SAE3 - Assistant domotique

Dossier de Fabrication

Guillaume Seimandi
Groupe ESE B

Sommaire

Introduction

Ce dossier s'appuie sur le dossier de conception et a pour vocation de passer en revue les différentes étapes qui ont trait à la fabrication et au dépannage de la carte d'assistance domotique. Ainsi, il est composé de deux sections. La première évoque la fabrication et comprends le routage de la carte ainsi que la nomenclature des composants. La deuxième partie couvre la procédure de test, sous un aspect pratique mais également théorique avec l'explication du programme de test.

Nomenclature et routage

Avant toute chose, il est nécessaire de présenter le schéma structurel. En effet, afin de simplifier le routage de la carte, certains branchements ont été modifiés au niveau de l'ESP32.

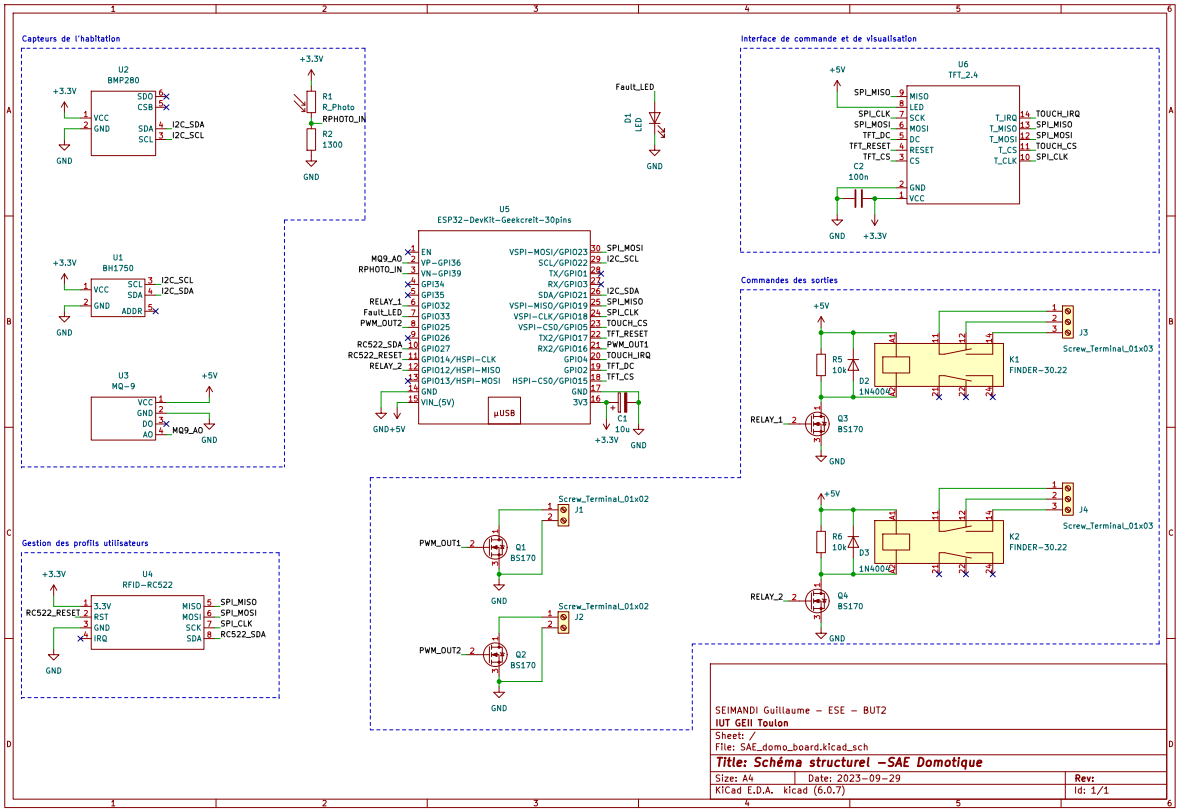

FIGURE 2.0.1 – Schéma structurel du système

2.1 Nomenclature

La nomenclature suivante s'appuie sur la numérotation présente sur le schéma structurel.

