Accepted manuscript

As a service to our authors and readers, we are putting peer-reviewed accepted manuscripts (AM) online, in the Ahead of Print section of each journal web page, shortly after acceptance.

Disclaimer

The AM is yet to be copyedited and formatted in journal house style but can still be read and referenced by quoting its unique reference number, the digital object identifier (DOI). Once the AM has been typeset, an ‘uncorrected proof’ PDF will replace the ‘accepted manuscript’ PDF. These formatted articles may still be corrected by the authors. During the Production process, errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal relate to these versions also.

Version of record

The final edited article will be published in PDF and HTML and will contain all author corrections and is considered the version of record. Authors wishing to reference an article published Ahead of Print should quote its DOI. When an issue becomes available, queuing Ahead of Print articles will move to that issue’s Table of Contents. When the article is published in a journal issue, the full reference should be cited in addition to the DOI.
Submitted: 16 December 2022

Published online in ‘accepted manuscript’ format: 23 August 2023

Manuscript title: Evaluation of low-cost microcontroller-based electronic systems for simple sensor applications

Authors: Benjamin Burse¹, Kristina Doycheva², Andreas Aicher³, Christian Walther⁴, Jan Oliver Ringert¹

Affiliations: ¹Software Engineering, Bauhaus-Universität Weimar, Weimar, Germany. ²Fraunhofer IVI, Fraunhofer Institute for Transportation and Infrastructure Systems, Dresden, Germany. ³Urban Water Management and Sanitation, Bauhaus-Universität Weimar, Weimar, Germany. ⁴Institute for Structural Mechanics, Bauhaus-Universität Weimar, Weimar, Germany.

Corresponding author: Benjamin Burse, Software Engineering, Bauhaus-Universität Weimar, Weimar, Germany.

E-mail: benjamin.burse@uni-weimar.de
Abstract

Single-board computers like Raspberry Pi, have enabled a wide variety of monitoring applications. Microcontroller development boards offer similar functionality, but at a lower cost. Still, they are not widely used, because they are supposed to be slow. This paper aims to evaluate the abilities of modern microcontrollers and compares them to single-board computers. For this purpose experiments concerning power consumption and performance evaluation is conducted. The results suggest that promising low-cost microcontrollers exist, which could be applied to reduce cost and energy consumption per node or increase the number of nodes used simultaneously in one monitoring system.
1. Introduction

In the modern world, many systems continuously record data about their environment, unnoticed and in the background. Many of them are helpful, some even essential. This process is called monitoring. If these systems also interact with their environment, it is called monitoring and control. The size of systems using monitoring and control ranges from small, everyday items to entire factories or cities (Su et al., 2014; Westkämper et al., 2013; Eremia et al., 2017; Zand et al., 2012). In the classical engineering sciences of civil and mechanical engineering, monitoring is used to understand materials and their properties and, thus, to improve them. The procedure of monitoring specific criteria over the component’s lifetime is referred to as structural health monitoring (SHM) (Malek and Kaouther, 2014; Sheng et al., 2011).

A hardware review conducted by Healy et al. (2008) lists more than 40 different sensor nodes produced since 1998 together with their hardware specifications. In cases where a single measuring device is not sufficient for a task or multiple devices are required to handle several tasks, wired networks need to be formed. Nowadays, these networks shift towards wireless technologies, as they enable more opportunities concerning their installation. In recent years, wireless sensor networks allowed for utilization in new SHM application domains (see comprehensive surveys in Karray et al. (2018); Khalifeh et al. (2022)). Wireless monitoring systems are cheaper and easier to install and maintain than conventional wired systems, and transmission times are negligible depending on the application Pentaris et al. (2013).

Many wireless sensor nodes are based on single-board computers (SBC) like the Raspberry Pi (Crookes et al., 2021), e.g., in applications for structural health monitoring.
Abdelgawad and Yelamarthi (2017); Morgenthal et al. (2019). The term “single-board computer” is used within this work in contrast to standard desktop computers on which each component like central processing unit (CPU), memory, or peripheral devices such as graphic card, sound card, etc. are separate components that are plugged into a linking circuit board most often referred to as motherboard (e.g. the IBM-like personal computer). Single-board computers are available in different sizes and appearances. Many, but not all, are also developer boards offering access to a wide variety of input and output interfaces (Ray and Al Dhaheri, 2017; Kanagachidambaresan, 2021).

The work presented in this paper assesses if microcontroller-based sensor node designs could complement or even replace existing non-microcontroller-based ones and thus increase cost-efficiency. Therefore, it focuses on evaluating technical criteria which are relevant for selecting project hardware. To accomplish this objective, a wireless sensor node system based on microcontrollers (MC) was designed and tests were conducted. Relevant criteria, such as power consumption, computational power, and connectivity were examined and results collected, analyzed, and interpreted in the context of similar single-board-computer-based solutions. Additional capabilities and features of SBCs against MCs, like higher processing capacity, memory size, and programming flexibility are very well described by Álvarez et al. (2021).

