

# Projet ETML-ES - Cahier des charges

## Commande IR domotique

N° projet 2209 V1

<b>Entreprise/Client:</b>	S. Castoldi	<b>Département:</b>	SLO
<b>Demandé par (Prénom, Nom):</b>	S. Castoldi	<b>Date:</b>	23.11.2022

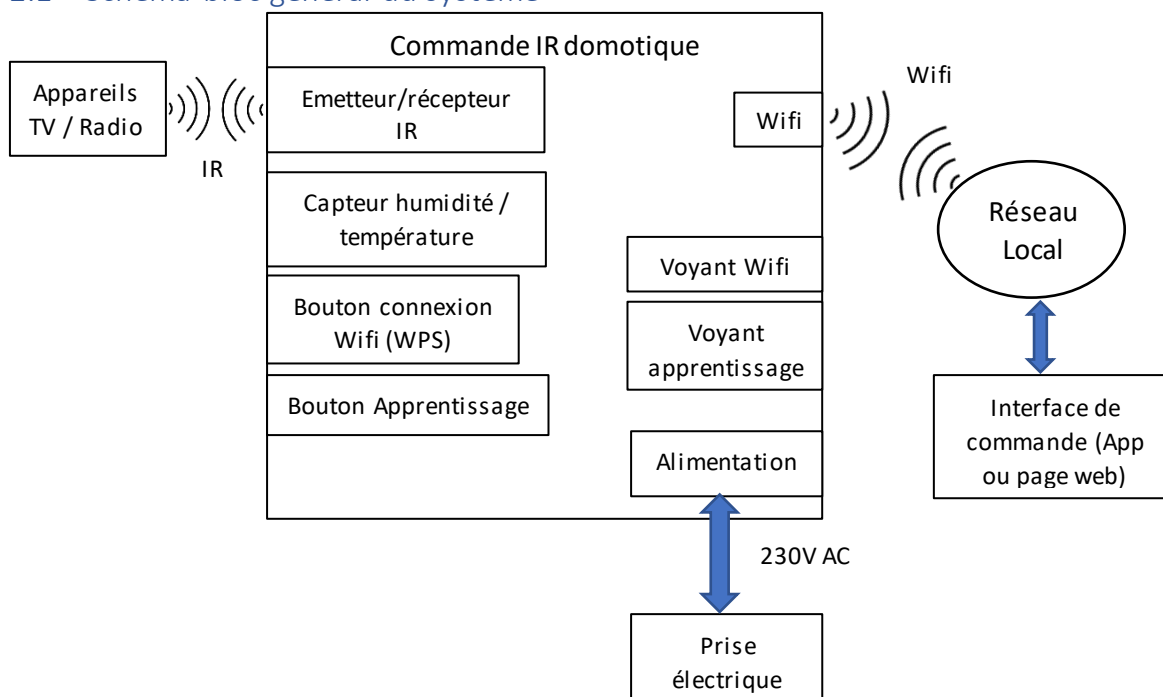
<b>Auteur (ETML-ES):</b>	Einar Farinas Arze	<b>Filière:</b>	SLO
		<b>Date:</b>	23.11.2022

## 1 But du projet

Le but de ce projet est de concevoir un système électronique permettant d'activer ou désactiver à des périodes précises des appareils dit « grand public » style TV / Radio / lumière. Ce système doit pouvoir se loger dans un boîtier style « bloc secteur » et doit se « plugger » dans une prise électrique. Le système sera connecté au réseau local par Wifi afin de pouvoir le commander et configurer à distance. Lorsque le système est connecté au 230V, et qu'il n'y a pas d'interaction (mode apprentissage – réception de trame IR / envoi de trame IR / réception de commande/ envoi d'information), le système ne devra pas dissiper plus d'un 1 Watt.

## 2 Spécifications du projet

### 2.1 Schéma bloc général du système



## 2.2 Alimentation

Le système sera branché dans une prise électrique et alimenté par celle-ci (230V<sub>AC</sub>). Si possible cette même prise pourra être réutilisée par un autre appareil. Le PCB sera protégé par un fusible.

## 2.3 Wifi

Le système doit se connecter au réseau local via Wifi. La connexion au réseau sera faite automatiquement à l'aide du protocole WPS. La connexion au réseau local permettra de commander les appareils à distance en utilisant des commandes TCP en format texte. Ce format a été défini dans la version précédente du projet.

## 2.4 Emetteur/récepteur IR

L'émetteur IR permettra d'envoyer des trames IR afin de commander les différents appareils. Le récepteur IR permettra d'enregistrer les différentes trames IR de commande des appareils via leur télécommande dans le mode apprentissage.

## 2.5 Capteur humidité/température

Ce capteur permettra de mesurer l'humidité et la température. La lecture sera faite à l'aide des commandes TCP en format texte. Les valeurs retournées devront être directement en [°C] pour la température et en [%] pour l'humidité.

## 2.6 Interface utilisateur (HMI)

### 2.6.1 Voyant Wifi

Une LED (deux couleurs) pour indiquer l'état du réseau Wifi.

- Couleur 1 allumée pour indiquer la connexion au réseau.
- Couleur 2 allumée pour indiquer que le système ne pas connecté au réseau.
- Clignotement de la couleur 1 pour indiquer la réception/envoi des données sur le Wifi.
- Clignotement de la couleur 2 pour indiquer l'attente de la connexion au réseau.

### 2.6.2 Voyant apprentissage

Une LED (deux couleurs) pour indiquer :

- Couleur 1 allumée pour indiquer si le mode apprentissage est actif.
- Clignotement de la couleur 1 pour indiquer la réception d'une trame IR.
- Clignotement (5x) de la couleur 2 pour indiquer l'enregistrement de la trame IR reçue.
- Clignotement (3x) de la couleur 2 pour indiquer l'effacement de la trame IR reçue.

### 2.6.3 Bouton connexion wifi (WPS)

Un bouton pour activer le protocole WPS et pouvoir se connecter par wifi au réseau. Ce même bouton permettra d'effacer les réglages sauvegardés du Wifi.

### 2.6.4 Bouton apprentissage

Un bouton pour activer/désactiver le mode apprentissage afin de lire et enregistrer les trames IR de la télécommande de l'appareil. Ce même bouton permettra d'effacer les trames enregistrées.

Un premier appui pour activer le mode apprentissage. Dans le mode apprentissage une fois que la trame IR a été lue, appuyer une fois sur le bouton pour l'enregistrer. Pour sortir du mode apprentissage, maintenir appuyé 5 secondes sur le bouton.

### 2.6.5 Interface de commande (optionnel)

Interface permettant d'envoyer les commandes texte au système via wifi. S'il reste de temps pour le faire.

## 3 Tâches à réaliser

### 3.1 Pré-étude :

- Recherche d'un boîtier différent pour le ESP32 ou utiliser un module Mikrobus pour faciliter son montage dans le PCB
- Recherche d'un boîtier avec une prise CH (type J) ou EU (type C)
- Analyser l'utilité de l'alimentation 5V

### 3.2 Design + modification et correction du schéma

Correction des erreurs du projet précédent :

- Correction du Footprint des MOSFET
- Correction des schémas en relation avec les différentes LEDs
- Modification liaison ESP32 : broche 3 pin pour sa programmation, IO8/IO7/IO2 pour activer le mode programmation, Tx et Rx pour l'envoi/réception des commandes AT (vérifier pin pour la programmation et envoi/réception)
- Recherche d'une LED IR avec la même longueur d'onde que le récepteur

Amélioration et continuation du projet :

- Recherche des LEDs SMD à deux couleurs
- Analyser le nombre de ports libres pour les nouvelles LEDs
- Recherche d'un convertisseur AC/DC plus petit
- Recherche d'un porte fusible plus petit
- Changer les points de test par de point de test SMD

### 3.3 Software

Corrections :

- Changer l'endroit d'enregistrement des trames IR (EEPROM à FLASH du  $\mu$ C)

Amélioration et continuation du projet :

- Modification du code pour le contrôle des nouvelles LEDs
- Gestion des différents modes pour les boutons
- Ajouter l'utilisation du capteur d'humidité/température dans le code
- Test du fonctionnement

### 3.4 Optionnel

- Développer le software pour l'interface pour l'envoi des commandes
- Ajouter le protocole TCP (MQTT) pour la lecture de la température et humidité

## 4 Jalons principaux

- Pré-étude	07/12/2022
- Design + Modification et correction du schéma	25/01/2023
- PCB	22/03/2023
- Montage des composants.	05/04/2023
- Modification du code et développement des nouvelles parties du software	14/06/2023
- Test et mise au point	14/06/2023
- Rapport	14/06/2023
- Présentation du projet	21/06/2023

Voir le planning en annexe pour plus d'informations.

## 5 Livrables

- Les fichiers sources de CAO électronique du PCB réalisé
- Tout le nécessaire à fabriquer un exemplaire hardware de chaque :
  - fichiers de fabrication (GERBER) / liste de pièces avec références pour commande / implantation (prototype) / modifications / dessins mécaniques, etc
- Les fichiers sources de programmation microcontrôleur (.c / .h)
- Tout le nécessaire pour programmer les microcontrôleurs (logiciel ou fichier .hex)
- Le cas échéant, les fichiers sources de programmation PC/Windows/Linux.
- Le cas échéant, tout le nécessaire à l'installation de programmes sur PC/Windows/Linux.
- Un mode d'emploi du système
- Un calcul / estimation des coûts
- Un rapport contenant les calculs - dimensionnement de composants - structogramme, etc.

## 6 Convention de nommage et liens

Le nom de ce fichier doit être unique et doit donc contenir le nom du projet avec le format suivant :

***aaii\_nomProjet-CDC\_Vn.docx***

avec :

- CDC : pour Cahier des charges
- aaii : numéro de projet, exemple 1708 pour projet de 2017 no 08
- nomProjet : comme son nom l'indique.
- Vn: ou n indique la version du document.

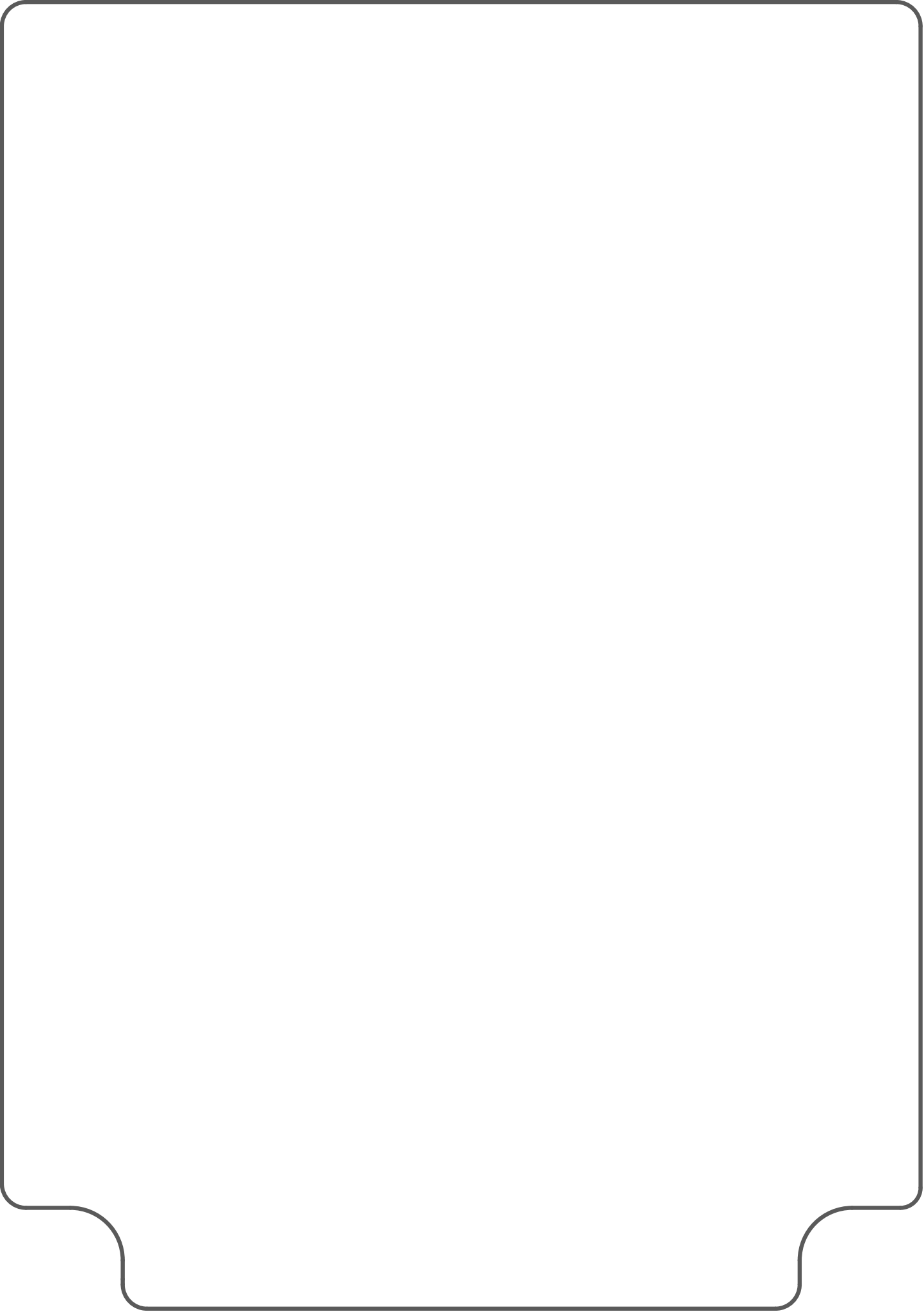
Exemple pour ce projet :

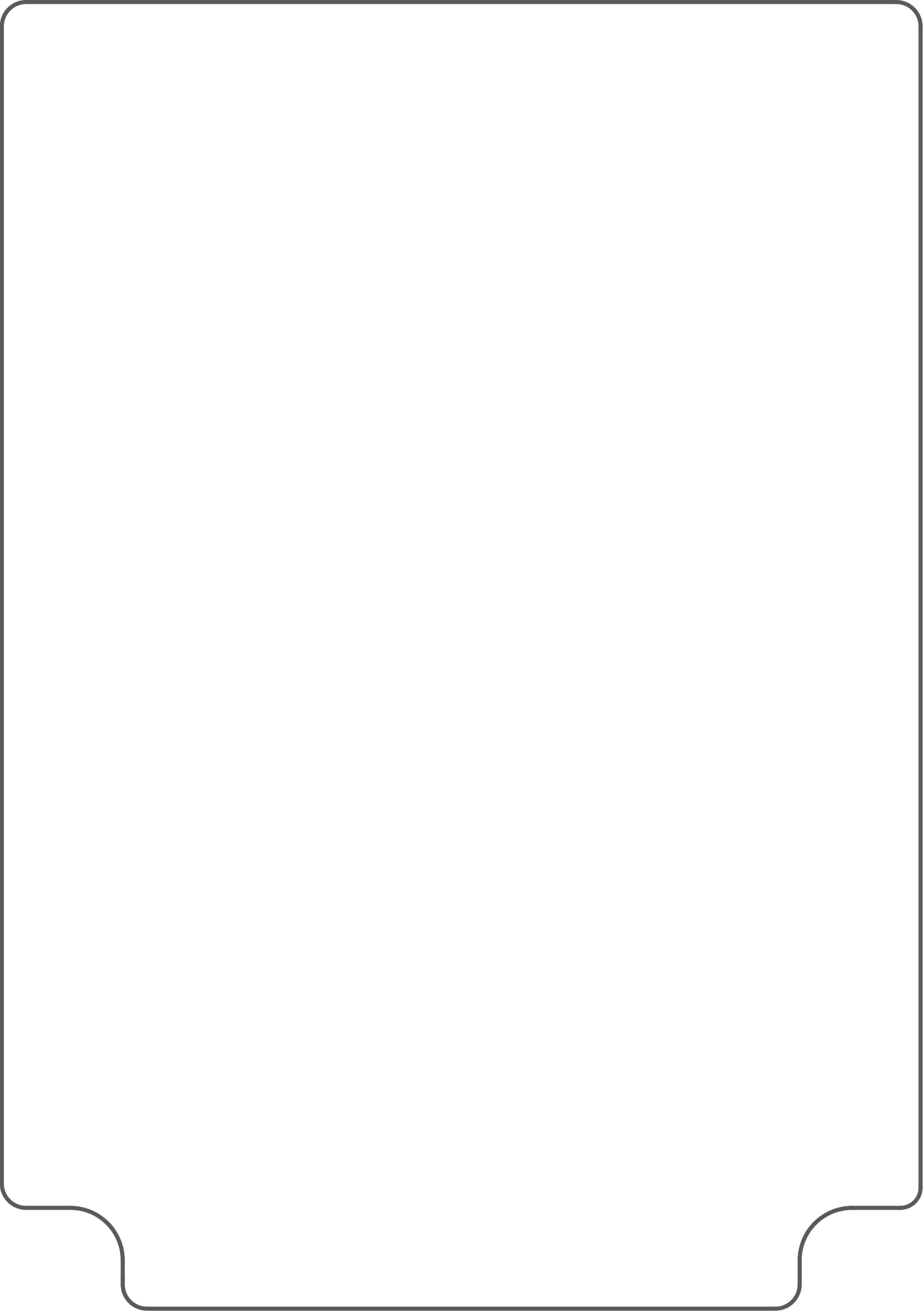
- **2209\_CommandeIRDomotique-CDC-V1.docx**

### 6.1 Stockage du fichier

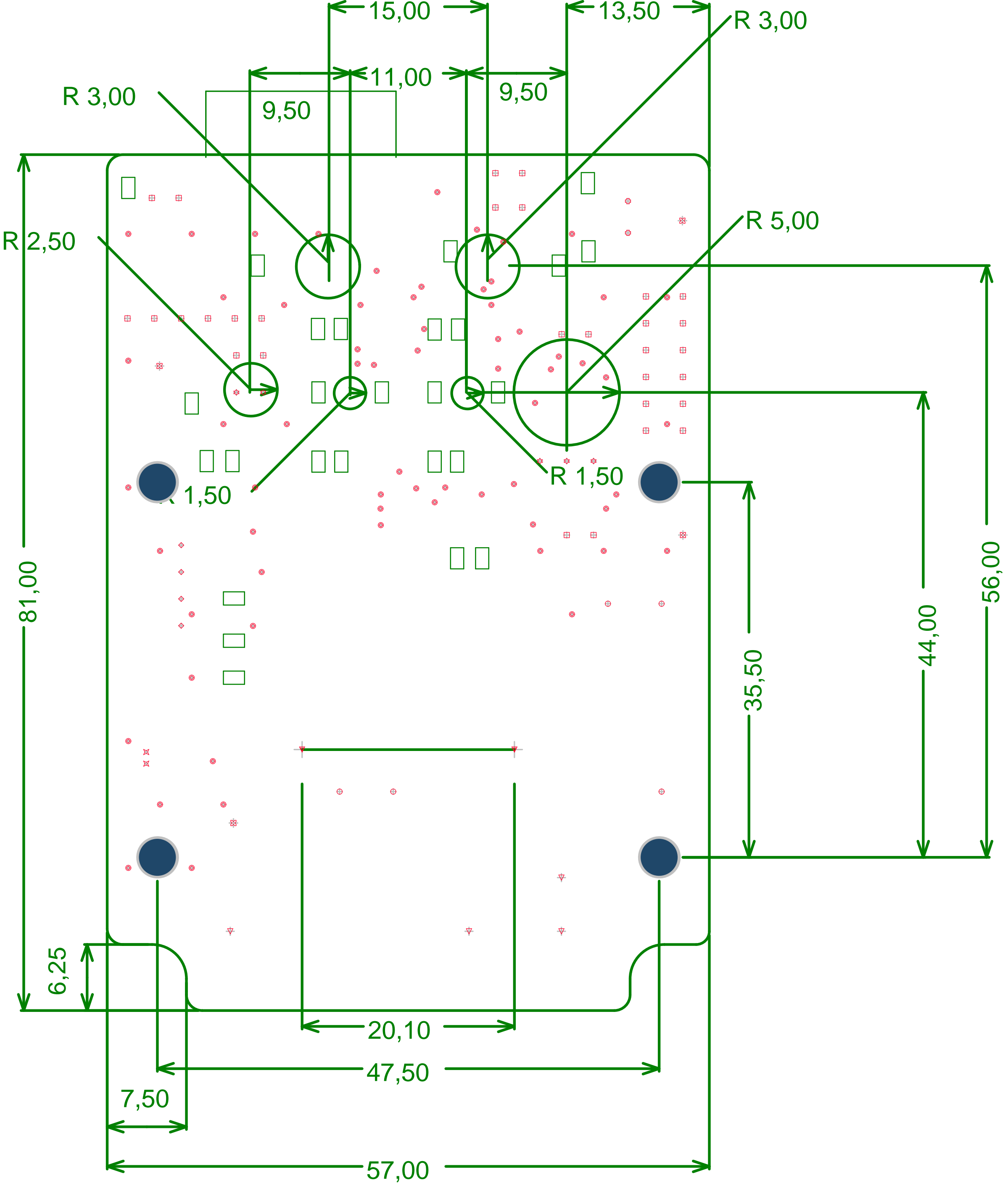
Ce fichier sera stocké à la racine du dossier **/doc** d'un projet.

Ainsi, tous les fichiers de documentation faisant partie du projet sont centralisés dans le même répertoire.

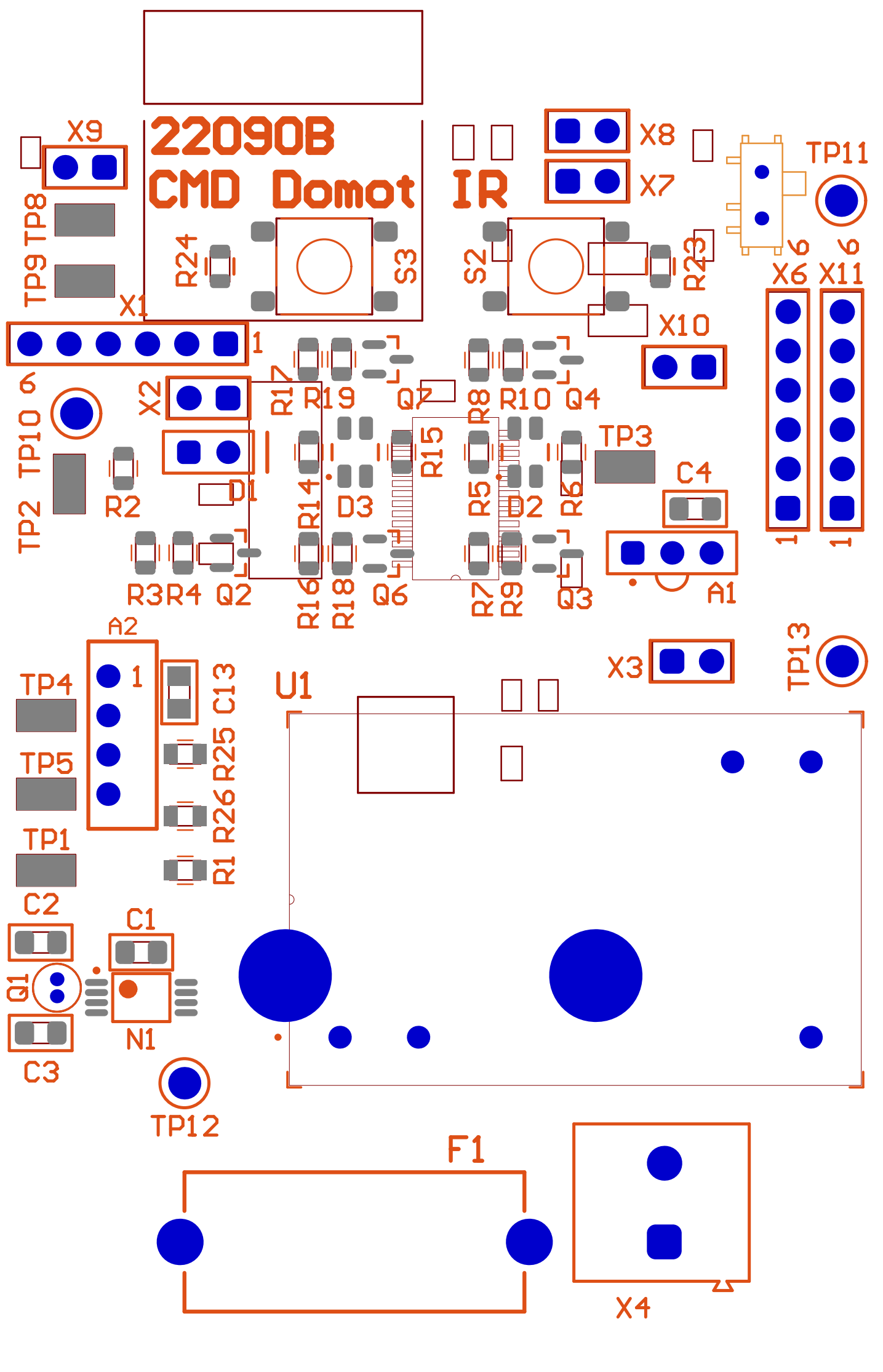


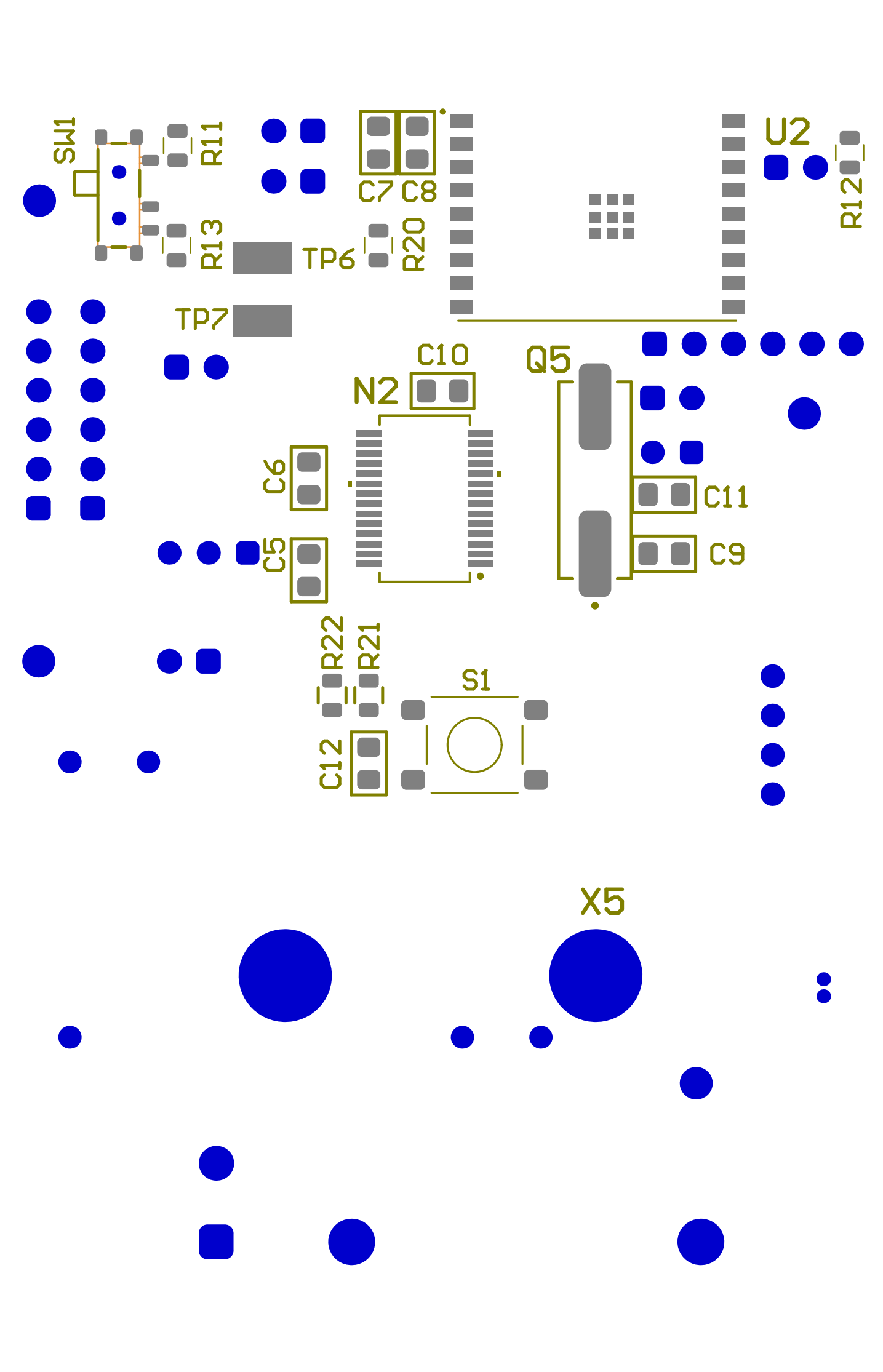


Comment	Description	Designator	Footprint	LibRef	Quantity
TSOP34836	IR Receiver Module 36.0KHZ 45M 950nm	A1	61300311821	TSOP34836	1
Adafruit_Sensor_3721		A2	3721_Adafruit_Sensor	Adafruit_Sensor_3721	1
Cap	Capacitor	C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11, C12, C13	RESC0805(2012)_M	Cap_1, Cap, Cap_2, Cap_3	13
CQY36N	EMITTER IR 950nm 100MA RADIAL. Angle of half intensity +- 55°	D1	61300211821-LED_IR	CQY36N	1
150141SV73110	LED RED & GREEN SMD	D2, D3	WL-SBTW_3528	LED RED GREEN SMD	2
250V 0.1A (5X20)	FUSE BLOK CARTRIDGE 600V 16A PCB	F1	Portefusible	CMP-26292-000005-1	1
MCP79411-I/ST	No Description Available	N1	TSOP65P640X120-8N-RTCC	MCP79411-I/ST	1
PIC32MX130F064B-I/SS	No Description Available	N2	SSOP28_MC_MCH	PIC32MX130F064B-I/SS	1
32kHz768		Q1	XTAL_ECS-.327-12.5-8X-C	Quartz	1
IRLML2402TRPBF	MOSFET N-CH 20V 1.2A SOT-23	Q2, Q3, Q4, Q6, Q7	FP-Micro3-IPC_B	CMP-32901-000179-1	5
10MHz		Q5	ABRA-ABLS-14.7456MHZ-B2-T_V	Quartz	1
Res2	Resistor	R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15, R16, R17, R18, R19, R20, R21, R22, R23, R24, R25, R26	WCAP-CSGP_0805_H=0.6mm_Reflow	Res2_1, Res2	26
Bouton	WS-TASV SMD Tact Switch 6X6 mm	S1, S2, S3	430182043816	CMP-1452-00013-1	3
450404015514	WS-SLSU Mini Slide Switch, Same Side Connection Side Push, SPDT, 5VDC, 300mA, 240g	SW1	450404015514	450404015514	1
5019	Test Point, 1 Position SMD, RoHS, Tape and Reel	TP1, TP2, TP3, TP4, TP5, TP6, TP7, TP8, TP9	KSTN-5019_V	CMP-1672-00008-1	9
5011	Test Point, Black, Through Hole, RoHS, Bulk	TP10, TP11, TP12, TP13	KSTN-5011_V	CMP-1672-00003-4	4
IRM-03-3.3	No Description Available	U1	IRM-03_MWU	IRM-03-3.3	1
ESP32-C3-WROOM-02	Bluetooth, WiFi 802.11b/g/n, Bluetooth v5.0 Transceiver Module 2.412GHz ~ 2.484GHz PCB Trace Surface Mount	U2	XCVR_ESP32-C3-WROOM-02	ESP32-C3-WROOM-02	1
6 pins Header		X1, X6, X11	6_pins_socket_header	6 pins Header	3
CurrentTxIR		X2	61300211121	Jumper	1
Current+5V		X3	61300211121	Jumper_1	1
691236510002	Serie 2365 - 5.08 mm Horizontal Cable Entry Modular with Rising Cage Clamp WR-TBL, 2 pin	X4	691236510002	CMP-1502-02613-1	1
CON_PRISE_C		X5	CON_PRISE_C	CON_PRISE_C	1
Current ESP32		X7, X8, X9, X10	61300211121	Jumper_2	4









