STUDIA UBB PHYSICA, Vol. 65 (LXV), 1-2, 2020, pp. 69-86
(RECOMMENDED CITATION)
doi:10.24193/subbphys.2020.08

P2P (BILATERAL) COMMUNICATION BETWEEN NODEMCU ESP8266 BOARDS USING ARDUINO IDE

M. TODICA[1,*]

ABSTRACT. Bilateral communication between the boards NodeMcu ESP826 is achieved using the WI FI capabilities of these devices and particularly code based on Arduino IDE. The system is used to control servos, DC motors or led. The duplex communication allows feedback action between sender and receiver. Real time feedback is obtained by particularly connection of the servo to the board.

Keywords: *Bilateral communication, NodeMcu ESP8266, Arduino.*

INTRODUCTION

The bilateral communication between two devices is now one of the most important features of the intelligent devices. The duplex communication is required especially in the long range remote control in order to ensure the feedback, the acknowledgement of the achievement of the transmitted order, [1, 2]. Generally two or many devices are connected to a router, which offer the possibility of connection between them trough a local network or trough the web. The system implies the use of an intermediate device, the router, and customized protocols. Sometimes, for short range of communication, we need a simple and direct communication between the devices, the so called peer to peer connection, (P 2 P). The intermediate device, the router, in not more necessarily and the communication is much faster. It is possible to achieve this task with simple equipment and popular software platforms as Arduino, [3]. One of the most popular devices able to fulfill this task is the NodeMcu ESP 8266 board. It is known especially for its capability to connect wirelessly to a local router (the internet), in the 2.4 GHz band, but with particular setting these boards can establish bilateral communication between them

[1] "Babes-Bolyai" University, Faculty of Physics, M. Kogalniceanu No 1, 400084 Cluj-Napoca, Romania.
** Corresponding author: mihai.todica@phys.ubbcluj.ro*

without any intermediate devices. Two or many devices can be connected together, but the communication can be established only between the desired ones, using the identification MAC address of each board. In this work we will show how to establish bilateral communication between two NodeMcu ESP 8266 boards to control DC, servos or led with feedback control.

EXPERIMENTAL

The system consists of two parts, the sender, (the Master), named Tx, and the receiver, (the Slave), named Rx. The system is conceived to control 4 led or two DC motors and one servo SG 90. The led or the DC motors are controlled by four push switches connected between the pins D3, D4, D5, D6 and GND. The servo is controlled by one 10K potentiometer connected at +3.3V, GND and A0 of the Tx NodeMcu ESP 8266 board. The pins D3-D6 are pulled up by 4x10K resistors connected to +3,3V. These pins are designed in the code by GPIO 0, GPIO 2, GPIO 14 and GPIO 12 respectively. For the feedback acknowledgement we used the monochrome Oled 0.94" display (SSD1306) connected at +3.3V, GND, D1, (Oled SCK), and D2, (Oled SDA), pins, (Fig, 1.a, b).