A preliminary version of this paper was published as Burse et al. (2022). The present version was extended to include studies on relevant energy supplies for microcontrollers and a significantly extended discussion. In particular, Sect. 5 was extended with experiments and data on the discharge of batteries and two additional runs analyzing different settings that
might have influenced early shutdowns of the microcontrollers observed in a first, previously reported, run. Major extensions of the conclusion compared to Burse et al. (2022) include a discussion of available hardware interfaces, modes of data storage, mesh network capabilities, and over-the-air maintenance of microcontrollers.

2. **A reference design of a wireless sensor node system based on microcontrollers**

This section presents a reference design for a wireless sensor node system consisting of three major components: a sensor node consisting of a microcontroller and sensors (S1, S2, etc.) attached to it through interfaces (iface), a gateway, and a server. Figure 1 presents a high-level overview of such a system. The gateway and the server are intentionally kept abstract. Furthermore, the power supply of each component is considered to be a separate building block, as it ideally could be chosen from a variety of options like batteries, solar modules, and generators. It is also possible to realize gateway and server within a single device. In this case, no Wide Area Network (WAN) connection would be required.

Based on the reference design of a wireless sensor node system, a specific implementation is proposed, as depicted in Figure 2. To transmit data between the sensor node and the gateway, the wireless transmission technology wireless fidelity (WiFi) was chosen over other options like Bluetooth, as it offers a high number of possible devices and a good transmission quality over long distances. This decision directly dictates the appearance of the gateway. For simplicity and as the most general solution, a regular WiFi router was chosen to function as a gateway. The ESP32 (Espressif Systems, 2021) and the ESP8266 (Espressif Systems, 2020) were selected as the microcontroller unit respectively. They offer decent
performance, a wide range of analog and digital interface connections, e.g. Inter-Integrated Circuit bus PC (NXP Semiconductors, 2021), low power consumption, and good documentation at low procurement costs. The sensor nodes were equipped to measure different physical quantities by attaching the following sensors to the PC interface: INA219 (current, voltage), BME280 (temperature, humidity, air pressure), and BH1750 (illuminance) (Texas Instruments, 2015; Bosch Sensortec, 2015; ROHM Semiconductor, 2011). Additionally, the analog temperature sensor DHT11 (Sunrom Technologies, 2012) was chosen to evaluate an interface without PC. For the server, a Raspberry Pi 3 was selected as it can be operated on a 24/7 basis with low power consumption. Furthermore, it offers enough non-volatile memory in the form of a secured digital (SD) card for all sorts of projects.

3. Evaluation criteria

The remainder of this paper is dedicated to the evaluation of different aspects related to sensor node systems based on selected microcontrollers compared to selected single-board computers.

First, Sect. 4 provides a comparison of various performance properties of selected microcontrollers and selected single-board computers. The performance criteria were selected due to their relevance to wireless sensor node systems: power consumption (see Sect. 4.1), execution time (see Sect. 4.2), and WiFi performance (see Sect. 4.3).

The microcontrollers and SBCs selected for comparison are meant to be representatives for their respective families of systems. In the realm of SBCs, the popular Raspberry Pi family was chosen, ranging from the low-performance Pi 0W to the highest-performance Pi 4B. It is believed that this range of systems is representative for general purpose SBCs, but it’s
performance may not be representative for special purpose SBCs, e.g., the Jetson series of NVIDIA Holly et al. (2020) for deep learning applications. As representatives for the microcontrollers, the powerful and popular ESP8266 and ESP32 modules Maier et al. (2017), mentioned in Sect.2, as well as the recently increasingly popular Raspberry Pi Pico, were chosen. While these choices are representative through their popularity, the field of microcontrollers used in sensor systems is very heterogeneous Karray et al. (2018); Khalifeh et al. (2022) and some results may not generalize.

Second, Sect. 5 focuses on the evaluation of a specific sensor node system implementation. This section compares the ESP8266 and ESP32 microcontrollers in terms of their abilities to run on rechargeable batteries (see Sect. 5.2 and Sect. 5.3) as well as the impact of their energy saving feature of deep sleep mode (see Sect. 5.4). The experiment description also contains qualitative discussions of collected sensor data (see Sect. 5.5).

Note that this experiment should be seen as a validation of the design suggested in Sect. 2 rather than a direct comparison between microcontrollers and SBCs as in Sect. 5. In particular, the power requirements of the Raspberry Pis is too high for the battery setup used in Sect. 5.

The comparison of market availability of the electronic components used in this paper is not part of the evaluation.