SW1

R11

C7 C8

R13

TP6

R20

TP7

U2

R12

C10

N2

Q5

C6

C5

C11

C9

C12

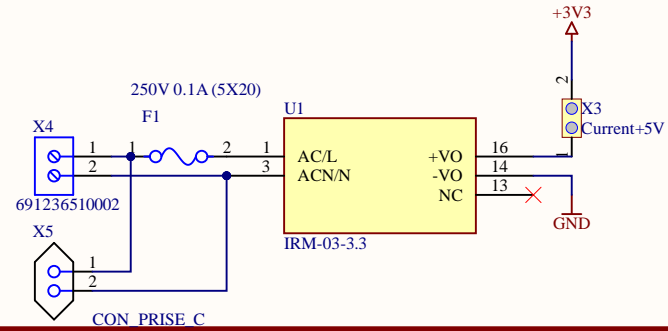
R22

R21

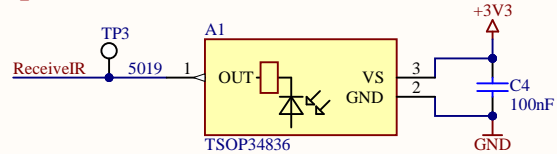
S1

X5

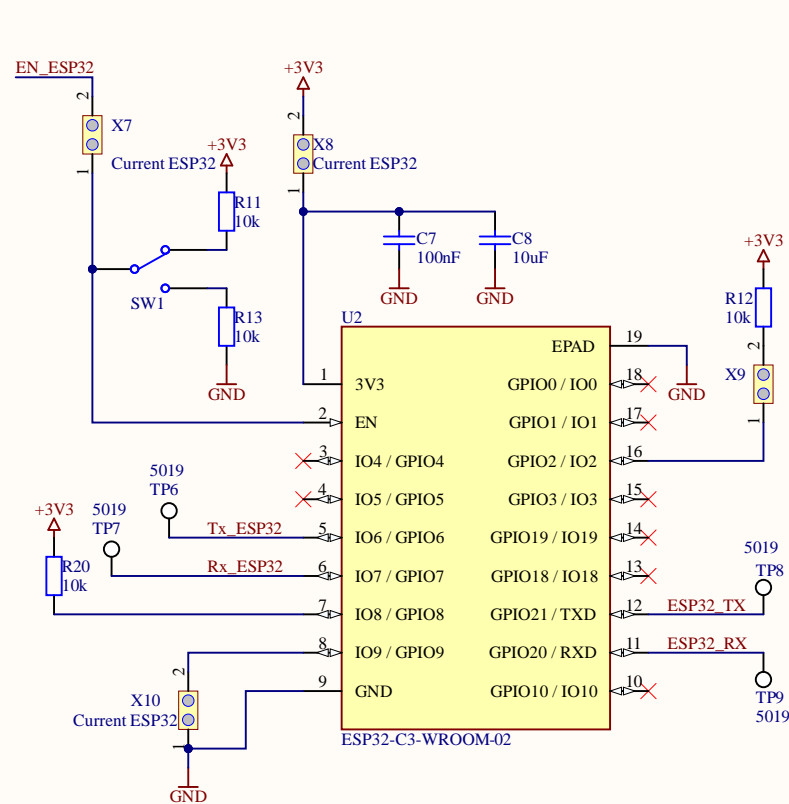
## Convertisseur 230VAC - 3V3DC



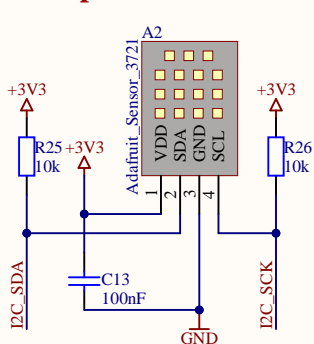
## Récepteur IR



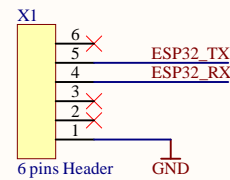
## Module WIFI - ESP32 C3



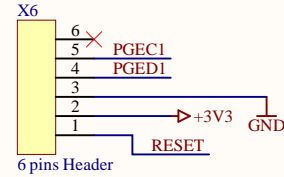
## Capteur température & humidité



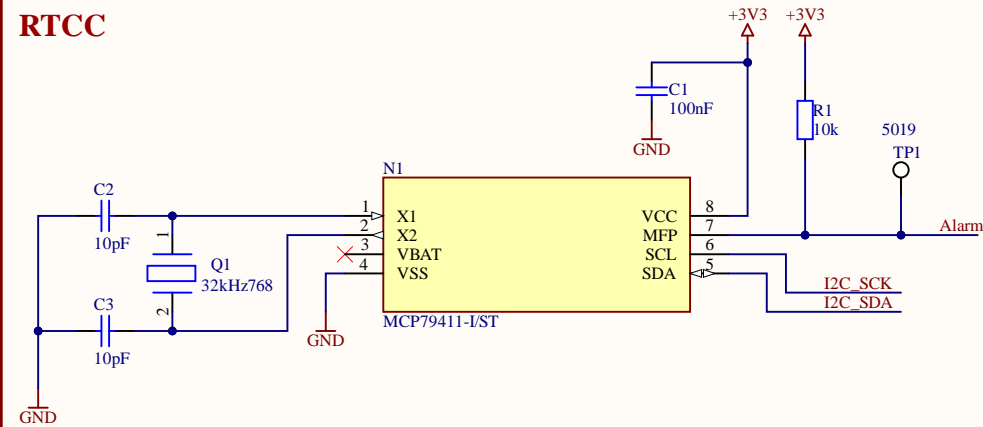
## Programmeur ESP32



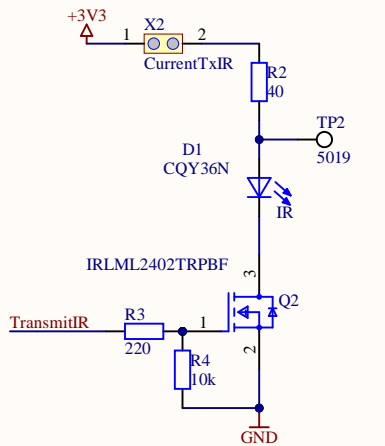
## Programmeur $\mu\text{C}$



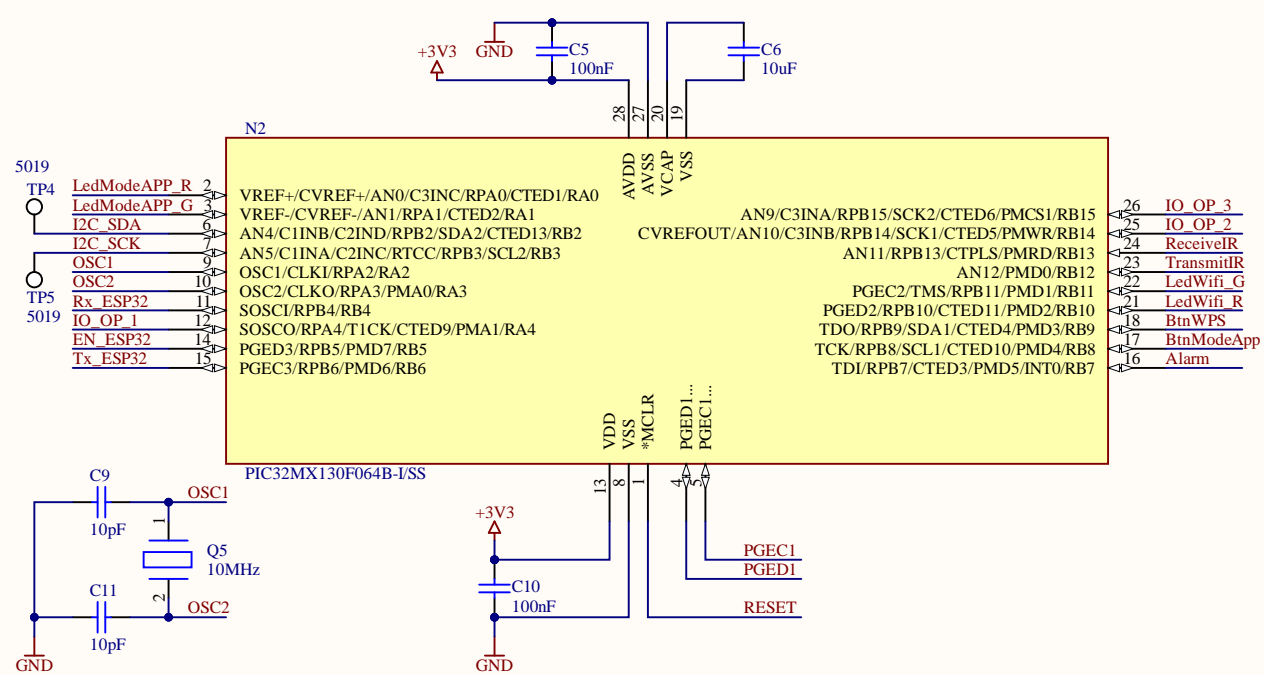
## RTCC



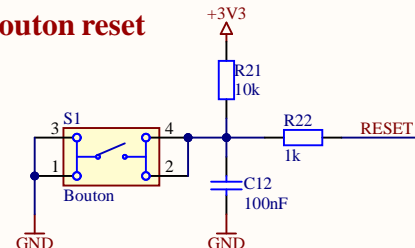
## Émetteur IR



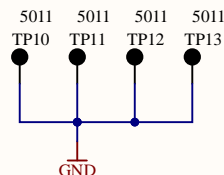
## Microcontrôleur PIC32



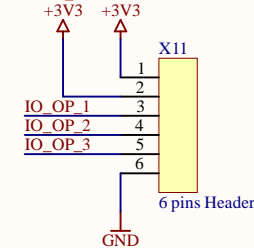
## Bouton reset



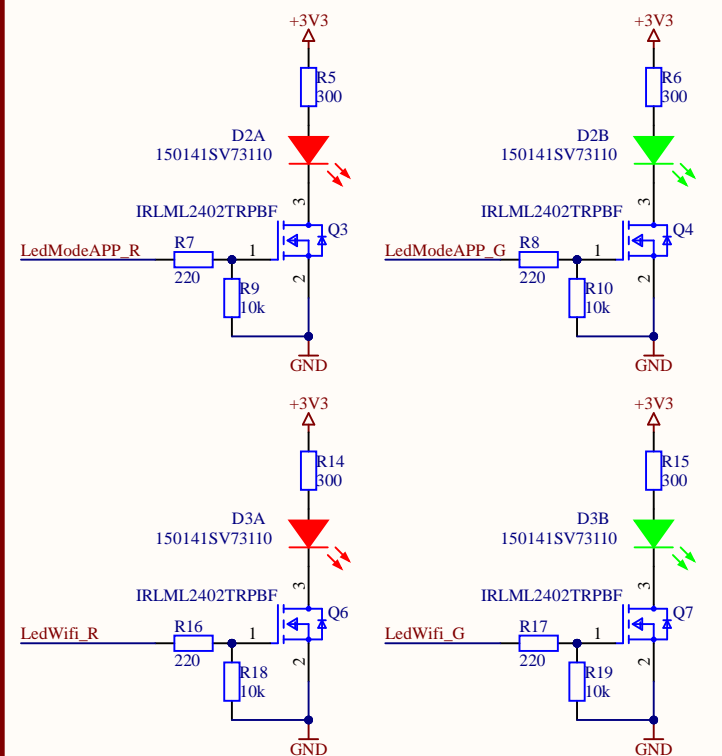
## GND - Mesures



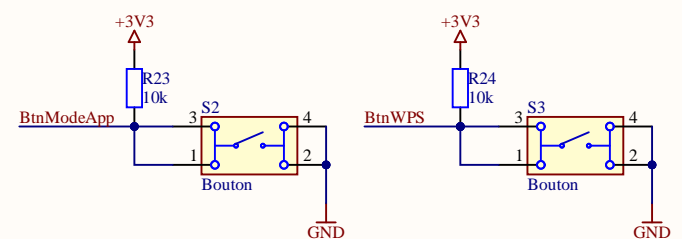
## IO optionnels




## Interface Homme Machine - LEDs



## Interface Homme Machine - Boutons



Fichier : 2209B_CommandeIRDomotique.SchDoc		Modif.	a	
Date : 16.06.2023	Version : B		b	
Heure : 05:22:58	Auteur : Einar Farinas		c	
			d	
 Avenue Recordon 1 1004 Lausanne Switzerland	Projet : 2209B_CommandeIRDomotique.PrjPcb			No :
				Nb feuilles 1
Chemin : C:\Users\veinfarinasa\OneDrive - Education Vaud\2 Annee\PROJ\2209x_CommandeIRDomotique\2209B\hard\2209B_CommandeIRDomotique.SchDoc				

```

1  /*****
2  MPLAB Harmony Application Source File
3
4  Company:
5  Microchip Technology Inc.
6
7  File Name:
8  AM2320.c
9  *****/
10
11  //-----|
12  //Réaliser par : Einar Farinas |
13  //-----|
14
15
16
17  // ****
18  // ****
19  // Section: Included Files
20  // ****
21  // ****
22
23  #include "app.h"
24  #include <stdlib.h>
25  #include "Mc32_I2cUtilCCS.h"
26  #include "AM2320.h"
27
28
29
30  // ****
31  // ****
32  // Section: Global Data Definitions
33  // ****
34  // ****
35
36  //Variables pour l'attente de reponse de l'esp32
37  //bool waitForResponseESP32;
38  //uint16_t consigneToWaitForResponseESP32;
39
40  //Config SOFTAP '!' signifie la fin
41  //Pas utilisé mais utile au besoin
42  //int8_t ssidEsp32 [SIZE_OF_SSID] ={
43  //    'E', 's', 'p', '_', 'A', 'P', '!'
44  //};
45
46  // ****
47  // ****
48  // Section: Functions
49  // ****
50  // ****
51
52  // Init I2C pour le Capteur de température & humidité
53  void AM2320_init(void)
54  {
55      bool Fast = true;
56      i2c_init( Fast );
57  }
58  void AM2320_WakeUp(void)
59  {
60      bool ack;
61      do
62      {
63          i2c_start();
64          ack = i2c_write(AM2320_ADR_W);
65      }while(ack == true);
66      delay_ms(1);
67      i2c_stop();
68      delay_ms(10);
69  }
70  void AM2320_ReadTempHum(s_AM2320 *values)
71  {
72      bool ack = false;
73      uint8_t tempval[4] = {0,0,0,0};

```

```

74
75     i2c_start();
76     ack = i2c_write(AM2320_ADR_W);
77     i2c_write(AM2320_R);
78     i2c_write(0x00);
79     i2c_write(0x04);
80     i2c_stop();
81     delay_ms(2);
82
83     i2c_start();
84     ack = i2c_write(AM2320_ADR_R);
85     delay_us(30);
86
87     tempval[0] = i2c_read(1);
88     tempval[1] = i2c_read(1);
89
90     values->Humidity.val8b.msb = i2c_read(1);
91     values->Humidity.val8b.lsb = i2c_read(1);
92     values->Temp.val8b.msb = i2c_read(1);
93     values->Temp.val8b.lsb = i2c_read(1);
94
95     tempval[2] = i2c_read(1);
96     tempval[3] = i2c_read(0);
97     i2c_stop();
98
99
100
101 }
102 void AM2320_status(uint8_t *Status)
103 {
104     i2c_start();
105     i2c_write(AM2320_ADR_W);
106     i2c_write(AM2320_R);
107     i2c_write(0x0F);
108     i2c_write(0x01);
109     i2c_stop();
110     delay_ms(2);
111
112     i2c_start();
113     i2c_write(AM2320_ADR_R);
114     delay_us(30);
115     *Status = i2c_read(0);
116     i2c_stop();
117     delay_ms(100);
118 }
119 void delay_lus(void)
120 {
121     delay_cycle();
122     delay_cycle();
123     delay_cycle();
124     delay_cycle();
125     delay_cycle();
126     delay_cycle();
127     delay_cycle();
128     delay_cycle();
129     delay_cycle();
130     delay_cycle();
131     delay_cycle();
132     delay_cycle();
133     delay_cycle();
134 }
135 void delay_us(unsigned int us)
136 {
137     uint16_t i = 0;
138     for(i = 0; i < us; i++)
139     {
140         delay_lus();
141     }
142 }
143 void delay_ms(unsigned int ms)
144 {
145     uint16_t i;
146

```

```

147     for(i=1;i<=ms;i++)
148     {
149         // 1ms = 1000us
150         delay_us(1000);
151     }
152 }
153 unsigned short crc16(unsigned char *ptr, unsigned char len)
154 {
155     uint8_t i = 0;
156     uint16_t crc =0xFFFF;
157     while(len--)
158     {
159         crc ^=*ptr++;
160         for(i=0;i<8;i++)
161         {
162             if(crc & 0x01)
163             {
164                 crc>>=1;
165                 crc^=0xA001;
166             }else
167             {
168                 crc>>=1;
169             }
170         }
171     }
172     return crc;
173 }
174 /*****
175 End of File
176 */
177

```

```

1  /*****
2  MPLAB Harmony Application Header File
3
4  Company:
5  Microchip Technology Inc.
6
7  File Name:
8  gestEsp32.h
9  *****/
10
11  //-----|
12  //Réaliser par : Rodrigo Lopes dos Santos |
13  //-----|
14
15
16
17  #ifndef _AM2320_H
18  #define _AM2320_H
19
20
21
22  // ****
23  // ****
24  // Section: Included Files
25  // ****
26  // ****
27
28  #include <stdint.h>
29  #include <stdbool.h>
30  #include <stddef.h>
31  #include <stdlib.h>
32  #include <stdio.h>
33  #include "app.h"
34  #include "system_config.h"
35  #include "system_definitions.h"
36
37
38
39  // DOM-IGNORE-BEGIN
40  #ifdef __cplusplus // Provide C++ Compatibility
41
42  extern "C" {
43
44  #endif
45
46  // ****
47  // ****
48  // Section: Constantes
49  // ****
50  // ****
51  #define AM2320_ADR_W          0xB8
52  #define AM2320_ADR_R          0xB9
53  #define AM2320_R              0x03
54  #define AM2320_W              0x10
55  #define HIGH_HUMIDITY         0x00
56  #define LOW_HUMIDITY          0x01
57  #define HIGH_TEMP             0x02
58  #define LOW_TEMP              0x03
59
60  //255.255.255.255 (12) + les point = 3 (15)
61  #define SIZE_MAX_CHAR_OF_IP_ADRESS 0x0F
62  #define delay_cycle() __asm__( "nop" )
63
64
65  // ****
66  // ****
67  // Section: Type Definitions
68  // ****
69  // ****
70
71  //ENUMERATION
72  //typedef enum
73  //{

```

```

74 // e_CMD_AT = 0,
75 // e_CMD_AT_RST = 1
76 //
77 //}e_CMD;
78
79 // Union
80 typedef union {
81     uint16_t val16b;
82     struct {uint8_t lsb;
83             uint8_t msb;} val8b;
84 } U_16bits;
85
86 //STRUCTURE
87 typedef struct
88 {
89     U_16bits Temp;
90     U_16bits Humidity;
91 }s_AM2320;
92
93
94 //UNION
95
96
97
98 // *****
99 // *****
100 // Section: Function prototype
101 // *****
102 // *****
103 //fonction d'initialisation
104 void AM2320_init(void);
105 void AM2320_WakeUp(void);
106 void AM2320_ReadTempHum(s_AM2320 *values);
107 void AM2320_status(uint8_t *Status);
108 unsigned short crc16(unsigned char *ptr, unsigned char len);
109 unsigned short crc16(unsigned char *ptr, unsigned char len);
110
111 void delay_lus(void);
112 void delay_us(unsigned int us);
113 void delay_ms(unsigned int ms);
114
115 #endif /* _AM2320_H */
116
117 //DOM-IGNORE-BEGIN
118 #ifdef __cplusplus
119 }
120 #endif
121 //DOM-IGNORE-END
122
123 /*****
124 End of File
125 */

```



```

1  /*****
2  MPLAB Harmony Application Source File
3
4  Company:
5      Microchip Technology Inc.
6
7  File Name:
8      app.c
9
10 Summary:
11     This file contains the source code for the MPLAB Harmony application.
12
13 Description:
14     This file contains the source code for the MPLAB Harmony application. It
15     implements the logic of the application's state machine and it may call
16     API routines of other MPLAB Harmony modules in the system, such as drivers,
17     system services, and middleware. However, it does not call any of the
18     system interfaces (such as the "Initialize" and "Tasks" functions) of any of
19     the modules in the system or make any assumptions about when those functions
20     are called. That is the responsibility of the configuration-specific system
21     files.
22 *****/
23
24 //-----|
25 //Réaliser par : Rodrigo Lopes dos Santos |
26 //-----|
27 //
28 // Modifié par : Einar Farinas Arze le 14/06/2023
29 //
30 //
31 // ****
32 // ****
33 // Section: Included Files
34 // ****
35 // ****
36
37 #include "app.h"
38
39 //Pour la gestion de l'uart
40 #include "gestUart.h"
41 //Pour la gestion de l'esp32
42 #include "gestEsp32.h"
43 //Pour la gestion de la rtc
44 #include "Mc32_gestI2cRtcc.h"
45 //Pour la gestion de la eeprom
46 #include "Mc32gestI2cSeeprom.h"
47 //Pour la gestion de l'infrarouge
48 #include "gestIR.h"
49
50 #include "peripheral/ic/plib_ic.h"
51 #include "peripheral/tmr/plib_tmr.h"
52
53 // Ajout par Einar Farinas
54 //=====
55 #include "Mc32NVMUtil.h" // Pour la sauvegarde des trames dans la flash du uC
56 #include "AM2320.h" // Librairie pour le capteur de température&humidité
57 //=====
58
59 // ****
60 // ****
61 // Section: Global Data Definitions
62 // ****
63 // ****
64
65 //APP_STATES state; --> Indique l'état de la machine d'état
66 //bool modeWPS; --> Indique si le mode WPS à été activé (Possible dans mode RUN)
67 //APP_MODE mode; --> Indique le mode apprentissage ou run
68 //APP_SAUVEGARDE sauvegardeData; --> Indique sauvegarde ou non trame IR reçu
69 //bool modeWasChanged; --> Indique un changement de mode
70 // --> Run à apprentissage et visversa
71 APP_DATA appData;
72
73 //Stockage de(s) adresse(s) IP(s)

```

```

74 s_ESP32_IPs adressIPEsp32;
75
76 //Variables pour l'attente entre deux transmissions I2C
77 bool waitBetweenTwoTransmissionsI2C;
78 uint16_t consigneToWaitBetweenTwoTransmissionsI2C;
79
80 //Variable pour le stockage de valeur de la réception IR
81 uint16_t dataIR [SIZE_OF_BUFFER_IR];
82 uint16_t timeOfDataIR [SIZE_OF_BUFFER_IR];
83 uint8_t lengthOfDataIR;
84 //uint16_t dataIR_1 [SIZE_OF_BUFFER_IR];
85 //uint16_t timeOfDataIR_1 [SIZE_OF_BUFFER_IR];
86 //uint8_t lengthOfDataIR_1;
87 //uint16_t dataIR_2 [SIZE_OF_BUFFER_IR];
88 //uint16_t timeOfDataIR_2 [SIZE_OF_BUFFER_IR];
89 //uint8_t lengthOfDataIR_2;
90
91
92 //uint8_t chxDataIR; // Version A
93
94
95 //Variable qui recuperer quelle intructions effectuer en fonction du temps de la rtc
96 uint8_t intrToDo;
97
98
99 //=====
100 s_AM2320 Cap;
101
102
103
104 //=====
105
106
107 // *****
108 // *****
109 // Section: Application Initialization
110 // *****
111 // *****
112
113 void APP_Initialize ( void )
114 {
115     //Passage à l'init de la machine d'état
116     APP_UpdateState(APP_STATE_INIT);
117     //Clear le mode WPS (désactivé)
118     APP_ClearModeWPS();
119     //Init le mode en run
120     APP_UpdateMode(APP_MODE_RUN);
121     //Initialiser la sauvegarde en attente
122     APP_UpdateSauvegarde(APP_SAVE_WAIT);
123     //Indiquer qu'il y aucun changement de mode
124     APP_ClearModeWasChanged();
125     //Indiquer aucune attente d'i2c
126     waitBetweenTwoTransmissionsI2C = false;
127
128     //Pointer sur le stockage de la premier trame
129     // chxDataIR = 1;
130 }
131
132
133
134 // *****
135 // *****
136 // Section: Application State Machine Functions
137 // *****
138 // *****
139
140 void APP_Tasks ( void )
141 {
142     //DECLARATION
143     //Stockage de trame recu par tcpip
144     int8_t dataTcpIp[SIZE_OF_MSG_TCPIP];
145
146     //Configuration du temps de la RTC