N°	Quantité	Description	Valeur	Taille/Boitier	Référence
C1	1	Condensateur	10u	SMD 1206	WURTH ELEKTRONIK 885012208018
C2	1	Condensateur	100n	SMD 1206	WURTH ELEKTRONIK 885012208009
D1	1	LED	-	SMD 1206	MULTICOMP PRO MP007092
D2 et D3	2	1N4004	-	-	ONSEMI 1N4004RLG
J1 et J2	2	Bornier à vis	-	2 pins	KF301-2P
J3 et J4	2	Bornier à vis	-	3 pins	KF301-3P
K1 et K2	2	Relais	-	-	FINDER-30.22.7.005
Q1 à Q4	4	Transistor	-	TO-92	BS170
R1	1	Photorésistance	-	-	GL5528
R2	1	Resistance	1,3k	SMD 0806	MULTICOMP PRO MP003722
R3 et R4	2	Resistance	10k	SMD 0806	VISHAY CRCW120610K0JNEA
U1	1	Capteur de luminosité	-	-	GY-302 BH1750
U2	1	Capteur de température et de pression	-	-	GY-BMP280
U3	1	Capteur de gaz	-	-	MQ-9
U4	1	Lecteur RFID	-	-	MFRC522_RFID
U5	1	ESP32	-	-	ESP-WR32
U6	1	TFT_2.4 (ILI9341) avec tactile (XPT2046)	-	-	TOUCH_TFT_LCD_2.4

Figure 2.1.1 – Nomenclature des composants

2.2 Routage de la carte

Le routage est réalisé avec les paramètres de pistes et de vias présentés dans le tableau ci-dessous. La largeur des pistes est conditionnée par le courant qui sera amené à circuler en elles. Or, le courant le plus élevé permis dans la carte est le 2 A au niveau des relais. Une largeur de piste de 1 mm convient pour le reste de la carte. Il est à noter que pour rentrer dans la norme IPC-2152, elle devrait être de 1.3mm à 2 A. Toutefois, une largeur de piste de 1.5mm a été choisie pour les pistes en les borniers et les relais.

Isolation	Largeur de Piste	Diamètre des vias	Trou des vias
$0,4mm$	$1mm$	$2mm$	$0,8mm$

FIGURE 2.2.1 – Paramètre de piste et de vias du routage

De plus, afin d'améliorer au maximum la qualité de la masse, un plan de masse est présent des deux côtés. Les figures suivantes représentent les couches de cuivre : supérieure (en rouge) et inférieure (en bleu).

