

---

Université de Toulon

IUT de Toulon

Département Génie Electrique et Informatique Industrielle (GEII)

# SAE

## Tests Unitaires, Integration et Validation

écrit le 5 janvier 2024

par

Bruno HANNA

*Encadrant universitaire :* Stephane PIGNOL



# Table des matières

<b>1</b>	<b>Test Unitaire</b>	<b>1</b>
1.1	Communication Client LabVIEW et Serveur netcat . . . . .	1
1.1.1	Configuration du Client LabVIEW . . . . .	1
1.1.2	Configuration et Vérification du Serveur netcat . . . . .	1
1.1.3	Vérification avec Wireshark . . . . .	2
1.2	Serveur TCP : hôte ESP 8266 avec un client telnet . . . . .	4
1.2.1	Explication du Programme ESP8266 . . . . .	4
1.2.2	Utilisation d'un Client Telnet . . . . .	6
1.2.3	Test de Vérification . . . . .	6
<b>2</b>	<b>Test d'intégration</b>	<b>9</b>
2.1	Configuration du Client LabVIEW . . . . .	9
2.1.1	Description . . . . .	9
2.1.2	Interface et Configuration TCP . . . . .	9
2.2	Explication du Programme ESP8266 . . . . .	10
2.2.1	Inclusion des Bibliothèques . . . . .	10
2.2.2	Définition des Identifiants WiFi . . . . .	10
2.2.3	Initialisation du Serveur TCP . . . . .	10
2.2.4	Configuration initiale dans setup() . . . . .	10
2.2.5	Fonction handleTCPClient() . . . . .	11
2.3	Vérification Port Série de l'Arduino . . . . .	12
2.4	Vérification Wireshark . . . . .	12
2.4.1	Analyse détaillée d'un paquet . . . . .	13
<b>3</b>	<b>Test de validation</b>	<b>14</b>
3.1	Explication du Code Source . . . . .	14
3.1.1	Inclusions de Bibliothèques . . . . .	14
3.1.2	Configuration du Wi-Fi . . . . .	14
3.1.3	Initialisation des Serveurs . . . . .	15
3.1.4	Configuration des Broches . . . . .	15
3.1.5	Fonction handleRoot . . . . .	15

---

3.1.6	Conception de l'Interface du Site Web . . . . .	17
3.1.7	Traitement de la Requête PWM . . . . .	17
3.1.8	Mise à jour de la Valeur PWM . . . . .	18
3.1.9	Contrôle du Relai . . . . .	18
3.1.10	Boucle Principale (loop) . . . . .	21
3.2	Développement de l'Interface LabVIEW . . . . .	22
3.2.1	Gestion du Relai via Boucle d'Événements . . . . .	22
3.2.2	Transmission de la Valeur PWM via Boucle d'Événements . . . . .	22
3.2.3	Interface Utilisateur Graphique . . . . .	23
3.3	Test et Validation . . . . .	24
3.3.1	Vérification via le port série Arduino . . . . .	24
3.3.2	Vérification via Oscilloscope . . . . .	24
3.3.3	Vérification Wireshark . . . . .	25
3.4	Client WEB . . . . .	26
<b>A</b>	<b>Code Source Complet</b>	<b>28</b>
A.1	Interface LabView pour le Client TCP . . . . .	28
A.2	Code Arduino pour le Serveur TCP . . . . .	29
A.3	Code Arduino Server TCP et WEB . . . . .	31
A.4	LabView Test de Validation . . . . .	36

# Chapitre 1

## Test Unitaire

### 1.1 Communication Client LabVIEW et Serveur netcat

Le test vise à établir une communication TCP/IP entre un client LabVIEW et un serveur netcat.

#### 1.1.1 Configuration du Client LabVIEW

La figure montre le programme client LabVIEW qui, lors de la modification de la valeur du slider, récupère cette valeur, la convertit en chaîne de caractères, ajoute "\r" comme terminateur de message, et l'envoie au serveur netcat.

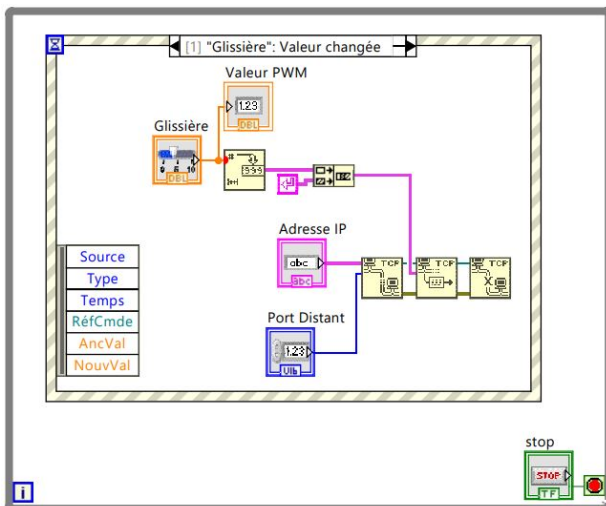


Figure 1.1 – Interface client LabVIEW

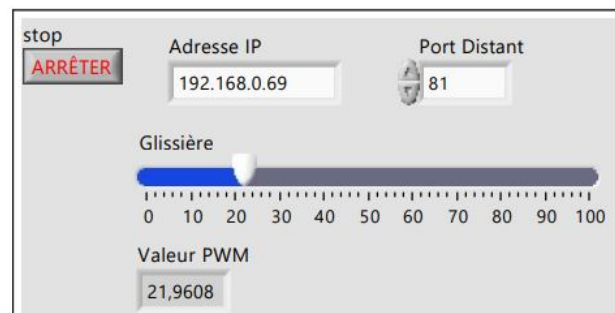


Figure 1.2 – Configuration TCP dans LabVIEW

#### 1.1.2 Configuration et Vérification du Serveur netcat

##### Acquisition de l'Adresse IP

La commande `ifconfig` a été utilisée pour obtenir l'adresse IP de la machine hôte Linux.

## Sélection du Port

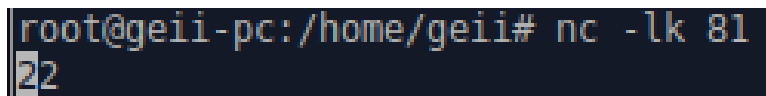
La commande `sudo nmap -p- 192.168.0.69` a permis de lister tous les ports inutilisés sur le réseau, afin de choisir un port pour le serveur netcat. Le port 81 a été sélectionné pour cette tâche.

## Démarrage du Serveur netcat

Le serveur a été lancé avec la commande `nc -lk 81`, mettant le serveur à l'écoute sur le port 81 pour les connexions entrantes.

## Réception de la Valeur par le Serveur netcat

La figure 1.3 illustre la console Linux confirmant que le serveur netcat a reçu la valeur "22" envoyée par le client LabVIEW.



```
root@geii-pc:/home/geii# nc -lk 81
22
```

Figure 1.3 – Console Linux confirmant la réception de la valeur par le serveur netcat

### 1.1.3 Vérification avec Wireshark

Pour assurer l'intégrité des données transmises du client LabVIEW au serveur netcat, l'outil Wireshark a été utilisé pour analyser les paquets réseau. Wireshark a confirmé que le message contenant la valeur "22" a été correctement formé et reçu par le serveur.

## Échange de données entre les adresses IP

La capture d'écran Wireshark suivante montre l'échange de paquets TCP entre les hôtes avec les adresses IP 192.168.0.58 et 192.168.0.69. L'explication détaillée de chaque ligne est fournie après l'image.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.0.58	192.168.0.69	TCP	66	50945 → 81 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256
2	0.000041562	192.168.0.69	192.168.0.58	TCP	66	81 → 50945 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=
3	0.000280347	192.168.0.58	192.168.0.69	TCP	60	50945 → 81 [ACK] Seq=1 Ack=1 Win=1051136 Len=0
4	0.651582806	192.168.0.58	192.168.0.69	TCP	60	50945 → 81 [PSH, ACK] Seq=1 Ack=1 Win=1051136 Len=3
5	0.651628786	192.168.0.69	192.168.0.58	TCP	54	81 → 50945 [ACK] Seq=1 Ack=4 Win=64256 Len=0
6	1.301473326	192.168.0.58	192.168.0.69	TCP	60	50945 → 81 [FIN, ACK] Seq=4 Ack=1 Win=1051136 Len=0
7	1.301563586	192.168.0.69	192.168.0.58	TCP	54	81 → 50945 [FIN, ACK] Seq=1 Ack=5 Win=64256 Len=0
8	1.301776121	192.168.0.58	192.168.0.69	TCP	60	50945 → 81 [ACK] Seq=5 Ack=2 Win=1051136 Len=0

Figure 1.4 – Échange de paquets TCP entre 192.168.0.58 et 192.168.0.69

Les échanges suivants ont été observés :

- Le paquet **No. 1** marque le début de la connexion TCP avec un drapeau [SYN] envoyé de 192.168.0.58 à 192.168.0.69 pour initialiser la synchronisation.
- Le paquet **No. 2** est la réponse de 192.168.0.69 avec un drapeau [SYN, ACK], reconnaissant la demande de synchronisation et préparant la connexion.

- Le paquet **No. 3**, envoyé par 192.168.0.58, contient le drapeau [ACK] pour confirmer la réception du paquet SYN-ACK précédent.
- Le paquet **No. 4** est une transmission de données avec le drapeau [PSH, ACK], indiquant que 192.168.0.58 pousse des données vers 192.168.0.69.
- Le paquet **No. 5** est l'accusé de réception [ACK] de 192.168.0.69, confirmant la réception des données envoyées précédemment.
- Les paquets **No. 6 et 7** montrent la fermeture de la connexion avec les drapeaux [FIN, ACK] envoyés par 192.168.0.69 et la réponse de 192.168.0.58, respectivement.
- Le paquet **No. 8** est l'accusé de réception final [ACK] de 192.168.0.69, terminant la séquence de fermeture de la connexion TCP.

### Analyse détaillée d'un paquet

0000	d8 50 e6 49 4f b8	d8 50 e6 4d d5 c5	08 00	45 00		P	I	O	P	M	E
0010	00 2b 84 d1 40 00	80 06 f4 2b c0 a8	00 3a c0 a8			+	0		+	:	
0020	00 45 c7 01 00 51	ae 88 1c 5f cf 20	70 1c	50 18		E	Q			p	P
0030	10 0a 0d 46 00 00	32 32	0d	00 00 00		F				22	

Figure 1.5 – Capture Wireshark confirmant la réception du message "22"

L'analyse octet par octet de la trame TCP capturée est présentée ci-dessous :

- Les octets d8 58 e6 49 4f 4f sont l'adresse MAC de destination.
- Les octets d8 50 e6 4d d5 c5 sont l'adresse MAC source.
- Les octets 08 00 indique un protocole IPv4.
- Les octets 70 1c renseigne sur le TTL et le protocole TCP.
- Les octets 32 32 dans les données représente la chaîne "22".

