No. 127/23, 27–41
ISSN 2657-6988 (online)
ISSN 2657-5841 (printed)
DOI: 10.26408/127.03

Submitted: 13.04.2023
Accepted: 18.04.2023
Published: 30.09.2023

AUTHENTICATED WIFI-BASED WIRELESS DATA TRANSMISSION FROM MULTIPLE SENSORS THROUGH A LABORATORY STAND BASED ON COLLABORATION BETWEEN ATMEGA2560 AND ESP32 MICROCONTROLLERS

Mostafa Abotaleb
PhD Student at The Doctoral School of Gdynia Maritime University, 81-87 Morska St., 81–225 Gdynia, Poland, ORCID 0000-0002-2670-5928, e-mail: m.abotaleb@sd.umg.edu.pl

Abstract: The use of wireless technology in the field of instrumentation is rapidly expanding due to many reasons such as lower installation and commissioning cost than those incurred by wired technologies and a higher level of security provided by authenticated encrypted data transaction, in addition to a decreased level of system complicity, manifested in eliminating or limiting the need for cabling or its accessories. This article will discuss a proposed technique to perform authenticated wireless WiFi data transmission from multiple sensors to the control station depending on a laboratory stand for the purpose of analysing the principle of coexistence between wireless technologies dedicated to industrial automation, such as wireless HART, and general purpose technologies, such as WiFi and Bluetooth Low Energy BLE.

Keywords: Data Authentication, WiFi, WebSerial, ESP32, Arduino Mega 2560, Python, Websockets.

1. INTRODUCTION

Wireless HART and ISA100.11a are the only technologies recognised as wireless standards dedicated to industrial automation. Other general purpose wireless technologies, such as WiFi and Bluetooth Low Energy BLE, can also be used in industrial automation. However, the level of security is lower than that rendered by technologies such as wireless HART and ISA100.11a. The principle of coexistence [LaSorte, Seidman and Quang 2016; Al Kala, Seidman and Quang 2018; Gomaa 2020;] between both types of technologies is a key element which plays an important role in enhancing the reliability level of the wireless network in the industrial plant. This enhancement is manifested by two concepts. The first concept is that the coexistence principle achieves an easier process of integration between wired classical analogue standards such as 4–20 mA and technologies such as wireless HART.

The second concept is that the coexistence principle renders a certain degree of verification for the measured data if a specific variable was measured by a wireless HART transmitter and also by a classical 4–20 mA transmitter, the reading from which is sent to the host using general purpose technologies such as WiFi or BLE. Thus, the host will compare the readings obtained from both transmitters to verify if they are identical or not. This verification process can upgrade the capability of the wireless network so that it could perform automation tasks dedicated to control and safety purposes and not only monitoring or alarming purposes.

In this article, a similar approach will be discussed through a laboratory stand based on using both ATmega2560 [Arduino 2023] and ESP32 [Espressif Systems 2023] controllers. The ATmega2560 controller was chosen for two reasons. The first reason is its ability to receive up to 16 analogue inputs, which will be helpful in the assessment process for the capability of performing authenticated data transmission from higher number of sensors. The second reason is its capability to receive analogue inputs with a voltage level from 0 to 5 VDC, which makes it an appropriate choice when processing signals from 4–20 mA analogue transmitters, as a 4–20 mA signal can easily be converted to a 1–5 VDC signal through a 250 ohms shunt resistor. The ESP32 controller was chosen particularly due to the diversity it renders when handling data wirelessly as it adopts both WiFi (in both ISM bands of 2.4 GHz and 5 GHz) and BLE technologies using multiple protocols such as ESP-Now [Randomnerdtutorials January 2020b], ESP webserial, ESP mesh [Randomnerd-tutorials November 2020] and ESP client-server [Randomnerdtutorials January 2020a].

The laboratory stand discussed in this article will be based on using WiFi technology in conjunction with the WebSerial websocket [Randomnerdtutorials August 2021; Techtutorialsx 2022]. Simulated measured data from 1 to 5 VDC will be applied to 8 analogue inputs of ATMEGA2560 controller through 8 potentiometers. The measured data detected by ATMEGA2560 will be sent to ESP32 controller using serial communication. ESP32 will send the measured data to the host through using WiFi technology and the host controller will receive the sent data through using the WebSerial websocket. The ATMEGA2560 controller will have a single PWM analogue output which will be identical to one of the measured analogue inputs according to a specific authenticating timing pattern toggling between the eight inputs to chose one of them to be passed as an output signal. This timing pattern will be based on timers toggling between the outputs of ATMEGA2560 controller and authenticated by a Python program accessing the WebSerial websocket [DelCastillo 2021] and sending confirmation messages to the ESP32 controller affirming that the analogue output of the ATMEGA2560 is correspondent to a specific analogue input at a specific time. This analogue PWM output will subsequently be converted to a 4–20 mA current signal, which will be integrated into a wireless HART network using a wireless HART adapter such as the BULLET adapter manufactured by Pepperl+Fuchs [Pepperl+Fuchs 2015], and then sent to the same host controller using wireless HART technology. At the host controller, a verification process will be used to check the degree of similarity between the two readings. The process of converting the PWM output from the ATMEGA2560 controller into 4–20 mA and its integration into wireless HART network will be discussed in a later article.

In this article, the discussion will be focused on the idea of the technique adopted to perform two types of communication tasks simultaneously. The first type is the serial communication Between ATMEGA2560 controller and ESP32 controller. The second type of communication is the WiFi based wireless communication between ESP32 controller and the host controller through the WebSerial websocket controlled by a Python program. A Graphical User Interface designed by Visual Basic at the host controller station will collect the data sent wirelessly from the 8 simulated sensors and display it to the user.

Ultimately, the article will highlight the recommended techniques proposed during programming both ATMEGA2560 and ESP32 controllers using arduino.ide when performing serial and WiFi based wireless communication tasks, so that problems which lead to the controllers crashing, such as overloading the serial buffer, or timing problems can be avoided. The recommended techniques are based on experimental trials conducted on the laboratory stand.

