

Rapport

Système d'allumage de feux d'artifice à distance avec le protocole de communication Wi-Fi



OZENNE Léo
GOMES Kilian
PHOUTTHASAK Anousith
TOURE Mouhamadou
LE FAOU Gregoire

Encadré par : C.DELEBARRE

5A MT - Axe ASM

Table des matières

1. Introduction.....	3
2. Choix du microcontrôleur et de l'IDE.....	5
3. Travails et tests réalisés.....	6
3.1 Communication Bluetooth.....	6
3.1.1 Pupitre de contrôle sur PC.....	6
3.1.2 Formatage des trames.....	7
3.2 Communication Wi-Fi entre la carte de contrôle et la carte d'allumage.....	7
3.2.1 Tests de prise en main.....	7
3.2.1.1 Code carte de contrôle (ESP8266).....	8
3.2.1.2 Code carte d'allumage (ESP8266).....	9
3.3 Communication I2C entre la carte d'allumage et l'Arduino Nano.....	9
3.3.1 Tests de prise en main.....	10
3.3.1.1 Allume la LED interne d'une arduino Nano via le bus I2C.....	10
3.3.1.1.a Code de la carte d'allumage (ESP8266).....	10
3.3.1.1.b Code de la carte Arduino Nano.....	11
3.3.1.2 Allume 2 LED externes avec 2 Arduino Nano via le bus I2C.....	11
3.3.1.2.a Code de la carte d'allumage (ESP8266).....	12
3.3.1.2.b Code de la première carte Arduino Nano.....	14
3.3.1.2.c Code de la seconde carte Arduino Nano (bas).....	14
3.3.2 Projet.....	15
3.3.2.1 Code carte d'allumage (ESP8266).....	15
3.3.2.2 Code des carte Arduino Nano.....	15
4. Références.....	16
5. Conclusion.....	16

1. Introduction

Nous avons pour objectif de développer un système sécurisé permettant de déclencher à distance l'allumage de feux d'artifice en utilisant le protocole de communication Wi-Fi. Étant donné que l'allumage de feux d'artifice présente des risques pour la sécurité des utilisateurs, il est préférable d'opter pour un système permettant un déclenchement à distance, offrant ainsi une meilleure sécurité. De plus, notre système d'allumage offrirait la possibilité de coordonner et synchroniser l'allumage des différents feux d'artifice, ainsi que de détecter d'éventuelles erreurs lors de l'allumage.

Pour établir la communication entre le système d'allumage et le système de contrôle, nous avons choisi d'utiliser le protocole Wi-Fi. Ce protocole de communication est principalement utilisé pour des applications de domotiques et d'objets connectés. L'avantage du Wi-Fi est que ce protocole possède une portée adaptée à ce qui est requis pour notre application (env. 50 m en extérieur), qu'il est facilement adaptable pour communiquer avec un microcontrôleur et qu'il est facile d'incorporer des fonctionnalités de cryptage pour communiquer en toute sécurité.

Notre système se compose de cinq tubes d'allumage, chacun équipé de cinq allumeurs électriques (relais), permettant un contrôle individuel des fusées.

Voici le schéma de notre système illustrant son fonctionnement et ses composants:

