# ANDROID & ECLIPSE

# Application ECG
# Communication en Bluetooth avec le protocole SPP

## 1. Description de l'application ECG

Le projet consiste à réaliser un électrocardiographe en utilisant une tablette Android comme terminal de visualisation.
Il s'agit d'un électrocardiographe simple, à 3 électrodes.
Les tablettes ou smartphones Android ne comportent pas, bien entendu, d'interface pour une connexion directe de ces électrodes.
Cette interface devra donc être réalisée de toutes pièces.
De plus, il n'est pas question de « bricoler » une tablette existante car leur durée de vie commerciale réduite rendrait vite cette application obsolète.
On utilise donc l'interface Bluetooth implantée aujourd'hui dans la majorité des terminaux Android.

Ce choix impose de créer une interface relativement complexe, comportant :

- Un module Bluetooth supportant le protocole série SPP,
- Un microcontrôleur réalisant les fonctions suivantes :
  - Configuration du module Bluetooth
  - Etablissement de la connexion avec le terminal Android
  - L'acquisition des échantillons à une fréquence de 250Hz et leur transmission au format série asynchrone vers le module Bluetooth
- Un amplificateur ECG à 3 électrodes

Le tout est alimenté par pile pour assurer la sécurité du patient.

## 2. Schéma de l'interface ECG

### 2.1 Choix du module Bluetooth

Deux modèles peuvent être utilisés :
**Module hybride Bluetooth™ "FB755AS"**


Le "FB755AS" de FIRMTECH est un module hybride DIL "OEM" subminiature Bluetooth™ Class 1 pré-qualifié faible consommation. Doté d'une antenne externe et d'une puissance d'émission de +12 dBm, il bénéficie d'une portée d'environ 100 mètres en terrain dégagé.

Extrêmement compact (20,5 x 27,7 x 12 mm - sans son antenne), performant et économique, le module hybride "FB755AS" est de part son format DIL (avec pas standard de 2,54 mm) associé à une "bonne" sensibilité (-83 dBm) et à une faible consommation, tout naturellement destiné à être intégré au sein d'applications embarquées les plus diverses. Le module est livré avec une petite antenne hélicoïdale (longueur 3 cm) déportée via un câble d'environ 10 cm relié sur un connecteur uFL.

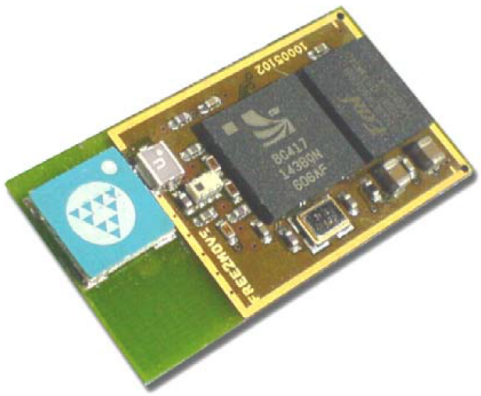
Capable de gérer des communications Bluetooth™ conformément aux spécifications v2, le Firmware de base chargé dans le "FB755AS" lui permet de supporter le protocole de communication **SPP** (Serial Port Profile). Avec ce protocole, toutes les données arrivant sur le port série du "FB755AS" seront automatiquement transférées de façon transparente au périphérique connecté sur la liaison Bluetooth™. La communication étant bien évidemment bidirectionnelle.

Il est commercialisé par LEXTRONIC : http://www.lextronic.fr/P20809-module-hybride-bluetooth-fb755as.html
Ce modèle nécessite une configuration via sa liaison série asynchrone pour être opérationnel dans cette application.
Ce module peut être mis en œuvre directement sur une plaquette de connexions sans soudure et est facilement interchangeable s'il est implanté sur une carte de câblage imprimé.

Il existe également une version avec l'antenne intégrée :