2. LABORATORY STAND

As described earlier and as illustrated in Figure 1, the laboratory stand consists of eight potentiometers, one ATMEGA2560 controller, one ESP32 controller and one LED. The eight potentiometers are connected to the analogue inputs of the ATMEGA2560 controller from A0 to A7.

Analogue output 4 is connected to the LED through a 230 Ω resistor. The serial communication [Arduino References 2019] between ATMEGA2560 and ESP32 controllers is performed through the second serial port at each of them. Tx2 of each controller is connected to Rx2. Similarly, Rx2 is connected to Tx2. Both serial wires are connected through 10 kΩ resistors in order to avoid damaging the ESP32 controller with the higher voltage level at the ATMEGA2560 controller. The resistors will reduce the 5 VDC voltage level from ATMEGA2560 to a 3 VDC voltage level at the ESP32 controller. For the same reason, ATMEGA2560 controller was chosen to receive the simulated 1–5 VDC signals as analogue inputs from sensors as the ESP32 controller can only tolerate analogue inputs of voltage up to 3.3 VDC. The lower range and span at the ESP32 controller may lead to less accurate readings with an increased margin of error when dealing with the 4–20 mA analogue standard as it is easier to be converted into 1-5 VDC (as in the case of most analogue input cards) than 1–3.3 VDC. Both the ATMEGA2560 and ESP32 controllers were programmed using arduino ide.

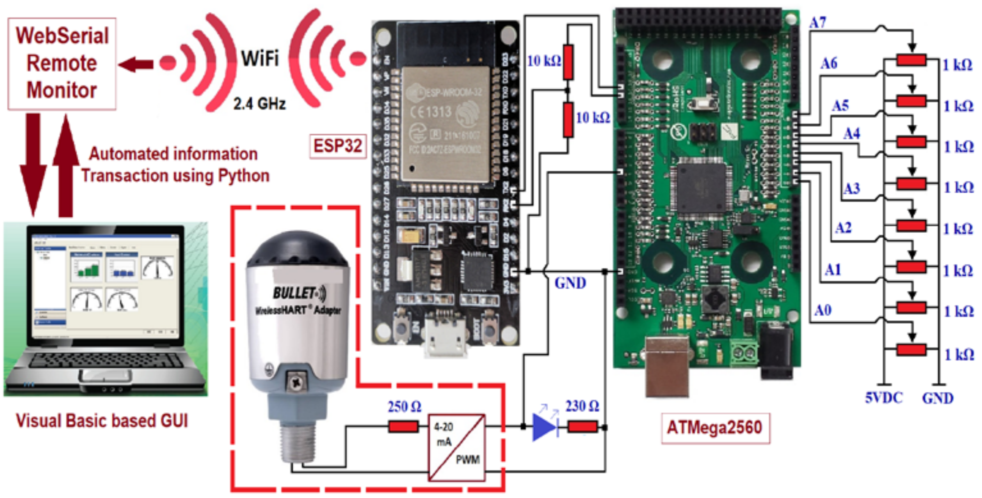


Fig. 1. The Laboratory Stand. The part surrounded by red dashed line will not be discussed in this article

In this article, the most important features of both codes uploaded to both controllers will be discussed. In the setup section of the program at ATMEGA2560 and as shown in figure 2, the first line of the code is used to start serial communication with the computer to which the ATMEGA2560 is connected through the serial port with a 9600 baud rate. Without the first line, it will not be possible to upload the code from the computer to the ATMEGA2560 controller. The second line of the code is dedicated to setting up the second serial port at the ATMEGA2560 controller so that it can be ready to communicate with the serial port of ESP32 controller.

```
Serial.begin(9600);        // Set up the serial port connection with PC
Serial2.begin(9600);       // Set up the serial port connection with ESP32
```

Fig. 2. Setup of The ATMEGA2560 Controller

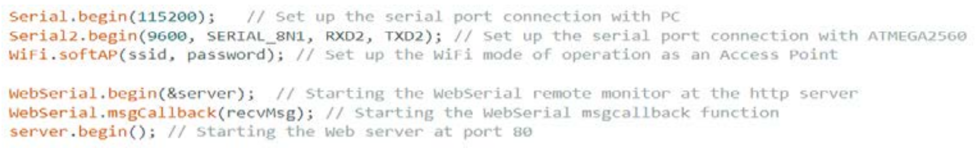
Similarly and as illustrated in Figure 3, the first line of the set up section in ESP32 is dedicated to communication between the computer serial port and the ESP32 controller with a baud rate of 115200, while the second line is dedicated to setting up the second serial port to communicate with ATMEGA2560 controller with a baud rate of 9600. Additionally, arduino.h library should be included so that serial communication can take place between both the ESP32 and ATMEGA2560 controllers. Here, it can be observed that the pins of Tx2 and Rx2 at ESP32 should be defined, which is not the case at ATMEGA2560 controller where Tx2 and Rx2 pins are dedicated only to performing serial communication.

However, in ESP32 the Tx2 and Rx2 pins can also be used as GPIOs (General Purpose Input Output). WebSerial [Sharma 2021] is a Serial monitor for ESP8266 and ESP32 microcontrollers that can be accessed remotely via a web browser. The webpage is stored in the program memory of the microcontroller.

Firstly, in order to set up the ESP32 controller to perform WiFi communication with the webserial websocket, the following libraries should be included in the code: *WiFi.h*, *ESPAsyncWebServer.h*, *WebSerial.h* and *AsyncTCP.h* [Randomnerd-tutorials August 2021; Techtutorialsx 2022]. After declaring these libraries at the beginning of the code, three important lines of code should be added to the setup function as illustrated in Figure 3. The first line is used to define the WiFi mode of operation.

In this article, the ESP32 will operate as a soft access point [Techtutorialsx 2022] using the *softAP()* method.

In this mode, the ESP32 controller will create its own WiFi network with an independent SSID and password defined by the user. Based on the *ESPAsyncWebServer.h* library included in the system, the *(http)* web server will be set up on port 80, and then WebSerial will be initialised on the web server through using the *begin()* method as illustrated in the fourth line of Figure 3. The fifth line of Figure 3 is used to start the http web server on port 80, while the sixth line is used to define the callback function using the *msgCallback()* method at the WebSerial object. The call back function is used to receive messages sent from the WebSerial websocket to the ESP32 controller.