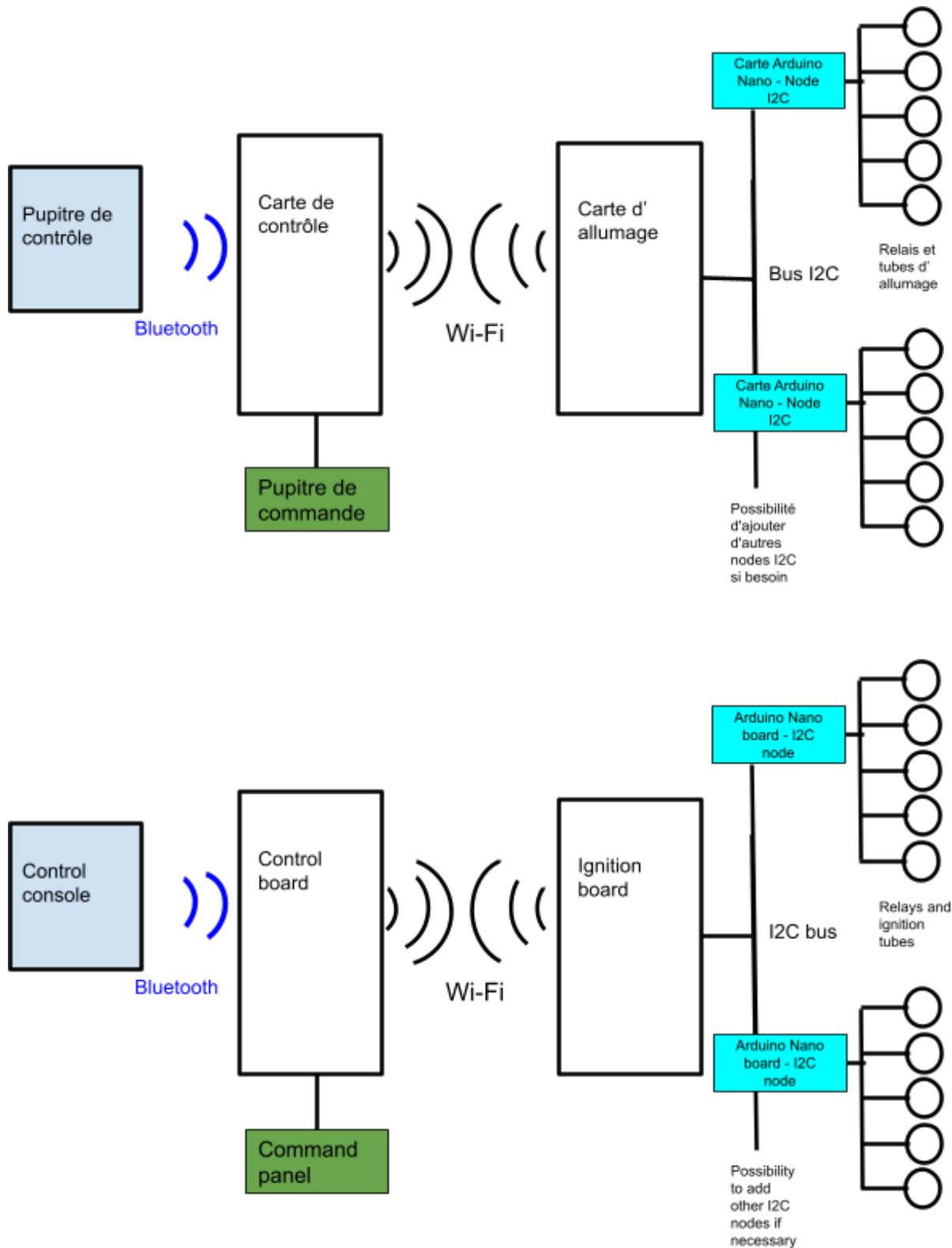


Figure 1: Schéma explicatif du fonctionnement de notre système

Ainsi, nous devons programmer trois cartes : la carte de contrôle, la carte d'allumage et la carte Arduino Nano. La carte de contrôle agit comme un relais entre les instructions provenant du pupitre de contrôle (bluetooth) et la carte de contrôle. Quant à la carte

d'allumage, elle recevra les instructions via le protocole Wi-Fi de la carte de contrôle et se chargera de communiquer avec nos cartes arduino nano via une communication I2C.

2.Choix du microcontrôleur et de l'IDE

Pour le choix de la carte de contrôle et de commande, nous avons tout d'abord étudié les 2 pistes les plus cohérentes pour notre application : un microcontrôleur programmable avec l'IDE de l'arduino et une carte de développement de Silicon Labs adaptée à la communication Z-wave (la Z-Wave Thunderboard 800 Series).

	Avantages	Inconvénients	Caractéristiques
ESP8266	<ul style="list-style-type: none"> -Dimension 58 x 31 x 12 mm -Utilisé pour la domotique -Portée (jusqu'à 50m) -Codage avec l'ide arduino -Capacité de calcul supérieur aux Arduino 	<ul style="list-style-type: none"> -Consommation d'énergie élevée 	<ul style="list-style-type: none"> • Tensilica 32-bit RISC CPU Xtensa LX10 • Mémoire : 512 Ko à 4 Mo flash, RAM 64 kB • Interface : 29 pin UART, SPI, I2C, GPIO, PWM, ADC, Wifi • Alimentation 3.3V
Serie 800 Thunderboard Silicon Labs	<ul style="list-style-type: none"> -Dimension assez compactes : 100*70mm -Longue portée (jusqu'à 150m) -Consommation d'énergie faible -Utilisé pour la domotique -Performance bien meilleure que la carte Arduino 	<ul style="list-style-type: none"> -Développement en C plus complexe que le langage Arduino. 	<ul style="list-style-type: none"> • ARM Cortex M33 32bits • Mémoire : 512 ko flash, 64 ko RAM • Transmission sans fil avec antenne SMA • Plusieurs capteurs intégrés • interface : 20 pin GPIO, UART, SPI, I2C, USB • Alimentation 3.6 à 6V CC

Concernant la communication entre la carte de commande et la carte de contrôle, nous étions partis sur le protocole de communication Z Wave qui était intégré à la carte Thunderboard Serie 800. Cependant, vu la complexité de l'IDE et du langage de programmation, nous avons finalement opté pour la carte ESP8266 et d'utiliser le protocole Wi-Fi (intégré) pour la communication. Nous allons donc intégrer un module bluetooth à notre carte de contrôle afin de pouvoir recevoir les données de notre pupitre de contrôle.