http://www.lextronic.fr/P20807-module-hybride-bluetooth-fb155bc.html

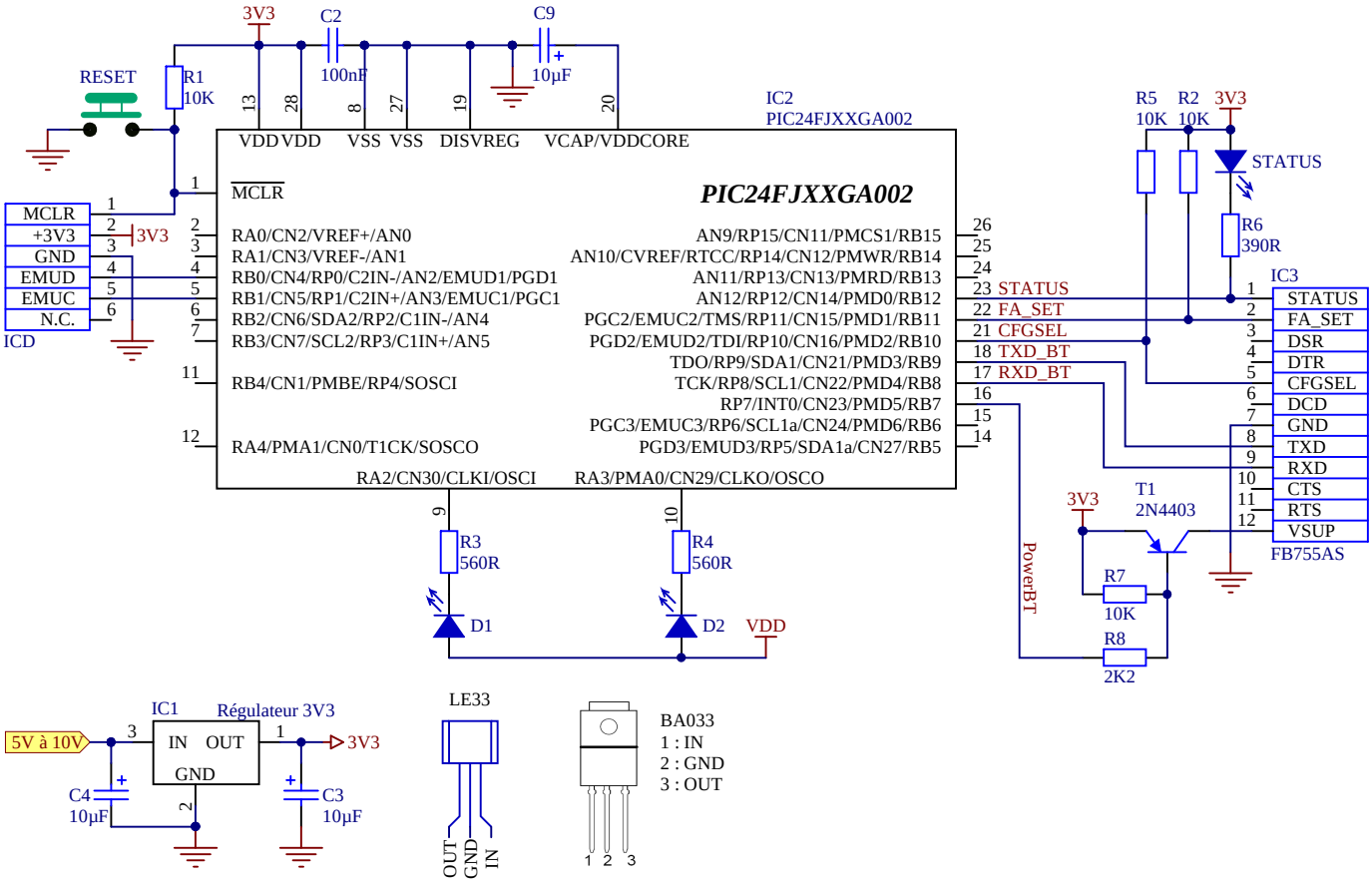
**Module hybride Bluetooth™ "F2M03GLA" de FREE2MOVE**



Le "F2M03GLA" est un module hybride subminiature Bluetooth™ v2+EDR pré-qualifié faible consommation. Doté d'une antenne intégrée et d'une puissance d'émission de +8 dBm, il dispose d'une portée maximale de l'ordre de 250 m en terrain dégagé.
Capable de gérer des communications Bluetooth™ v2+EDR (Enhanced Data Rate) conformément aux spécifications v2.0.E.2 pour les modulations en mode 2 Mbps et 3 Mbps, le "F2M03GLA" dispose d'une connexion série UART (jusqu'à 4 Mbps) ou USB (V2.0 compliant) ainsi que de diverses entrées/sorties numériques et analogiques. Disponible en gamme industrielle (-40°C à +85°C), le module dispose également de fonctionnalités "Piconet" et "Scatternet" en étant capable de supporter jusqu'à 7 "esclaves".
Il est commercialisé par LEXTRONIC : http://www.lextronic.fr/P1107-module-hybride-bluetooth-f2m03gla.html

Ce module est plus compact que le précédent, mais il s'agit d'un macro-composant CMS à souder sur la carte de câblage imprimé. Par ailleurs, il est opérationnel pour cette application dès la mise sous tension, sans nécessiter de paramétrage.