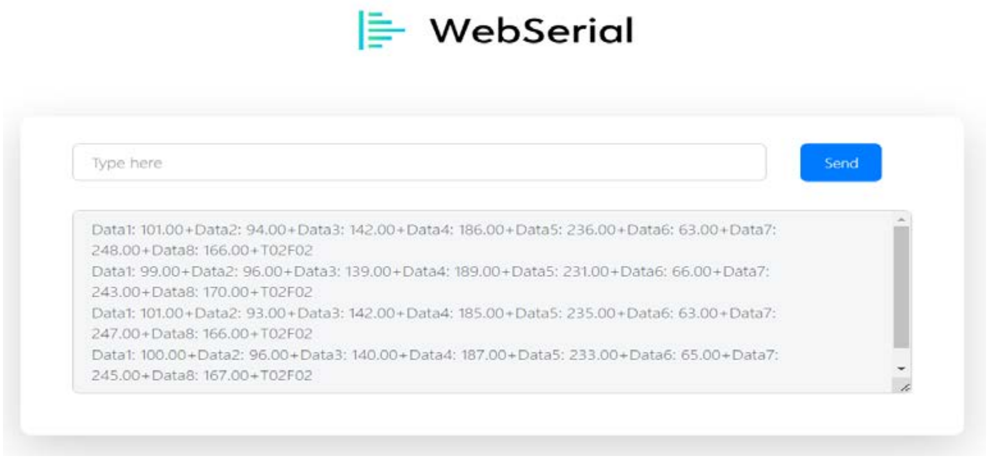


```
Serial.begin(115200);    // Set up the serial port connection with PC
Serial2.begin(9600, SERIAL_8N1, RXD2, TXD2); // Set up the serial port connection with ATMEGA2560
WiFi.softAP(ssid, password); // Set up the WiFi mode of operation as an Access Point

WebSerial.begin(&server);  // Starting the WebSerial remote monitor at the http server
WebSerial.msgCallback(recvMsg); // Starting the WebSerial msgcallback function
server.begin(); // Starting the Web server at port 80
```

Fig. 3. Setup of The ESP32 Controller

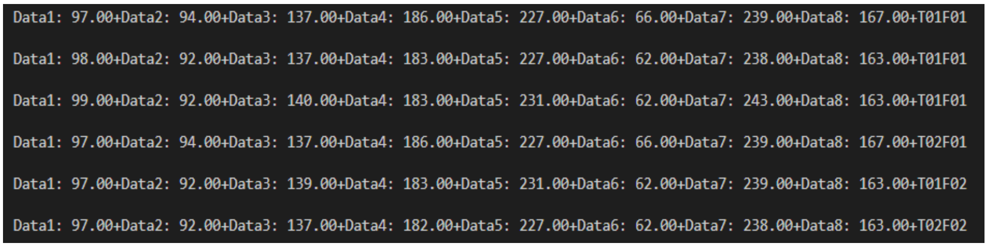
As illustrated in Figure 4, WebSerial is continuously receiving the messages sent from ESP32 controller. However, if it is meant to send messages from the host computer to the ESP32 controller, this should be done manually by writing the message in the space indicatedand then pressing the (Send) command button.



```
Data1: 101.00+Data2: 94.00+Data3: 142.00+Data4: 186.00+Data5: 236.00+Data6: 63.00+Data7:
248.00+Data8: 166.00+T02F02
Data1: 99.00+Data2: 96.00+Data3: 139.00+Data4: 189.00+Data5: 231.00+Data6: 66.00+Data7:
243.00+Data8: 170.00+T02F02
Data1: 101.00+Data2: 93.00+Data3: 142.00+Data4: 185.00+Data5: 235.00+Data6: 63.00+Data7:
247.00+Data8: 166.00+T02F02
Data1: 100.00+Data2: 96.00+Data3: 140.00+Data4: 187.00+Data5: 233.00+Data6: 65.00+Data7:
245.00+Data8: 167.00+T02F02
```

Fig. 4. The WebSerial Remote Monitor

Accordingly, and in order to implement an automated authentication process for the data sent from the ESP32 controller to the host computer, a Python program is used to undertake such a task. Simplicity, availability of certified references and documentation on the Internet and being user friendly are the most important advantages of Python as a programming language. By using a specific Python code, The WebSerial websocket will be accessed and controlled as indicated in Figure 5.



```
Data1: 97.00+Data2: 94.00+Data3: 137.00+Data4: 186.00+Data5: 227.00+Data6: 66.00+Data7: 239.00+Data8: 167.00+T01F01

Data1: 98.00+Data2: 92.00+Data3: 137.00+Data4: 183.00+Data5: 227.00+Data6: 62.00+Data7: 238.00+Data8: 163.00+T01F01

Data1: 99.00+Data2: 92.00+Data3: 140.00+Data4: 183.00+Data5: 231.00+Data6: 62.00+Data7: 243.00+Data8: 163.00+T01F01

Data1: 97.00+Data2: 94.00+Data3: 137.00+Data4: 186.00+Data5: 227.00+Data6: 66.00+Data7: 239.00+Data8: 167.00+T02F01

Data1: 97.00+Data2: 92.00+Data3: 139.00+Data4: 183.00+Data5: 231.00+Data6: 62.00+Data7: 239.00+Data8: 163.00+T01F02

Data1: 97.00+Data2: 92.00+Data3: 137.00+Data4: 182.00+Data5: 227.00+Data6: 62.00+Data7: 238.00+Data8: 163.00+T02F02
```

Fig. 5. The WebSerial Remote Monitor Automated using Python

An automated response will be activated, confirming the reception of a specific piece information. The code is based on three libraries: *websockets*, *asyncio* and *time*. The *websocket* and *asyncio* libraries are used to connect to the URL of the websocket and to send/receive information to/from the websocket [Solomon 2019; DelCastillo 2021]. *async with websockets.connect(url) as ws:* is the command used to connect to the websocket URL. *await ws.recv()* is the command used to receive data from the URL into a string variable, while *await ws.send ()* is the command used to send data from a string variable to the websocket. Without the *time* library, the Python code will be executed only once.