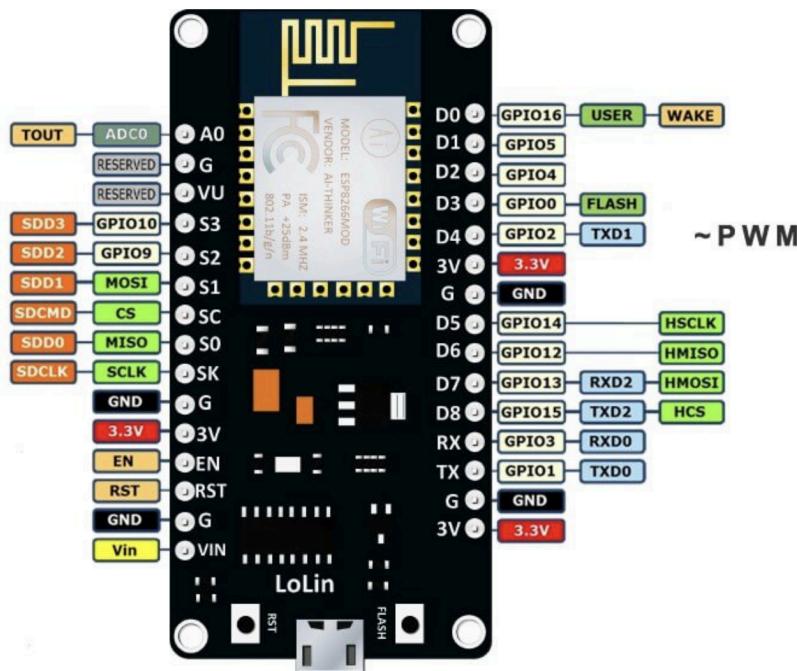


Figure 2: Ports/pin de la ESP8266

3. Travails et tests réalisés

3.1 Communication Bluetooth

3.1.1 Pupitre de contrôle sur PC

Pour le pupitre de contrôle envoyant les trames nécessaires à l'allumage des différents cylindres en individuel ou en séquence préprogrammée, le choix a été de le développer en python pour en faire une application PC.



Figure 3 : Pupitre de contrôle

Ce pupitre se sépare en deux parties. La partie Settings à gauche permettant de se connecter à la carte de contrôle via Bluetooth, d'allumer les différents cylindres, d'envoyer une séquence d'allumage et d'envoyer l'instruction de début de séquence afin que la carte d'allumage appelle les différents cylindres de façon automatique. La partie droite est une console constituée d'une zone de texte affichant les logs des différents messages envoyés

ainsi que tous les messages envoyés par la carte de contrôle afin de monitorer le bon déroulement des différentes actions réalisées par les cartes.

3.1.2 Formatage des trames

Afin de pouvoir communiquer entre les différents supports matériels et de la compréhension entre les différents langages de programmations (Python et Arduino) nous avons fait le choix de définir un système de trame de données. Ces trames sont des chaînes de caractères de maximum 64 caractères codés en utf-8. Afin de différencier les différentes instructions, nous avons mis en place des flags uniques pour chaque instructions.

Pour l'allumage de cylindre individuel, la trame est construite de la manière suivante : le flag M afin de prévenir que c'est un allumage manuel d'un cylindre, une lettre en minuscule représentant l'identifiant d'une Arduino nano puis finalement d'un chiffre représentant l'identifiant du cylindre lié à cette Arduino non. Nous avons alors une trame type : Mb2 allumage manuel du cylindre numéro 2 de l'Arduino nano b.

Pour ce qui est des séquences d'allumages nous allons avoir deux formes de trames car le module Bluetooth HC-05 ne peut recevoir que 64 caractères par envoies. Pour les séquence de moins de 60 caractères nous avons alors la trame type suivante : le flag S afin de prévenir que c'est une transmission de séquence, 00 l'identifiant de la trame composé de deux chiffres (ici 00 signifie que la trame n'est pas segmentée, il n'est donc pas nécessaire d'attendre une autre trame pour compléter les données), le flag T qui indique le début des données "utiles" de la trame. Pour une trame segmentée, nous avons une trame type de la forme suivante : le flag S, l'identifiant de la trame (01 pour la première trame, 02 pour la seconde, ...), le flag T. La donnée "utile" de ces trames est alors formatée à l'aide de deux flags. Le flag C représente l'allumage d'un cylindre, ce flag est suivi d'un identifiant d'Arduino nano puis d'un identifiant de cylindre comme pour l'allumage manuel d'un cylindre. Le flag D représente un délai, ce flag est suivi d'une suite de chiffres représentant la durée du délai en millisecondes. Le flag E représente la fin de la séquence. Pour une séquence de moins de 60 caractères nous avons alors une trame de la forme : S00TCb3Ca1D1000Ca2E. Pour une séquence de plus de 60 caractères nous avons alors des trames de la forme : S01TCb3Ca1... / S02TD1000Cc2E.

