

Using the ESP32 Microcontroller for Data Processing

Marek Babiuch

*Department of Control Systems and
Instrumentation*

*VSB - Technical University of Ostrava
Ostrava, Czech Republic
marek.babiuch@vsb.cz*

Petr Foltýnek

*Department of Control Systems and
Instrumentation*

*VSB - Technical University of Ostrava
Ostrava, Czech Republic
petr.foltýnek@vsb.cz*

Pavel Smutný

*Department of Control Systems and
Instrumentation*

*VSB - Technical University of Ostrava
Ostrava, Czech Republic
pavel.smutny@vsb.cz*

Abstract — This article deals with experiences with the development of applications of the ESP32 microcontrollers and provides a comprehensive review of the possibilities of applications development on this platform in the area of data measurement and processing. Microcontrollers usually connect with IoT modules and other smart sensors and provide data to the superior system. This paper also describes implementation of application with the version of connected OLED display and with ESP32 Wrover development board with integrated display.

Keywords—ESP32, IoT, microcontroller, measurement, display, development boards, environment

I. INTRODUCTION

At the present time, IoT, smart home automation and embedded systems are rapidly developing. This is closely related to the development of available hardware modules and processors. Development boards are produced on which a communication interface and peripherals are implemented together with the main processor chip. Nowadays, the popularity of the ESP32 chip grows and now both hardware variants of this chip and various branches of its software development are developing. A wide community of developers, but also academics deal with utilizing the higher generation ESP32 chip as the successor of ESP8266 microcontroller. The latest scientific articles prove the wide use of the ESP32 chip in various areas. The general possibilities of using a microcontroller with a recommendation for electronic projects are described in the article [1]. Paper provides a comparative analysis of the ESP32 with some other market competitors and introduces the microcontroller specification, features and programming details. An interesting solution to implement the ESP32 chip as a web server that works in real-time and can be effectively used for monitoring small solar energy systems is described in [2]. The use of a microcontroller is possible with a wide range of environmental monitoring sensors [3] whether it concerns air pollution or directly the implementation of monitoring LPG leakage [4]. In contrast, article [5] describes the use of the chip as a monitoring IoT system in the health sector. The use of the chip as a secure communication system with LoRa modules and implementation of cryptographic standards is described in the article [6], which deals with the Communication System for Remote Microgrids using AES Cryptography on ESP32. The use of ESP32 as a control system also supports wireless networks using the article [7]. The ESP32 chip can also be used to monitor alarms of technological equipment for operator workplaces [8], vibration monitoring [9] and nowadays practically in all areas where a monitoring embedded system is needed.

These articles summarize the main advantages of the ESP32 chip, which includes: wide deployment capabilities, support for Wi-Fi standards and protocols, low-cost solution. However, these articles describe deploying a microcontroller as a system that provides data as a web server and communicates with the environment and the superior system. In this article we will also attempt to describe the possibility of displaying measured data also on different types of displays directly on the microcontroller and displaying the status of the current data or system configuration for an immediate check of the system status.

II. ESP32 MICROCONTROLLERS

ESP32 is powerful SoC (System on Chip) microcontroller with integrated Wi-Fi 802.11 b/g/n, dual mode Bluetooth version 4.2 and variety of peripherals. It is an advanced successor of the 8266 chip primarily in the implementation of two cores clocked in different version up to 240 MHz. Compared to its predecessor, except these features, it also extends the number of GPIO pins from 17 to 36, the number of PWM channels per 16 and is equipped with 4MB of flash memory.

The ESP32 chip has been developed by the Espressif Systems company, which currently offers several ESP32 versions of the SoC in the form of ESP32 Developer Kit, the ESP32 Wrover Kit, which also includes an SD card and 3.2" LCD display and last but not least the ESP32 Azure IoT kit with USB Bridge and other built-in sensors. In addition to Espressif Systems, other producers are devoted to these chips - SparkFun with ESP32 Thing DB, WeMoS with its TTGO, D1, Lolin32 and Lolin D32, Adafruit (with Huzzah32), DF Robot (ESP32 FireBeeatle) and many other manufacturers sometimes offer good and sometimes bad clones.