## 1.2 Serveur TCP : hôte ESP 8266 avec un client telnet

### 1.2.1 Explication du Programme ESP8266

#### Inclusion des Bibliothèques

```
1 #include <ESP8266WiFi.h>
2 #include <WiFiClient.h>
```

Cette section inclut les bibliothèques nécessaires pour manipuler les fonctionnalités WiFi de l'ESP8266. 'ESP8266WiFi.h' est utilisée pour se connecter au réseau WiFi, tandis que 'WiFiClient.h' est requise pour créer des clients TCP.

#### Définition des Identifiants WiFi

```
1 #ifndef STASSID
2 #define STASSID "iPhone"
3 #define STAPSK "framboise"
4 #endif
5
6 const char* ssid = STASSID;
7 const char* password = STAPSK;
```

Cette partie définit les identifiants du réseau WiFi : le SSID et le mot de passe. Ils sont définis comme des macros pour faciliter la modification et l'accès dans le programme.

#### Initialisation du Serveur TCP

```
1 WiFiServer tcpServer(81);
```

Crée une instance de 'WiFiServer' écoutant sur le port 81. Cela permet d'initialiser le serveur TCP qui attendra les connexions entrantes sur ce port.

#### Configuration initiale dans setup()

```
1 void setup(void) {
2   Serial.begin(115200);
3   WiFi.mode(WIFI_STA);
4   WiFi.begin(ssid, password);
5   while (WiFi.status() != WL_CONNECTED) {
6     delay(500);
7     Serial.print(".");
8   }
9   Serial.println("");
```

```
10 Serial.print("Connected to ");
11 Serial.println(ssid);
12 Serial.print("IP address: ");
13 Serial.println(WiFi.localIP());
14 tcpServer.begin();
15 Serial.println("Serveur TCP démarre");
16 }
```

Configure la communication série, initialise le mode WiFi de l'ESP8266, se connecte au réseau WiFi spécifié, et démarre le serveur TCP. Cette section contient également une boucle d'attente pour la connexion WiFi et affiche l'adresse IP une fois connectée.

### Gestion des Clients TCP dans loop()

```
1 void loop(void) {
2   handleTCPClient(); // Gerer les clients TCP
3 }
```

La fonction principale 'loop()' appelle 'handleTCPClient()', qui gère les connexions des clients TCP. Cette gestion inclut l'acceptation de nouveaux clients et la lecture des données envoyées par ces derniers.

### Fonction handleTCPClient()

La fonction `handleTCPClient()` est cruciale pour la gestion des communications TCP entre l'ESP8266 et un client TCP. Elle débute par la vérification de la présence d'un nouveau client TCP tentant de se connecter, en utilisant la commande :

```
1 WiFiClient newClient = tcpServer.available();
2 if (newClient) {
```

Si un nouveau client est détecté, le programme vérifie ensuite s'il n'y a pas déjà un client actif connecté. Cette étape est essentielle pour s'assurer que le serveur gère un seul client à la fois, pour une communication ordonnée et sans interférence :

```
1 if (!activeTcpClient || !activeTcpClient.connected()) {
2   activeTcpClient = newClient;
3   Serial.println("Nouveau client TCP connecte");
4 }
```

Après avoir établi la connexion avec le nouveau client, la fonction procède à la lecture des données envoyées. Cette lecture continue jusqu'à ce que le caractère de retour chariot ('\r') soit détecté, marquant la fin d'un message :

```
1 if (activeTcpClient.connected()) {
2   while (activeTcpClient.available()) {
```



```
3   String line = activeTcpClient.readStringUntil('\r');
4   Serial.print("Donnees TCP recues: ");
5   Serial.println(line);
```

Finalement, après avoir lu les données, la fonction déconnecte le client. Cette déconnexion est signalée dans la console série, permettant ainsi de libérer la connexion pour le prochain client :

```
1   activeTcpClient.stop();
2   Serial.println("Deconnexion de l'utilisateur TCP");
3 }
4 }
```

Cette approche garantit que chaque client est traité de manière séquentielle, permettant une gestion efficace des échanges de données sur le réseau TCP. La fonction `handleTCPClient()` illustre bien la capacité de l'ESP8266 à gérer les connexions réseau TCP, en assurant une communication fluide et ordonnée avec les clients TCP.

### 1.2.2 Utilisation d'un Client Telnet

Pour interagir avec l'ESP8266 à travers le serveur TCP mis en place, on peut utiliser un client Telnet. Telnet est un protocole réseau permettant d'effectuer une communication bidirectionnelle textuelle. Pour envoyer des caractères à l'ESP8266, la commande suivante peut être utilisée depuis un terminal ou une invite de commande :

```
1 telnet 172.20.10.4 81
```

Cette commande établit une connexion au serveur TCP de l'ESP8266, qui écoute sur l'adresse IP '172.20.10.4' et le port '81'. Une fois la connexion établie, il est possible d'envoyer des données à l'ESP8266. Par exemple, si l'on envoie la chaîne de caractères "25".

### 1.2.3 Test de Vérification

Pour valider le bon fonctionnement du programme, un test de vérification est réalisé. Ce test consiste à envoyer un paquet depuis un client Telnet vers l'ESP8266 et à vérifier la réception correcte de ce paquet.

#### Réception du Paquet sur l'Arduino

La figure 3.5 montre la console série de l'Arduino, où l'on peut observer la réception correcte du paquet envoyé par le client Telnet.

#### Vérification Wireshark

Pour vérifier l'intégrité des données transmises entre un client Telnet et un serveur TCP hébergé sur un ESP8266, Wireshark a été utilisé pour capturer et analyser les paquets réseau. Cette analyse

```

22:40:32.218 -> Connected to iPhone
22:40:32.218 -> IP address: 172.20.10.4
22:40:32.218 -> Serveur TCP démarre
22:41:14.720 -> Nouveau client TCP connecte
22:42:08.450 -> Donnees TCP recues: 25
22:42:08.450 -> Deconnexion de l'utilisateur

```

Figure 1.6 – Réception du paquet sur l'Arduino

a confirmé que le message contenant la valeur "25" a été correctement formé et reçu par le serveur.

La capture d'écran Wireshark montre l'échange de paquets TCP entre les hôtes avec les adresses IP 172.20.10.11 (client Telnet) et 172.20.10.4 (ESP8266). L'explication détaillée de chaque ligne, basée sur les nouvelles données générées, est fournie ci-dessous.

1	0.000000000	172.20.10.14	172.20.10.4	TCP	54	1820 → 81 [SYN] Seq=0 Win=64240 Len=0
2	0.000001000	172.20.10.4	172.20.10.14	TCP	54	81 → 1820 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0
3	0.000002000	172.20.10.14	172.20.10.4	TCP	54	1820 → 81 [ACK] Seq=1 Ack=1 Win=4112 Len=0
4	0.000003000	172.20.10.14	172.20.10.4	TCP	57	1820 → 81 [PSH, ACK] Seq=1 Ack=1 Win=4112 Len=3
5	0.000004000	172.20.10.4	172.20.10.14	TCP	54	1820 → 81 [ACK] Seq=1 Ack=1 Win=64240 Len=0
6	0.000005000	172.20.10.14	172.20.10.4	TCP	54	1820 → 81 [FIN, ACK] Seq=4 Ack=1 Win=4112 Len=0
7	0.000006000	172.20.10.4	172.20.10.14	TCP	54	1820 → 81 [FIN, ACK] Seq=1 Ack=2 Win=64240 Len=0
8	0.000007000	172.20.10.14	172.20.10.4	TCP	54	[TCP ACKed unseen segment] 1820 → 81 [ACK] Seq=5 Ack=2 Win=4112 Len=0

Figure 1.7 – Échange de paquets TCP entre 172.20.10.14 et 172.20.10.4

Les échanges suivants ont été observés :

- Le paquet **No. 1** marque le début de la connexion TCP avec un drapeau [SYN] envoyé de 172.20.10.14 à 172.20.10.4 pour initialiser la synchronisation.
- Le paquet **No. 2** est la réponse de 172.20.10.4 avec un drapeau [SYN, ACK], reconnaissant la demande de synchronisation et préparant la connexion.
- Le paquet **No. 3**, envoyé par 172.20.10.14, contient le drapeau [ACK] pour confirmer la réception du paquet SYN-ACK précédent.
- Le paquet **No. 4** représente une transmission de données avec le drapeau [PSH, ACK], où 172.20.10.11 envoie "25" à 172.20.10.4.
- Le paquet **No. 5** est l'accusé de réception [ACK] de 172.20.10.4, confirmant la réception des données envoyées précédemment.
- Les paquets **No. 6 et 7** illustrent la fermeture de la connexion avec les drapeaux [FIN, ACK] envoyés par 172.20.10.14 et la réponse de 172.20.10.4, respectivement.
- Le paquet **No. 8** est l'accusé de réception final [ACK] de 172.20.10.4, terminant la séquence de fermeture de la connexion TCP.

Cette séquence démontre une communication réussie et ordonnée entre le client Telnet et le serveur ESP8266, depuis l'établissement de la connexion jusqu'à sa fermeture, après l'échange de données.

## Analyse détaillée d'un paquet

0000	d4 3d 7e ff fe 01	f8 2f a8 fe fa 21	08 00	45 00	·=·.../ ...!·E·
0010	00 30 1a 2e 40 00	40 06 3c 58 ac 14	0a 0e ac 14		·0·.@·@· <X·...·
0020	0a 04 07 1c 00 51	00 00 00 01 00 00	00 01 00 00 01	50 18	·...Q·... ·...P·
0030	10 10 7d 7a 00 00	32 35	0d		··}z··25 ·

Figure 1.8 – Capture Wireshark confirmant la réception du message "25"

L'analyse octet par octet de la trame TCP capturée est présentée ci-dessous :

- Les octets d4 3d 7e ff fe 01 sont l'adresse MAC de destination.
- Les octets f8 2f a8 fe fa 21 sont l'adresse MAC source.
- Les octets 08 00 indiquent un protocole IPv4.
- Les octets 32 35 dans les données représentent la chaîne "25".

# Chapitre 2

## Test d'intégration

### 2.1 Configuration du Client LabVIEW

#### 2.1.1 Description

Le programme client LabVIEW est conçu pour interagir avec un serveur netcat. Lorsqu'un utilisateur modifie la valeur du curseur (slider), le programme exécute les actions suivantes : récupération de la valeur du curseur, conversion de cette valeur en chaîne de caractères, ajout d'un terminateur de message "\r", et envoi de cette chaîne au serveur netcat.