Le dernier flag définit est celui représentant l'instruction de démarrage de la séquence d'allumage. Ce flag est A.

3.2 Communication Wi-Fi entre la carte de contrôle et la carte d'allumage

Une fois la liaison Bluetooth établie, la carte de contrôle doit lire le message reçu afin de le transmettre à la carte d'allumage par WIFI.

3.2.1 Tests de prise en main

L'objectif est de prendre en main la communication WIFI entre la carte de contrôle et la carte d'allumage en respectant les trames définies précédemment

3.2.1.1 Code carte de contrôle (ESP8266)

```

1 #include <SoftwareSerial.h>
2 #include <Arduino.h>
3 #include <ESP8266WiFi.h>
4 #include <ESPAsyncTCP.h>
5 #include <ESPAsyncWebSrv.h>
6
7 // Replace with your network credentials
8 const char* ssid = "WebSocketTest";
9 const char* password = "123456789";
10
11 // Create AsyncWebServer object on port 80
12 AsyncWebServer server(80);
13
14 // Create a WebSocket object
15 AsyncWebSocket ws("/");
16
17 // Timer variables
18 unsigned long lastTime = 0;
19 unsigned long timerDelay = 30000;
20
21 //Create Bluetooth Variables
22 String cmd="";
23 SoftwareSerial hc05(13, 15);
24
25 // Initialize WiFi
26 void initWiFi() {
27
28     // Setting the ESP as an access point
29     Serial.print("Setting AP (Access Point)...");
30
31     // Remove the password parameter, if you want the AP (Access Point) to be open
32     WiFi.softAP(ssid, password);
33
34     IPAddress IP = WiFi.softAPIP();
35     Serial.println("AP IP address: ");
36     Serial.println(IP);
37 }
38
39 void handleWebSocketMessage(void *arg, uint8_t *data, size_t len) {
40     AwsFrameInfo *info = (AwsFrameInfo*)arg;
41     if (info->final && info->index == 0 && info->len == len && info->opcode == WS_TEXT) {
42         notifyClients();
43     }
44 }
45

```

```

46 void onEvent(AsyncWebSocket *server, AsyncWebSocketClient *client, AwsEventType type, void *arg, uint8_t *data, size_t len) {
47     switch (type) {
48         case WS_EVT_CONNECT:
49             Serial.printf("WebSocket client #u connected from %s\n", client->id(), client->remoteIP().toString().c_str());
50             break;
51         case WS_EVT_DISCONNECT:
52             Serial.printf("WebSocket client #u disconnected\n", client->id());
53             break;
54         case WS_EVT_DATA:
55             handleWebSocketMessage(arg, data, len);
56             break;
57         case WS_EVT_PONG:
58         case WS_EVT_ERROR:
59             break;
60     }
61 }
62
63 void initWebSocket() {
64     ws.onEvent(onEvent);
65     server.addHandler(&ws);
66 }
67
68 void setup() {
69
70     Serial.begin(115200);
71     hc05.begin(38400);
72
73     initWiFi();
74     initWebSocket();
75
76     // Start server
77     server.begin();
78 }
79

80 void loop() {
81     while(hc05.available()>0) {
82         cmd += (char)hc05.read();
83         delay(1);
84     }
85     if(cmd != "") {
86         Serial.print("Command received : ");
87         Serial.println(cmd);
88         Serial.println(cmd.length());
89         ws.textAll(cmd);
90         cmd = "";
91     }
92 }