ESP32 includes two core (Xtensa LX6 processor made with 40 nm technology). CPU cores can be individually controlled. There is 520 KB of on-chip SRAM for data and instructions available. Some SoC modules such as ESP32-Wrover features 4 MB of external SPI flash and an additional 8 MB of SPI PSRAM (Pseudo static RAM). We have the possibility to use SPI, I2S, I2C, CAN, UART, Ethernet MAC, and IR in various quantities, depending on the type of the board. Standard equipment also includes Hall Effect sensor, temperature sensor and touch sensor, other built-in sensors are implemented in Azure IoT and Developer kit. SoC also provides Cryptographic hardware acceleration: AES, SHA-2, RSA, and Elliptic Curve Cryptography (ECC) and random

number generator (RNG). Figure 1 shows the ESP32 microcontroller variants with which we are working at the Department of Control System and Instrumentation.

ESP32 boards are produced in prototype designs that can be used in smart home applications, automation, wearables, audio applications, cloud-based IoT applications, and more. It is possible to choose a specific development kit or to design a custom embedded system built on the ESP32 microcontroller.

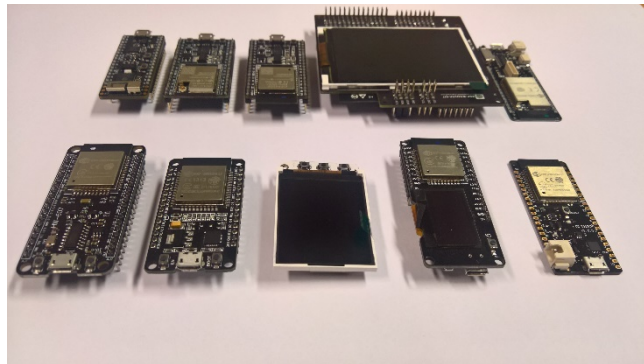


Fig. 1. Various ESP32 boards which we tested

III. HOW TO DEVELOP APPLICATIONS

Development on the ESP32 platform gives us many ways to use. This section describes some basic setup and configuration for the ESP32 platform environment. The ESP32 device can be developed from any Windows, Linux or MacOS operating system. We will deal with this article primarily about setting up and configuring the development environment on the Windows platform. We have the possibility to work with the ESP32 add-on environment for Arduino, the Espressif IoT Development Framework native platform and the Python environment using the Micropython engine.

A. Arduino Core for ESP32

The easiest way to start writing code for the ESP32 platform is to use the Arduino platform. This is an open-source platform designed for fast prototyping based on Atmel microcontrollers. Software development for microcontrollers is developed in the Arduino integrated development environment, which is written in Java. This is a software developed from the learning environment, which has been slightly modified and enhanced by certain features such as Wiring Language. Wiring is a programming language designed to program a microcontroller without specific hardware knowledge and has a C++ framework similarity. Wiring requires a microcontroller with a boot program. Nowadays Arduino IDE does not only support Atmel microcontrollers, but other types of the microcontroller can be installed. Espressif therefore decided to use this option to create a plugin called the Arduino core for the ESP32 Wi-Fi chip that installs support for the development of ESP32 microcontrollers in the Arduino IDE environment. The Arduino core extension for ESP32 Wi-Fi chip is released as an open-source project whose source codes are located at [10].

Espressif Company released the first stable version of the Arduino core for ESP32 Wi-Fi chip in mid-2018S and has standardized how to install support for Arduino IDE. FreeRTOS system support is provided to the Arduino core for ESP32 Wi-Fi chip to enable task-based processing because the ESP32 Chip is natively dual-core. Together with it provides support for optimizing the processing of specific tasks on a particular core of the ESP32 chip. The start of the standard program is, as with the Arduino platform, assigned to the setup function, which initializes the startup values of the microcontroller and the loop function that is responsible for running the program. When developing ESP32 microcontrollers, the loop function is always run on the first processor core. However, we need to realize that the ESP32 core numbering starts from the zero index.

The benefits of using this environment are its simple configuration and good community support. Using this environment, you can very quickly start prototyping the ESP32 microcontroller. This environment is designed for the development of simpler projects, with complex projects being a loss of clarity and sustainability of the project. At the same time, this platform is ideal for achieving experience with the development of the ESP32 microcontroller.