#### 2.1.2 Interface et Configuration TCP

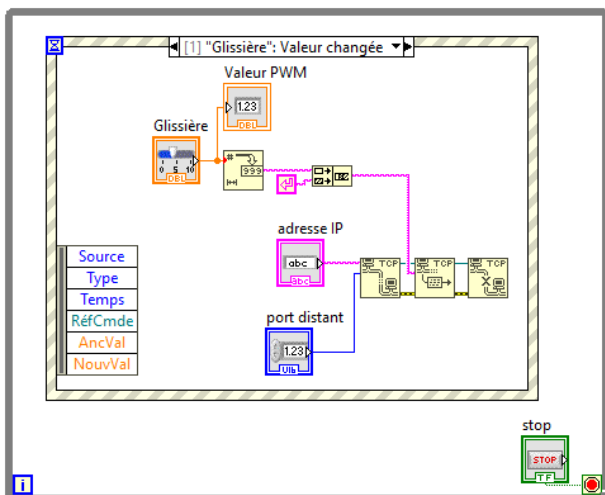


Figure 2.1 – Interface client LabVIEW

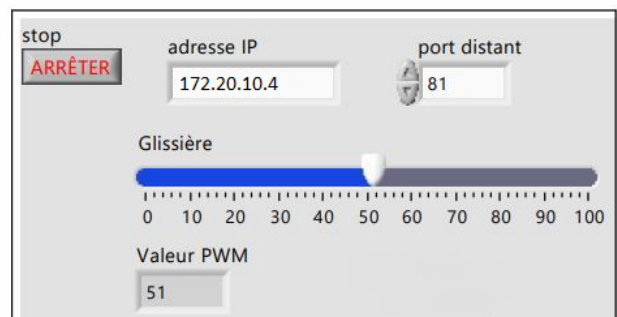


Figure 2.2 – Configuration TCP dans LabVIEW

## 2.2 Explication du Programme ESP8266

### 2.2.1 Inclusion des Bibliothèques

Pour manipuler les fonctionnalités WiFi de l'ESP8266, les bibliothèques suivantes sont incluses :

```
1 #include <ESP8266WiFi.h>
2 #include <WiFiClient.h>
```

### 2.2.2 Définition des Identifiants WiFi

Les identifiants pour la connexion au réseau WiFi sont définis comme suit, permettant une modification et un accès facilités dans tout le programme :

```
1 #ifndef STASSID
2 #define STASSID "iPhone"
3 #define STAPSK "framboise"
4 #endif
5
6 const char* ssid = STASSID;
7 const char* password = STAPSK;
```

### 2.2.3 Initialisation du Serveur TCP

Un serveur TCP écoutant sur le port 81 est initialisé pour attendre les connexions entrantes :

```
1 WiFiServer tcpServer(81);
```

### 2.2.4 Configuration initiale dans setup()

```
1 void setup(void) {
2   Serial.begin(115200);
3   WiFi.mode(WIFI_STA);
4   WiFi.begin(ssid, password);
5   while (WiFi.status() != WL_CONNECTED) {
6     delay(500);
7     Serial.print(".");
8   }
9   Serial.println("");
10  Serial.print("Connected to ");
11  Serial.println(ssid);
12  Serial.print("IP address: ");
13  Serial.println(WiFi.localIP());
```

```
14 tcpServer.begin();
15 Serial.println("Serveur TCP démarre");
16 }
```

Configure la communication série, initialise le mode WiFi de l'ESP8266, se connecte au réseau WiFi spécifié, et démarre le serveur TCP. Cette section contient également une boucle d'attente pour la connexion WiFi et affiche l'adresse IP une fois connectée.

### 2.2.5 Fonction handleTCPClient()

La fonction `handleTCPClient()` est cruciale pour la gestion des communications TCP entre l'ESP8266 et un client TCP. Elle débute par la vérification de la présence d'un nouveau client TCP tentant de se connecter, en utilisant la commande :

```
1 WiFiClient newClient = tcpServer.available();
2 if (newClient) {
```

Si un nouveau client est détecté, le programme vérifie ensuite s'il n'y a pas déjà un client actif connecté. Cette étape est essentielle pour s'assurer que le serveur gère un seul client à la fois, pour une communication ordonnée et sans interférence :

```
1 if (!activeTcpClient || !activeTcpClient.connected()) {
2   activeTcpClient = newClient;
3   Serial.println("Nouveau client TCP connecte");
4 }
```

Après avoir établi la connexion avec le nouveau client, la fonction procède à la lecture des données envoyées. Cette lecture continue jusqu'à ce que le caractère de retour chariot ('\r') soit détecté, marquant la fin d'un message :

```
1 if (activeTcpClient.connected()) {
2   while (activeTcpClient.available()) {
3     String line = activeTcpClient.readStringUntil('\r');
4     Serial.print("Donnees TCP recues: ");
5     Serial.println(line);
```

Finalement, après avoir lu les données, la fonction déconnecte le client. Cette déconnexion est signalée dans la console série, permettant ainsi de libérer la connexion pour le prochain client :

```
1   activeTcpClient.stop();
2   Serial.println("Deconnexion de l'utilisateur TCP");
3 }
4 }
```

Cette approche garantit que chaque client est traité de manière séquentielle, permettant une gestion efficace des échanges de données sur le réseau TCP. La fonction `handleTCPClient()` illustre

bien la capacité de l'ESP8266 à gérer les connexions réseau TCP, en assurant une communication fluide et ordonnée avec les clients TCP.

## 2.3 Vérification Port Série de l'Arduino

La figure montre la console série de l'Arduino, où l'on peut observer la réception correcte du paquet envoyé par le client LabView.

```
Connected to iPhone
IP address: 172.20.10.4
Serveur TCP démarre
Nouveau client TCP connecte
Donnees TCP recues: 51
Deconnexion de l'utilisateur TCP
```

Figure 2.3 – Réception du paquet sur l'Arduino

## 2.4 Vérification Wireshark

Pour vérifier l'intégrité des données transmises entre un client Telnet et un serveur TCP hébergé sur un ESP8266, Wireshark a été utilisé pour capturer et analyser les paquets réseau. Cette analyse a confirmé que les échanges de messages entre le client et le serveur ont été correctement formés et reçus.

La capture d'écran Wireshark montre l'échange de paquets TCP entre les hôtes avec les adresses IP 172.20.10.14 (client Telnet) et 172.20.10.4 (ESP8266). L'explication détaillée de chaque ligne, basée sur les nouvelles données générées, est fournie ci-dessous.

1 0.000000000	172.20.10.14	172.20.10.4	TCP	54 1820 → 81 [SYN] Seq=0 Win=64240 Len=0
2 0.000001000	172.20.10.4	172.20.10.14	TCP	54 81 → 1820 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0
3 0.000002000	172.20.10.14	172.20.10.4	TCP	54 1820 → 81 [ACK] Seq=1 Ack=1 Win=4112 Len=0
4 0.000003000	172.20.10.14	172.20.10.4	TCP	57 1820 → 81 [PSH, ACK] Seq=1 Ack=1 Win=4112 Len=3
5 0.000004000	172.20.10.4	172.20.10.14	TCP	54 1820 → 81 [ACK] Seq=1 Ack=1 Win=64240 Len=0
6 0.000005000	172.20.10.14	172.20.10.4	TCP	54 1820 → 81 [FIN, ACK] Seq=4 Ack=1 Win=4112 Len=0
7 0.000006000	172.20.10.4	172.20.10.14	TCP	54 1820 → 81 [FIN, ACK] Seq=1 Ack=2 Win=64240 Len=0
8 0.000007000	172.20.10.14	172.20.10.4	TCP	54 [TCP ACKed unseen segment] 1820 → 81 [ACK] Seq=5 Ack=2 Win=4112 Len=0

Figure 2.4 – Échange de paquets TCP entre 172.20.10.14 et 172.20.10.4

Les échanges suivants ont été observés :

- Le paquet **No. 1** marque le début de la connexion TCP avec un drapeau [SYN] envoyé de 172.20.10.14 à 172.20.10.4 pour initialiser la synchronisation.
- Le paquet **No. 2** est la réponse de 172.20.10.4 avec un drapeau [SYN, ACK], reconnaissant la demande de synchronisation et préparant la connexion.
- Le paquet **No. 3**, envoyé par 172.20.10.14, contient le drapeau [ACK] pour confirmer la réception du paquet SYN-ACK précédent.





# Chapitre 3

## Test de validation

### 3.1 Explication du Code Source

Le code source présenté est destiné à être utilisé avec le module ESP8266, cette section décrit la mise en place et le fonctionnement d'un serveur Web et d'un serveur TCP sur un microcontrôleur ESP8266, détaillant le rôle de chaque partie du code.

#### 3.1.1 Inclusions de Bibliothèques

- `#include <ESP8266WiFi.h>` : Cette bibliothèque permet de gérer la connexion Wi-Fi du module ESP8266.
- `#include <WiFiClient.h>` : Fournit des fonctions client pour se connecter à des serveurs et envoyer/recevoir des données.
- `#include <ESP8266WebServer.h>` : Utilisée pour créer un serveur Web sur le module, permettant de gérer des requêtes HTTP.
- `#include <ESP8266mDNS.h>` : Permet au module ESP8266 de résoudre les noms de domaine en adresses IP via le protocole MDNS, facilitant l'accès au module dans le réseau local sans connaître son adresse IP.

#### 3.1.2 Configuration du Wi-Fi

```
1 // Definition des identifiants WiFi
2 #ifndef STASSID
3 #define STASSID "iPhone"
4 #define STAPSK "framboise"
5 #endif
6
7 const char* ssid = STASSID; // SSID du reseau WiFi
8 const char* password = STAPSK; // Mot de passe du reseau WiFi
```

Listing 3.1 – Configuration du Wi-Fi

- Les identifiants Wi-Fi sont définis à l'aide de directives préprocesseur pour éviter de les inclure directement dans le code, améliorant ainsi la sécurité. STASSID représente le SSID (nom) du réseau Wi-Fi, et STAPSK est le mot de passe.
- Ces identifiants sont ensuite assignés aux variables `ssid` et `password`, utilisées lors de l'initialisation de la connexion Wi-Fi.

### 3.1.3 Initialisation des Serveurs

```
1 ESP8266WebServer server(80); // serveur Web sur le port 80
2 WiFiServer tcpServer(81);    // serveur TCP sur le port 81
```

Listing 3.2 – Initialisation des Serveurs

- `ESP8266WebServer server(80);` crée une instance de serveur Web qui écoute sur le port 80, le port standard pour le trafic HTTP. Cela permet au module ESP8266 de servir des pages Web.
- `WiFiServer tcpServer(81);` initialise un serveur TCP sur le port 81 pour la communication de données brutes.

