Advanced Sensorless Weather Station Implementation Using ESP32

Jugurtha Chettouh[1] and Samia Mezzah[2(✉)]

[1] Faculty of Technology, Bejaia University, 06000 Bejaia, Algeria
[2] Laboratory of Medical Informatics (LIMED), Bejaia University, 06000 Bejaia, Algeria
samia.mezzah@univ-bejaia.dz

Abstract. The Internet-of-Things (IoT) concept has been largely integrated into heterogeneous systems and it has recently become more relevant to practical implementations because of the growth of ubiquitous communication and remote data access techniques. This paper presents an IoT system that basically gathers weather-related data information from a forecast API (Application Programming Interface) for a specific configurable location and displays them on TFT touch screen. The implemented system uses a low-cost ESP32 microcontroller and exploits its integrated Wi-Fi to achieve transmitting and receiving communications. Moreover, we develop an embedded web server on ESP32 using SPIFFS (Serial Peripheral Interface Flash File System) to monitor and control power consumption mode over a local network and a secured internet connection via a laptop, cellphone, or tablet. These functionalities are implemented by using FreeRTOS that provides multitasking support for the ESP32 dual-core microcontroller.

Keywords: IoT system · ESP32 FreeRTOS · Sensorless weather station

1 Introduction

Progress in electronic and telecommunication fields are currently boosting an important technological shift that changes human world interaction [1]. These new technologies enhance greatly data availability and enable access to both useful services, like information aggregation for well-being, comfort and productivity, and vital services such as health care and education. IoT, which is the most important emerging technology, consists of connecting physical objects to the Internet [2]. The size, power consumption, and cost of electronic components that are needed to support Internet use capabilities play a critical role in the widespread adoption of IoT [3]. In this sense, System-on-Chip (SoC) solutions, like ESP microcontroller chips, offer low cost and efficient components that integrate all the necessary connectivity modules of the complete system on a single chip.

Initially, the ESP chip, with its built-in IEEE 802.11 b/g/n networking protocol support module, was mainly used as a second low-cost controller to provide Wi-Fi capabilities and network access for another controller via a serial port [4]. However, more and more projects began using the spare processing power of these chips to also provide

M. R. Senouci et al. (Eds.): CSA 2020, LNNS 199, pp. 165–174, 2021.
https://doi.org/10.1007/978-3-030-69418-0_15

general I/O control so that in many cases no additional control processor was required [5, 6]. This prompted Espressif, the chip's manufacturer, to bring out the next generation of chips, ESP32, that contain additional functional modules and more powerful processing cores. ESP32 chips work with a clock rate of up to 240 MHz and integrate [7]: a Tensilica Xtensa LX6 32-bit CPU in both dual-core and single-core versions, an additional power-efficient coprocessor, 36 GPIO pins, dual-mode BLE (Bluetooth Low Energy), 520 KB SRAM, 448 KB ROM, in-built antenna switches, power amplifier, low-noise receive amplifier, hardware accelerators for AES/SSL/TLS and many Input/Output peripherals that include capacitive touch, ADCs, DACs, I^2C, UART, SPI, I^2S, RMII, …etc.

The ESP32 Chip is highly complex and cannot easily be used on its own especially for prototyping. Therefore, several development boards are available that incorporate the ESP32 chip and associated hardware to simplify the task of project development. Some of the most popular ESP32 development boards are: Adafruit Huzzah32, NodeMCU-32 s and ESP32 DevKitC [5]. In this work, we used an Adafruit Huzzah32 board that contains the ESP32 WROOM32 module, which, in turn, contains the dual-core ESP32 D0WDQ6 chip and a 4 MB SPI flash memory (see Fig. 1).