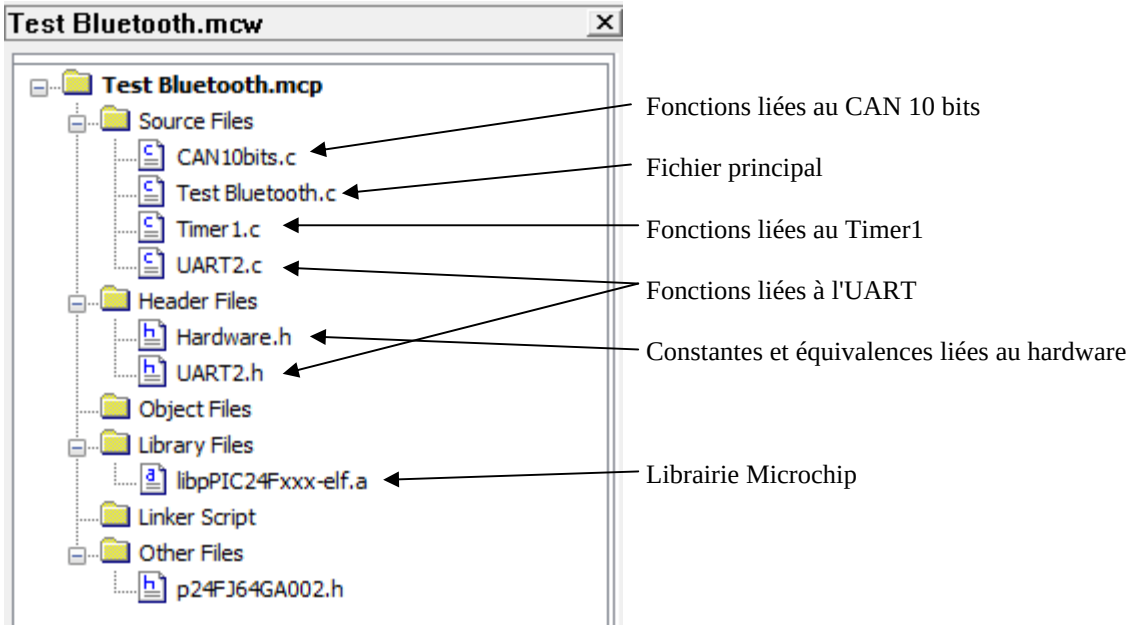
Schéma sur plaquette de test avec le module Firmtech FB755AS




On utilise ici le module FIRMTECH FB755AS car il est adapté à un câblage sur plaquette.
Le signal "Power_BT" pilote la mise sous tension du module Bluetooth pour réduire la consommation en veille et réaliser un "reset hard" souvent nécessaire si la mise sous tension principale est trop progressive.
Les signaux "FA_SET" et "CFGSEL" permettent de réaliser un "reset soft"
Les signaux "TXD_BT" et "RXD_BT" sont les signaux de communication au format série asynchrone.

# 3. Sources du programme PIC24

## 3.1 Projet




## 3.2 "Hardware.h"

```
/*-----------------------------------------------------------------------------
  Nom           :  Hardware.h
  Auteur        :  20/10/2011 - CREMMEL Marcel
  Historique    :  20/10/2011 - Première version
-----------------------------------------------------------------------------*/

#ifndef _HARDWARE_H
#define _HARDWARE_H

/*-----------------------------------------------------------------------------
                    Librairies et fichiers inclus
-----------------------------------------------------------------------------*/
#include <p24Fxxxx.h>

/*-----------------------------------------------------------------------------
                      Définition des types spécifiques
-----------------------------------------------------------------------------*/
#define uint unsigned int

/*------------------
       Divers
--------------------*/
#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
```

```
/*-----------
 Horloge CPU
-------------*/
#define  FCY 5045000 // 4MHz

/*-------------------------------------------------------------------------------
                                  Câblage
-------------------------------------------------------------------------------*/
/*
 On décrit ici le câblage du microcontrôleur
*/

/*-------------------
  Mapping des signaux
---------------------*/

/* Diodes electroluminesentes
 ****************************/
#define LedD1_On  _LATA2=0
#define LedD1_Off _LATA2=1
#define LedD1     _LATA2
#define LedD1_Out _TRISA2=0;
#define LedD2_On  _LATA3=0
#define LedD2_Off _LATA3=1
#define LedD2     _LATA3
#define LedD2_Out _TRISA3=0;

/* UART1 : liaison RS232 vers le PC
 **********************************/
#define MappingTX1 _RP5R=3  // TX1 mappée sur RP5=RB5
#define MappingRX1 _U1RXR=6 // RX1 mappée sur RP6=RB6

/* UART2 : liaison module Bluetooth
 **********************************/
#define MappingTX2 _RP8R=5  // TX2 mappée sur RP8=RB8=RXD_BT
#define MappingRX2 _U2RXR=9 // RX2 mappée sur RP9=RB9=TXD_BT

/* Module Bluetooth FB755AS
 *************************/
#define STATUS      _RB12      // Signal STATUS
#define FA_SET      _RB11      // Signal FA_SET
#define FA_SET_Out  _TRISB11=0 // Signal FA_SET produit par le PIC24
#define CFGSEL      _RB10      // Signal CFGSEL : "1" : mode PC, "0" : mode "cmdes AT"
#define CFGSEL_Out  _TRISB10=0 // Signal CFGSEL produit par le PIC24
#define PowerBT     _RB7       // Signal PowerBT
#define PowerBT_Out _TRISB7=0  // Signal PowerBT produit par le PIC24

/*------------------
   Signaux de test
--------------------*/
#define TEST1     _LATB15
#define TEST1_Out _TRISB15=0
#define TEST2     _LATB14
#define TEST2_Out _TRISB14=0
#define TEST3     _LATB13
#define TEST3_Out _TRISB13=0

#endif
```

Valeur particulière due à une anomalie du PIC24 utilisé. Normalement : 4MHz

Utiles à la mise au point et aux validations

## 3.3 "Test Bluetooth.c"

Il s'agit du fichier principal comportant notamment la fonction "main".
Il comporte également les fonctions de dialogue avec la tablette via le module Bluetooth FB755.