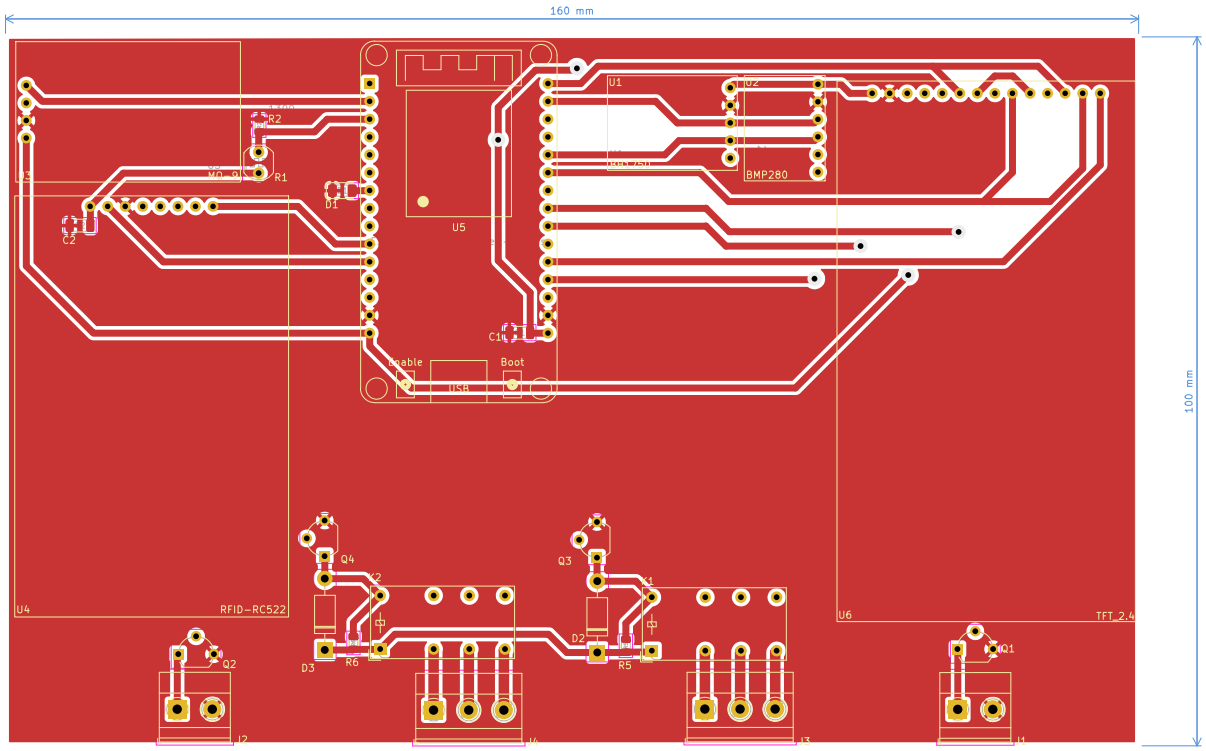

FIGURE 2.2.2 – Couche de cuivre supérieure vue de dessus

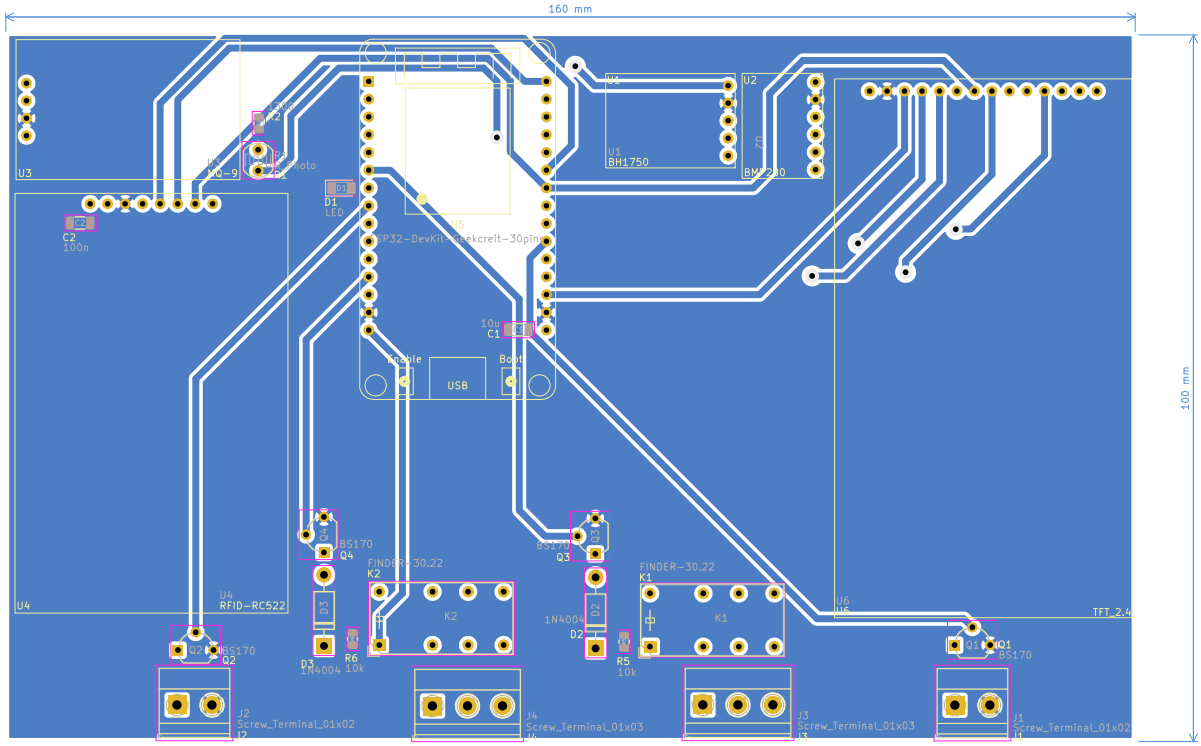

FIGURE 2.2.3 – Couche de cuivre inférieure vue de dessus

Programme et procédure de test

3.1 Programme de test

Le programme de test à proprement parler est un script Python qui interagit avec la carte via le port série. Pour ce faire, le programme de la carte expliqué dans le dossier de conception comprend la gestion de commande sur le port série. Toutefois, le programme teste aussi la bonne mise en place du site web, du websocket, le fonctionnement des sorties et la validité mesures des capteurs. Dans l'optique de vérifier un maximum de paramètre du système, le port série s'est avéré comme étant la meilleure option pour interagir avec la carte.