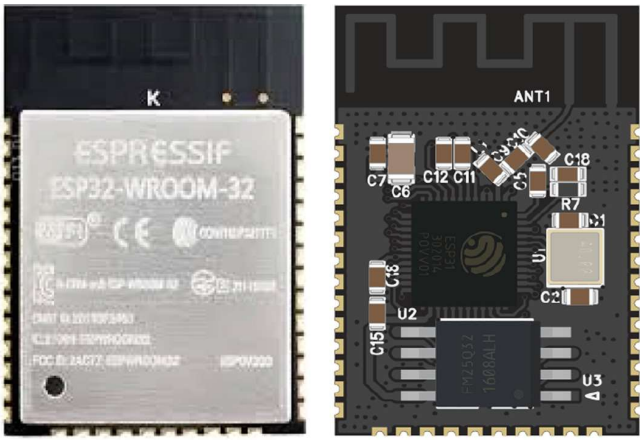

Fig. 1. The ESP-WROOM-32 module

2 System Description

Traditional weather station systems require many sensors, data monitoring and collection systems as well as regular maintenance equipment and skilled operators [8]. Alternatively, sensorless weather stations offer a good solution for low-cost application with minimum effort. The realized prototype is deployed to get and display weather information of a specified location in real-time [9]. Both the transmitting and receiving communications are managed using integrated WiFi. This system pulls weather data using the OpenWeatherMap API. Beside this information gathering functionality, the ESP32 assure 3 other functions which are (see Fig. 2):

1. Display control: to display date, time, Wi-Fi strength, current weather conditionals, 4-day forecast and moon phases.

2. Web server: to allow remote control of power mode (normal, screen off or deep sleep) through a Web portal or a mobile application.
3. Touch detection: to manage activation/deactivation of deep-sleep mode using a touch sensor.

3 Hardware Components

In this work, we use the HUZZAH 32 board [10] and a TFT FeatherWing touch screen [11]. The TFT FeatherWing is a display shield with built-in microSD card socket. This TFT display is 2.4 inches diagonal with a bright four white-LED backlight. It displays 16-bit color images of 240 × 320 pixels. This display comes with an attached resistive touchscreen. It also includes a reset button and an on/off switch connected to the Feather Enable pin. The TFT display, SD card and touch screen use the SPI interface to communicate with the controller; That means MISO, MOSI and SCK ports are used whenever either are accessed.

Great addition to the used board is the incorporated Lipoly battery charger. This will let the feather run on a rechargeable battery. Hence, to supply this embedded system with power, we used 3.7 V Lithium Polymer (Lipo) battery. To economize battery energy, we