### 3.3.1 "CmdeFB755(char *Cmde)"

Cette fonction s'occupe de transmettre au format série asynchrone via l'UART2 la chaîne donnée en argument.
Elle attend la réponse pour vérifier la prise en compte de la commande.

```
/* Commandes
 ***********/
// Déclencher un "inquiry scan" et un "page scan"
const char BTSCAN[]={'A','T','+','B','T','S','C','A','N',CR,0};
// Affecter un nouveau code PIN; ici "1234"
const char BTKEY[] ={'A','T','+','B','T','K','E','Y','=','1','2','3','4',CR,0};
// Annulation du caractère "ESC" pour éviter sa détection (et éviter un blocage)
const char SETESC[]={'A','T','+','S','E','T','E','S','C',',','0','0',CR,0};
// Annulation du caractère "Debug" pour éviter sa détection (et éviter un blocage)
const char SETDEBUG[]={'A','T','+','S','E','T','D','E','B','U','G',',','0','0',CR,0};
// Réponses
const char ReponseOK[]={CR,LF,'O','K',CR,LF,0};

/* Réponses
 **********/
const char ReponseOK[]={CR,LF,'O','K',CR,LF,0};

/*-------------------------------------------------------
  Function   : void CmdeFB755(*char Cmde)
  Description: Transmettre une commande au FB755 et vérifier la réponse
  Paramètre  : Pointeur vers la chaîne de la commande
  Retour     : Aucun
-------------------------------------------------------*/
void CmdeFB755(char *Cmde)
{
 int Nb;
 putsUART2(Cmde);              // Transmission de la commande
 Delay_mS(500);                // Attendre transmission et réception réponse
 Nb=NbCharBufRX2();            // Nombre de caractères non lus dans le buffer RX2
 ReadStringRXD2(ReponseFB755); // Lecture de la réponse
 Nb=memcmp(ReponseOK,ReponseFB755,Nb); // Réponse attendue = ReponseOK
 if (Nb!=0) Flags.FB755_OK=0;  // Si réponse non conforme : FB755_OK=0
}
```

Les relevés ont montré que 500mS suffisent largement

### 3.3.2 "TestMessageRX_BT"

Cette fonction est appelée régulièrement dans la boucle sans fin de la fonction "main" pour détecter si un message en provenance du module Bluetooth a été reçu et dans ce cas réagir en conséquence.
Les commentaires donnent les détails nécessaires.

- Alimentation du module Bluetooth FB755
- Délai de 2s

```
/*-------------------------------------------------------
  Function   : void TestMessageRX_BT(void)
  Description: Tester si un message BT a été reçu et effectuer le traitement correspondant
  Traitement :
    - aucun caractère reçu ou nombre inférieur à la longueur des messages
      : ne rien faire
    - message reçu = CR-LF-"CONNECT... : affecter BT_Connected à "1"
                                         affecter BT_Scan à "0"
                                         démarrer le CAN
                                         allumer la led D2
    - message reçu = CR-LF-"DISCONNECT... : affecter BT_Connected à "0"
                                         affecter BT_Scan à "1"
                                         arrêter le CAN
                                         éteindre la led D2
    - message reçu = "RUN"-CR-LF :       démarrer le CAN
                                         allumer led D2
    - message reçu = "STOP"-CR-LF :      arrêter le CAN
                                         éteindre led D2
  Paramètre  : Aucun
  Retour     : Aucun
-------------------------------------------------------*/
// Réponses
```

```
const char CONNECT[]   ={CR,LF,'C','O','N','N','E','C','T',0};
const char DISCONNECT[]={CR,LF,'D','I','S','C','O','N','N','E','C','T',0};
const char RUN[]       ={'R','U','N',0};
const char STOP[]      ={'S','T','O','P',0};
void TestMessageRX_BT(void)
{
 int Nb;
 Nb=NbCharBufRX2(); // Nombre de caractères non lus dans le buffer RX2
 if (Nb<5) return;  // Les messages font toujours plus de 5 caractères
 else
 {
  Delay_mS(100);              // Attendre réception réponse complète
  Nb=NbCharBufRX2();          // Nombre de caractères non lus dans le buffer RX2
  ReadStringRXD2(ReponseFB755); // Lecture de la réponse complète
  // Test des messages
  Nb=memcmp(CONNECT,ReponseFB755,9); // Réponse = CR-LF-"CONNECT" ?
  if (Nb==0) {// Oui
              Flags.BT_Connected=1;
              Flags.BT_Scan=0;      // Arrêt écoute Bluetooth
              AD1CON1bits.ADON=1;   // Démarrer le CAN
              LedD2_On;             // Allumer led D2
              return;
             }
  Nb=memcmp(RUN,ReponseFB755,3);    // Réponse = "RUN" ?
  if (Nb==0) {// Oui
              AD1CON1bits.ADON=1;   // Démarrer le CAN
              LedD2_On;             // Allumer led D2
              return;
             }
  Nb=memcmp(STOP,ReponseFB755,3);   // Réponse = "STOP" ?
  if (Nb==0) {// Oui
              AD1CON1bits.ADON=0;   // Stopper le CAN
              LedD2_Off;            // Eteindre led D2
              return;
             }
  Nb=memcmp(DISCONNECT,ReponseFB755,12); // Réponse = CR-LF-"DISCONNECT" ?
  if (Nb==0) {// Oui
              Flags.BT_Connected=0;
              Flags.BT_Scan=1;      // Ecoute Bluetooth en cours
              AD1CON1bits.ADON=0;   // Stopper le CAN
              LedD2_Off;            // Eteindre led D2
              return;
             }
 }
}
```

