

ANNEXES DU PROJET DOSIMETRE

Riehl Alexandre



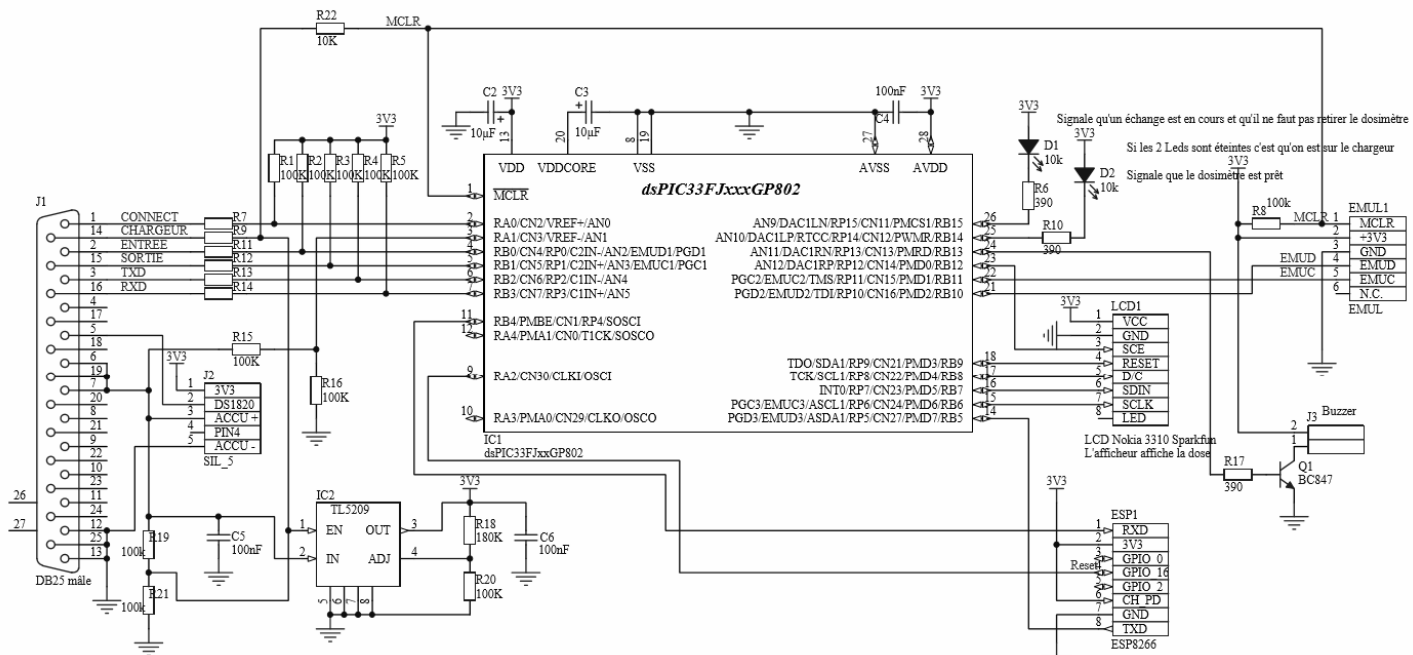
Lycée des Métiers
Paul Emile Victor

26 MAI 2016

Lycée

COUFFIGNAL


1. SCHEMA DE LA CARTE DU DOSIMETRE



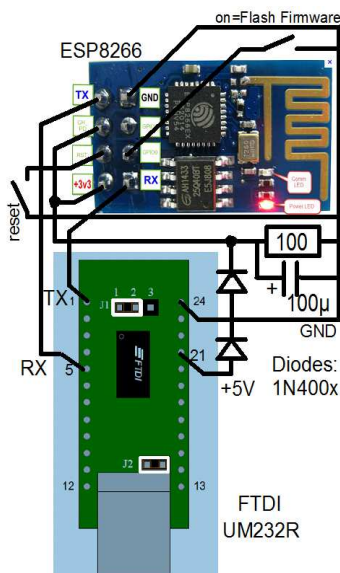
Si CHARGEUR est à 0 le dosimètre est désalimenté

Si CONNECT est à 1 le programme dosimètre est activé mais le comptage ne commence qu'après la phase ENTREE et ne s'arrête qu'après la phase SORTIE
Si CONNECT et ENTREE sont à 0 le programme de chargement des données de la personne est activé
Si CONNECT et SORTIE sont à 0 le programme de téléchargement des données de la personne est activé
L'identification du dosimètre se fait par l'adresse unique des DS18B20 qui mesurent la température des accus

Si CONNECT est à 1 le programme dosimètre est activé

* Dosimètre 4 mars			Lycée Louis Couffignal 11, route de la Fédération 67100 STRASBOURG	
* Classe: *				
Date: 03/03/2016	Time: 12:42:12	Sheet * of *		
File: W:\client d'Althum DXP2004 projets divers\Projet 2016\Doni dosimètre\Dosimètre.SCHDOC				

3. MANUEL DE MISE EN ŒUVRE DU WIFI



Mise à jour du firmware

Pour configurer le module en mode « flashage », fermer l'interrupteur nommé « Flash Firmware » sur le schéma ci-joint puis appuyé rapidement sur « reset ». Avec l'utilitaire "esp8266_flasher.exe", sélectionner le numéro port COM-USB (Partie périphériques sur le PC) puis flasher le module avec le fichier binaire : « ai-thinker-v1.1.1.bin ».

A la fin du transfert, ouvrir l'interrupteur "Flash Firmware" et appuyer à nouveau sur "reset". Le module est normalement prêt au fonctionnement.

Généralités sur le fonctionnement du module

La liaison série du module est par défaut à 115200bauds. Si des erreurs d'affichage sont constatées, on aura intérêt à ralentir le débit à 57600 bauds avec la commande AT+UART_DEF. La connexion WIFI ne se fait plus correctement pour des vitesses inférieures.

Toutes les commandes AT envoyées au module par la liaison série doivent se terminer par CR LF (« \r\n ») pour être traitées par le module.

L'utilitaire explorer.jar permet de tester l'envoi des commandes sur le module, pour les détails des commandes utiles consultez la partie 4 de l'annexe.

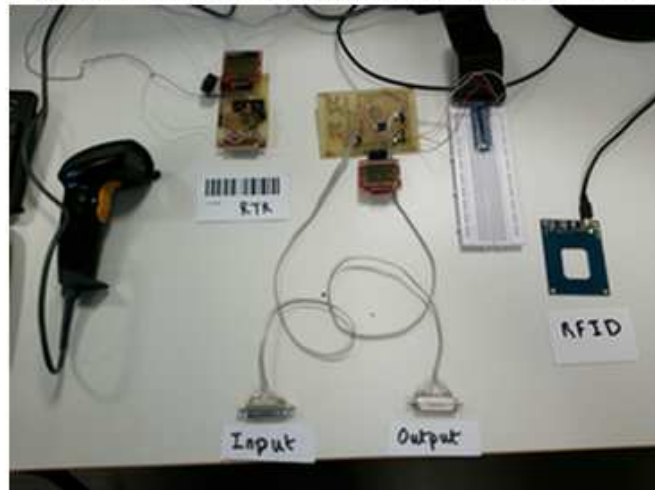
4. COMMANDES AT POUR LE MODULE WIFI

ESP8266 AT Command Set

Function	AT Command	Response
Working	AT	OK
Restart	AT+RST	OK [System Ready, Vendor:www.ai-thinker.com]
Firmware version	AT+GMR	AT+GMR 0018000902 OK
List Access Points	AT+CWLAP	AT+CWLAP +CWLAP:{4,"RocheFortSurLac",-38,"70:62:b8:6f:6d:58",1} +CWLAP:{4,"LiliPad2.4",-83,"f8:7b:8c:1e:7c:6d",1} OK
Join Access Point	AT+CWLAP? AT+CWLAP="SSID","Password"	Query AT+CWLAP? +CWLAP:"RocheFortSurLac" OK
Quit Access Point	AT+CWLAP=? AT+CWLAP	Query OK
Get IP Address	AT+CIFSR	AT+CIFSR 192.168.0.105 OK
Set Parameters of Access Point	AT+ CWSAP? AT+ CWSAP= <ssid>,<pwd>,<chl>,<ecn>	Query ssid, pwd chl = channel, ecn = encryption
WiFi Mode	AT+CWMODE? AT+CWMODE=1 AT+CWMODE=2 AT+CWMODE=3	Query STA AP BOTH
Set up TCP or UDP connection	AT+CIPSTART=? (CIPMUX=0) AT+CIPSTART = <type>,<addr>,<port> (CIPMUX=1) AT+CIPSTART= <id><type>,<addr>,<port>	Query id = 0-4, type = TCP/UDP, addr = IP address, port= port
TCP/UDP Connections	AT+ CIPMUX? AT+ CIPMUX=0 AT+ CIPMUX=1	Query Single Multiple
Check join devices' IP	AT+CWLIF	
TCP/IP Connection Status	AT+CIPSTATUS	AT+CIPSTATUS? no this fun
Send TCP/IP data	(CIPMUX=0) AT+CIPSEND=<length>; (CIPMUX=1) AT+CIPSEND= <id>,<length>	
Close TCP / UDP connection	AT+CIPCLOSE=<id> or AT+CIPCLOSE	
Set as server	AT+ CIPSERVER= <mode>[,<port>]	mode 0 to close server mode; mode 1 to open; port = port
Set the server timeout	AT+CIPSTO? AT+CIPSTO=<time>	Query <time>0~28800 in seconds
Baud Rate*	AT+CIOBAUD? Supported: 9600, 19200, 38400, 74880, 115200, 230400, 460800, 921600	Query AT+CIOBAUD? +CIOBAUD:9600 OK
Check IP address	AT+CIFSR	AT+CIFSR 192.168.0.106 OK
Firmware Upgrade (from Cloud)	AT+CIUPDATE	1. +CIPUPDATE:1 found server 2. +CIPUPDATE:2 connect server 3. +CIPUPDATE:3 got edition 4. +CIPUPDATE:4 start update
Received data	+IPD	(CIPMUX=0): + IPD, <len>: (CIPMUX=1): + IPD, <id>, <len>: <data>
Watchdog Enable*	AT+CSYSWDTENABLE	Watchdog, auto restart when program errors occur: enable
Watchdog Disable*	AT+CSYSWDTDISABLE	Watchdog, auto restart when program errors occur: disable

5. PROCÉDURES D'ENTRÉE EN ZONE CONTRÔLÉ

Vue d'ensemble de la simulation du portique d'entrée



Étape 1 : Brancher le dosimètre sur "Input"



Étape 2 : Passer le badge RFID au lecteur



Étape 3 : Scanner la fiche individuelle



Étape 4 : Entrer en ZC



6. MANUEL D'UTILISATION DE LA TELECOMMANDE

Étape 1 : Menu du dosimètre

Touche retour arrière : *

Appuyez sur la touche 1 du clavier



Étape 2 : Sélectionner un dosimètre
en composant son numéro puis #



Étape 3 : Gestion du dosimètre



Deboost le débit
Booster le débit
RAZ le débit

7. CODE SOURCE

7.1. Dosimètre

configure.c

```
#include <p33Fxxxx.h>    // A modifier suivant le uC utilisé
#include "defines.h"

extern long sendCount;
extern int Tdelais;
extern long countStepSeq;
extern long countStepIncrease;
extern int alarmTimeout;
extern int alarmState;

/*-----
Function    : void InitCPU_Clock(void)
Description: Initialisation de l'horloge CPU
    - Les bits de configuration (déjà programmés) réalisent les sélections
      suivantes :
        - horloge de base = FRC = 7.37MHz à 2%
        - PLL activée (sélection S1)
    - Les registres du module horloge sont affectés par cette fonction pour
      obtenir :
        - FRCDIVN = 7,37MHz
        - Fosc = 73,7 MHz (7,37*10)
        - FCY  = 36,85MHz à 2% près
-----*/
void InitCPU_Clock(void)
{
    CLKDIVbits.FRCDIV=0;    // FRC non divisée : 7,37MHz
    CLKDIVbits.PLLPRE=0;    // Prédvision par 2 avant PLL : 3,685MHz
    PLLFBDbits.PLLDIV=40-2; // PLL : multiplication par 40 : 147,4MHz
    CLKDIVbits.PLLPOST=0;   // Division par 2 après PLL : Fosc = 73,7MHz
}

/*-----
```



```

Nom          : void InitTimer1 (void)
Description : Programme d'initialisation du timer 1 en base de temps 1ms
Arguments   : aucun
Valeur renvoyée : aucune
*
*
* Temps d'actualisation : FcY / 256 / 144 = 1 ms
-----*/
void InitTimer1 (void)
{

    PR1=144; //base de temps
    T1CON=0x8030; // TCKPS=10 FcY div 256 ,Timer on, TGATE=0 TCS=0
    T1CONbits.TON=1 ; //timer on
    IEC0bits.T1IE=1 ; // validation des interruptions du Timer 1
    TMR1=0;
}
/*-----
Nom          : void InitBuzzer (void)
Description : Programme d'initialisation du buzze
Arguments   : aucun
Valeur renvoyée : aucune
*
*
-----*/
void InitBuzzer (void)
{
    OC1CON = 0x0005;
    OC1R = 0;
    OC1RS = 620;
    RPOR6 = 0x1200;
    /*PR2=1240; //base de temps 100ms
    T2CON=0x8010; // TCKPS=10 FcY div 16 ,Timer on, TGATE=0 TCS=0
    TMR2=0;*/
}

void _ISR _T1Interrupt(void) //
```

```
{
    TMR1=0; // RAZ de TMR1
    IFS0bits.T1IF=0; // Acquitement interruption
    Tdelais++;
    if(alarmState)
    {
        if(alarmTimeout == 0) StopSound();
        else if(alarmTimeout > 0) alarmTimeout--;
    }
}

/*-----
Nom          : void Delay(int delais)
Description : Programme de temporisation
Arguments    : durée en ms
Valeur renvoyée : aucune
-----*/
void Delay(int delais)
{
    Tdelais=0;
    while (Tdelais < delais){}
}
```

main.c

```
/*-----
Librairies et fichiers inclus
-----*/

#include <p33Fxxxx.h>
#include <stdlib.h>
#include <string.h>
#include "wifi.h"
#include "defines.h"

/*~~~~~ Bits de configuration ~~~~~*/
_FBS(RBS_LARGE_RAM & BSS_NO_FLASH & BWRP_WRPTECT_OFF);
// Large Sized Boot Ram
```

```

// No Boot Segment Program Memory
// Write Protect Disabled
_FSS(RSS_NO_RAM & SSS_NO_FLASH & SWRP_WRPROTECT_OFF);
// Secure Segment Data Ram: No Secure Ram
// Secure Segment Program Memory: No Secure Segment
// Write Protect Disabled
_FGS(GSS_OFF & GCP_OFF & GWRP_OFF);
// Code Protect off
// Write Protect Disabled
_FOSCSSEL(FNOSC_FRCPLL & IESO_OFF);
// Fast RC oscillator w/ divide and PLL
// Two-speed Oscillator Startup disabled
_FOSC(FCKSM_CSECMD & IOL1WAY_OFF & OSCIOFNC_OFF & POSCMD_NONE);
// Clock Switching is enabled and Fail Safe Clock Monitor is disabled
// Single configuration for remappable I/O disabled
// OSC2 Pin Function: OSC2 is RA3
// Horloge externe inhibée
_FWDT(FWDTEN_OFF & WINDIS_OFF);
// Watchdog Timer disabled by user software
// Windowed WDT disabled
_FPOR(ALT12C_OFF & FPWRT_PWR128);
// I2C mapped to SDA1/SCL1
// Power-on Reset Value: 128mS
_FICD(JTAGEN_OFF & ICS_PGD1);
// JTAG is disabled
// ICD communicate on PGEC1/EMUEC1
/*~~~~~*/

/*-----
Déclarations des équivalences
-----*/
#define FCY          36850000 // Fosc= ... MHz et Fcy= ... MHz

/*-----
Déclarations des variables globales
-----*/

char espResponse[SIZE_ESPBUFF]={0}; // Tableau contenant la réponse aux commandes AT
(RESET = 270 car.)
int countU1=0; // Compteur permettant d'accéder au tableau espResponse

char seqResponse[SIZE_SEQBUFF]={0}; // Tableau contenant la réponse envoyé par la
telecommande
int countU2=0; // Compteur permettant d'accéder au tableau seqReponse

int Tdelais=0; // Compteur pour le timer
/*
Structure contenant les variables pour configurer le wifi.
*/

extern Wifi configurationWifi; // Structure pour configurer le wifi

/*
* Variables contenant les informations de l'utilisateur
*/

int userID = 0, activityID = 0;

/*