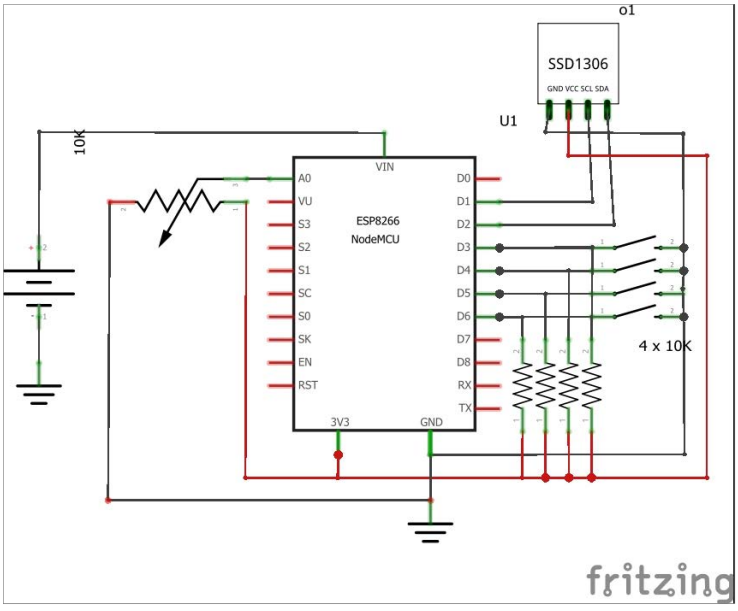

Fig. 1. a. *The electric diagram of the Tx board*

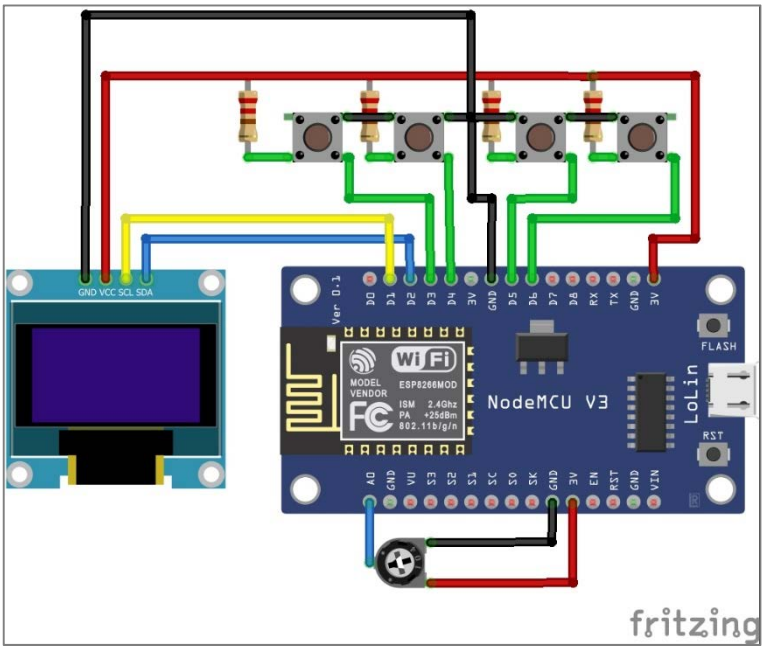

Fig. 1. b. *The physical connection of the electric parts of the Tx board*

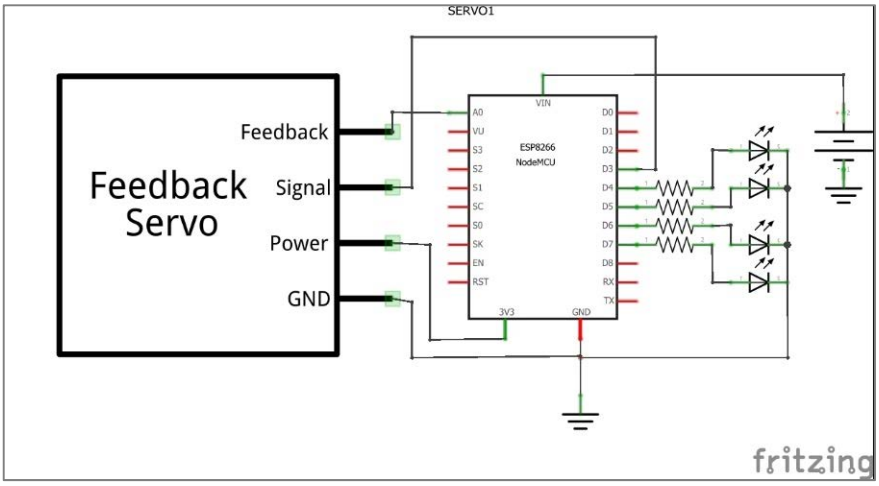

Fig. 2. a. *The electric diagram of the receiver with feedback servo and 4 led.*

The Rx contains 4 led connected to pins D5, D6, D7, D8, (designed by GPIO 14, GPIO 12, GPIO 13, GPIO 15 in the code), and GND and one servo connected to +3.3V, GND and D3, (GPIO 0 in the code), (Fig. 2 a). The wiper of the servo is connected to A0 pin for feedback. To access the wiper of the potentiometer we must open the servo and solder a wire as shown in figure 2. b, [4, 5]. Another version of the receiver contains two DC motors driven by the H bridge L 298 (or MX 1508) and the servo SG 90. The entries IN1-IN4 of the H bridge are connected to pins D5-D8, (Fig. 3 a, b). The codes for both versions of the receiver are the same. All parts of the system, the electronics and the motors are powered by a single 5V supply voltage connected between Vin and GND of NodeMcu board, but for high torque DC motors the bridge must be powered by a separately source, [6, 7].