### 3.1.4 Configuration des Broches

```
1 const int pwmPin = 12; // numero de la broche pour le signal PWM
2 const int relai = 13;  // numero de la broche pour le relai
```

Listing 3.3 – Configuration des Broches

- Les broches GPIO du module ESP8266 sont configurées pour différentes fonctions. `pwmPin` est défini pour envoyer un signal de modulation de largeur d'impulsion (PWM).
- `relai` sert à contrôler un relais

### 3.1.5 Fonction `handleRoot`

La fonction `handleRoot` est responsable de la gestion de la page d'accueil du serveur web embarqué. Elle offre une interface utilisateur pour le contrôle de la modulation de largeur d'impulsion (PWM) à travers un formulaire HTML.

#### Définition et Fonctionnement

```
1 void handleRoot() {
2   int lastValue = 0;
3   if (server.hasArg("pwmValue")) {
4     lastValue = server.arg("pwmValue").toInt(); // Recuperer la derniere
        valeur PWM du formulaire
```

5 }

Listing 3.4 – Fonction handleRoot - Définition

Au début de la fonction, une variable locale `lastValue` est déclarée et initialisée à 0. Cette variable est destinée à stocker la dernière valeur PWM envoyée par l'utilisateur.

La fonction vérifie ensuite si la requête HTTP contient un argument nommé "pwmValue". Si c'est le cas, cela signifie que l'utilisateur a soumis une nouvelle valeur PWM via le formulaire. La méthode `server.arg("pwmValue").toInt()` est utilisée pour extraire cette valeur de la requête et la convertir en entier, qui est ensuite stocké dans `lastValue`.

```

1  String html = "\
2  <html>\
3  <head>\
4  <title>SAE PIGNOL</title>\
5  <script>\
6  function updateTextInput(val) {\
7  document.getElementById('textInput').value=val;\
8  }\
9  </script>\
10 </head>\
11 <body>\
12 <h1>PWM Control</h1>\
13 <form action=\"/setpwm\" method=\"POST\">\
14 <input type=\"range\" name=\"pwmValue\" min=\"0\" max=\"100\" value=\"\"
15       + String(lastValue) + \"\" oninput=\"updateTextInput(this.
16         value)\" style=\"width: 20%;\">\
17 <input type=\"text\" style=\"width: 5%;\" id=\"textInput\" readonly>\
18 <input type=\"submit\" value=\"Set PWM\">\
19 </form>\
20 <h2>Bruno Hanna</h2>\
21 </body>\
22 </html>";
23 server.send(200, "text/html", html); // Envoyer la page HTML au client

```

Listing 3.5 – Fonction handleRoot - Fonctionnement

Une chaîne de caractères `html` est définie pour contenir le code HTML de la page web. Ce code inclut un formulaire permettant à l'utilisateur de soumettre une nouvelle valeur PWM. Le formulaire comprend :

1. Un élément `<input>` de type "range" pour ajuster la valeur PWM entre 0 et 100.
2. Un champ de texte `<input>` pour afficher la valeur actuelle sélectionnée par l'utilisateur, mise à jour dynamiquement grâce à un script JavaScript.

3. Un bouton de soumission pour envoyer la valeur au serveur.

Le script JavaScript incorporé, fonction `updateTextInput(val)`, met à jour le champ de texte avec la valeur actuelle du curseur à chaque fois que l'utilisateur le déplace.

Enfin, la fonction utilise `server.send(200, "text/html", html);` pour envoyer le code HTML généré au client, lui présentant ainsi l'interface de contrôle PWM.

### 3.1.6 Conception de l'Interface du Site Web

L'interface du site web a été conçue dans un souci de simplicité et d'efficacité, visant à fournir à l'utilisateur une expérience fluide et intuitive pour le contrôle à distance du système.

L'interface utilisateur du site web intègre les composants essentiels pour l'interaction avec le système embarqué, notamment :

Un **slider** permettant à l'utilisateur de sélectionner facilement une valeur spécifique pour le paramètre désiré, tel que la puissance de sortie PWM.

Un **bouton de confirmation** est présent pour soumettre la valeur choisie via le slider. Ce bouton sert de déclencheur pour l'envoi de la commande au système embarqué.

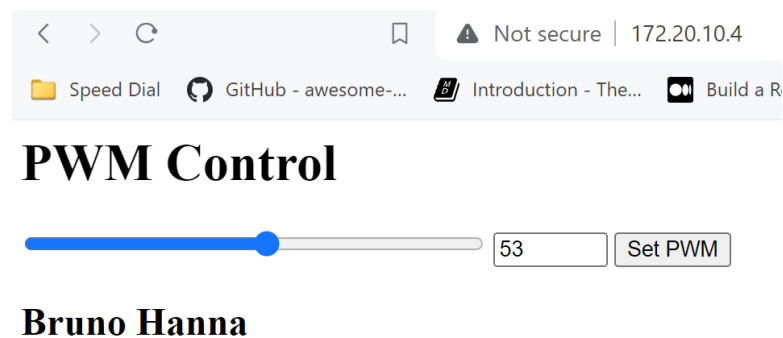


Figure 3.1 – Interface du site web pour le contrôle du système.

### 3.1.7 Traitement de la Requête PWM

```
1 void handleSetPWM() {
2   if (server.hasArg("pwmValue")) {
3     int pwmValue = server.arg("pwmValue").toInt(); // Recuperer la
    valeur PWM
4     int value = map(pwmValue, 0, 100, 0, 1023); // Mapper la valeur a
    0-1023
5     analogWrite(pwmPin, value); // Mettre a jour le
    signal PWM
6     Serial.println("Valeur PWM mise a jour: " + String(value));
7   }
8   server.sendHeader("Location", "/"); // Rediriger vers la page racine
```

```
9  server.send(303); // Envoyer une reponse de
   redirection
10 }
```

Listing 3.6 – Traitement de la requête PWM

- Cette fonction est appelée pour traiter les requêtes de mise à jour de la valeur PWM via le formulaire web. Elle vérifie d'abord si la requête contient un argument nommé "pwmValue".
- La valeur PWM soumise est récupérée, convertie en entier, puis mappée à une plage de 0 à 1023 pour s'adapter à la résolution de la sortie PWM de l'ESP8266.
- La valeur mappée est ensuite utilisée pour mettre à jour le signal PWM sur la broche spécifiée, et un message est envoyé sur le port série pour indiquer la mise à jour.
- Enfin, la fonction redirige le client vers la page d'accueil, indiquant que la mise à jour a été effectuée.

### 3.1.8 Mise à jour de la Valeur PWM

```
1 void updatePWM(int value) {
2   if (value >= 0 && value <= 100) {
3     int pwmValue = map(value, 0, 100, 0, 1023);
4     analogWrite(pwmPin, pwmValue);
5     Serial.println("Valeur PWM mise a jour: " + String(value));
6   }
7 }
```

Listing 3.7 – Mise à jour de la valeur PWM

- La fonction updatePWM prend une valeur entière en argument et ajuste le signal PWM en conséquence, après avoir mappé cette valeur à la plage de résolution de l'ESP8266.
- Elle assure que la valeur d'entrée est dans la plage permise (0 à 100) avant de procéder à la mise à jour, garantissant ainsi que les signaux générés restent dans des limites sûres.

### 3.1.9 Contrôle du Relai

```
1 void updateRelai(bool value) {
2   digitalWrite(relai, value);
3   Serial.println(value ? "Relai Active" : "Relai Desactive");
4 }
```

Listing 3.8 – Contrôle du relai

- La fonction updateRelai contrôle l'état d'un relai connecté à une des broches de l'ESP8266, en activant ou désactivant le relai en fonction de l'argument booléen fourni.

- Un message est envoyé sur le port série pour indiquer l'état du relai, fournissant un feedback instantané sur l'opération effectuée.

#### Gestion des Clients TCP

La fonction `handleTCPClient` joue un rôle central dans la gestion des connexions TCP, en permettant au module ESP8266 de répondre aux requêtes des clients TCP.

```

1 void handleTCPClient() {
2   static WiFiClient activeTcpClient; // Client TCP pour une connexion
   active
3
4   WiFiClient newClient = tcpServer.available();
5   if (newClient) {
6     if (!activeTcpClient || !activeTcpClient.connected()) {
7       activeTcpClient = newClient; // Accepter le nouveau client
8       Serial.println("Nouveau client TCP connecte");
9     }
10  }
11
12  if (activeTcpClient.connected()) {
13    while (activeTcpClient.available()) {
14      String line = activeTcpClient.readStringUntil('\r'); // Lire la
        ligne
15      Serial.print("Donnees TCP recues: ");
16      Serial.println(line);
17      activeTcpClient.stop();
18      Serial.println("Deconnexion de l'utilisateur TCP");
19
20      for (int i = 0; i < line.length(); i++) {
21        char ch = line.charAt(i);
22        if (ch == 'N') {
23          updateRelai(1); // Activer le relai
24        } else if (ch == 'F') {
25          updateRelai(0); // Desactiver le relai
26        } else if (isdigit(ch)) {
27          String numStr = "";
28          while (i < line.length() && isdigit(line.charAt(i))) {
29            numStr += line.charAt(i++);
30          }
31          int pwmValue = numStr.toInt(); // Convertir en entier
32          updatePWM(pwmValue);           // Mettre a jour la PWM
33          break; // Sortie apres traitement
34        }
      }
    }
  }
}

```

```

35     }
36   }
37 }
38 }

```

Listing 3.9 – Gestion des clients TCP

- La fonction commence par vérifier si un nouveau client TCP tente de se connecter. Si c'est le cas, et s'il n'y a pas déjà un client actif ou si le client actif est déconnecté, le nouveau client est accepté et sauvegardé comme le client actif.
- Elle lit ensuite les données envoyées par le client actif, ligne par ligne, jusqu'à rencontrer un retour chariot ('\r'). Chaque ligne reçue est traitée pour déterminer la commande envoyée.
- Les commandes pour activer ou désactiver le relai sont identifiées par les caractères 'N' et 'F', respectivement. Si la ligne contient un nombre, celui-ci est interprété comme une valeur PWM à appliquer.
- Après le traitement des données, la connexion avec le client TCP est fermée, et un message de déconnexion est envoyé sur le port série.