```

```

Variables contenant les informations de l'activités

    ded: valeur du débit équivalent dose fixé en ammont par l'instructeur
    coeffExpo: coefficient d'exposition ""
    limitDose: Dose théorique calculée pour l'activité ""

    Alarme sur débit : ded > 1.20 * ded
    Alarme sur dose : dose > 1.20 * dose théorique
    Pré-alarme sur dose : dose > 1.05 * dose théorique
*/

float limitDed, coeffExpo, limitDose, impuls;

/*
Variables contenant les valeurs de retour de l'activité et de calcul des valeurs
*
* dedMax : débit equivalent dos maximum enregistré lors de l'activité
* userDose : dose accumulé par la personne lors de l'activité
* numAlarm: nombre total d'alarme déclenchée
* typeAlarm : type d'alarme déclenchée (DED, PREDOSE, DOSE)
* alarmState : variable utile au fonctionnement interne, elle définit l'état de l'alarme
* alarmPreDose : "" "", elle définit si l'alarme
predose à déjà été déclenché ou no
* alarmTimeout : "" "", elle contient la valeur en ms
du temps de fonctionnement de l'alarme
* time : "" "", elle contient la valeur du
temps écoulés depuis le début de l'activité.
* timeDed : "" "", elle contient la fréquence
des BIP lors d'une alarme de débit
*/
float dedMax, userDose;
int numAlarm, alarmState, alarmPreDose, alarmPreDed, alarmDose;
char typeAlarm[32] = {0};
long alarmTimeout;
float time, timeDed;

/*-----
Retourne 1 si le caractère est un chiffre sinon 0
-----*/

int IsCharDigit(char c)
{
    switch(c)
    {
        case '1': return 1;
        case '2': return 1;
        case '3': return 1;
        case '4': return 1;
        case '5': return 1;
        case '6': return 1;
        case '7': return 1;
        case '8': return 1;
        case '9': return 1;
        default : return 0;
    }
    return 0;
}

/*-----

```

```

    void PlaySound(int timeout)
    *
    *   Joue un son
    *   timeout = temps en ms avant la fin du son (-1 = infini)
    -----*/
void PlaySound(long timeout)
{
    if(!alarmState)
    {
        RPOR6 = 0x1200;
        alarmTimeout = timeout;
        alarmState = 1;
    }
}

/*-----
    void StopSound()
    *
    *   Arrête le son
    -----*/
void StopSound()
{
    if(alarmState)
    {
        RPOR6 = 0;
        alarmTimeout = 0;
        alarmState = 0;
    }
}

/*-----
    int IsAlarmRing()
    *
    *   Retourne l'état de l'alarme
    -----*/
int IsAlarmRing()
{
    return (alarmState);
}

/*-----
    Fonction principale
    -----*/

int main(void)
{
    InitCPU_Clock();
    InitTimer1();
    InitBuzzer();

    RPOR6 = 0; // Eteindre le buzzer, aucun signal n'est injecté sur la patte

    AD1PCFGL=0x01FF; // Pas d'entrée analogique

    InitUART1(57600);
    InitUART2(57600);
    LcdInit();
    InitWifi(4);

    Delay(1000);

```

```

StartServer();
Delay(2000);

start:

userID = 0, activityID = 0, limitDed = 0, coeffExpo = 0, limitDose = 0, impuls = 0,
numAlarm = 0, dedMax = 0;
int i = 0;
for(i;i < sizeof(typeAlarm);i++) typeAlarm[i] = 0;

StopSound();

Fill_Lcd(0);
LcdGotoXY(0,0);
LcdString("En attente d'authentification...");

ClearUartSeqBuffer();

while(!userID)
{
    /*
        Format d'authentification : x AUTH
    */
    if(strstr(seqResponse, "AUTH"))
    {
        char *end;
        userID = strtol(seqResponse, &end, 10); // On extrait l'entier
        break;
    }
}

Fill_Lcd(0);
LcdGotoXY(0,0);
LcdString("En attente de RTR...");

ClearUartSeqBuffer();

while(!activityID || !limitDed || !coeffExpo || !limitDose)
{
    /*
        Format d'activité : DED x.xx DOSE x.xxx COEFF x.xxx ACT
    */
    if(strstr(seqResponse, "ACT"))
    {
        /*
            Extraction des parties flottantes dans la chaine de caractère
        */
        char *end;
        activityID = strtol (seqResponse, &end, 10);
        limitDed = (float)strtod(end, &end);
        dedMax = limitDed;
        limitDose = (float)strtod(end, &end);
        coeffExpo = (float)strtod(end, &end);

        Fill_Lcd(0);
        LcdGotoXY(0,0);
    }
}

```

```

        char activity[128] = {0};
        snprintf(activity, sizeof(activity), "%d %f %f %f", activityID, limitDed,
limitDose, coeffExpo);
        LcdString(activity);
        Delay(5000);
        break;
    }
}

ClearUartEspBuffer();

Fill_Lcd(0);
LcdGotoXY(0,0);
LcdString("Debranchez le dosimetre...");

while(_RA0 == 0) {}

while(1)
{
    // Si la patte _RA0 est à l'état bas, cela signifie que le dosimètre est branché en
RS232
    if(_RA0 == 0)
    {
        /*
        Formatage et envoi des valeurs de sortie sur l'UART2
        */
        Delay(1000);
        char activity[128];
        snprintf(activity, sizeof(activity), "%d %d %f %f %d DED+DOSE", userID,
activityID, userDose, dedMax, numAlarm);
        SendData2(activity);
        Fill_Lcd(0);
        LcdGotoXY(0,0);
        LcdString(activity);
        Delay(5000);
        goto start;
    }
    Fill_Lcd(0);
    // Si un boost est detecté sur la liaison série
    if(strstr(espResponse, "!BOOST"))
    {
        LcdGotoXY(0,0);
        LcdString("Boost");

        ClearUartEspBuffer();
        impuls++;
    }
    // Si un deboost est detecté sur la liaison série
    else if(strstr(espResponse, "!DEBOOST"))
    {
        LcdGotoXY(0,0);
        LcdString("deboost");

        ClearUartEspBuffer();
        if((impuls-1) >= 0) impuls--;
    }
    // Si un raz est detecté sur la liaison série
    else if(strstr(espResponse, "!RAZ"))
    {
        LcdGotoXY(0,0);
    }
}

```



```

        LcdString("raz");

        ClearUartEspBuffer();
        impuls=0;
    }

    time+=0.5;

    // Condition permettant de vérifier et d'affecter le nouveau Débit max
    if(limitDed * (1 + impuls * 0.05) > dedMax) dedMax = limitDed * (1 + impuls *
0.05);

    // Si la dose a dépassé la limite de dose et qu'elle n'est pas déclenchée
    if(userDose > (limitDose * 1.20) && !alarmDose)
    {
        PlaySound(-1); // Déclenchement de l'alarme
        numAlarm++;
        // Formatage des valeurs pour typeAlarm
        if(strlen(typeAlarm) == 0) sprintf(typeAlarm, sizeof(typeAlarm), "%s",
"DOSE");
        else sprintf(typeAlarm, sizeof(typeAlarm), "%s+%s", typeAlarm, "DOSE");
    }

    // Si la dose a dépassé la limite de pré-dose et qu'elle n'est pas déclenchée
    else if((userDose > (limitDose * 1.05)) && !alarmPreDose)
    {
        alarmPreDose = 1; // Flag de l'alarme pre dose mis à 1
        PlaySound(20000); // Déclenchement de l'alarme pendant 20 secondes (20000 ms)
        numAlarm++;
        // Formatage des valeurs pour typeAlarm
        if(strlen(typeAlarm) == 0) sprintf(typeAlarm, sizeof(typeAlarm), "%s",
"PREDOSE");
        else sprintf(typeAlarm, sizeof(typeAlarm), "%s+%s", typeAlarm, "PREDOSE");
    }

    // Si le débit a dépassé la limite de débit
    else if(limitDed * (1 + impuls * 0.05) > (1.20 * limitDed))
    {
        // Si l'alarme n'est pas déclenché
        if(!IsAlarmRing())
        {
            // Si c'est la première fois que l'alarme de preded
            if(!alarmPreDed)
            {
                if(strlen(typeAlarm) == 0) sprintf(typeAlarm, sizeof(typeAlarm), "%s",
"DED");
                else sprintf(typeAlarm, sizeof(typeAlarm), "%s+%s", typeAlarm, "DED");
                alarmPreDed = 1;
                numAlarm++;
            }

            /*
            Génération d'un BIP pendant 200 ms tous les 500 ms pour un
            son discontinu
            */
            if(!timeDed) timeDed = time;

            if( (timeDed + 500) > time)
                PlaySound(200),
                timeDed=0;
        }
    }
}

```

```
        // Calcul de la dose
        userDose += limitDed * (1 + impuls * 0.05) * coeffExpo * (time/3600);
        // Affichage de la dose
        LcdGotoXY(0,2);
        char string[32] = {0};
        snprintf(string, sizeof(string), "DOSE= %.3fmSv", userDose);
        LcdString(string);
        Delay(500);
    }
}
```

Nokia.c

```
#include <p33Fxxxx.h>

#define BIT0  0b0000000000000001
#define BIT1  0b0000000000000010
#define BIT2  0b0000000000000100
#define BIT3  0b0000000000001000
#define BIT4  0b0000000000010000
#define BIT5  0b0000000001000000
#define BIT6  0b0000000010000000
#define BIT7  0b0000000100000000
#define BIT8  0b0000001000000000
#define BIT9  0b0000010000000000
#define BIT10 0b0000100000000000
#define BIT11 0b0001000000000000
#define BIT12 0b0010000000000000
#define BIT13 0b0100000000000000
#define BIT14 0b1000000000000000
#define BIT15 0b1000000000000000

/*-----
   LCD Nokia 3310
   -----*/

//Sans SPI
//8 RES   7 VOUT   6 GND   5 CS   4 DC   3 DIN   2 SCL   1 VDD
//Avec SPI

#define Lcd_DIN    LATBbits.LATB7    // patte 7
#define Lcd_CLK    LATBbits.LATB6    // patte 15
#define Lcd_DC      LATBbits.LATB8    // patte 10
```

```

#define Lcd_CE      LATBbits.LATB12  // patte 3
#define Lcd_RST     LATBbits.LATB9   // patte 2
#define LCD_PinsOut TRISB&=~(BIT6|BIT7|BIT8|BIT9|BIT12);
//TRISA&=~(BIT0|BIT1|BIT3|BIT4|BIT5);
#define LCDInitState LATB|=(BIT9|BIT12); // CE=RST="1"

// Mapping des signaux SPI1 (LCD Nokia)
// #define MapLCDSCLK _RP6R=8 // SCK2 mappée sur RP6=RB6=LCDCLK patte 15
// #define MapLCDSDIN _RP3R=7 // SDO2 mappée sur RP3=RB3=LCDSDIN patte 7
// #define SPIPutLCD SPI2Put // Coupleur SPI2r

/*-----
                                LCD Nokia 3310
-----*/

#define PosParam      32
// Résolution
#define Lcd_X_res     84
#define Lcd_Y_res     48
// Commandes controleur PCD8544
#define Function_Set   0x20 //Bit 2 = PD, Bit 1 = V, Bit 1 = H
#define Lcd_ON_Norm_H  0x20 //Lcd actif, adressage H, mode normal
#define Lcd_ON_Norm_V  0x22 //Lcd actif, adressage V, mode normal
#define Lcd_ON_Etendu_H 0x21 //Lcd actif, adressage H, mode étendu (H=1)
#define Lcd_OFF_Norm_H 0x00 //Lcd "power down", adressage H, mode normal
// Commandes normales (bit H="0")
#define Set_Config     0x08 //Bit 2 = D, bit 0 = E
#define Set_Adr_Y      0x40 //Adresse Y dans 3 bits LSB (0 à 5)
#define Set_Adr_X      0x80 //Adresse X dans 7 bits LSB (0 à 83)
// Commandes étendues (bit H="1")
#define Set_Bias       0x10 //Bias Mode dans bits 2 à 0
#define Set_Vop        0x80 //Vop dans bits 6 à 0
#define Set_TC         0x04 //Coef de température dans bits 1 à 0

/*-----
                                Variables spécifiques
-----*/

char Adr_X ; // Copie logicielle de l'adresse X du PCD8544 (0 à 83)
char Adr_Y ; // Copie logicielle de l'adresse Y du PCD8544 (0 à 5)

```

```

typedef enum
{
    Lcd_Cmd   = 0,
    Lcd_Data  = 1
} LcdCmdData; // Type de transfert : commande ou données

typedef enum
{
    Blanc = 0,
    Noir  = 1
} Couleur; // Type de couleur

/*-----
                                Constantes en mémoire flash
-----*/

/*-----
Générateur de caractère en mémoire flash
Code Ascii, format 5x7 pixels.
Format : Y      X      1      2      3      4      5
          1      D10 D20 D30 D40 D50
          2      D11 D21 D31 D41 D51
          3      D12 D22 D32 D42 D52
          4      D13 D23 D33 D43 D53
          5      D14 D24 D34 D44 D54
          6      D15 D25 D35 D45 D55
          7      D16 D26 D36 D46 D56
1° octet : D16-D15-D14-D13-D12-D11-D10, bit 7 toujours à "0"
2° octet : D26-D25-D24-D23-D22-D21-D20, bit 7 toujours à "0"
3° octet : D36-D35-D34-D33-D32-D31-D30, bit 7 toujours à "0"
4° octet : D46-D45-D44-D43-D42-D41-D40, bit 7 toujours à "0"
5° octet : D56-D55-D54-D53-D52-D51-D50, bit 7 toujours à "0"
-----*/

const char FontLookup[][5] =
{
    { 0x00, 0x00, 0x00, 0x00, 0x00 }, // sp
    { 0x00, 0x00, 0x2f, 0x00, 0x00 }, // !
    { 0x00, 0x07, 0x00, 0x07, 0x00 }, // "
    { 0x14, 0x7f, 0x14, 0x7f, 0x14 }, // #
    { 0x24, 0x2a, 0x7f, 0x2a, 0x12 }, // $