```

```

147 s_Time timeRtc;
148 //Variable qui recupere et met à jour le nombre d'instruction de la eeprom
149 uint8_t nbIntrEeprom;
150
151 //Variable qui récup les datas de la trame TCP pour y intégrer à la eeprom
152 APP_DATA_TCP_EEPROM tcpToEeprom;
153 APP_DATA_TCP_EEPROM getTcpFromEeprom;
154 //Variable pour récup le nombre de bytes dispo dans le socket du esp32
155 static uint32_t sizeofDataSocket = 0;
156
157 //      uint8_t ConnectionState;
158
159 //=====
160 // Variable pour sauvegarder la minnute précédente
161 // utilisé pour controler l'état de connexion au wifi tous les minutes
162 static uint8_t min_1 = 0;
163 //=====
164
165
166 //STATE MACHINE
167 switch ( appData.state )
168 {
169     case APP_STATE_INIT:
170     {
171         //INITIALIZATION
172
173         //Indiquer l'initialisation
174         //En allumant constamment les leds
175         APP_SetLeds(LED_ON);
176
177         //Initialisation de l'UART
178         DRV_USART0_Initialize();
179
180         //Initialisation de la fifo pour UART
181         InitFifoSoft();
182
183         //Initialisation des buffer IR
184         IR_Initialize();
185
186         DRV_IC0_Open();
187         PLIB_IC_ModeSelect(IC_ID_1, IC_INPUT_CAPTURE_DISABLE_MODE);
188
189         //Init I2c et RTCC
190         Rtcc_Init();
191
192         //start RTCC (Oscillateur)
193         Rtcc_Start();
194
195         //Mettre à 0 le nombre d'instuctions
196         //A enlever dans le code final
197         APP_SetNumberOfInstructionEEPROM((uint8_t)(0));
198
199         //Lancement du timer pour temps d'attente reception uart de l'esp32
200         //IMPORTANT : AVANT INIT ESP32
201         DRV_TMR0_Start();
202         // =====
203         //Pour la porteuse de 36 kHz
204         DRV_TMR3_Start();
205         // =====
206
207         //Initialisation de l'ESP32
208         Esp32_Initialize();
209
210         //Définition du mode du wifi
211         Esp32_SetWifiMode(e_SOFTAP_STATION_MODE);
212
213
214         // =====
215         DRV_TMR1_Start();
216
217         // Contrôler si il y a des trames enregistrés dans la flash du uC
218         CheckFlashMemory();
219         // =====

```

```

220
221 //MAJ du nombre d'instruction
222 APP_SetNumberOfInstructionEEPROM(0);
223 //Passage à l'état attente
224 APP_UpdateState(APP_STATE_WAIT);
225 //Des que c'est initialisé, éteindre les leds
226 APP_SetLeds(LED_OFF);
227 //END
228 break;
229 }
230 case APP_STATE_WAIT:
231 {
232 // Rien faire
233 break;
234 }
235 case APP_STATE_IDLE:
236 {
237 //WAIT
238 //Test pour entrée dans les état apprentissage et run
239 //Apprentissage : Rentre dans l'état apprentissage
240 //Run : Test --> WPS / Info TCP / Comparaison trame eeprom avec
Instruction TCP
241 switch(appData.mode)
242 {
243 case APP_MODE_APPRENTISSAGE:
244 {
245 //Passage à l'état apprentissage
246 APP_UpdateState(APP_STATE_APPRENTISSAGE);
247 break;
248 }
249 case APP_MODE_RUN:
250 {
251 //Passage à l'état run
252 //Init : Interface Homme-Machine & désactivation timer et IC
(Apprentissage)
APP_InitModeRun();
//Activation du mode WPS
if(appData.modeWPS)
{
APP_UpdateState(APP_STATE_MODE_WPS);
}

//Est qu'il y a des octect à lire ?
sizeofDataSocket = Esp32_GetSizeOfDataSocket();
if(sizeofDataSocket >= SIZE_OF_MSG_TCPIP)
{
APP_UpdateState(APP_STATE_READ_TCPIP);
}

//Lecture du temps de la RTC
APP_GetTimeRtc(&timeRtc);
//Instruction TCP = Heure actuelle ?
if(APP_CompareTimeRtcWithIntructionsEEPROM(&timeRtc, &intrToDo))
{
//Recup la trame selon IntrToDo
if(CheckSavedCommand(intrToDo))
{
//Réaliser via l'émission IR
APP_UpdateState(APP_STATE_TRANSMIT_IR);
}
}
// Contrôle de l'état de la connexion au wifi toutes les minutes
// car le controle avec Esp32_Connected() prendre 1s à s'executer
if(min_1 == (timeRtc.min - 1))
{
if(Esp32_Connected() == true)
{
LedWifi_GOn();
}
else
{
LedWifi_GOff();
}
}
}
}

```

```

291     }
292     min_1 = timeRtc.min;
293     //     AM2320_status(&State);
294     //     AM2320_WakeUp();
295     //     AM2320_ReadTempHum(&Cap);
296     //     AM2320_status(&State);
297     break;
298 }
299 }
300 //     LedModeAPP_RTogge();
301 //END
302 break;
303 }
304 case APP_STATE_APPRENTISSAGE:{
305     //APPRENTISSAGE
306     //Init : Interface Homme-Machine & activation timer et IC
307     APP_InitModeApprentissage();
308
309
310     // Modifié par Einar Farinas
311     // =====
312     // Selon s'il y a sauvegarde ou non
313     switch (appData.sauvegardeData)
314     {
315         // Si sauvegarde
316         case APP_SAVE_OK:
317         {
318             // Pour sauvegarde qu'une fois pendant le clignotement de la LED
319             if(appData.saveFlag == true)
320             {
321                 appData.saveFlag = false;
322                 //SAUVEGARDE
323                 LedModeAPP_ROff();
324                 //Arreter la réception IR en attente la validation ou non de
325                 //l'utilisateur
326                 APP_StopIC();
327                 //Transfer des datas appris dans le buffer ir de l'app
328                 //lengthOfDataIR = IR_SaveData(&dataIR[0], &timeOfDataIR[0]);
329                 // Sauvegarde de la trame reçue dans la flash du uC
330                 IR_SaveData();
331                 // Pour tester l'envoi de trames IR
332                 GetIRData(1);
333                 TestCnt++;
334                 GetIRData(TestCnt);
335
336                 IR_SendCommandIR();
337             }
338             // Clignotement 3 fois de la LED pour indiquer la sauvegarde
339             if(BlinkLed(LED_MODE_APP_G, NB_BLINK_SAVE) == true)
340             {
341                 //Sortie du mode apprentissage à run
342                 APP_UpdateMode(APP_MODE_RUN);
343
344                 //Indiquer un changement de mode
345                 APP_SetModeWasChanged();
346
347                 //Passage à l'état wait
348                 APP_UpdateSauvegarde(APP_SAVE_WAIT);
349             }
350             break;
351         }
352         // Si pas de sauvegarde
353         case APP_SAVE_NOK:
354         {
355             //Eteindre la led d'apprentissage verte pour indiquer une non
356             //sauvegarde
357             LedModeAPP_GOff();
358
359             //Non sauvegarde des datas (effacement du buffer IR)
360             IR_NotSaveData();
361
362             //Relancement de l'IC pour une nouvelle reception IR
363             APP_StartIC();

```

```

362
363         //Passage à l'état wait
364         APP_UpdateSauvegarde(APP_SAVE_WAIT);
365         break;
366     }
367     case APP_DELETE_ALL:
368     {
369         if(appData.deleteFlag == true)
370         {
371             appData.deleteFlag = false;
372             // Reset le nombre d'instructions à faire
373             APP_SetNumberOfInstructionEEPROM(0);
374             // Effacer la page dans la flash
375             Init_Page();
376             // Reset tableau d'adresse des trames enregistrés
377             ResetTbFlashStart();
378             // Ecriture de la valeur de FlashFlag dans le début de la
379             // mémoire flash
380             // car la page a été effacée
381             CheckFlashMemory();
382         }
383         if(BlinkLed(LED_MODE_APP_RG, NB_BLINK_DELETE) == true)
384         {
385             //Sortie du mode apprentissage à run
386             APP_UpdateMode(APP_MODE_RUN);
387             //Passage à l'état wait
388             APP_UpdateSauvegarde(APP_SAVE_WAIT);
389         }
390         break;
391     }
392     case APP_SAVE_WAIT:
393     {
394         //Est-ce qu'une trame à été reçue ?
395         if(IR_DataIsAvailable())
396         {
397             //Indiquer qu'une trame à été reçue
398             LedModeAPP_ROn();
399         }
400         break;
401     }
402     //passage à l'état attente
403     APP_UpdateState(APP_STATE_WAIT);
404     //END
405     break;
406 }
407 case APP_STATE_TRANSMIT_IR:{
408     //TRANSMIT IR
409
410     //!!! LA EEPROM N'ETANT PAS ASSEZ GRANDE, CHOIX SELON TABLEAU LOCAL
411     //!!!
412
413     //Recup la trame à effectuer dans la eeprom
414     //APP_GetDataInEeprom(&dataIR[0], &timeOfDataIR[0], intrToDo);
415
416     GetIRData(intrToDo);
417     //Si j'ai l'autorisation d'émettre:
418     //Envoie Commande;
419     IR_SendCommandIR();
420
421     //Passage à l'état attente
422     APP_UpdateState(APP_STATE_WAIT);
423     //END
424     break;
425 }
426 case APP_STATE_MODE_WPS:
427 {
428     //MODE WPS
429
430     //Activer le mode WPS
431     if (Esp32_ModeWPS(e_ENABLE_WPS) == true){
432         //Rester tant que je suis pas connecter
433         while(Esp32_Connected() == false){

```

```

433     }
434 }
435
436 //Obtenir les adresses IPs
437 Esp32_GetIpsAdress (&adressIEsp32);
438
439 //Création du serveur TCPIP
440 while(!APP_CreateTcpServer(e_SOCKET_MODE_PASSIF, 300)){
441
442 }
443
444 //Désactiver le mode WPS
445 APP_ClearModeWPS();
446 // Laisser allumée la LED wifi rouge pour indiquer que la connexion
447 // au AP est établie mais l'heure et la date n'est pas mise à jour
448 LedWifi_ROn();
449 //Passage à l'état READ_TIME_SNTP
450 APP_UpdateState(APP_STATE_READ_TIME_SNTP);
451 //END
452 break;
453 }
454 case APP_STATE_READ_TIME_SNTP:{
455     // READ TIME SNTP
456
457     //Obtenir l'heure via SNTP
458     Esp32_GetTimeSNTP (&timeRtc);
459
460     //Mise à jour de l'heure de la RTC
461     APP_SetTimeRtc (&timeRtc);
462     // Éteindre la LED wifi rouge pour indiquer que l'heure et la date
463     // ont été mise à jour
464     LedWifi_ROff();
465     //Des que la connexion est établie, éteindre les leds
466     APP_SetLeds(LED_S_OFF);
467
468     //Passage à l'état WAIT
469     APP_UpdateState(APP_STATE_WAIT);
470     //END
471     break;
472 }
473 case APP_STATE_READ_TCPIP:
474 {
475     //READ INFO TCPIP
476
477     //Si la trame TCP est correcte et a bien été Récup
478     if (Esp32_GetDataFromSocket(&dataTcpIp[0], sizeofDataSocket)){
479
480         //On recup le nombre d'instruction déjà présente
481         nbIntrEeprom = APP_GetNumberOfInstructionEEPROM();
482
483         //On y ajoute +1 pour une nouvelle instruction
484         nbIntrEeprom++;
485
486         //MAJ du nombre d'instruction
487         APP_SetNumberOfInstructionEEPROM(nbIntrEeprom);
488
489         //Traitement de la trame TCP reçu
490         APP_ConvTcpDataToEeprom (&tcpToEeprom, &dataTcpIp[0]);
491
492         //Mettre la trame TCP dans l'eeprom
493         //-1 pour ne pas prendre en compte la nouvelle adresse
494         APP_PutTcpDataInEeprom(nbIntrEeprom, &tcpToEeprom);
495
496         //Recup de la trame TCP dans l'eeprom
497         //Pour test, à enlever dans le vrai programme
498         APP_GetTcpDataInEeprom(nbIntrEeprom, &getTcpFromEeprom);
499     }
500
501     //remise à zero des datas à lire du socket
502     sizeofDataSocket = 0;
503
504     //Passage à l'état WAIT
505     APP_UpdateState(APP_STATE_WAIT);

```

```

505         //END
506         break;
507     }
508     default:
509     {
510         //ETAT DE DEFAULT
511
512         //END
513         break;
514     }
515 }
516 }
517
518
519
520
521 // *****
522 // *****
523 // Section: Functions
524 // *****
525 // *****
526
527 //Fonction servant a changer d'etat
528 void APP_UpdateState ( APP_STATES NewState )
529 {
530     appData.state = NewState;
531 }
532
533 APP_STATES GetAppState(void)
534 {
535     return appData.state;
536 }
537 //Fonction servant à activer le clignotement du mode WPS
538 void APP_SetModeWPS(void){
539     appData.modeWPS = true;
540 }
541
542 //Fonction servant à désactiver le clignotement du mode WPS
543 void APP_ClearModeWPS(void){
544     appData.modeWPS = false;
545 }
546
547 //Fonction servant à retourner s'il faut ou non un clignotement pour mode WPS
548 bool APP_GetModeWPS(void){
549     return (appData.modeWPS);
550 }
551
552 //MAJ du mode (RUN ou APPRENTISSAGE)
553 void APP_UpdateMode(APP_MODE newMode){
554     appData.mode = newMode;
555 }
556
557 //Recuperer le mode actuel pour toggle le mode de RUN à APPRENTISSAGE et visversa
558 APP_MODE APP_GetCurrentMode(void){
559     return (appData.mode);
560 }
561
562 //MAJ s'il faut ou non sauvegarder la trame IR
563 void APP_UpdateSauvegarde(APP_SAUVEGARDE newSave){
564     appData.sauvegardeData = newSave;
565 }
566
567 //Activer l'indication de changement de mode (RUN à APPRENTISSAGE)
568 void APP_SetModeWasChanged(void){
569     appData.modeWasChanged = true;
570 }
571
572 //Désactiver l'indication de changement de mode (RUN à APPRENTISSAGE)
573 void APP_ClearModeWasChanged(void){
574     appData.modeWasChanged = false;
575 }
576
577 //Allumer ou éteindre les quatre leds

```



```

578 void APP_SetLeds(bool onOff){
579     if(onOff == true){
580         LedModeAPP_GOn();
581         LedModeAPP_ROn();
582         LedWifi_GOn();
583         LedWifi_ROn();
584     }
585     else{
586         LedModeAPP_GOff();
587         LedModeAPP_ROff();
588         LedWifi_GOff();
589         LedWifi_ROff();
590     }
591 }
592
593 //Mettre à jour les registres de temps de la RTC
594 void APP_SetTimeRtc(s_Time *time){
595     //Arrete du timer utilisé dans le mode WPS (Pour las interrompre l'I2C)
596     DRV_TMR0_Stop();
597
598     Rtcc_SetTime(time->sec, time->min, time->hour, time->day, time->date, time->month,
599                 time->year);
600
601     //Relancement du timer après communication I2C
602     DRV_TMR0_Start();
603 }
604
605 //Créer un Server TCP via les fonction de gestEsp32.c
606 bool APP_CreateTcpServer(e_SOCKET_MODE modeSocket, uint16_t timeOutServer_s){
607     bool createServer = false;
608     bool socketMode = false;
609     bool timeOut = false;
610
611     Esp32_SetTCPServer(0);
612     //Création du serveur TCP
613     createServer = Esp32_SetTCPServer(1);
614
615     //Définir le mode du socket tcpip
616     //Passif - Conserve les données du socket dans un tampon interne
617     //En attente d'une demande de l'hôte (PIC32)
618     //Actif (Par défaut) - Envoie le socket directement à l'hôte
619     socketMode = Esp32_SetSocketMode(modeSocket);
620
621     //Définir le timeout du serveur TCP (300s = 5m)
622     timeOut = Esp32_SetTimeOutServerTCP(timeOutServer_s);
623
624     //Retourner true si tout es bon
625     if (createServer && socketMode && timeOut){
626         return (true);
627     }
628     else{
629         return (false);
630     }
631 }
632
633 //Retourner combien d'instruction sont disponible dans la eeprom
634 uint8_t APP_GetNumberOfInstructionEEPROM(void){
635     uint8_t nbOfInstructions;
636
637     //Arrete du timer 1 utilisé dans le mode WPS (Pour pas interrompre l'I2C)
638     DRV_TMR0_Stop();
639
640     //Lecture du nombre d'instructions
641     I2C_ReadOneByteEEPROM((void*)(&nbOfInstructions),
642                           ADRESSE_EEPROM_NB_OF_INSTRUCTIONS);
643
644     //Relancement du timer après communication I2C
645     DRV_TMR0_Start();
646
647     return(nbOfInstructions);
648 }

```

```

649 //MAJ du nombre d'instructions disponible sur la eeprom
650 void APP_SetNumberOfInstructionEEPROM(uint8_t newNbOfIntructions){
651     //Arrete du timer utilisé dans le mode WPS (Pour las interrompre l'I2C)
652     DRV_TMR0_Stop();
653
654     //Écriture du nouveau nombre d'instructions
655     I2C_WriteOneByteSEEPROM((void*)(&newNbOfIntructions),
656                             ADRESSE_EEPROM_NB_OF_INSTRUCTIONS);
657
658     //Relancement du timer après communication I2C
659     DRV_TMR0_Start();
660 }
661 //Convertir la trame TCP en data pour la eeprom
662 void APP_ConvTcpDataToEeprom (APP_DATA_TCP_EEPROM *conv, int8_t *buff){
663     uint8_t i;
664     uint8_t val;
665
666     //Déplacement et récup au numéro de trame
667     buff += 6;
668     conv->typeTrame = (*buff - ZERO_ASCII);
669
670     //Recup des parametre mois date heure et minut
671     for (i = 0; i < 4; i++){
672         buff += 2;
673         val = ((*buff - ZERO_ASCII)*10);
674         buff ++;
675         val += (*buff - ZERO_ASCII);
676         switch (i){
677             case 0 :{
678                 conv->month = val;
679                 break;
680             }
681             case 1 :{
682                 conv->date = val;
683                 break;
684             }
685             case 2 :{
686                 conv->hour = val;
687                 break;
688             }
689             case 3 :{
690                 conv->min = val;
691                 break;
692             }
693         }
694     }
695
696     //END
697 }
698
699 //Mise des data tcp reçu et traité dans la eeprom
700 void APP_PutTcpDataInEeprom (uint8_t offset, APP_DATA_TCP_EEPROM *conv){
701     uint32_t newOffsetAdress;
702     uint8_t tb[NUMBER_OF_PARAMETERS_FROM_TCP_DATA];
703
704     //Arrete du timer utilisé dans le mode WPS (Pour las interrompre l'I2C)
705     DRV_TMR0_Stop();
706
707     //Si trame 1, alors positionner sur la trame 0 (Meme fonctionnement qu'un tableau)
708     offset -= 1;
709
710     //Mise des informations dans un tableau
711     tb[0] = conv->typeTrame;
712     tb[1] = conv->month;
713     tb[2] = conv->date;
714     tb[3] = conv->hour;
715     tb[4] = conv->min;
716
717     //+1 pour éviter le byte ou il y a le nombre d'instructions
718     //offset * 5 (NUMBER..) --> 1 trame = 5 bytes donc s'il ya deja une
719     //il y aura un nouveau offset de 5 bytes pour ne pas écraser les trames existante
720     newOffsetAdress = (uint32_t)((offset * NUMBER_OF_PARAMETERS_FROM_TCP_DATA) + 1);

```

```

721
722 //écriture tout d'un coup
723 //Mise des différents parametre selon l'ordre
724 //Numéro trame / Mois / Date / Heure / Minute
725 I2C_WriteSEEPROM((void*)(&tb[0]), newOffsetAdress,
NUMBER_OF_PARAMETERS_FROM_TCP_DATA);
726
727
728 //Relancement du timer après communication I2C
729 DRV_TMR0_Start();
730
731 //END
732 }
733
734 void APP_RePlaceAdressNewDataInEeprom(uint8_t nbOfData, uint8_t* adress){
735     uint8_t i;
736     uint8_t endData;
737
738     //Placer l'offset sur l'adresse après le nombre de trame
739     *adress = (ADRESSE_EEPROM_NB_OF_DATA + 1);
740
741     //Se déplacer dans la eeprom jusqu'à l'emplacement de stockage
742     for(i = 0; i < nbOfData; i++){
743         //Lecture de l'adresse de fin de la trame
744         I2C_ReadOneByteSEEPROM((void*)(&endData), *adress);
745
746         //Placer l'offset à la fin de la trame +1 pour le debut d'une nouvelle
747         //Si c'est possible (Depassement de l'écriture)
748         *adress = (endData + 1);
749     }
750 }
751
752 //Met la trame apprise dans la eeprom
753 void APP_PutIrDataInEeprom(uint8_t sizeData){
754     uint8_t nbOfData;
755     uint8_t offset;
756     uint8_t endData;
757
758     //Recuperer le nombre de datas dispo
759     nbOfData = APP_GetNumberOfDataEEPROM();
760
761     //Lire les datas dispo et remplacer l'adresse à l'espace libre
762     APP_RePlaceAdressNewDataInEeprom(nbOfData, &offset);
763
764     //Placer fin de la trame et la trame dans la eeprom
765     //Si c'est possible (*3 car 1 bytes/data + 2 bytes/temps data)
766     if ((offset + (sizeData * 3)) < ADRESSE_EEPROM_MAX){
767         //Mettre à jour le nombre de data
768         APP_SetNumberOfDataEEPROM(nbOfData + 1);
769
770         //Mettre dans la eeprom la fin des datas
771         endData = (sizeData*3) + offset;
772         I2C_WriteOneByteSEEPROM((void*)(&endData), offset);
773
774         //Mettre les datas
775         I2C_WriteSEEPROM((void*)(&dataIR[0]), (offset + 1), sizeData);
776         I2C_WriteSEEPROM((void*)(&timeOfDataIR[0]), (offset + 2), (sizeData*2));
777     }
778 }
779
780 void APP_ClearIntructionInEeprom(uint8_t instructionToClear){
781     uint32_t offset;
782     uint8_t tb[NUMBER_OF_PARAMETERS_FROM_TCP_DATA] = {0,0,0,0,0};
783
784     //Se positionner sur le début de l'instruction
785     offset = (uint32_t)((instructionToClear * NUMBER_OF_PARAMETERS_FROM_TCP_DATA) + 1);
786
787     //Clears les datas
788     I2C_WriteSEEPROM((void*)(&tb[0]), offset, NUMBER_OF_PARAMETERS_FROM_TCP_DATA);
789 }
790
791 //Recup la trame choisi dans la eeprom

```

```

792 void APP_GetDataInEeprom(uint8_t* data, uint16_t* timeOfData, uint8_t nbData){
793     uint8_t nbOfData;
794     uint8_t sizeOfDataToRead;
795     uint8_t offset;
796
797     //Recuperer le nombre de datas dispo
798     nbOfData = APP_GetNumberOfDataEEPROM();
799
800     //Lire les datas dispo et remplacer l'adresse à l'espace libre
801     APP_RePlaceAdressNewDataInEeprom(nbOfData, &offset);
802
803     //Recuperer la taille des datas à lire
804     I2C_ReadOneByteSEEPROM((void*)(&sizeOfDataToRead), offset);
805
806     //Recup des datas
807     I2C_ReadSEEPROM((void*)(data), (offset+1), (sizeOfDataToRead/3));
808
809     //Recup le temps des datas
810     I2C_ReadSEEPROM((void*)(timeOfData), (offset+1+(sizeOfDataToRead/3)), (
811         sizeOfDataToRead/3)*2);
812 }
813
814 //Retourner combien de trame sont disponible dans la eeprom
815 uint8_t APP_GetNumberOfDataEEPROM(void){
816     uint8_t nbOfData;
817
818     //Arrete du timer utilisé dans le mode WPS (Pour las interrompre l'I2C)
819     DRV_TMR0_Stop();
820
821     //Lecture du nombre de datas
822     I2C_ReadOneByteSEEPROM((void*)(&nbOfData), ADRESSE_EEPROM_NB_OF_DATA);
823
824     //Relancement du timer après communication I2C
825     DRV_TMR0_Start();
826
827     return(nbOfData);
828 }
829
830 //MAJ du nombre d'instructions disponible sur la eeprom
831 void APP_SetNumberOfDataEEPROM(uint8_t newNbOfData){
832     //Arrete du timer utilisé dans le mode WPS (Pour les interrompre l'I2C)
833     DRV_TMR0_Stop();
834
835     //Écriture du nouveau nombre de datas
836     I2C_WriteOneByteSEEPROM((void*)(&newNbOfData), ADRESSE_EEPROM_NB_OF_DATA);
837
838     //Relancement du timer après communication I2C
839     DRV_TMR0_Start();
840 }
841
842 //recuperer les data tcp de la eeprom
843 void APP_GetTcpDataInEeprom (uint8_t offset, APP_DATA_TCP_EEPROM *conv){
844     uint32_t newOffsetAdress;
845     uint8_t tb[NUMBER_OF_PARAMETERS_FROM_TCP_DATA];
846
847     //Arrete du timer utilisé dans le mode WPS (Pour las interrompre l'I2C)
848     DRV_TMR0_Stop();
849
850     //Si trame 1, alors positionner sur la trame 0 (Meme fonctionnement qu'un tableau)
851     offset -= 1;
852
853     //+1 pour éviter le byte ou il y a le nombre d'instructions
854     //offset * 5 (NUMBER_..) --> 1 trame = 5 bytes donc s'il ya deja une
855     //il y aura un nouveau offset de 5 bytes pour ne pas écraser les trames existante
856     newOffsetAdress = (uint32_t)((offset * NUMBER_OF_PARAMETERS_FROM_TCP_DATA) + 1);
857
858     //Lire tout d'un coup
859     //Récup des différents paramètres selon l'ordre
860     //Numéro trame / Mois / Date / Heure / Minute
861     I2C_ReadSEEPROM((void*)(&tb[0]), newOffsetAdress,
862         NUMBER_OF_PARAMETERS_FROM_TCP_DATA);
863 }

```

```

863 //Transfer des datas
864 conv->typeTrame = tb[0];
865 conv->month = tb[1];
866 conv->date = tb[2];
867 conv->hour = tb[3];
868 conv->min = tb[4];
869
870 //Relancement du timer après communication I2C
871 DRV_TMR0_Start();
872
873 //END
874 }
875
876 //Réalise un temps d'attente entre deux transmissions I2C
877 void APP_WaitBetweenTwoTransmissionsI2C_ms(uint16_t timeToWait_ms){
878 //Relancement du timer après communication I2C
879 DRV_TMR0_Start();
880
881 //Fixer le temps d'attente
882 consigneToWaitBetweenTwoTransmissionsI2C = timeToWait_ms;
883
884 //Debut de l'attente de reponse de l'esp32
885 waitBetweenTwoTransmissionsI2C = true;
886
887 //Rester ici temps que le temps c'est pas écouler
888 while(waitBetweenTwoTransmissionsI2C){
889 }
890
891 //Arrete du timer utilisé dans le mode WPS (Pour las interrompre l'I2C)
892 DRV_TMR0_Stop();
893 //END
894 }
895
896 //Recuprer l'indicateur s'il faut démarrer une attente entre deux
897 //communication d'I2C
898 bool APP_GetWaitBetweenTwoTransmissionsI2C(void){
899 return (waitBetweenTwoTransmissionsI2C);
900 }
901
902 //Mettre à jour l'indicateur d'attente entre deux communications I2C
903 void APP_ClearWaitBetweenTwoTransmissionsI2C(void){
904 waitBetweenTwoTransmissionsI2C = false;
905 }
906
907 //Recuperer la consigne, temps d'attente, entre deux communications I2C
908 uint16_t APP_GetConsigneToWaitBetweenTwoTransmissionsI2C(void){
909 return (consigneToWaitBetweenTwoTransmissionsI2C);
910 }
911
912 //Lecture du temps de la rtc
913 void APP_GetTimeRtc(s_Time *time){
914 //Arrete du timer utilisé dans le mode WPS (Pour las interrompre l'I2C)
915 DRV_TMR0_Stop();
916
917 //lecture du temps
918 time->month = Rtcc_ReadMonth();
919 time->date = Rtcc_ReadDate();
920 time->hour = Rtcc_ReadHour();
921 time->min = Rtcc_ReadMin();
922 time->sec = Rtcc_ReadSec();
923 time->year = Rtcc_ReadYear();
924
925 //Relancement du timer après communication I2C
926 DRV_TMR0_Start();
927
928 //END
929 }
930
931 //Comparer le temps de la rtc avec les instructions de la eeprom
932 bool APP_CompareTimeRtcWithInstructionsEEPROM(s_Time *time, uint8_t *instructToDo){
933 uint8_t nbOfInstructions, i;
934 APP_DATA_TCP_EEPROM timeIntru;
935 bool doTask = false;