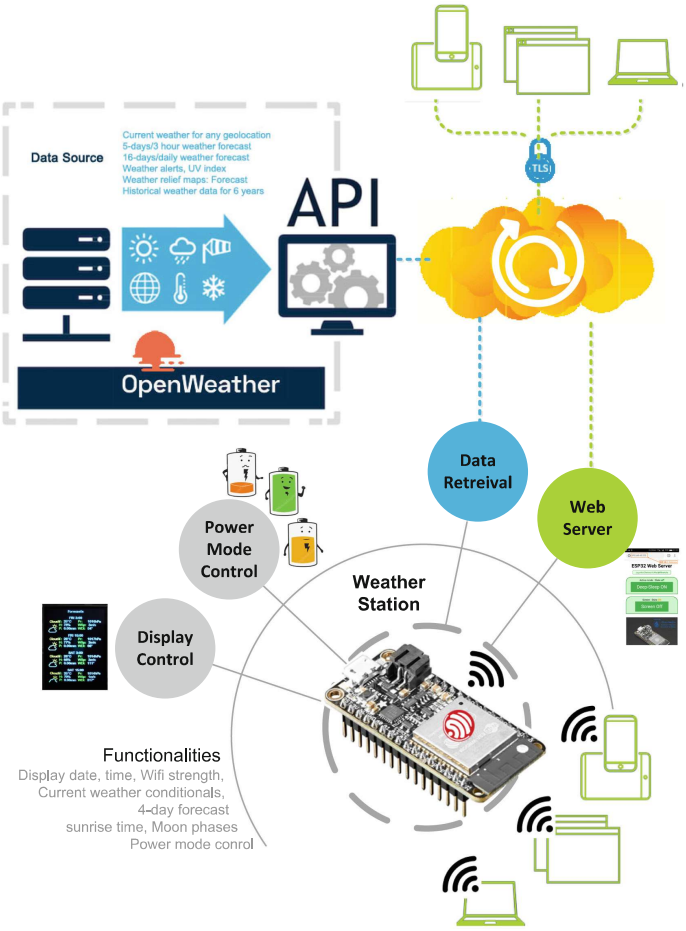

Fig. 2. System functionalities

added power mode control using a touch sensor capability of pin A4. Details of hardware components and pin assignment are given in Fig. 3.

4 Software Implementation

Since we use compatible hardware components, the prototype assembly is simple without any extra wire or need for electronics components. Hence, a major effort is devoted to software development.

The ESP32 application code was developed using Arduino 1.8.7 IDE, FreeRTOS and many hardware libraries. To program the ESP32 development board using Arduino IDE, the ESP32 toolchain has to be installed which uses "xtensa-esp32-elf-gcc" & "xtensa-esp32-elf-g++ " compilers. The most important extension feature of this library is the FreeRTOS kernel support. The implementation of FreeRTOS support for the ESP32 microcontroller is very important because it provides support for multitasking.

4.1 Multitasking Programming

The used ESP-IDF (IoT Development Framework) FreeRTOS is a modified version of vanilla FreeRTOS which supports symmetric multiprocessing (SMP) [12]. ESP32 is dual-core containing a Protocol CPU (known as Core 0) and an Application CPU (known as Core 1). The two cores are identical in practice and share the same memory. This allows the two cores to run tasks interchangeably between them.

In order to optimize the code execution, it is recommended to run modem tasks (Connecting to Wi-Fi, transmitting and receiving packets …etc.) independently from the other tasks; So that those processes will not interrupt the main core which may have heavy tasks (display tasks in our case). Since the graphical tasks should be initialized before modem tasks, we put them on core1, and gave them higher priority and stack size because those tasks have higher CPU and memory usage (see Fig. 4).

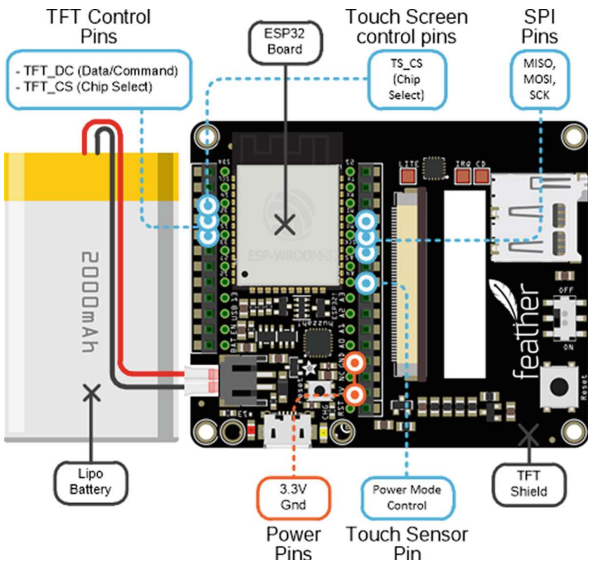

Fig. 3. Weather Station Hardware components.

4.2 Display Tasks

For display code, we use Mini_Grafx library which is a graphics library for embedded devices containing a framebuffer to avoid flickering: all drawing operations are made on buffer stored in memory. When drawing writing is finished, the whole content of the frame buffer is displayed. The library contains drawing routines and the driver for ILI9341 based displays. This driver considers 565 coded color (5bit for red, 6bit for green and 5bit for blue). The available configuration function allows the setting of bit depth and palettes which determine the buffer size.