B. Espressif IoT Development Framework

If we decide to develop more sophisticated and optimized embedded systems built on the ESP32 microcontroller, it is advisable to choose the Espressif IoT Development Framework to enable us to develop applications in a native way. The Espressif IoT Development Framework is an open source project which is available at GitHub source page [11]. This framework includes basic configuration options for ESP, source code compilation, and firmware downloader for ESP32. The C language is used to develop applications. This framework is multiplatform but is natively developed under the Linux operating system where it can be configured most easily. To use a framework under the Windows operating system, you need to install UNIX-like / POSIX emulation support, or use the Windows Subsystem for Linux (WSL) option as a part of Windows 10. Espressif in its

installation wizard brings support for UNIX-like / POSIX environments using the MSYS2 engine, which is distributed by a zip file. This MSYS2 file contains all configurations for the development environment for ESP32 microcontrollers. The Espressif IoT Development Framework-based project management system is called *Make*. At the present time, the manager of the *Make* project migrates to support *CMake*, which has more tools for development environments. The current version of the framework is 3.2 and is constantly using the MAKE project report, but it is planned that version 4 will be a temporary version, where the project management to *CMake* is fully migrated. Like the Arduino core for ESP32 Wi-Fi microcontroller extension to the Arduino platform, the IoT Development Framework also implements the FreeRTOS system support. Programming with the Espressif IoT Development Framework require some procedures to be followed, otherwise the resulting project will not be compilable. Each project must contain a *sdkconfig* file that stores information about the framework configuration for the project. The project configuration interface is invoked by executing the *make menuconfig* command. For example, this interface configures the port on which the device is located, the frequency at which the individual bus works, or whether the FreeRTOS program is used in the configuration of one or two ESP32 cores (Figure no. 2.) These data are important because they generate dependencies on the build process based on the configured project.

The next step is to build the program itself, which is called by the *make build* command. If the build is all right, it is possible to upload the compiled program to the ESP32 microcontroller. This is provided by the built-in firmware flashing utility, which is called using the *make flash* command. To monitor the application, you can use a built-in monitor that runs with the *make monitor* command.

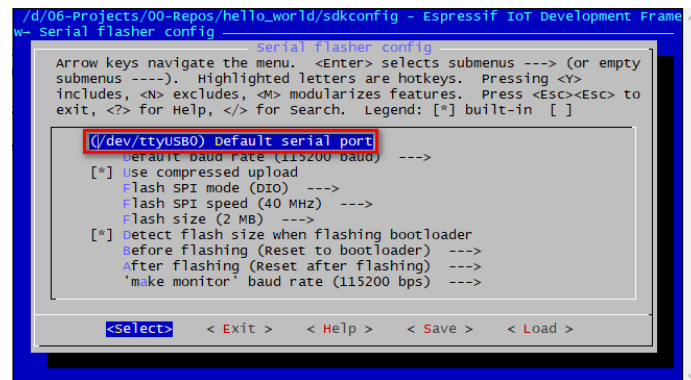


Fig. 2. The configuration of ESP project in IDF

The great advantage of using this framework is that you can create applications at the lowest level. Therefore, this framework is suited to the development of large and complex projects. The disadvantage of using a framework is the relatively high initial experience of the developer that the embedded system will develop. The community around this framework is also not as wide as the Arduino platform.

C. MicroPython

Some manufacturers that produce ESP32 chip-based development boards now use MicroPython environments as default firmware. The most common are ESP32-Wrover chips, which are usually available with PSRAM. MicroPython is a lightweight implementation of the Python 3 programming language that includes a subset of Python's standard libraries and is optimized for running on microcontrollers. The advantage of Python as a programming language is that it can be learned quite quickly. In case the ESP32 microcontroller does not contain MicroPython firmware, it is possible to upload this firmware to the given microcontroller. This is done using the *esptools* utility, which is written in Python and distributed via a pip package system that is native in Python environment. Subsequently, you need to download the MicroPython Firmware, which is available on [12].

Uploading MicroPython Firmware is done using the following command: *esptool.py --chip esp32 -p <USB-to-Serial Port> write_flash -z 0x1000 <path to .bin>*

where a COM port through which the ESP32 microcontroller communicates with the development computer is added to the *<USB-to-Serial Port>*, and a file name and a MicroPython firmware path location are added to the *<path to .bin>*. Now, after preparing the MicroPython environment on the ESP32 microcontroller, we will learn some basic programming rules. Each solution should contain at least two files, *boot.py* and *main.py*. It corresponds to the concept used in the Arduino platform. The *boot.py* file defines the initialization of the variables to be set at the start of the microcontroller, and *main.py* contains the microcontroller program that is executed without delay after initializing the variables in the *boot.py* file. If we need to use other library functions that are not implemented in the fundamental MicroPython firmware, these library functions must also be flashed to the device. Typically, these are libraries that provide communication and behavior of a particular hardware element, such as displays, sensors and IoT modules, etc. Uploading the program and library functions to the microcontroller can be done using the python utility *ampy*, which provides communication with the microcontroller over the serial line. Despite this utility, any manipulation of the files on the microcontroller is performed. The command to upload the *main.py* file to the microcontroller looks like:

```
ampy -p <USB-to-Serial Port> put main.py
```