#### Fonction de Configuration (setup)

La fonction setup est exécutée une seule fois au démarrage du programme. Elle est utilisée pour initialiser les paramètres, configurer les broches et établir la connexion réseau.

```

1 void setup(void) {
2   pinMode(pwmPin, OUTPUT);    // Configurer la broche PWM en sortie
3   pinMode(relai, OUTPUT);     // Configurer la broche RELAI en sortie
4   analogWrite(pwmPin, 0);     // Initialiser la valeur PWM a 0
5   analogWriteResolution(10);  // Resolution du PWM sur 10 bits
6   analogWriteFreq(30000);     // Frequence du PWM a 30kHz
7
8   Serial.begin(115200);
9   WiFi.mode(WIFI_STA);
10  WiFi.begin(ssid, password); // Connexion au reseau WiFi
11
12  // Attente de la connexion WiFi
13  while (WiFi.status() != WL_CONNECTED) {
14    delay(500);
15    Serial.print(".");
16  }
17  Serial.println("");
18  Serial.print("Connected to ");
19  Serial.println(ssid);
20  Serial.print("IP address: ");
21  Serial.println(WiFi.localIP());

```

```
22
23 if (MDNS.begin("ouecgreg")) { Serial.println("MDNS ouecgreg.local
    disponible"); }
24
25 // Configuration des routes pour le serveur Web
26 server.on("/", handleRoot);
27 server.on("/setpwm", handleSetPWM);
28 server.onNotFound([]() {
29     server.send(404, "text/plain", "Not found");
30 });
31
32 server.begin(); // Demarrage du serveur Web
33 tcpServer.begin(); // Demarrage du serveur TCP
34
35 Serial.println("Serveurs Web et TCP démarres");
36 }
```

Listing 3.10 – Configuration initiale du système

- Les broches pour le signal PWM et le relai sont configurées en sortie.
- La résolution du PWM est définie sur 10 bits, offrant 1024 niveaux de précision pour le signal de sortie, et la fréquence du PWM est réglée à 30 kHz, comme demandés dans le cahier des charges.
- La connexion au réseau WiFi est établie, et l'adresse IP du module est affichée une fois connecté.
- Les routes pour le serveur web sont configurées pour répondre aux requêtes HTTP spécifiques.
- Les serveurs Web et TCP sont démarrés, permettant au module de traiter les requêtes entrantes.

### 3.1.10 Boucle Principale (loop)

La boucle loop est exécutée en continu après la fonction setup. Elle permet de gérer les interactions réseau et de répondre aux commandes des clients.

```
1 void loop(void) {
2     server.handleClient(); // Gerer les clients Web
3     MDNS.update();        // Mettre a jour le MDNS
4     handleTCPClient();    // Gerer les clients TCP
5 }
```

Listing 3.11 – Boucle principale du programme

- La fonction `server.handleClient()` gère les requêtes HTTP reçues par le serveur web.



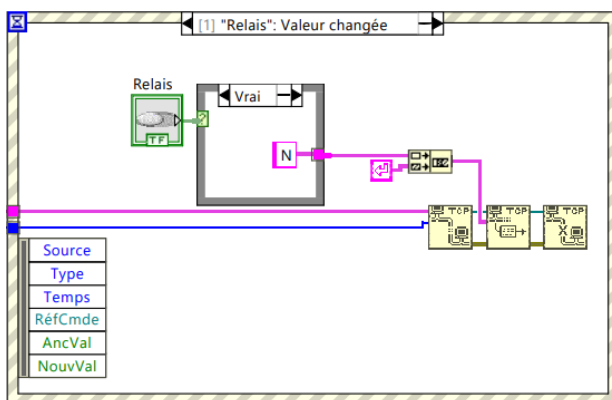
- `MDNS.update()` permet au module de répondre aux requêtes MDNS, facilitant l'accès au serveur via un nom de domaine local plutôt qu'une adresse IP.
- `handleTCPClient()` prend en charge les connexions TCP, permettant au module de recevoir et de traiter des commandes spécifiques à travers une connexion TCP.

## 3.2 Développement de l'Interface LabVIEW

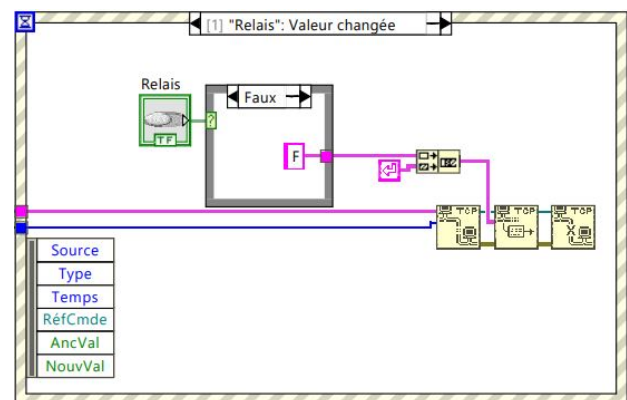
L'interface développée sous LabVIEW s'appuie sur une architecture robuste centrée autour des boucles d'événements, permettant une interaction efficace et dynamique avec le système embarqué via une connexion TCP. Cette section décrit en détail les composants clés de l'interface, en soulignant leur fonctionnement et leur intégration au sein de l'application.

### 3.2.1 Gestion du Relai via Boucle d'Événements

La première boucle d'événements est dédiée au contrôle du relai. Elle réagit aux modifications de l'état du relai en envoyant les caractères "N" pour activer (oN) ou "F" pour désactiver (oFf) le relai, suivi d'un caractère de retour chariot ("`\r`") signalant la fin de la commande à l'Arduino. Cette approche garantit que la commande est correctement reçue et traitée par le système embarqué. L'exécution de cette boucle est conditionnée par un changement d'état du relai, ce qui optimise l'efficacité de l'interface en évitant les itérations inutiles. La logique conditionnelle intégrée vérifie l'état actuel du relai et envoie la commande appropriée en conséquence.



(a) Relais Activé



(b) Relais Désactivé

### 3.2.2 Transmission de la Valeur PWM via Boucle d'Événements

Une seconde boucle d'événements prend en charge la transmission de la valeur PWM. Lorsque la valeur du slider est ajustée, cette boucle envoie la valeur, préalablement convertie de double à chaîne de caractères, au système embarqué. Comme pour la commande du relai, un caractère de retour chariot ("`\r`") est ajouté à la fin de la commande pour indiquer la fin de l'envoi. Cette méthode assure une communication précise et réactive du niveau PWM souhaité, dans la plage de 0 à 100.

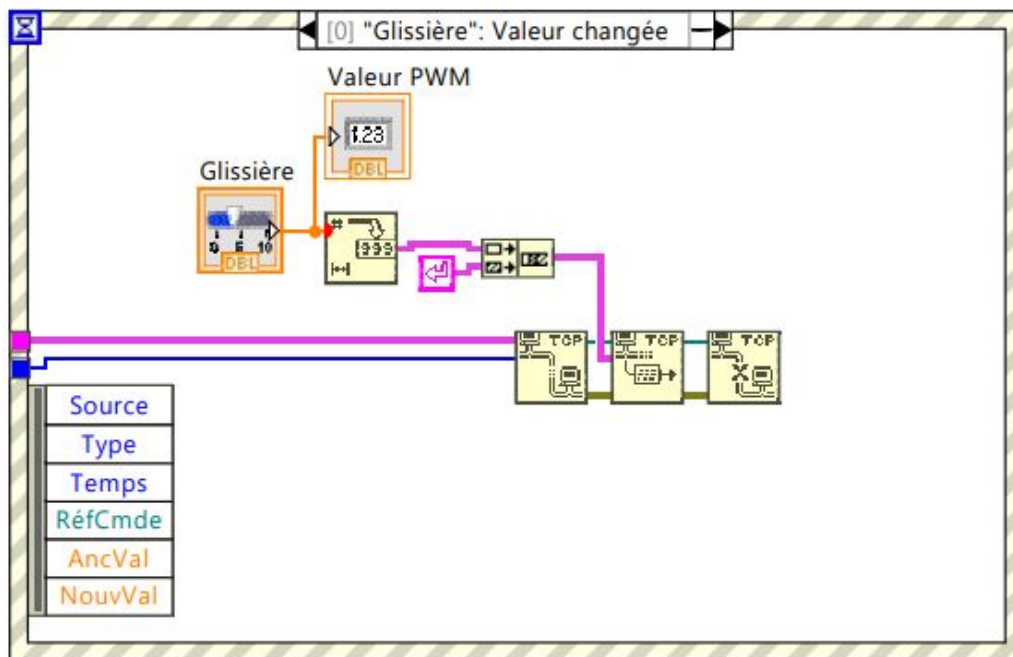


Figure 3.3 – Relais Activé

### 3.2.3 Interface Utilisateur Graphique

L'interface utilisateur graphique constitue le point d'interaction principal avec l'utilisateur. Elle est conçue pour offrir une expérience utilisateur intuitive et efficace, permettant la configuration de l'adresse IP et du port du serveur TCP, le contrôle de la valeur PWM via un slider, ainsi que la gestion de l'état du relai à l'aide d'un bouton à deux positions. Cette interface rassemble toutes les fonctionnalités nécessaires pour une interaction complète avec le système embarqué, assurant une manipulation aisée des paramètres de contrôle.

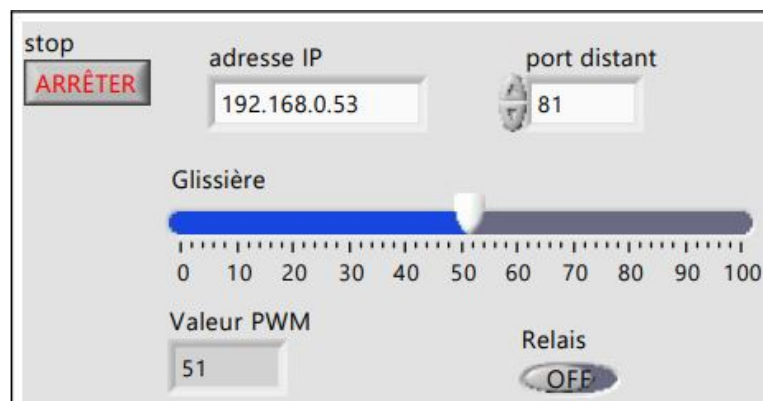


Figure 3.4 – Interface Utilisateur Graphique

Cette documentation fournit une vue d'ensemble approfondie de l'interface LabVIEW, depuis la gestion bas niveau des commandes jusqu'à l'interaction utilisateur de haut niveau, illustrant l'efficacité et la flexibilité de l'approche adoptée pour le contrôle et la supervision de systèmes embarqués.

## 3.3 Test et Validation

La validation du système s'effectue par l'observation de la sortie PWM à l'aide d'un oscilloscope et par l'analyse de trames avec Wireshark. Cette double vérification permet d'examiner la fréquence, l'amplitude et le rapport cyclique du signal PWM, ainsi que l'intégrité et la structure des données transmises sur le réseau.