3.1.1 Description

Tout d'abord, le programme est écrit en Python parce que Python grâce à ses librairies permet de gérer de manière simple et efficace : la liaison série, le protocole HTTP et la connexion au Websocket. Par ailleurs, le programme ne disposant pas d'interface graphique, toutes les actions se réalisent depuis la console. Le programme s'exécute en suivant différentes étapes de tests : chacune s'appuyant sur la précédente. Finalement, le programme peut exporter les résultats du diagnostic dans une fiche d'intervention partiellement complétée. Pour ce faire, le programme fait appel à un compilateur LaTeX : pdflatex.

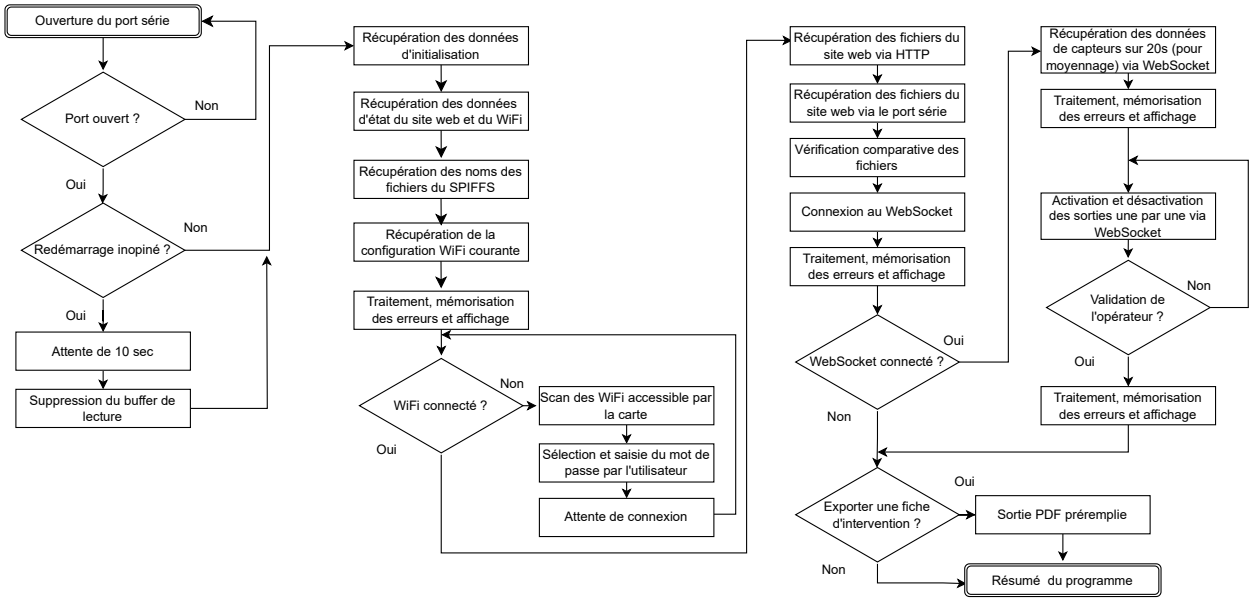

FIGURE 3.1.1 – Diagramme de fonctionnement du programme de test

3.1.2 Interaction avec la carte

Comme explicité plus haut, les interactions entre la carte et l'ordinateur s'effectuent grâce au port série. Plus précisément, par le biais de commandes envoyées sur le port série (voir tableaux ci-après). L'implémentation de la commande diag se fait dans le fichier *SAEDomoDiag.cpp* et est appelée depuis *SAEDomoLIB.cpp*.

Commande	Argument	Paramètre	Description
diag	init	-	Demande toutes les informations liées à la dernière initialisation des composants et des librairies
	fsfl	-	Demande la liste des fichiers présents dans la SPIFFS
	web	-	Demande toutes les informations liées à l'état de la connectivité web (configuration WiFi non-présente : voir l'argument *file*)
	auth	-	Demande le statue du gestionnaire de profile
	wifiscan	-	Demande à la carte d'effectuer un scan des WiFi disponibles et renvoie la liste
	file	<Nom du fichier>	Renvoi le contenue du fichier s'il est présent dans la SPIFFS
setPWD	-	<Mot de Passe>	Modifie le mot de passe de la configuration WiFi de la carte
setSSID	-	<SSID>	Modifie le SSID de la configuration WiFi de la carte
connect	-	-	Lance une tentative de connexion au WiFi