A terminal connection can be used to debug Python on a microcontroller, for example, using the *PuTTY* program. You can use any available text editor to write Python programs. We use Visual Studio Code for these purposes.

This development approach is advantageous for prototyping and developing new algorithmic solutions for embedded systems based on MicroPython firmware. The development difficulty is somewhat higher than on the Arduino platform because there is no simple interface, but it is secured by a set of command line utilities.

D. Other Development Possibilities

There are still other ways to develop an application for ESP32 microcontrollers, for example by using the Lua programming language with the appropriate firmware, or by using the JavaScript programming language. Another option is to use some of the commercial projects such as Zerynth [13] that support ESP32 microcontrollers too. However, we have not tested these platforms yet. Our goal in this section was to describe the most commonly used open-source approaches for embedded system development on the ESP32 microcontroller on Windows platforms and to compare the experience with their use.

IV. ESP32 APPLICATION

The experience gained from the use of individual development environments on the Windows platform has been used in the practical implementation of various applications in the field of acquisition and processing of measured data from IoT sensors on the ESP32 microcontroller. The block diagram of the embedded system is shown in Figure 3.

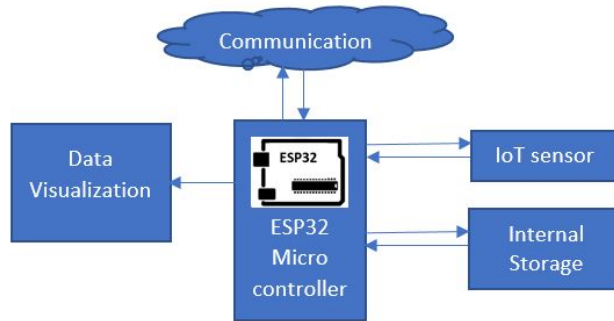


Fig. 3. Block scheme of the embedded system

In this article, we focused on the visualization of measured data on a display that is part of our embedded system. The communication buses are different depending on the type and size of the display and the size of the display depends on what kind of information we will display. For lower resolution displays, we decided to show only the status information and we sent the measured data for the evaluation to the superior system. The information we show on the given display is, for example, whether the microcontroller has a network connection, which current job is processing, what the actual measurement cycle is, etc. On displays that have a higher size and resolution, we also show the graphical flow of the measured values in addition to the status information. In case we simultaneously measured multiple variables at the same time, only the chart for one quantity was displayed, with the possibility of switching between graphs of the measured quantities. We have used multiple types of displays for our embedded system.

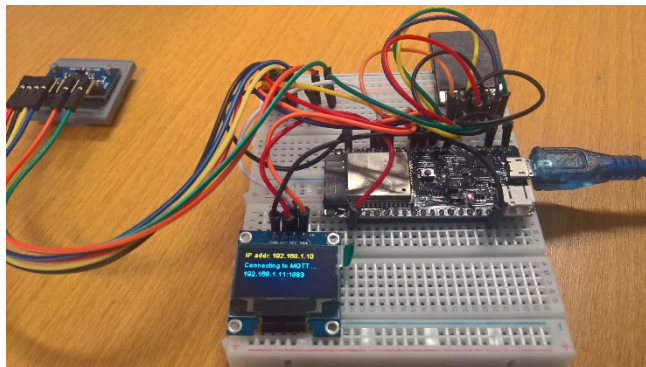


Fig. 4. ESP32 microcontroller, sensors, display and SD card storage

A. Application with OLED Display 0,91"

This display has a resolution of 128x32 pixels and its communication interface is I2C. This small display resolution enable showing only the status information of the application. For this OLED display, the SSD1306 library is used. This library can be

downloaded for Arduino Platform from [14], for ESP IDF framework from [15] and library for MicroPython from [16]. These OLED displays work so that only activated points are lights. Using this library in the source code is not complicated. Communication initialization (I2C or SPI) is performed to create a display object over which individual methods are invoked to design the graphical appearance of the display. Each platform has different call methods for uploading information to the display buffer. Figure 5 shows the example of the status information about the wireless network connection (shown in cutting part) and the status information with the number of measured data and the output file.