```

```

936
937 //Mettre l'instruction $ faire à 0 si aucun type de trame ne doit etre fait
938 *instructToDo = 0;
939
940 //Arrete du timer utilisé dans le mode WPS (Pour las interrompre l'I2C)
941 DRV_TMR0_Stop();
942
943 //On recup le nombre d'instructions déjà présente pour comparer avec le temps
944 nbOfInstructions = APP_GetNumberOfInstructionEEPROM();
945
946 //S'il y a une ou des instructions à lire
947 if(nbOfInstructions > 0){
948     for(i = 1; i < nbOfInstructions + 1; i++){
949         APP_GetTcpDataInEeprom ((i), &timeIntru);
950         if((time->date == timeIntru.date) &&
951            (time->month == timeIntru.month) &&
952            (time->hour == timeIntru.hour) &&
953            (time->min == timeIntru.min) &&
954            (time->sec < 2) &&
955            (time->sec >= 0)) // ajout des secondes pour ne pas envoyer la
                               commande pendant 1 min
956         {
957             APP_ClearIntructionInEeprom((i + 1));
958             doTask = true;
959             *instructToDo = timeIntru.typeTrame;
960         }
961     }
962 }
963
964
965 //Relancement du timer après communication I2C
966 DRV_TMR0_Start();
967
968 return(doTask);
969 //END
970 }
971
972 //Initialisation é faire une seule fois à chaque entre dans le mode en question
973 void APP_InitModeApprentissage(void) {
974     //Le deuxième test sers à éviter lors d'un changement de mode par le bouton
975     //Et que je rentre en même temps dans l'ancien mode d'activer ou désactiver
976     //Certain option non voulu. (passage mode run, entre mode apprentissage,
977     //activation IC1 --> Pas besoin)
978     if((appData.modeWasChanged)&&(appData.mode == APP_MODE_APPRENTISSAGE)){
979         //Indiquer l'entrée dans le mode app
980         LedModeAPPOn();
981         LedModeAPP_GOn();
982         //Clear l'indicateur de changement de mode
983         APP_ClearModeWasChanged();
984         //Lancement de l'IC pour la réception IR
985         APP_StartIC();
986     }
987 }
988
989 //Initialisation é faire une seule fois à chaque entre dans le mode en question
990 void APP_InitModeRun(void) {
991     //Le deuxième test sers à éviter lors d'un changement de mode par le bouton
992     //Et que je rentre en même temps dans l'ancien mode d'activer ou désactiver
993     //Certain option non voulu. (passage mode apprentissage, entre mode run,
994     //désactiver IC1 --> Pas besoin)
995     if((appData.modeWasChanged)&&(appData.mode == APP_MODE_RUN)){
996         //Indiquer la sortie du mode apprentissage
997         LedModeAPPOff();
998         LedAPPOkOff();
999         LedModeAPP_GOff();
1000        LedModeAPP_ROff();
1001        //Clear l'indicateur de changement de mode
1002        APP_ClearModeWasChanged();
1003        //Arret de l'IC pour stoper la réception IR
1004        APP_StopIC();
1005    }
1006 }

```

```

1006 //Start du timer 2 et l'ic1 pour la réception IR
1007 void APP_StartIC (void){
1008     //Lancement du timer lié à l'IC
1009     //    DRV_TMR1_Start();
1010     //Lancement de l'IC pour la réception de la trame IR
1011     //    DRV_IC0_Start();
1012     //    DRV_IC0_Open();
1013
1014     //=====
1015     // Activer l'input capture sur tous les flancs
1016     PLIB_IC_ModeSelect(IC_ID_1, IC_INPUT_CAPTURE_EVERY_EDGE_MODE);
1017
1018     //=====
1019 }
1020
1021 //Stop du timer 2 et l'ic1 pour la réception IR
1022 void APP_StopIC (void){
1023     //Arret de l'IC pour la réception de la trame IR
1024     //    DRV_IC0_Stop();
1025     //    DRV_IC0_Close();
1026     //Arret du timer lié à l'IC
1027     //    DRV_TMR1_Stop();
1028
1029     //=====
1030     // Desactiver l'input capture
1031     PLIB_IC_ModeSelect(IC_ID_1, IC_INPUT_CAPTURE_DISABLE_MODE);
1032
1033     //=====
1034 }
1035
1036 //=====
1037 // Pas utilisé dans cette version (version B)
1038 //=====
1039 //fonction pour transferer la trame à effectuer dans l'interruption pour tx IR
1040 //void APP_GetBufferIRApp(uint16_t *buff, uint16_t *timeOfBuff){
1041 //    uint8_t i;
1042 //
1043 //    for(i = 0; i < (lengthOfDataIR - 1); i ++){
1044 //        {
1045 //            *buff = dataIR[i + 1];
1046 //            *timeOfBuff = timeOfDataIR[i + 1];
1047 //
1048 //            buff ++;
1049 //            timeOfBuff ++;
1050 //        }
1051 //    }
1052 //}
1053 //=====
1054 // Pas utilisé dans cette version (version B)
1055 //=====
1056 //fonctions pour transferer la taille de la trame à effectuer dans l'interruption tx
1057 //IR
1058 //uint8_t APP_GetLengthOfData(void){
1059 //    return (lengthOfDataIR - 1);
1060 //    if(intrToDo == 1){
1061 //        //indiquer qu'il n'y a plu d'instruction à faire
1062 //        intrToDo = 0;
1063 //
1064 //        return (lengthOfDataIR_1 - 1);
1065 //    }
1066 //    else{
1067 //        //indiquer qu'il n'y a plu d'instruction à faire
1068 //        intrToDo = 0;
1069 //
1070 //        return (lengthOfDataIR_2 - 1);
1071 //    }
1072 //}
1073 //=====
1074 //
1075 // Ajout par Einar Farinas

```

```

1070 //=====
1071 // Fonction pour faire clignoter X fois les LEDs
1072 // LED      = choix de la LED à faire clignoter
1073 // NbNlink  = nombre de clignotement
1074 //
1075 // Return :
1076 //      false => si la LED n'as pas fini de clignoter le nombre de fois voulue
1077 //      true  => si la LED a clignoté le nombre de fois voulue
1078 bool BlinkLed(APP_LEDS LED, uint8_t NbBlink)
1079 {
1080     // Compteur pour le temps de clignotement
1081     static uint16_t CntBlink = 0;
1082     // Compteur pour le nombre de fois que la LED a clignoté
1083     static uint8_t CntNbBlink = 0;
1084
1085     // Si la LED n'a pas clignoté le nombre de fois voulu
1086     if(CntNbBlink < NbBlink)
1087     {
1088         // début de la période de clignotement
1089         // Allumer la LED voulue
1090         if(CntBlink == 0)
1091         {
1092             switch(LED)
1093             {
1094                 case LED_MODE_APP_R:
1095                     LedModeAPP_ROn();
1096                     break;
1097                 case LED_MODE_APP_G:
1098                     LedModeAPP_GOn();
1099                     break;
1100                 case LED_MODE_APP_RG:
1101                     LedModeAPP_ROn();
1102                     LedModeAPP_GOn();
1103                     break;
1104                 case LED_WIFI_R:
1105                     LedWifi_ROn();
1106                     break;
1107                 case LED_WIFI_G:
1108                     LedWifi_GOn();
1109                     break;
1110                 case LED_WIFI_RG:
1111                     LedWifi_ROn();
1112                     LedWifi_GOn();
1113                     break;
1114                 case ALL_LEDS:
1115                     LedModeAPP_ROn();
1116                     LedModeAPP_GOn();
1117                     LedWifi_ROn();
1118                     LedWifi_GOn();
1119                     break;
1120                 default:
1121                     break;
1122             }
1123         }
1124         // Si moitié de la période
1125         // éteindre la LED voulue
1126         else if(CntBlink == BLINK_PERIODE / 2)
1127         {
1128             switch(LED)
1129             {
1130                 case LED_MODE_APP_R:
1131                     LedModeAPP_ROff();
1132                     break;
1133                 case LED_MODE_APP_G:
1134                     LedModeAPP_GOff();
1135                     break;
1136                 case LED_MODE_APP_RG:
1137                     LedModeAPP_ROff();
1138                     LedModeAPP_GOff();
1139                     break;
1140                 case LED_WIFI_R:
1141                     LedWifi_ROff();
1142                     break;

```



```

1143         case LED_WIFI_G:
1144             LedWifi_GOff();
1145             break;
1146         case LED_WIFI_RG:
1147             LedWifi_ROff();
1148             LedWifi_GOff();
1149             break;
1150         case ALL_LEDS:
1151             LedModeAPP_ROff();
1152             LedModeAPP_GOff();
1153             LedWifi_ROff();
1154             LedWifi_GOff();
1155             break;
1156         default:
1157             break;
1158     }
1159     // Incrémenter le compteur de clignotement
1160     CntNbBlink++;
1161 }
1162 // Compteur de 0 à la valeur de la période de clignotement
1163 CntBlink = (CntBlink + 1) % BLINK_PERIODE;
1164 return false;
1165 }
1166 // si fin clignotement
1167 else
1168 {
1169     // Reset des compteurs
1170     CntNbBlink = 0;
1171     CntBlink = 0;
1172     return true;
1173 }
1174
1175 }
1176 //=====
1177 //                                Ajout par Einar Farinas
1178 //=====
1179 // Fonction pour set le flag d'effacement
1180 // utilisé pour effacer une seule fois pendant le clignotement des LEDs
1181 void SetDeleteFlag(void)
1182 {
1183     appData.deleteFlag = true;
1184 }
1185 //=====
1186 //                                Ajout par Einar Farinas
1187 //=====
1188 // Fonction pour set le flag de sauvegarde
1189 // utilisé pour sauvegarder une seule fois pendant le clignotement des LEDs
1190 void SetSaveFlag(void)
1191 {
1192     appData.saveFlag = true;
1193 }
1194
1195 /*****
1196 End of File
1197 */
1198

```

```

1  /*****
2  MPLAB Harmony Application Header File
3
4  Company:
5  Microchip Technology Inc.
6
7  File Name:
8  app.h
9
10 Summary:
11 This header file provides prototypes and definitions for the application.
12
13 Description:
14 This header file provides function prototypes and data type definitions for
15 the application. Some of these are required by the system (such as the
16 "APP_Initialize" and "APP_Tasks" prototypes) and some of them are only used
17 internally by the application (such as the "APP_STATES" definition). Both
18 are defined here for convenience.
19 *****/
20
21 //-----|
22 //Réaliser par : Rodrigo Lopes dos Santos |
23 //-----|
24
25
26
27 #ifndef _APP_H
28 #define _APP_H
29
30
31
32 // ****
33 // ****
34 // Section: Included Files
35 // ****
36 // ****
37
38 #include <stdint.h>
39 #include <stdbool.h>
40 #include <stddef.h>
41 #include <stdlib.h>
42 #include "system_config.h"
43 #include "system_definitions.h"
44
45 #include "gestEsp32.h"
46
47
48 // DOM-IGNORE-BEGIN
49 #ifdef __cplusplus // Provide C++ Compatibility
50
51 extern "C" {
52
53 #endif
54
55
56
57 // ****
58 // ****
59 // Section: Type Definitions
60 // ****
61 // ****
62
63 //CONSTANTES
64 #define LEDS_ON 1
65 #define LEDS_OFF 0
66
67 #define ADRESSE_EEPROM_NB_OF_INSTRUCTIONS 0x00
68 #define ADRESSE_EEPROM_NB_OF_DATA 0x10
69 #define ADRESSE_EEPROM_MAX 0x7F
70 // type + mois + date + heure + minute = 5
71 #define NUMBER_OF_PARAMETERS_FROM_TCP_DATA 5
72 #define NUMBER_OF_BYTE_FOR_ONE_PARAMETER 1
73

```

```

74  #define NB_BLINK_SAVE          3
75  #define NB_BLINK_DELETE       10
76  // BLINK_PERIODE x Temps d'execution = 80 x 10 ms == 800 ms
77  #define BLINK_PERIODE         80
78
79  //ENUMERATION
80  typedef enum
81  {
82      /* Application's state machine's initial state. */
83      APP_STATE_INIT=0,
84      APP_STATE_WAIT,
85      APP_STATE_IDLE,
86      APP_STATE_MODE_WPS,
87      APP_STATE_READ_TIME_Sntp,
88      APP_STATE_READ_TCPIP,
89      APP_STATE_TRANSMIT_IR,
90      APP_STATE_APPRENTISSAGE
91
92      /* TODO: Define states used by the application state machine. */
93  } APP_STATES;
94
95  typedef enum
96  {
97      APP_MODE_APPRENTISSAGE = 0,
98      APP_MODE_RUN
99
100  }APP_MODE;
101
102  typedef enum{
103      APP_SAVE_WAIT = 0,
104      APP_SAVE_OK,
105      APP_SAVE_NOK,
106      APP_DELETE_ALL
107  }APP_SAUVEGARDE;
108
109  typedef enum
110  {
111      APP_TYPE_IR =0,
112      APP_TYPE_SENSOR = 1
113  } APP_TYPE_OF_INSTRUCTIONS;
114
115  typedef enum
116  {
117      LED_MODE_APP_R = 0,
118      LED_MODE_APP_G,
119      LED_MODE_APP_RG,
120      LED_WIFI_R,
121      LED_WIFI_G,
122      LED_WIFI_RG,
123      ALL_LEDS,
124  } APP_LEDS;
125
126
127  //STRUCURE
128  typedef struct
129  {
130      APP_STATES state;
131      bool modeWPS;
132      APP_MODE mode;
133      APP_SAUVEGARDE sauvegardeData;
134      bool modeWasChanged;
135      bool deleteFlag;
136      bool saveFlag;
137
138  } APP_DATA;
139
140  typedef struct
141  {
142      uint8_t typeTrame;
143      uint8_t month;
144      uint8_t date;
145      uint8_t hour;
146      uint8_t min;

```

```

147
148 } APP_DATA_TCP_EEPROM;
149
150
151
152
153 //UNION
154
155
156
157
158 // *****
159 // *****
160 // Section: Function prototype
161 // *****
162 // *****
163
164 //fonctions d'initialisation
165 void APP_Initialize(void);
166 void APP_InitModeApprentissage(void);
167 void APP_InitModeRun(void);
168
169 //fonction de la machine d'état
170 void APP_Tasks(void);
171
172 //fonctions utilitaires de la machine d'état
173 void APP_UpdateState (APP_STATES NewState);
174 APP_STATES GetAppState(void);
175 void APP_SetLeds(bool onOff);
176
177 //fonctions pour le temps d'attente I2C
178 void APP_WaitBetweenTwoTransmissionsI2C_ms(uint16_t timeToWait_ms);
179 bool APP_GetWaitBetweenTwoTransmissionsI2C(void);
180 void APP_ClearWaitBetweenTwoTransmissionsI2C(void);
181 uint16_t APP_GetConsigneToWaitBetweenTwoTransmissionsI2C(void);
182
183 //fonctions de la gestion de la machine d'état (interface homme-machine)
184 void APP_UpdateMode(APP_MODE newMode);
185 void APP_UpdateSauvegarde(APP_SAUVEGARDE newSave);
186 APP_MODE APP_GetCurrentMode(void);
187 void APP_SetModeWasChanged(void);
188 void APP_ClearModeWasChanged(void);
189
190 //fonctions pour la gestion du mode WPS
191 void APP_SetModeWPS(void);
192 void APP_ClearModeWPS(void);
193 bool APP_GetModeWPS(void);
194
195 //fonction pour la création d'un serveur tcp
196 bool APP_CreateTcpServer(e_SOCKET_MODE modeSocket, uint16_t timeOutServer_s);
197
198 //fonctions pour communication avec la rtc
199 void APP_SetTimeRtc(s_Time *time);
200 void APP_GetTimeRtc(s_Time *time);
201
202 //Traducteur entre valeur de temps rtc et intruction eeprom
203 bool APP_CompareTimeRtcWithIntructionsEEPROM(s_Time *time, uint8_t *intructToDo);
204
205 //fonctions pour la gestion de la eeprom avec instructions tcp
206 uint8_t APP_GetNumberOfInstructionEEPROM(void);
207 void APP_SetNumberOfInstructionEEPROM(uint8_t newNbOfIntructions);
208 void APP_ConvTcpDataToEeprom (APP_DATA_TCP_EEPROM *conv, int8_t *buff);
209 void APP_PutTcpDataInEeprom (uint8_t offset, APP_DATA_TCP_EEPROM *conv);
210 void APP_GetTcpDataInEeprom (uint8_t offset, APP_DATA_TCP_EEPROM *conv);
211 void APP_ClearIntructionInEeprom(uint8_t instructionToClear);
212
213 //fonctions pour l'entrée et sortie du mode apprentissage (réception IR)
214 void APP_StartIC (void);
215 void APP_StopIC (void);
216
217 //fonctions pour transfer des datas dans l'interruption du timer d'émission IR
218 void APP_GetBufferIRApp(uint16_t *buff, uint16_t *timeOfBuff);
219 uint8_t APP_GetLengthOfData(void);

```

```
220
221 //fonctions pour la gestion de la eeprom avec trame IR
222 void APP_PutIrDataInEeprom(uint8_t sizeData);
223 void APP_GetDataInEeprom(uint8_t* data, uint16_t* timeOfData, uint8_t nbData);
224 void APP_RePlaceAdressNewDataInEeprom(uint8_t nbOfData, uint8_t* adress);
225 uint8_t APP_GetNumberOfDataEEPROM(void);
226 void APP_SetNumberOfDataEEPROM(uint8_t newNbOfData);
227
228
229 bool BlinkLed(APP_LEDS LED, uint8_t NbBlink);
230 void SetDeleteFlag(void);
231 void SetSaveFlag(void);
232
233 #endif /* _APP_H */
234
235 //DOM-IGNORE-BEGIN
236 #ifdef __cplusplus
237 }
238 #endif
239 //DOM-IGNORE-END
240
241 /*****
242 End of File
243 */
244
245
```

```

1  /*****
2  MPLAB Harmony Application Source File
3
4  Company:
5  Microchip Technology Inc.
6
7  File Name:
8  gestEsp32.c
9  *****/
10
11  //-----|
12  //Réaliser par : Rodrigo Lopes dos Santos |
13  //-----|
14
15
16
17  // ****
18  // ****
19  // Section: Included Files
20  // ****
21  // ****
22
23  #include "app.h"
24  #include "gestEsp32.h"
25  #include "gestUart.h"
26  #include "AM2320.h"
27  #include <stdlib.h>
28
29
30
31  // ****
32  // ****
33  // Section: Global Data Definitions
34  // ****
35  // ****
36
37  //Variables pour l'attente de reponse de l'esp32
38  bool waitForResponseESP32;
39  uint16_t consigneToWaitForResponseESP32;
40
41  //Buffer qui stock la réponse reçu
42  int8_t bufferMsgReceiveESP32 [SIZE_BUFFER];
43
44  //Stockage de(s) adresse(s) IP(s)
45  s_ESP32_IPs adressIP;
46
47
48
49  //Config SOFTAP '!' signifie la fin
50  //Pas utilisé mais utile au besoin
51  int8_t ssidEsp32 [SIZE_OF_SSID] ={
52      'E', 's', 'p', '_', 'A', 'P', '!'
53  };
54  int8_t pwdEsp32 [SIZE_OF_SSID] ={
55      '2', '2', '0', '9', 'A', 'P', '!'
56  };
57  int8_t idChannelEsp32 [SIZE_OF_ID_CHANNEL] ={
58      '1', '!'
59  };
60  int8_t numberOfConnectedMaxEsp32 [SIZE_OF_MAX_NB_CONNECTED_AP] ={
61      '1', '!'
62  };
63
64  //PortDuServeurTCP '!' signifie la fin
65  int8_t portServerTCP [SIZE_OF_PORT_SERVER_TCP] ={
66      '3', '3', '3', '!'
67  };
68
69
70
71  //Config du SNTP '!' signifie la fin
72  //0 = timeZone (2 --> +2 = Berne)
73  //1,2,3 = Server SNTP

```

```

74  int8_t param0Sntp [SIZE_OF_0_PARAM_Sntp] ={
75      '2', '!'
76  };
77  int8_t param1Sntp [SIZE_OF_1_PARAM_Sntp] ={
78      '"', 'c', 'n', '.', 'n', 't', 'p', '.', 'o', 'r', 'g', '.', 'c', 'n', '"', '!'
79  };
80  int8_t param2Sntp [SIZE_OF_2_PARAM_Sntp] ={
81      '"', 'n', 't', 'p', '.', 's', 'j', 't', 'u', '.', 'e', 'd', 'u', '.', 'c', 'n',
82      '"', '!'
83  };
84  int8_t param3Sntp [SIZE_OF_3_PARAM_Sntp] ={
85      '"', 'u', 's', '.', 'p', 'o', 'o', 'l', '.', 'n', 't', 'p', '.', 'o', 'r', 'g',
86      '"', '!'
87  };
88
89  //taille d'un message tcp
90  int8_t paramSizeMsgTCP[SIZE_OF_MSG_TCPIP_STR] ={
91      '2', '0', '!'
92  };
93
94
95
96  //Commandes AT
97  int8_t cmdAT[SIZE_OF_CMD_AT] ={
98      'A', 'T', 0x0D, 0x0A
99  };
100  int8_t cmdAtRST [SIZE_OF_CMD_AT_RST] ={
101      'A', 'T', '+', 'R', 'S', 'T', 0x0D, 0x0A
102  };
103  //+1 en prevision d'un parametre
104  int8_t cmdAtCWMODE [(SIZE_OF_CMD_AT_CWMODE + 1)] ={
105      'A', 'T', '+', 'C', 'W', 'M', 'O', 'D', 'E', '=', 0x0D, 0x0A
106  };
107  //+1 en prevision d'un parametre
108  int8_t cmdAtWPS [(SIZE_OF_CMD_AT_WPS + 1)] ={
109      'A', 'T', '+', 'W', 'P', 'S', '=', 0x0D, 0x0A
110  };
111  int8_t cmdAtCIFSR [SIZE_OF_CMD_AT_CIFSR] ={
112      'A', 'T', '+', 'C', 'I', 'F', 'S', 'R', 0x0D, 0x0A
113  };
114  int8_t cmdAtCWJAP [SIZE_OF_CMD_AT_CWMJAP] ={
115      'A', 'T', '+', 'C', 'W', 'J', 'A', 'P', '?', 0x0D, 0x0A
116  };
117  //+50 en provision de plusieurs parametre
118  int8_t cmdAtCWSAP [SIZE_OF_CMD_AT_CWSAP + 50] ={
119      'A', 'T', '+', 'C', 'W', 'S', 'A', 'P', '=', 0x0D, 0x0A
120  };
121  //+10 en provision d'un parametre
122  int8_t cmdAtCIPSERVER [SIZE_OF_CMD_AT_CIPSERVER + 10] ={
123      'A', 'T', '+', 'C', 'I', 'P', 'S', 'E', 'R', 'V', 'E', 'R', '=', 0x0D, 0x0A
124  };
125  int8_t cmdAtCIPSNTPTIME [SIZE_OF_CMD_AT_CIPSNTPTIME] ={
126      'A', 'T', '+', 'C', 'I', 'P', 'S', 'N', 'T', 'P', 'T', 'I', 'M', 'E', '?', 0x0D,
127      0x0A
128  };
129  //+50 en provision de plusieurs parametre
130  int8_t cmdAtCIPSNTPCFG [SIZE_OF_CMD_AT_CIPSNTPCFG + 60] ={
131      'A', 'T', '+', 'C', 'I', 'P', 'S', 'N', 'T', 'P', 'C', 'F', 'G', '=', 0x0D, 0x0A
132  };
133  //+1 en provision d'un parametre
134  int8_t cmdAtCIPRECVMODE[SIZE_OF_CMD_AT_CIPRECVMODE + 1] ={
135      'A', 'T', '+', 'C', 'I', 'P', 'R', 'E', 'C', 'V', 'M', 'O', 'D', 'E', '=', 0x0D,
136      0x0A
137  };
138  int8_t cmdAtCIPRECLEN[SIZE_OF_CMD_AT_CIPRECLEN] ={
139      'A', 'T', '+', 'C', 'I', 'P', 'R', 'E', 'C', 'V', 'L', 'E', 'N', '?', 0x0D, 0x0A
140  };
141  //+50 en provision de plusieurs parametre
142  int8_t cmdAtCIPRECVDATA[SIZE_OF_CMD_AT_CIPRECVDATA + 50] ={
143      'A', 'T', '+', 'C', 'I', 'P', 'R', 'E', 'C', 'V', 'D', 'A', 'T', 'A', '=', 0x0D,
144      0x0A

```

```

142 };
143 //+10 en provision de plusieurs parametre
144 int8_t cmdAtCIPSTO[SIZE_OF_CMD_AT_CIPSTO + 10] ={
145     'A', 'T', '+', 'C', 'I', 'P', 'S', 'T', 'O', '=', 0x0D, 0x0A
146 };
147 //+1 en provision de plusieurs parametre
148 int8_t cmdAtCIPMUX[SIZE_OF_CMD_AT_CIPMUX + 1] ={
149     'A', 'T', '+', 'C', 'I', 'P', 'M', 'U', 'X', '=', 0x0D, 0x0A
150 };
151
152 int8_t cmdAtCIPSTATUS[SIZE_OF_CMD_AT_CIPSTATUS + 1] ={
153     'A', 'T', '+', 'C', 'I', 'P', 'S', 'T', 'A', 'T', 'U', 'S', 0x0D, 0x0A
154 };
155
156
157
158 //Reponse possible de l'ESP32
159 int8_t ResponseOK[2] = {
160     'O', 'K'
161 };
162
163 int8_t ResponseERROR[5] ={
164     'E', 'R', 'R', 'O', 'R'
165 };
166
167
168
169 // *****
170 // *****
171 // Section: Functions
172 // *****
173 // *****
174
175 //initialise l'ESP32 (celui-ci est déjà programmé)
176 void Esp32_Initialize(void){
177     //Aucune response à attendre
178     waitForResponseESP32 = false;
179
180     //Aucune consigne d'attente
181     consigneToWaitForResponseESP32 = 0;
182
183     //Init le buffer de reception
184     Esp32_ClearBufferReceive();
185
186     //Init le buffer des IPs
187     Esp32_ClearAdressIPs(e_SOFTAP_STATION_MODE);
188
189     //Activer le enable de l'esp32
190     EN_ESP32On();
191
192     //Vérifier si les commande AT sont OK
193     //Reste bloquer si les commandes AT sont pas ok
194     while (!Esp32_CmdAtIsOk()){
195         Esp32_CmdAtRst();
196     }
197 }
198
199 //Met des zéros dans la structures contenant les adresses IPs
200 void Esp32_ClearAdressIPs(e_MODE_WIFI ipsToClear){
201     int i;
202
203     //Clear le buffer qui contiens les ips
204     for(i = 0; i < SIZE_MAX_CHAR_OF_IP_ADRESS; i++){
205         if((ipsToClear == e_STATION_MODE) || (ipsToClear == e_SOFTAP_STATION_MODE)){
206             adressIP.stationIpAddress[i] = 0;
207         }
208         if((ipsToClear == e_SOFTAP_MODE) || (ipsToClear == e_SOFTAP_STATION_MODE)){
209             adressIP.apIpAddress[i] = 0;
210         }
211     }
212
213     //Clear le nombre de char par ips
214     if((ipsToClear == e_STATION_MODE) || (ipsToClear == e_SOFTAP_STATION_MODE)){