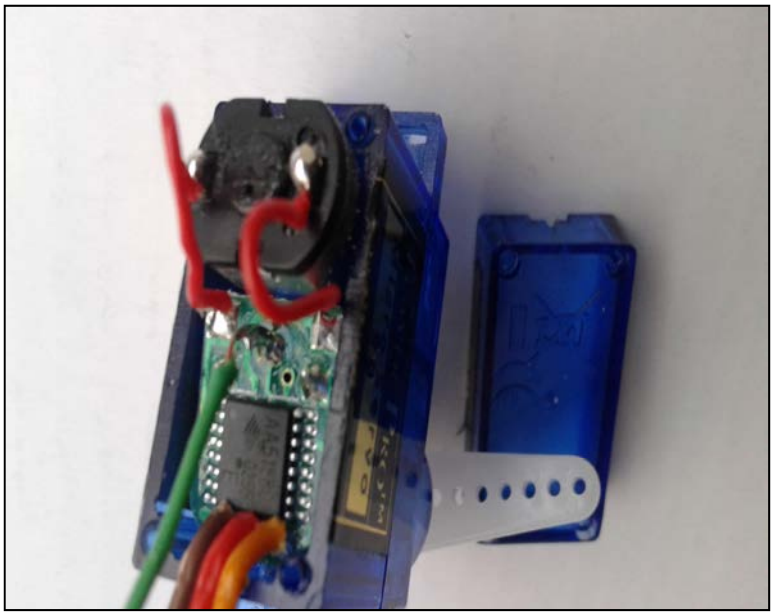

Fig. 2. b. *Feedback connection of the servo SG 90. The green wire is connected to the wiper of the servo.*