```

3.2.1.2 Code carte d'allumage (ESP8266)

CODE GREG

3.3 Communication I2C entre la carte d'allumage et l'Arduino Nano

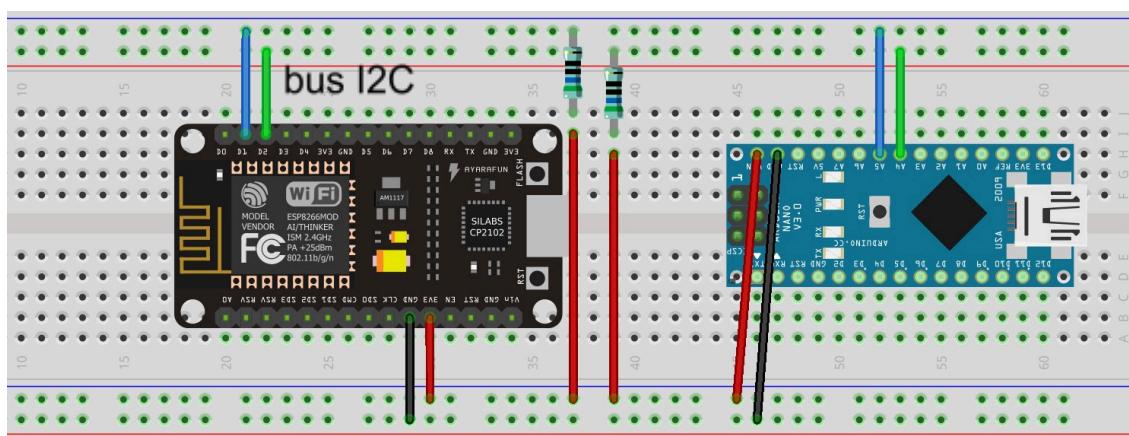
Une fois les informations d'allumage reçues par la carte de contrôle, ces dernières doivent être transmises sur le bus I2C pour allumer le bon tube d'allumage. Dans cette partie nous avons commencé par ne pas prendre en compte le wifi afin d'être déjà sur dans un premier temps que le bus I2C fonctionne bien

3.3.1 Tests de prise en main

Avant de passer au projet, nous avons d'abord essayé de prendre en main la communication I2C entre la carte d'allumage (ESP8266) et l'Arduino Nano via des tests simples.

3.3.1.1 Allume la LED interne d'une arduino Nano via le bus I2C

Pour tester notre bus I2C, on effectue un premier test avec une carte ESP8266 (Carte d'allumage, I2C Master) et une carte Arduino Nano (I2C Slave). Pour ce test, on fait clignoter la led interne à la carte Nano au moyen d'instructions I2C.



(Les résistance de l'I2C sont des 560 Ohm sur le schéma mais en réalité on utilise des 5.6k Ohm)

Nous utiliserons la bibliothèque Wire qui gère les communications I2C sur Arduino

3.3.1.1.a Code de la carte d'allumage (ESP8266)

```
#include <Wire.h>

void setup() {
    Wire.begin(); // On initialise la carte ESP8266 en mode Master I2C
}

void loop() {
    Wire.beginTransmission(1); // On envoie une instruction au Slave I2C d'adresse 1 (Arduino Nano)
    Wire.write("ON"); // L'envoi de l'instruction ON permet d'allumer la led
    Wire.endTransmission();
    delay(1000);

    Wire.beginTransmission(1); // On re-effectue la même opération pour éteindre la led
    Wire.write("OFF");
    Wire.endTransmission();
    delay(1000);
}
```

3.3.1.1.b Code de la carte Arduino Nano

```
#include <Wire.h>

void setup() {
    Wire.begin(1); // Définissez l'adresse I2C comme 1
    Wire.onReceive(receiveEvent); // Lorsqu'un message est reçu, appelez la fonction receiveEvent
    // initialize digital pin LED_BUILTIN as an output.
    pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
    delay(100);
}

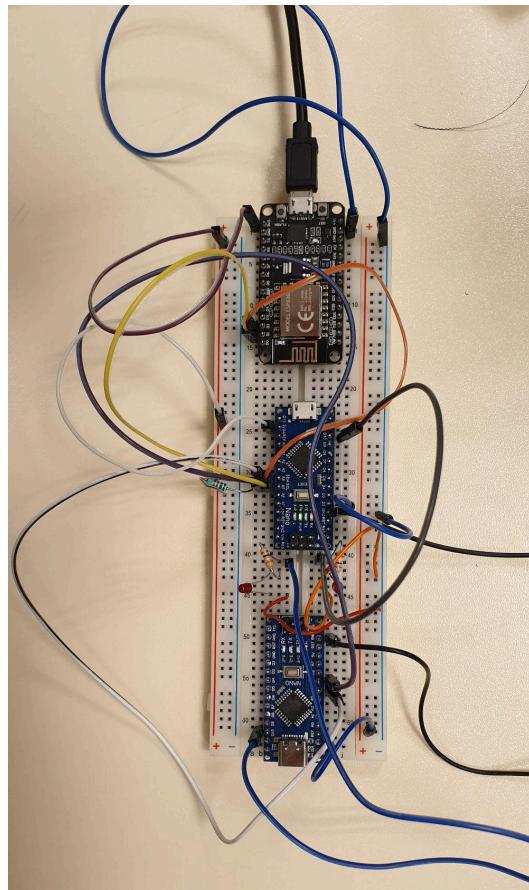
void receiveEvent() {
    String command = "";
    while (Wire.available()) {
        char c = Wire.read();
        command += c;
    }

    if (command == "ON") {
        digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level)
    } else if (command == "OFF") {
        digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making the voltage LOW
    }
}
```