```



```

215         adressIP.sizeOfCharStationIpAddress = 0;
216     }
217     if((ipsToClear == e_SOFTAP_MODE) || (ipsToClear == e_SOFTAP_STATION_MODE)){
218         adressIP.sizeOfCharApIpAddress = 0;
219     }
220 }
221
222 //Envoie et réception de commande
223 void Esp32_SendGetMessageCmd(int8_t* cmd, int8_t* buffRx, int8_t sizeCmd, uint16_t
wait_ms){
224     //Envoie de la commande
225     SendMessageCmd(cmd, sizeCmd);
226
227     //Attente pour s'arrurer de la réception du message complet
228     Esp32_WaitForResponse(wait_ms);
229
230     //réception message
231     GetMessageCmd(buffRx, SIZE_BUFFER);
232 }
233
234 //Controle si un OK est recu qui valide l'execution de la commande
235 bool Esp32_ControlResponseOkError(int8_t *buffRx){
236     bool check;
237
238     //cmd ok ?
239     if(strstr((char *) (buffRx), (char *) (&ResponseOK[0])) != NULL){
240         check = true;
241     }
242     else{
243         check = false;
244     }
245
246     //renvoie si la commande est ok ou non
247     return(check);
248 }
249
250 //Controler que les commande AT sont actives
251 bool Esp32_CmdAtIsOk(void){
252     bool cmdOK;
253
254     //Envoie commande AT + reception message + taille de la commande
255     //+ temps d'attente entre émission et réception de 10ms
256     Esp32_SendGetMessageCmd(&cmdAT[0], &bufferMsgReceiveESP32[0], SIZE_OF_CMD_AT, 10);
257
258     //cmd ok ?
259     cmdOK = Esp32_ControlResponseOkError(&bufferMsgReceiveESP32[0]);
260
261     //effacer le buffer après traitement
262     Esp32_ClearBufferReceive();
263
264     return (cmdOK);
265 }
266
267
268 //Reste l'esp32 si les commande ne sont pas actives
269 void Esp32_CmdAtRst(void){
270     //Envoie de la commande AT pour vérifier si les cmd sont actives
271     SendMessageCmd(&cmdAtRST[0], SIZE_OF_CMD_AT_RST);
272
273     //Attente de 1s pour garantir un bon reset
274     Esp32_WaitForResponse(1000);
275 }
276
277
278 //Défini le mode Wifi
279 // 1 - Client
280 // 2 - Serveur
281 // 3 - Client & Serveur
282 bool Esp32_SetWifiMode (e_MODE_WIFI mode){
283     bool cmdOK;
284     int8_t newSizeOfCmd = 0;
285
286     //Addition du parametre

```

```

287     Esp32_AddParameterToAtCmd(&cmdAtCWMODE[0], SIZE_OF_CMD_AT_CWMODE, (int8_t)(mode));
288
289     //Nouvelle taille de commande avec parametre
290     newSizeOfCmd = (SIZE_OF_CMD_AT_CWMODE + 1);
291
292     //Envoie commande AT + reception message + taille de la commande
293     //+ temps d'attente entre émission et réception de 10ms
294     Esp32_SendGetMessageCmd(&cmdAtCWMODE[0], &bufferMsgReceiveESP32[0], newSizeOfCmd,
295                             10);
296
297     //cmd ok ?
298     cmdOK = Esp32_ControlResponseOkError(&bufferMsgReceiveESP32[0]);
299
300     //effacer le buffer après traitement
301     Esp32_ClearBufferReceive();
302
303     return (cmdOK);
304 }
305
306 //Configuration du softAP
307 // 1 - ssid (login)
308 // 2 - pwd (password)
309 // 3 - channel id (4 id disponible : 0,1,2,3)
310 // 4 - combien de device peuvent se connecter
311 // !!PAS UTILISE MAIS PEUT ETRE UTILISE POUR D'AUTRE UTILISATION!!
312 bool Esp32_ConfigurationSoftAP(void) {
313     bool cmdOK;
314     int8_t newSizeOfCmd = 0;
315
316     //Additionner le parametre ssid
317     Esp32_AddParameterStringToAtCmd(&cmdAtCWSAP[0], SIZE_OF_CMD_AT_CWSAP, &ssidEsp32[0],
318                                     SIZE_OF_SSID, newSizeOfCmd);
319     newSizeOfCmd = SIZE_OF_SSID;
320     //Additionner le parametre password
321     Esp32_AddParameterStringToAtCmd(&cmdAtCWSAP[0], SIZE_OF_CMD_AT_CWSAP, &pwdEsp32[0],
322                                     SIZE_OF_PWD, newSizeOfCmd);
323     newSizeOfCmd += SIZE_OF_PWD;
324     //Additionner le parametre du id du chenal
325     Esp32_AddParameterStringToAtCmd(&cmdAtCWSAP[0], SIZE_OF_CMD_AT_CWSAP, &
326                                     idChannelEsp32[0], SIZE_OF_ID_CHANNEL, newSizeOfCmd);
327     newSizeOfCmd += SIZE_OF_ID_CHANNEL;
328     //Additionner combien station peuvent se connecter à l'AP esp32
329     Esp32_AddParameterStringToAtCmd(&cmdAtCWSAP[0], SIZE_OF_CMD_AT_CWSAP, &
330                                     numberOfConnectedMaxEsp32[0], SIZE_OF_MAX_NB_CONNECTED_AP,
331                                     newSizeOfCmd);
332     newSizeOfCmd += SIZE_OF_MAX_NB_CONNECTED_AP;
333
334     //Envoie commande AT + reception message + taille de la commande
335     //+ temps d'attente entre émission et réception de 50ms
336     Esp32_SendGetMessageCmd(&cmdAtCWSAP[0], &bufferMsgReceiveESP32[0], newSizeOfCmd,
337                             50);
338
339     //cmd ok ?
340     cmdOK = Esp32_ControlResponseOkError(&bufferMsgReceiveESP32[0]);
341
342     //effacer le buffer après traitement
343     Esp32_ClearBufferReceive();
344
345     return (cmdOK);
346 }
347
348 //Active/Désactive le mode WPS (Connexion auto sans login et mot de passe)
349 bool Esp32_ModeWPS(e_MODE_WPS onOff) {
350     bool cmdOK;
351     int8_t newSizeOfCmd = 0;
352
353     //Addition du parametre
354     Esp32_AddParameterToAtCmd(&cmdAtWPS[0], SIZE_OF_CMD_AT_WPS, (int8_t)(onOff));
355
356     //Nouvelle taille de commande avec parametre
357     newSizeOfCmd = (SIZE_OF_CMD_AT_WPS + 1);
358
359     //Envoie commande AT + reception message + taille de la commande
360     //+ temps d'attente entre émission et réception de 2s

```

```

354     Esp32_SendGetMessageCmd(&cmdAtWPS[0], &bufferMsgReceiveESP32[0], newSizeOfCmd,
355                             2000);
356
357     //cmd ok ?
358     cmdOK = Esp32_ControlResponseOkError(&bufferMsgReceiveESP32[0]);
359
360     //effacer le buffer après traitement
361     Esp32_ClearBufferReceive();
362
363     return (cmdOK);
364 }
365
366 //Vérifie si l'esp32 (station) est connecté à un AP
367 //Commande renvoie SSID, ...
368 bool Esp32_Connected(void){
369     bool cmdOK;
370     int i;
371     int8_t sizeofParameter = 0;
372
373     //Envoie commande AT + reception message + taille de la commande
374     //+ temps d'attente entre émission et réception de 1s
375     Esp32_SendGetMessageCmd(&cmdAtCWJAP[0], &bufferMsgReceiveESP32[0],
376                             SIZE_OF_CMD_AT_CWMJAP, 1000);
377
378     //Vérifier si des informations sont présentes (SSID, macAdres, ..)
379     for(i = 0; i < SIZE_BUFFER; i++){
380         if (bufferMsgReceiveESP32[i] == 0){
381             break;
382         }
383         sizeofParameter ++;
384     }
385
386     //cmd ok ? si ok et si taille des datas > 20 car reponse :
387     AT+CWMJAP!\r\n+CWMJAP:"SSID"
388     //Jusqu'au : = 20 caractère
389     if((Esp32_ControlResponseOkError(&bufferMsgReceiveESP32[0]) == true) && (
390         sizeofParameter > 20)){
391         cmdOK = true;
392     }
393     else{
394         cmdOK = false;
395     }
396
397     //effacer le buffer après traitement
398     Esp32_ClearBufferReceive();
399
400     return (cmdOK);
401 }
402
403 //Pour la lecture des adresses ip
404 // STAIP = adresse ip de la station
405 // APIP = adresse ip du soft ap
406 bool Esp32_CmdCIFSR(void){
407     bool cmdOK = false;
408     bool stationIP, softApIP;
409     char* pIp;
410     int8_t c;
411     int i;
412
413     //Envoie commande AT + reception message + taille de la commande
414     //+ temps d'attente entre émission et réception de 10ms
415     Esp32_SendGetMessageCmd(&cmdAtCIFSR[0], &bufferMsgReceiveESP32[0],
416                             SIZE_OF_CMD_AT_CIFSR, 10);
417
418     //Adresse IP station ?
419     if(strstr((char *)(&bufferMsgReceiveESP32[0]), "STAIP") != NULL){
420         //Clear l'IP de la station (Si jamais déjà une)
421         Esp32_ClearAdressIPs(e_STATION_MODE);
422         //Pointer à l'adresse du début de l'IP
423         pIp = strstr((char *)(&bufferMsgReceiveESP32[0]), "STAIP");
424         //Pas besoin de passer après la , car la taille du /0 de STAIP est
425         //comprise
426         pIp += (sizeof("STAIP") + 1); //Pour éviter l'ouverture de ""

```

```

421 //Mise de l'IP dans le buffer d'IP
422 for(i = 0 ; i < SIZE_MAX_CHAR_OF_IP_ADRESS; i++){
423     c = (int8_t)(*pIp);
424     if (c == (int8_t)('')){
425         break;
426     }
427     else{
428         if(c == '0'){
429             stationIP = false;
430         }
431         else{
432
433             stationIP = true;
434         }
435         adressIP.stationIpAddress[i] = c;
436         pIp ++;
437         adressIP.sizeOfCharStationIpAddress ++;
438     }
439 }
440 }
441
442 //Adresse IP AP ?
443 if(strstr((char *)(&bufferMsgReceiveESP32[0]), "APIP") != NULL){
444     //Clear l'IP de la AP (Si jamais déjà une)
445     Esp32_ClearAdressIPs(e_SOFTAP_MODE);
446     //Pointer à l'adresse du début de l'IP
447     pIp = strstr((char *)(&bufferMsgReceiveESP32[0]), "APIP");
448     //Pas besoin de passer après la , car la taille du /0 de STAIP est
449     //comprise
450     pIp += (sizeof("APIP")+ 1); //Pour éviter l'ouverture de ""
451     //Mise de l'IP dans le buffer d'IP
452     for(i = 0 ; i < SIZE_MAX_CHAR_OF_IP_ADRESS; i++){
453         c = (int8_t)(*pIp);
454         if (c == (int8_t)('')){
455             break;
456         }
457         else{
458             if(c == '0'){
459                 softApIP = false;
460             }
461             else{
462
463                 softApIP = true;
464             }
465             adressIP.apIpAddress[i] = c;
466             pIp ++;
467             adressIP.sizeOfCharApIpAddress ++;
468         }
469     }
470 }
471
472 if(softApIP & stationIP){
473     cmdOK = true;
474 }
475 //effacer le buffer après traitement
476 Esp32_ClearBufferReceive();
477
478 return (cmdOK);
479 }
480
481 //configuration du sntp
482 // 1 - activation du sntp
483 // 2 - timeZone
484 // 3,4,5 - serveur sntp
485 bool Esp32_ConfigSNTP(void){
486     bool cmdOK;
487     int8_t sizeofNewCommand = 0;
488     //_____CONFIGURATION DU SNTP_____
489     //Addition du parametre =1 (enable SNTP)
490     Esp32_AddParameterToAtCmd(&cmdAtCIPSNTPCFG[0], SIZE_OF_CMD_AT_CIPSNTPCFG, 1);
491     //---Addition d'autres parametres
492     //Timezone 2

```

```

493 Esp32_AddParameterStringToAtCmd(&cmdAtCIPSNTPCFG[0], SIZE_OF_CMD_AT_CIPSNTPCFG, &
param0Sntp[0], SIZE_OF_0_PARAM_Sntp, 2);
494 sizeofNewCommand = SIZE_OF_0_PARAM_Sntp + 2;
495 //Serveur sntp 1
496 Esp32_AddParameterStringToAtCmd(&cmdAtCIPSNTPCFG[0], SIZE_OF_CMD_AT_CIPSNTPCFG, &
param1Sntp[0], SIZE_OF_1_PARAM_Sntp, sizeofNewCommand);
497 sizeofNewCommand += SIZE_OF_1_PARAM_Sntp;
498 //Serveur sntp 2
499 Esp32_AddParameterStringToAtCmd(&cmdAtCIPSNTPCFG[0], SIZE_OF_CMD_AT_CIPSNTPCFG, &
param2Sntp[0], SIZE_OF_2_PARAM_Sntp, sizeofNewCommand);
500 sizeofNewCommand += SIZE_OF_2_PARAM_Sntp;
501 //Serveur sntp3
502 Esp32_AddParameterStringToAtCmd(&cmdAtCIPSNTPCFG[0], SIZE_OF_CMD_AT_CIPSNTPCFG, &
param3Sntp[0], SIZE_OF_3_PARAM_Sntp, sizeofNewCommand);
503 sizeofNewCommand += SIZE_OF_3_PARAM_Sntp;
504
505 //set le size of new command
506 sizeofNewCommand += SIZE_OF_CMD_AT_CIPSNTPCFG;
507
508 //Envoie commande AT + reception message + taille de la commande
509 //+ temps d'attente entre émission et réception de 500ms
510 Esp32_SendGetMessageCmd(&cmdAtCIPSNTPCFG[0], &bufferMsgReceiveESP32[0],
sizeofNewCommand, 500);
511
512 //commande ok ?
513 cmdOK = Esp32_ControlResponseOkError(&bufferMsgReceiveESP32[0]);
514
515 //effacer le buffer après traitement
516 Esp32_ClearBufferReceive();
517
518 return (cmdOK);
519 }
520
521 //Réceptionner le temps via sntp
522 bool Esp32_GetTimeSNTP(s_Time* timeToSet){
523     bool cmdOK;
524
525
526
527 //config ok ?
528 if(Esp32_ConfigSNTP() == true){
529     //_____RECUP DE L'HEURE_____
530
531     //Envoie commande AT + reception message + taille de la commande
532     //+ temps d'attente entre émission et réception de 50ms
533     // Ajout de la boucle do while pour attendre la mise à jour de l'heure et la
    date
534     // La mise à jour prendre quelques secondes
535     // Par défaut la date et l'heure est :
536     // 1 janvier 1970 2:00:00
537     do
538     {
539         Esp32_SendGetMessageCmd(&cmdAtCIPSNTPTIME[0], &bufferMsgReceiveESP32[0],
SIZE_OF_CMD_AT_CIPSNTPTIME, 50);
540     }while((bufferMsgReceiveESP32[50] == '1') && (bufferMsgReceiveESP32[51] == '9'
) && (bufferMsgReceiveESP32[52] == '7') && (bufferMsgReceiveESP32[53] == '0'
));
541
542
543 //cmd ok ?
544 if(Esp32_ControlResponseOkError(&bufferMsgReceiveESP32[0]) == true){
545     //Traitement des valeurs recues
546     Esp32_TreatmentDataSTNP(timeToSet);
547     cmdOK = true;
548 }
549 else{
550     cmdOK = false;
551 }
552 }
553 else{
554     //config non ok
555     cmdOK = false;
556 }

```

```

557
558 //effacer le buffer après traitement
559 Esp32_ClearBufferReceive();
560
561 return (cmdOK);
562 }
563
564 //Activer connexion unique ou connexion multiple
565 bool Esp32_SetDisableMultipleConnections(e_MULTIPLE_SINGLE_CONNECTION connection){
566     bool cmdOK;
567     int8_t newSizeOfCmd = 0;
568
569     //Addition du parametre
570     Esp32_AddParameterToAtCmd(&cmdAtCIPMUX[0], SIZE_OF_CMD_AT_CIPMUX, (int8_t)(
        connection));
571
572     //nouvelle taille de la commande avec le parametre
573     newSizeOfCmd = (SIZE_OF_CMD_AT_CIPMUX + 1);
574
575     //Envoie commande AT + reception message + taille de la commande
576     //+ temps d'attente entre émission et réception de 10ms
577     Esp32_SendGetMessageCmd(&cmdAtCIPMUX[0], &bufferMsgReceiveESP32[0], newSizeOfCmd,
        10);
578
579     //commande ok ?
580     cmdOK = Esp32_ControlResponseOkError(&bufferMsgReceiveESP32[0]);
581
582     //effacer le buffer après traitement
583     Esp32_ClearBufferReceive();
584
585     return(cmdOK);
586 }
587
588 //création du serveur tcp (sur la station esp32)
589 // 1 - activer plusieurs connexion
590 // 2 - création du serveur
591 bool Esp32_SetTCPServer(uint8_t SetReset){
592     bool cmdOK = false;
593     int8_t newSizeOfCmd = 0;
594
595     //Activation d'un connexion simple
596     Esp32_SetDisableMultipleConnections(e_MULTIPLE_CONNECTION);
597
598     //Addition du parametre =1 (enable server tcp)
599     //                               =0 (disable server tcp)
600     Esp32_AddParameterToAtCmd(&cmdAtCIPSERVER[0], SIZE_OF_CMD_AT_CIPSERVER, SetReset);
601
602     //Addition du parametre du port
603     Esp32_AddParameterStringToAtCmd(&cmdAtCIPSERVER[0], SIZE_OF_CMD_AT_CIPSERVER, &
        portServerTCP[0], SIZE_OF_PORT_SERVER_TCP, 2);
604
605     //nouvelle taille de la commande avec les parametre
606     newSizeOfCmd = (SIZE_OF_CMD_AT_CIPSERVER + SIZE_OF_PORT_SERVER_TCP + 2);
607
608     //Envoie commande AT + reception message + taille de la commande
609     //+ temps d'attente entre émission et réception de 1s
610     Esp32_SendGetMessageCmd(&cmdAtCIPSERVER[0], &bufferMsgReceiveESP32[0],
        newSizeOfCmd, 1000);
611
612     //commande ok ?
613     cmdOK = Esp32_ControlResponseOkError(&bufferMsgReceiveESP32[0]);
614
615     //effacer le buffer après traitement
616     Esp32_ClearBufferReceive();
617
618     return(cmdOK);
619 }
620
621 //Configurer le mode du socket
622 //Définir le mode du socket tcpip
623 //Passif - Conserve les données du socket dans un tampon interne
624 //En attente d'une demande de l'hôte (PIC32)

```

```

625 //Actif (Par défaut) - Envoie le socket directement à l'hôte
626 bool Esp32_SetSocketMode(e_SOCKET_MODE mode){
627     bool cmdOK;
628     int8_t newSizeOfCmd = 0;
629
630     //Addition du parametre
631     Esp32_AddParameterToAtCmd(&cmdAtCIPRECVMODE[0], SIZE_OF_CMD_AT_CIPRECVMODE, (
        int8_t)(mode));
632
633     //nouvelle taille de la commande avec les parametre
634     newSizeOfCmd = (SIZE_OF_CMD_AT_CIPRECVMODE + 1);
635
636     //Envoie commande AT + reception message + taille de la commande
637     //+ temps d'attente entre émission et réception de 50ms
638     Esp32_SendGetMessageCmd(&cmdAtCIPRECVMODE[0], &bufferMsgReceiveESP32[0],
        newSizeOfCmd, 200);
639
640     //cmd ok ?
641     cmdOK = Esp32_ControlResponseOkError(&bufferMsgReceiveESP32[0]);
642
643     //effacer le buffer après traitement
644     Esp32_ClearBufferReceive();
645
646     return (cmdOK);
647 }
648
649 //Obtenir et retourner le nombre de bytes disponible dans le socket
650 uint32_t Esp32_GetSizeOfDataSocket(void){
651     uint32_t sizeOfData;
652     uint8_t i;
653     uint8_t j = 0;
654     char sizeInStr[SIZE_OF_DIGIT_UINT32];
655
656     //Envoie commande AT + reception message + taille de la commande
657     //+ temps d'attente entre émission et réception de 10ms
658     Esp32_SendGetMessageCmd(&cmdAtCIPRECVLEN[0], &bufferMsgReceiveESP32[0],
        SIZE_OF_CMD_AT_CIPRECVLEN, 10);
659
660     //cmd ok ?
661     if(Esp32_ControlResponseOkError(&bufferMsgReceiveESP32[0]) == true){
662         //Récup de la taille des octets disponible
663         for(i = 0; i < SIZE_BUFFER; i++){
664             if ((bufferMsgReceiveESP32[i] > ZERO_ASCII) && (bufferMsgReceiveESP32[i] <
                NINE_ASCII)){
665                 sizeInStr[j] = (char)(bufferMsgReceiveESP32[i]);
666                 j ++;
667             }
668             else if(j > 0){
669                 break;
670             }
671         }
672         sizeOfData = atol(&sizeInStr[0]);
673     }
674     else{
675         sizeOfData = 0;
676     }
677
678     //effacer le buffer après traitement
679     Esp32_ClearBufferReceive();
680
681     return (sizeOfData);
682 }
683
684 //Obtenir et transferer les bytes disponible dans le socket
685 bool Esp32_GetDataFromSocket(int8_t* buff, uint32_t nbByteToRead){
686     bool cmdOk;
687     uint8_t i;
688     uint8_t j = 0;
689     int8_t dataInStr[SIZE_OF_MSG_TCPIP];
690     int8_t nbByteToRead_str[SIZE_OF_MSG_TCPIP_STR] = {0,0,0};
691     int8_t newSizeOfCmd = 0;
692
693     //Addition du parametre

```

```

694 // 0 = Link ID où sont les informations
695 Esp32_AddParameterToAtCmd(&cmdAtCIPRECVDATA[0], SIZE_OF_CMD_AT_CIPRECVDATA, 0);
696
697 //convertir le nombre de datas à lire en str pour ajouter à la commande at
698 nbByteToRead_str[0] = (int8_t)((nbByteToRead/10) + ZERO_ASCII);
699 nbByteToRead_str[1] = (int8_t)((nbByteToRead - ((nbByteToRead/10)*10)) +
700 ZERO_ASCII);
701 nbByteToRead_str[2] = '!';
702
703 //---Addition d'autres parametres
704 //taille du message à recevoir
705 Esp32_AddParameterStringToAtCmd(&cmdAtCIPRECVDATA[0], SIZE_OF_CMD_AT_CIPRECVDATA,
706 &nbByteToRead_str[0], SIZE_OF_MSG_TCPIP_STR, 2);
707
708 //Nouvelle taille de la commande avec les parametres
709 newSizeOfCmd = (SIZE_OF_CMD_AT_CIPRECVDATA + SIZE_OF_MSG_TCPIP_STR + 2);
710
711 //Envoie commande AT + reception message + taille de la commande
712 //+ temps d'attente entre émission et réception de 10ms
713 Esp32_SendGetMessageCmd(&cmdAtCIPRECVDATA[0], &bufferMsgReceiveESP32[0],
714 newSizeOfCmd, 10);
715
716 //cmd ok ?
717 if(Esp32_ControlResponseOkError(&bufferMsgReceiveESP32[0]) == true){
718 //Récup de la taille des octets disponible
719 for(i = 0; i < SIZE_BUFFER; i++){
720 if ((bufferMsgReceiveESP32[i] == START_MSG_TCPIP) || (dataInStr[0] ==
721 START_MSG_TCPIP)){
722 if((j > 0) && (bufferMsgReceiveESP32[i] == STOP_MSG_TCPIP)){
723 dataInStr[j] = bufferMsgReceiveESP32[i];
724 j ++;
725 if(j >= SIZE_OF_MSG_TCPIP){
726 cmdOk = true;
727 }
728 else{
729 cmdOk = false;
730 }
731 break;
732 }
733 else{
734 dataInStr[j] = bufferMsgReceiveESP32[i];
735 j ++;
736 }
737 }
738 cmdOk = false;
739 }
740
741 for(i = 0; i < j; i++){
742 *buff = dataInStr[i];
743 buff ++;
744 }
745 }
746 else{
747 cmdOk = false;
748 }
749
750 //effacer le buffer après traitement
751 Esp32_ClearBufferReceive();
752
753 return (cmdOk);
754 }
755
756 //Régler le timeout du serveur tcp (combien de temps avant déconnexion auto)
757 bool Esp32_SetTimeoutServerTCP(uint16_t timeOut_s){
758 bool cmdOK;
759 int8_t newSizeOfCmd = 0;
760
761 // + 1 Pour y mettre le '!'
762 int8_t paramStr[SIZE_MAX_DIGIT_TIMEOUT + 1];
763
764 //Convertir le parametre en chaine de caractère
765 //sprintf((char*)&paramStr[0]), (char*)("%s"), timeOut_s);
766 itoa((char*)&paramStr[0]), (int)(timeOut_s), 10);

```



```

763     strcat((char*)(&paramStr[0]), (char*)("!!"));
764
765     //---Addition d'autres parametres
766     //valeur du nouveau timeout
767     Esp32_AddParameterStringToAtCmd(&cmdAtCIPSTO[0], SIZE_OF_CMD_AT_CIPSTO, &paramStr[
0], strlen((char*)(&paramStr[0])), 0);
768
769     //Nouvelle taille de la commande avec parametres
770     newSizeOfCmd = (SIZE_OF_CMD_AT_CIPSTO + strlen((char*)(&paramStr[0])));
771
772     //Envoie commande AT + reception message + taille de la commande
773     //+ temps d'attente entre émission et réception de 200ms
774     Esp32_SendGetMessageCmd(&cmdAtCIPSTO[0], &bufferMsgReceiveESP32[0], newSizeOfCmd,
200);
775
776     //cmd ok ?
777     cmdOK = Esp32_ControlResponseOkError(&bufferMsgReceiveESP32[0]);
778
779     //effacer le buffer après traitement
780     Esp32_ClearBufferReceive();
781
782     return (cmdOK);
783 }
784
785
786
787 //_____FONCTION UTILITAIRES_____
788 //transfere la date le mois le jour l'heure les minutes les secondes
789 //Recu via stnp (datas disponible dans le buffer receive)
790 void Esp32_TreatmentDataSTNP (s_Time* dataToPut){
791     int8_t offsetTime;
792
793     offsetTime = Esp32_ConvDayMonthStnpToRtc(&dataToPut->day, &dataToPut->month);
794     Esp32_ConvDateHourMinSecStnpToRtc(offsetTime, &dataToPut->date, &dataToPut->hour,
&dataToPut->min, &dataToPut->sec, &dataToPut->year);
795 }
796
797 //Transfer la date l'heure les minutes les secondes récup sur le buffer receive
798 //offset = case du buffer receive qui pointe sur le début de la date
799 void Esp32_ConvDateHourMinSecStnpToRtc(int8_t offset, uint8_t *date, uint8_t *hour,
uint8_t *min, uint8_t *sec, uint8_t *year){
800     int i;
801     int8_t strTime[2];
802
803     //récupération de la date l'heure minute seconde
804     for(i = 0; i < 5; i++){
805         strTime[0] = bufferMsgReceiveESP32[offset];
806         offset++;
807         strTime[1] = bufferMsgReceiveESP32[offset];
808         switch (i){
809             case 0:{
810                 *date = (uint8_t) (atoi((char*)(&strTime[0])));
811                 break;
812             }
813             case 1:{
814                 *hour = (uint8_t) (atoi((char*)(&strTime[0])));
815                 break;
816             }
817             case 2:{
818                 *min = (uint8_t) (atoi((char*)(&strTime[0])));
819                 break;
820             }
821             case 3:{
822                 *sec = (uint8_t) (atoi((char*)(&strTime[0])));
823                 break;
824             }
825             // ajout pour l'année
826             case 4:
827             {
828                 *year = (uint8_t) (atoi((char*)(&strTime[0])));
829                 break;
830             }
831         }