FIGURE 3.1.2 – Liste des commandes du port série

Pour une raison de commodité, toutes les réponses aux commandes issues de *diag* se font en JSON. En effet, ce format permet de savoir facilement si la réponse est finie ; ce qui est très utile notamment lors de l'envoi de fichiers. Dans les figures qui suivent, il y a un exemple de réponse à la commande *diag wifiscan* et de son traitement par le programme. Il est à noter que cette commande est utilisée pour connecter la carte au WiFi depuis le programme de test. Ainsi, elle n'est appelée que dans ce contexte.

```
{'wifiscan':
    [
        {'SSID': 'SFR-d578', 'RSSI': -55, 'encryption': 'WPA WPA2'},
        {'SSID': 'SFR-3510', 'RSSI': -72, 'encryption': 'WPA WPA2'},
        {'SSID': 'SFR-0d48', 'RSSI': -74, 'encryption': 'WPA WPA2'},
        {'SSID': 'FreeWifi_secure', 'RSSI': -78, 'encryption': 'WPA2 Entreprise'},
        {'SSID': 'Freebox-65F15F', 'RSSI': -79, 'encryption': 'WPA2'},
        {'SSID': 'SFR-4d10', 'RSSI': -84, 'encryption': 'WPA WPA2'},
        {'SSID': 'eduroam', 'RSSI': -86, 'encryption': 'WPA2 Entreprise'},
        {'SSID': 'HOTSPOT Dominos', 'RSSI': -86, 'encryption': 'Open'},
        {'SSID': 'Pulse Store Network', 'RSSI': -86, 'encryption': 'WPA2'},
        {'SSID': 'hugo', 'RSSI': -88, 'encryption': 'WPA WPA2'},
        {'SSID': 'Bbox-A98B4CF3', 'RSSI': -89, 'encryption': 'WPA WPA2'},
        {'SSID': 'Freebox-45464A', 'RSSI': -90, 'encryption': 'WPA2'},
        {'SSID': 'freebox_BRNHRS', 'RSSI': -91, 'encryption': 'WPA'},
        {'SSID': 'FreeWifi_secure', 'RSSI': -91, 'encryption': 'WPA2 Entreprise'},
        {'SSID': 'FreeWifi_secure', 'RSSI': -92, 'encryption': 'WPA2 Entreprise'}
    ]
}
```

Exemple de réponse à la commande *diag wifiscan*

```
####################################
[ERROR] The WiFi configuration appears to be incorrect. Would you like to modify it? y / n y
Scanning Networks...
Following network accessible on your device :
    1: SFR-d578            ▓▓▓▓▓▓ [WPA WPA2]
    2: SFR-0d48            ▓▓▓▓   [WPA WPA2]
    3: SFR-3510            ▓▓▓▓   [WPA WPA2]
    4: FreeWifi_secure     ▓▓▓▓   [WPA2 Entreprise]
    5: Freebox-65F15F      ▓▓▓▓   [WPA2]
    6: SFR-4d10            ▓▓     [WPA WPA2]
    7: Bbox-A98B4CF3       ▓▓     [WPA WPA2]
    8: HOTSPOT Dominos     ▓▓     [Open]
    9: Pulse Store Network ▓▓     [WPA2]
    10: Bbox-64E5F6D8      ▓▓     [WPA2]
    11: FreeWifi_secure    ▓▓     [WPA2 Entreprise]
    12: freebox_BRNHRS     ▒      [WPA]
    13: FreeWifi_secure    ▒      [WPA2 Entreprise]
Choose SSID (enter its number): 3
Enter Password (SFR-3510): Q7LPTE6G44AL
New configuration : SSID -> SFR-3510  PWD -> Q7LPTE6G44AL
Is it good? y / n y
Waiting for connexion...
####################################
```

FIGURE 3.1.3 – Exemple de traitement de la réponse ci-dessus : affichage console

Les réponses en JSON permettent une bonne lisibilité, cependant, la commande *diag init* renvoie un mot de 32 bits : typiquement *009201FF* lorsque tout s'est bien passé. Ce code se décompose comme suit :

```c
typedef struct {
    uint16_t device_RFID_state : 1;
    uint16_t device_RFID_autotest : 1;
    uint16_t device_TFT_state : 1;
    uint16_t device_BH1750_state : 1;
    uint16_t device_BMP280_state : 1;
    uint16_t SPIFFS_mounted : 1;
    uint16_t web_wifi_state : 1;
    uint16_t web_server_state : 1;
    uint16_t auth_handler_init : 1;

    uint8_t device_RFID_version;
    uint8_t stuffing; // Pour pouvoir cast en uint32_t
} InitInfo;
```

Structure InitInfo (SAEDomoDiag.h)

Il est à noter que cette structure est remplie à chaque initialisation de la carte par la fonction *setup* du sketch Arduino.