3.3.1.2 Allume 2 LED externes avec 2 Arduino Nano via le bus I2C

Pour notre bus I2C, on effectue un nouveau test avec une carte ESP8266 (Carte d'allumage, I2C Master) et deux cartes Arduino Nano (I2C Slave). Pour ce test, on fait clignoter deux led externes à tour de rôle au moyen d'instructions I2C (1 led externe branché par carte arduino nano). Ainsi, nous pourrons démontrer l'efficacité du bus I2C. On appellera LED1 la led associée à la Nano du haut et LED2 pour celle associée à la Nano du bas. Cela

permet de tester différentes commandes pour des adresses I2C de communication différente.



3.3.1.2.a Code de la carte d'allumage (ESP8266)

```
#include <Wire.h>

void setup() {
    Wire.begin();
}

void loop() {
    Wire.beginTransmission(1);
    Wire.write("LED1_ON");
    Wire.endTransmission();
    delay(1000);

    Wire.beginTransmission(1);
    Wire.write("LED1_OFF");
    Wire.endTransmission();
    delay(1000);

    Wire.beginTransmission(2);
    Wire.write("LED2_ON");
    Wire.endTransmission();
    delay(1000);

    Wire.beginTransmission(2);
    Wire.write("LED2_OFF");
    Wire.endTransmission();
    delay(1000);
}
```

3.3.1.3.b Code de la première carte Arduino Nano

```
#include <Wire.h>

void setup() {
    Wire.begin(1); // Définissez l'adresse I2C comme 1
    Wire.onReceive(receiveEvent); // Lorsqu'un message est reçu, appelez la fonction receiveEvent
    // initialize digital pin D10 as an output.
    pinMode(10, OUTPUT);
}

void loop() {
    delay(100);
}

void receiveEvent() {
    String command = "";
    while (Wire.available()) {
        char c = Wire.read();
        command += c;
    }

    if (command == "LED1_ON") {
        digitalWrite(10, HIGH); // turn the LED on (HIGH is the voltage level)
    } else if (command == "LED1_OFF") {
        digitalWrite(10, LOW); // turn the LED off by making the voltage LOW
    }
}
```

3.3.1.4.c Code de la seconde carte Arduino Nano (bas)

```
#include <Wire.h>

void setup() {
    Wire.begin(2); // Définissez l'adresse I2C comme 2
    Wire.onReceive(receiveEvent); // Lorsqu'un message est reçu, appelez la fonction receiveEvent
    // initialize digital pin D10 as an output.
    pinMode(10, OUTPUT);
}

void loop() {
    delay(100);
}

void receiveEvent() {
    String command = "";
    while (Wire.available()) {
        char c = Wire.read();
        command += c;
    }

    if (command == "LED2_ON") {
        digitalWrite(10, HIGH); // turn the LED on (HIGH is the voltage level)
    } else if (command == "LED2_OFF") {
        digitalWrite(10, LOW); // turn the LED off by making the voltage LOW
    }
}
```

3.3.2 Projet

A présent on définit le code pour la carte d'allumage et les cartes arduino nano mais en respectant les trames pour utiliser cette fois-ci des relais. Pour ce faire, on définit 3 tubes par carte arduino nano et la carte d'allumage s'adressera directement à la carte arduino nano via le bus I2C pour allumer l'un des tubes. Ainsi, le code de chaque arduino est le même et les tubes sont branchés à chaque fois sur les même PIN (D10, D9 et D8).

3.3.2.1 Code carte d'allumage (ESP8266)

CODE GREG

3.3.2.2 Code des carte Arduino Nano

```
#include <Wire.h>

void setup() {
    Wire.begin(1); // Définissez l'adresse I2C comme 1
    Wire.onReceive(receiveEvent); // Lorsqu'un message est reçu, appelez la fonction receiveEvent
    // initialize digital pin D10, D9 and D8 as an output
    pinMode(10, OUTPUT); // PIN D10 tube3
    pinMode(9, OUTPUT); // PIN D9 tube2
    pinMode(8, OUTPUT); // PIN D8 tube1
}

void loop() {
    delay(100);
}

void receiveEvent() {
    String command = "";
    while (Wire.available()) {
        char c = Wire.read();
        command += c;
    }

    if (command == "tube1") {
        digitalWrite(8, HIGH); // turn the relay on (HIGH is the voltage level)
        delay(5000);
        digitalWrite(8, LOW); // turn the relay off by making the voltage LOW

    } else if (command == "tube2") {
        digitalWrite(9, HIGH); // turn the relay on (HIGH is the voltage level)
        delay(5000);
        digitalWrite(9, LOW); // turn the relay off by making the voltage LOW

    } else if (command == "tube3") {
        digitalWrite(10, HIGH); // turn the relay on (HIGH is the voltage level)
        delay(5000);
        digitalWrite(10, LOW); // turn the relay off by making the voltage LOW
    }
}
```