```

```

832     if(i == 3)
833     {
834         //déplacer de l'unité et du .
835         offset += 2;
836     }
837     //déplacer de l'unité et du .
838     offset += 2;
839 }
840 }
841
842 //Transfer le jour le mois récup sur le buffer receive
843 //retour = case du buffer receive qui pointe sur le début de la date
844 int8_t Esp32_ConvDayMonthStnpToRtc(uint8_t *day, uint8_t *month){
845     int i;
846     int8_t adressAfterMonth;
847
848     //___Traitement du jour
849     for(i = 0; i < 50; i++){
850         if(bufferMsgReceiveESP32[i] == ':'){
851             i++;
852             switch(bufferMsgReceiveESP32[i]){
853                 case 'M':{
854                     *day = 1;
855                     break;
856                 }
857                 case 'T':{
858                     if(bufferMsgReceiveESP32[i+1] == 'u'){
859                         *day = 2;
860                     }
861                     else{
862                         *day = 4;
863                     }
864                     break;
865                 }
866                 case 'W':{
867                     *day = 3;
868                     break;
869                 }
870                 case 'F':{
871                     *day = 5;
872                     break;
873                 }
874                 case 'S':{
875                     if(bufferMsgReceiveESP32[i+1] == 'u'){
876                         *day = 7;
877                     }
878                     else{
879                         *day = 6;
880                     }
881                     break;
882                 }
883             }
884             adressAfterMonth = i;
885             break;
886         }
887     }
888 }
889
890 //___Traitement des mois
891 //Déplacement avant le mois
892 //--> "Mon "Dec.. (4)
893 adressAfterMonth += 4;
894 switch(bufferMsgReceiveESP32[adressAfterMonth]){
895     case 'M':{
896         if (bufferMsgReceiveESP32[adressAfterMonth + 2] == 'r'){
897             *month = 3;
898         }
899         else{
900             *month = 5;
901         }
902         break;
903     }
904     case 'J':{

```

```

905         if (bufferMsgReceiveESP32[adressAfterMonth + 2] == 'a'){
906             *month = 1;
907         }
908         else if (bufferMsgReceiveESP32[adressAfterMonth + 2] == 'n'){
909             *month = 6;
910         }
911         else{
912             *month = 7;
913         }
914         break;
915     }
916     case 'A':{
917         if (bufferMsgReceiveESP32[adressAfterMonth + 2] == 'g'){
918             *month = 8;
919         }
920         else{
921             *month = 4;
922         }
923         break;
924     }
925     case 'O':{
926         *month = 10;
927         break;
928     }
929     case 'N':{
930         *month = 11;
931         break;
932     }
933     case 'D':{
934         *month = 12;
935         break;
936     }
937     case 'S':{
938         *month = 9;
939         break;
940     }
941 }
942
943 //Passage au début de l'heure
944 //--> "Dec 1".. (4)
945 adressAfterMonth += 4;
946
947 //Retour de la position du début de l'heure
948 return (adressAfterMonth);
949 }
950
951 //Pour définir le temps d'attente entre l'émission d'une commande
952 //et un réception de la réponse
953 void Esp32_WaitForResponse (uint16_t timeToWait_ms){
954     //Fixer le temps d'attente
955     consigneToWaitForResponseESP32 = timeToWait_ms;
956
957     //Debut de l'attente de reponse de l'esp32
958     waitForResponseESP32 = true;
959
960     //Rester ici temps que le temps c'est pas écouler
961     while(waitForResponseESP32){
962     }
963 }
964
965 //__FONCTIONS POUR LE TRAITEMENT DE L'ATTENTE
966 //LIER A UN TIMER AVEC INTERRUPTION 1ms
967 bool Esp32_GetWaitForResponse(void){
968     return (waitForResponseESP32);
969 }
970 void Esp32_ClearWaitForResponseESP32(void){
971     waitForResponseESP32 = false;
972 }
973 uint16_t Esp32_GetConsigneToWaitForResponse(void){
974     return (consigneToWaitForResponseESP32);
975 }
976
977 //Ajouter un seul parametre à une commande

```

```

978 //A ajouter dans le switch si le parametre est différent de : 0,1,2,3,8
979 void Esp32_AddParameterToAtCmd(int8_t *buff, int8_t sizeofCmd, int8_t paramToAdd){
980     int8_t charToAdd;
981     int8_t i;
982
983     //Conversion du parametre en caractère
984     switch (paramToAdd){
985         case 0 :{
986             charToAdd = '0';
987             break;
988         }
989         case 1 :{
990             charToAdd = '1';
991             break;
992         }
993         case 2 :{
994             charToAdd = '2';
995             break;
996         }
997         case 3 :{
998             charToAdd = '3';
999             break;
1000         }
1001         case 8 :{
1002             charToAdd = '8';
1003         }
1004     }
1005
1006     //Placement du caractère après le = et reordonner la commande
1007     for(i=0; i < sizeofCmd; i++){
1008         //Traitement de la case de la commande
1009         if (*buff == '='){
1010             //Addition du parametre en prochain char
1011             buff++;
1012             *buff = charToAdd;
1013
1014             //Mettre le CF LR
1015             buff++;
1016             *buff = 0x0D;
1017             buff++;
1018             *buff = 0x0A;
1019
1020             //sortir de la boucle
1021             break;
1022         }
1023         //prochaine case de la commande
1024         buff++;
1025     }
1026 }
1027
1028 //Addition de plusieurs parametre à une commande (ex: parametre = string)
1029 void Esp32_AddParameterStringToAtCmd(int8_t *buff, int8_t sizeofCmd, int8_t *
paramToAdd, int8_t sizeofParam, int8_t offsetOfCmd){
1030     int8_t i, j;
1031
1032     //Placement du caractère après le = et reordonner la commande
1033     for(i=0; i < sizeofCmd; i++){
1034         //Traitement de la case de la commande
1035         if (*buff == '='){
1036             //Si un parametre est deja la alors aller jusqu'à la fin et mettre une ,
1037             if (offsetOfCmd != 0){
1038                 buff += offsetOfCmd;
1039                 *buff = ',';
1040             }
1041             //Passer au caractère suivante soit apres = soit apres ,
1042             buff++;
1043             //Mettre le parametre
1044             for(j = 0; j < sizeofParam; j++){
1045                 if(*paramToAdd == '!'){
1046                     break;
1047                 }
1048                 else{
1049                     *buff = *paramToAdd;

```

```

1050             buff ++;
1051             paramToAdd ++;
1052         }
1053     }
1054     *buff = 0x0D;
1055     buff++;
1056     *buff = 0x0A;
1057     //sortir de la boucle
1058     break;
1059 }
1060 //prochaine case de la commande
1061 buff++;
1062 }
1063 }
1064
1065 //Effacer le buffer receive
1066 void Esp32_ClearBufferReceive(void){
1067     int i;
1068
1069     //effacement de toutes les cases
1070     for (i = 0; i < SIZE_BUFFER; i++){
1071         bufferMsgReceiveESP32[i] = 0;
1072     }
1073 }
1074
1075 //Obtenir et transferer les adresses IP
1076 void Esp32_GetIpsAdress (s_ESP32_IPs *buff){
1077     int i;
1078
1079     //Commande CWJAP
1080     while (!Esp32_CmdCIFSR()){
1081
1082     }
1083
1084     //Transfer du nombre de char que contient l'adresse ip
1085     buff->sizeofCharStationIpAddress = adressIP.sizeOfCharStationIpAddress;
1086     buff->sizeofCharApIpAddress = adressIP.sizeOfCharApIpAddress;
1087
1088     //Transfer de l'IP station
1089     for(i = 0; i < adressIP.sizeOfCharStationIpAddress; i++){
1090         buff->stationIpAddress[i] = adressIP.stationIpAddress[i];
1091     }
1092
1093     //Transfer de l'IP soft AP
1094     for(i = 0; i < adressIP.sizeOfCharApIpAddress; i++){
1095         buff->apIpAddress[i] = adressIP.apIpAddress[i];
1096     }
1097 }
1098 }
1099
1100
1101 bool Esp32_GetState(uint8_t *State)
1102 {
1103     bool cmdOK;
1104     int i;
1105     uint8_t sizeofParameter = 0;
1106     char *PtVal;
1107
1108     //Envoie commande AT + reception message + taille de la commande
1109     //+ temps d'attente entre émission et réception de ls
1110     Esp32_SendGetMessageCmd(&cmdAtCIPSTATUS[0], &bufferMsgReceiveESP32[0],
1111     SIZE_OF_CMD_AT_CIPSTATUS, 1000);
1112
1113     //Vérifier si des informations sont présentes (SSID, macAdres, ..)
1114     for(i = 0; i < SIZE_BUFFER; i++){
1115         if (bufferMsgReceiveESP32[i] == 0){
1116             break;
1117         }
1118         sizeofParameter ++;
1119     }
1120
1121     //cmd ok ? si ok et si taille des datas > 20 car reponse :
1122     AT+CWJAP!\r\n+CWJAP:"SSID"

```

```
1121 //Jusqu'au := 20 caractère
1122 if((Esp32_ControlResponseOkError(&bufferMsgReceiveESP32[0]) == true) && (
1123     sizeofParameter > 20)){
1124     cmdOK = true;
1125     PtVal = strstr((char*)bufferMsgReceiveESP32, (char*)"STATUS:");
1126     if(PtVal != NULL)
1127     {
1128         PtVal = PtVal + 7;
1129         *State = atoi((char *)PtVal);
1130     }
1131 }
1132 else{
1133     cmdOK = false;
1134 }
1135 //effacer le buffer après traitement
1136 Esp32_ClearBufferReceive();
1137
1138 return (cmdOK);
1139 }
1140 /*****
1141 End of File
1142 */
1143
```

```

1  /*****
2  MPLAB Harmony Application Header File
3
4  Company:
5  Microchip Technology Inc.
6
7  File Name:
8  gestEsp32.h
9  *****/
10
11  //-----|
12  //Réaliser par : Rodrigo Lopes dos Santos |
13  //-----|
14
15
16
17  #ifndef _GEST_ESP_32_H
18  #define _GEST_ESP_32_H
19
20
21
22  // ****
23  // ****
24  // Section: Included Files
25  // ****
26  // ****
27
28  #include <stdint.h>
29  #include <stdbool.h>
30  #include <stddef.h>
31  #include <stdlib.h>
32  #include <stdio.h>
33  #include "app.h"
34  #include "system_config.h"
35  #include "system_definitions.h"
36
37
38
39  // DOM-IGNORE-BEGIN
40  #ifdef __cplusplus // Provide C++ Compatibility
41
42  extern "C" {
43
44  #endif
45
46  // ****
47  // ****
48  // Section: Constantes
49  // ****
50  // ****
51  #define SIZE_BUFFER 150
52  //255.255.255 (12) + les point = 3 (15)
53  #define SIZE_MAX_CHAR_OF_IP_ADRESS 0x0F
54
55  #define SIZE_OF_SSID 0x07
56  #define SIZE_OF_PWD 0x07
57  #define SIZE_OF_ID_CHANNEL 0x02
58  #define SIZE_OF_MAX_NB_CONNECTED_AP 0x02
59
60  #define SIZE_OF_CMD_AT 0x04
61  #define SIZE_OF_CMD_AT_RST 0x08
62  #define SIZE_OF_CMD_AT_CWMODE 0x0C
63  #define SIZE_OF_CMD_AT_WPS 0x09
64  #define SIZE_OF_CMD_AT_CIFSR 0x0A
65  #define SIZE_OF_CMD_AT_CWMJAP 0x0B
66  #define SIZE_OF_CMD_AT_CWSAP 0x0B
67
68  #define SIZE_OF_CMD_AT_CIPSTATUS 0x0E
69  #define SIZE_OF_CMD_AT_CIPSERVER 0x0F
70  #define SIZE_OF_CMD_AT_CIPSTO 0x0C
71  #define SIZE_OF_CMD_AT_CIPMUX 0x0C
72
73  #define SIZE_OF_CMD_AT_CIPRECVMODE 0x11

```

```

74 #define SIZE_OF_CMD_AT_CIPRECVLEN 0x10
75 #define SIZE_OF_CMD_AT_CIPRECVDATA 0x11
76
77 #define SIZE_OF_CMD_AT_CIPSNTPTIME 0x11
78 #define SIZE_OF_CMD_AT_CIPSNTPCFG 0x10
79 #define SIZE_OF_0_PARAM_Sntp 0x02
80 #define SIZE_OF_1_PARAM_Sntp 0x10
81 #define SIZE_OF_2_PARAM_Sntp 0x12
82 #define SIZE_OF_3_PARAM_Sntp 0x12
83
84 // #TRIR_X_AA_BB_CC_DD! (20 bytes)
85 // X=numéro trame / AA=Date / BB=Mois / CC=Heure/ DD=Minutes
86 #define SIZE_OF_MSG_TCPIP 0x14
87 #define SIZE_OF_MSG_TCPIP_STR 0x03
88 #define START_MSG_TCPIP '#'
89 #define STOP_MSG_TCPIP '!'
90 #define SIZE_OF_PORT_SERVER_TCP 0x04
91
92 #define SIZE_OF_DIGIT_UINT32 0x0A
93
94 #define ZERO_ASCII '0'
95 #define NINE_ASCII '9'
96
97 #define SIZE_MAX_DIGIT_TIMEOUT 0x04
98
99
100 // *****
101 // *****
102 // Section: Type Definitions
103 // *****
104 // *****
105
106 //ENUMERATION
107 typedef enum
108 {
109     e_CMD_AT = 0,
110     e_CMD_AT_RST = 1
111
112 }e_CMD;
113
114 typedef enum
115 {
116     e_STATION_MODE = 1,
117     e_SOFTAP_MODE = 2,
118     e_SOFTAP_STATION_MODE = 3
119 }e_MODE_WIFI;
120
121 typedef enum
122 {
123     e_DISABLE_WPS = 0,
124     e_ENABLE_WPS = 1
125 }e_MODE_WPS;
126
127 typedef enum
128 {
129     e_SOCKET_MODE_ACTIF = 0,
130     e_SOCKET_MODE_PASSIF = 1
131 }e_SOCKET_MODE;
132
133 typedef enum
134 {
135     e_SINGLE_CONNECTION = 0,
136     e_MULTIPLE_CONNECTION = 1
137 }e_MULTIPLE_SINGLE_CONNECTION;
138
139
140 //STRUCTURE
141 typedef struct
142 {
143     int8_t stationIpAddress[SIZE_MAX_CHAR_OF_IP_ADRESS];
144     int8_t sizeOfCharStationIpAddress;
145     int8_t apIpAddress[SIZE_MAX_CHAR_OF_IP_ADRESS];
146     int8_t sizeOfCharApIpAddress;

```



```

147 }s_ESP32_IPs;
148
149 typedef struct
150 {
151     uint8_t sec;
152     uint8_t min;
153     uint8_t hour;
154     uint8_t day;
155     uint8_t date;
156     uint8_t month;
157     uint8_t year;
158 }s_Time;
159
160
161 //UNION
162
163
164
165 // *****
166 // *****
167 // Section: Function prototype
168 // *****
169 // *****
170 //fonction d'initialisation
171 void Esp32_Initialize(void);
172
173 //fonctions d'émission/réception uart pour commande AT
174 void Esp32_SendGetMessageCmd(int8_t* cmd, int8_t* buffRx, int8_t sizeCmd, uint16_t
wait_ms);
175 bool Esp32_ControlResponseOkError(int8_t *buffRx);
176
177 //fonctions pour commande AT pour initialisation
178 bool Esp32_CmdAtIsOk(void);
179 void Esp32_CmdAtRst(void);
180 bool Esp32_CmdCIFSR(void);
181
182 //fonctions pour la configuration wifi
183 bool Esp32_SetWifiMode (e_MODE_WIFI mode);
184 bool Esp32_ConfigurationSoftAP(void);
185 bool Esp32_ModeWPS(e_MODE_WPS onOff);
186 bool Esp32_Connected(void);
187
188 //fonctions pour la configurations serveur tcp
189 bool Esp32_SetTCPServer(uint8_t SetReset);
190 bool Esp32_SetSocketMode(e_SOCKET_MODE mode);
191 uint32_t Esp32_GetSizeOfDataSocket(void);
192 bool Esp32_GetDataFromSocket(int8_t* buff, uint32_t nbByteToRead);
193 bool Esp32_SetTimeOutServerTCP(uint16_t timeOut_s);
194 bool Esp32_SetDisableMultipleConnections(e_MULTIPLE_SINGLE_CONNECTION connection);
195
196 //fonctions pour le temps d'attente entre émission/réception commande AT
197 void Esp32_WaitForResponse (uint16_t timeToWait);
198 bool Esp32_GetWaitForResponse(void);
199 void Esp32_ClearWaitForResponseESP32(void);
200 uint16_t Esp32_GetConsigneToWaitForResponse(void);
201
202 //fonctions utilitaires pour gestion de commandes, buffer, adresses ips
203 void Esp32_AddParameterToAtCmd(int8_t *buff, int8_t sizeOfCmd, int8_t paramToAdd);
204 void Esp32_AddParameterStringToAtCmd(int8_t *buff, int8_t sizeOfCmd, int8_t *
paramToAdd, int8_t sizeOfParam, int8_t offsetOfCmd);
205 void Esp32_ClearBufferReceive(void);
206 void Esp32_ClearAdressIPs(e_MODE_WIFI ipsToClear);
207 void Esp32_GetIpsAdress (s_ESP32_IPs *buff);
208
209 //fonctions pour la configurations sntp
210 bool Esp32_ConfigSNTP(void);
211 bool Esp32_GetTimeSNTP(s_Time* timeToSet);
212 void Esp32_TreatmentDataSTNP (s_Time* dataToPut);
213 int8_t Esp32_ConvDayMonthStnpToRtc(uint8_t *day, uint8_t *month);
214 //void Esp32_ConvDateHourMinSecStnpToRtc(int8_t offset, uint8_t *date, uint8_t *hour,
uint8_t *min, uint8_t *sec);
215 void Esp32_ConvDateHourMinSecStnpToRtc(int8_t offset, uint8_t *date, uint8_t *hour,
uint8_t *min, uint8_t *sec, uint8_t *year);

```

```
216
217
218  bool Esp32_GetState(uint8_t *State);
219
220  #endif /* _GEST_ESP_32_H */
221
222  //DOM-IGNORE-BEGIN
223  #ifdef __cplusplus
224  }
225  #endif
226  //DOM-IGNORE-END
227
228  /*****
229  End of File
230  */
```

```

1  /*****
2  MPLAB Harmony Application Source File
3
4  Company:
5  Microchip Technology Inc.
6
7  File Name:
8  gestIR.c
9  *****/
10
11  //-----|
12  //Réaliser par : Rodrigo Lopes dos Santos |
13  //-----|
14
15
16
17  /* *****/
18  /* *****/
19  /* Section: Included Files */
20  /* *****/
21  /* *****/
22  #include "gestIR.h"
23  #include "peripheral/tmr/plib_tmr.h"
24  #include "driver/tmr/drv_tmr_static.h"
25  #include "Mc32NVMUtil.h"
26
27
28
29  /* *****/
30  /* *****/
31  /* Section: File Scope or Global Data */
32  /* *****/
33  /* *****/
34
35  //Pour stocker les trames apprises IR avec le temps de chaque bit (buffer)
36  uint16_t dataReceiveIR [SIZE_OF_BUFFER_IR];
37  uint16_t timeOfDataReceiveIR [SIZE_OF_BUFFER_IR];
38
39  //Sers à se déplacer sur les deux buffer
40  uint8_t idxBuffer;
41
42  //Variable qui va être utilisé par l'IC1 afin de savoir si c'est la première
43  //Interruption de l'IC1 donc un start
44  bool isTheStart;
45
46  //=====
47  // Tableau pour sauvegarder les valeurs de la première page de la flash du uC
48  // Pour la indiquer si des trammes sont enregistrées dans la flash du uC
49  // case 0 => valeur pour indiquer si des trames ont été enregistrés
50  // Reste de cases => adresse de la première valeur de chaque trame enregistré
51  uint32_t FlashStartRow[32] = {0x12345678, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF,
52                                0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF,
53                                0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF,
54                                0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF,
55                                0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF,
56                                0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF, 0xFFFFFFFF,
57                                0xFFFFFFFF, 0xFFFFFFFF};
58
59  /* *****/
60  /* *****/
61  // Section: Local Functions */
62  /* *****/
63  /* *****/
64
65  //Intialise les buffer de réception IR
66  void IR_Initialize (void){
67      IR_ClearBufferIR();
68      isTheStart = true;
69      idxBuffer = 0;
70      PLIB_TMR_Period16BitSet(TMR_ID_3, 65535);
71  }
72
73  //Sers à mettre que des default value sur le buffer de valeur IR

```

```

74  //--> Dans ce buffer des '1' et '0' seront stocker
75  //--> default value permet de détecter la fin des donnée
76  //Met des 0 sur le buffer de temps IR
77  void IR_ClearBufferIR (void){
78      uint8_t i;
79
80      for (i = 0; i < SIZE_OF_BUFFER_IR; i++){
81          dataReceiveIR[i] = DEFAULT_VALUE_BUFFER;
82          timeOfDataReceiveIR[i] = 32767;
83      }
84  }
85
86  //fonctions pour la gestions de l'IC
87  void IR_ClearIsTheStart (void){
88      isTheStart = false;
89  }
90  void IR_SetIsTheStart (void){
91      isTheStart = true;
92  }
93  bool IR_GetIsTheStart (void){
94      return (isTheStart);
95  }
96  //Permet de remplir valeur par valeur les buffers depuis l'isr de l'ic
97  void IR_PutDataInBuffer (uint16_t data, uint16_t timeOfData){
98      if (idxBuffer < (SIZE_OF_BUFFER_IR -1)){
99          dataReceiveIR[idxBuffer] = data;
100          timeOfDataReceiveIR[idxBuffer] = timeOfData;
101          idxBuffer++;
102      }
103  }
104
105  //Sers à détecter si une trame à été apprise (valeurs stocker dans les buffers)
106  bool IR_DataIsAvailable (void){
107      uint8_t i;
108      uint8_t j = 0;
109      bool dataIsAvailable;
110
111      for(i = 0; i <SIZE_OF_DATA_RC5; i ++){
112          if (dataReceiveIR[i] != DEFAULT_VALUE_BUFFER){
113              j ++;
114          }
115      }
116
117      if(j == SIZE_OF_DATA_RC5){
118          dataIsAvailable = true;
119      }
120      else{
121          dataIsAvailable = false;
122      }
123
124      return(dataIsAvailable);
125  }
126
127  //Sers à transférer les buffers dans l'app
128  //uint8_t IR_SaveData (uint16_t *bufferApp, uint16_t *TimeOfBufferApp)
129  void IR_SaveData (void)
130  {
131      //DÉCLARATION
132      //uint8_t i;
133
134      //=====
135      //uint8_t sizeofDataSave = 0;
136      static uint32_t Adress = 0x80;
137
138      //TACHE
139      //Transfert des données dans les buffers de l'app.c
140      // for(i = 0; i <SIZE_OF_BUFFER_IR; i++)
141      // {
142      //     //Si plus de datas, sortir de la boucle
143      //     if (dataReceiveIR[i] == DEFAULT_VALUE_BUFFER){
144      //         break;
145      //     }
146      //     else{

```

```

147 // //Transfer des datas
148 // *bufferApp = dataReceiveIR[i];
149 // *TimeOfBufferApp = timeOfDataReceiveIR[i];
150 // //Déplacer la case du buffer pour les prochains datas
151 // TimeOfBufferApp ++;
152 // bufferApp ++;
153 // sizeofDataSave ++;
154 // }
155 // }
156 // Sauvegarde de la trames dans la flash du uC
157 NVM_WriteBlock((uint32_t*)timeOfDataReceiveIR, 138, Adress);
158 // Sauvegarde de l'adresse du début de la trame enregistré dans FlashStartRow
159 FlashStartRow[Adress/0x80] = NVM_PROGRAM_PAGE + 0x80+(0x8C*((Adress/0x80)-1));
160 // Ecriture de la valeur de FlashFlag dans le début de la mémoire flash allué
161 NVMwriteRow(NVM_PROGRAM_PAGE, (uint32_t)&FlashStartRow[0]);
162 // Incréments la valeur d'adresse
163 Adress += 0x80;
164
165 //Effacer le buffer IR après le transfert
166 // IR_Initialize();
167
168 //Retourner le nombre de datas transférés
169 //return(sizeofDataSave);
170 }
171
172 //Si la trame n'est pas save, remettre les buffers à zero
173 void IR_NotSaveData (void){
174     IR_Initialize();
175 }
176
177 //Sers à activer le timer d'émission pour démarrer l'émission
178 void IR_SendCommandIR(void){
179
180     //Calibre le période counter à 0 pour une interruption la plus rapide possible
181     // Dans la version A TMR_ID_3 utilise
182     // PLIB_TMR_Period16BitSet(TMR_ID_2, 0);
183
184     //Lancement du timer
185     DRV_TMR2_Start();
186 }
187 //=====
188 // Ajout par Einar Farinas
189 //=====
190 // Pour contrôler si des trames sont enregistrés dans la flash
191 // si pas de trames enregistrés, prepare la mémoire pour la sauvegarde
192 void CheckFlashMemory(void)
193 {
194     uint32_t ReadFlag = 0;
195
196
197     // Lecture de la première adresse de la mémoire allué
198     NVM_ReadData(&ReadFlag ,0);
199     if(ReadFlag == FLASH_FLAG)
200     {
201         // Récouperer le nombre de trames enregistrés et leur adresse de début
202         NVM_ReadBlock((uint32_t*)FlashStartRow, 130, 0);
203     }
204     else
205     {
206         // Effacer la page dans la flash
207         Init_Page();
208         // Ecriture de la valeur de FlashFlag dans le début de la mémoire flash
209         NVMwriteRow(NVM_PROGRAM_PAGE, (uint32_t)&FlashStartRow[0]);
210     }
211 }
212 // Pour contrôler si des trames ont été enregistrés
213 bool CheckSavedCommand(uint8_t NbCommand)
214 {
215     if(FlashStartRow[NbCommand] != 0xFFFFFFFF)
216     {
217         return true;
218     }
219     else

```

```
220     {
221         return false;
222     }
223 }
224 // Pour reset le tableau de sauvegarde des adresses de chaque trame enregistré
225 void ResetTbFlashStart(void)
226 {
227     uint8_t i;
228
229     for(i = 1; i < sizeof(FlashStartRow); i++)
230     {
231         FlashStartRow[i] = 0xFFFFFFFF;
232     }
233 }
234
235 /* *****
236 End of File
237 */
238
```

```

1  /*****
2  MPLAB Harmony Application Header File
3
4  Company:
5  Microchip Technology Inc.
6
7  File Name:
8  gestIR.h
9  *****/
10
11  //-----|
12  //Réaliser par : Rodrigo Lopes dos Santos |
13  //-----|
14
15
16
17  #ifndef _GEST_IR_H
18  #define _GEST_IR_H
19
20
21
22  // ****
23  // ****
24  // Section: Included Files
25  // ****
26  // ****
27
28  #include <stdint.h>
29  #include <stdbool.h>
30  #include <stddef.h>
31  #include <stdlib.h>
32  #include "system_config.h"
33  #include "system_definitions.h"
34
35
36
37  // DOM-IGNORE-BEGIN
38  #ifdef __cplusplus // Provide C++ Compatibility
39
40  extern "C" {
41
42  #endif
43
44
45
46  // ****
47  // ****
48  // Section: Type Definitions
49  // ****
50  // ****
51
52  //CONSTANTE
53
54  //Le buffer contiendras des 1 et des 0, afin de détecter une fin de trame,
55  //255 sera une valeur au hazard choisi différente de 1 et 0 afin de
56  //détecter une fin de trame (dès la vue de 255)
57  #define DEFAULT_VALUE_BUFFER 255
58
59  //Défini la taille du buffer IR
60  #define SIZE_OF_BUFFER_IR 69 // Valeur version A => 100 //137
61
62  //taille d'une trame RC5 ( 14bits)
63  #define SIZE_OF_DATA_RC5 68 // Valeur version A => 14
64
65
66  #define FLASH_FLAG 0x12345678
67  //ENUMERATION
68
69
70  //STRUCTURE
71
72
73  //UNION