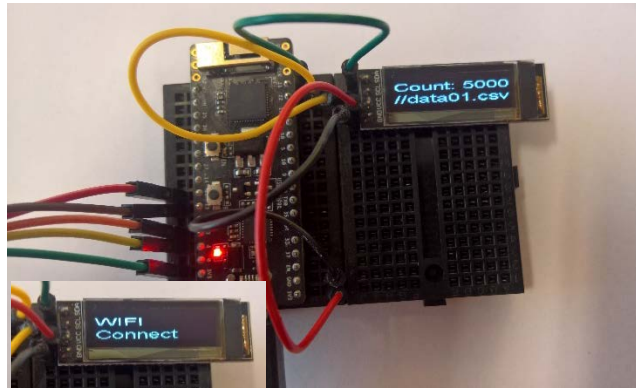


Fig. 5. Status information of microcontroller on 0,91" display

The use of SSD1306 may be different depending on which program environment (Arduino, ESP IDF, Micropython) we are programming.

B. Application with OLED Display 0,96"

This display has a resolution of 128x64 pixels. It is produced in the I2C or SPI version. We have used the I2C communication interface in our application, because we wanted to reduce the HW complexity of our solution. The display is programmed the same as the 0.91-inch OLED display variant, the only difference being in the display number of pixels and rows. Although this display is 2 times larger than the previous one, it is inappropriate to draw the measurement chart. Therefore we display more detailed status information on this display with this resolution than on the previous one. The figure 4 shows using of 0.96-inch display. ESP32 microcontrollers are produced in different versions and one of the variants is an integrated 0.96 inch display directly on the board. If this is sufficient for our application, we can use this development board. Figure 6 shows the integrated display on the ESP32 board with status information. We displays wireless connection again, this time with IP address, number of measured samples and output file.

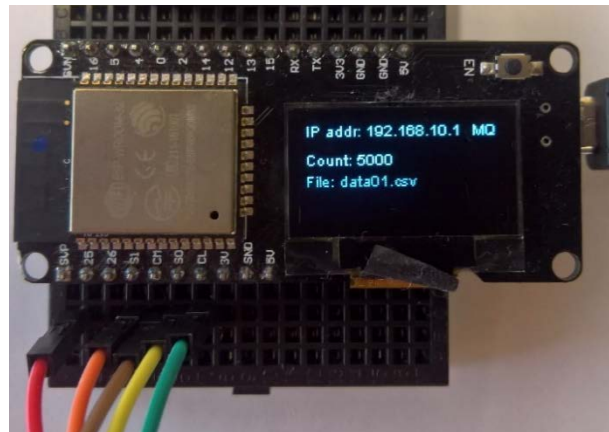


Fig. 6. Using of ESP32 board with integrated OLED display

C. Application with Integrated Display 2,4"

Espressif Company with introducing the new ESP32 Wrover chip has also launched a new development kit that uses this microcontroller.