Toutefois, dans les phases 2, 3 et 4 du programme de test (voir figure 3.1.1), le programme n'utilise plus le port série. Il utilise le WebSocket, mais aussi directement le serveur web via des requêtes *HTTP_GET* et simule un client Web pour récupérer les données des capteurs, mais aussi pour actionner les sorties et comparer les fichiers fournis par le site web aux fichiers présent dans la SPIFFS. Le programme principal de la carte actualise les valeurs sur les WebSocket toutes les secondes. Ainsi, pour pouvoir faire une moyenne, le programme de test attend 23 données, soit 23 secondes d'attentes.

Pour ce faire, le programme de test se procure les informations liées au serveur web via la commande *diag web* : la réponse à celle-ci est présente dans la figure ci-dessous.

```json
{
    'web': {
        'wifi': {
            'state': True,
            'connected': True,
            'address': '192.168.0.12'
        },
        'server': {
            'state': True,
            'port': 80
        }
    }
}
```

Réponse type à la commande *diag web* (JSON)

Les réponses JSON non-exposées sont visibles dans le programme de la carte dans le fichier SAEDomoDiag.cpp. *Elles ne sont pas montrées ici car elles ne présentent pas de difficulté particulière et sont calquées sur celles présentées.*

Par ailleurs, comme il a été expliqué plus haut, Python permet de réaliser des requêtes HTTP et de se connecter à une WebSocket de manière simple comme le montre les figures suivantes.

```
requestHTTPIndex = urllib.request.urlopen("http://" + systemData['web']['wifi']['address']).
    read().decode('utf-8')
[...]
requestHTTPIndex = urllib.request.urlopen(
            "http://" + systemData['web']['wifi']['address'] + "/style.css").read().decode('
    utf-8')
[...]
requestHTTPIndex = urllib.request.urlopen(
            "http://" + systemData['web']['wifi']['address'] + "/script.js").read().decode('
    utf-8')
```

Requêtes *HTTP_GET* sur le serveur web (main.py : Python)

```
ws = None
try:
    ws = websocket.create_connection("ws://" + systemData['web']['wifi']['address'] + "/ws")
    wsDataIn.append(ws.recv())
    print('WebSocket connexion: OK')
except websocket.WebSocketBadStatusException:
    errorList2Pass.append('WebSocket: ERROR')
    print('WebSocket connexion: 500 - Server Internal ERROR')
```

Connexion et utilisation du WebSocket (main.py : Python)

Dans ces deux dernières figures, deux structures de données sont récurrentes au long du programme : *systemData* et *errorList2Pass*. Ces deux structures permettent de centraliser les données récoltées par le programme de test de manière organisée. La structure *systemData*, utilisée tout au long des étapes, est de type dict. Il est à savoir que le type dict en Python est très proche dans son utilisation d'une structure JSON ; ce qui la rend particulièrement adaptée. Elle permet de rassembler les informations collectées. À titre d'exemple, la figure suivante montre la variables *systemData* en fin d'exécution.

```
{
    'device': {
        'device_RFID': {
            'version': 146,
            'state': True,
            'autotest': True
        },
        'device_TFT': {
            'state': True
        },
        'device_BH1750': {
            'state': True
        },
        'device_BMP280': {
            'state': True
        }
    },
    'SPIFFS': {
        'mounted': True,
        'content': ['index.html', 'profiles.json', 'script.js', 'style.css', 'wifi_conf.json
    ']
    },
    'web': {
        'wifi': {
            'state': True,
            'connected': True,
            'address': '192.168.0.12',
```

```
27                'SSID': 'SFR-3510',
28                'password': 'Q7LPTE6G44AL'
29            }, 'server': {
30                'state': True,
31                'port': 80
32            }
33        },
34        'auth': {
35            'state': True
36        }
37    }
```

Exemple de la complétion de la variable *systemData* en fin d'exécution

Il est à noter que toutes les informations qui ne sont pas réutilisées dans d'autres étapes ne figurent pas dans *systemData*.

La structure *errorList2Pass* est également de type dict et permet de rassembler les erreurs de la phase 2 : dans l'optique de réaliser un résumé en fin d'exécution. De même, il existe une errorListXPass pour chaque étape du programme.