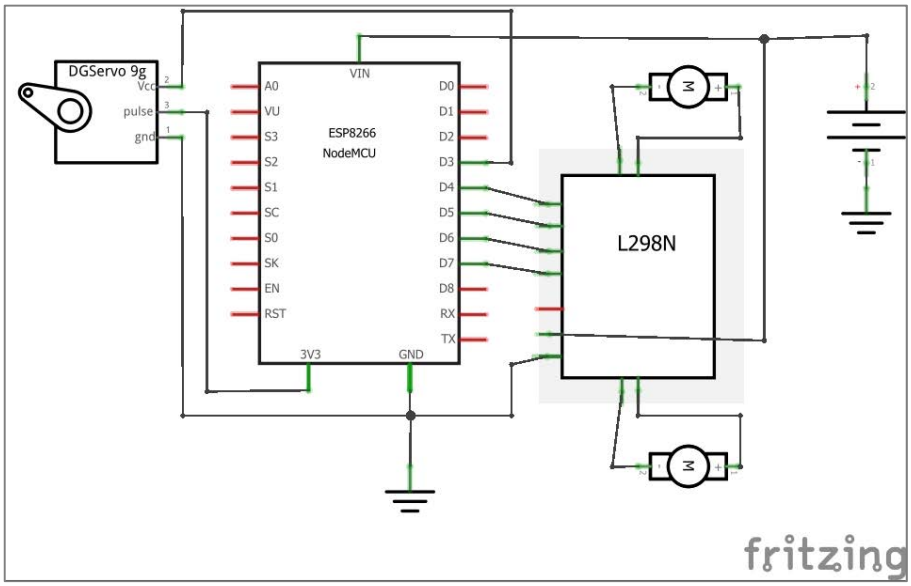

Fig. 3. a. *The electric diagram of the receiver with simple servo and 2 DC .*

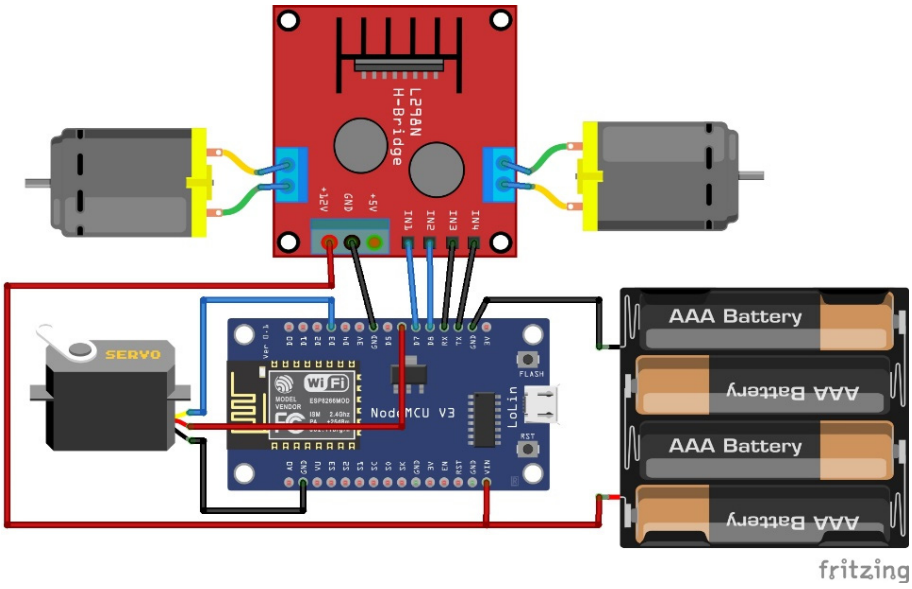

Fig. 3. b. *The physical connection of the parts of the receiver with simple servo and 2 DC.*

RESULTS AND DISCUSSION

The two-way communication between the ESP8266 NodeMCU boards is based on the use of the ESP-NOW protocol developed by Espressif, (ESP-NOW), [8]. This is a fast communication protocol that enables multiple devices to exchange small messages (up to 250 bytes) between them. It is very versatile, allowing one-way or two-way communication without using Wi-Fi. The main feature of this protocol is the possibility to establish connection between the desired device using theirs own MAC addresses. The use of this protocol with Arduino IDE implies some stages. First we need to set the Arduino IDE to recognize and communicate with the ESP 8266 board. Only the latest versions of Arduino, i.e. 1.8.13 version, have this facility. To do this we need to insert in the Arduino IDE > File**>** Preferences the following address: http://arduino.esp8266.com/stable/package_esp8266com_index.json.

Then on the Boards Manager we must install the **"**ESP8266 by Community" board, [9]. Then we must install the libraries requested by the code. The espnow.h library comes installed by default when installing the ESP8266 board. The ESP-NOW protocol allows to define from the beginning the role of each device, "master" or "slave", but also to change the role of devices from "master" to "slave" and vice versa. The first feature is used for the unilateral communication, when one device is set as transmitter and the other one as receiver, the role of each device remaining unchanged. The second feature is used for bilateral communications, when the transmitter sends the message to the receiver, after that it passes on the receiver mode, waiting the acknowledgement message from the receiver. The second device is set as receiver, but after receiving the message from the transmitter, it passes on the sender mode in order to transmit the acknowledgement message. After that the device returns to its initial state of receiver, waiting from a new message. To perform this operation each device must know the MAC address of its correspondent. In this way the connection is established only between desired devices, allowing other devices to use the same frequency channels. The MAC addresses can be obtained with a supplementary code presented below. The cod must be uploaded to both devices. After running the code the MAC address is displayed on the Serial Monitor, [10]. When compiling the code, we must select the ESP8266 board in the Boards menu.

```
// The MAC addresses code:

#ifdef ESP32
#include <WiFi.h>
#else
#include <ESP8266WiFi.h>
#endif

void setup(){
Serial.begin(115200);
Serial.println();
Serial.print("ESP Board MAC Address:  ");
Serial.println(WiFi.macAddress());
}

void loop(){
}
// end of the code
```

In the unilateral communication the MAC address of the receiver must be introduced only in the code of the sender. The role of each device, sender or receiver is established from the beginning by the code. For bilateral communication the MAC addresses are necessarily for both the partners of connection. The devices change the role, sender or receiver, in function of the code, for which reason each board need to known the other MAC address.