4. Références

[1] : ESP8266

https://www.espressif.com/sites/default/files/documentation/esp8266-technical_reference_en.pdf

[2] : Arduino Nano datasheet

<https://docs.arduino.cc/resources/datasheets/A000005-datasheet.pdf>

[3] : I2C Arduino Nano Guide

<https://docs.arduino.cc/tutorials/nano-every/i2c/>

[4] : HC-05 Bluetooth module datasheet

https://components101.com/sites/default/files/component_datasheet/HC-05%20Datasheet.pdf

[5] : HC-05 Guide

<https://arduino-france.site/bluetooth-hc-05/>

[6] : Pull up resistor value

<https://docs.particle.io/reference/device-os/api/wire-i2c/pull-up-resistors-i2c/>

[7] : AESLib encryption

<https://www.arduino.cc/reference/en/libraries/aeslib/>

5. IEEE

0. Abstract

This project aims to develop a secure system for remotely triggering fireworks using the Wi-Fi communication protocol. The motivation behind this system is to enhance safety by enabling remote ignition and providing the capability to coordinate and synchronize multiple fireworks while detecting any errors during ignition. The system comprises three main components: the control board, the ignition board, and Arduino Nano boards.

To establish communication, Wi-Fi protocol was chosen due to its adaptability, sufficient range, and ease of incorporating encryption for secure communication. The communication involves a control board acting as a relay between Bluetooth instructions from the control console and the ignition board. The ignition board receives Wi-Fi instructions from the control board and communicates with Arduino Nano boards via I2C.

The report details the selection of microcontrollers and development environments, ultimately opting for the ESP8266 card for the control and ignition boards due to its compatibility with Wi-Fi and Bluetooth modules. The communication between the control

console and the control board is established using Python for the PC application, and a data frame system is implemented to standardize communication between different hardware and programming languages.

Safety encryption is implemented using the AES library on the Arduino IDE to protect the system from cyber-attacks. The report further discusses the Wi-Fi communication between the control board and the ignition board, and the I2C communication between the ignition board and Arduino Nano boards for individual tube control.

The presented system offers a secure and synchronized approach to fireworks ignition, emphasizing safety and control. The communication protocols, encryption measures, and hardware integration contribute to a comprehensive solution for remote firework displays.

I. Introduction

We aim to develop a secure system to remotely trigger the lighting of fireworks using the Wi-Fi communication protocol. Since the lighting of fireworks presents risks for the safety of users, it is better to opt for a system that allows remote triggering, thus providing better security. In addition, our ignition system would offer the possibility to coordinate and synchronize the ignition of different fireworks, as well as detect any errors during the ignition.

To establish the communication between the ignition system and the control system, we chose to use the Wi-Fi protocol. This communication protocol is mainly used for home automation and connected objects applications. The advantage of Wi-Fi is that this protocol has a range adapted to what is required for our application (approx. 50 m outdoors), that it is easily adaptable to communicate with a microcontroller and that it is easy to incorporate encryption features to communicate securely.

Our system consists of five ignition tubes, each equipped with five electric igniters (relays), allowing individual control of rockets.

Thus, we have to program three boards: the control board, the ignition board and the Arduino Nano board. The control board acts as a relay between the instructions from the control console (Bluetooth) and the control board. As for the ignition board, it will receive instructions via the Wi-Fi protocol of the control board and will communicate with our Arduino Nano boards via an I2C communication.

II. CHOICE OF MICROCONTROLLER AND IDE

For the choice of the control and control board, we first studied the 2 most consistent tracks for our application: a programmable microcontroller with the Arduino IDE and a Silicon Labs development board adapted to Z-Wave communication (the Z-Wave Thunderboard 800 Series).

Regarding the communication between the control board and the ignition board, we started with the Z-Wave communication protocol which was integrated into the Thunderboard 800 Series card. However, given the complexity of the IDE and programming language, we finally opted for the ESP8266 card and to use the Wi-Fi protocol (integrated) for communication. We will integrate a Bluetooth module (HC-05 module) to our control board in order to receive data from our control panel.

III. Work and tests carried out

A. Bluetooth communication between the control console and the control board

1) Control console on computer

For the control console sending the necessary frames for the ignition of the different cylinders individually or in pre-programmed sequence, the choice was to develop it in Python programming language to make it a PC application.