Pour les tests seulement

### 3.3.3 Fonction "main"

Le traitement commence classiquement par les initialisations des ports d'E/S et des modules intégrés dans le PIC24 (Timer1, UART1, UART2, ADC 10 bits).
Il se poursuit par une séquence d'échanges avec le module Bluetooth pour le configurer conformément aux besoins :

- Délai de 1s de stabilisation des alimentations
- Mise sous tension du module Bluetooth avec CFGSEL="0" (mode cmdes AT) et FASET="1"
- Délai de 1s pour attendre la fin de la transmission du message initial du module Bluetooth
- Calcul du nombre de caractères reçus
- Le module FB755 est présent et opérationnel si ce nombre de caractères est compris entre 30 et 35
- Transmission de la commande "AT+BTKEY=1234" pour affecter le code PIN de la liaison Bluetooth
- Transmission de la commande "AT+SETESC,00" pour supprimer l'identification de la séquence "+++"
- Transmission de la commande "AT+SETDEBUG,00" pour supprimer l'identification du caractère "debug"
  Ces 2 dernières commandes évitent des blocages de la liaison pendant la transmission des échantillons.
- Transmission de la commande "AT+BTSCAN" pour autoriser l'identification Bluetooth du module. La connexion s'établit depuis la tablette Android.

Le programme entre alors dans une boucle sans fin qui appelle régulièrement la fonction "TestMessageRX_BT". Celle-ci interprète les messages du module Bluetooth, notamment celui qui confirme l'établissement de la connexion avec la tablette Android.

```
/*-----------------------------------------------------------------
                Fonction principale
-----------------------------------------------------------------*/
int main(void)
{
 uint Test;
 // Initialisations des variables

 // Initialisation des ports E/S
 AD1PCFG=0xFFFF; // Pour placer tous les ports mixtes en mode "digital"
 TEST1_Out;      // TEST1 en "sortie"
 TEST2_Out;      // TEST2 en "sortie"
 PowerBT=1;      // Module Bluetooth non alimenté
 PowerBT_Out;    // Signal PowerBT produit par le PIC24
 CFGSEL=0;       // Mode "commandes AT"
 CFGSEL_Out;     // Signal CFGSEL produit par le PIC24
 LedD1_Out; LedD2_Out; // Ports leds en sortie
 LedD1_Off; LedD2_Off; // Leds éteintes
 // Initialisation des périphériques intégrés
 _RCDIV=0;    // FRC divisé par 1
 // Timer1 pour disposer de la fonction "Delay_mS"
 InitTimer1(); // Fréquence RTI = 1000Hz
 // UART1 : liaison PC à 9600 bauds
 //iPPSOutput(OUT_PIN_PPS_RP5,OUT_FN_PPS_U1TX);
 MappingTX1; // TX1 mappé sur l'entrée série asynchrone T1IN du MAX202
 MappingRX1; // RX1 mappé sur la sortie série asynchrone R1OUT du MAX202
 // UART1 : 8 bits, pas de parité, 1 bit Stop, 9600 bauds
 OpenUART1(UART_EN|UART_NO_PAR_8BIT|UART_1STOPBIT|UART_BRGH_FOUR,UART_TX_ENABLE,FCY/((long)4*9600)-1);
 // UART2 : liaison module Bluetooth à 9600 bauds
 MappingTX2; // TX2 mappé sur l'entrée série asynchrone RXD_BT du module Bluetooth
 MappingRX2; // RX2 mappé sur la sortie série asynchrone TXD_BT du module Bluetooth
 // UART2 : 8 bits, pas de parité, 1 bit Stop, 9600 bauds
 // Validation interruption RX2
 ConfigIntUART2(UART_RX_INT_EN |UART_RX_INT_PR6);
 OpenUART2(UART_EN|UART_NO_PAR_8BIT|UART_1STOPBIT|UART_BRGH_FOUR,UART_TX_ENABLE,FCY/((long)4*9600)-1);
 // CAN 10 bits
 InitADC10bits(4000,5); // Fréquence d'échantillonnage = 4000Hz, entrée AN5
 // Mise sous tension et reset "hardware" du module Bluetooth
 Delay_mS(1000); // Attendre stabilisation des alims
 PowerBT=0;      // Module Bluetooth alimenté
 AnnulBufRX2();  // Vider le buffer RX2
 Delay_mS(1000); // Attendre fin du message série asynchrone émis par le module
 Test=NbCharBufRX2(); // Nombre de caractères non lus dans le buffer RX2
 if ((Test>30) && (Test<35)) Flags.FB755_OK=1; // Taille nominale du message au reset = 33
 AnnulBufRX2();    // Vider le buffer RX2
 CmdeFB755(BTKEY); // Transmission de la commande "AT+BTKEY=1234", FB755_OK reste à "1" si OK
 if (Flags.FB755_OK) CmdeFB755(SETESC);   // Transmission de la commande "AT+SETESC,00" pour
                                          // supprimer l'identification de la séquence "+++"
 if (Flags.FB755_OK) CmdeFB755(SETDEBUG); // Transmission de la commande "SETDEBUG,00" pour
                                          // supprimer le "debug character"
 if (Flags.FB755_OK) CmdeFB755(BTSCAN);   // Transmission de la commande "AT+BTSCAN"
 if (Flags.FB755_OK) Flags.BT_Scan=1;     // Ecoute Bluetooth en cours
 // Boucle sans fin principale
 while(1)
 {
  TestMessageRX_BT(); // Lecture et traitement de l'éventuel message reçu
 }
}
```