4.3 Weather Data Collection

ESP32 can operate in three different modes: Station mode, Soft Access Point mode, and both at the same time. In this work, we configured the ESP-32 to act in station mode since it needs internet access to send and receive requests. Modem tasks are based on not blocking WiFi library function calls. For data retrieval, ESP32 sends HTTP requests to OpenWeatherMap API which will respond with a JSON file containing weather information (forecasts, Current Conditions…etc.) of the given location as shown by Fig. 5. The GET HTTP request used contains the location (City ID), an identification key (API key), and additional configuration parameters.

The Json_Streaming_Parser library is used for (byte by byte) parsing json streams because it is well adapted to embedded systems with limited memory by using Json-Listner module that generate notification when interesting parts in the feed are received. The esp8266-weather-station library is also used. It contains client modules to fetch data from given number of weather servers, Time client module to set the clock and functions to calculate lunar phase and illumination, sun and moon properties for a given date, time and location.

```
//graphical task definition with priority 3 to be executed on core1
xTaskCreatePinnedToCore(
  Graphical_tasks,     /*Task function*/
  "Task1",             /*Task name*/
  12000,               /*Task Stack Size*/
  NULL,                /*Task parameter pointer*/
  3,                   /*Task priority level = 3*/
  &Task1,              /*Task handle*/
  1);                  /*Execution core1 selected*/
Delay(500);
//Modem task definition with priority 1 to be executed on core0
xTaskCreatePinnedToCore(
  Modem_tasks,
  "Task2",
  10000,
  NULL,
  1,                   /*Task priority level = 1*/
  &Task2,
  0);                  /*Execution core1 selected*/
Delay(500);
```

Fig. 4. Multitasking parameters

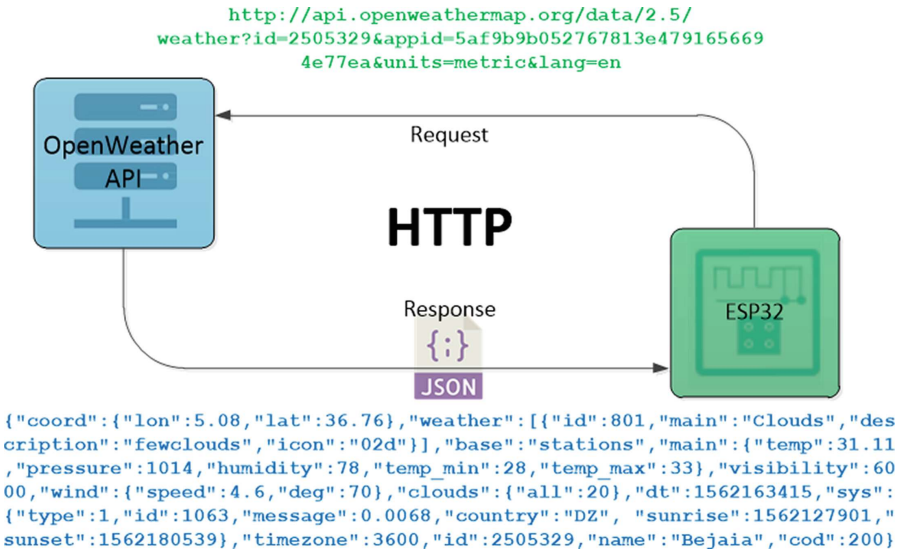

Fig. 5. Data retrieval protocol

4.4 ESP-32 Web Server

The web server is used to control power consumption mode through the web. Hence, Users can control the Screen and Deep-Sleep status using the web server page. Figure 6 explains This process. The developed web server, which serves HTML and CSS files stored on the ESP32 filesystem, is based on SPIFFS (Serial Peripheral Interface Flash File System) library. SPIFFS is a file system intended for SPI flash devices on embedded targets.