Therefore, a timer function is required to repeat the execution of the code and also to pause the program after information transaction (sending /receiving) so that the program does not crash.

3. CONTROLLERS' PROGRAMMING AND AUTHENTICATION

At first sight, it might seem very easy to send the measured data from multiple sensors connected to the ATMEGA2560 controller to the ESP32 controller through serial communication between each of them. The ESP32 controller will then send it to WebSerial through using WiFi. However, if authenticated transmission is required, this will impose an additional complication on the code developed, This is necessary for the WebSerial remote monitor to be allowed to send automated authenticating messages for the data sent. Here, authentication is required mostly to ensure that both sides (ATMEGA2560 and ESP32 on one side and the WebSerial websocket on the other side) will know that the current analogue output will correspond to which analogue input from which sensor. Transition of the analogue output values from sensor to another will be controlled by a timing pattern at the ATMEGA2560 controller. For the ESP32 controller, it will have to receive the authentication messages from WebSerial through WiFi and also receive the data measured from the ATMEGA2560 controller through serial communication. It will also start the communication process by sending a command to the ATMEGA2560 controller through the serial port in addition to sending the data measured to WebSerial. Accordingly, the ESP32 controller will have to perform simultaneous serial and wireless communication tasks. This is not recommended as the probability of overloading the serial buffer will be very high if the timing pattern is not considered carefully. If the serial buffer is overloaded, it will immediately lead to the ESP32 controller crashing, and it will have to be reset. In order to evade such a scenario, some common programming techniques should be avoided.

As illustrated in Figure 6, some programmers may depend on the *delay()* function in creating their timing pattern to toggle between the readings from multiple sensors so that one of them can be passed to the analogue output. The excessive use of the *delay()* function is not recommended [Santos 2015; Arduino Forum 2017; Cura 2020] due to the fact that the timers generated by such a function are not as accurate as they should be. Using the *delay()* function is only recommended to pause the program for no more than a few seconds after giving commands related to sending data serially or wirelessly, in order to give the other side a chance to receive the data comfortably without overloading of the serial buffer Therefore, it is not advisable to make the core of the developed code based on the *delay()* function. As also illustrated in Figure 6, the data measured from each sensor are sent separately in an independent message and authenticated by another independent message from WebSerial. For a large number of sensors, this will increase the complexity of the code as well as the latency of the system. Additionally, using such a technique will lead to an increased number of messages waiting at the serial buffer, which will eventually lead to the controller crashing (ATMEGA2560 or ESP32) due to an overloaded serial buffer.

As illustrated in Figure 7, some programmers might tend to use built-in timers in both ESP32 [Joseph 2022] or ATMEGA2560 [Liang 2013; Stoffregen 2015; Toptechboy 2018] controllers in conjunction with interrupt subroutines to perform tasks like reading to/from the serial port at specific time intervals, however attention should be paid that any timer subroutine will stop the execution of the main program at the loop function till this subroutine is executed, which might be a real problem particularly with the ESP32 controller, as stopping the main program will lead to disconnection of the WiFi network generated by the controller, which will consequently lead to the Python program crashing when accessing the WebSerial websocket at the host controller station.

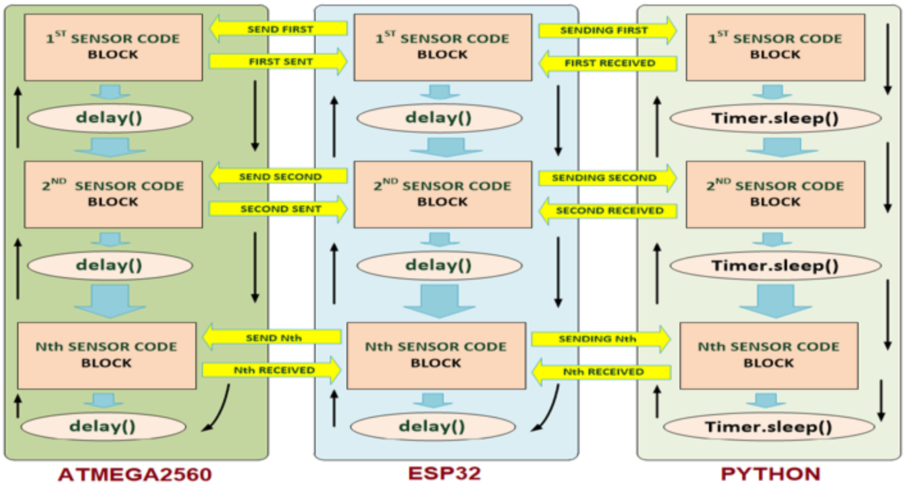
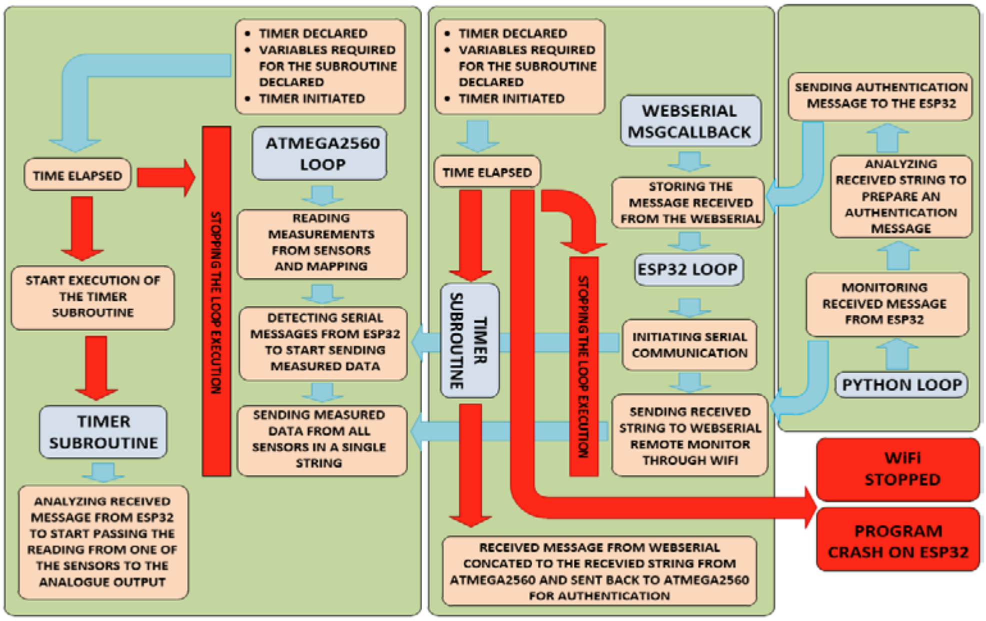