```

```

{ 0xc4, 0xc8, 0x10, 0x26, 0x46 }, // %
{ 0x36, 0x49, 0x55, 0x22, 0x50 }, // &
{ 0x00, 0x05, 0x03, 0x00, 0x00 }, // '
{ 0x00, 0x1c, 0x22, 0x41, 0x00 }, // (
{ 0x00, 0x41, 0x22, 0x1c, 0x00 }, // )
{ 0x14, 0x08, 0x3E, 0x08, 0x14 }, // *
{ 0x08, 0x08, 0x3E, 0x08, 0x08 }, // +
{ 0x00, 0x00, 0x50, 0x30, 0x00 }, // ,
{ 0x10, 0x10, 0x10, 0x10, 0x10 }, // -
{ 0x00, 0x60, 0x60, 0x00, 0x00 }, // .
{ 0x20, 0x10, 0x08, 0x04, 0x02 }, // /
{ 0x3E, 0x51, 0x49, 0x45, 0x3E }, // 0
{ 0x00, 0x42, 0x7F, 0x40, 0x00 }, // 1
{ 0x42, 0x61, 0x51, 0x49, 0x46 }, // 2
{ 0x21, 0x41, 0x45, 0x4B, 0x31 }, // 3
{ 0x18, 0x14, 0x12, 0x7F, 0x10 }, // 4
{ 0x27, 0x45, 0x45, 0x45, 0x39 }, // 5
{ 0x3C, 0x4A, 0x49, 0x49, 0x30 }, // 6
{ 0x01, 0x71, 0x09, 0x05, 0x03 }, // 7
{ 0x36, 0x49, 0x49, 0x49, 0x36 }, // 8
{ 0x06, 0x49, 0x49, 0x29, 0x1E }, // 9
{ 0x00, 0x36, 0x36, 0x00, 0x00 }, // :
{ 0x00, 0x56, 0x36, 0x00, 0x00 }, // ;
{ 0x08, 0x14, 0x22, 0x41, 0x00 }, // <
{ 0x14, 0x14, 0x14, 0x14, 0x14 }, // =
{ 0x00, 0x41, 0x22, 0x14, 0x08 }, // >
{ 0x02, 0x01, 0x51, 0x09, 0x06 }, // ?
{ 0x32, 0x49, 0x59, 0x51, 0x3E }, // @
{ 0x7E, 0x11, 0x11, 0x11, 0x7E }, // A
{ 0x7F, 0x49, 0x49, 0x49, 0x36 }, // B
{ 0x3E, 0x41, 0x41, 0x41, 0x22 }, // C
{ 0x7F, 0x41, 0x41, 0x22, 0x1C }, // D
{ 0x7F, 0x49, 0x49, 0x49, 0x41 }, // E
{ 0x7F, 0x09, 0x09, 0x09, 0x01 }, // F
{ 0x3E, 0x41, 0x49, 0x49, 0x7A }, // G
{ 0x7F, 0x08, 0x08, 0x08, 0x7F }, // H
{ 0x00, 0x41, 0x7F, 0x41, 0x00 }, // I
{ 0x20, 0x40, 0x41, 0x3F, 0x01 }, // J
{ 0x7F, 0x08, 0x14, 0x22, 0x41 }, // K
{ 0x7F, 0x40, 0x40, 0x40, 0x40 }, // L

```

```

{ 0x7F, 0x02, 0x0C, 0x02, 0x7F }, // M
{ 0x7F, 0x04, 0x08, 0x10, 0x7F }, // N
{ 0x3E, 0x41, 0x41, 0x41, 0x3E }, // O
{ 0x7F, 0x09, 0x09, 0x09, 0x06 }, // P
{ 0x3E, 0x41, 0x51, 0x21, 0x5E }, // Q
{ 0x7F, 0x09, 0x19, 0x29, 0x46 }, // R
{ 0x46, 0x49, 0x49, 0x49, 0x31 }, // S
{ 0x01, 0x01, 0x7F, 0x01, 0x01 }, // T
{ 0x3F, 0x40, 0x40, 0x40, 0x3F }, // U
{ 0x1F, 0x20, 0x40, 0x20, 0x1F }, // V
{ 0x3F, 0x40, 0x38, 0x40, 0x3F }, // W
{ 0x63, 0x14, 0x08, 0x14, 0x63 }, // X
{ 0x07, 0x08, 0x70, 0x08, 0x07 }, // Y
{ 0x61, 0x51, 0x49, 0x45, 0x43 }, // Z
{ 0x00, 0x7F, 0x41, 0x41, 0x00 }, // [
{ 0x55, 0x2A, 0x55, 0x2A, 0x55 }, // 55
{ 0x00, 0x41, 0x41, 0x7F, 0x00 }, // ]
{ 0x04, 0x02, 0x01, 0x02, 0x04 }, // ^
{ 0x40, 0x40, 0x40, 0x40, 0x40 }, // _
{ 0x00, 0x01, 0x02, 0x04, 0x00 }, // '
{ 0x20, 0x54, 0x54, 0x54, 0x78 }, // a
{ 0x7F, 0x48, 0x44, 0x44, 0x38 }, // b
{ 0x38, 0x44, 0x44, 0x44, 0x20 }, // c
{ 0x38, 0x44, 0x44, 0x48, 0x7F }, // d
{ 0x38, 0x54, 0x54, 0x54, 0x18 }, // e
{ 0x08, 0x7E, 0x09, 0x01, 0x02 }, // f
{ 0x0C, 0x52, 0x52, 0x52, 0x3E }, // g
{ 0x7F, 0x08, 0x04, 0x04, 0x78 }, // h
{ 0x00, 0x44, 0x7D, 0x40, 0x00 }, // i
{ 0x20, 0x40, 0x44, 0x3D, 0x00 }, // j
{ 0x7F, 0x10, 0x28, 0x44, 0x00 }, // k
{ 0x00, 0x41, 0x7F, 0x40, 0x00 }, // l
{ 0x7C, 0x04, 0x18, 0x04, 0x78 }, // m
{ 0x7C, 0x08, 0x04, 0x04, 0x78 }, // n
{ 0x38, 0x44, 0x44, 0x44, 0x38 }, // o
{ 0x7C, 0x14, 0x14, 0x14, 0x08 }, // p
{ 0x08, 0x14, 0x14, 0x18, 0x7C }, // q
{ 0x7C, 0x08, 0x04, 0x04, 0x08 }, // r
{ 0x48, 0x54, 0x54, 0x54, 0x20 }, // s
{ 0x04, 0x3F, 0x44, 0x40, 0x20 }, // t

```

```

    { 0x3C, 0x40, 0x40, 0x20, 0x7C }, // u
    { 0x1C, 0x20, 0x40, 0x20, 0x1C }, // v
    { 0x3C, 0x40, 0x30, 0x40, 0x3C }, // w
    { 0x44, 0x28, 0x10, 0x28, 0x44 }, // x
    { 0x0C, 0x50, 0x50, 0x50, 0x3C }, // y
    { 0x44, 0x64, 0x54, 0x4C, 0x44 } // z
};

/*-----
Générateur de caractères
Nombre de caractères : 10
Résolution X : 10
Résolution Y : 14 */
const char MyFontTab[][10] =
{
    { 252, 254, 3, 3, 3, 3, 3, 3, 254, 252 }, // 0 octet LSB
    { 0, 16, 24, 12, 6, 255, 255, 0, 0, 0 }, // 1 octet LSB
    { 2, 3, 131, 131, 131, 131, 131, 195, 254, 124 }, // 2 octet LSB
    { 2, 3, 3, 195, 195, 195, 195, 199, 254, 60 }, // 3 octet LSB
    { 192, 224, 48, 24, 12, 198, 195, 1, 0, 0 }, // 4 octet LSB
    { 31, 63, 99, 99, 99, 99, 99, 99, 195, 131 }, // 5 octet LSB
    { 252, 254, 195, 195, 195, 195, 195, 195, 131, 0 }, // 6 octet LSB
    { 2, 3, 3, 3, 131, 195, 99, 51, 31, 15 }, // 7 octet LSB
    { 124, 254, 195, 195, 195, 195, 195, 195, 254, 60 }, // 8 octet LSB
    { 124, 254, 195, 195, 195, 195, 195, 195, 254, 252 }, // 9 octet LSB
    { 15, 31, 48, 48, 48, 48, 48, 48, 31, 15 }, // 0 octet MSB
    { 0, 0, 0, 48, 48, 63, 63, 48, 48, 0 }, // 1 octet MSB
    { 62, 63, 51, 49, 49, 49, 49, 49, 48, 48 }, // 2 octet MSB
    { 16, 48, 48, 48, 48, 48, 48, 56, 31, 15 }, // 3 octet MSB
    { 3, 3, 3, 3, 3, 63, 63, 3, 3, 3 }, // 4 octet MSB
    { 16, 48, 48, 48, 48, 48, 48, 48, 31, 15 }, // 5 octet MSB
    { 15, 31, 48, 48, 48, 48, 48, 48, 31, 15 }, // 6 octet MSB
    { 56, 60, 6, 3, 1, 0, 0, 0, 0, 0 }, // 7 octet MSB
    { 15, 31, 48, 48, 48, 48, 48, 48, 31, 15 }, // 8 octet MSB
    { 0, 48, 48, 48, 48, 48, 48, 48, 31, 15 } // 9 octet MSB
};

/*-----
Déclarations des fonctions en mémoire flash
-----*/

```



```

/*-----
Nom          :  LcdSend
Description  :  Envoi d'un octet vers le controleur du LCD
Arguments    :  data -> Donnée ou commande à transmettre
                  cd  -> Type de transfert : commande (Lcd_Cmd)
                  ou donnée (Lcd_Data)
Valeur renvoyée : aucune.
-----*/

void LcdSend(char Data, LcdCmdData CD)
{
    int i;
    // Sélection du controleur (actif à "0").
    Lcd_CE=0; // CE = "0"
    // positionnement de la patte Data/Commande
    if (CD == Lcd_Data) Lcd_DC=1;
        else Lcd_DC=0;
    // Transmission série de l'octet sur DIN rythmé par CLK
    for (i=0; i<8; i++)
    {
        if ((Data & 0x80)==0) Lcd_DIN=0;
            else Lcd_DIN=1;

        Data <<= 1;
        Lcd_CLK=1; // pour assurer durée mini 125ns de état haut avec FCY = 40MHz
        Lcd_CLK=1;
        Lcd_CLK=1;
        Lcd_CLK=1;
        Lcd_CLK=0;
    }
    // Dé-sélection du controleur
    Lcd_CE=1;
}

/*-----
Nom          :  Fill_Lcd
Description  :  Remplir le LCD de la même couleur
Arguments    :  Coul_Lcd : Noir ou Blanc
Valeur renvoyée : aucune.
-----*/

void Fill_Lcd(Couleur Coul_Lcd)
{

```

```

int x,y;
char b;
if (Coul_Lcd==Noir) b=255;
    else b=0;
for (y=0; y<Lcd_Y_res/8+1; y++)
{
    LcdSend(Set_Adr_X+0,Lcd_Cmd); //Adresse X = 0
    LcdSend(Set_Adr_Y+y,Lcd_Cmd); //Adresse Y
    for (x=0; x<Lcd_X_res; x++) LcdSend(b,Lcd_Data);
}
Adr_X=0; Adr_Y=0;
LcdSend(Set_Adr_X+0,Lcd_Cmd); //Adresse X = 0
LcdSend(Set_Adr_Y+0,Lcd_Cmd); //Adresse Y = 0
}
/*-----
Nom          :  LcdChar
Description   :  Dessin d'un caractère à la position actuelle des compteurs X et Y
Arguments     :  Code Ascii du caractère
Valeur renvoyée : aucune.
-----*/
void LcdChar(char c)
{
    int i;
    for (i=0; i<5; i++) LcdSend(FontLookup[c-32][i],Lcd_Data);
    Adr_X += 6;
}
/*-----
Nom          :  LcdString
Description   :  Dessin d'une phrase à la position actuelle des compteurs X et Y
Arguments     :  Pointeur de la chaîne de caractères
Valeur renvoyée : aucune.
-----*/
void LcdString(char *Phrase)
{
    while (*Phrase)
    {
        LcdChar(*Phrase++);
        LcdSend(0,Lcd_Data); // Espace inter-caractères
        Adr_X++;
    }
}

```

```

}
/*-----
Nom      :  LcdBigDigit
Description :  Dessin d'un chiffre 14x10 pixels à la position actuelle des
               compteurs X et Y
Arguments  :  Chiffre codé BCD
Valeur renvoyée : aucune.
-----*/
void LcdBigDigit(char b)
{
    int i;
    for (i=0; i<10; i++) LcdSend(MyFontTab[b][i],Lcd_Data); // D'abord les octets LSB
    LcdSend(Set_Adr_X+Adr_X,Lcd_Cmd); // Retour au début du dessin
    LcdSend(Set_Adr_Y+Adr_Y+1,Lcd_Cmd); // Octets MSB à la ligne de 8 pixels inférieure
    for (i=0; i<10; i++) LcdSend(MyFontTab[b+10][i],Lcd_Data); // Puis les octets MSB
    Adr_X += 12;
    LcdSend(Set_Adr_Y+Adr_Y,Lcd_Cmd); //Prochain chiffre sur la même ligne
    LcdSend(Set_Adr_X+Adr_X,Lcd_Cmd); //et 12 pixels à droite
}
/*-----
Nom      :  LcdBigString
Description :  Dessin d'une suite de chiffres à la position actuelle des compteurs X et
               Y
Arguments  :  Pointeur de la chaîne de caractères
Valeur renvoyée : aucune.
-----*/
void LcdBigString(char *Chaine)
{
    while (*Chaine) LcdBigDigit(*Chaine++ - '0');
}
/*-----
Nom      :  LcdGotoXY
Description :  Affecter les adresses X et Y du PCD8544
Arguments  :  X : position H (0 à 83)
               Y : position V (0 à 5)
Valeur renvoyée : aucune.
-----*/
void LcdGotoXY(char x, char y)
{
    Adr_X=x; Adr_Y=y;

```

```

    LcdSend(Set_Adr_X+Adr_X,Lcd_Cmd); // Affectation adresse X
    LcdSend(Set_Adr_Y+Adr_Y,Lcd_Cmd); // Affectation adresse Y
}

/*-----
    Nom          :   BarGraphLcd
    Description   :   Dessiner une barre noire horizontale sur le LCD
    Arguments    :   Y : position verticale (Y:0 à 5)
                    LngLine : longueur de la barre en pixels
    Valeur renvoyée : aucune.
-----*/

void BarGraphLcd(char Y, int LngLine)
{
    int i;
    LcdGotoXY(0,Y);
    for (i=0; i<Lcd_X_res; i++)
    {
        if (i<2) LcdSend(0,Lcd_Data);
        else if (i<LngLine) LcdSend(0x3C,Lcd_Data);
        else LcdSend(0,Lcd_Data);
    }
}

/*-----
    Nom          :   LcdClrLine
    Description   :   Effacer la ligne complète donnée en argument
    Arguments    :   Y : numéro de la ligne
    Valeur renvoyée : aucune.
-----*/

void LcdClrLine(char Y)
{
    int i;
    LcdGotoXY(0,Y);
    for (i=0; i<Lcd_X_res; i++)
    {
        LcdSend(0,Lcd_Data); // Effacer une ligne verticale de 8 pixels
    }
}

/*-----
    Nom          :   write
    Description   :   Sorties de caractères des fonctions "printf"
    Arguments    :

```

```

    Valeur renvoyée :
    -----*/
int write(int handle, void *buffer, unsigned int len)
{
    int i;
    if (Adr_X >= Lcd_X_res) return(len);
    switch (handle)
    {
        case 0:
        case 1:
        case 2:
            for (i = len; i; --i)
            {
                LcdChar(*(char*)buffer);
                LcdSend(0,Lcd_Data); // Espace inter-caractères
                Adr_X++;
            }
            break;
        default: break;
    }
    return(len);
}

/*-----
** Delay utilisant TMR1 en t en ms à vérifier
-----*/

void Delay( unsigned int t)
{
    while (t--)
    {
        TMR1 = 0;
        while (TMR1<16000);
    }
}
*/
/*-----
Nom          : LcdInit
Description   : Initialisation du PCD8544
Arguments    : aucun
Valeur renvoyée : aucune.