We present only the codes for bilateral communication.

```
// The sender code.

#define analogPin A0 //potentiometer connected to A0
#include <Wire.h>

#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels
```

```
// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)

#define OLED_RESET -1 // Reset pin # (or -1 if sharing Arduino reset pin)
Adafruit_SSD1306 display (SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

#include <ESP8266WiFi.h>
#include <espnow.h>

// REPLACE WITH THE MAC Address of your receiver
uint8_t broadcastAddress[] = {0x84, 0x0D, 0x8E, 0xAA, 0xA3, 0xA3};

// Define the variables to be sent for each button
int temperatureX;
int temperatureY;
int temperatureZ;
int temperatureQ;
float analog;

// Define variables to store incoming readings
int incomingTempX;// led 1
int incomingTempY;// led 2
int incomingTempZ;// led 3
int incomingTempQ;// led 4

float incomingAn;//for servo

// Updates readings every 0.2 seconds initial,
const long interval = 200;
unsigned long previousMillis = 0; // will store last time button data were updated

// Variable to store if sending data was successful
String success;

//Structure example to send data
//Must match the receiver structure
typedef struct struct_message {
   int tempX;
   int tempY;
```

```
    int tempZ;
    int tempQ;
    float ana;
} struct_message;

// Create a structure message called Readings to hold sensor readings
struct_message Readings;

// Callback when data is sent
void OnDataSent(uint8_t *mac_addr, uint8_t sendStatus) {
  Serial.print("Last Packet Send Status: ");
  if (sendStatus == 0){
  Serial.println("Delivery success");
  }
  else{
  Serial.println("Delivery fail");
  }
}

// Callback when data is received
void OnDataRecv(uint8_t * mac, uint8_t *incomingData, uint8_t len) {
  memcpy(&incomingReadings, incomingData, sizeof(incomingReadings));
  Serial.print("Bytes received: ");
  Serial.println(len);
  incomingTempX = incomingReadings.tempX;
  incomingTempY = incomingReadings.tempY;
  incomingTempZ = incomingReadings.tempZ;
  incomingTempQ = incomingReadings.tempQ;

  incomingAn = incomingReadings.ana;
  //for Servo
}

void getReadings(){

  temperatureX=digitalRead(12);
  //button on D6 defined in the code by GPIO 12
  temperatureY=digitalRead(14);
```

```
  //button on D5 defined in the code by GPIO 14
  temperatureZ=digitalRead(0);
  //button on D3 defined in the code by GPIO 0
  temperatureQ=digitalRead(2);
  //button on D4 defined in the code by GPIO 2

  analog = analogRead(analogPin);
  //potentiometer for servo
}

void printIncomingReadings(){
  //display on OLED
     display.clearDisplay();
     display.setTextSize(2);
     display.setCursor(0,0);
     display.print("An:");
     display.print(incomingAn);
     display.print(" ");

     display.setCursor(0, 25);
     display.print("Y:");
     display.print(incomingTempY);
  //receive back from the Rx the data sent for led command
     display.print(" X:");
     display.print(incomingTempX);

     display.setCursor(0, 50);
     display.print("Z:");
     display.print(incomingTempZ);

     display.print(" Q:");
     display.print(incomingTempQ);
     display.setCursor(110, 56);
     display.display();
}

void setup() {
pinMode(14, INPUT);
```

```
pinMode(12, INPUT);
pinMode(0, INPUT);
pinMode(2, INPUT);
// buttons connected to D3, D4, D5, D6

if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
  Serial.println(F("SSD1306 allocation failed"));
  for(;;); // Don't proceed, loop forever
}
  display.clearDisplay();
  display.setTextColor(WHITE);

  Serial.begin(115200);

// Set device as a Wi-Fi Station
WiFi.mode(WIFI_STA);
WiFi.disconnect();

// Init ESP-NOW
if (esp_now_init() != 0) {
  Serial.println("Error initializing ESP-NOW");
  return;
}

// Set ESP-NOW Role
esp_now_set_self_role(ESP_NOW_ROLE_COMBO);

// Once ESPNow is successfully initialized, we will register for Send CB to
// get the status of transmitted packet
esp_now_register_send_cb(OnDataSent);

// Register peer
esp_now_add_peer(broadcastAddress, ESP_NOW_ROLE_COMBO, 1, NULL, 0);

// Register for a callback function that will be called when data is received
esp_now_register_recv_cb(OnDataRecv);
}
```

```
void loop() {
unsigned long currentMillis = millis();
if (currentMillis - previousMillis >= interval) {
   previousMillis = currentMillis;

//Get buttons readings
getReadings();
//Set values to send
Readings.tempX = temperatureX;
Readings.tempY = temperatureY;
Readings.tempZ = temperatureZ;
Readings.tempQ = temperatureQ;

Readings.ana = analog;
//read potentiometer
// Send message via ESP-NOW
esp_now_send(broadcastAddress, (uint8_t *) &Readings, sizeof(Readings));

// Print incoming readings
printIncomingReadings();
}
}
// End of the code
```