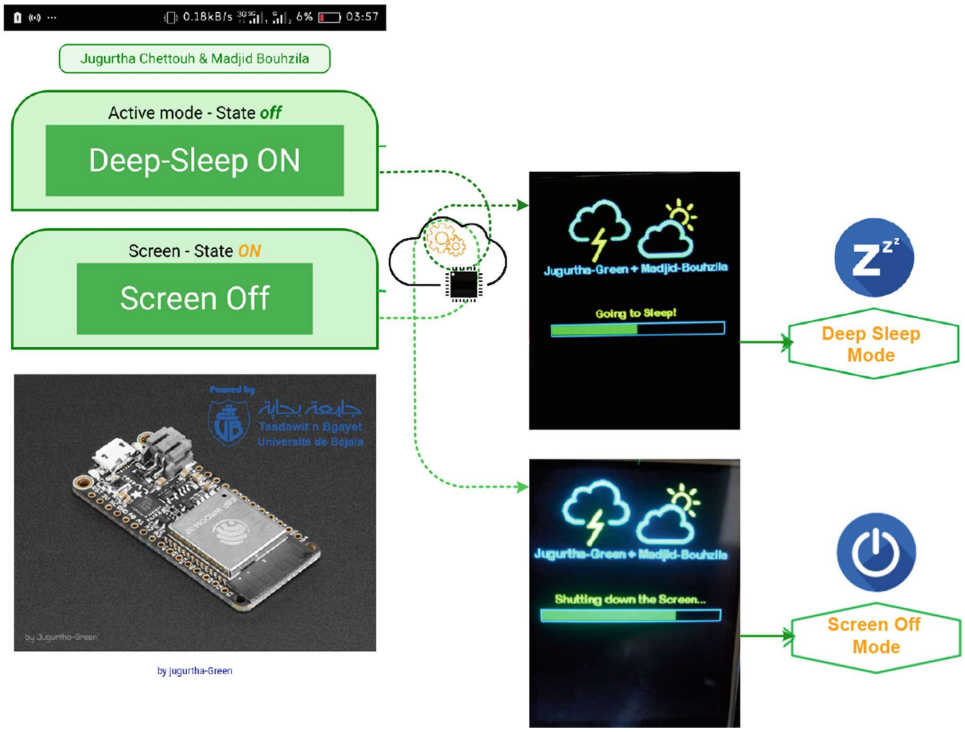

Fig. 6. Web Server page and Power Control functions.

mDNS Web Server implementation. One of the greatest features ESP32 provides is that it cannot only connect to an existing WiFi network and act as a Web Server, but it can also set up a network of its own, allowing other devices to connect directly to it and access web pages. mDNS stands for multicast Domain Name System. It is a network protocol for resolving hostnames to IP addresses in small local networks. Hence, we can easily access to the ESP-32 Web Server using local name server (in our case it is esp32.local) instead of IP address. Users can also access the ESP-32 Web Server from outside (over Internet) without the need for Port Forwarding, this is done by an intermediate server like serveo.net or ngrok.io which are a combination of SSH, HTTP & TCP server that create a sub-domain (ours is "esp32-jugu.serveo.net") that forward traffics to the local network from the internet. In addition, we created an android application, which allows access to the ESP Web Server in one click.

Security Layer implementation. We added an extra layer of security for control access capabilities to deny access for unauthorized users when a user tries to access the ESP-32 over internet. The authorized user has to authenticate using "username" and "password" in order to control the device using http requests over the ESP-32 Web Server. Authentication operations are done with an external server using PHP as a back-end programming language. Figure 7 shows the process of Authentication. The login page has been created using HTML5, CSS3, and JavaScript. The web server code uses WiFiClintSecure library that implement support for secure connections using TLS/SSL (Transport Layer security/Secure Socket Layer). There are three ways to establish a secure connection using WiFiClintSecure: using a root Certificate Authority (CA) cert, a root CA cert plus a client cert and key or a Pre-Shared Key (PSK).