4. Hardware component benchmark tests

To evaluate the characteristics of microcontrollers and compare them to single-board computers, a hardware benchmark was performed. All experiments considered for the benchmark were conducted indoors in a regular middle-European environment at
approximately 22°C. In order to make the results of the following benchmarks between microcontrollers and single-board computers comparable and to rule out the influence of differently charged batteries, all devices were supplied with power via a laboratory power supply. Beside the previously mentioned ESP32 and the ESP8266, Raspberry Pico (Raspberry Pi Trading Ltd, 2021) also represented the microcontrollers, while some already existing models of the Raspberry Pi family constituted the group of single-board computers, namely the Raspberry Pi 2B, Raspberry Pi 3B, Raspberry Pi 3B+, Raspberry Pi 4B and the Raspberry Pi Zero W. These devices exemplarily represent the different performance classes available on the market. The Raspberry Pi 2B was added to the selection to examine if devices without a built-in WiFi module are still relevant by the usage of an additional USB WiFi dongle. The microcontrollers were run at different clock frequencies because it is expected that they will consume less power while operating at low than at high frequencies.

4.1 Power consumption

An experiment was conducted to record the power consumption of the selected microcontrollers and single-board computers. Both the microcontrollers and the single-board computers were supplied with a voltage of 5V via universal serial bus (USB). The flowing current in milliamperes was measured between the positive output of the power supply and the positive input of the device. For this purpose, a UNI-T UT139B multimeter set to max-setting, which only displays the highest seen value, was used. Measurements are reported in Table 1.

Since neither microcontrollers nor single-board computers are continuously operated under full load, the measurements for all devices were performed in different scenarios: idle,
high load, network transmission, if available without a keyboard, without a connected screen, and without a connected WiFi dongle. Some conditions were also measured in combination.

**Idle** (Table 1, columns 2-3) For the idle state, two approaches were used. On one hand, measurements of the Raspberry Pi devices were conducted after the system had fully loaded. On the other hand, measurements of the microcontrollers were performed by executing a program consisting of only a single sleep instruction.

**High load** (Table 1, columns 7-8) For the high load scenario, a self-built prime number search was used. This program creates different amounts of CPU stress based on the upper search limit and the number of threads. Both the single-board computers and the microcontrollers used an upper limit of one million. For the single-board computers, the maximum number of available threads was used. The microcontrollers used a slightly modified version without threading, as this functionality is not supported in the same way as in regular C++.

**Network** (Table 1, column 9-10) Different approaches on the single-board computers and the microcontrollers were used for the networking scenario. The single-board computers used the Linux tool iperf3 (networking speed measurement) to create a high load on the networking device. On the microcontrollers, a self-built program was used, which transmitted random numbers as quickly as possible.

**Keyboard/screen/WiFi dongle** (Table 1, columns 4-6) All single-board computer devices were used with a keyboard and a screen attached. As the Raspberry Pi 2 has no built-in WiFi module, an additional WiFi dongle was used. By systematically unplugging these connections,
their influence was measured. All of these measurements were performed in an idle state.

**Deep sleep** (Table 1, column 11) One crucial feature of the microcontrollers, which is not available on the Raspberry Pi, is *deep sleep*. A Raspberry Pi has different power states as well but is not able to wake up or restart by itself. The microcontrollers either have a timer-based wake-up, an event-based wake-up (e.g., voltage high/low on a specific pin), or both. As all manufacturers claim that their devices require a low amount of power in the deep sleep state, a test was conducted to prove this claim. Based on the microcontroller software development kit (SDK) documentation, a deep sleep program was created for all microcontrollers, and the flowing current was measured again.

The results of the power consumption benchmarks are summarized in Table 1. They clearly show that microcontrollers require significantly less energy than single-board computers, which was expected. But the magnitude of the differences in the different scenarios is interesting. While there is only a slight difference between the idle and full load state with the microcontrollers, single-board computers require between 1.8 to 3 times more energy in the full load state than in the idle state. This also shows that the tested microcontrollers do not have any automatic power-saving mechanisms. If, similar to the case with single-board computers, energy is to be saved when the load is low, the developer himself must carry out the necessary steps. This can be achieved, for example, by dynamically adapting the CPU clock rate or by switching various components of the microcontroller off. However, such an approach leads to much more complex programs.

### 4.2 Execution time of programming languages
To assess the computational capabilities of microcontrollers and SBCs in the comparison, a unified benchmark was conducted. The time taken by a single core per device to calculate prime numbers up to the limit of one million was measured. The running times of the programs are reported in Table 2.

**Performance with C++** The program for this benchmark was written in C++ and it is identical for the single-board computers and microcontrollers.

**Performance with Python** All chosen microcontrollers support MicroPython in addition to C++. Therefore, the existing prime search benchmark was ported from C++ to python, and the experiment was repeated for all devices with an identical program. All tests were conducted in the MicroPython runtime environment version 20210418-v1.15. No additional libraries except those from the MicroPython environment were used to implement the benchmark.