Fig. 6. Programming Technique based on using delay() function and multiple authentication messages (NOT RECOMMENDED)

In order to avoid the complications described above, the following technique was adopted (Fig. 8) in this laboratory stand to guarantee a stable and accurate performance. Regarding the timing pattern, which will toggle between the data measured from the simulated sensors to pass one of them as an analogue output, it will be based on using the *millis()* function [Cura 2020; Arduino References 2022] to generate more accurate time intervals without the need to pause the program when these timers are running. Regarding the format by which serial communication between the ATMEGA2560 and ESP32 controllers will take place, it will take the form of a single string variable containing all the measured data from all sensors

[Arduino References 2021]. This string variable will be sent wirelessly to the WebSerial websocket. The Python program at the host controller station will save the received data string into a text file. By using text manipulation in Python, the last part of the string will be detected. By manipulating the last part of the received string, the received information will be authenticated. The last part of the string received will include two letters (T) and (F). (T) refers to the information sent to the WebSerial remote monitor from the controllers' side, while (F) refers to the information sent from the WebSerial remote monitor to the controllers side.



TIMER DECLARED
VARIABLES REQUIRED FOR THE SUBROUTINE DECLARED
TIMER INITIATED
ATMEGA2560 LOOP
TIME ELAPSED
STOPPING THE LOOP EXECUTION
START EXECUTION OF THE TIMER SUBROUTINE
TIMER SUBROUTINE
ANALYZING RECEIVED MESSAGE FROM ESP32 TO START PASSING THE READING FROM ONE OF THE SENSORS TO THE ANALOGUE OUTPUT
READING MEASUREMENTS FROM SENSORS AND MAPPING
DETECTING SERIAL MESSAGES FROM ESP32 TO START SENDING MEASURED DATA
SENDING MEASURED DATA FROM ALL SENSORS IN A SINGLE STRING
TIMER DECLARED
VARIABLES REQUIRED FOR THE SUBROUTINE DECLARED
TIMER INITIATED
TIME ELAPSED
TIMER SUBROUTINE
STOPPING THE LOOP EXECUTION
WEBSERIAL MSGCALLBACK
STORING THE MESSAGE RECEIVED FROM THE WEBSERIAL
ESP32 LOOP
INITIATING SERIAL COMMUNICATION
SENDING RECEIVED STRING TO WEBSERIAL REMOTE MONITOR THROUGH WIFI
RECEIVED MESSAGE FROM WEBSERIAL CONCATED TO THE RECEVIED STRING FROM ATMEGA2560 AND SENT BACK TO ATMEGA2560 FOR AUTHENTICATION
SENDING AUTHENTICATION MESSAGE TO THE ESP32
ANALYZING RECEIVED STRING TO PREPARE AN AUTHENTICATION MESSAGE
MONITORING RECEIVED MESSAGE FROM ESP32
PYTHON LOOP
WIFI STOPPED
PROGRAM CRASH ON ESP32

Fig. 7. Programming Technique based on using buit-in timers and their interrupts subroutines (NOT RECOMMENDED)

Table 1 illustrated how authentication process is achieved through the significance of the digits detected following both of the letters T and F. As illustrated in Figure 9, the reading from each transmitter will be extracted from the stored information in the text file through using text manipulation in Visual Basic 6. Additionally, Visual Basic GUI will display from which simulated sensor the data will be passed to the analogue output after being authenticated. The displayed readings from the simulated sensors are mapped from the (0–255) range to the (1–5 VDC) range. In order to avoid the programs crashing at either the ATMEGA2560 controller or the ESP32 controller due to simultaneous serial and wireless communication, two important methods were used. The first method of the WiFi object is *disconnect()*, while the second method of the same object is *reconnect()* [Randomnerdtutorials February 2021]. The *disconnect()* method was used in case a serial information was detected at the serial buffer of the ESP32's second serial buffer (*Serial2.available* > *0*). The *reconnect()* method was used when the ESP32 controller was about to send data to the WebSerial websocket.