```

```

-----*/
void LcdInit(void)
{
    LCD_PinsOut; // instruction d'initialisation des ports définie dans les #define
    LCDInitState; // instruction d'initialisation des ports définie dans les #define
    Lcd_CE=1; // Controleur non sélectionné (CE = "1")
    Lcd_RST=1; // et pas de reset (RST = "1")
    Delay(500); // Délai de 0,5s
    Lcd_RST=0; // Reset PCD8544 (RST = "0")
    Delay(500); // Délai de 0,5s
    Lcd_RST=1; // Arrêt reset PCD8544 (RST = "1")
    Adr_X=0; Adr_Y=0;
    LcdSend(Lcd_ON_Etendu_H, Lcd_Cmd); //Mode commandes étendues 0x21
    LcdSend(Set_Vop+72,Lcd_Cmd); //Set LCD Vop (contrast) 0x80
    LcdSend(Set_TC+2,Lcd_Cmd); //Set temp. coefficient 0x04
    LcdSend(Set_Bias+3,Lcd_Cmd); //LCD Bias mode 1:48 0x10
    LcdSend(Lcd_ON_Norm_H,Lcd_Cmd); //LCD standard commands. Horizontal addressing mode. 0x20
    LcdSend(Set_Config+4,Lcd_Cmd); //LCD in normal mode (DE = 10) 0x08
}

/*****
Nom          :   Conv_Bin8_String
Description   :   Conversion d'une valeur codée 8 bits en binaire
                  -> chaîne de caractères
Arguments    :   x : nombre binaire à convertir
                  *String : pointeur vers la chaîne de destination
Valeur renvoyée : aucune
*****/
void Conv_Bin8_String(char *String,unsigned char x)
{
    String[0]=x/100; // Centaines
    String[1]=(x-(100*String[0]))/10; // Dizaines
    String[2]=x-(100*String[0])-10*String[1]+'0'; // Unités en Ascii
    String[0]+='0'; // Conversion BCD -> Ascii des centaines
    String[1]+='0'; // Conversion BCD -> Ascii des dizaines
    String[3]=0; // Termineur de chaîne
}

```

```

/*
SPI1CON1 = 0x0423; // Master, 16 bit, disable SCK, disable SS, prescaler 1:8
SPI1STAT = 0x8000; // Enable SPI port
_SPI1IP=6;          // Priorité élevée
_SPI1IF=0;
_SPI1IE=1;

/*-----
Function   : void _ISR _SPI1Interrupt(void)
Description:
    - Production du signal vidéo de contrôle
    - Déclenchée à la fin de la transmission du mot précédent
Traitements :
    - Décrémenter "CptWord" et détecter son état pour transmettre le bon nombre
      de mots
    - Transmission le cas échéant du mot suivant depuis "Image"
-----*/
/*void _ISRFAST _SPI1Interrupt(void)
{
    int dummy;
    dummy = SPI1BUF; // Lecture du dernier mot reçu pour acquitter l'écriture précédente
    CptWord--;
    if (CptWord)
    {
        if (CptWord==1) SPI1BUF = (*ImagePtrOut++)&0xFFFE; // Forcer le dernier bit à 0
        else SPI1BUF = *ImagePtrOut++;
    }
    else
    {
        _SPI1IE=0; // Dernier mot transmis : inhiber les interruptions
    }
    _SPI1IF = 0;
}
*/

```

convert.c

```

typedef union
{
    char charval[4];

```



```

    unsigned long result;
}   BCDval;

char LCDtext[6];

/* Function prototypes */

unsigned long hex2bin (char *);
unsigned long bin2BCD_conv (unsigned long );
char BCD_adj(char);
char upper_char(char);
char lower_char(char);

/*****
/* Convert data and display 8-digit result on LCD
   On entry, global array RFID contains the hexadecimal data to be converted.
   Leading zeros are not suppressed.
*****/
void disp_data (unsigned long HEXin)
{
    HEXin=HEXin & 0xFFFF;
    BCDval temp;
    char RFID[6];
    RFID[0]=0; // n'est pas effacé par l'initialisation
    char i, j;
    for (i = 5 , j = 0 ; i > 0 ; i--,j++)
    {
        RFID[i]=HEXin >>(4*j) & 0x0F; //transformation du nombre Hexa en tableau
    }
        //de caractères précédé par 0 et justifié à droite

    temp.result = bin2BCD_conv (hex2bin(RFID));
    for (i = 0, j = 0; i < 3; i++)
    {
        LCDtext[j+1] = upper_char (temp.charval[i]);
        LCDtext[j] = lower_char (temp.charval[i]);
        j += 2;
    }
}

```

```

    LCDtext[6] = 0; /* Null terminate string data */
}

/*****
/* Hexadecimal -> binary conversion
The array data referenced by the pointer hexseq must contain only one
nybble in each char of the array, right justified.
The binary value returned is a 24-bit value, right justified.
*/
unsigned long hex2bin (char *hexseq)
{
    char i;
    unsigned long temp = 0;

    for (i = 0; i < 6; i++) {
        temp = (temp << 4) | *hexseq;
        hexseq++;
    }
    return (temp);
}

/*****
/* Binary -> BCD conversion
A 24-bit binary value is converted to eight BCD digits, with two digits
packed per return char.
*/
unsigned long bin2BCD_conv (unsigned long val)
{
    BCDval res;
    char i;
    res.result = 0;
    for (i = 0; i < 24; i++) {
        res.charval[0] = BCD_adj (res.charval[0]);
        res.charval[1] = BCD_adj (res.charval[1]);
        res.charval[2] = BCD_adj (res.charval[2]);
        res.charval[3] = BCD_adj (res.charval[3]);
        res.result <= 1;
        if (val & 0x800000)

```

```

        res.result += 1;
        val <= 1;
    }
    return (res.result);
}

/*****
*/
char BCD_adj (char val)
{
    if ((val & 0x0F) >= 5)
        val += 3;
    if ((val & 0xF0) >= 0x50)
        val += 0x30;
    return (val);
}

/*****
/* Unpack BCD digit and convert to ASCII value
*/
char upper_char (char value)
{
    return (((value >> 4) & 0x0F) + 0x30);
}
char lower_char (char value)
{
    return ((value & 0x0F) + 0x30);
}

/*****
/* Write ASCII string to LCD

void printf_LCD_4bits(char fila, char columna, char *texto)
{
    char adrs;
    adrs = columna - 1;
    if (fila == 2)
        adrs = adrs | 0x40;
    Ctrl14bits(adrs | 0x80);

```

```

    while (*texto)
        Datos4bits(texto++);
}
*/

```

uart.c

```

#include <p33Fxxxx.h>
#include "defines.h"

extern int countU1;
extern int countU2;
extern char espResponse[SIZE_ESPBUFF];
extern char seqResponse[SIZE_SEQBUFF];

/*-----
                                Câblage
-----*/

#define U1_Tx RPOR2=0x0003;    // U1Tx du pic câblé sur RP4
#define U1_Rx RPINR18=0x1F05; // U1Rx du pic câblé sur RP5

#define U2_Tx RPOR1=0x0005;    // U2Tx du pic câblé sur RP2
#define U2_Rx RPINR19=0x1F03; // U2Rx du pic câblé sur RP3

/*-----
Nom          : void InitUART1 (long Bauds)
Description  : Programme d'interruption déclenché à la réception
                d'un caractère
Arguments    : Bauds
Valeur renvoyée : aucune
-----*/

void InitUART1(long Bauds)
{
    U1_Tx; // câblage Tx
    U1_Rx; // câblage Rx
}

```

```

    U1MODE=0x8000; // Module UART validé, 8 bits, pas de parité, 1 bit stop
                  // Pas d'inversion des signaux RX1 et TX1
                  // RTS et CTS non utilisés
    U1STA =0x0400; // UART TX validé
                  // Interruption RX à chaque caractère
    U1BRG =FCY/((long)16*Bauds)-1; // Vitesse de transmission

    _U1RXIP=7;    // Priorité maximum car buffer hard de faible capacité
    _U1RXIF=0;    // Raz indicateur interruption RX
    _U1RXIE=1;    // Validation interruption RX

    ClearUartEspBuffer();
}

/*-----
Nom          : void InitUART1 (long Bauds)
Description : Programme d'interruption déclenché à la réception
              d'un caractère
Arguments    : Bauds
Valeur renvoyée : aucune
-----*/

void InitUART2(long Bauds)
{
    U2_Tx; // câblage Tx
    U2_Rx; // câblage Rx

    U2MODE=0x8000; // Module UART validé, 8 bits, pas de parité, 1 bit stop
                  // Pas d'inversion des signaux RX1 et TX1
                  // RTS et CTS non utilisés
    U2STA =0x0400; // UART TX validé
                  // Interruption RX à chaque caractère
    U2BRG =FCY/((long)16*Bauds)-1; // Vitesse de transmission

    _U2RXIP=7;    // Priorité maximum car buffer hard de faible capacité
    _U2RXIF=0;    // Raz indicateur interruption RX
    _U2RXIE=1;    // Validation interruption RX

```

```

    ClearUartSeqBuffer();
}

/*-----
Nom          : void _ISR _U1RXInterrupt (void)
Description : Programme d'interruption déclenché à la réception
              d'un caractère
Arguments    : aucun
Valeur renvoyée : aucune
-----*/

void _ISR _U1RXInterrupt(void)
{
    _U1RXIF=0;           // Clear interrupt flag
    while (U1STAbits.URXDA) // Boucle pour lire ts les car. ds le buf. de l'UART
    {
        if(countU1 + 1 < sizeof(espResponse))
            espResponse[countU1++]=U1RXREG;
    }
}

/*-----
Nom          : void _ISR _U1RXInterrupt (void)
Description : Programme d'interruption déclenché à la réception
              d'un caractère
Arguments    : aucun
Valeur renvoyée : aucune
-----*/

void _ISR _U2RXInterrupt(void)
{
    _U2RXIF=0;           // Clear interrupt flag
    while (U2STAbits.URXDA) // Boucle pour lire ts les car. ds le buf. de l'UART
    {
        seqResponse[countU2++] = U2RXREG;
    }
}

```

```

    }
}

void SendData(char* message)
{
    int i=0, var=1;

    for(i=0;message[i]!='\0';i++)
    {
        while (U1STAbits.TRMT==0){}
        U1TXREG=message[i];
        int j=0;
        for(j=0;j<1000;j++){var++;}
    }
}

void SendData2(char* message)
{
    int i=0, var=1;
    for(i=0;message[i]!='\0';i++)
    {
        while (U2STAbits.TRMT==0){}
        U2TXREG=message[i];
        int j=0;
        for(j=0;j<1000;j++){var++;}
    }
}

/*-----
Nom          : void ClearEspBuffer(void)
Description : Programme qui vide le Buffer de réception RXD
Arguments    : durée en ms
Valeur renvoyée : aucune
-----*/

void ClearUartEspBuffer(void)
{

```



```
    int i;
    for (i=0;i<sizeof(espResponse);i++) espResponse[i]=0;
    countU1=0;
}

void ClearUartSeqBuffer(void)
{
    int i;
    for (i=0;i<sizeof(seqResponse);i++) seqResponse[i]=0;
    countU2=0;
}

int IsUart2Connected()
{
    return !_RA0;
}
```

wifi.h

```
/*
 * File:    wifi.h
 * Author:  ariehl
 *
 * Created on 16 mars 2016, 16:41
 */

#ifndef WIFI_H
#define WIFI_H

typedef struct{

    int mode; // Mode client ou serveur
    char* IP; // IP du serveur ou du client
    char* SSID; // SSID du serveur (si configuré en serveur)
    char* ssidPass; // Mot de passe du SSID serveur (si configuré en serveur)
    char* port; // Port du serveur (si configuré en serveur)
    int connected;

}Wifi;
```

```
#define MODE_CLIENT 1
#define MODE_SERVER 2

#endif      /* WIFI_H */
```

wifi.c

```
#include <p33Fxxxx.h>
#include "wifi.h"
#include "defines.h"
#include <string.h>

Wifi configurationWifi;
extern char espResponse[SIZE_ESPBUFF];

void SetWifiMode(int mode)
{
    if(mode == MODE_CLIENT)
    {
        SendData("AT+CWMODE=1\r\n");
    }
    else if(mode == MODE_SERVER)
    {
        SendData("AT+CWMODE=2\r\n");
    }
}

void SetWifiIP(char* IP)
{
    char atIP[128];
    snprintf(atIP, sizeof(atIP), "AT+CIPAP=\"%s\"\r\n", IP);
    SendData(atIP);
}

void SetMultiConnection(int cipmux)
{
    if(!cipmux) SendData("AT+CIPMUX=0\r\n");
    else SendData("AT+CIPMUX=1\r\n");
}

void InitSSID(char* SSID, char* pass, int canal)
{
    char atSSID[128];
    snprintf(atSSID, sizeof(atSSID), "AT+CWSAP=\"%s\", \"%s\", %d, 0\r\n", SSID,
pass, canal);
    SendData(atSSID);
}

void StartServer()
```

```

{
    SendData("AT+CIPSERVER=1,1500\r\n");
}

void StopServer()
{
    SendData("AT+CIPSERVER=0,1500\r\n");
}

void InitWifi(int dosiNum)
{
    /*
        Paramètre du module
    */

    char IP[32] = {0};
    sprintf(IP, sizeof(IP), "192.168.1.%d", dosiNum);

    configurationWifi.IP = IP;
    configurationWifi.mode = 2;
    configurationWifi.connected = 0;

    Fill_Lcd(0);
    LcdGotoXY(0,0);
    LcdString("Initialisation");
    LcdGotoXY(0,2);
    LcdString(IP);
    LcdGotoXY(0,3);
    if(configurationWifi.mode == 1)
        LcdString("Mode : client");
    else if(configurationWifi.mode == 2)
        LcdString("Mode : serveur");

    SendData("AT+RST\r\n");
    Delay(5000); // Temporisation pour le reset

    ClearUartEspBuffer();

    /*
        Configuration du module en mode client
    */

    SetWifiMode(configurationWifi.mode);
    Delay(100); // Attente de la réponse

    ClearUartEspBuffer();
    /*
        Configuration de l'ip du module
    */

    SetWifiIP(configurationWifi.IP);
    Delay(250); // Attente d'une réponse
}

```

```
ClearUartEspBuffer();

/*
  Modification du nombre de connexion en simultanés (0 pour un client)
*/
SetMultiConnection(1);
Delay(100);
ClearUartEspBuffer();

char SSID[8] = {0};
snprintf(SSID, sizeof(SSID), "DOSI%d!", dosiNum);
InitSSID(SSID, "", 1);
Delay(4000);

ClearUartEspBuffer();
}
```

defines.h

```
/*
 * File:    defines.h
 * Author:  ariehl
 *
 * Created on 16 mars 2016, 16:55
 */

#ifndef DEFINES_H
#define    DEFINES_H

#define SIZE_ESPBUFF            800
#define SIZE_SEQBUFF           350
#define MAX_DOSIMETRES         50

#define FCY        36850000 // Fosc= ... MHz et Fcy= ... MHz

#endif    /* DEFINES_H */
```

7.2. Télécommande

configure.c

```
#include <p33Fxxxx.h> // A modifier suivant le uC utilisé
```

```
#include "defines.h"

extern long sendCount;
extern int Tdelais;

/*-----
Function    : void InitCPU_Clock(void)
Description: Initialisation de l'horloge CPU
    - Les bits de configuration (déjà programmés) réalisent les sélections
      suivantes :
        - horloge de base = FRC = 7.37MHz à 2%
        - PLL activée (sélection S1)
    - Les registres du module horloge sont affectés par cette fonction pour
      obtenir :
        - FRCDIVN = 7,37MHz
        - Fosc = 73,7 MHz (7,37*10)
        - FCY = 36,85MHz à 2% près
-----*/
void InitCPU_Clock(void)
{
    CLKDIVbits.FRCDIV=0;    // FRC non divisée : 7,37MHz
    CLKDIVbits.PLLPRE=0;    // Prédvision par 2 avant PLL : 3,685MHz
    PLLFBDbits.PLLDIV=40-2; // PLL : multiplication par 40 : 147,4MHz
    CLKDIVbits.PLLPOST=0;   // Division par 2 après PLL : Fosc = 73,7MHz
}

/*-----
Nom          : void InitTimer1 (void)
Description  : Programme d'initialisation du timer 1 en base de temps 1ms
Arguments    : aucun
Valeur renvoyée : aucune
*
*
* Temps d'actualisation : FCY / 256 / 144 = 1 ms
-----*/
void InitTimer1 (void)
{
```

```

PR1=144; //base de temps
T1CON=0x8030; // TCKPS=10 FcY div 256 ,Timer on, TGATE=0 TCS=0
T1CONbits.TON=1 ; //timer on
IEC0bits.T1IE=1 ; // validation des interruptions du Timer 1
TMR1=0;
}

/*-----
Nom          : void InitTimer1 (void)
Description :  Programme d'interruption du timer 1
Arguments    : aucun
Valeur renvoyée : aucune
-----*/
void _ISR _T1Interrupt(void) //
{
    TMR1=0; // RAZ de TMR1
    IFS0bits.T1IF=0; // Acquitement interruption
    Tdelais++;
    sendCount--;
}

/*-----
Nom          : void Delay(int delais)
Description :  Programme de temporisation
Arguments    : durée en ms
Valeur renvoyée : aucune
-----*/
void Delay(int delais)
{
    Tdelais=0;
    while (Tdelais < delais){}
}