The learning effort of python is lower than that of C++ Balogun (2022); Alzahrani et al. (2018) and tool chains like CMake. This enables quicker development and less complex code. However, C++ is a statically typed, compiled language, and MicroPython is a dynamically typed, interpreted language. In addition, in MicroPython, there is no compilation into bytecode as in the regular python version. The direct comparison from the test results (see Table 2) shows that, at least for large computational tasks, python is not well suited within a microcontroller environment. Over all tested microcontrollers and frequencies, C++ is approximately 47 times faster on average, and on the ESP32 more than 67 times.

### 4.3 WiFi performance

A separate experiment was conducted to determine whether the available WiFi bandwidth is
sufficient for the data transmission of the microcontrollers and how many messages can be sent per second. For this purpose, messages containing 100 characters and messages containing 1000 characters were transmitted. The minimal, maximal, and average transmission time for 100 runs were determined.

The measured values of the WiFi transmission experiment (Table 3) clearly show that both ESPs can send more than 20 messages per second on average. Interestingly, despite the older CPU architecture and only one core, the ESP8266 requires an average of 50% less time to transmit messages. In addition, it only takes an average of 20% more time to transmit a message that is 10 times longer.

5. Monitoring experiments

To evaluate the application of the wireless sensor node system for monitoring purposes, additional experiments were conducted. The aim of performing these experiments was to assess the runtime available for monitoring processes and to check whether microcontroller-based nodes are suitable for measuring environmental parameters, in particular temperature, humidity, and illuminance. An illustration of the sensor node setup is shown in Figure 3.

5.1 Experiment 1: battery discharge

To evaluate the use in an off-the-grid setting, the capabilities of rechargeable batteries had to be tested. Previous tests of the current and voltage measuring sensor INA219 and the intended battery capacity allowed a rough estimate of the MCs’ running time of up to 24 hours on battery. However, this estimation assumes a constant current and voltage output over the whole
runtime. Therefore, to visualize the batteries’ behavior, a small battery discharge experiment was performed.

The experiment consisted of an ESP32 powered by rechargeable batteries and an INA219 voltage and current sensor. A second set of rechargeable batteries together with a power resistor formed the measurement circuit. The INA219 was installed on the circuit’s HIGH side, directly behind the batteries, for correct voltage measurements. Only for current flow measurements, the position in the circuit would have been irrelevant. Figure 4 shows the schematic configuration of the experiment. Before the experiment could be started, a discharge current had to be set. As indicated by research results on battery discharge currents, a constant current flow of more than one ampere may damage the batteries or lead to other unwanted effects. Furthermore, even if up to three amperes would have been possible with the INA219, the other cables needed to withstand the current flow too. Based on the battery pack’s voltage of not more than 6V and a maximum current of 1A, the resistance had to be at least 6 Ohms. Together with the batteries’ capacity of 2800mAh, an experiment duration of approximately 3 hours was estimated.

Figure 5 shows the results of the current and voltage measurements during the batteries’ discharge. The plot clearly shows that current and voltage are directly connected: if the voltage decreases, the current does too. By looking at the initial peak, one can see that the maximum voltage and current can only be held for a brief moment. After 30 to 40 minutes, the decrease settles for a short period before slowly decreasing again. After approximately 3 hours and 40 minutes, the voltage drops significantly, which is also referred to as voltage breakdown. After four and a half hours, the voltage has fallen below 1V. In the following hours, the current and
voltage gradually approached zero. The entire record consisted of more than 50,600 measurements in 14 hours. The collected data after hour eight is not considered relevant, thus the graph only shows the first eight hours. A summation of the current measurements over the first four hours resulted in an electric charge delivery of around 2470mAh. Additional 160mAh were delivered from hours 4 to 5 (2630mAh total). Over the whole 14 hours, around 2720mAh were delivered. This means that the effective charge delivery before the voltage breakdown is about 88 to 93 percent of the batteries rated capacity of 2800mAh. A possible reason for the deviation could be their age. Even if bought as new, there is some timespan between production and delivery, where they are not used, which degrades the effective charge.

5.2 Experiment 2: runtime of ESP sensor nodes

For this experiment, two sensor nodes were utilized. One node was an ESP8266, and the other one was an ESP32. Both nodes were equipped with an identical sensor configuration. One node was placed in the same room as the WiFi router, whereas the other was placed in the next room to evaluate the WiFi transmission quality of the microcontrollers’ small antennas. Both nodes were placed on a windowsill where the window had no louver or drapes.