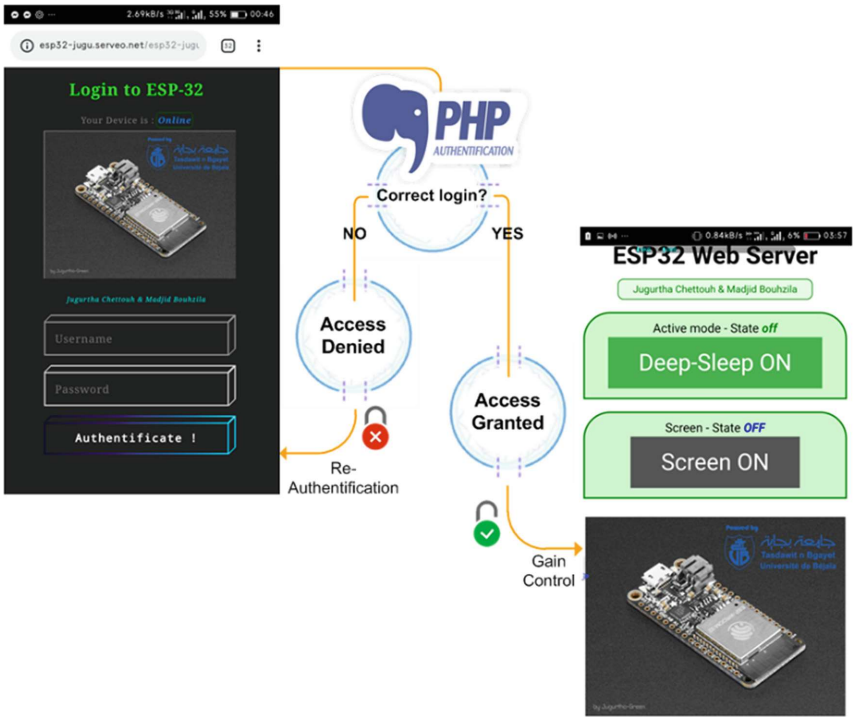

Fig. 7. Authentication process

5 Implementation Results

For the implementation, we chose the Arduino platform as a development environment, into which we have installed ESP32 development support. We proceeded to implement each functionality separately and then test the components. After completing the entire assembly, we performed the testing of the entire embedded system.

Figure 8 shows the prototype of our embedded system. As shown in Fig. 9 and Fig. 10, the developed code controls the TFT screen to display weather information, according to the data retrieval configuration, and to show the status of different functional processes (initialization, updating data, activation of power modes …etc.).

Fig. 8. The final prototype

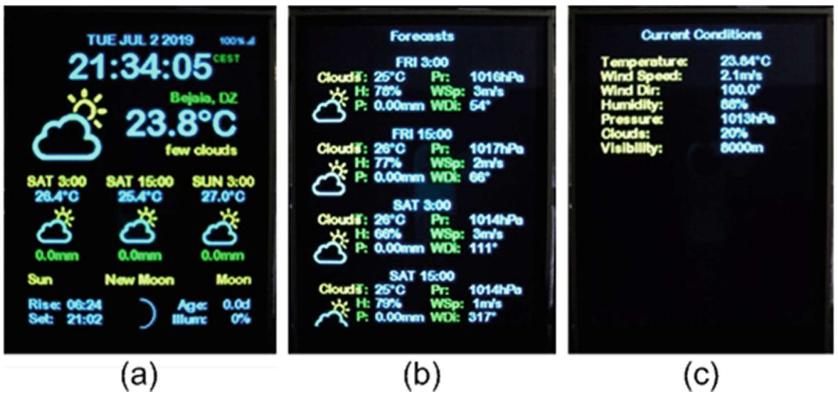

Fig. 9. Weather information screens: (a) main screen (b) forecasts (c) current conditions