### 3.3.4 Fichier complet

```
/*********************************************************************************
 ***                          Projet ECG sur Android                           ***
 ***  Description :                                                            ***
 ***    Interface ECG-Bluetooth pour PIC24                                     ***
 ***     - un module Bluetooth supportant le protocole série SPP,              ***
 ***     - un microcontrôleur réalisant les fonctions suivantes :              ***
 ***     - configuration du module Bluetooth                                   ***
 ***     - établissement de la connexion avec le terminal Android              ***
 ***     - l'acquisition des échantillons à une fréquence de 250Hz et leur     ***
 ***       transmission au format série asynchrone vers le module Bluetooth    ***
 ***     - un amplificateur ECG à 3 électrodes                                 ***
 ***                                                                           ***
 ***  Auteur  : CREMMEL Marcel                                                 ***
 ***  Version : V1.1                                                           ***
 ***  Date de création :     20/11/2011                                        ***
 ***  Dernière mise à jour : 20/11/2011                                        ***
 *********************************************************************************/

// Option d'usage des librairies Microchip
#define USE_AND_OR // Car la plupart des bits des registres du module UART
                   // sont à "0" au reset -> on ne cite que ceux qui changent
                   // d'état.

/*-------------------------------------------------------------------------------
                    Librairies et fichiers inclus
-------------------------------------------------------------------------------*/
#include "Hardware.h"
#include "UART2_ECG.h"
#include <uart.h>
#include <string.h>
//#include <pps.h>

/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 CONFIG 1 :
 **********
   JTAG "off" pour disposer des broches affectées à JTAG
   GCP "off" et GWRP "off" : mémoire non protégée
   BKBUG_OFF "off" : Background Debugger "off"
   COE "off" : Clip-on Emulation mode "off"
   FWDTEN "off" : Watchdog inhibé
   ICS_PGx1 : EMUC/EMUD share PGC1/PGD1
*/
_CONFIG1(JTAGEN_OFF&GCP_OFF&GWRP_OFF&BKBUG_OFF&COE_OFF&FWDTEN_OFF&ICS_PGx1)

/* CONFIG 2 :
 **********
   IESO_OFF     : Two Speed Start-up "off"
   FNOSC_FRCDIV : Fast RC oscillator and divide
   POSCMOD_NONE : Primary disabled
   FCKSM_CSECMD : Only clock switching enabled
   OSCIOFNC_ON  : OSCO/RA3 function : RA3
   IOL1WAY_OFF  : Unlimited Writes To RP Registers
   I2C1SEL_PRI  : Use Primary I2C1 pins
*/
_CONFIG2(IESO_OFF&FNOSC_FRCDIV&FCKSM_CSECMD&OSCIOFNC_ON&IOL1WAY_OFF&I2C1SEL_PRI)
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

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

Fonctions d'interruption, de lecture et d'écriture

Fonctions d'initialisations de Microchip

Pour utiliser les fonctions de "mapping" de Microchip

Pour la fonctions "memcmp"

```
/* Indicateurs de fonctionnement
 ******************************/
struct
{
 unsigned FB755_OK     : 1; // Module Bluetooth présent et opérationnel
 unsigned BT_Scan      : 1; // Scan Bluetooth en cours
 unsigned BT_Connected : 1; // Connexion Bluetooth établie
 unsigned ECG_Run      : 1; // Acquisition ECG Run (1) ou Stop(0)
 unsigned unused       : 12;
}Flags;

char ReponseFB755[32];
```

Pour mémoriser les réponses du module Bluetooth

```
/*-----------------------------------------------------------------------------
            Déclarations des constantes mémorisées en flash
-----------------------------------------------------------------------------*/

/*-------------------------------------------------------------
                Fonctions de gestion du FB755AS
-------------------------------------------------------------*/