One of the most important questions was how long the nodes could run on battery power. Based on the power consumption measurements from the previous section, the battery capacity of 2800mAh, and 5-10 seconds for a complete measurement iteration, an uptime of one week was expected. In total, the ESP32-based node achieved an uptime of 2.5 days, whereas the ESP8266-based one achieved a little more than six days. This can be seen in the data evaluation in Figure 6. For better readability, the daytime of all measurements was aligned with the x-axis.
For example, 6 AM on the first day of the experiment corresponds to 0.25 on the x-axis. By looking at the voltage levels of both nodes, one can see that the ESP32 was more sensitive to the voltage decrease. It stopped working at a converter input voltage of 4.9V, whereas the ESP8266 was still operational down to 4.2V. The reason for this behavior was the chosen voltage converter LM317 (Texas Instruments, 2020). It was configured to deliver 3.3V using the battery’s fully charged supply voltage of 5.8V. Any other configuration would have delivered a too-high voltage, resulting in possibly damaging the ESPs. However, this configuration led to the earlier shutdown of the ESP32. Yet, even if not entirely successful, the results enabled many different analysis options. Some of these are presented in section 5.5.

5.3 Confirming early shutdown of ESP32

A second run was done to check whether the early shutdown of the ESP32-based node could have been battery-related. Figure 7 shows a comparison of the voltage measurements from these two passes. It’s visible that the ESP32 did not perform much differently. Again, after 2.5 days, the ESP32 stopped working. The measurements for the ESP8266 were then also stopped. Therefore, one could conclude from this evaluation that the early breakdown is not related to faulty batteries. Interestingly, the battery pack’s voltage decreased much faster compared to the first pass of the experiment. To check a possible dependency on the temperature influences, Figure 7 also contains the temperatures measured in both experiments. One can see that at the time of the second run, the average temperature was only a little higher than in the first run. However, especially around noon, very high temperatures of up to 50°C were measured in the second run. Even if the ESP32 can be used up to 105°C or 125°C, according to its datasheet
Systems (2021), these high temperatures could have already influenced the device. An indicator of this influence is two voltage drops around noon. Generally, high ambient temperatures significantly decrease the heat dissipation capabilities of the components to the surrounding air, which can lead to continuous heat up of the device.

5.4 Impact of disabling deep sleep mode

An additional iteration of the experiment was performed to evaluate the impact of deep sleep mode on power consumption. In this experiment deep sleep mode was turned off. The ESP32 node used a more rigid battery case, but the same batteries in all other runs and both nodes were placed beside each other. The evaluation of the measurement results did not show any significant impact of changing the battery box of the ESP32.

Figure 8 shows the voltage measurements of the first and third iteration in comparison. One can clearly see that the usage of the deep sleep mode roughly doubled the runtime of both nodes. Interestingly in this iteration, the ESP32-based node was operational down to a voltage converter input voltage of 4.6V, which is an increase of voltage range by 0.3V or 42 percent. Even if 0.3V is not a large difference, the increased range shows that the used wires (especially their wire gauge) significantly influence the devices’ power supply. This observation is consistent with the ones made during the MCs’ and SBCs’ current measurements. Therefore, it can be said that the largest possible cable cross-section should be selected to keep voltage losses as low as possible.

5.5 Collected sensor data
This section discusses observations of the sensor data collected during the experiment described in Sect. 5.2.

5.5.1 Temperature and illuminance

In the same experiment, the temperature and illumination measured by the two sensor nodes were compared. By looking at Figure 9, one can see that most measurements of the ESP32-based node are above the ones from the ESP8266-based node. This result is not surprising, as the windowsill on which the ESP32 was placed faced roughly south, whereas the ESP8266’s windowsill faced roughly north. As a consequence, the ESP8266 node received more sunlight in the early hours of the day, but shortly before noon, until the later afternoon, the ESP32 received up to five times the amount of sunlight. The combination of the temperature and the illuminance shows that the temperature is directly related to the sun. If this experiment had been conducted in the winter months, it would have probably looked different.

5.5.2 Temperature and humidity

As already mentioned in the experimental setup, each node was equipped with a BME280 and a DHT11 sensor, which makes a comparison of the temperature and air humidity values possible. Figure 10 shows the results of measuring the same quantity with different sensors. At first look, the deviation between both sensors is not high. However, by additionally calculating the absolute difference between both, the picture changes slightly. Especially the ESP32-node’s measurements in the warm and sunny hours around noon reached a difference of up to 14 degrees. In comparison, the ESP8266-node’s measurements in a shady location only differed by
some minor degrees.

Compared to the temperature values, the humidity measurements seem to have a nearly constant difference of approximately 5% - 10% depending on the node’s location. In combination with the daytime, one can see that the humidity is highest at the night and lowest around noon. Furthermore, a high sun intensity lets the humidity drop significantly, which can be seen around noon on day two. At the end of the experiment, the DHT11 on the ESP8266 was artificially exposed to higher humidity and the measurement data reflects this (the node reached almost 100 percent humidity).