```

main.c

```

/*-----
                Librairies et fichiers inclus
-----*/
#include <p33Fxxxx.h>
#include <stdlib.h>
#include <string.h>
#include "wifi.h"
#include "defines.h"

/*~~~~~ Bits de configuration ~~~~~*/
_FBS(RBS_LARGE_RAM & BSS_NO_FLASH & BWRP_WRPROTECT_OFF);
    // Large Sized Boot Ram
    // No Boot Segment Program Memory
    // Write Protect Disabled
_FSS(RSS_NO_RAM & SSS_NO_FLASH & SWRP_WRPROTECT_OFF);
    // Secure Segment Data Ram: No Secure Ram
    // Secure Segment Program Memory: No Secure Segment
    // Write Protect Disabled
_FGS(GSS_OFF & GCP_OFF & GWRP_OFF);
    // Code Protect off
    // Write Protect Disabled
_FOSCSEL(FNOSC_FRCPLL & IESO_OFF);
    // Fast RC oscillator w/ divide and PLL
    // Two-speed Oscillator Startup disabled
_FOSC(FCKSM_CSECMD & IOL1WAY_OFF & OSCIOFNC_OFF & POSCMD_NONE);
    // Clock Switching is enabled and Fail Safe Clock Monitor is disabled
    // Single configuration for remappable I/O disabled
    // OSC2 Pin Function: OSC2 is RA3
    // Horloge externe inhibée
_FWDT(FWDTEN_OFF & WINDIS_OFF);
    // Watchdog Timer disabled by user software
    // Windowed WDT disabled

```

```

_FPOR(ALT12C_OFF & FPWRT_PWR128);
    // I2C mapped to SDA1/SCL1
    // Power-on Reset Value: 128mS
_FICD(JTAGEN_OFF & ICS_PGD1);
    // JTAG is disabled
    // ICD communicate on PGEC1/EMUEC1
/*~~~~~*/

/*-----
                Déclarations des équivalences
-----*/
#define FCY          36850000 // Fosc= ... MHz et Fcy= ... MHz
#define SEND_TIME    1000

/*-----
                Déclarations des variables globales
-----*/

char espResponse[SIZE_ESPBUFF]={0};      // Tableau contenant la réponse aux
commandes AT (RESET = 270 car.)
int countU1=0;                          // Compteur permettant d'accéder au tableau
espResponse

char seqResponse[SIZE_SEQBUFF]={0};      // Tableau contenant la réponse Le
séquence envoyé par le raspberry
int countU2=0;                          // Compteur permettant d'accéder au tableau
seqReponse

int Tdelais=0;                          // Compteur pour le timer

/*
    Structure contenant les variables pour configurer le wifi.
*/
Wifi configurationWifi; // Structure pour configurer le wifi

/*
    * Variable contenant l'activité chargé

```



```
*/  
char sequence[SIZE_SEQBUFF] = {0};  
  
/*  
 * Tableau contenant la liste des dosimetres  
 */  
int dosimetres[MAX_DOSIMETRES] = {0};  
  
/*-----  
   Vider le tableau contenant la séquence chargé dans la télécommande  
-----*/  
  
void ClearSeqBuffer(void)  
{  
    int i;  
    for (i=0;i<sizeof(sequence);i++) sequence[i]=0;  
}  
  
/*-----  
   Retourne 1 si le caractère est un chiffre sinon 0  
-----*/  
  
int IsCharDigit(char c)  
{  
    switch(c)  
    {  
        case '1': return 1;  
        case '2': return 1;  
        case '3': return 1;  
        case '4': return 1;  
        case '5': return 1;  
        case '6': return 1;  
        case '7': return 1;  
        case '8': return 1;  
        case '9': return 1;
```

```
        default : return 0;
    }
    return 0;
}

/*-----
           Fonction principale
-----*/

int main(void)
{
    InitCPU_Clock();
    InitTimer1();
    AD1PCFGL=0x01FF; // Pas d'entrée analogique

    InitClav();
    InitUART1(57600);
    InitUART2(57600);
    LcdInit();
    InitWifi();

    ShowMenu();

    while(1)
    {}
}
```

clavier.c

```
/*-----
           Librairies et fichiers inclus
-----*/
#include <p33Fxxxx.h>
#include "defines.h"

#define C1 _RC8
#define C2 _RB10
#define C3 _RB11

#define L1 _LATB12
```

```

#define L2 _LATB13
#define L3 _LATB14
#define L4 _LATB15

void InitClav()
{
    /*
     Configuration des bits B11,10 et C8 en entrée
    */
    _TRISB10 = 1, _TRISB11 = 1;
    _TRISC8=1;

    _CN20PUE=1;_CN16PUE=1; _CN15PUE=1; // Mise à 1 des résistances pullup du
    clavier
}

char Lecture_Clav(void)
{
    _TRISB12 = 0;
    Delay(20);

    if(C1==0) { _TRISB12 = 1; return('1'); }
    if(C2==0) { _TRISB12 = 1; return('2'); }
    if(C3==0) { _TRISB12 = 1; return('3'); }
    _TRISB12 = 1;

    _TRISB13 = 0;
    Delay(20); // antirebonds

    if(C1==0) { _TRISB13 = 1; return('4'); }
    if(C2==0) { _TRISB13 = 1; return('5'); }
    if(C3==0) { _TRISB13 = 1; return('6'); }
    _TRISB13 = 1;

    _TRISB14 = 0;
    Delay(20); // antirebonds

    if(C1==0) { _TRISB14 = 1; return('7'); }
    if(C2==0) { _TRISB14 = 1; return('8'); }
    if(C3==0) { _TRISB14 = 1; return('9'); }

    _TRISB14 = 1;

    _TRISB15 = 0;
    Delay(20); // antirebonds

    if(C1==0) { _TRISB15 = 1; return('*'); }
    if(C2==0) { _TRISB15 = 1; return('0'); }
    if(C3==0) { _TRISB15 = 1; return('#'); }

    _TRISB15 = 1;
    return 'n';
}

```

defines.h

```

/*
 * File:    defines.h
 * Author:  ariehl
 *
 * Created on 16 mars 2016, 16:55
 */

#ifndef DEFINES_H
#define    DEFINES_H

#define DEBUG_MODE 1 // Mode debug desactivé = 0 | activé = 1

#define BIT0  0b00000000000000000001
#define BIT1  0b00000000000000000010
#define BIT2  0b00000000000000000100
#define BIT3  0b00000000000000001000
#define BIT4  0b000000000000010000
#define BIT5  0b000000000001000000
#define BIT6  0b000000000100000000
#define BIT7  0b000000001000000000
#define BIT8  0b000000010000000000
#define BIT9  0b000000100000000000
#define BIT10 0b000001000000000000
#define BIT11 0b000010000000000000
#define BIT12 0b000100000000000000
#define BIT13 0b001000000000000000
#define BIT14 0b010000000000000000
#define BIT15 0b100000000000000000

/*-----
LCD Nokia 3310
-----*/

#define Lcd_DIN    LATAbits.LATA9    // patte 35
#define Lcd_CLK    LATAbits.LATA4    // patte 34
#define Lcd_DC      LATCbits.LATC3    // patte 36
#define Lcd_CE      LATCbits.LATC5    // patte 38
#define Lcd_RST     LATCbits.LATC4    // patte 37
#define LCD_PinsOut TRISC&=~(BIT3|BIT4|BIT5); TRISA&=~(BIT4|BIT9);
#define LCDInitState LATC|=(BIT4|BIT5); // CE=RST="1"

#define SIZE_ESPBUFF      800
#define SIZE_SEQBUFF      350
#define MAX_DOSIMETRES    50

#define FCY      36850000 // Fosc= ... MHz et Fcy= ... MHz

```

```
#endif      /* DEFINES_H */
```

menu.h

```
/*
 * File:    menu.h
 * Author:  ariehl
 *
 * Created on 19 avril 2016, 14:02
 */

#ifndef MENU_H
#define MENU_H

void ShowMenu();
void ShowSeqMenu();
void ShowDosiMenu();

#endif      /* MENU_H */
```

menu.c

```
#include <p33Fxxxx.h>
#include "menu.h"
#include "wifi.h"
#include "defines.h"

extern char seqResponse[SIZE_SEQBUFF];
extern char espResponse[SIZE_ESPBUFF];
extern char sequence[SIZE_SEQBUFF];
extern int dosimetres[MAX_DOSIMETRES];

/*
    Tableau contenant le numéro du dosimètre entré pour le sélectionné
 */
```

```
char numberInput[2] = {' '};
int inputCount = 0;

void ClearDosimetreList()
{
    int i = 0;
    for(i; i < MAX_DOSIMETRES;i++)
        dosimetres[i] = 0;
}

void ShowMenu()
{
    Fill_Lcd(0);
    LcdGotoXY(0,0);
    LcdString("1: Dosimetres");

    while(1)
    {
        switch(Lecture_Clav())
        {
            case '1':
            {
                ShowDosiListMenu();
                break;
            }
        }
    }
}

void ShowDosiListMenu()
{
    Delay(500); // Temporisation pour éviter le rebond
    Fill_Lcd(0);
    LcdGotoXY(0,0);
    LcdString("Recuperation des dosimetres...");
}
```

```

GetAccessPointsList();
Fill_Lcd(0);

ClearDosimetreList();
int i = 0, indexCount = 0;
for(i; i < MAX_DOSIMETRES; i++)
{
    char listNum[5] = {0}, listName[20] = {0}, dosiName[10] = {0};

    //snprintf(dosiMac, sizeof(dosiMac), "%x:%x:%x:%x:%x:00", 'D', 'O', 'S',
'I', (char)(i + 1));

    snprintf(dosiName, sizeof(dosiName), "DOSI%d!", i + 1);

    if(strstr(espResponse, dosiName))
    {
        snprintf(listNum, sizeof(listNum), "%d", indexCount + 1);
        snprintf(listName, sizeof(listName), "%s : %s", listNum, dosiName);
        LcdGotoXY(0, indexCount);
        LcdString(listName);
        dosimetres[indexCount] = (i + 1);
        indexCount++;
    }
}
if(strlen(dosimetres) == 0)
{
    Fill_Lcd(0);
    LcdGotoXY(0, 0);
    LcdString("Aucun dosimetres.");
    LcdGotoXY(0, 3);
    LcdString("Touche * pour retour...");
}

while(1)
{
    /*

```

```

    Si des dosimetres ont été trouvés
    */
    if(strlen(dosimetres) > 0)
    {
        char tempInput = Lecture_Clav();
        if(tempInput != 'n' && tempInput != '*' && tempInput != '#')
        {
            numberInput[inputCount] = tempInput;
            if(inputCount < 1) inputCount++;
            Delay(100);

        }
        else if(tempInput == '#')
        {
            int dosiNum = 0, i = 0;

            /*
            Nous convertissons les deux chiffres entré en un int
            */
            if(!IsCharDigit(numberInput[1]) && IsCharDigit(numberInput[0]))
dosiNum = (numberInput[0] - '0');
            else if(!IsCharDigit(numberInput[0]) &&
IsCharDigit(numberInput[1])) dosiNum = (numberInput[1] - '0');
            else if(IsCharDigit(numberInput[0]) &&
IsCharDigit(numberInput[1]))
            {
                dosiNum = (numberInput[0] - '0') * 10 + numberInput[1] - '0';
            }
            numberInput[0] = ' ', numberInput[1] = ' ';
            inputCount = 0;

            for(i; i < MAX_DOSIMETRES;i++)
            {
                if((i + 1) == dosiNum && dosimetres[i] > 0)
                {
                    ShowDosiMenu(dosimetres[i]);
                    break;
                }
            }
        }
    }

```



```

        }
    }
}

if(Lecture_Clav() == '*' )
{
    ShowMenu();
    break;
}
}
}

void ShowDosiMenu(int dosinum)
{
    char dosiName[25] = {0};
    snprintf(dosiName, sizeof(dosiName), "DOSI%d!", dosinum);

    Fill_Lcd(0);
    LcdGotoXY(0,0);
    LcdString("Connexion...");
    /*
    * Connexion au point d'accès
    */
    ClearUartEspBuffer();
    ConnectToSSID(dosiName, "");
    Delay(5000);

    Fill_Lcd(0);
    LcdGotoXY(0,0);
    LcdString(dosiName);

    LcdGotoXY(0,2);
    LcdString("4 : Deboost");
    LcdGotoXY(0,3);
    LcdString("6 : Boost");
    LcdGotoXY(0, 4);
}

```

```

    LcdString("0 : RAZ");
    char impulsType = 0;
    while(1)
    {
        if( (impulsType = Lecture_Clav()) == '6' || (impulsType = Lecture_Clav())
== '4' || (impulsType = Lecture_Clav()) == '0')
        {
            int count = 0;
            while (!configurationWifi.connected && count++ < 5)
            {
                ClearUartEspBuffer();

                char IP[32] = {0};

                snprintf(IP, sizeof(IP), "192.168.1.%d", dosinum);

                ConnectToServer(IP, "1500");
                Delay(100);
                /*
                 * Fonction permettant de vérifier n fois si la connexion est
établie
                 */
                WaitForConnect(1);

            }

            /*
             * Si la connexion a bien été établie
            */
            if (configurationWifi.connected == 1)
            {
                ClearUartEspBuffer();
                char atSend[16] = {0}, impuls[10] = {0};

                if(impulsType == '6') snprintf(impuls, sizeof(impuls), "!BOOST");
                else if(impulsType == '4') snprintf(impuls, sizeof(impuls),
"!DEBOOST");
                else snprintf(impuls, sizeof(impuls), "!RAZ");
            }
        }
    }

```

```

                snprintf(atSend, sizeof(atSend), "AT+CIPSEND=%d\r\n",
strlen(impuls) + 4);
                SendData(atSend);
                Delay(100);
                ClearUartEspBuffer();
                SendData(impuls);
            }

            Delay(100);
        }
        else if(Lecture_Clav() == ' * ' )
        {
            DisconnectFromServer();
            Delay(200);
            ShowDosiListMenu();
            break;
        }
    }
}

```

nokia.c

```

#include <p33Fxxx.h>

#define BIT0  0b0000000000000001
#define BIT1  0b0000000000000010
#define BIT2  0b0000000000000100
#define BIT3  0b0000000000001000
#define BIT4  0b0000000000010000
#define BIT5  0b0000000000100000
#define BIT6  0b0000000001000000
#define BIT7  0b0000000010000000
#define BIT8  0b0000000100000000
#define BIT9  0b0000001000000000
#define BIT10 0b0000010000000000
#define BIT11 0b0000100000000000
#define BIT12 0b0001000000000000
#define BIT13 0b0010000000000000
#define BIT14 0b0100000000000000
#define BIT15 0b1000000000000000