This console is divided into two parts. The Settings section on the left to connect to the control board via Bluetooth, turn on the different cylinders, send an ignition sequence and send the sequence start instruction so that the ignition board automatically calls the different cylinders. The right part is a console consisting of a text box displaying the logs of the various messages sent as well as all the messages sent by the control board in order to monitor the progress of the various actions performed by the boards.

2) Frame formatting

In order to be able to communicate between the different hardware supports and the understanding between the different programming languages (Python and Arduino) we have chosen to define a data frame system. These frames are strings of up to 64 characters encoded in utf-8. In order to differentiate the different instructions, we have set up unique flags for each instruction.

For individual cylinder ignition, the frame is constructed as follows: the flag M to inform that it is a manual ignition of a cylinder, a letter in lower case representing the identifier of an Arduino Nano board then finally a number representing the identifier of the cylinder linked to this board. We then have a typical frame: Mb2 manual ignition of the number 2 cylinder of the Arduino Nano b.

For the sequences of ignitions, we will have two forms of frames because the HC-05 Bluetooth module can only receive 64 characters by sending. For sequences of less than 60 characters we then have the following frame type: the flag S to inform that it is a transmission of sequence, 00 the identifier of the frame composed of two digits (here 00

means that the frame is not segmented, so it is not necessary to wait for another frame to complete the data), the flag T which indicates the beginning of the “useful” data of the frame. For a segmented frame, we have a standard frame of the following shape: the flag S, the identifier of the frame (01 for the first frame, 02 for the second, etc.), the flag T. The “useful” data of these frames is then formatted using two flags. The flag C represents the ignition of a cylinder, this flag is followed by an identifier of Arduino Nano then a cylinder identifier as for the manual ignition of a cylinder. Flag D represents a delay, this flag is followed by a series of digits representing the duration of the delay in milliseconds. The flag E represents the end of the sequence. For a sequence of less than 60 characters we then have a frame of the form: S00TCb3Ca1D1000Cc2E. For a sequence of more than 60 characters we then have frames of the form: S01TCb3Ca1... / S02TD1000Cc2E.

The last flag defined is the one representing the start instruction of the ignition sequence. This flag is A.

B. Safety encryption

In order to protect the functioning of our system from a cyber-attack, it is necessary to set up a safety encryption.

To do this, we need to install AES library on Arduino IDE

C. Wi-Fi communication between the control board and the ignition board

Once the Bluetooth communication and safety encryption are established, the control board must read the received message in order to transmit it to the ignition board by Wi-Fi.

D. I2C communication between the ignition board and Arduino Nano boards

Once the ignition information is received by the control board, it must be transmitted to the I2C bus to turn on the correct ignition tube.

To do this, we connected two Arduino Nano boards a and b to the ignition board (ESP8266) by connecting the SDA and SCL ports of the ESP8266 and the Arduino Nano boards to each other via the I2C bus.

Then, we connected three ignition tubes to each Arduino Nano via electrical relays; tubes number 1, 2 and 3 being respectively connected to the Arduino Nano a on ports D8, D9 and D10, and tubes number 4, 5 and 6 being respectively connected to the Arduino Nano b on

ports D8, D9 and D10. As previously said in the Bluetooth communication part, typical frames Ma1, Ma2, Ma3, Mb1, Mb2 and Mb3 respectively correspond to the manual ignition of tubes number 1, 2, 3, 4, 5 and 6.

IV. Conclusion

The development and implementation of a secure remote ignition system for fireworks offer substantial advantages in terms of safety and coordination. By leveraging the Wi-Fi communication protocol, our system provides an adequate range for outdoor applications, approximately 50 meters, while allowing seamless adaptability for communication with microcontrollers.

The choice of the ESP8266 card for the control and ignition boards, coupled with the integration of a Bluetooth module (HC-05), enhances the versatility of the system. The utilization of Python for the control console facilitates a user-friendly interface, enabling the manual ignition of individual cylinders or the execution of pre-programmed sequences.

A robust safety encryption layer, implemented through the AES library on the Arduino IDE, safeguards the system against potential cyber threats. This encryption ensures the integrity of communication channels, mitigating risks associated with unauthorized access or tampering.

The structured communication framework, involving data frames and unique flags for instructions, streamlines interworking between different hardware supports and programming languages. This approach allows for clear and concise communication between the control console and the control board, ensuring efficient transmission of commands.

Incorporating I2C communication between the ignition board and Arduino Nano boards facilitates individual tube control, enabling precise and synchronized ignition sequences. The seamless integration of these technologies positions our system as a comprehensive solution for secure and coordinated firework displays.

As technology continues to advance, our work represents a meaningful contribution to the field of pyrotechnics, emphasizing safety, precision, and the potential for innovative applications in event coordination. The presented system aligns with contemporary trends in wireless communication and IoT applications, showcasing the adaptability and scalability of our approach.