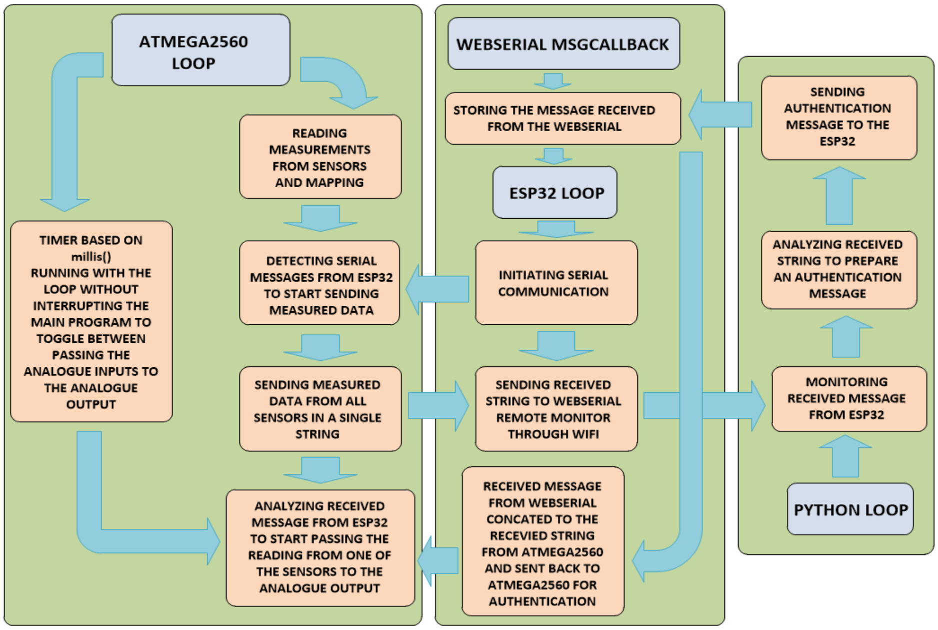
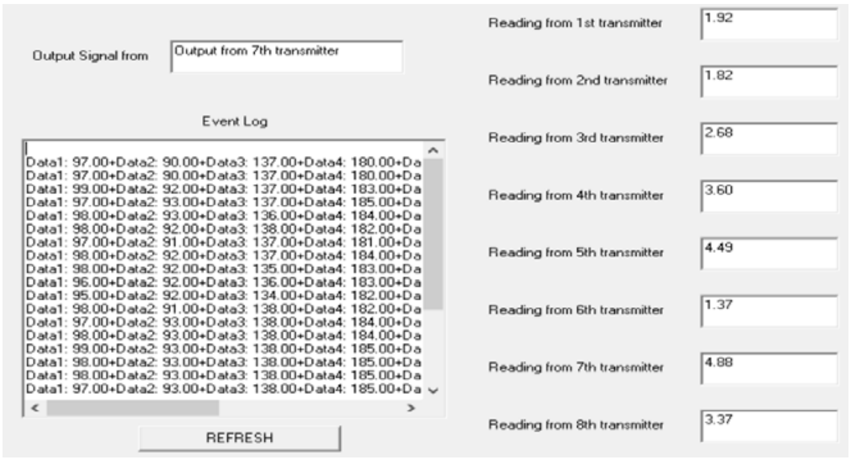

ATMEGA2560 LOOP
READING MEASUREMENTS FROM SENSORS AND MAPPING
TIMER BASED ON millis() RUNNING WITH THE LOOP WITHOUT INTERRUPTING THE MAIN PROGRAM TO TOGGLE BETWEEN PASSING THE ANALOGUE INPUTS TO THE ANALOGUE OUTPUT
DETECTING SERIAL MESSAGES FROM ESP32 TO START SENDING MEASURED DATA
SENDING MEASURED DATA FROM ALL SENSORS IN A SINGLE STRING
ANALYZING RECEIVED MESSAGE FROM ESP32 TO START PASSING THE READING FROM ONE OF THE SENSORS TO THE ANALOGUE OUTPUT
WEBSERIAL MSGCALLBACK
STORING THE MESSAGE RECEIVED FROM THE WEBSERIAL
ESP32 LOOP
INITIATING SERIAL COMMUNICATION
SENDING RECEIVED STRING TO WEBSERIAL REMOTE MONITOR THROUGH WIFI
RECEIVED MESSAGE FROM WEBSERIAL CONCATED TO THE RECEIVED STRING FROM ATMEGA2560 AND SENT BACK TO ATMEGA2560 FOR AUTHENTICATION
SENDING AUTHENTICATION MESSAGE TO THE ESP32
ANALYZING RECEIVED STRING TO PREPARE AN AUTHENTICATION MESSAGE
MONITORING RECEIVED MESSAGE FROM ESP32
PYTHON LOOP

Fig. 8. Proposed technique in the laboratory stand based on millis() function and single string message with authenticating part concated to the end of the message



Output Signal from: Output from 7th transmitter

Reading from 1st transmitter	1.92
Reading from 2nd transmitter	1.82
Reading from 3rd transmitter	2.68
Reading from 4th transmitter	3.60
Reading from 5th transmitter	4.49
Reading from 6th transmitter	1.37
Reading from 7th transmitter	4.88
Reading from 8th transmitter	3.37

```
Data1: 97.00+Data2: 90.00+Data3: 137.00+Data4: 180.00+Da
Data1: 97.00+Data2: 90.00+Data3: 137.00+Data4: 180.00+Da
Data1: 99.00+Data2: 92.00+Data3: 137.00+Data4: 183.00+Da
Data1: 97.00+Data2: 93.00+Data3: 137.00+Data4: 185.00+Da
Data1: 98.00+Data2: 93.00+Data3: 136.00+Data4: 184.00+Da
Data1: 98.00+Data2: 92.00+Data3: 138.00+Data4: 182.00+Da
Data1: 97.00+Data2: 91.00+Data3: 137.00+Data4: 181.00+Da
Data1: 98.00+Data2: 92.00+Data3: 137.00+Data4: 184.00+Da
Data1: 98.00+Data2: 92.00+Data3: 135.00+Data4: 183.00+Da
Data1: 96.00+Data2: 92.00+Data3: 136.00+Data4: 183.00+Da
Data1: 95.00+Data2: 92.00+Data3: 134.00+Data4: 182.00+Da
Data1: 98.00+Data2: 91.00+Data3: 138.00+Data4: 182.00+Da
Data1: 97.00+Data2: 93.00+Data3: 138.00+Data4: 184.00+Da
Data1: 98.00+Data2: 93.00+Data3: 138.00+Data4: 184.00+Da
Data1: 99.00+Data2: 93.00+Data3: 138.00+Data4: 185.00+Da
Data1: 98.00+Data2: 93.00+Data3: 138.00+Data4: 185.00+Da
Data1: 98.00+Data2: 93.00+Data3: 138.00+Data4: 185.00+Da
Data1: 97.00+Data2: 93.00+Data3: 138.00+Data4: 185.00+Da
```

Fig. 9. GUI based on Visual Basic

Table 1. Examples of authentication messages and their significance. Authentication messages are illustrated at the ends of the string stream shown in Figure 5