```

```

/*-----
LCD Nokia 3310
-----*/
//Sans SPI
//8 RES    7 VOUT    6 GND    5 CS    4 DC    3 DIN    2 SCL    1 VDD
//Avec SPI

#define Lcd_DIN      LATBbits.LATB7    // patte 7
#define Lcd_CLK      LATBbits.LATB6    // patte 15
#define Lcd_DC       LATBbits.LATB8    // patte 10
#define Lcd_CE       LATBbits.LATB12   // patte 3
#define Lcd_RST      LATBbits.LATB9    // patte 2
#define LCD_PinsOut  TRISB&=~(BIT6|BIT7|BIT8|BIT9|BIT12);
//TRISA&=~(BIT0|BIT1|BIT3|BIT4|BIT5);
#define LCDInitState LATB|=(BIT9|BIT12); // CE=RST="1"

// Mapping des signaux SPI1 (LCD Nokia)
//#define MapLCDSCLK _RP6R=8 // SCK2 mappée sur RP6=RB6=LCDCLK    patte 15
//#define MapLCDSDIN _RP3R=7 // SDO2 mappée sur RP3=RB3=LCDSDIN    patte 7
//#define SPIPutLCD SPI2Put    // Coupleur SPI2r

/*-----
                                LCD Nokia 3310
-----*/

#define PosParam      32
// Résolution
#define Lcd_X_res      84
#define Lcd_Y_res      48
// Commandes controleur PCD8544
#define Function_Set    0x20    //Bit 2 = PD, Bit 1 = V, Bit 1 = H
#define Lcd_ON_Norm_H    0x20    //Lcd actif, adressage H, mode normal
#define Lcd_ON_Norm_V    0x22    //Lcd actif, adressage V, mode normal
#define Lcd_ON_Etendu_H  0x21    //Lcd actif, adressage H, mode étendu (H=1)
#define Lcd_OFF_Norm_H   0x00    //Lcd "power down", adressage H, mode normal
// Commandes normales (bit H="0")
#define Set_Config      0x08    //Bit 2 = D, bit 0 = E
#define Set_Adr_Y       0x40    //Adresse Y dans 3 bits LSB (0 à 5)
#define Set_Adr_X       0x80    //Adresse X dans 7 bits LSB (0 à 83)
// Commandes étendues (bit H="1")

```

```

#define Set_Bias      0x10    //Bias Mode dans bits 2 à 0
#define Set_Vop       0x80    //Vop dans bits 6 à 0
#define Set_TC        0x04    //Coef de température dans bits 1 à 0

/*-----
                               Variables spécifiques
-----*/

char Adr_X ;                // Copie logicielle de l'adresse X du PCD8544 (0 à 83)
char Adr_Y ;                // Copie logicielle de l'adresse Y du PCD8544 (0 à 5)

typedef enum
{
    Lcd_Cmd   = 0,
    Lcd_Data  = 1
} LcdCmdData; // Type de transfert : commande ou données

typedef enum
{
    Blanc = 0,
    Noir  = 1
} Couleur; // Type de couleur

/*-----
                               Constantes en mémoire flash
-----*/

/*-----
Générateur de caractère en mémoire flash
Code Ascii, format 5x7 pixels.
Format : Y      X      1      2      3      4      5
          1      D10 D20 D30 D40 D50
          2      D11 D21 D31 D41 D51
          3      D12 D22 D32 D42 D52
          4      D13 D23 D33 D43 D53
          5      D14 D24 D34 D44 D54
          6      D15 D25 D35 D45 D55
          7      D16 D26 D36 D46 D56
1° octet : D16-D15-D14-D13-D12-D11-D10, bit 7 toujours à "0"
2° octet : D26-D25-D24-D23-D22-D21-D20, bit 7 toujours à "0"
3° octet : D36-D35-D34-D33-D32-D31-D30, bit 7 toujours à "0"

```

4° octet : D46-D45-D44-D43-D42-D41-D40, bit 7 toujours à "0"

5° octet : D56-D55-D54-D53-D52-D51-D50, bit 7 toujours à "0"

```
-----*/
const char FontLookup[][5] =
{
    { 0x00, 0x00, 0x00, 0x00, 0x00 }, // sp
    { 0x00, 0x00, 0x2f, 0x00, 0x00 }, // !
    { 0x00, 0x07, 0x00, 0x07, 0x00 }, // "
    { 0x14, 0x7f, 0x14, 0x7f, 0x14 }, // #
    { 0x24, 0x2a, 0x7f, 0x2a, 0x12 }, // $
    { 0xc4, 0xc8, 0x10, 0x26, 0x46 }, // %
    { 0x36, 0x49, 0x55, 0x22, 0x50 }, // &
    { 0x00, 0x05, 0x03, 0x00, 0x00 }, // '
    { 0x00, 0x1c, 0x22, 0x41, 0x00 }, // (
    { 0x00, 0x41, 0x22, 0x1c, 0x00 }, // )
    { 0x14, 0x08, 0x3E, 0x08, 0x14 }, // *
    { 0x08, 0x08, 0x3E, 0x08, 0x08 }, // +
    { 0x00, 0x00, 0x50, 0x30, 0x00 }, // ,
    { 0x10, 0x10, 0x10, 0x10, 0x10 }, // -
    { 0x00, 0x60, 0x60, 0x00, 0x00 }, // .
    { 0x20, 0x10, 0x08, 0x04, 0x02 }, // /
    { 0x3E, 0x51, 0x49, 0x45, 0x3E }, // 0
    { 0x00, 0x42, 0x7F, 0x40, 0x00 }, // 1
    { 0x42, 0x61, 0x51, 0x49, 0x46 }, // 2
    { 0x21, 0x41, 0x45, 0x4B, 0x31 }, // 3
    { 0x18, 0x14, 0x12, 0x7F, 0x10 }, // 4
    { 0x27, 0x45, 0x45, 0x45, 0x39 }, // 5
    { 0x3C, 0x4A, 0x49, 0x49, 0x30 }, // 6
    { 0x01, 0x71, 0x09, 0x05, 0x03 }, // 7
    { 0x36, 0x49, 0x49, 0x49, 0x36 }, // 8
    { 0x06, 0x49, 0x49, 0x29, 0x1E }, // 9
    { 0x00, 0x36, 0x36, 0x00, 0x00 }, // :
    { 0x00, 0x56, 0x36, 0x00, 0x00 }, // ;
    { 0x08, 0x14, 0x22, 0x41, 0x00 }, // <
    { 0x14, 0x14, 0x14, 0x14, 0x14 }, // =
    { 0x00, 0x41, 0x22, 0x14, 0x08 }, // >
    { 0x02, 0x01, 0x51, 0x09, 0x06 }, // ?
    { 0x32, 0x49, 0x59, 0x51, 0x3E }, // @
    { 0x7E, 0x11, 0x11, 0x11, 0x7E }, // A
    { 0x7F, 0x49, 0x49, 0x49, 0x36 }, // B

```

```

{ 0x3E, 0x41, 0x41, 0x41, 0x22 }, // C
{ 0x7F, 0x41, 0x41, 0x22, 0x1C }, // D
{ 0x7F, 0x49, 0x49, 0x49, 0x41 }, // E
{ 0x7F, 0x09, 0x09, 0x09, 0x01 }, // F
{ 0x3E, 0x41, 0x49, 0x49, 0x7A }, // G
{ 0x7F, 0x08, 0x08, 0x08, 0x7F }, // H
{ 0x00, 0x41, 0x7F, 0x41, 0x00 }, // I
{ 0x20, 0x40, 0x41, 0x3F, 0x01 }, // J
{ 0x7F, 0x08, 0x14, 0x22, 0x41 }, // K
{ 0x7F, 0x40, 0x40, 0x40, 0x40 }, // L
{ 0x7F, 0x02, 0x0C, 0x02, 0x7F }, // M
{ 0x7F, 0x04, 0x08, 0x10, 0x7F }, // N
{ 0x3E, 0x41, 0x41, 0x41, 0x3E }, // O
{ 0x7F, 0x09, 0x09, 0x09, 0x06 }, // P
{ 0x3E, 0x41, 0x51, 0x21, 0x5E }, // Q
{ 0x7F, 0x09, 0x19, 0x29, 0x46 }, // R
{ 0x46, 0x49, 0x49, 0x49, 0x31 }, // S
{ 0x01, 0x01, 0x7F, 0x01, 0x01 }, // T
{ 0x3F, 0x40, 0x40, 0x40, 0x3F }, // U
{ 0x1F, 0x20, 0x40, 0x20, 0x1F }, // V
{ 0x3F, 0x40, 0x38, 0x40, 0x3F }, // W
{ 0x63, 0x14, 0x08, 0x14, 0x63 }, // X
{ 0x07, 0x08, 0x70, 0x08, 0x07 }, // Y
{ 0x61, 0x51, 0x49, 0x45, 0x43 }, // Z
{ 0x00, 0x7F, 0x41, 0x41, 0x00 }, // [
{ 0x55, 0x2A, 0x55, 0x2A, 0x55 }, // 55
{ 0x00, 0x41, 0x41, 0x7F, 0x00 }, // ]
{ 0x04, 0x02, 0x01, 0x02, 0x04 }, // ^
{ 0x40, 0x40, 0x40, 0x40, 0x40 }, // _
{ 0x00, 0x01, 0x02, 0x04, 0x00 }, // '
{ 0x20, 0x54, 0x54, 0x54, 0x78 }, // a
{ 0x7F, 0x48, 0x44, 0x44, 0x38 }, // b
{ 0x38, 0x44, 0x44, 0x44, 0x20 }, // c
{ 0x38, 0x44, 0x44, 0x48, 0x7F }, // d
{ 0x38, 0x54, 0x54, 0x54, 0x18 }, // e
{ 0x08, 0x7E, 0x09, 0x01, 0x02 }, // f
{ 0x0C, 0x52, 0x52, 0x52, 0x3E }, // g
{ 0x7F, 0x08, 0x04, 0x04, 0x78 }, // h
{ 0x00, 0x44, 0x7D, 0x40, 0x00 }, // i
{ 0x20, 0x40, 0x44, 0x3D, 0x00 }, // j

```

```

    { 0x7F, 0x10, 0x28, 0x44, 0x00 }, // k
    { 0x00, 0x41, 0x7F, 0x40, 0x00 }, // l
    { 0x7C, 0x04, 0x18, 0x04, 0x78 }, // m
    { 0x7C, 0x08, 0x04, 0x04, 0x78 }, // n
    { 0x38, 0x44, 0x44, 0x44, 0x38 }, // o
    { 0x7C, 0x14, 0x14, 0x14, 0x08 }, // p
    { 0x08, 0x14, 0x14, 0x18, 0x7C }, // q
    { 0x7C, 0x08, 0x04, 0x04, 0x08 }, // r
    { 0x48, 0x54, 0x54, 0x54, 0x20 }, // s
    { 0x04, 0x3F, 0x44, 0x40, 0x20 }, // t
    { 0x3C, 0x40, 0x40, 0x20, 0x7C }, // u
    { 0x1C, 0x20, 0x40, 0x20, 0x1C }, // v
    { 0x3C, 0x40, 0x30, 0x40, 0x3C }, // w
    { 0x44, 0x28, 0x10, 0x28, 0x44 }, // x
    { 0x0C, 0x50, 0x50, 0x50, 0x3C }, // y
    { 0x44, 0x64, 0x54, 0x4C, 0x44 } // z
};

/*-----
Générateur de caractères
Nombre de caractères : 10
Résolution X : 10
Résolution Y : 14 */
const char MyFontTab[][10] =
{
    { 252, 254, 3, 3, 3, 3, 3, 3, 254, 252 }, // 0 octet LSB
    { 0, 16, 24, 12, 6, 255, 255, 0, 0, 0 }, // 1 octet LSB
    { 2, 3, 131, 131, 131, 131, 131, 195, 254, 124 }, // 2 octet LSB
    { 2, 3, 3, 195, 195, 195, 195, 199, 254, 60 }, // 3 octet LSB
    { 192, 224, 48, 24, 12, 198, 195, 1, 0, 0 }, // 4 octet LSB
    { 31, 63, 99, 99, 99, 99, 99, 99, 195, 131 }, // 5 octet LSB
    { 252, 254, 195, 195, 195, 195, 195, 195, 131, 0 }, // 6 octet LSB
    { 2, 3, 3, 3, 131, 195, 99, 51, 31, 15 }, // 7 octet LSB
    { 124, 254, 195, 195, 195, 195, 195, 195, 254, 60 }, // 8 octet LSB
    { 124, 254, 195, 195, 195, 195, 195, 195, 254, 252 }, // 9 octet LSB
    { 15, 31, 48, 48, 48, 48, 48, 48, 31, 15 }, // 0 octet MSB
    { 0, 0, 0, 48, 48, 63, 63, 48, 48, 0 }, // 1 octet MSB
    { 62, 63, 51, 49, 49, 49, 49, 49, 48, 48 }, // 2 octet MSB
    { 16, 48, 48, 48, 48, 48, 48, 56, 31, 15 }, // 3 octet MSB
    { 3, 3, 3, 3, 3, 63, 63, 3, 3, 3 }, // 4 octet MSB

```



```

    { 16, 48, 48, 48, 48, 48, 48, 48, 31, 15 }, // 5 octet MSB
    { 15, 31, 48, 48, 48, 48, 48, 48, 31, 15 }, // 6 octet MSB
    { 56, 60, 6, 3, 1, 0, 0, 0, 0, 0 }, // 7 octet MSB
    { 15, 31, 48, 48, 48, 48, 48, 48, 31, 15 }, // 8 octet MSB
    { 0, 48, 48, 48, 48, 48, 48, 48, 31, 15 } // 9 octet MSB
};

/*-----
    Déclarations des fonctions en mémoire flash
-----*/

/*-----
    Nom          : LcdSend
    Description   : Envoi d'un octet vers le controleur du LCD
    Arguments    : data -> Donnée ou commande à transmettre
                  cd   -> Type de transfert : commande (Lcd_Cmd)
                  ou donnée (Lcd_Data)
    Valeur renvoyée : aucune.
-----*/

void LcdSend(char Data, LcdCmdData CD)
{
    int i;
    // Sélection du controleur (actif à "0").
    Lcd_CE=0; // CE = "0"
    // positionnement de la patte Data/Commande
    if (CD == Lcd_Data) Lcd_DC=1;
        else Lcd_DC=0;

    // Transmission série de l'octet sur DIN rythmé par CLK
    for (i=0; i<8; i++)
    {
        if ((Data & 0x80)==0) Lcd_DIN=0;
            else Lcd_DIN=1;

        Data <<= 1;
        Lcd_CLK=1; // pour assurer durée mini 125ns de état haut avec FCY = 40MHz
        Lcd_CLK=1;
        Lcd_CLK=1;
        Lcd_CLK=1;
        Lcd_CLK=0;
    }
    // Dé-sélection du controleur

```

```

    Lcd_CE=1;
}

/*-----
    Nom          :   Fill_Lcd
    Description   :   Remplir le LCD de la même couleur
    Arguments    :   Coul_Lcd : Noir ou Blanc
    Valeur renvoyée : aucune.
-----*/

void Fill_Lcd(Couleur Coul_Lcd)
{
    int x,y;
    char b;
    if (Coul_Lcd==Noir) b=255;
        else b=0;
    for (y=0; y<Lcd_Y_res/8+1; y++)
    {
        LcdSend(Set_Adr_X+0,Lcd_Cmd); //Adresse X = 0
        LcdSend(Set_Adr_Y+y,Lcd_Cmd); //Adresse Y
        for (x=0; x<Lcd_X_res; x++) LcdSend(b,Lcd_Data);
    }
    Adr_X=0; Adr_Y=0;
    LcdSend(Set_Adr_X+0,Lcd_Cmd); //Adresse X = 0
    LcdSend(Set_Adr_Y+0,Lcd_Cmd); //Adresse Y = 0
}

/*-----
    Nom          :   LcdChar
    Description   :   Dessin d'un caractère à la position actuelle des compteurs X et Y
    Arguments    :   Code Ascii du caractère
    Valeur renvoyée : aucune.
-----*/

void LcdChar(char c)
{
    int i;
    for (i=0; i<5; i++) LcdSend(FontLookup[c-32][i],Lcd_Data);
    Adr_X += 6;
}

/*-----
    Nom          :   LcdString
    Description   :   Dessin d'une phrase à la position actuelle des compteurs X et Y
    Arguments    :   Pointeur de la chaîne de caractères