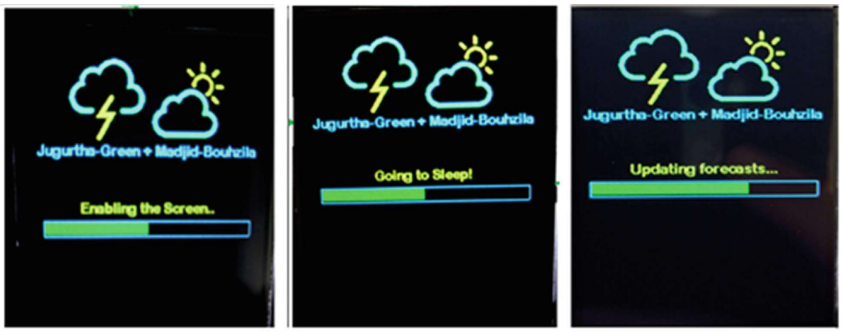

Fig. 10. Some Processing screens

6 Conclusion

The contribution describes the realization of an Internet of Things device with an ESP32 microcontroller with the dual-core implementation of data retrieval and processing beside wireless communication tasks. The main objective was to design an application for an ESP32 chip that will use most of its capabilities.

Hardware components used are the adafruit HUZZAH 32 board and the 2.4" TFT FeatherWing. The main part of this realization is software components development. We used accessible hardware and communication libraries for the development of different routines for data collection, processing and display as well as web server and security layer implementation.

The first leading idea was to use low-end hardware and software optimization to build an IoT embedded system, however the developed system is generic and could be reused and easily adapted to deploy other embedded systems because it includes main components that characterize an IoT embedded system which are network connectivity (Wi-Fi), HTTP based communication, data collection and remote system control (internal web server).

The presented work can be extended in two directions; First, it will be useful to include OTA (Over-The-Air) Programming feature which allows updating microcontroller code to the ESP32 using a browser, without the need to make a physical connection between the ESP32 and the computer. Second, the software design of this embedded system may be refined to maximize the use of ESP32 capabilities to create architectural design patterns and methods that are independent and reusable to implement any IoT embedded systems. This general framework will be characterized by: Device configurability, more functionalities for remote access (Control, reporting, monitoring … etc.) as well as self-test and diagnostic capability.

References

1. Elazhary, H.: Internet of Things (IoT), mobile cloud, cloudlet, mobile IoT, IoT cloud, fog, mobile edge, and edge emerging computing paradigms: Disambiguation and research directions. J. Netw. Comput. Appl. **128**, 105–14 (2019)
2. Nord, J.H., Koohang, A., Paliszkiewicz, J.: The Internet of Things: review and theoretical framework. Exp. Syst. Appl. **133**, 97–108 (2019). ISSN 0957–4174

3. Ojo, M.O., Giordano, S., Procissi, G., Seitanidis, I.N.: A review of low-end, middle-end, and high-end Iot devices. IEEE Access **6**, 70528–70554 (2018)
4. Ibrahim, D.: The Complete ESP32 Projects Guide, 59 Experiments with Arduino IDE and Python. Elektor International Media (2019). ISBN 978-1-907920-75-2
5. Babiuch, M., Foltýnek, P., Smutný, P.: Using the ESP32 microcontroller for data processing. In: 20th International Carpathian Control Conference (ICCC), Krakow-Wieliczka, Poland, pp. 1–6 (2019)
6. Maier, A., Sharp, A., Vagapov, Y.: Comparative analysis and practical implementation of the ESP32 microcontroller module for the Internet of Things. In: Proceedings of Internet Technology Applications (ITA), pp. 143–148 (2017)
7. Espressif Systems. ESP32 Technical Reference Manual, 2018 Version 4
8. Morón, C., Diaz, J.P., Ferrández, D., Saiz, P.: Design, development and implementation of a weather station prototype for renewable energy systems. Energies **11**, 2234 (2018)
9. Adafruit Learning System. ESP8266 WiFi Weather Station with Color TFT Display (2018)
10. Adafruit Learning System. Adafruit HUZZAH32 - ESP32 Feather (2019)
11. Adafruit Learning System. Adafruit 2.4 TFT FeatherWing (2019)
12. Espressif Systems. FreeRTOS ESP-IDF Programming Guide Version 4.1