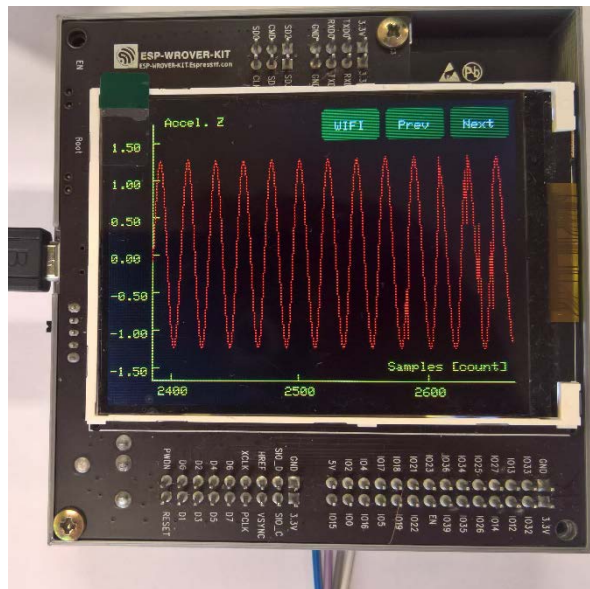


Fig. 7. Measured data on ESP32 Wrover integrated display

The big advantage is that a 320x240 color LCD display is already natively installed on this kit, which communicates via the 4-wire Serial Peripheral Interface (SPI). Espessif's GitHub [17] contains a free library for working with the LCD display on the Wrover development kit. From the point of view of programming, access is similar to previous displays. Because it is a larger resolution of the display, it is possible to display the graphical waveform of the measured values on this display. Due to the fact that the library provides only the Application Programming Interface (API), it is necessary to program the plotted charts. This part is relatively difficult for debugging and testing compared to displaying only the status information on the display. Therefore, it is very important to consider whether we want to use the feature on our embedded system. If we decide that charts are necessary, it's a good idea to create a graph component that makes it easier for us to work with charts. Figure 7 shows measured data directly on the integrated display. Because we measured more quantities at the same time, we have programmed a change in the display of quantities over a certain time interval. Figure 8 shows other measured variables. The display of accelerometer sensor values and temperature measurement values changes cyclically.

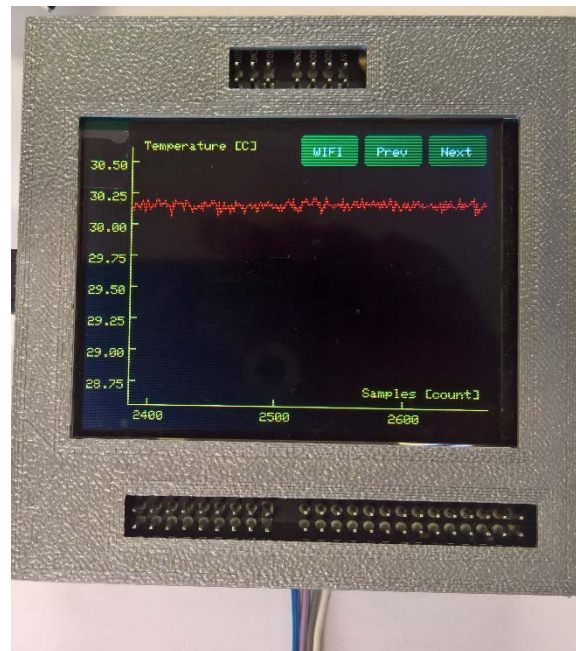


Fig. 8. Boxed embedded ESP32 system with the integrated display

V. CONCLUSION

We summarized in this article the most commonly used application development platforms for the ESP32 microcontroller. For each platform, we have described the benefits, recommendations for which type of application the platform is suitable for, and at the same time we have compared the level of prerequisites for that platform in terms of software skills and experience of the developer. We recommend to use the Arduino ESP 32 Core extension platform for beginning developers on the ESP32 platform and also for quickly verification of application design, but for the product development and better use of hardware properties (programming of two RTOS cores), we recommend the ESP IDF platform, in which development is provided through C language with a native firmware downloader.

We can encounter quite often MicroPython on new boards with ESP32 Wrover chip, which features advantages of Python language properties and allows application development both procedurally and with the use of object-oriented design.

We often encounter the problems of visualization of measured data and system diagnostics during developing an embedded system. Some development boards with ESP32 microcontrollers are already produced with an integrated display of various sizes. In our article, we describe an experience with an application design that uses different types of displays. We can recommend from our experiences with application development that the status information is better to display on small displays, and larger displays can also display real-time data. The more graphic information we want to show on the display, the more difficult the application is and the more demanding for the developer. So, if we decide to display graphics in real-time, it's a good idea to create a graph component that has the responsibility for chart plotting and is independent of the hardware implementation of the display work.

ACKNOWLEDGMENT

This work was supported by the European Regional Development Fund in the Research Centre of Advanced Mechatronic Systems project, CZ.02.1.01/0.0/0.0/16_019_/0000867 within the Operational Programme Research, Development and Education and the project SP2019/51 Applied Research in the Area of Machine and Process Control supported by the Ministry of Education, Youth and Sports.