/* Commandes
 ***********/
// Déclencher un "inquiry scan" et un "page scan"
const char BTSCAN[]={'A','T','+','B','T','S','C','A','N',CR,0};
// Affecter un nouveau code PIN; ici "1234"
const char BTKEY[] ={'A','T','+','B','T','K','E','Y','=','1','2','3','4',CR,0};
// Annulation du caractère "ESC" pour éviter sa détection (et éviter un blocage)
const char SETESC[]={'A','T','+','S','E','T','E','S','C',',','0','0',CR,0};
// Annulation du caractère "Debug" pour éviter sa détection (et éviter un blocage)
const char SETDEBUG[]={'A','T','+','S','E','T','D','E','B','U','G',',','0','0',CR,0};
// Réponses
const char ReponseOK[]={CR,LF,'O','K',CR,LF,0};

/* Réponses
 **********/
const char ReponseOK[]={CR,LF,'O','K',CR,LF,0};

/*-------------------------------------------------------
  Function   : void CmdeFB755(*char Cmde)
  Description: Transmettre une commande au FB755 et vérifier la réponse
  Paramètre  : Pointeur vers la chaîne de la commande
  Retour     : Aucun
-------------------------------------------------------------*/
void CmdeFB755(char *Cmde)
{
 int Nb;
 putsUART2(Cmde);              // Transmission de la commande
 Delay_mS(500);                // Attendre transmission et réception réponse
 Nb=NbCharBufRX2();            // Nombre de caractères non lus dans le buffer RX2
 ReadStringRXD2(ReponseFB755); // Lecture de la réponse
 Nb=memcmp(ReponseOK,ReponseFB755,Nb); // Réponse attendue = ReponseOK
 if (Nb!=0) Flags.FB755_OK=0;  // Si réponse non conforme : FB755_OK=0
}

/*-------------------------------------------------------
  Function   : void TestMessageRX_BT(void)
  Description: Tester si un message BT a été reçu et effectuer le traitement correspondant
  Traitement :
    - aucun caractère reçu ou nombre inférieur à la longueur des messages
      : ne rien faire
    - message reçu = CR-LF-"CONNECT... : affecter BT_Connected à "1"
                                         affecter BT_Scan à "0"
                                         démarrer le CAN
                                         allumer la led D2
    - message reçu = CR-LF-"DISCONNECT... : affecter BT_Connected à "0"
                                         affecter BT_Scan à "1"
                                         arrêter le CAN
                                         éteindre la led D2
    - message reçu = "RUN"-CR-LF :       démarrer le CAN
                                         allumer led D2
    - message reçu = "STOP"-CR-LF :      arrêter le CAN
```

```
                                        éteindre led D2
  Paramètre  : Aucun
  Retour     : Aucun
-------------------------------------------------------------*/
// Réponses
const char CONNECT[]   ={CR,LF,'C','O','N','N','E','C','T',0};
const char DISCONNECT[]={CR,LF,'D','I','S','C','O','N','N','E','C','T',0};
const char RUN[]       ={'R','U','N',0};
const char STOP[]      ={'S','T','O','P',0};
void TestMessageRX_BT(void)
{
 int Nb;
 Nb=NbCharBufRX2();            // Nombre de caractères non lus dans le buffer RX2
 if (Nb<5) return;
 else
 {
  Delay_mS(100);                // Attendre réception réponse complète
  Nb=NbCharBufRX2();            // Nombre de caractères non lus dans le buffer RX2
  ReadStringRXD2(ReponseFB755); // Lecture de la réponse
  // Test des messages
  Nb=memcmp(CONNECT,ReponseFB755,9); // Réponse = CR-LF-"CONNECT" ?
  if (Nb==0) {// Oui
              Flags.BT_Connected=1;
              Flags.BT_Scan=0;      // Arrêt écoute Bluetooth
              AD1CON1bits.ADON=1;   // Démarrer le CAN
              LedD2_On;             // Allumer led D2
              return;
             }
  Nb=memcmp(RUN,ReponseFB755,3);    // Réponse = "RUN" ?
  if (Nb==0) {// Oui
              AD1CON1bits.ADON=1;   // Démarrer le CAN
              LedD2_On;             // Allumer led D2
              return;
             }
  Nb=memcmp(STOP,ReponseFB755,3);   // Réponse = "STOP" ?
  if (Nb==0) {// Oui
              AD1CON1bits.ADON=0;   // Stopper le CAN
              LedD2_Off;            // Eteindre led D2
              return;
             }
  Nb=memcmp(DISCONNECT,ReponseFB755,12); // Réponse = CR-LF-"DISCONNECT" ?
  if (Nb==0) {// Oui
              Flags.BT_Connected=0;
              Flags.BT_Scan=1;      // Ecoute Bluetooth en cours
              AD1CON1bits.ADON=0;   // Stopper le CAN
              LedD2_Off;            // Eteindre led D2
              return;
             }
 }
}

/*-----------------------------------------------------------------
               Fonction principale
-----------------------------------------------------------------*/
int main(void)
{
 uint Test;
 // Initialisations des variables