### 3.3.1 Vérification via le port série Arduino

La figure 3.5 montre la réception d'une valeur de 51% sur l'Arduino, indiquant que les serveurs TCP et Web ont été démarrés avec succès et que la communication de test est fonctionnelle.

```
Connected to iPhone
IP address: 172.20.10.4
Serveur TCP démarre
Nouveau client TCP connecte
Donnees TCP recues: 51
Deconnexion de l'utilisateur TCP
```

Figure 3.5 – Réception du paquet sur l'Arduino

La vérification se poursuit avec l'oscilloscope pour confirmer la précision de la sortie PWM.

### 3.3.2 Vérification via Oscilloscope

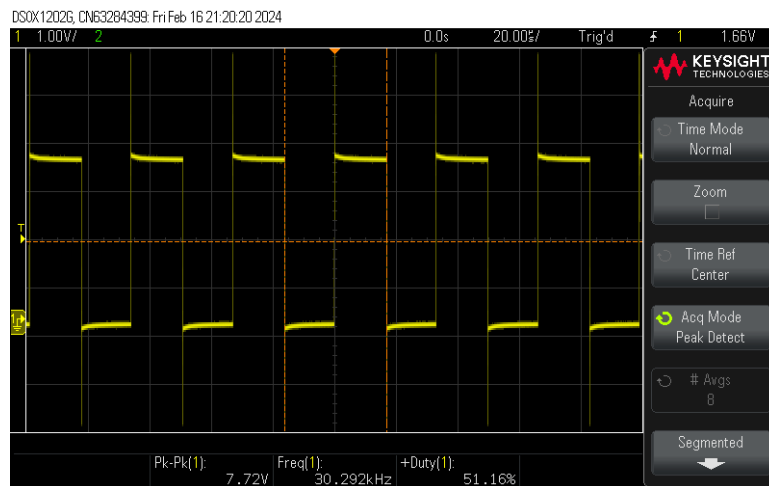


Figure 3.6 – Visualisation du signal PWM sur l'oscilloscope.

La procédure de test est la suivante :

1. **Configuration de l'Oscilloscope** : Réglage des paramètres pour une visualisation optimale.
2. **Génération du Signal** : Activation du système pour produire un signal PWM à un rapport cyclique spécifique.

3. **Analyse** : Mesure du signal sur l'oscilloscope pour vérifier la conformité avec les spécifications.
4. **Validation** : Comparaison des résultats avec les objectifs de performance.

La figure 3.6 confirme un rapport cyclique de 51% et une fréquence de 30kHz, ce qui est conforme aux spécifications.

L'analyse de la communication réseau via Wireshark est l'étape suivante pour assurer le bon fonctionnement de la transmission des données.

### 3.3.3 Vérification Wireshark

#### Client TCP

Pour vérifier l'intégrité des données transmises entre un client Telnet et un serveur TCP hébergé sur un ESP8266, Wireshark a été utilisé pour capturer et analyser les paquets réseau. Cette analyse a confirmé que les échanges de messages entre le client et le serveur ont été correctement formés et reçus.

La capture d'écran Wireshark montre l'échange de paquets TCP entre les hôtes avec les adresses IP 172.20.10.14 (client LabView) et 172.20.10.4 (ESP8266). L'explication détaillée de chaque ligne, basée sur les nouvelles données générées, est fournie ci-dessous.

1	0.000000000	172.20.10.14	172.20.10.4	TCP	54	1820 → 81 [SYN] Seq=0 Win=64240 Len=0
2	0.000001000	172.20.10.4	172.20.10.14	TCP	54	81 → 1820 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0
3	0.000002000	172.20.10.14	172.20.10.4	TCP	54	1820 → 81 [ACK] Seq=1 Ack=1 Win=4112 Len=0
4	0.000003000	172.20.10.14	172.20.10.4	TCP	57	1820 → 81 [PSH, ACK] Seq=1 Ack=1 Win=4112 Len=3
5	0.000004000	172.20.10.4	172.20.10.14	TCP	54	1820 → 81 [ACK] Seq=1 Ack=1 Win=64240 Len=0
6	0.000005000	172.20.10.14	172.20.10.4	TCP	54	1820 → 81 [FIN, ACK] Seq=4 Ack=1 Win=4112 Len=0
7	0.000006000	172.20.10.4	172.20.10.14	TCP	54	1820 → 81 [FIN, ACK] Seq=1 Ack=2 Win=64240 Len=0
8	0.000007000	172.20.10.14	172.20.10.4	TCP	54	[TCP ACKed unseen segment] 1820 → 81 [ACK] Seq=5 Ack=2 Win=4112 Len=0

Figure 3.7 – Échange de paquets TCP entre 172.20.10.14 et 172.20.10.4

Les échanges suivants ont été observés :

- Le paquet **No. 1** marque le début de la connexion TCP avec un drapeau [SYN] envoyé de 172.20.10.14 à 172.20.10.4 pour initialiser la synchronisation.
- Le paquet **No. 2** est la réponse de 172.20.10.4 avec un drapeau [SYN, ACK], reconnaissant la demande de synchronisation et préparant la connexion.
- Le paquet **No. 3**, envoyé par 172.20.10.14, contient le drapeau [ACK] pour confirmer la réception du paquet SYN-ACK précédent.
- Le paquet **No. 4** représente une transmission de données avec le drapeau [PSH, ACK], où 172.20.10.14 envoie "51" à 172.20.10.4.
- Le paquet **No. 5** est l'accusé de réception [ACK] de 172.20.10.4, confirmant la réception des données envoyées précédemment.
- Les paquets **No. 6 et 7** illustrent la fermeture de la connexion avec les drapeaux [FIN, ACK] envoyés par 172.20.10.14 et la réponse de 172.20.10.4, respectivement.

- Le paquet **No. 8** est l'accusé de réception final [ACK] de 172.20.10.4, terminant la séquence de fermeture de la connexion TCP.

Cette séquence démontre une communication réussie et ordonnée entre le client LabView et le serveur ESP8266, depuis l'établissement de la connexion jusqu'à sa fermeture, après l'échange de données.

### Analyse détaillée d'un paquet

0000	d4 3d 7e ff fe 01 f8 2f a8 fe fa 21 08 00	45 00	..~.../...!..E.
0010	00 30 1a 2e 40 00 40 06 3c 58 ac 14 0a 0e ac 14		..0..@..@<X.....
0020	0a 04 07 1c 00 51 00 00 00 01 00 00 01 50 18		.....Q.....P.
0030	10 10 7d 7a 00 00 35 31 0d		..}z..51.

Figure 3.8 – Capture Wireshark confirmant la réception de données

L'analyse octet par octet de la trame TCP capturée est présentée ci-dessous :

- Les octets d4 3d 7e ff fe 01 sont l'adresse MAC de destination.
- Les octets f8 2f a8 fe fa 21 sont l'adresse MAC source.
- Les octets 08 00 indiquent un protocole IPv4.
- Les octets 35 31 dans les données représentent la chaîne "51".

## 3.4 Client WEB

La capture d'écran Wireshark montre l'échange de paquets WEB entre les hôtes avec les adresses IP 192.168.0.53 (client WEB) et 192.168.0.75 (ESP8266). L'explication détaillée de chaque ligne, basée sur les nouvelles données générées, est fournie ci-dessous.

1	0.000000000	192.168.0.75	192.168.0.53	TCP	74	35404 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3161497774 TSecr=0 WS=128
2	0.003945298	192.168.0.53	192.168.0.75	TCP	62	80 → 35404 [SYN, ACK] Seq=0 Ack=1 Win=2144 Len=0 MSS=536 SACK_PERM=1
3	0.004087993	192.168.0.75	192.168.0.53	TCP	54	35404 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
4	0.004258964	192.168.0.75	192.168.0.53	HTTP	586	POST /setpwm HTTP/1.1 (application/x-www-form-urlencoded)
5	0.020962260	192.168.0.53	192.168.0.75	HTTP	178	HTTP/1.1 303 See Other
6	0.021009085	192.168.0.75	192.168.0.53	TCP	54	35404 → 80 [ACK] Seq=513 Ack=125 Win=64116 Len=0
7	0.037192605	192.168.0.75	192.168.0.53	HTTP	450	GET / HTTP/1.1
8	0.047718443	192.168.0.53	192.168.0.75	TCP	60	80 → 35404 [ACK] Seq=125 Ack=909 Win=1772 Len=0
9	0.050258685	192.168.0.53	192.168.0.75	TCP	169	80 → 35404 [PSH, ACK] Seq=125 Ack=909 Win=1772 Len=115 [TCP segment of a reassembled PDU]
10	0.091530860	192.168.0.75	192.168.0.53	TCP	54	35404 → 80 [ACK] Seq=909 Ack=240 Win=64001 Len=0
11	0.096795411	192.168.0.53	192.168.0.75	HTTP	552	HTTP/1.1 200 OK (text/html)
12	0.096813502	192.168.0.75	192.168.0.53	TCP	54	35404 → 80 [ACK] Seq=909 Ack=738 Win=64001 Len=0
13	2.052705242	192.168.0.53	192.168.0.75	TCP	60	80 → 35404 [FIN, ACK] Seq=738 Ack=909 Win=1772 Len=0
14	2.052952247	192.168.0.75	192.168.0.53	TCP	54	35404 → 80 [FIN, ACK] Seq=909 Ack=739 Win=64001 Len=0
15	2.056795176	192.168.0.53	192.168.0.75	TCP	60	80 → 35404 [ACK] Seq=739 Ack=910 Win=1771 Len=0

Figure 3.9 – Échange de paquets TCP entre 192.168.0.53 et 192.168.0.75

Les échanges suivants ont été observés

- **Établissement de connexion TCP** : Un paquet avec le flag [SYN] est envoyé de 192.168.0.53 à 192.168.0.75 sur le port 80 pour initier une nouvelle connexion TCP.
- **Acquittement de connexion TCP (SYN-ACK)** : L'hôte de destination répond avec un paquet [SYN, ACK], indiquant que la demande de connexion a été acceptée.
- **Confirmation de connexion TCP (ACK)** : L'hôte source envoie un paquet [ACK] pour confirmer l'établissement de la connexion TCP.