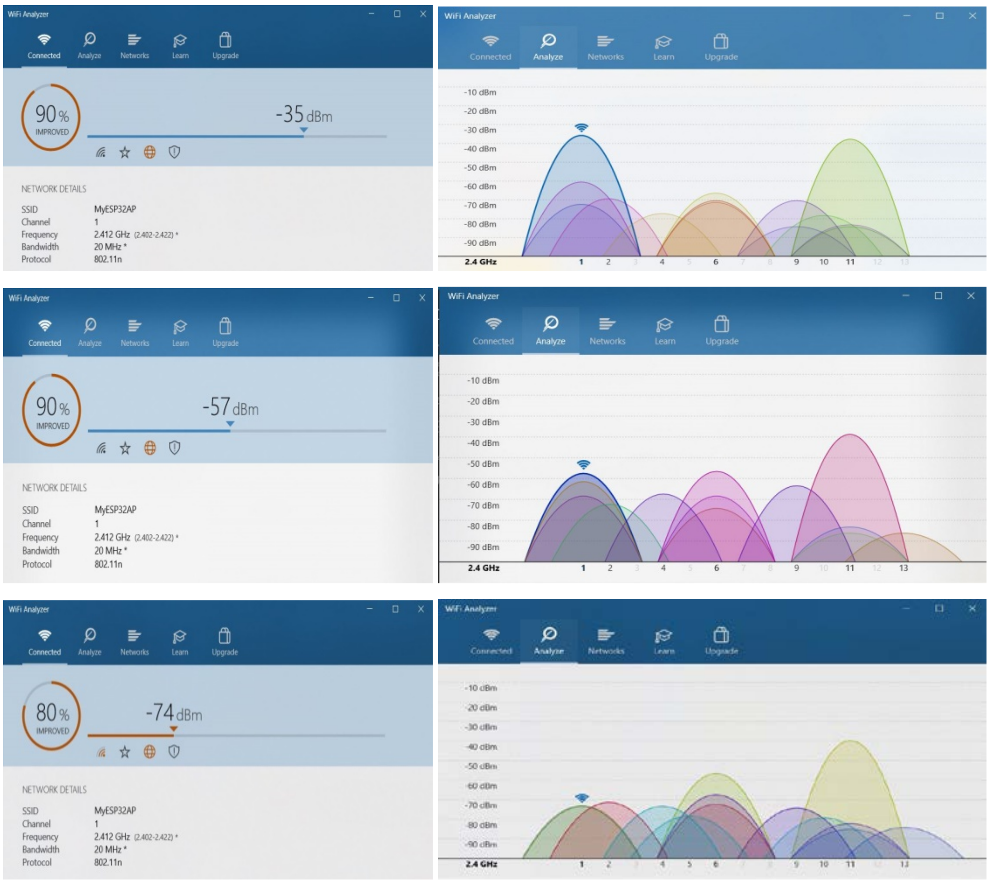
Message	**Previous Analogue Output**	**Current Analogue Output**	**Significance**
T01F	Null	Passed from 1st sensor	Controllers informs WebSerial that Analogue output has started being passed from 1st sensor
TF01	Null	Passed from 1st sensor	WebSerial confirms that analogue output corresponds to the reading of the 1st sensor
T01F01	Passed from 1st sensor	Passed from 1st sensor	Outputting from 1st sensor till timer toggles the output
T02F01	Passed from 1st sensor	Passed from 2nd sensor	Controllers informs WebSerial that Analogue output has started being passed from 2nd sensor
T01F02	Passed from 1st sensor	Passed from 2nd sensor	WebSerial confirms that analogue output corresponds to the reading of the 2nd sensor
T02F02	Passed from 2nd sensor	Passed from 2nd sensor	Outputting from 2nd sensor till timer toggles the output
T02F03	Passed from 2nd sensor	Passed from 3rd sensor	WebSerial confirms that analogue output corresponds to the reading of the 3rd sensor
T03F03	Passed from 3rd sensor	Passed from 3rd sensor	Outputting from 3rd sensor till timer toggles the output
T01F08	Passed from 8th sensor	Passed from 1st sensor	Controllers informs WebSerial that Analogue output started to be passed from the 8th sensor
T08F01	Passed from 8th sensor	Passed from 1st sensor	WebSerial confirms that analogue output corresponds to the reading of the 8th sensor
T08F08	Passed from 8th sensor	Passed from 8th sensor	Outputting from 8th sensor till timer toggles the output

Source: own study.

4. FUTURE WORK

In order to improve understanding of the principle of coexistence between wireless technologies, the laboratory stand will be the backbone of several future articles demonstrating the improved performance of a given wireless instrumentation system, rendered by the principle of coexistence between wireless technologies.

In order to improve understanding of such a phenomenon, Figure 10 illustrates three RSSI (Received Signal Strength Indicator) levels of the wireless WiFi signal generated by the ESP32 controller and measured by a WiFi analyser with respect to the distance from the host controller. One example for future research is to explore the possibilities of increasing the RSSI levels of the WiFi signal generated by the ESP32 controller at farther distances through using other protocols, such as ESP-NOW, in conjunction with ESP-WebSerial.



NETWORK DETAILS

SSID	MyESP32AP
Channel	1
Frequency	2.412 GHz (2.402-2.422) *
Bandwidth	20 MHz *
Protocol	802.11n

Fig. 10. Decreased (RSSI) Levels of -35,-57 and -74 dbm corresponding to distances of 1, 6 and 12 meters from Host Controller, respectively

5. DISCUSSION

This article has introduced a description for a laboratory stand, through which an authenticated wireless WiFi communication and serial communication.can take place simultaneously. The purpose of this stand is to analyse the possibilities of coexistence between wireless technologies: those dedicated to industrial automation, such as wireless HART, and those dedicated to general purpose use, such as WiFi and BLE. The authentication principle of transmitted information from multiple sensors was implemented in this laboratory stand through programming techniques based on the cooperation between ATMEGA2560 and ESP32 controllers on one side and the WebSerial remote monitor (Automated by Python) on the other side. Based on the experimental trials conducted on the stand, the article recommended avoiding some programming techniques in order to achieve successful simultaneous serial and wireless communication when using both ATMEGA2560 and ESP32 controllers.

Techniques based on excessive use of the *delay()* function and individual authentication messages from each sensor will lead to the program crashing due to overloading of the serial buffer with huge number of messages corresponding to each sensor. Techniques based on using timers and the interrupt subroutine should be considered carefully, especially with the ESP32 controller, as activating the interrupt subroutine immediately deactivates the WiFi generated by the ESP32 controller. The programming technique adopted by the laboratory stand and recommended by the article is based on three basic principles.