6. Observations and possible extensions

From a programming perspective, a major difference is that abstraction layers and services of regular operating systems offering unified access to resources are typically missing from microcontrollers. Within a microcontroller environment, solutions are often coupled to a specific chip or configuration. Sometimes connections have to even be hard-wired to become functional. Additionally, the consumption of resources, like time and memory, by algorithms need to be considered more carefully, as they are only available in very limited quantities.

All the microcontrollers used offered a variety of interfaces via their pins. By using PC for all experiments carried out, at least three sensors could be operated via the same interface, while the specification theoretically allows up to 127 devices. Additionally, one sensor was operated via the GPIO interface and a debugging device was connected via the UART. In total three interfaces with five devices were operated in parallel. If the number of PC devices is exhausted on ESP32s, more could be added via a second channel provided. However, in direct
comparison to Raspberry Pi extension boards, e.g. the PiXtend, interfaces like RS232, RS485, etc. are not provided by the used microcontrollers and would require specialized adapter boards.

One direction of further research could be reduced power consumption. As the used wireless transmission technology WiFi had a peak current of 500mA, specialized, IoT-centered, and low-power transmission technologies like ZigBee or WiFi 802.11ah (WiFi HaLow) are promising. Furthermore, a custom-built printed circuit board (PCB), tailored toward energy efficiency, may further reduce power consumption. Additionally, recharging the batteries via solar panels for outdoor use-cases could be evaluated.

Throughout this work, at most, three microcontrollers were used in parallel. However, with a large-scale deployment, issues of managing distributed, parallel systems will arise. These problems mainly relate to shared resource access and timing. One example of these issues are messages arriving at the server simultaneously. Therefore the server must be capable of handling these requests without mutual exclusion.

The experiments carried out focused on non-local storage by sending measurement results to an external device (server). However, a network connection is not always available thus local storage can be an option. The market research showed that Micro SD card readers with an PC interface are available for around five euros, thereby enabling data storage on non-volatile memory. For example, the battery discharge experiment produced 5 megabytes of data in 11 hours, taking measurements every second. This corresponds to approximately 2250 hours (93 days) of recorded data per gigabyte SD card. Therefore local storage is considered being an promising alternative to external storage or as an additional backup.
Only a single WiFi router was necessary for the small spatial expansion of the experiments. However, applications are also conceivable in which longer distances have to be covered, or in which a single router cannot cover the entire area of the application. An example of this would be the use of a sensor network over several floors of a building. Depending on the structural conditions, it may not be possible to install additional WiFi access points. In such a case, mesh networks are a potential solution. The manufacturer of the ESP microcontroller already offers its own framework for this purpose. There is also a framework called “painlessMesh” that aims to make setting up such a mesh network particularly easy. Regardless of which framework is used, there are also other advantages, such as time synchronization across all participating nodes or automatic selection of the best route. However, the individual nodes must remain switched on for this and cannot switch to sleep mode. In general, however, the use of a mesh network enables a much simpler installation without additional cabling and is therefore also well suited for temporary measurements.

Typically, microcontrollers have to be flashed by plugging in a cable. This means that either the microcontroller has to be brought to the computer or vice versa. This solution is unsatisfying or even impossible, especially if the sensors are to be used in hard-to-reach places or permanently installed. One possible solution to this problem is over-the-air (OTA) updates. In this case, the regular program code is expanded to include an updater with various update strategies. This means that the program on the microcontroller can be changed at a later date, even without a separate cable. In combination with an SBC, a remote update could be carried out for all sensor nodes in the WiFi. A catch in this solution, however, is the WiFi connection.
No problems should arise for a WiFi-based network, but if, for example, ZigBee is used, a WiFi connection must be established for the update in addition. Furthermore, depending on the strategy chosen, some errors can occur so that the microcontroller is no longer accessible, does not work correctly, or cannot be updated again. Therefore, update strategies should also include a solution for a possible failed update.

7. Conclusion

In general, microcontrollers have proven to be an alternative to single-board computers for monitoring purposes (see comprehensive surveys in Karray et al. (2018); Khalifeh et al. (2022)). Although they are available at low prices, they provide a variety of interfaces via their pins, which makes the attachment of several types of sensors possible.

