systems; 2022. Available from: https://github.com/espressif/arduino-esp32/blob/3e65a5721ae17b578136db1c55accb549a4d0204/libraries/SD_MMC/examples/SDMMC_Test/SDMMC_Test.ino [Accessed: August 11, 2022]