The first principle is to limit the number of messages processed by the stand to a single string message in order to avoid the program crashing. This string message contains the readings from all sensors in addition to the authentication information at the end of the message. The authentication information in the last part of the message will change according to a specific timing pattern.

The second principle on which the proposed technique is based is to depend on the *millis()* function to create the timing pattern, which toggles between the sensors to obtain a more accurate timing pattern and also to avoid the execution of the main program being stopped by using the *delay()* function.

The third principle is to disconnect (using the *disconnect()* method) the WiFi at the ESP32 controller, whenever serial information was detected from the ATMEGA2560 controller, and then reconnecting the WiFi (using the *reconnect()* method) when the serial information is received. In the future, more elements will be added to this laboratory stand to achieve the principle of coexistence between different wireless technologies. Wireless HART adapters, other wireless WiFi and BLE modules are examples of these elements. Additionally, more ESP32 controllers will be attached to the stand in order to examine the coexistence between different wireless protocols provided by ESP32 controller, such as ESP-NOW, ESP- mesh and ESP Client-Server.

REFERENCES

Al Kala, M.O., Seidman, S., Quang, J., 2018, *An Outlook on Wireless Coexistence with Focus on Medical Devices*, IEEE Electromagnetic Compatibility Magazine, vol. 7, no. 3, pp. 60–64.

Arduino Forum, 2017, *Why Is Using Delay Bad?* https://forum.arduino.cc/t/why-is-using-delay-bad/465334/1 (11.04.2023).

Arduino References, 2019, *Communication Function: Serial*, https://www.arduino.cc/reference/en/language/functions/communication/serial/ (11.04.2023).

Arduino References, 2021, *Datatypes: Stringobject String()*, https://www.arduino.cc/reference/en/language/variables/data-types/stringobject/ (11.04.2023).

Arduino References, 2022, *Time Function Millis()*, https://reference.arduino.cc/reference/en/language/functions/time/millis/ (11.04.2023).

Arduino, 2023, *Arduino® MEGA 2560 Rev3Product Refrence Manual*, SKU: A00067, https://fiona.dmcs.pl/~cmaj/SM_2st/arduino-mega2560_R3-datasheet.pdf (12.04.2023).

Cura, M.G., 2020, *Industrial Arduino Millis () vs Delay ()*, Boot & Work Corp. S.L., https://www.industrialshields.com/blog/arduino-industrial-1/post/industrial-arduino-millis-vs-delay-248 (11.04.2023).

DelCastillo, G., 2021, *Python Websocket Server*, https://github.com/ParametricCamp/Tutorial Files/tree/master/Misc/WebSockets (11.04.2023).

Espressif Systems, 2023, *Arduino-ESP32 Release 2.0.6*.

Gomaa, R., 2020, *Coexistence Study of 2.4 GHZ Wireless Technologies for Nuclear and Radiological Applications*, International Journal of Engineering Research & Technology (IJERT), vol. 9, no. 8.

Joseph, J., 2022, *ESP32 Timers & Timer Interrupts*, https://circuitdigest.com/microcontroller-projects/esp32-timers-and-timer-interrupts (11.04.2023),

LaSorte, N.J., Seidman, S., Quang, J., 2016, *Experimental Method for Evaluating Wireless Coexistence of Wi-Fi Medical Devices*, Biomedical Instrumentation and Technology, vol. 50(s6), pp. 18–25.

Liang, O., 2013, *Arduino Timer and Interrupt Tutorial*, https://oscarliang.com/arduino-timer-and-interrupt- tutorial /#:~:text=On%20the%20Arduino% 20Mega%20we,and%2013%3A%20 controlled%20by%20Timer0 (11.04.2023),

Pepperl+Fuchs, 2015, *WHA-BLT-F9D0-N-A0 BULLET Wireless HART Adapter Instruction Manual*, http://files.pepperl-fuchs.com/webcat/navi/productInfo/doct/tdoct4909__eng.pdf ?v=20230403152657 (12.04.2023).

Randomnerdtutorials, January 2020a, *ESP32 Client-Server Wi-Fi Communication Between Two Boards*, https://randomnerdtutorials.com/esp32-client-server-wi-fi/ (12.04.2023).

Randomnerdtutorials, January 2020b, *Getting Started with ESP-NOW (ESP32 with Arduino IDE)*, https://randomnerdtutorials.com/esp-now-esp32-arduino-ide/ (12.04.2023).

Randomnerdtutorials, November 2020, *ESP-MESH with ESP32 and ESP8266: Getting Started (PainlessMesh Library)*, https://randomnerdtutorials.com/esp-mesh-esp32-esp8266-painless-mesh/ (12.04.2023).

Randomnerdtutorials, February 2021, *ESP32 Useful Wi-Fi Library Functions (Arduino IDE)*, https://randomnerdtutorials.com/esp32-useful-wi-fi-functions-arduino/ (11.04.2023).

Randomnerdtutorials, August 2021, *ESP32 WebSerial: Web-based Remote Serial Monitor*, https://randomnerdtutorials.com/esp32-webserial-library/ (11.04.2023).

Santos, R., 2015, *Why You Shouldn't Always Use the Arduino Delay Function*, https://randomnerdtutorials.com/why-you-shouldnt-always-use-the-arduino-delay-function/ (11.04.2023).

Sharma, A., 2021, *WebSerial: Remote Serial Monitor for ESP8266 and ESP32*, https://github.com/ayushsharma82/WebSerial (11.04.2023).

Solomon, B., 2019, *ESP32: Async IO in Python: A Complete Walkthrough*, https://realpython.com/async-io-python/ (11.04.2023).

Stoffregen, P., 2015, *TimerOne Modified Library*, https://github.com/PaulStoffregen/TimerOne (11.04.2023).

Techtutorialsx, 2022, *ESP32: WebSerial Console Over Soft AP*, https://techtutorialsx.com/2021/07/23/esp32-webserial-console-over-soft-ap/ (11.04.2023).

Toptechboy, 2018, *Lesson 30: Advanced Software Interrupt Techniques for Reading Serial Data with Arduino*, https://toptechboy.com/lesson-30-advanced-software-interrupt-techniques-for-reading-serial-data-with-arduino/ (11.04.2023).