The MAC address of the receiver is introduced into the code of the transmitter by the following line of the code:

```
// REPLACE WITH THE MAC Address of your receiver
“ uint8_t broadcastAddress[] = {0x84, 0x0D, 0x8E, 0xAA, 0xA3, 0xA3}; ”
```

The transmitter is set as sender by default, but after transmitting the order it passes in receiver mode and wait for the feedback message. The following line of the code is responsible for this job.

```
“esp_now_set_self_role(ESP_NOW_ROLE_COMBO);”
```

Other explanations are included into the code.

The corresponding receiver code is presented below:

```
// Receiver code

#include <Servo.h>
Servo Servo1;
int angle=80;
#define analogPin A0 //leg pot la A0

#include <Wire.h>

#include <ESP8266WiFi.h>
#include <espnow.h>

// REPLACE WITH THE MAC Address of the sender
uint8_t broadcastAddress[] = {0x84, 0x0D, 0x8E, 0xB0, 0xCE, 0x62};

int temperatureX;
int temperatureY;
int temperatureZ;
int temperatureQ;
// variables for led control
float analog;

// Define variables to store incoming readings
int incomingTempX;// led 1
int incomingTempY;// led 2
int incomingTempZ;// led 3
int incomingTempQ;// led 4
float incomingAn;

const long interval = 200;
unsigned long previousMillis = 0;
// Variable to store if sending data was successful
String success;

//Structure example to send data
//Must match the sender structure
typedef struct struct_message {
   int tempX;
```

```
    int tempY;
    int tempZ;
    int tempQ;

    float ana;
} struct_message;
struct_message Readings;

// Create a struct_message to hold incoming data
struct_message incomingReadings;

// Callback when data is sent
void OnDataSent(uint8_t *mac_addr, uint8_t sendStatus) {
  Serial.print("Last Packet Send Status: ");
  if (sendStatus == 0){
  Serial.println("Delivery success");
}
  else{
  Serial.println("Delivery fail");
}
}

// Callback when data is received
void OnDataRecv(uint8_t * mac, uint8_t *incomingData, uint8_t len) {
  memcpy(&incomingReadings, incomingData, sizeof(incomingReadings));
  Serial.print("Bytes received: ");
  Serial.println(len);
  incomingTempX = incomingReadings.tempX;
  incomingTempY = incomingReadings.tempY;
  incomingTempZ = incomingReadings.tempZ;
  incomingTempQ = incomingReadings.tempQ;

  incomingAn = incomingReadings.ana;
  //for servo
}

void getReadings(){
```

```
temperatureX=incomingTempX;
temperatureY=incomingTempY;
temperatureZ=incomingTempZ;
temperatureQ=incomingTempQ;
analog =  analogRead(analogPin);
}
void setup() {
  pinMode(15, OUTPUT);
  pinMode(14, OUTPUT);
  pinMode(13, OUTPUT);
  pinMode(12, OUTPUT);

  digitalWrite(15, LOW);
  digitalWrite(14, LOW);
  digitalWrite(13, LOW);
  digitalWrite(12, LOW);

  Servo1.attach(0);//servo on D3

  Serial.begin(115200);

  // Set device as a Wi-Fi Station
  WiFi.mode(WIFI_STA);
  WiFi.disconnect();

  // Init ESP-NOW
  if (esp_now_init() != 0) {
  Serial.println("Error initializing ESP-NOW");
  return;
  }

  // Set ESP-NOW Role
  esp_now_set_self_role(ESP_NOW_ROLE_COMBO);

  // Once ESPNow is successfully Init, we will register for Send CB to
  // get the status of Trasnmitted packet
  esp_now_register_send_cb(OnDataSent);
```

```
  // Register peer
  esp_now_add_peer(broadcastAddress, ESP_NOW_ROLE_COMBO, 1, NULL, 0);

  // Register for a callback function that will be called when data is received
  esp_now_register_recv_cb(OnDataRecv);
}
void loop() {
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;
    getReadings();

    // Set values to send
    Readings.tempX = temperatureX;
    Readings.tempY = temperatureY;
    Readings.tempZ = temperatureZ;
    Readings.tempQ = temperatureQ;
    Readings.ana = analog;

    // Send message via ESP-NOW
    esp_now_send(broadcastAddress, (uint8_t *) &Readings, sizeof(Readings));

if (incomingTempX == 1) {
    digitalWrite(14, LOW);
    }
    else {
    digitalWrite(14, HIGH);
    }

    if (incomingTempY == 1) {
    digitalWrite(15, LOW);
    }
    else {
    digitalWrite(15, HIGH);
    }

if (incomingTempZ == 1) {
    digitalWrite(13, LOW);
```

```
   }
   else {
   digitalWrite(13, HIGH);
   }
   if (incomingTempQ == 1) {
   digitalWrite(12, LOW);
   }
   else {
   digitalWrite(12, HIGH);
   }

angle = map(incomingAn, 0, 1023, 0, 180);
//Map the readings values to an angle from 0 to 180
   Servo1.write(angle);
  }
}
// End of the code
```