REFERENCES

- [1] A. Maier, A. Sharp, Y. Vagapov, "Comparative analysis and practical implementation of the ESP32 microcontroller module for the internet of things" 2017 Internet Technologies and Applications, ITA 2017 - Proceedings of the 7th International Conference IEEE, pp 143-148, November 2017, DOI: 10.1109/ITECHA.2017.8101926.
- [2] I. Allafi, T. Iqbal, "Design and implementation of a low cost web server using ESP32 for real-time photovoltaic system monitoring" 2017 IEEE Electrical Power and Energy Conference, EPEC 2017, pp 1-5, February 2018, DOI: 10.1109/EPEC.2017.8286184.
- [3] B.S. Sarjerao, A. Prakasarao, "A Low Cost Smart Pollution Measurement System Using REST API and ESP32" 3rd International Conference for Convergence in Technology, I2CT 2018, November 2018, DOI: 10.1109/I2CT.2018.8529500.
- [4] A.H. Abdullah, S. Sudin, M.I.M. Ajit, F.S.A. Saad, K. Kamaruddin, F. Ghazali, Z.A. Ahmad, M.A.A. Bakar, "Development of ESP32-based Wi-Fi Electronic Nose System for Monitoring LPG Leakage at Gas Cylinder Refurbish Plant" 2018 International Conference on Computational Approach in Smart Systems Design and Applications, August 2018, DOI: 10.1109/ICASSDA.2018.8477594
- [5] D. Ghosh, A. Agrawal, N. Prakash, P. Goyal "Smart Saline Level Monitoring System Using ESP32 And MQTT-S" 20th International Conference on e-Health Networking, Applications and Services, Healthcom 2018, DOI: 10.1109/HealthCom.2018.8531172.
- [6] A. Iqbal, T. Iqbal, "Low-cost and Secure Communication System for Remote Micro-grids using AES Cryptography on ESP32 with LoRa Module" 2018 IEEE Electrical Power and Energy Conference (EPEC), October 2018, DOI: 10.1109/EPEC.2018.8598380.
- [7] S. Bipasha Biswas, M. Tariq Iqbal "Solar Water Pumping System Control Using a Low Cost ESP32 Microcontroller" Canadian Conference on Electrical and Computer Engineering, CCECE 2018, May 2018, DOI: 10.1109/CCECE.2018.8447749.
- [8] P. Urban, L. Landryova, "Collaborative Operations Using Process Alarm Monitoring" IFIP WG 5.7 International Conference on Advances in Production Management Systems, APMS 2017, September 2017, DOI: 10.1007/978-3-319-66923-6_52.
- [9] G. Takacs, J. Vachálek, B. Rohal'-Ilkiv, "Online Structural Health Monitoring and Parameter Estimation for Vibrating Active Cantilever Beams Using Low-Priced Microcontrollers", *Shock and Vibration*, Volume 2015, DOI: 10.1155/2015/506430.
- [10] GitHub "Arduino core for ESP32 WiFi chip" [online], 2019, available at: <https://github.com/espressif/arduino-esp32>
- [11] GitHub "Espressif IoT Development Framework" [online], 2019, available at: <https://github.com/espressif/esp-idf/>
- [12] MicroPython "Firmware for ESP32 boards" [online], 2019, available at: <https://micropython.org/download/#esp32>
- [13] Zerynth "The Middleware for IoT" [online], 2019, available at: <https://www.zerynth.com/>
- [14] GitHub "OLED SSD1306" [online], 2019, available at: <https://github.com/ThingPulse/esp8266-oled-ssd1306>
- [15] GitHub "Sample code for driving 128x64 OLED display with SSD1306 driver via ESP-IDF's I2C master driver" [online], 2019, available at: <https://github.com/yanbe/ssd1306-esp-idf-i2c>
- [16] GitHub "Adafruit CircuitPython driver for SSD1306 or SSD1305 OLED displays" [online], 2019, available at: https://github.com/adafruit/Adafruit_CircuitPython_SSD1306
- [17] GitHub "Library for the Adafruit ILI9341 display products" [online], 2019, available at: https://github.com/espressif/WROVER_KIT_LCD/

This is preprint version of this article:

BABIUCH, M., FOLTÝNEK, P., SMUTNÝ, P. Using the ESP32 microcontroller for data processing. In *Proceedings of 20th International Carpathian Control Conference ICCC'2019*. Krakow - Wieliczka; Poland; May 26-29, 2019, pp. 88-93. ISBN: 978-172810701-1, DOI: 10.1109/CarpathianCC.2019.8765944.