```

```

74
75
76
77 // *****
78 // *****
79 // Section: Function prototype
80 // *****
81 // *****
82
83 //fonction d'initialisation des buffers
84 void IR_Initialize (void);
85 //fonction qui reset/clear les buffers
86 void IR_ClearBufferIR (void);
87
88 //fonctions pour la gestions de l'IC
89 void IR_ClearIsTheStart (void);
90 void IR_SetIsTheStart (void);
91 bool IR_GetIsTheStart (void);
92
93 // =====
94 //          Ajout par Einar Farinas
95 void CheckFlashMemory(void);
96 // =====
97
98 //fonction pour remplir les buffers
99 void IR_PutDataInBuffer (uint16_t data, uint16_t timeOfData);
100
101 //fonction qui indique si les buffers contiennent une trame apprise
102 bool IR_DataIsAvailable (void);
103
104 //Transferer la trame apprise dans l'app
105 //uint8_t IR_SaveData (uint16_t *bufferApp, uint16_t *TimeOfBufferApp);
106 void IR_SaveData (void); // Modification par Einar Farinas
107
108 //fonction qui reset/clear les buffers si la trame ne doit pas etre save
109 void IR_NotSaveData (void);
110
111 //fonction qui permet d'activer l'émission d'une trame
112 void IR_SendCommandIR(void);
113
114 bool CheckSavedCommand(uint8_t NbCommand);
115 void ResetTbFlashStart(void);
116
117 #endif /* _GEST_IR_H */
118
119 //DOM-IGNORE-BEGIN
120 #ifdef __cplusplus
121 }
122 #endif
123 //DOM-IGNORE-END
124
125 /*****
126 End of File
127 */

```



```

1  /*****
2  MPLAB Harmony Application Source File
3
4  Company:
5  Microchip Technology Inc.
6
7  File Name:
8  Mc32gestI2cSeeprom.c
9  *****/
10
11  //-----|
12  //Réaliser par : Rodrigo Lopes dos Santos |
13  //-----|
14
15
16
17  // ****
18  // ****
19  // Section: Included Files
20  // ****
21  // ****
22
23  #include "Mc32gestI2cSeeprom.h"
24  #include "Mc32_I2cUtilCCS.h"
25  #include "app.h"
26
27
28
29  // ****
30  // ****
31  // Section: Constantes
32  // ****
33  // ****
34
35  // Définition pour MCP79411
36  #define MCP79411_EEPROM_R    0xAF        // MCP79411 address for read
37  #define MCP79411_EEPROM_W    0xAE        // MCP79411 address for write
38  // La EEPROM du 79411 est de 1 Kbits donc 128 octets
39  #define MCP79411_EEPROM_BEG  0x00        // addr. début EEPROM
40  #define MCP79411_EEPROM_END  0x7F        // addr. fin EEPROM
41
42  // Definitions du bus (pour mesures)
43  // #define I2C-SCK    SCL2/RA2        PORTAbits.RA2    pin 58
44  // #define I2C-SDA    SDA2/RA3        PORTAbits.RA3    pin 59
45
46
47
48  // ****
49  // ****
50  // Section: Functions
51  // ****
52  // ****
53
54  // Initialisation de la communication I2C et du MCP79411
55  // -----
56  void I2C_InitMCP79411(void)
57  {
58      bool Fast = true;
59      i2c_init( Fast );
60  } //end I2C_InitMCP79411
61
62
63  // Ecriture d'un bloc dans l'EEPROM du MCP79411
64  void I2C_WriteSEEPROM(void *SrcData, uint32_t EepromAddr, uint16_t NbBytes)
65  {
66      //      uint8_t i, j;
67      //      uint8_t* scrByteData = SrcData; //Evite de caster *SrcData
68      //      bool ack;
69      //      uint16_t saveNbBytes; // = NbBytes; //Save de NbBytes pour manipulations
70      //
71      //      uint8_t pageStart, pageEnd, nbPages;
72      //      //____Determine le nombre de page à écrire____
73      //      pageStart = EepromAddr / 8;

```

```

74 // pageEnd = (EEpromAddr + NbBytes) / 8;
75 // nbPages = pageEnd - pageStart;
76 //
77 // //_____SI TOUT TIEN SUR UNE PAGE_____
78 // if(nbPages == 0){
79 //     do{
80 //         i2c_start();
81 //         ack = i2c_write(MCP79411_EEPROM_W);
82 //     }while(!ack);
83 //
84 //     i2c_write(EEpromAddr);
85 //
86 //     for ( i = 0 ; i < NbBytes; i++ ) {
87 //         i2c_write(*scrByteData);
88 //         scrByteData++;
89 //     }
90 //
91 //     i2c_stop();
92 // }
93 // else{
94 //     for (j= 0 ; j < nbPages ; j++)
95 //     {
96 //         if ( NbBytes > 8)
97 //             saveNbBytes = 8;
98 //         else
99 //             saveNbBytes = NbBytes;
100 //
101 //         do{
102 //             i2c_start();
103 //             ack = i2c_write(MCP79411_EEPROM_W);
104 //         }while(!ack);
105 //
106 //         i2c_write(EEpromAddr);
107 //
108 //         for ( i = 0 ; i < saveNbBytes; i++ ) {
109 //             i2c_write(*scrByteData);
110 //             scrByteData++;
111 //         }
112 //         i2c_stop();
113 //
114 //         EEpromAddr += saveNbBytes;
115 //         NbBytes -= saveNbBytes;
116 //     }
117 // }
118
119 // Variables locales
120 // =====
121 // Pointeur de 8 bits pour envoyer chaque 8 bits à l'EEPROM de la RTCC
122 uint8_t *ptByteData = SrcData;
123 uint32_t AdresseCnt = EEpromAddr;
124 uint8_t i = 0;
125 bool ack;
126
127 // Start
128 do
129 {
130     i2c_start();
131     // Ecriture du byte de controle de la RTCC en mode W
132     ack = i2c_write(MCP79411_EEPROM_W);
133 }while(!ack); // Attente d'un ACK
134
135 do
136 {
137     ack = i2c_write(EEpromAddr); // Adresse de début de la mémoire
138 }while(!ack);
139
140 // envoie le nombre de bytes
141 for(i=0; i<NbBytes;i++)
142 {
143     do
144     {
145         ack = i2c_write(*ptByteData); // Envoie du byte
146     }while(!ack);

```

```

147 // Incrémentation du pointeur de datas pour sélectionner le prochain byte
148 ptByteData++;
149 AdresseCnt++;
150 // Si dernier byte de la page
151 if((AdresseCnt % 8) == 0)
152 {
153     // Faire un stop
154     i2c_stop();
155     // Faire un nouveau start avec la première adresse de la page suivante
156     do
157     {
158         i2c_start();
159         ack = i2c_write(MCP79411_EEPROM_W);
160     }while(!ack);
161     do
162     {
163         ack = i2c_write(AdresseCnt); // Adresse dans la mémoire
164     }while(!ack);
165     }
166 }
167 i2c_stop(); // I2C stop
168
169 } // end I2C_WriteSEEPROM
170
171
172 // Ecriture d'un byte dans l'EEPROM du MCP79411
173 void I2C_WriteOneByteSEEPROM(void *SrcData, uint32_t EEpromAddr/*, uint16_t NbBytes*/)
174 {
175     //uint8_t *p = SrcData;
176
177     i2c_start();
178     i2c_write(MCP79411_EEPROM_W);
179
180     //rajouter le dowhile
181     i2c_write(EEpromAddr); //00000000
182     i2c_write(*((uint8_t*)SrcData));
183     i2c_stop();
184
185 } // end I2C_WriteOneByteSEEPROM
186
187
188 // Lecture d'un bloc dans l'EEPROM du MCP79411
189 void I2C_ReadSEEPROM(void *DstData, uint32_t EEpromAddr, uint16_t NbBytes)
190 {
191     // int i;
192     // bool ack;
193     // //Realisation du start
194     // do{
195     //     i2c_start();
196     //     ack = i2c_write(MCP79411_EEPROM_W);
197     // }while(!ack);
198     // //Fin du start
199     // i2c_write(EEpromAddr); //Adress
200     // i2c_reStart(); //Restart
201     // i2c_write(MCP79411_EEPROM_R);
202     // for ( i = 0 ; i < NbBytes; i++ ) {
203     //     if (i == (NbBytes - 1)){
204     //         *((uint8_t*)DstData) = i2c_read(0); ///NO ACK
205     //     }
206     //     else{
207     //         *((uint8_t*)DstData) = i2c_read(1); // ACK
208     //     }
209     //     (((uint8_t*)DstData) ++);
210     //     DstData = ((uint8_t*)DstData) + 1;
211     // }
212     // i2c_stop();
213
214     // Variables locales
215     // =====
216     // Pointeur de 8 bits pour envoyer chaque 8 bits à l'EEPROM de la RTCC
217     uint8_t *ptByteData = DstData;
218     uint8_t AdresseCnt = 0;
219     uint8_t i = 0;

```

```

220
221     bool ack;
222
223     AdresseCnt = EEPromAddr;
224
225     // Start
226     do{
227         i2c_start();
228         // Mode écriture
229         ack = i2c_write(MCP79411_EEPROM_W);
230     }while(!ack);
231     // Adresse de debut de lecture
232     do{
233         ack = i2c_write(EEPromAddr);
234     }while(!ack);
235     // Restart
236     do{
237         i2c_reStart();
238         // Mode lecture
239         ack = i2c_write(MCP79411_EEPROM_R);
240     }while(!ack);
241
242     for ( i = 0 ; i < NbBytes; i++ )
243     {
244         AdresseCnt++;
245         // Si dernier byte envoie d'un Non ack
246         if (i == (NbBytes - 1))
247         {
248             *ptByteData = i2c_read(0); //NO ACK
249         }
250         // Si dernier byte de la page
251         else if((AdresseCnt % 8) == 0)
252         {
253             // Faire un no ACK
254             *ptByteData = i2c_read(0); //NO ACK
255             // Faire un stop
256             i2c_stop();
257             // Faire un nouveau start avec la première adresse de la page suivante
258             do{
259                 i2c_start();
260                 ack = i2c_write(MCP79411_EEPROM_W);
261             }while(!ack);
262             do{
263                 ack = i2c_write(AdresseCnt);
264             }while(!ack);
265             // Restart
266             do{
267                 i2c_reStart();
268                 ack = i2c_write(MCP79411_EEPROM_R);
269             }while(!ack);
270         }
271         // Ecriture d'un byte, envoi d'un ACK
272         else
273         {
274             *ptByteData = i2c_read(1); // ACK
275         }
276         ptByteData++;
277     }
278     i2c_stop();
279 } // end I2C_ReadSEEPROM
280
281
282 // Lecture d'un byte dans l'EEPROM du MCP79411
283 void I2C_ReadOneByteSEEPROM(void *DstData, uint32_t EEPromAddr/*, uint16_t NbBytes*/)
284 {
285     uint8_t *p = DstData;
286     bool ack;
287
288     do {
289         i2c_start();
290         ack = i2c_write(MCP79411_EEPROM_W);
291     } while (!ack);
292

```

```
293     i2c_write(EEpromAddr);
294     i2c_reStart();
295     i2c_write(MCP79411_EEPROM_R);
296     *p = i2c_read(0);
297
298     i2c_stop();
299 } // end I2C_ReadSEEPROM
300
301
302
303
304
305
306
307
308
```

```

1  #ifndef Mc32GestI2CSEEPROM_H
2  #define Mc32GestI2CSEEPROM_H
3  /*****
4      MPLAB Harmony Application Source File
5
6      Company:
7          Microchip Technology Inc.
8
9      File Name:
10         Mc32gestI2cSeeprom.h
11     *****/
12
13     //-----|
14     //Réaliser par : Rodrigo Lopes dos Santos |
15     //-----|
16
17
18
19     // ****
20     // ****
21     // Section: Included Files
22     // ****
23     // ****
24
25     #include <stdint.h>
26
27
28
29     // ****
30     // ****
31     // Section: Constantes
32     // ****
33     // ****
34
35     #define ADRESSE_STRUCT_EEPROM 0x00
36
37
38
39     // ****
40     // ****
41     // Section: Function prototype
42     // ****
43     // ****
44
45     //fonction d'initialisation
46     void I2C_InitMCP79411(void);
47
48     //fonctions de lecture de la eeprom
49     void I2C_ReadSEEPROM(void *DstData, uint32_t EEpromAddr, uint16_t NbBytes);
50     void I2C_ReadOneByteSEEPROM(void *DstData, uint32_t EEpromAddr);
51
52     //fonction d'écriture de la eeprom
53     void I2C_WriteSEEPROM(void *SrcData, uint32_t EEpromAddr, uint16_t NbBytes);
54     void I2C_WriteOneByteSEEPROM(void *SrcData, uint32_t EEpromAddr);
55
56     #endif
57

```

```

1  /*-----*/
2  // Mc32NVMUtil.c
3  /*-----*/
4  // Description : Utilitaire gestion écriture dans memoire programme (NVM)
5  // Auteur      : C. HUBER
6  // Création    : 17.02.2015
7  // Sur la base de l'exemple Harmony sous :
8  // C:\microchip\harmony\v1_00\apps\examples\peripheral\nvm\flash_modify
9  // Environnement d'origine :
10 // Version KIT   PCB 11020_B
11 // Version      : V1.61
12 // Compilateur   : XC32 V1.40 + Harmony 1.06
13 //
14 // LISTE DES MODIFICATIONS :
15 // 24.03.16      CHR correction dans NVMpageErase
16 // 08.03.17 v1.61 SCA correction attente fin écriture dans NVMwriteRow
17 //               compilateur xc32 1.42 avec Harmony 1.08
18 //
19 /*-----*/
20
21 #include "Mc32NVMUtil.h"
22 #include "peripheral\NVM\plib_nvm.h"
23 #include <sys/kmem.h>
24
25
26 // Row dans flash pour data
27 const uint32_t eedata_addr[DEVICE_ROW_SIZE_DIVIDED_BY_4] __attribute__((aligned(1024
28 ), space(prog)));
29
30 // Zone ram source pour copie row
31 uint32_t databuff[DEVICE_ROW_SIZE_DIVIDED_BY_4] __attribute__((coherent));
32
33
34 void Init_DataBuff(void)
35 {
36     int i;
37     for (i= 0 ; i < DEVICE_ROW_SIZE_DIVIDED_BY_4 ; i++) {
38         //databuff[i] = i * 10;
39         databuff[i] = 0xFFFFFFFF;
40     }
41 }
42
43
44 /*-----*/
45 /*
46  Function:
47     uint32_t virtualToPhysical (uint32_t address)
48
49  Summary:
50     Converts a virtual memory address to a physical one
51  */
52 uint32_t virtualToPhysical(uint32_t address)
53 {
54     return (address & 0x1FFFFFFF);
55 }
56
57 /*-----*/
58 /*
59  Function:
60     void NVMpageErase (uint32_t address)
61
62  Summary:
63     Erases a page in flash memory (4 KB)
64  */
65 void NVMpageErase(uint32_t address)
66 {
67     bool status;
68
69     // Base address of page to be erased
70     PLIB_NVM_FlashAddressToModify(NVM_ID_0, virtualToPhysical(address));
71
72     // Disable flash write/erase operations

```

```

73     PLIB_NVM_MemoryModifyInhibit(NVM_ID_0);
74
75     // Select page erase function & enable flash write/erase operations
76     PLIB_NVM_MemoryOperationSelect(NVM_ID_0, PAGE_ERASE_OPERATION);
77
78     // Allow memory modifications
79     PLIB_NVM_MemoryModifyEnable(NVM_ID_0);
80
81     // Write the unlock key sequence
82     PLIB_NVM_FlashWriteKeySequence(NVM_ID_0, 0x0);
83     PLIB_NVM_FlashWriteKeySequence(NVM_ID_0, 0xAA996655);
84     PLIB_NVM_FlashWriteKeySequence(NVM_ID_0, 0x556699AA);
85
86     // Start the operation
87     // PLIB_NVM_FlashWriteStart(NVM_ID_0);
88     PLIB_NVM_FlashEraseStart(NVM_ID_0);    // CHR correction du 24.03.2016
89
90     // Wait until the operation has completed
91     do {
92         status = PLIB_NVM_FlashWriteCycleHasCompleted(NVM_ID_0);
93     } while (status == false);
94
95     // Disable flash write/erase operations
96     PLIB_NVM_MemoryModifyInhibit(NVM_ID_0);
97
98     // Test si l'action c'est terminée prématurément
99     //if (PLIB_NVM_WriteOperationHasTerminated(NVM_ID_0) == false) {
100         // OK quittance par LED_7
101         // LED7_W = 0;
102     //}
103 }
104
105 /*
106
107 Function:
108     void NVMwriteRow(uint32_t address, uint32_t dataAddress)
109
110 Summary:
111     Writes a row in flash memory (2KB)
112 */
113 void NVMwriteRow(uint32_t destAddr, uint32_t srcAddr)
114 {
115     // Base address of row to be written to (destination)
116     PLIB_NVM_FlashAddressToModify(NVM_ID_0, virtualToPhysical(destAddr));
117
118     // Data buffer address (source)
119     PLIB_NVM_DataBlockSourceAddress(NVM_ID_0, virtualToPhysical(srcAddr));
120
121     // Disable flash write/erase operations
122     PLIB_NVM_MemoryModifyInhibit(NVM_ID_0);
123
124     // Select row write function & enable flash write/erase operations
125     PLIB_NVM_MemoryOperationSelect(NVM_ID_0, ROW_PROGRAM_OPERATION);
126
127     // Allow memory modifications
128     PLIB_NVM_MemoryModifyEnable(NVM_ID_0);
129
130     // Write the unlock key sequence
131     PLIB_NVM_FlashWriteKeySequence(NVM_ID_0, 0xAA996655);
132     PLIB_NVM_FlashWriteKeySequence(NVM_ID_0, 0x556699AA);
133
134     // Start the operation
135     PLIB_NVM_FlashWriteStart(NVM_ID_0);
136
137     //Correction SCA 8.03 : attente fin de l'écriture
138     while (!PLIB_NVM_FlashWriteCycleHasCompleted(NVM_ID_0));
139 }
140
141
142 uint32_t NVM_ArrayRead(uint32_t index)
143 {
144     uint32_t Res;
145     Res = eedata_addr[index];

```



```

146     return Res;
147 }
148
149
150 int NVMCheckLoop(void)
151 {
152     int i, stat;
153
154     stat = 0;
155     for (i=0; i < DEVICE_ROW_SIZE_DIVIDED_BY_4 ; i++) {
156         if (eedata_addr[i] != databuff[i]) {
157             stat = 100000 + i;
158         }
159     }
160     return stat;
161 };
162
163 // Cette fonction écrit un bloc de data au début de la zone flash
164 // PData correspond à l'adresse du bloc de donnée
165 // DataSize est la taille en octets du bloc de donnée
166
167 void NVM_WriteBlock(uint32_t *pData, uint32_t DataSize, uint32_t Page)
168 {
169     int i = 0, j, iMax;
170     static int j_old = 0;
171     uint32_t PageAddress = NVM_PROGRAM_PAGE + Page;
172
173     // Efface la page dans la flash
174     //NVMpageErase(NVM_PROGRAM_PAGE);
175
176     if ( (DataSize % 4) != 0 ) {
177         iMax = (DataSize / 4 ) + 1;
178     } else {
179         iMax = DataSize / 4 ;
180     }
181
182
183     while(i < iMax)
184     {
185         // Copie de 128 bytes (1 row) dans databuff
186         for(j = j_old; j < 32; j++)
187         {
188             if(i < iMax)
189             {
190                 i++;
191             }
192             else
193             {
194                 break;
195             }
196             // Copie de datas dans le buffer
197             databuff[j] = *pData;
198             pData++;
199         }
200         // Ecriture d'un raw dans la flash
201         NVMwriteRow(PageAddress, DATA_BUFFER_START);
202         // Incrémente l'adresse au début du prochain row pour écrire
203         // plusieurs bytes à la suite
204         PageAddress += 0x80;
205
206         if(j == 32)
207         {
208             j_old = 0;
209             Init_DataBuff();
210         }
211     }
212     // Sauvegarde de la valeur de j après avoir écrit une trame
213     // Ceci permet de pas écraser les 12 bytes écrits dans la row suivante
214     j_old = j;
215 }
216
217 void NVM_ReadBlock(uint32_t *pData, uint32_t DataSize, uint32_t Adress)
218 {

```

```

219     int i, iMax;
220
221     if ( (DataSize % 4) != 0 ) {
222         iMax = (DataSize / 4 ) + 1;
223     } else {
224         iMax = DataSize / 4 ;
225     }
226     for ( i = 0 ; i < iMax; i++ ) {
227         *pData = eedata_addr[i+Adress];
228         pData++;
229     }
230 }
231 void Init_Page(void)
232 {
233     // Efface la page dans la flash
234     NVMpageErase(NVM_PROGRAM_PAGE);
235 }
236 void NVM_ReadData(uint32_t *pData, uint32_t DataAdress)
237 {
238     *pData = eedata_addr[DataAdress];
239 }

```

```

1  /*-----*/
2  // Mc32NVMUtil.h
3  /*-----*/
4  // Description : Utilitaire gestion écriture dans memoire programme (NVM)
5  // Auteur      : C. HUBER
6  // Création    : 17.02.2015
7  // Sur la base de l'exemple Harmony sous :
8  // C:\microchip\harmony\v1_00\apps\examples\peripheral\nvm\flash_modify
9  // Environnement d'origine :
10 // Version KIT   PCB 11020_B
11 // Version      : V1.61
12 // Compilateur   : XC32 V1.40 + Harmony 1.06
13 //
14 // LISTE DES MODIFICATIONS :
15 // 24.03.16      CHR correction dans NVMpageErase
16 // 08.03.17 v1.61 SCA correction attente fin écriture dans NVMwriteRow
17 //               compilateur xc32 1.42 avec Harmony 1.08
18 //
19 /*-----*/
20
21 #ifndef Mc32NVMUtil_H
22 #define Mc32NVMUtil_H
23
24 #include <stdint.h>
25
26 /* Row size for pic32mx795 device is 512 bytes */
27 // #define DEVICE_ROW_SIZE_DIVIDED_BY_4 128
28 /* Page size for pic32mx795 device is 4 Kbytes */
29 // #define DEVICE_PAGE_SIZE_DIVIDED_BY_4 1024
30
31 //=====
32 // Modification pour le pic32mx130
33 //=====
34 /* Row size for pic32mx130 device is 128 bytes */
35 #define DEVICE_ROW_SIZE_DIVIDED_BY_4 32 // 128 / 4
36 /* Page size for pic32mx130 device is 1 Kbytes */
37 #define DEVICE_PAGE_SIZE_DIVIDED_BY_4 256 // 1024 / 4
38
39 extern const uint32_t eedata_addr[DEVICE_ROW_SIZE_DIVIDED_BY_4] __attribute__((
aligned(4096), space(prog)));
40
41 #define NVM_PROGRAM_PAGE ((uint32_t)&eedata_addr[0])
42 // prototypes des fonctions
43
44 // Zone ram source
45 extern uint32_t databuff[DEVICE_ROW_SIZE_DIVIDED_BY_4] __attribute__((coherent));
46
47 #define DATA_BUFFER_START ((uint32_t)&databuff[0])
48
49
50 void Init_DataBuff(void); // pour test
51 uint32_t NVM_ArrayRead(uint32_t index); // pour test
52
53 // Fonction de base
54 void NVMpageErase(uint32_t address);
55 void NVMwriteRow(uint32_t destAddr, uint32_t srcAddr);
56
57 // pour stockage et lecture d'une structure
58 void NVM_ReadBlock(uint32_t *pData, uint32_t DataSize, uint32_t Adress);
59 void NVM_WriteBlock(uint32_t *pData, uint32_t DataSize, uint32_t Page);
60 void Init_Page(void);
61 void NVM_ReadData(uint32_t *pData, uint32_t DataAdress);
62
63 #endif
64

```

```

1  /*****
2  System Interrupts File
3
4  File Name:
5      system_interrupt.c
6
7  Summary:
8      Raw ISR definitions.
9
10 Description:
11     This file contains a definitions of the raw ISRs required to support the
12     interrupt sub-system.
13
14 Summary:
15     This file contains source code for the interrupt vector functions in the
16     system.
17
18 Description:
19     This file contains source code for the interrupt vector functions in the
20     system. It implements the system and part specific vector "stub" functions
21     from which the individual "Tasks" functions are called for any modules
22     executing interrupt-driven in the MPLAB Harmony system.
23
24 Remarks:
25     This file requires access to the systemObjects global data structure that
26     contains the object handles to all MPLAB Harmony module objects executing
27     interrupt-driven in the system. These handles are passed into the individual
28     module "Tasks" functions to identify the instance of the module to maintain.
29 *****/
30
31 #include "framework/driver/tmr/drv_tmr_static.h"
32
33
34 // DOM-IGNORE-BEGIN
35 /*****
36 Copyright (c) 2011-2014 released Microchip Technology Inc. All rights reserved.
37
38 Microchip licenses to you the right to use, modify, copy and distribute
39 Software only when embedded on a Microchip microcontroller or digital signal
40 controller that is integrated into your product or third party product
41 (pursuant to the sublicense terms in the accompanying license agreement).
42
43 You should refer to the license agreement accompanying this Software for
44 additional information regarding your rights and obligations.
45
46 SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,
47 EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF
48 MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.
49 IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER
50 CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR
51 OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES
52 INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR
53 CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF
54 SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES
55 (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.
56 *****/
57 // DOM-IGNORE-END
58
59 //-----|
60 //Réaliser par : Rodrigo Lopes dos Santos |
61 //-----|
62
63
64 // ****
65 // ****
66 // Section: Included Files
67 // ****
68 // ****
69
70 #include "system/common/sys_common.h"
71 #include "app.h"
72 #include "system_definitions.h"
73

```

```

74 #include "peripheral/ic/plib_ic.h"
75 #include "peripheral/tmr/plib_tmr.h"
76 #include "gestEsp32.h"
77 #include "gestIR.h"
78 #include "Mc32Debounce.h"
79 #include "Mc32NVMUtil.h"
80
81 #define TIMEBTN          600
82 #define EXECUTE_TIME     20 // temps de execution en ms
83 // *****
84 // *****
85 // Section: Variables Globales
86 // *****
87 // *****
88
89 //Pour stocker les trames apprises IR avec le temps de chaque bit pour émettre
90 uint16_t ISRdataReceiveIR [SIZE_OF_BUFFER_IR] = {1,0,1,0,1,0,1,0,1,0,
91                                                    1,0,1,0,1,0,1,0,1,0,
92                                                    1,0,1,0,1,0,1,0,1,0,
93                                                    1,0,1,0,1,0,1,0,1,0,
94                                                    1,0,1,0,1,0,1,0,1,0,
95                                                    1,0,1,0,1,0,1,0,1,0,
96                                                    1,0,1,0,1,0,1,0,0
97 };
98 //uint16_t ISRdataReceiveIR [SIZE_OF_BUFFER_IR];
99 uint16_t ISRtimeOfDataReceiveIR [SIZE_OF_BUFFER_IR];
100
101 //Décripteur pour la librairies Debounce (anti-rebonds)
102 S_SwitchDescriptor DescrBtWps;
103 S_SwitchDescriptor DescrAppPMode;
104
105 // =====
106 //                               Ajout par Einar Farinas
107 // Flag pour permettre l'émission des IR avec une porteuse de 36kHz
108 // signal OOK généré
109 static bool EnableIR = false;
110
111 // =====
112
113 // *****
114 // *****
115 // Section: System Interrupt Vector Functions
116 // *****
117 // *****
118
119
120 //_____TIMER1_____
121 //Timer 1 pour Lancement du timer pour temps d'attente reception uart de l'esp32
122 //Interruption toutes les lms
123 void __ISR(_TIMER_1_VECTOR, IPL1AUTO) IntHandlerDrvTmrInstance0(void)
124 {
125     //Déclaration
126     static uint16_t countWait = 0; //Compteur pour attente réception cmd esp32
127     static uint16_t countWaitI2C = 0; //Compteur pour attente entre deux
128     communication I2c
129     static uint16_t countWPS = 0; //Compteur pour clignotement led lors du mode WPS
130     // static uint16_t saveData = 0; //Compteur qui compte le temps d'appui de save
131
132     static uint16_t btnAppTime = 0;
133     static uint8_t btnAppCnt = 0;
134     static uint16_t btnHoldTime = 0;
135     static uint8_t CntAppExecute = 0;
136
137     static bool hold = true;
138
139     //_____ANTI-REBONDS_____
140     DoDebounce(&DescrAppPMode, BtnModeAppStateGet());
141
142     btnAppTime = (btnAppTime+1) % TIMEBTN;
143     CntAppExecute = (CntAppExecute+1) % EXECUTE_TIME;
144
145     //_____CHANGEMENT DE MODE AVEC BtnModeApp_____
146     //Détecer de l'appuie d'entrée sortie dans le mode apprentissage