In this work, a wireless sensor node system was been proposed and implemented. Furthermore, microcontrollers have also been evaluated as devices for monitoring applications. These microcontrollers were compared to single-board computers by conducting hardware benchmark tests. The results of the power measurements have shown that MC-based systems can be operated at significantly lower power levels, enabling extended self-sufficient measurement recording compared to SBC-based systems. The ESP8266 and the ESP32 have proven to be alternatives to SBCs for data acquisition, whereas the ESP8266 could be operated for longer, while the ESP32 could handle more concurrent tasks. Furthermore, the execution speed of C++ and python on MCs was also evaluated, showing that C++ wins for speed, whereas python convinces with its ease of use.
There is a wide variety of SBC and MC systems on the market. The choice of system depends on the requirements. Compared to MCs, SBCs have more processing power, integrated graphics, and more memory. On the other hand, they usually require more power to operate. On the software side, SBCs can run high-level real-time operating systems, while MC systems can only run very simple ones. In principle, SBCs and MC systems offer different hardware and software interfaces, while wireless and standardized radio modules for WiFi and Bluetooth, for example, are often only provided on SBCs. These key differences between SBCs and MCs have implications for the complexity of building application-oriented electronics. Because SBCs are mostly complete computers, there is little need to extend them to meet requirements. This includes, for example, installing an operating system provided by the manufacturer or supported by the SBC, or connecting certain peripherals, such as sensors and their drivers, as needed. In contrast to SBCs, when MC systems are put into operation, the corresponding firmware or machine code must always be developed according to the required functionality. In some cases, prefabricated solutions and libraries can be used if standard functions are to be used. If additional peripherals are to be connected, the firmware must be extended accordingly. SBCs and MCs are therefore complex hardware and software systems. However, a microcontroller-based system requires additional hardware and software development for application-oriented commissioning.

Acknowledgements

This work was partially supported by the Carl Zeiss Foundation. The financial support is gratefully appreciated. Any opinions, findings, conclusions, or recommendations expressed in
this paper are those of the authors and do not necessarily reflect the views of the aforementioned institutions.
References


Bosch Sensortec (2015) BME280 - Combined humidity and pressure sensor (Datasheet - Revision 1.6). Bosch Sensortec GmbH.


Kanagachidambaresan GR (2021) Role of Single Board Computers (SBCs) in rapid IoT Prototyping. Springer Cham.


standard microcomputers and smartphones for applications in structural health

NXP Semiconductors (2021) PC-bus specification and user manual (Revision 7.0). NXP
Semiconductors.

systems and an experimental set-up towards an improved design. Eurocon 2013 :
275–282.

microcontroller board. Raspberry Pi Trading Ltd.

Study. Recent Advances in Information Systems and Technologies, Springer International

Type Ambient Light Sensor IC (Datasheet). ROHM Co., Ltd.

Sheng C, Li Z, Qin L, Guo Z and Zhang Y (2011) Recent progress on mechanical condition


Technologies.


**Table 1.** Power consumption of all devices measured in mA. All microcontrollers were benchmarked with only a single core

<table>
<thead>
<tr>
<th>Device</th>
<th>idle</th>
<th>idle (BT+WiFi on)</th>
<th>idle no keyboard (BT+WiFi on)</th>
<th>idle no screen (BT+WiFi on)</th>
<th>idle no wifi dongle</th>
<th>high load</th>
<th>high load (BT+WiFi on)</th>
<th>network</th>
<th>network + benchmark</th>
<th>deep sleep</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pi 0W</td>
<td>89</td>
<td>93</td>
<td>86</td>
<td>88</td>
<td>-</td>
<td>161</td>
<td>186</td>
<td>276</td>
<td>297</td>
<td>-</td>
</tr>
<tr>
<td>Pi 2B</td>
<td>231</td>
<td>290</td>
<td>287</td>
<td>288</td>
<td>200</td>
<td>424</td>
<td>490</td>
<td>387</td>
<td>527</td>
<td>-</td>
</tr>
<tr>
<td>Pi 3B</td>
<td>225</td>
<td>224</td>
<td>221</td>
<td>221</td>
<td>-</td>
<td>677</td>
<td>687</td>
<td>422</td>
<td>823</td>
<td>-</td>
</tr>
<tr>
<td>Pi 3B+</td>
<td>368</td>
<td>370</td>
<td>367</td>
<td>368</td>
<td>-</td>
<td>945</td>
<td>936</td>
<td>605</td>
<td>1148</td>
<td>-</td>
</tr>
<tr>
<td>Pi 4B</td>
<td>388</td>
<td>394</td>
<td>389</td>
<td>388</td>
<td>-</td>
<td>875</td>
<td>889</td>
<td>656</td>
<td>1117</td>
<td>-</td>
</tr>
<tr>
<td>ESP8266 @160MHz</td>
<td>76</td>
<td>84</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>81</td>
<td>-</td>
<td>84</td>
<td>-</td>
<td>5</td>
</tr>
<tr>
<td>ESP8266 @80MHz</td>
<td>75</td>
<td>79</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>77</td>
<td>-</td>
<td>81</td>
<td>-</td>
<td>5</td>
</tr>
<tr>
<td>ESP32 @240MHz</td>
<td>51</td>
<td>160</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>71</td>
<td>-</td>
<td>149</td>
<td>-</td>
<td>11</td>
</tr>
<tr>
<td>ESP32 @160MHz</td>
<td>40</td>
<td>120</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>52</td>
<td>-</td>
<td>138</td>
<td>-</td>
<td>11</td>
</tr>
<tr>
<td>ESP32 @80MHz</td>
<td>34</td>
<td>102</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>40</td>
<td>-</td>
<td>120</td>
<td>-</td>
<td>11</td>
</tr>
<tr>
<td>Pico @300MHz (1.25V)</td>
<td>42</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>51</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td>Pico @240MHz (1.10V)</td>
<td>35</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>36</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td>Pico @160MHz (1.10V)</td>
<td>25</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>25</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td>Pico @125MHz (1.10V)</td>
<td>21</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>21</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td>Pico @80MHz (1.10V)</td>
<td>21</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>18</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>2</td>
</tr>
</tbody>
</table>
**Table 2.** Execution time of the programming languages in milliseconds