- **Requête HTTP GET** : L'hôte source envoie une requête HTTP GET, demandant des données au serveur.
- **Continuation de la requête HTTP** : Suite de la requête HTTP GET envoyée au serveur, potentiellement en raison de la segmentation des paquets.
- **Transmission de données TCP** : Échange de paquets TCP transportant les données de la session en cours.
- **Transmission de données TCP** : Suite de l'échange de données entre les deux hôtes.
- **Transmission de données TCP** : Paquet TCP portant des données ou des segments d'une session HTTP.
- **Transmission de données TCP** : Échange de données continu, possiblement associé à la réponse HTTP du serveur.
- **Transmission de données TCP** : Derniers segments de données de la session TCP en cours.
- **Réponse HTTP 200 OK** : Le serveur répond avec une réponse HTTP 200 OK, indiquant que la requête a été traitée avec succès.
- **Confirmation de fermeture de connexion TCP** : Un paquet [FIN, ACK] est envoyé pour initier la fermeture de la connexion TCP.
- **Acquittement de fermeture de connexion TCP** : L'hôte de destination envoie un paquet [ACK] pour reconnaître la demande de fermeture de la connexion.
- **Terminaison de connexion TCP** : L'hôte de destination envoie un paquet [FIN, ACK] pour confirmer la fermeture de son côté de la connexion.
- **Confirmation finale de terminaison TCP** : L'hôte source envoie un dernier paquet [ACK] pour achever la terminaison de la connexion TCP.

### Analyse détaillée d'un paquet

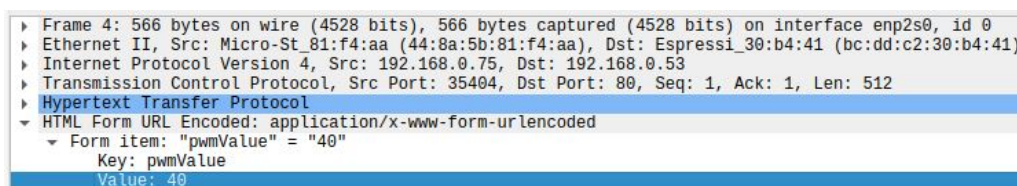


Figure 3.10 – Capture Wireshark confirmant la réception de données

La taille accrue des paquets dans les interactions web par rapport aux paquets TCP s'explique par les en-têtes HTTP supplémentaires, le contenu du corps des messages, ainsi que les données de sécurité telles que celles introduites par TLS/SSL pour les connexions HTTPS.

# Annexe A

## Code Source Complet

Les annexes suivantes contiennent les listes complètes des codes sources utilisés dans le cadre de ce projet. Ces codes représentent les composants essentiels du système et illustrent les implémentations spécifiques des fonctionnalités discutées dans les chapitres précédents.

### A.1 Interface LabView pour le Client TCP

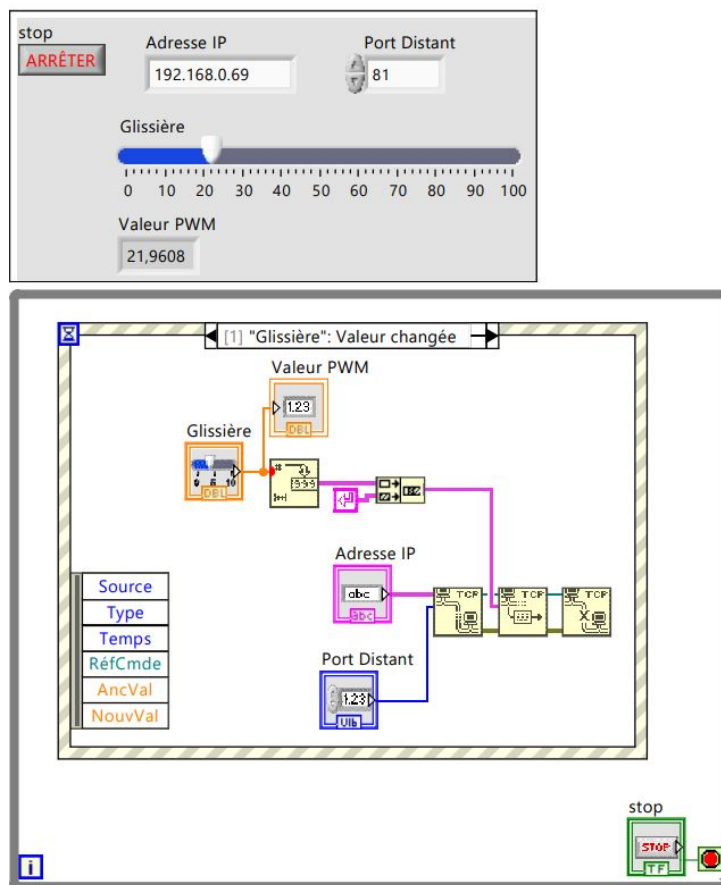


Figure A.1 – Capture Du Programme LabView



## A.2 Code Arduino pour le Serveur TCP

```
1 #include <ESP8266WiFi.h>
2 #include <WiFiClient.h>
3
4 // Definition des identifiants WiFi
5 #ifndef STASSID
6 #define STASSID "tp5&6"
7 #define STAPSK "geii2021"
8 #endif
9
10 const char* ssid = STASSID;
11 const char* password = STAPSK;
12
13 WiFiServer tcpServer(81); // Creer une instance du serveur TCP sur le
    port 81
14
15 void setup(void) {
16     Serial.begin(115200);
17     WiFi.mode(WIFI_STA);
18     WiFi.begin(ssid, password); // Se connecter au reseau WiFi
19
20     // Attendre la connexion WiFi
21     while (WiFi.status() != WL_CONNECTED) {
22         delay(500);
23         Serial.print(".");
24     }
25
26     // Afficher l'adresse IP une fois connecte
27     Serial.println("");
28     Serial.print("Connected to ");
29     Serial.println(ssid);
30     Serial.print("IP address: ");
31     Serial.println(WiFi.localIP());
32
33     // Demarrer le serveur TCP
34     tcpServer.begin();
35     Serial.println("Serveur TCP démarre");
36 }
37
38 // Fonction pour gerer les clients TCP
```



```
39 void handleTCPClient() {
40     static WiFiClient activeTcpClient; // Client TCP pour maintenir une
        connexion active
41
42     // Verifier si un nouveau client est disponible et si aucun client
        actif n'est deja connecte
43     WiFiClient newClient = tcpServer.available();
44     if (newClient) {
45         if (!activeTcpClient || !activeTcpClient.connected()) {
46             activeTcpClient = newClient; // Accepter le nouveau client
47             Serial.println("Nouveau client TCP connecte");
48         }
49     }
50
51     // Traiter les donnees si le client est connecte
52     if (activeTcpClient.connected()) {
53         while (activeTcpClient.available()) {
54             String line = activeTcpClient.readStringUntil('\r'); // Lire la
                ligne recue
55             Serial.print("Donnees TCP recues: ");
56             Serial.println(line);
57             activeTcpClient.stop();
58             Serial.print("Deconnexion de l'utilisateur TCP");
59         }
60     }
61 }
62
63 void loop(void) {
64     handleTCPClient(); // Gerer les clients TCP
65 }
```

Listing A.1 – Code Arduino pour le Serveur TCP

## A.3 Code Arduino Server TCP et WEB

```
1 #include <ESP8266WiFi.h>
2 #include <WiFiClient.h>
3 #include <ESP8266WebServer.h>
4 #include <ESP8266mDNS.h>
5
6 // Definir les identifiants WiFi
7 #ifndef STASSID
8 #define STASSID "tp5&6"
9 #define STAPSK "geii2021"
10 #endif
11
12 const char* ssid = STASSID;
13 const char* password = STAPSK;
14
15 ESP8266WebServer server(80); // Creer une instance du serveur Web sur le
    port 80
16 WiFiServer tcpServer(81);    // Creer une instance du serveur TCP sur le
    port 81
17
18 const int pwmPin = 12; // Definir le numero de la broche utilisee pour
    le signal PWM
19 const int relai = 13;    // Definir le numero de la broche utilisee pour
    le relai
20
21 void handleRoot() {
22     int lastValue = 0;
23     if (server.hasArg("pwmValue")) {
24         lastValue = server.arg("pwmValue").toInt(); // Recuperer la derniere
            valeur PWM du formulaire
25     }
26
27     String html = "\
28 <html>\
29 <head>\
30 <title>SAE PIGNOL</title>\
31 <script>\
32 function updateTextInput(val) {\
33 document.getElementById('textInput').value=val;\
34 }\
```

```

35 </script>\
36 </head>\
37 <body>\
38 <h1>PWM Control</h1>\
39 <form action=\"/setpwm\" method=\"POST\">\
40 <input type=\"range\" name=\"pwmValue\" min=\"0\" max=\"100\" value=\"\"
41       + String(lastValue) + \"\" oninput=\"updateTextInput(this.
42       value)\" style=\"width: 20%;\">\
43 <input type=\"text\" style=\"width: 5%;\" id=\"textInput\" readonly>\
44 <input type=\"submit\" value=\"Set PWM\">\
45 </form>\
46 <h2>Bruno Hanna</h2>\
47 </body>\
48 </html>";
49 server.send(200, "text/html", html); // Envoyer la page HTML au client
50 }
51 void handleSetPWM() {
52   if (server.hasArg("pwmValue")) {
53     int pwmValue = server.arg("pwmValue").toInt(); // Recuperer la
54     valeur PWM du formulaire
55     int value = map(pwmValue, 0, 100, 0, 1023); // Mapper la
56     valeur de 0-100 a 0-1023
57     analogWrite(pwmPin, value); // Mettre a jour le
58     signal PWM
59     Serial.println("Valeur PWM mise a jour: " + String(value) + " ( " +
60     String(pwmValue) + " )");
61   }
62   server.sendHeader("Location", "/"); // Rediriger vers la page racine
63   server.send(303); // Envoyer une reponse de
64   redirection
65 }
66 void updatePWM(int value) {
67   if (value >= 0 && value <= 100) {
68     int pwmValue = map(value, 0, 100, 0, 1023); // Mapper la valeur de
69     0-100 a 0-1023
70     analogWrite(pwmPin, pwmValue); // Mettre a jour le
71     signal PWM
72     Serial.println("Valeur PWM mise a jour: " + String(value) + " ( " +
73     String(pwmValue) + " )");