After receiving the data the receiver passes in sender mode, in order to ensure the feedback. The following line is responsible for this action:

<esp_now_set_self_role(ESP_NOW_ROLE_COMBO);>

The receivers read the state of the led and of the servo and send back these data to the sender for feedback.

<void getReadings(){….}>

After that it come back in the receiver mode and waits for new data.

Other explanations of how the code is working are included into the code itself. It works with both the circuits presented in figures 2 and 3. Every time when a button is pressed on the Tx board, the corresponding led on the Rx board, or the corresponding DC motor, pass in ON state and the OLED display change from 1 to zero. When the potentiometer is rotate on the Tx board, the servo attached to Rx board rotate with the corresponding angle, and the value of the real angle of rotation is send back to the Tx. This value is displayed on the OLED.

CONCLUSION

The work demonstrates the possibility to establish direct bilateral communication between two NodeMcu ESP8266 boards without the use of intermediate devices. The system is used to control servos, DC motors or led. During the transmission each board change the role from receiver to sender and vice versa, ensuring the duplex communication. This action is ordered by specific programming code installed into each device. The communication is established only between desired devices, identified by theirs own MAC addresses. The duplex communication allows the feedback action between the receiver and sender. Real time feedback is achieved by particularly connection of the servo to the NodeMcu ESP8266 board.

REFERENCES

1. M. Todica, Feedback control of DC motors with long range HC 12 Transceiver and Arduino, Studia Univ. "Babes-Bolyai", Physica, Vol. 64 (LXIV), 1-2, 2019, pp. 91-100 doi:10.24193/subbphys.2019.10
2. M. Todica, Software feedback with HC 12 transceivers and Arduino Research Gate DOI:10.13140/RG.2.2.24560.61444
3. Arduino Website (www.arduino.cc/en/Guide/ Introduction)
4. M. Todica, Servo and RGB led feedback with smartphone and Blynk, Research Gate, DOI:10.13140/RG.2.2.23512.08960
5. http://dr-iguana.com/prj_TPro_SG90_2/
6. T. A. Antal, Acta Technica Napocensis, Series: Applied Mathematics, Mechanics, and Engineering, Vol. 60, Issue I, March, (2017).
7. S. D. Anghel, Bazele Electronicii analogice si digitale, Presa Universitară Clujeană, Cluj-Napoca, 2007, ISBN: 978-973-610-554-8
8. Rui Santos, Getting Started with ESP-NOW (ESP8266 NodeMCU with Arduino IDE), https://randomnerdtutorials.com/esp-now-esp8266-nodemcu-arduino-ide/
9. Rui Santos, Installing ESP8266 Board in Arduino IDE (Windows, Mac OS X, Linux), https://randomnerdtutorials.com/how-to-install-esp8266-board-arduino-ide/
10. Rui Santos**,** Get ESP32/ESP8266 MAC Address and Change It (Arduino IDE) https://randomnerdtutorials.com/get-change-esp32-esp8266-mac-address-arduino/