Maintenant, que les principes de bases de communication sont établis, un test s'impose. La figure suivante montre la sortie du programme avec l'utilisation de toutes ses fonctionnalités. Dans cet exemple, la configuration WiFi de la carte est vide et le BMP280 est débranché.

```
1  Waiting for device to load...
2  ----------------------------
3  SmartCheese diagnostics
4      Date: 2024-01-07 15:03:38.003500
5      Port: COM3
6  ------------------------------------------------------------
7  First pass : States of devices, SPIFFS, Web and AuthHandler
8  ----------------------------
9      Devices:
10         RFID:
11             State: OK
12             Autotest: OK
13             Version: 146
14         TFT: OK
15         BH1750: OK
16         BMP280: ERROR
17     SPIFFS:
18         State: OK
19         Content: OK
20     Web:
21         Wifi:
22             State: OK
23             Connected: OFFLINE
24             Address: 0.0.0.0
25             SSID: ddadz
26             Password: Q7LPTE6G44AL
27         Server:
28             State: OK
29             Port: 80
30         Auth: OK
31 ------------------------------------------------------------
32 Second pass : Connectivity tests
33 ----------------------------
34 ABORTED: Board is offline
35 ########################################
36 [ERROR] The WiFi configuration appears to be incorrect. Would you like to modify it? y / n y
37 Scanning Networks...
```

```
38 Following network accessible on your device :
39      1: SFR-d578             ##### [WPA WPA2]
40      2: SFR-3510             ##### [WPA WPA2]
41      3: FreeWifi_secure      ###   [WPA2 Entreprise]
42      4: SFR-0d48             ###   [WPA WPA2]
43      5: Freebox-65F15F       ###   [WPA2]
44      6: RoadEyes_recONE      ##    [WPA2]
45      7: SFR-4d10             ##    [WPA WPA2]
46      8: Bbox-A98B4CF3        ##    [WPA WPA2]
47      9: FreeWifi_secure      ##    [WPA2 Entreprise]
48      10: Pulse Store Network##     [WPA2]
49      11: Freebox-45464A      ##    [WPA2]
50      12: Bbox-64E5F6D8       ##    [WPA2]
51      13: khan 1881           ##    [WPA2]
52      14: HOTSPOT Dominos     ##    [Open]
53 Choose SSID (enter its number): 2
54 Enter Password (SFR-3510): Q7LPTE6G44AL
55 New configuration : SSID -> SFR-3510  PWD -> Q7LPTE6G44AL
56 Is it good? y / n y
57 Waiting for connexion...
58 ########################################
59 Trying HTTP connexion on 192.168.0.20...
60 File index.html from HTTP: CONFORM
61 File style.css from HTTP: CONFORM
62 File script.js from HTTP: CONFORM
63 Trying WebSocket connexion...
64 WebSocket connexion: OK
65 ------------------------------------------------------------
66 Third pass : WebSocket based data analysis
67 -------------------------
68 Collecting Data: ---------------------------------------------------- | 100%
69 Data analysis (avg. value / 20s):
70      BH1750: 56.633330841064456 lx (seems CORRECT)
71      LDR: 10.907168006896972 lx (seems CORRECT)
72      BMP280 / Temp: 0.0 oC (seems INCORRECT)
73      BMP280 / Press: 0.0 hPa (seems INCORRECT)
74      MQ-9: 0.013812053985893726 ppm (seems CORRECT)
75 ------------------------------------------------------------
76 Fourth pass : WebSocket based output tests
77 -------------------------
78 Testing digital outputs:
79      Relay 1 Test - ON - OFF
80      Relay 2 Test - ON - OFF
81 Testing analog outputs (0% -> 75%):
82     PWM 1 Test - 75% - 0%
83     PWM 2 Test - 75% - 0%
84 Retry? y / n n
85 ############################################################
86 Summary of the diagnostic
87 ########################
88 1st Pass : States of devices, SPIFFS, Web and AuthHandler ->
89      device.device_BMP280.state: ERROR
90 2nd Pass : Connectivity tests -> OK
91 3rd Pass : WebSocket based data analysis ->
92      BMP280 / Temp seems INCORRECT : WARNING
93      BMP280 / Press seems INCORRECT : WARNING
94 4th Pass : WebSocket based output tests -> seems OK
95 ############################################################
96 Export PDF ? y / n y
97 Reading Templates...
98 Filling blanks...
99 Exported file can be found in the same directory as the main.py
```

Exemple de sortie du programme

Il a noté qu'à la ligne 96, le programme demande si l'utilisateur veut exporter un PDF. Si celui-ci répond oui, le programme exporte une fiche d'intervention pré-remplie pour simplifier la tâche du technicien en charge de l'intervention.