```

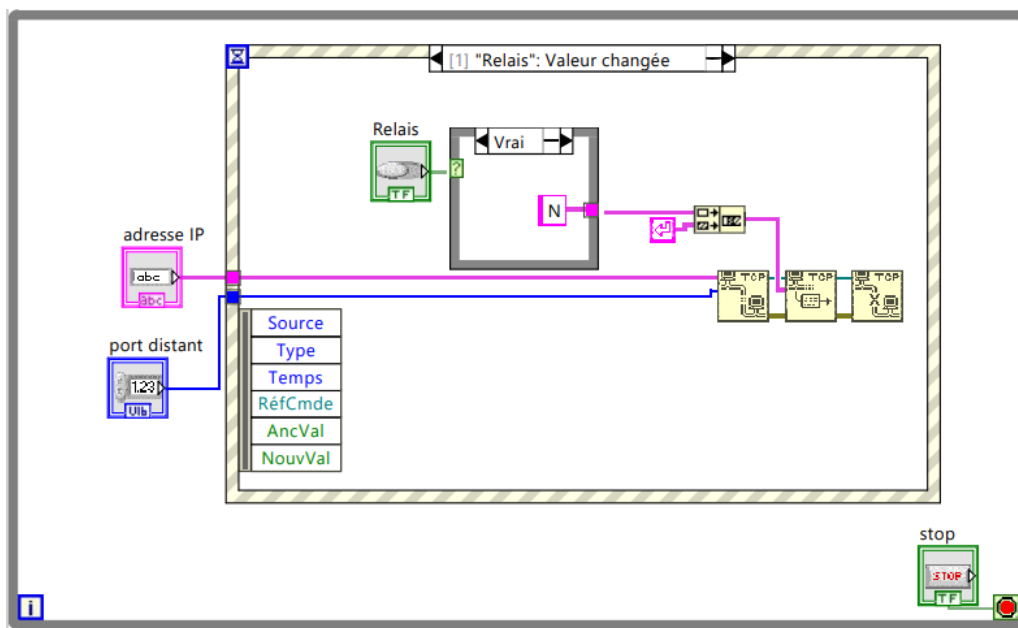
```
67 }
68 }
69
70 void updateRelai(bool value) {
71     if (value) {
72         digitalWrite(relai, value); // Mettre a jour le relai
73         Serial.println("Relai Active");
74     } else if (!(value)) {
75         digitalWrite(relai, value); // Mettre a jour le relai
76         Serial.println("Relai Desactive");
77     }
78 }
79
80 void handleTCPClient() {
81     static WiFiClient activeTcpClient; // Client TCP pour maintenir une
82     connexion active
83
84     WiFiClient newClient = tcpServer.available();
85     if (newClient) {
86         if (!activeTcpClient || !activeTcpClient.connected()) {
87             activeTcpClient = newClient; // Accepter le nouveau client
88             Serial.println("Nouveau client TCP connecte");
89         }
90     }
91
92     if (activeTcpClient.connected()) {
93         while (activeTcpClient.available()) {
94             String line = activeTcpClient.readStringUntil('\r'); // Lire la
95             ligne recue
96             Serial.print("Donnees TCP recues: ");
97             Serial.println(line);
98             activeTcpClient.stop();
99             Serial.println("Deconnexion de l'utilisateur TCP");
100
101             for (int i = 0; i < line.length(); i++) {
102                 char ch = line.charAt(i);
103                 if (ch == 'N') {
104                     updateRelai(1); // Commande pour activer le relai
105                 } else if (ch == 'F') {
106                     updateRelai(0); // Commande pour desactiver le relai
107                 } else if (isdigit(ch)) {
```

```
106     String numStr = "";
107     while (i < line.length() && isdigit(line.charAt(i))) {
108         numStr += line.charAt(i);
109         i++;
110     }
111     int pwmValue = numStr.toInt(); // Convertir la chaine de
    chiffres en entier
112     updatePWM(pwmValue);           // Mettre a jour la PWM
113     break;                         // Sortir de la boucle apres
    avoir traite les chiffres
114 }
115 }
116 }
117 }
118 }
119
120 void setup(void) {
121     pinMode(pwmPin, OUTPUT); // Configurer la broche PWM en sortie
122     pinMode(relai, OUTPUT);  // Configurer la broche RELAI en sortie
123     analogWrite(pwmPin, 0);  // Initialiser la valeur PWM a 0
124     analogWriteResolution(10); // On met la resolution du PWM sur 10 bits
    (1024 valeurs)
125     analogWriteFreq(30000);   // On met la frequence a 30kHz
126
127     Serial.begin(115200);
128     WiFi.mode(WIFI_STA);
129     WiFi.begin(ssid, password); // Se connecter au reseau WiFi
130
131     // Attendre la connexion WiFi
132     while (WiFi.status() != WL_CONNECTED) {
133         delay(500);
134         Serial.print(".");
135     }
136
137     // Afficher l'adresse IP une fois connecte
138     Serial.println("");
139     Serial.print("Connected to ");
140     Serial.println(ssid);
141     Serial.print("IP address: ");
142     Serial.println(WiFi.localIP());
143 }
```

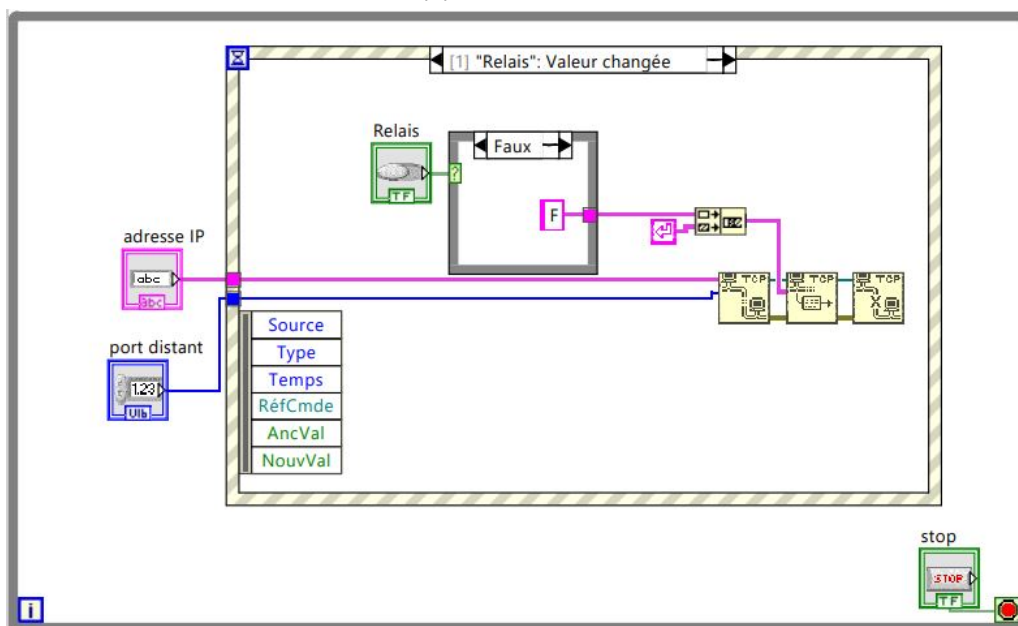
```
144 // Demarrer le serveur MDNS
145 if (MDNS.begin("ouecgreg")) { Serial.println("ouecgreg.local est
    disponible"); }
146
147 // Configurer les routes pour le serveur Web
148 server.on("/", handleRoot);
149 server.on("/setpwm", handleSetPWM);
150 server.onNotFound([]() {
151     server.send(404, "text/plain", "Not found");
152 });
153
154 // Demarrer les serveurs Web et TCP
155 server.begin();
156 tcpServer.begin();
157
158 Serial.println("Serveurs Web et TCP démarres");
159 }
160
161 void loop(void) {
162     server.handleClient(); // Gerer les clients Web
163     MDNS.update();        // Mettre a jour le service MDNS
164     handleTCPClient();    // Gerer les clients TCP
165 }
```

Listing A.2 – Code Arduino pour le Serveur TCP et WEB

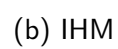
## A.4 LabView Test de Validation



(a) Relais Activé



(b) Relais Désactivé





Capture en cours de emp250 (host 192.168.0.53 and tcp)

Fichier Editer Vue Aller Capture Analyseur Statistiques Telephone Wireless Outils Aide

Apply a display filter ... <Ctrl+>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.0.75	192.168.0.53	TCP	74	35404 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=316197774 TSecr=0 NS=128
2	0.003945298	192.168.0.53	192.168.0.75	TCP	62	80 → 35404 [SYN, ACK] Seq=0 Ack=1 Win=2144 Len=0 MSS=536 SACK_PERM=1
3	0.004897993	192.168.0.75	192.168.0.53	TCP	64	35404 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
4	0.005000000	192.168.0.53	192.168.0.75	HTTP	178	HTTP/1.1 303 See Other (application/x-www-form-urlencoded)
5	0.020922650	192.168.0.53	192.168.0.75	HTTP	54	35404 → 80 [ACK] Seq=513 Ack=125 Win=64116 Len=0
6	0.021609085	192.168.0.75	192.168.0.53	TCP	450	GET / HTTP/1.1
7	0.037192695	192.168.0.75	192.168.0.53	HTTP	125	Seq=125 Ack=909 Win=1772 Len=0
8	0.047718443	192.168.0.53	192.168.0.75	TCP	60	80 → 35404 [PSH, ACK] Seq=125 Ack=909 Win=1772 Len=115 [TCP segment of a reassembled PDU]
9	0.050250685	192.168.0.75	192.168.0.53	TCP	169	80 → 35404 [PSH, ACK] Seq=909 Ack=249 Win=64001 Len=0
10	0.051530860	192.168.0.75	192.168.0.53	HTTP	54	35404 → 80 [ACK] Seq=909 Ack=738 Win=64001 Len=0
11	0.052000000	192.168.0.53	192.168.0.75	TCP	60	80 → 35404 [FIN, ACK] Seq=738 Win=1772 Len=0
12	0.056813502	192.168.0.75	192.168.0.53	TCP	60	80 → 35404 [FIN, ACK] Seq=909 Ack=739 Win=64001 Len=0
13	2.052705242	192.168.0.75	192.168.0.53	TCP	54	35404 → 80 [FIN, ACK] Seq=739 Ack=910 Win=1771 Len=0
14	2.052952247	192.168.0.75	192.168.0.53	TCP	60	80 → 35404 [ACK] Seq=739 Ack=910 Win=1771 Len=0
15	2.056785176	192.168.0.75	192.168.0.53	TCP	60	80 → 35404 [ACK] Seq=739 Ack=910 Win=1771 Len=0

Frame 4: 566 bytes on wire (4528 bits), 566 bytes captured (4528 bits) on interface emp250, id 0

- Ethernet II, Src: Micro-St, 81:74:aa (44:8a:5b:81:74:aa), Dst: Espressi\_30:b4:41 (bc:dd:c2:30:b4:41)
- Internet Protocol Version 4, Src: 192.168.0.75, Dst: 192.168.0.53
- Transmission Control Protocol, Src Port: 35404, Dst Port: 80, Seq: 1, Ack: 1, Len: 512
- Hypertext Transfer Protocol
  - HTML form URL Encoded: application/x-www-form-urlencoded
  - Form item name/value = "49"
  - Value: 49

Paquets: 15 / Affichés: 15 (100.0%)

Profile: Default

Figure A.4 – Trame WireShark WEB