<table>
<thead>
<tr>
<th>Device</th>
<th>C++ in ms</th>
<th>Python in ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pi 0W</td>
<td>3964</td>
<td>108925</td>
</tr>
<tr>
<td>Pi 2B</td>
<td>3613</td>
<td>83689</td>
</tr>
<tr>
<td>Pi 3B</td>
<td>2233</td>
<td>49851</td>
</tr>
<tr>
<td>Pi 3B+</td>
<td>1917</td>
<td>42465</td>
</tr>
<tr>
<td>Pi 4B</td>
<td>1284</td>
<td>14438</td>
</tr>
<tr>
<td>ESP8266 @160MHz</td>
<td>25296</td>
<td>504000</td>
</tr>
<tr>
<td>ESP8266 @80MHz</td>
<td>50592</td>
<td>1016000</td>
</tr>
<tr>
<td>ESP32 @240MHz</td>
<td>2430</td>
<td>165000</td>
</tr>
<tr>
<td>ESP32 @160MHz</td>
<td>3658</td>
<td>247000</td>
</tr>
<tr>
<td>ESP32 @80MHz</td>
<td>7400</td>
<td>494000</td>
</tr>
<tr>
<td>Pico @240MHz (1.10V)</td>
<td>4477</td>
<td>205000</td>
</tr>
<tr>
<td>Pico @160MHz (1.10V)</td>
<td>6734</td>
<td>307000</td>
</tr>
<tr>
<td>Pico @125MHz (1.10V)</td>
<td>8638</td>
<td>393000</td>
</tr>
<tr>
<td>Pico @80MHz (1.10V)</td>
<td>13575</td>
<td>614000</td>
</tr>
</tbody>
</table>
Table 3. Time required to transmit a message of 100 and 1000 characters via WiFi in milliseconds (aggregated over 100 runs)

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Frequency</th>
<th>( t_{100\text{characters}} ) [( ms )]</th>
<th>( t_{1000\text{characters}} ) [( ms )]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Min</td>
<td>Max</td>
</tr>
<tr>
<td>ESP32</td>
<td>240MHz</td>
<td>27.0</td>
<td>105.0</td>
</tr>
<tr>
<td>ESP32</td>
<td>160MHz</td>
<td>34.0</td>
<td>194.0</td>
</tr>
<tr>
<td>ESP32</td>
<td>80MHz</td>
<td>45.0</td>
<td>235.0</td>
</tr>
<tr>
<td>ESP8266</td>
<td>160MHz</td>
<td>13.0</td>
<td>93.0</td>
</tr>
<tr>
<td>ESP8266</td>
<td>80MHz</td>
<td>16.0</td>
<td>214.0</td>
</tr>
</tbody>
</table>
Figure 1. A reference design of a wireless sensor node system based on microcontrollers. S1 - S4 denote sensors and Iface denotes an interface.
**Figure 2.** Concretization of the reference design from Figure 1 where ESP32 / ESP8266 are chosen as microcontrollers and a Raspberry Pi 3 is chosen as server component.
Figure 3. The ESP8266 sensor node setup for monitoring experiments of Sect. 5
Figure 4. Battery discharge schematic structure
Figure 5. The result of the battery discharge measurement described in Sect. 5.1 plotting voltage and amperage over time in hours.
Figure 6. Results of the voltage measurement
Figure 7. Comparison of the voltage and temperature measurements (exp01 from Sect. 5.2 and exp02 from Sect. 5.3)
Figure 8. Comparison of the voltage measurements with and without deep sleep (exp01 from Sect. 5.2 and exp03 from Sect. 5.4)
Figure 9. Results of the temperature and illuminance measurement (see Sect. 5.5.1)
Figure 10. Results of the temperature and humidity measurement (see Sect. 5.5.2)