Ce dossier a abordé plus haut, la création de ce PDF : elle se fait par l'appel à un compilateur LaTeX.

```
os.system('pdflatex -quiet -enable-installer -aux-directory="./tempComp" -include-directory
    ="./docTemplate" -output-directory="./" ./docTemplate/SAE3-Fiche_Intervention-GSeimandi.
    tex')
```

Appel au compilateur LaTeX(main.py : Python)

Toutefois, l'explication détaillée des templates finissant par l'extension .tex, présents dans le dossier *./docTemplate* dépassent le cadre de ce dossier. En effet, LaTeX, à l'image d'HTML, n'est qu'une manière de mettre en page des PDF et l'explication détaillé n'est donc pas pertinentes tant les pré-requis à la compréhension sont grand et car il n'y a aucune vraie complexité. Toutefois, ces templates comportent des balises qui ne sont pas issues de la syntaxe de LaTeX : *?nom_var ?*. Celles-ci permettent au programme Python de connaître les endroits à modifier et où insérer les données.

```
\section{Tests Fonctionnels}
\noindent
\noindent
\centering
\begin{tabular}{|C{8cm}|C{1cm}|C{6cm}|}
\hline \textbf{Fonction} & \textbf{\'Etat} & \textbf{Commentaire} \tabularnewline
\hline Connexion WiFi & ?wifi_connected? & - \tabularnewline
\hline Visualisation via web serveur & ?server_state? & - \tabularnewline
\hline Visualisation via écran TFT &  & - \tabularnewline
\hline Gestion des profils & ?auth_state? & - \tabularnewline
\hline Contenu de la SPIFFS & ?spiffs_state? & ?spiffs_comment? \tabularnewline
\hline Mesure de luminosité & ?lum_state? & ?lum_comment? \tabularnewline
\hline Mesure de température et de pression & ?tp_state? & ?tp_comment? \tabularnewline
\hline Mesure de gaz & ?gaz_state? & - \tabularnewline
\hline Sorties analogiques &  & - \tabularnewline
\hline Sorties TOR &  & - \tabularnewline
\hline

\end{tabular}
```

Exemple de templates non-formatées (docTemplate.tex : LaTeX)

3.2 Environnement et procédure de test

Dans un premier, il faut se munir d'un ordinateur avec le programme de test, un interpréteur Python avec les librairies serial, urllib, json et websocket installées et d'un compilateur LaTeXbasé sur pdflatex. Ceci fait, il faut inspecter les soudures de la carte pour éviter les courts-circuits. Finalement, il faut brancher la carte et lancer le programme de test. Une fois le programme fini, il ne reste plus qu'à suivre la fiche d'intervention (voir annexe). De plus, le programme est cross-plateform et peut-être appelé de la manière suivante.

```
python3 ./main.py [Port]
```

Lancement du programme de test (CMD ou Bash)

Si plusieurs composant apparaissent comme défectueux, notamment dans le cas des composants qui utilisent un bus de données (I2C, SPI), il convient de tester indépendamment les composants. Pour ce faire, il suffit de débrancher la carte puis de débrancher tout les composants défectueux sauf un. Ensuite, il faut relancer le programme de test et réitérer l'opération avec tout les composants défectueux. Cette manipulation permet d'isoler le ou les composants qui posent problème. En effet, un composant défectueux sur un bus peut rendre inutilisable les autres composants sur ce même bus.

Conclusion

Ce dossier permet la mise en production de la carte : du routage, jusqu'au test de la carte. Il décrit le programme de test qui peut être utilisé dans le cadre d'un SAV ou pour un test de sortie d'usine, ainsi que les procédures pour mener ce test à bien.

Annexes

SAE_Domo.ino : Code source Arduino.

SAEDomoLIB.zip : Code source de la librairie SAEDomoLIB.

index.html, style.css, script.js, wifi_conf.json et profiles.json : Fichiers contenus dans la mémoire de l'ESP32.

SAE3-Fiche_Intervention-GSeimandi.pdf : Fiche d'intervention vierge.

SAE3-Fiche_Intervention-GSeimandi_EXEMPLE.pdf : Fiche d'intervention de l'exemple (sous-section 3.1.2).

SAE3-TestUtils.zip : Programme de test la carte (à exécuter à l'intérieur du dossier fourni).