```

```

146 //Si appuie sur le bouton
147 if (DebounceIsPressed(&DescriAppPMode))
148 {
149     // =====
150     btnAppTime = 0;
151     if(btnAppCnt < 3)
152     {
153         btnAppCnt++;
154     }
155     //Effacer l'indicateur d'appuie car possibilité d'un appuie continu
156     DebounceClearPressed(&DescriAppPMode);
157     DebounceClearReleased(&DescriAppPMode);
158 }
159 // 4 cylces pour que l'entree passe de nouveau à 1
160 if((DebounceGetInput(&DescriAppPMode) == 0) && (APP_GetCurrentMode() ==
APP_MODE_APPRENTISSAGE) && (hold == true))
161 {
162     btnAppTime = 0;
163     if(btnHoldTime < 5000)
164     {
165         btnHoldTime++;
166     }
167     else
168     {
169         hold = false;
170         btnAppCnt = 4;
171     }
172 }
173 else if(DebounceGetInput(&DescriAppPMode) == 1)
174 {
175     hold = true;
176     btnHoldTime = 0;
177 }
178
179 if(((btnAppTime == (TIMEBTN-1)) && (btnAppCnt != 0))
180    || ((btnHoldTime == 5000) && (btnAppCnt == 4)))
181 {
182     APP_SetModeWasChanged();
183
184     // Selon le mode actuel, passage à l'autre mode
185     switch(btnAppCnt)
186     {
187         case 1:
188             switch(APP_GetCurrentMode())
189             {
190                 case APP_MODE_APPRENTISSAGE:
191                 {
192                     //Passage au mode run
193                     APP_UpdateMode(APP_MODE_RUN);
194                     IR_Initialize();
195                     break;
196                 }
197                 case APP_MODE_RUN:
198                 {
199                     //Passage au mode apprentissage
200                     APP_UpdateMode(APP_MODE_APPRENTISSAGE);
201                     IR_Initialize();
202                     break;
203                 }
204             }
205             break;
206         case 2:
207             if(APP_GetCurrentMode() == APP_MODE_APPRENTISSAGE)
208             {
209                 APP_UpdateSauvegarde(APP_SAVE_OK);
210                 SetSaveFlag();
211             }
212             break;
213         case 3:
214             if(APP_GetCurrentMode() == APP_MODE_APPRENTISSAGE)
215             {
216                 APP_UpdateSauvegarde(APP_SAVE_NOK);
217             }

```

```

218         break;
219     case 4:
220         if(APP_GetCurrentMode() == APP_MODE_APPRENTISSAGE)
221         {
222             APP_UpdateSauvegarde(APP_DELETE_ALL);
223             SetDeleteFlag();
224             LedModeAPP_GOff();
225         }
226         break;
227     default:
228         break;
229 }
230 btnAppCnt = 0;
231 }
232
233 //____ANTI-REBONDS____
234 //Anti-Rebonds software du bouton WPS
235 //Seulement si je ne suis pas en mode apprentissage
236 if (APP_GetCurrentMode() == APP_MODE_RUN){
237     DoDebounce(&DescrBtWps, BtnWPSStateGet());
238 }
239
240 //____ENTREE DANS LE MODE WPS____
241 //Détecter de l'appuie du bouton WPS pour activer le mode en question
242 //Si appuie sur le bouton
243 if (DebounceIsPressed(&DescrBtWps)){
244     //Indiquer l'entrée dans le mode WPS
245     APP_SetModeWPS();
246
247     //Effacer toutes valeurs du bouton WPS si jamais il y a eu un appui
248     //dans les autres modes
249     DebounceClearPressed(&DescrBtWps);
250     DebounceClearReleased(&DescrBtWps);
251 }
252
253
254 //____ATTENTE REPONSE ESP32____
255 //Attente pour reponse esp32 (remplir fifo complet)
256 if(Esp32_GetWaitForResponse() == true){
257     if(countWait < Esp32_GetConsigneToWaitForResponse())
258     {
259         countWait ++;
260     }
261     else
262     {
263         Esp32_ClearWaitForResponseESP32();
264         countWait = 0;
265     }
266 }
267
268 //____ATTENTE ENTRE DEUX COMMUNICATIONS I2C____
269 if(APP_GetWaitBetweenTwoTransmissionsI2C() == true){
270     if(countWaitI2C < APP_GetConsigneToWaitBetweenTwoTransmissionsI2C()){
271         countWaitI2C ++;
272     }
273     else{
274         APP_ClearWaitBetweenTwoTransmissionsI2C();
275         countWaitI2C = 0;
276     }
277 }
278
279 //____CLIGNOTEMENT WPS____
280 if(APP_GetModeWPS() == true){
281     //Attendre 500ms
282     if(countWPS < 500){
283         countWPS ++;
284     }
285     else{
286         // Toggle la LED wifi rouge
287         LedWifi_RToggle();
288
289         //Remise counter à 0
290         countWPS = 0;

```

```

291     }
292 }
293 if((CntAppExecute == 0))// && (GetAppState() == APP_STATE_WAIT)
294 {
295     APP_UpdateState(APP_STATE_IDLE);
296 }
297 // LedModeAPP_GToggle();
298 //Clear interrupt
299 PLIB_INT_SourceFlagClear(INT_ID_0,INT_SOURCE_TIMER_1);
300 }
301
302
303 //_____TIMER2_____
304 //Lié à l'IC1
305 void __ISR(_TIMER_2_VECTOR, IPL0AUTO) IntHandlerDrvTmrInstance1(void)
306 {
307     PLIB_INT_SourceFlagClear(INT_ID_0,INT_SOURCE_TIMER_2);
308 }
309
310 //_____TIMER3_____
311 //Sers à émettre l'IR
312 void __ISR(_TIMER_3_VECTOR, IPL2AUTO) IntHandlerDrvTmrInstance2(void)
313 {
314     static uint8_t idx = 0;
315     static bool getBuff = true;
316     static uint16_t nbCycle = 0;
317
318     if (getBuff)
319     {
320         APP_GetBufferIRApp(&ISRdataReceiveIR[0], &ISRtimeOfDataReceiveIR[0]);
321         nbCycle = APP_GetLengthOfData();
322         //NVM_ReadBlock((uint32_t*) ISRtimeOfDataReceiveIR, 138, 32);
323         nbCycle = 67;
324         getBuff = false;
325     }
326     if(ISRdataReceiveIR[idx] == 1){
327         TransmitIROn();
328         EnableIR = true;
329     }
330     else{
331         TransmitIROff();
332         EnableIR = false;
333     }
334
335     PLIB_TMR_Period16BitSet(TMR_ID_3,ISRtimeOfDataReceiveIR[idx]);
336
337     if (idx < nbCycle){
338         idx ++;
339     }
340     else{
341         idx = 0;
342         getBuff = true;
343         nbCycle = 0;
344         DRV_TMR2_Stop();
345     }
346
347     //_____CLEAR_INTERRUPT_____
348     PLIB_INT_SourceFlagClear(INT_ID_0,INT_SOURCE_TIMER_3);
349 }
350 //_____TIMER 4_____
351 // Pour la génération de la porteuse de 36 kHz
352 void __ISR(_TIMER_4_VECTOR, IPL1AUTO) IntHandlerDrvTmrInstance3(void)
353 {
354     if(EnableIR)
355     {
356         TransmitIRToggle();
357     }
358     else
359     {
360         TransmitIROff();
361     }
362     PLIB_INT_SourceFlagClear(INT_ID_0,INT_SOURCE_TIMER_4);
363 }

```



```

364
365 //_____INPUT CAPTURE(IC1)_____
366 //IC1 (Input Capture) pour la réception infrarouge
367 void __ISR(__INPUT_CAPTURE_1_VECTOR, IPL3AUTO) _IntHandlerDrvICInstance0(void)
368 {
369     //Pour calcul du temps entre deux interruption
370     static uint16_t oldFlank = 0;
371     uint16_t flank = 0;
372     uint16_t timeBetweenFlank = 0;
373     //Sers à stocker l'ancienne valeur (état)
374     static uint16_t oldData = 1;
375
376
377 //=====
378 // Clignotement de la LED verte pour indiquer la réception de la trame IR
379 LedModeAPP_GToggle(); // min 330 us
380 //=====
381
382 //____STOCKAGE & CALCULS DATAS____
383 // if(IR_GetIsTheStart()){
384 //     //Sers à éviter de compter la première interruption du IC1 lors de
385 //     //son démarrage (Ligne à 1 à ce moment / au repos)
386 //     //Trame reçu, flanc descendant et passage à l'état 0
387 //     if(PORTBbits.RB13 == 0){
388 //         IR_ClearIsTheStart();
389 //         flank = PLIB_IC_Buffer16BitGet(IC_ID_1);
390 //     }
391 // }
392 // else /*if((IR_GetIsTheStart() == false)) && (IR_GetMessageState() == true)*/
393 // {
394 // Récupérer la valeur du période compteur
395 flank = PLIB_IC_Buffer16BitGet(IC_ID_1);
396
397 if(IR_GetIsTheStart() == false)
398 {
399     //Calcul temps
400
401     //Valeur flanc descendant moins le montant pour le delta
402     timeBetweenFlank = flank - oldFlank;
403
404     // timeBetweenFlank * 1.6(1/(40MHz/Prescaler-64)) (=temps entre 2 periode
405     //compteur)(us)
406     //timeBetweenFlank = (timeBetweenFlank * 1.6); //us
407     //Voici comment convertir en us, dans mon cas pas besoin car le timer
408     //d'envoi IR
409     //a la même configuration de l'IC1 et donc le temps en période counter sera
410     //compris
411
412     //Stockage du data et son temps
413     IR_PutDataInBuffer(oldData, timeBetweenFlank);
414
415 }
416 else //if(flank > 100)
417 {
418     PLIB_IC_Buffer16BitGet(IC_ID_1);
419     IR_ClearIsTheStart();
420 }
421
422 oldFlank = flank;
423 // }
424
425 //____STOCKAGE DES VALEUR POUR LE PROCHAIN TRAITEMENT ET CALCUL____
426 //Stockage de la valeur afin qu'à la prochaine interruption on stock
427 //sa valeur ainsi que le calcul de son temps..
428 //Inversion de l'état car un 0 reçu = 1 envoyé
429 if(PORTBbits.RB13 == 1){
430     oldData = 0;
431 }
432 else{
433     oldData = 1;
434 }

```

```
434
435
436     //____CLEAR_INTERRUPTION____
437     PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_INPUT_CAPTURE_1);
438 }
439 // Pour récupérer les valeurs des trames sauvegardés dans la flash du uC
440 void GetIRData(uint32_t StartAdress)
441 {
442     StartAdress = 32+(35*(StartAdress-1));
443     NVM_ReadBlock((uint32_t*)ISRtimeOfDataReceiveIR, 138, StartAdress);
444 }
445 /*****
446 End of File
447 */
448
```

## RESUMÉ - **Projet**

Einar Farinas  
SLO  
2023

### Titre:

**2209B Commande IR domotique**

### Contexte et objectifs:

Reprise du projet de diplôme 2209A Commande IR domotique. Le but de ce projet est de concevoir un système électronique permettant d'activer ou désactiver à des périodes précises des appareils commandés par des infrarouges. Ce système sera dans boîtier de type « bloc secteur » pour être alimenté avec du 230V AC. Le système sera connecté au Wifi afin de pouvoir le commander et configurer à distance.

Dans cette nouvelle version, le CDC a été modifié pour améliorer l'HMI du projet. La tâche principale est de pouvoir avoir un système fonctionnel. Donc il faudra tester et vérifier les fonctions déjà faites. Dans le PCB et le schéma il y a des parties à corriger et à changer. Dans le soft, il faudra le modifier pour être compatible avec la nouvelle HMI. Aussi il faudra implémenter le capteur de température et faire des quelques modifications.

### Résultats obtenus et conclusion :

Dans la pré-étude et la phase de design les corrections dans le PCB et schéma ont été faites. Durant cette dernière étape du projet, le PCB a été réalisé en tenant en compte principalement la distance d'isolation des pistes pour les 230V ainsi que la position de l'antenne du module ESP32. Deux erreurs ont été faites lors de la réalisation du PCB. Une pin du  $\mu C$  n'a pas été reliée à la masse à cause d'un oubli lors de la copie du schéma dans la nouvelle version d'Altium. Cependant le  $\mu C$  fonctionne correctement. L'autre erreur, les pins Tx et Rx pour la communication avec l'ESP32 étaient inversés. Pour régler ces deux erreurs des pistes ont été coupées et des fils ont été brasés.

Pour la partie software du projet, comme prévu, comprendre le code a pris du temps. Surtout avec les bugs à corriger de la version précédente. Chaque partie du software a été testée pour vérifier le bon fonctionnement du système. Lors de tests, le code de l'émission IR ne fonctionnait pas correctement. Aussi l'enregistrement des trames fonctionnait une seule fois. Ces deux parties ont été corrigées et testées.

Par la suite, la sauvegarde des trames dans le flash du  $\mu C$  a été implémentée. Ensuite, l'envoi de commandes AT par Wifi a été testé. Pendant le test, 3 bugs ont été trouvés. Le premier un mauvais format de la date et l'heure. Le deuxième la sauvegarde des informations pour les commandes AT n'a pas été correctement implémentée dans la version précédente. Puis le dernier, le déclenchement des commandes AT qui se réalise une fois sur 2 commandes AT reçues. Les deux premiers bugs ont été corrigés, le dernier n'a pas encore été résolu.

Pour finir, la librairie pour le capteur de température a été commencée mais pas finie. Un test pour la lecture des valeurs a été réalisé avec cette librairie.

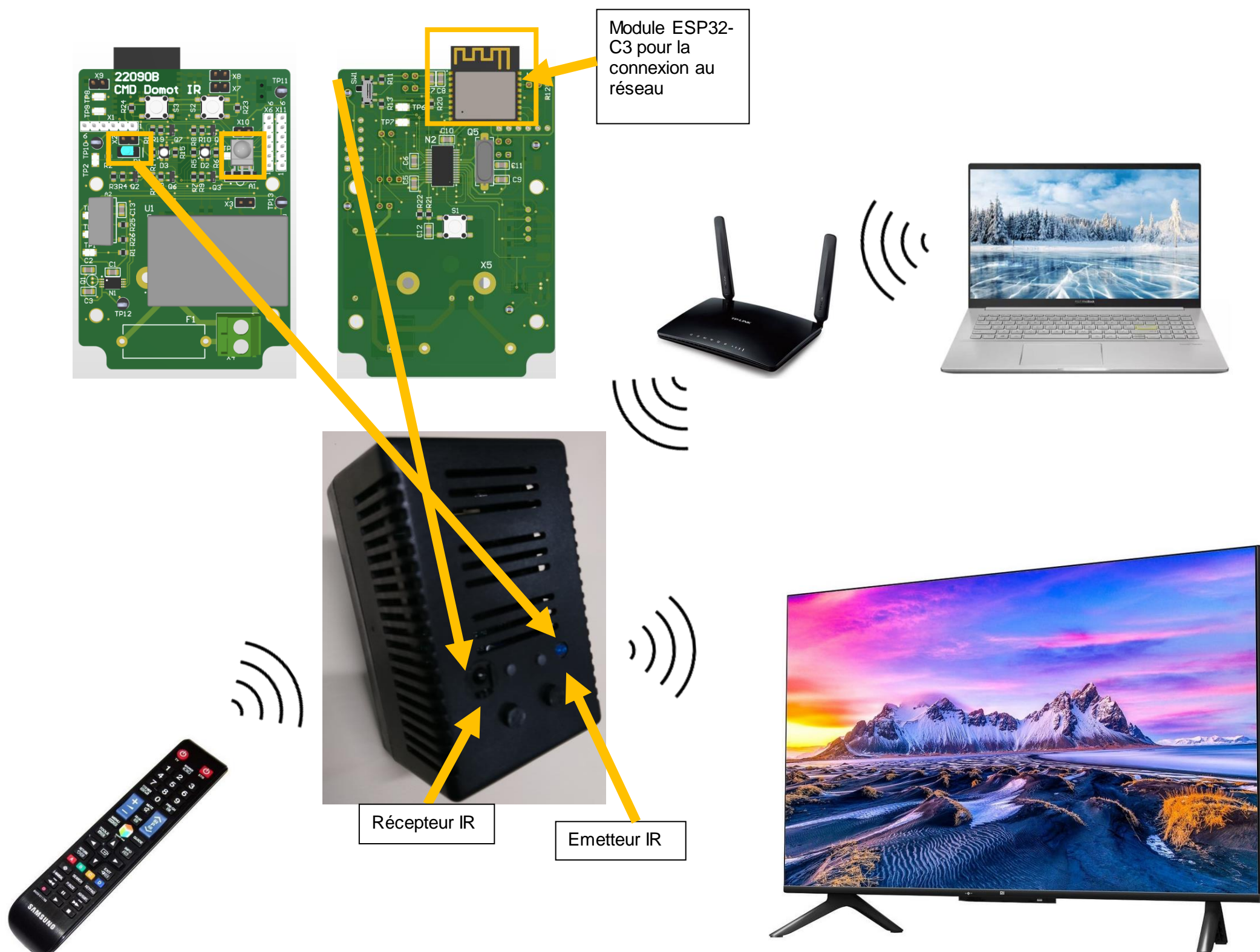
En conclusion quasiment toutes les tâches à réaliser ont été faites sauf pour la lecture du capteur de température & humidité. Le système est presque totalement fonctionnel il reste un seul bug à corriger pourrait être fait avant la présentation finale.

Maître(s) de projet: SCA JMO  
Entreprise mandataire: SCA



Cette affiche est destinée à être publiée sur notre site internet. Elle s'adresse à de futurs employeurs, de futurs étudiants et des entreprises à la recherche d'un partenariat avec l'ETML-ES (stage, diplôme, projet de semestre, ...)

Version B du projet avec des modifications et corrections effectuées. Le but de ce projet est de concevoir un système électronique permettant d'activer ou désactiver à des périodes précises des appareils commandés par l'infrarouge comme la TV / Radio / lumière. Ce système sera installé dans un boîtier style « bloc secteur » et sera alimenté par le réseau électrique. Le système sera connecté au réseau local par Wifi afin de pouvoir le



Ce projet m'a permis de découvrir le fonctionnement de trames IR et aussi le module ESP32. Ce projet est quasiment fonctionnel, il reste encore un bug à corriger. Durant la réalisation de ce projet plusieurs corrections ont été faites. Pour la suite du projet, il faudra développer une application permettant de configurer et commander facilement le système.

# Projet ETML-ES – Modification

<b>PROJET :</b>	Commande IR Domotique		
<b>Entreprise/Client :</b>	Serge Castoldi	<b>Département :</b>	-
<b>Demandé par (Prénom, Nom) :</b>	ETML-ES	<b>Date :</b>	16.06.2023
<b>Objet (No ou réf, pièce, PCB...)</b>	2209B		
<b>Version à modifier :</b>	B		

<b>Auteur (ETML-ES) :</b>	Einar Farinas Arze	<b>Filière :</b>	SLO2
<b>Nouvelle version :</b>	-	<b>Date :</b>	16.06.2023

## 1 Description ou justification

Dans la version B du projet, deux erreurs ont été fait sur le PCB. 1 pin n'est pas branché à la masse et les Tx et Rx du UART sont inversés.

## 2 Référence conception

K:\ES\PROJETS\SLO\2209x\_CommandeIRDomotique\2209B\doc

## 3 Détail des modifications

#	Description	Fait	Approuvé
1	Pin 19 n'est pas branché au GND dans le PCB 22090B Correction fait dans une nouvelle version du PCB => 22091B	OUI	
2	Rx et Tx du UART inversés Correction fait dans une nouvelle version du PCB => 22091B	OUI	
3			
4			

## 4 Remarques

Dans le PCB 22090B, les erreurs ont été fait en coupant les pistes et en brasant des fils au pin.  
Une nouvelle version du PCB a été faite avec les correction requises.

The module should be placed as close to the edge of the base board as possible. The PCB antenna area should be placed outside the base board whenever possible. In addition, the feed point of the antenna should be closest to the board, as shown in Figure 13 and Figure 12.

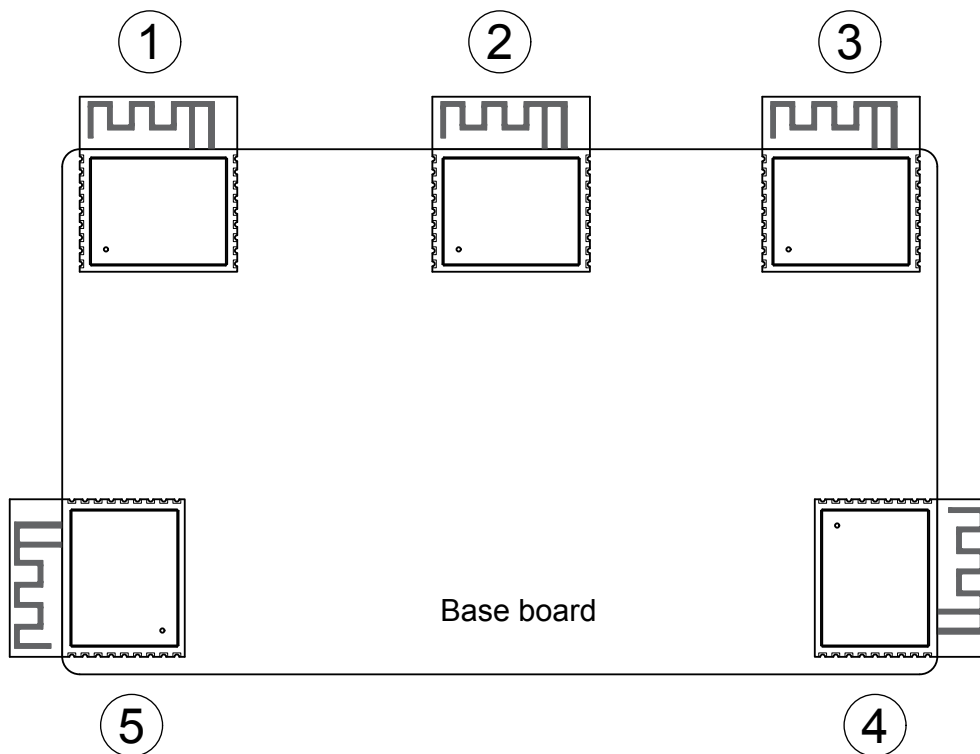


Figure 12: Placement of ESP32-C3 Modules on Base Board. Antenna Feed Point on the Right

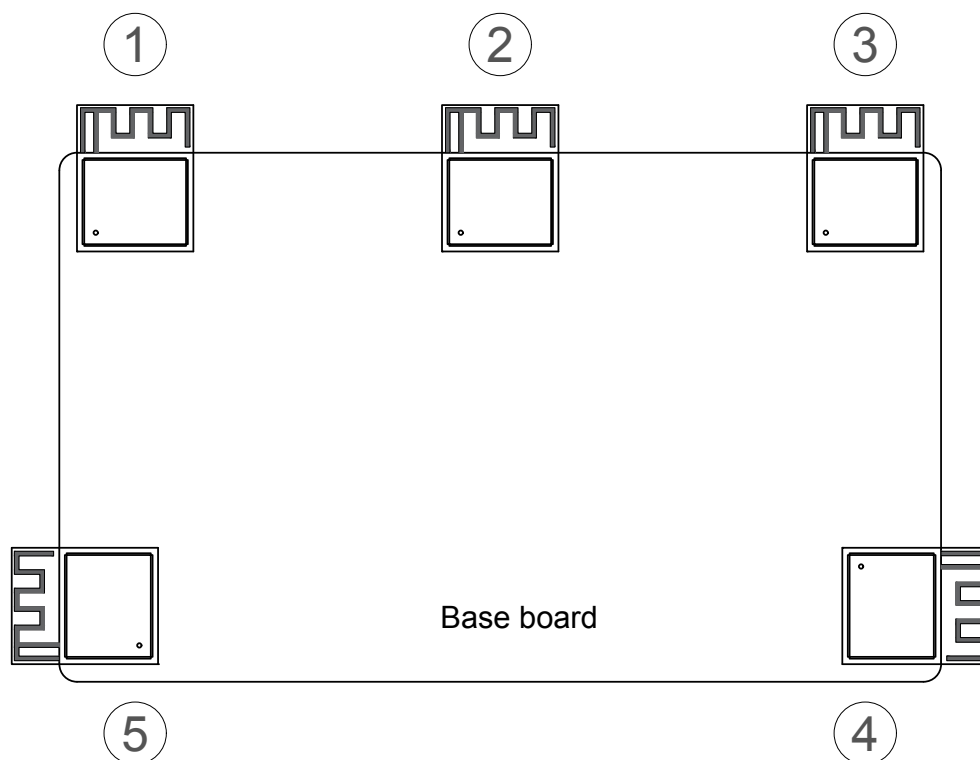


Figure 13: Placement of ESP32-C3 Modules on Base Board. Antenna Feed Point on the Left

**Note:**

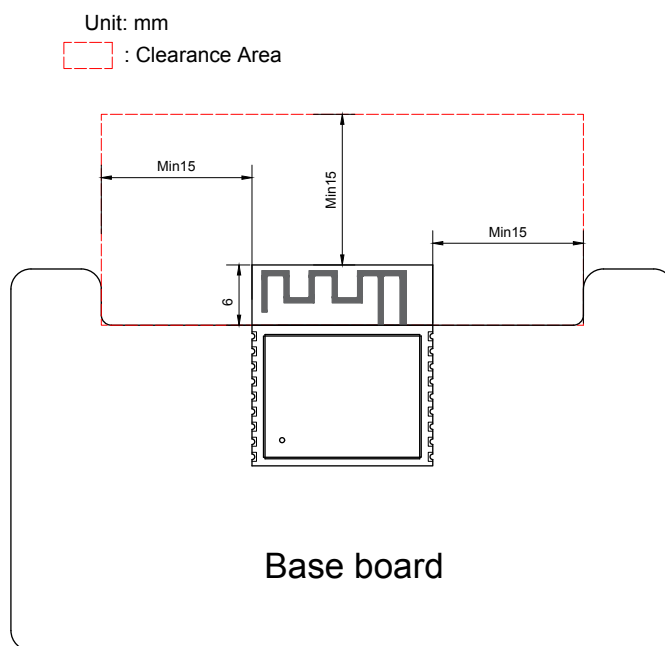
In Figure 12, the recommended position of ESP32-C3 modules (feed point on the right) on the base board should be:

- Position 3, 4: Highly recommended;
- Position 1, 2, 5: Not recommended.

In Figure 13, the recommended position of ESP32-C3 modules (feed point on the left) on the base board should be:

- Position 1, 5: Highly recommended;
- Position 2, 3, 4: Not recommended.

If the positions recommended are not feasible, please make sure that the module is not covered by any metal shell. Besides, the antenna area of the module and the area 15 mm outside the antenna should be kept clean, (namely no copper, routing, components on it) as shown in Figure 14.



**Figure 14: Keepout Zone for ESP32-C3 Module's Antenna on the Base Board**

If there is base board under the antenna area, it is recommended to cut it off to minimize its impact on the antenna. When designing an end product, pay attention to the impact of enclosure on the antenna.