```

```

    Valeur renvoyée : aucune.
-----*/
void LcdString(char *Phrase)
{
    while (*Phrase)
    {
        LcdChar(*Phrase++);
        LcdSend(0,Lcd_Data); // Espace inter-caractères
        Adr_X++;
    }
}
/*-----
    Nom          : LcdBigDigit
    Description   : Dessin d'un chiffre 14x10 pixels à la position actuelle des
                   compteurs X et Y
    Arguments    : Chiffre codé BCD
    Valeur renvoyée : aucune.
-----*/
void LcdBigDigit(char b)
{
    int i;
    for (i=0; i<10; i++) LcdSend(MyFontTab[b][i],Lcd_Data); // D'abord les octets LSB
    LcdSend(Set_Adr_X+Adr_X,Lcd_Cmd); // Retour au début du dessin
    LcdSend(Set_Adr_Y+Adr_Y+1,Lcd_Cmd); // Octets MSB à la ligne de 8 pixels inférieure
    for (i=0; i<10; i++) LcdSend(MyFontTab[b+10][i],Lcd_Data); // Puis les octets MSB
    Adr_X += 12;
    LcdSend(Set_Adr_Y+Adr_Y,Lcd_Cmd); //Prochain chiffre sur la même ligne
    LcdSend(Set_Adr_X+Adr_X,Lcd_Cmd); //et 12 pixels à droite
}
/*-----
    Nom          : LcdBigString
    Description   : Dessin d'une suite de chiffres à la position actuelle des compteurs X et
    Y
    Arguments    : Pointeur de la chaîne de caractères
    Valeur renvoyée : aucune.
-----*/
void LcdBigString(char *Chaine)
{
    while (*Chaine) LcdBigDigit(*Chaine++ - '0');
}

```

```

/*-----
Nom          :  LcdGotoXY
Description   :  Affecter les adresses X et Y du PCD8544
Arguments    :  X : position H (0 à 83)
                Y : position V (0 à 5)
Valeur renvoyée : aucune.
-----*/
void LcdGotoXY(char x, char y)
{
    Adr_X=x; Adr_Y=y;
    LcdSend(Set_Adr_X+Adr_X,Lcd_Cmd); // Affectation adresse X
    LcdSend(Set_Adr_Y+Adr_Y,Lcd_Cmd); // Affectation adresse Y
}

/*-----
Nom          :  BarGraphLcd
Description   :  Dessiner une barre noire horizontale sur le LCD
Arguments    :  Y : position verticale (Y:0 à 5)
                LngLine : longueur de la barre en pixels
Valeur renvoyée : aucune.
-----*/
void BarGraphLcd(char Y, int LngLine)
{
    int i;
    LcdGotoXY(0,Y);
    for (i=0; i<Lcd_X_res; i++)
    {
        if (i<2) LcdSend(0,Lcd_Data);
        else if (i<LngLine) LcdSend(0x3C,Lcd_Data);
        else LcdSend(0,Lcd_Data);
    }
}

/*-----
Nom          :  LcdClrLine
Description   :  Effacer la ligne complète donnée en argument
Arguments    :  Y : numéro de la ligne
Valeur renvoyée : aucune.
-----*/
void LcdClrLine(char Y)
{
    int i;

```

```

LcdGotoXY(0,Y);
for (i=0; i<Lcd_X_res; i++)
{
    LcdSend(0,Lcd_Data); // Effacer une ligne verticale de 8 pixels
}
}

/*-----
Nom          :   write
Description   :   Sorties de caractères des fonctions "printf"
Arguments    :
Valeur renvoyée :
-----*/

int write(int handle, void *buffer, unsigned int len)
{
    int i;
    if (Adr_X >= Lcd_X_res) return(len);
    switch (handle)
    {
        case 0:
        case 1:
        case 2:
            for (i = len; i; --i)
            {
                LcdChar(*(char*)buffer);
                LcdSend(0,Lcd_Data); // Espace inter-caractères
                Adr_X++;
            }
            break;
        default: break;
    }
    return(len);
}

/*-----
** Delay utilisant TMR1 en t en ms à vérifier
-----*/

void Delay( unsigned int t)
{
    while (t--)
    {

```

```

        TMR1 = 0;
        while (TMR1<16000);
    }
}
*/
/*-----
Nom          :  LcdInit
Description   :  Initialisation du PCD8544
Arguments    :  aucun
Valeur renvoyée : aucune.
-----*/
void LcdInit(void)
{
    LCD_PinsOut; // instruction d'initialisation des ports définie dans les #define
    LCDInitState; // instruction d'initialisation des ports définie dans les #define
    Lcd_CE=1; // Controleur non sélectionné (CE = "1")
    Lcd_RST=1; // et pas de reset (RST = "1")
    Delay(500); // Délai de 0,5s
    Lcd_RST=0; // Reset PCD8544 (RST = "0")
    Delay(500); // Délai de 0,5s
    Lcd_RST=1; // Arrêt reset PCD8544 (RST = "1")
    Adr_X=0; Adr_Y=0;
    LcdSend(Lcd_ON_Etendu_H, Lcd_Cmd); //Mode commandes étendues 0x21
    LcdSend(Set_Vop+72,Lcd_Cmd); //Set LCD Vop (contrast) 0x80
    LcdSend(Set_TC+2,Lcd_Cmd); //Set temp. coefficient 0x04
    LcdSend(Set_Bias+3,Lcd_Cmd); //LCD Bias mode 1:48 0x10
    LcdSend(Lcd_ON_Norm_H,Lcd_Cmd); //LCD standard commands. Horizontal addressing mode. 0x20
    LcdSend(Set_Config+4,Lcd_Cmd); //LCD in normal mode (DE = 10) 0x08
}

/*****
Nom          :  Conv_Bin8_String
Description   :  Conversion d'une valeur codée 8 bits en binaire
                -> chaîne de caractères
Arguments    :  x : nombre binaire à convertir
                *String : pointeur vers la chaîne de destination
Valeur renvoyée : aucune
*****/
void Conv_Bin8_String(char *String, unsigned char x)
{

```

```
String[0]=x/100; // Centaines
String[1]=(x-(100*String[0]))/10; // Dizaines
String[2]=x-(100*String[0])-10*String[1]+'0'; // Unités en Ascii
String[0]+='0'; // Conversion BCD -> Ascii des centaines
String[1]+='0'; // Conversion BCD -> Ascii des dizaines
String[3]=0; // Termineur de chaîne
}

/*
SPI1CON1 = 0x0423; // Master, 16 bit, disable SCK, disable SS, prescaler 1:8
SPI1STAT = 0x8000; // Enable SPI port
_SPI1IP=6; // Priorité élevée
_SPI1IF=0;
_SPI1IE=1;

/*-----
Function : void _ISR _SPI1Interrupt(void)
Description:
- Production du signal vidéo de contrôle
- Déclenchée à la fin de la transmission du mot précédent
Traitements :
- Décrémenter "CptWord" et détecter son état pour transmettre le bon nombre
de mots
- Transmission le cas échéant du mot suivant depuis "Image"
-----*/

/*void _ISRFAST _SPI1Interrupt(void)
{
int dummy;
dummy = SPI1BUF; // Lecture du dernier mot reçu pour acquitter l'écriture précédente
CptWord--;
if (CptWord)
{
if (CptWord==1) SPI1BUF = (*ImagePtrOut++)&0xFFFE; // Forcer le dernier bit à 0
else SPI1BUF = *ImagePtrOut++;
}
else
{
_SPI1IE=0; // Dernier mot transmis : inhiber les interruptions

```

```
}  
_SPI1IF = 0;  
}  
*/
```

uart.c

```
#include <p33Fxxxx.h>  
#include "defines.h"  
  
extern int countU1;  
extern int countU2;  
extern char espResponse[SIZE_ESPBUFF];  
extern char seqResponse[SIZE_SEQBUFF];  
/*-----  
                                Câblage  
-----*/  
#define U1_Tx RPOR9=0x0003;    // U1Tx du pic câblé sur RP18  
#define U1_Rx RPINR18=0x1F11;  // U1Rx du pic câblé sur RP17  
  
#define U2_Tx RPOR1=0x0005;    // U2Tx du pic câblé sur RP2  
#define U2_Rx RPINR19=0x1F03;  // U2Rx du pic câblé sur RP3  
  
/*-----  
Nom          : void InitUART1 (long Bauds)  
Description  : Programme d'interruption déclenché à la réception  
                d'un caractère  
Arguments    : Bauds  
Valeur renvoyée : aucune  
-----*/  
  
void InitUART1(long Bauds)  
{  
    U1_Tx; // câblage TxD
```



```

U1_Rx; // câblage Rx

U1MODE=0x8000; // Module UART validé, 8 bits, pas de parité, 1 bit stop
                // Pas d'inversion des signaux RX1 et TX1
                // RTS et CTS non utilisés
U1STA =0x0400; // UART TX validé
                // Interruption RX à chaque caractère
U1BRG =FCY/((long)16*Bauds)-1; // Vitesse de transmission

_U1RXIP=7;      // Priorité maximum car buffer hard de faible capacité
_U1RXIF=0;      // Raz indicateur interruption RX
_U1RXIE=1;      // Validation interruption RX

ClearUartEspBuffer();
}

/*-----
Nom          : void InitUART1 (long Bauds)
Description : Programme d'interruption déclenché à la réception
                d'un caractère
Arguments    : Bauds
Valeur renvoyée : aucune
-----*/

void InitUART2(long Bauds)
{
    U2_Tx; // câblage Tx
    U2_Rx; // câblage Rx

    U2MODE=0x8000; // Module UART validé, 8 bits, pas de parité, 1 bit stop
                    // Pas d'inversion des signaux RX1 et TX1
                    // RTS et CTS non utilisés
    U2STA =0x0400; // UART TX validé
                    // Interruption RX à chaque caractère
    U2BRG =FCY/((long)16*Bauds)-1; // Vitesse de transmission

    _U2RXIP=7;      // Priorité maximum car buffer hard de faible capacité

```

```

_U2RXIF=0;      // Raz indicateur interruption RX
_U2RXIE=1;      // Validation interruption RX

ClearUartSeqBuffer();
}

/*-----
Nom          : void _ISR _U1RXInterrupt (void)
Description : Programme d'interruption déclenché à la réception
              d'un caractère
Arguments    : aucun
Valeur renvoyée : aucune
-----*/

void _ISR _U1RXInterrupt(void)
{
    _U1RXIF=0;      // Clear interrupt flag
    while (U1STAbits.URXDA) // Boucle pour lire ts les car. ds le buf. de l'UART
    {
        if(countU1 + 1 < sizeof(espResponse))
            espResponse[countU1++]=U1RXREG;
    }
}

/*-----
Nom          : void _ISR _U1RXInterrupt (void)
Description : Programme d'interruption déclenché à la réception
              d'un caractère
Arguments    : aucun
Valeur renvoyée : aucune
-----*/

void _ISR _U2RXInterrupt(void)
{
    _U2RXIF=0;      // Clear interrupt flag
    while (U2STAbits.URXDA) // Boucle pour lire ts les car. ds le buf. de l'UART

```

```

    {
        seqResponse[countU2++] = U2RXREG;
    }
}

void SendData(char* message)
{
    int i=0, var=1;

    for(i=0;message[i]!='\0';i++)
    {
        while (U1STAbits.TRMT==0){}
        U1TXREG=message[i];
        int j=0;
        for(j=0;j<1000;j++){var++;}
    }
}

void SendData2(char* message)
{
    int i=0, var=1;
    for(i=0;message[i]!='\0';i++)
    {
        while (U2STAbits.TRMT==0){}
        U2TXREG=message[i];
        int j=0;
        for(j=0;j<1000;j++){var++;}
    }
}

/*-----
Nom          : void ClearEspBuffer(void)
Description : Programme qui vide le Buffer de réception RXD
Arguments    : durée en ms
Valeur renvoyée : aucune
-----*/

```

```
void ClearUartEspBuffer(void)
{
    int i;
    for (i=0;i<sizeof(espResponse);i++) espResponse[i]=0;
    countU1=0;
}

void ClearUartSeqBuffer(void)
{
    int i;
    for (i=0;i<sizeof(seqResponse);i++) seqResponse[i]=0;
    countU2=0;
}

int IsUart2Connected()
{
    return !_RA0;
}
```

wifi.h

```
/*
 * File:    wifi.h
 * Author:  ariehl
 *
 * Created on 16 mars 2016, 16:41
 */

#ifndef WIFI_H
#define WIFI_H

typedef struct{

    int mode; // Mode client ou serveur
    char* IP; // IP du serveur ou du client
    char* SSID; // SSID du serveur (si configuré en serveur)
    char* ssidPass; // Mot de passe du SSID serveur (si configuré en serveur)
    char* port; // Port du serveur (si configuré en serveur)
    int connected;

}Wifi;
```

```
extern Wifi configurationWifi;

#define MODE_CLIENT 1
#define MODE_SERVER 2

#endif      /* WIFI_H */
```

Wifi.C

```
#include <p33Fxxxx.h>
#include "wifi.h"
#include "defines.h"
#include <string.h>

extern char espResponse[SIZE_ESPBUFF];

void SetWifiMode(int mode)
{
    if(mode == MODE_CLIENT)
    {
        SendData("AT+CWMODE=1\r\n");
    }
    else if(mode == MODE_SERVER)
    {
        SendData("AT+CWMODE=2\r\n");
    }
}

void SetWifiIP(char* IP)
{
    char atIP[128];
    snprintf(atIP, sizeof(atIP), "AT+CIPSTA=\"%s\"\r\n", IP);
    SendData(atIP);
}
```

```

void SetMultiConnection(int cipmux)
{
    if(!cipmux) SendData("AT+CIPMUX=0\r\n");
    else SendData("AT+CIPMUX=1\r\n");
}

void ConnectToSSID(char* SSID, char* password)
{
    char atSSID[128];
    snprintf(atSSID, sizeof(atSSID), "AT+CWJAP=\"%s\", \"%s\"\r\n", SSID,
password);
    SendData(atSSID);
}

void ConnectToServer(char* IP, char* port)
{
    char atServ[128];
    snprintf(atServ, sizeof(atServ), "AT+CIPSTART=\"TCP\", \"%s\", %s\r\n", IP,
port);
    SendData(atServ);
}

void WaitForConnect(int essai)
{
    while(configurationWifi.connected == 0)
    {
        if(essai == 0) break;
        essai--;
        int i = 0;
        char *p=espResponse;
        for (i = 0; i < strlen(espResponse); i++)
        {
            p++;
            if ((*p=='N') && (*(p+1)=='E') && (*(p+2)=='C') && (*(p+3)=='T')
&& (*(p+4)=='\r') && (*(p+5)=='\n'))
            {
                configurationWifi.connected = 1;
            }
        }
    }
}

```

```

    }
}

}

}

void DisconnectFromServer()
{
    if(configurationWifi.connected)
    {
        SendData("AT+CIPCLOSE'\r\n");
        configurationWifi.connected = 0;
    }
}

void InitWifi()
{
    /*
        Paramètre du module
    */
    char IP[] = "192.168.1.254";
    configurationWifi.IP = IP;
    configurationWifi.mode = 1;
    configurationWifi.connected = 0;

    Fill_Lcd(0);
    LcdGotoXY(0,0);
    LcdString("Initialisation");
    LcdGotoXY(0,2);
    LcdString(IP);
    LcdGotoXY(0,3);
    if(configurationWifi.mode == 1)
        LcdString("Mode : client");
    else if(configurationWifi.mode == 2)
        LcdString("Mode : serveur");
}

```

```
SendData("AT+RST\r\n");
Delay(500); // Temporisation pour le reset

ClearUartEspBuffer();

/*
    Configuration du module en mode client
*/

SetWifiMode(configurationWifi.mode);
Delay(100); // Attente de la réponse

ClearUartEspBuffer();

/*
    Configuration de l'ip du module
*/

SetWifiIP(configurationWifi.IP);
Delay(250); // Attente d'une réponse

ClearUartEspBuffer();

/*
    Modification du nombre de connexion en simultanés (0 pour un client)
*/
SetMultiConnection(0);
Delay(100);
ClearUartEspBuffer();
}

void GetAccessPointsList()
{
    ClearUartEspBuffer();
    SendData("AT+CWLAP\r\n");
```



```
    Delay(5000);  
}
```

7.3. Interface web

activity.php

```
<?php  
include ( "../includes/mysql.php" );  
include ( "../includes/head.php" );  
include ( "../includes/header.php" );  
  
$db = enableConnection();  
?>  
  
<!DOCTYPE html>  
<html>  
    <?php setHead("Gestion des activités", "UTF-8", "activity"); ?>  
    <body>  
        <?php setHeader(); ?>  
        <div class="container valign-wrapper">  
            <div class="row" style="width: 100%;">  
                <div id="table_activity" class="col s12" style="margin:  
10px"><!--Div contenant la table de la base de donn"es--></div>  
                <div id="modal_activity" class="col s12" style="margin:  
10px"><!--Div contenant les modals pour modifier la base de données activity--  
></div>  
            </div>  
        </div>  
    </body>  
</html>
```

ajax_activity.php

```
<?php  
  
include ( "../includes/mysql.php" );  
  
$db = enableConnection();  
  
function GetUsersList()  
{  
    $db = enableConnection();  
    $query = $db->query("SELECT Prenom, Nom, ID_Personne FROM users");  
    $result = $query->fetchAll();  
    $db = NULL;
```

```

        return $result;
    }

    if(isset($_POST['type']) && $_POST['type'] == "view_rtr")
    {
        $query = $db->prepare("SELECT aIZ FROM activity_config WHERE aID=:aid AND
aUserID=:userid");
        $query->execute(array("aid" => $_POST['activityid'], "userid" =>
$_POST['userid']));
        $result = $query->fetch();
        echo $result['aIZ'];
    }

    if(isset($_POST['type']) && $_POST['type'] == "add_user")
    {
        srand();
        $iz = rand (1000, 5000);
        $iz *= 9999;

        $query = $db->prepare("INSERT INTO activity_config
(aID,aUserID,aIZ,aDED,aDose,aCoeff) VALUES (:aid, :userid, :aiz,:aded, :adose,
:acoeff)");
        $query->execute(array(
            "aid" => $_POST['activityid'],
            "userid" => $_POST['userID'],
            "aiz" => $iz,
            "aded" => $_POST['ded'],
            "adose" => $_POST['dose'],
            "acoeff" => $_POST['coeff']
        ));
    }

    if(isset($_POST['type']) && $_POST['type'] == "refresh_users_list")
    {
        if(isset($_POST['activityid']) && $_POST['activityid'] > 0)
        {
            $query = $db->prepare("SELECT * FROM activity_config WHERE aID =
:id");
            $query->execute(array("id" => $_POST['activityid']));
        }
    }

```

```

        $count = $query->rowCount();
        if($count)
        {
            while($result = $query->fetch())
            {
                $query2 = $db->prepare("SELECT * FROM users WHERE
ID_Personne = :id");
                $query2->execute(array("id" => $result['aUserID']));
                $user = $query2->fetch();
                echo "<tr>";
                echo "<td>".$user['Nom']. " ".$user['Prenom']. "</td>";
                echo "<td>".$result['aDED']. "</td>";
                echo "<td>".$result['aDose']. "</td>";
                echo "<td>".$result['aCoeff']. "</td>";
                echo "<td><a href='#' class='view_rtr'
userid='".$user['ID_Personne']."' >Voir</a></td>";
                echo "<td><a href='#' class='remove_user_act'
userid='".$user['ID_Personne']."' >Retirer</a></td>";
                echo "</tr>";
            }
        }
        else echo "<tr><td>Aucun utilisateurs lié à cette activité</td></tr>";
    }
}

if(isset($_POST["type"]) && $_POST["type"] == "modals")
{
    $var="
<div id='modal_add_act' class='modal'>
    <div class='modal-content'>
        <div class='row'>
            <div class='input-field col s6'>
                <input id='act_name' type='text' class='validate'>
                <label for='act_name'>Nom de l'activité</label>
            </div>
            <div class='input-field col s6'>
                <input id='act_class' type='text' class='validate'>
                <label for='act_class'>Classe</label>

```

```

        </div>
        <div class='input-field col s12'>
            <a id='btn_next_act' class='right waves-effect
waves-light btn'>Suivant</a>
        </div>
    </div>
</div>
<div id='modal_add_users_act' class='modal'>
    <div class='modal-content'>
        <div class='row'>
            <div class='input-field col s12'>
                <div class='input-field col s4'>
                    <select id='usersSelect' class='browser-
default'>
                        <option value='' disabled
selected>Choisissez un utilisateur</option>
                        ";
                        $userslist = GetUsersList();
                        foreach($userslist as $key)
                        {
                            $var .= "<option
value='\".$key['ID_Personne'].\"'>\".$key['Nom'].\" \".$key['Prenom'].\"";
                        }
                        $var .= "
                    </select>
                </div>
                <div class='input-field col s2'>
                    <input id='act_ded' type='text'
class='validate'>
                    <label for='act_ded'>DED</label>
                </div>
                <div class='input-field col s2'>
                    <input id='act_dose' type='text'
class='validate'>
                    <label for='act_dose'>Dose théorique</label>
                </div>
                <div class='input-field col s2'>
                    <input id='act_coeff' type='text'

```

```

class='validate'>
    <label for='act_coeff'>Coefficient
d'exposition</label>
    </div>
    <div class='input-field col s2'>
        <a id='btn_add_user_act' class='btn-floating
btn-large waves-effect waves-light blue'><i class='material-icons'>add</i></a>
    </div>
</div>
<div id='table_act_config' class='input-field col s12'>
    <h5>Utilisateurs ajoutés</h5>
    <table class='bordered'>
        <thead>
            <tr>
                <th data-
field='PNom'>Utilisateur</th>
                <th data-field='DED'>DED</th>
                <th data-field='Dose'>Dose
théorique</th>
                <th data-
field='CoeffExp'>Coefficient d'exposition</th>
                <th data-field='RTR'>RTR</th>
                <th data-
field='Retirer'>Retirer</th>
            </tr>
        </thead>
        <tbody>" ;
            if(isset($_POST['activityid']) &&
                {
                    $query = $db->prepare("SELECT *
FROM activity_config WHERE aID = :id");
                    $query->execute(array("id" =>
$_POST['activityid']));
                    $count = $query->rowCount();
                    if($count)
                    {
                        while($result = $query-
>fetch())
                            {

```

```

>prepare("SELECT * FROM users WHERE ID_Personne = :id");
>execute(array("id" => $result['aUserID']));
>fetch();

$var .= "<tr>";
$var .=
$var .=
$var .=
$var .=
$var = "<td><a
href='#' class='view_rtr' userid='".$user['ID_Personne']."'>Voir</a></td>";
$var .= "<td><a
href='#' class='remove_user_act' userid='".$user['ID_Personne']."'>Retirer</a></td>";
$var .= "</tr>";
}
}
else $var .= "<tr><td>Aucun
utilisateurs lié à cette activité</td></tr>";
}
$var .=
"</tbody>
</table>
</div>
<div class='input-field col s12'>
<a id='btn_submit_act' class='right waves-effect
waves-light btn'>Fermer</a>
</div>
</div>
</div>";
echo $var;
echo $_SESSION['aID'];
}
// chargement de toute le table activity dans la page activity.php

```

```

if(isset($_POST["type"]) && $_POST["type"] == "activity")
{
    $rep = $db->query('SELECT * FROM activity ORDER BY aID');
    $count = $rep->rowCount();
    $var = "
    <a class='waves-effect waves-light btn' id='btn_add_act'>Ajouter</a>
    <a class='waves-effect waves-light btn' id='btn_delete_all_act'>Supprimer
tout</a>
    <h5>Liste des activités</h5>
    <table class='bordered'>
        <thead>
            <tr>
                <th data-field='Nom'>Nom</th>
                <th data-field='Date'>Date de création</th>
                <th data-field='Classe'>Classe</th>
                <th data-field='Modification'>Modification</th>
                <th data-field='Suppression'>Suppression</th>
            </tr>
        </thead>
        <tbody>
            ";
    if($count)
    {
        while($result = $rep->fetch(PDO::FETCH_ASSOC))
        {
            $checked = "";
            if($result['aUse']) $checked = "checked='checked' ";
            $var .= "<tr>";
            $var .= "<td>".$result['aName']. "</td>";
            $var .= "<td>".$result['aDate']. "</td>";
            $var .= "<td>".$result['aClasse']. "</td>";
            $var .= "<td><a href='#' class='modif_act'
activityid='".$result['aID']."'>Voir</a></td>";
            $var .= "<td><a href='#' class='delete_act'
activityid='".$result['aID']."'>Supprimer</a></td>";
            $var .= "</tr>";
        }
    }
    else $var .= "<tr><td>Aucune activité.</td></tr>";
}

```

```

        $var .= "</tbody></table>";
        echo $var;
    }

    if(isset($_POST["type"]) && $_POST["type"] == "add_act")
    {

        $query = $db->prepare("INSERT INTO activity (aName, aClasse, aDate) VALUES
(:name, :classe, NOW())");
        $query->execute(array(
            "name" => $_POST["name"],
            "classe" => $_POST["classe"]));
        echo $db->lastInsertId();
    }

    if(isset($_POST["type"]) && $_POST["type"] == "remove_user_act")
    {
        $query = $db->prepare("DELETE FROM activity_config WHERE aID=:id AND aUserID
= :userid");
        $query->execute(array("id" => $_POST["id"], "userid" => $_POST["userid"]));
    }

    if(isset($_POST["type"]) && $_POST["type"] == "delete_act")
    {
        $query = $db->prepare("DELETE FROM activity WHERE aID=:id");
        $query->execute(array("id" => $_POST["id"]));
        $query = $db->prepare("DELETE FROM activity_config WHERE aID=:id");
        $query->execute(array("id" => $_POST["id"]));
    }

    if(isset($_POST["type"]) && $_POST["type"] == "delete_all_act")
    {
        $query = $db->query("DELETE FROM activity");
        $query = $db->query("DELETE FROM activity_config");
    }

    ?>

```



```

jQuery.fn.extend({
  live: function (event, callback) {
    if (this.selector) {
      jQuery(document).on(event, this.selector, callback);
    }
  }
});

function containsWord(haystack, needle) {
  return (" " + haystack + " ").indexOf(" " + needle + " ") !== -1;
}

$( document ).ready(function() {

  $.ajax({
    method: "POST",
    url: "./ajax/ajax_activity.php",
    data: {type: "activity"}
  })
  .done (function(msg){
    $("#table_activity").html(msg);
  });

  /*
  Chargement des modals
  */
  $.ajax({
    method: "POST",
    url: "./ajax/ajax_activity.php",
    data: {type: "modals", activityid : 0}
  })
  .done (function(msg){
    $("#modal_activity").html(msg);
  });

  /*
  Suppression d'une activité
  */
  $(".delete_act").live("click", function() {

```

```

$.ajax({
  method: "POST",
  url: "../ajax/ajax_activity.php",
  data: { type: "delete_act", id: $(this).attr("activityid") }
})
.done (function(msg){
  $.ajax({
    method: "POST",
    url: "../ajax/ajax_activity.php",
    data: {type: "activity"}
  })
  .done (function(msg){

    $("#table_activity").html(msg);

  });
});
/*
  Suppression de toutes les activités
*/
$("#btn_delete_all_act").live("click", function() {

  $.ajax({
    method: "POST",
    url: "../ajax/ajax_activity.php",
    data: { type: "delete_all_act" }
  })
  .done (function(msg){
    $.ajax({
      method: "POST",
      url: "../ajax/ajax_activity.php",
      data: {type: "activity"}
    })
    .done (function(msg){

      $("#table_activity").html(msg);

```

```

        });

    });

    /**
     * Suppression d'un utilisateur d'une activité
     */
    $("#remove_user_act").live("click", function(){

        /**
         * Rafraichissement du tableau contenant la liste des utilisateurs
         */
        $.ajax({
            method: "POST",
            url: "./ajax/ajax_activity.php",
            data: {type: "remove_user_act", id :
$("#modal_add_users_act").attr("activityid"), userid : $(this).attr("userid")}
        })
        .done (function(msg){
            $.ajax({
                method: "POST",
                url: "./ajax/ajax_activity.php",
                data: {type: "refresh_users_list", activityid :
$("#modal_add_users_act").attr("activityid")}
            })
            .done (function(msg){
                $("#table_act_config tbody").html(msg);
            });
        });
    });

    /**
     * Création de l'activité
     */

    $("#btn_next_act").live("click", function() {

```

```

$.ajax({
  method: "POST",
  url: "../ajax/ajax_activity.php",
  data: { type: "add_act", name: $("#act_name").val(), classe:
$("#act_class").val() }
})
.done (function(msg){
  /*
    Rafrachissement du tableau qui affiche les activités
  */
  $.ajax({
    method: "POST",
    url: "../ajax/ajax_activity.php",
    data: {type: "activity"}
  })
  .done (function(msg){
    $("#table_activity").html(msg);
  });

  $("#modal_add_act").closeModal();
  $('#modal_add_users_act').openModal();

  /*
    Rafrachissement du tableau contenant la liste des
utilisateurs
  */
  var activityid = msg;
  $('#modal_add_users_act').attr("activityid", activityid);
  $.ajax({
    method: "POST",
    url: "../ajax/ajax_activity.php",
    data: {type: "refresh_users_list", activityid :
activityid}
  })
  .done (function(msg){
    $("#table_act_config tbody").html(msg);
  });
});

```

```

});

/*
    Ajout d'utilisateur dans une activité
*/
$("#btn_add_user_act").live("click", function() {

    $.ajax({
        method: "POST",
        url: "../ajax/ajax_activity.php",
        data: { type: "add_user", activityid:
$('#modal_add_users_act').attr("activityid"), userID: $("#usersSelect").val(),
ded: $("#act_ded").val(), dose : $("#act_dose").val(), coeff:
$("#act_coeff").val() }
    })
    .done (function(msg){
        $.ajax({
            method: "POST",
            url: "../ajax/ajax_activity.php",
            data: {type: "refresh_users_list", activityid :
$('#modal_add_users_act').attr("activityid")})
        })
        .done (function(msg){
            console.log(msg);
            $("#table_act_config tbody").html(msg);
        });
    });

});

/*
    Modification d'une activité
*/
$(".modif_act").live("click", function(){

    $('#modal_add_users_act').openModal();
    var activityid = $(this).attr("activityid");
    $('#modal_add_users_act').attr("activityid", activityid);

```

```

        $.ajax({
            method: "POST",
            url: "../ajax/ajax_activity.php",
            data: {type: "refresh_users_list", activityid :
activityid}
        })
        .done (function(msg){
            $("#table_act_config tbody").html(msg);
        });
    });
    /*
    Visualisation du RTR
    */
    $(".view_rtr").live("click", function(){

        var activityid = $(this).attr("activityid");
        $.ajax({
            method: "POST",
            url: "../ajax/ajax_activity.php",
            data: {type: "view_rtr", activityid :
$('#modal_add_users_act').attr("activityid"), userid : $(this).attr("userid")}
        })
        .done (function(msg){

            window.open("../includes/rtr.php?iz="+msg, "blank");

        });
    });

    /*
    Ouverture des modals
    */
    $("#btn_add_act").live("click", function() {
        $('#modal_add_act').openModal();
    });
});

```