 // Initialisation des ports E/S
 AD1PCFG=0xFFFF; // Pour placer tous les ports mixtes en mode "digital"
 TEST1_Out;      // TEST1 en "sortie"
 TEST2_Out;      // TEST2 en "sortie"
 PowerBT=1;      // Module Bluetooth non alimenté
 PowerBT_Out;    // Signal PowerBT produit par le PIC24
 CFGSEL=0;       // Mode "commandes AT"
 CFGSEL_Out;     // Signal CFGSEL produit par le PIC24
 LedD1_Out; LedD2_Out; // Ports leds en sortie
 LedD1_Off; LedD2_Off; // Leds éteintes
 // Initialisation des périphériques intégrés
 _RCDIV=0;    // FRC divisé par 1
 // Timer1 pour disposer de la fonction "Delay_mS"
```

Fonction de "mapping" de Microchip

```
 InitTimer1(); // Fréquence RTI = 1000Hz
 // UART1 : liaison PC à 9600 bauds
 //iPPSOutput(OUT_PIN_PPS_RP5,OUT_FN_PPS_U1TX);
 MappingTX1; // TX1 mappé sur l'entrée série asynchrone T1IN du MAX202
 MappingRX1; // RX1 mappé sur la sortie série asynchrone R1OUT du MAX202
 // UART1 : 8 bits, pas de parité, 1 bit Stop, 9600 bauds
 OpenUART1(UART_EN|UART_NO_PAR_8BIT|UART_1STOPBIT|UART_BRGH_FOUR,UART_TX_ENABLE,FCY/((long)4*9600)-1);
 // UART2 : liaison module Bluetooth à 9600 bauds
 MappingTX2; // TX2 mappé sur l'entrée série asynchrone RXD_BT du module Bluetooth
 MappingRX2; // RX2 mappé sur la sortie série asynchrone TXD_BT du module Bluetooth
 // UART2 : 8 bits, pas de parité, 1 bit Stop, 9600 bauds
 // Validation interruption RX2
 ConfigIntUART2(UART_RX_INT_EN |UART_RX_INT_PR6);
 OpenUART2(UART_EN|UART_NO_PAR_8BIT|UART_1STOPBIT|UART_BRGH_FOUR,UART_TX_ENABLE,FCY/((long)4*9600)-1);
 // CAN 10 bits
 InitADC10bits(4000,5); // Fréquence d'échantillonnage = 4000Hz, entrée AN5
 // Mise sous tension et reset "hardware" du module Bluetooth
 Delay_mS(2000); // Attendre stabilisation des alims
 PowerBT=0;      // Module Bluetooth alimenté
 AnnulBufRX2();  // Vider le buffer RX2
 Delay_mS(1000); // Attendre fin du message série asynchrone émis par le module
 Test=NbCharBufRX2(); // Nombre de caractères non lus dans le buffer RX2
 if ((Test>30) && (Test<35)) Flags.FB755_OK=1; // Taille nominale du message au reset = 33
 AnnulBufRX2();    // Vider le buffer RX2
 CmdeFB755(BTKEY); // Transmission de la commande "AT+BTKEY=1234", FB755_OK reste à "1" si OK
 if (Flags.FB755_OK) CmdeFB755(SETESC);   // Transmission de la commande "AT+SETESC,00" pour
                                          // supprimer l'identification de la séquence "+++"
 if (Flags.FB755_OK) CmdeFB755(SETDEBUG); // Transmission de la commande "SETDEBUG,00" pour
                                          // supprimer le "debug character"
 if (Flags.FB755_OK) CmdeFB755(BTSCAN);   // Transmission de la commande "AT+BTSCAN"
 if (Flags.FB755_OK) Flags.BT_Scan=1;     // Ecoute Bluetooth en cours
 // Boucle sans fin principale
 while(1)
 {
  TestMessageRX_BT(); // Lecture et traitement de l'éventuel message reçu
 }
}
```

Pour "effacer" les éventuels caractères parasites reçus

Commentaires :

- De la librairie <uart.h> de Microchip, seules sont utilisées les fonctions d'initialisation "*OpenUARTx*", "*ConfigIntUARTx*" et les fonctions de transmission "*putsUARTx(unsigned int *buffer)*" et "*WriteUARTx(unsigned int data)*"
- Les fonctions d'interruption à la réception et de lecture de l'UART2 sont déclarées dans le fichier "UART2.c"
- Les commandes "AT+SETESC,00" et "AT+SEDEBUG,00" vers le module FB755 annulent respectivement les séquences "Esc" et le caractère "debug" pour le programme du module ne perde pas de temps à les détecter et n'y réagisse pas (pas de blocage).
- La boucle sans fin principale ne comporte qu'une seule fonction à l'écoute des éventuels messages reçus du module Bluetooth (commandes "Run" et "Stop" ou connexion et déconnexion Bluetooth)

## 3.4 "Timer1.c"

Le "Timer1" est configuré pour provoquer une interruption toutes les mS.
La fonction d'interruption fait clignoter la led D1 et est utilisée par la fonction "Delay_mS" pour produire des délais logiciels.

```
/*****************************************************************************
 ***                 Fonctions liées au Timer1 d'un PIC24                  ***
 ***     - Interruptions "temps réel"                                      ***
 ***     - fonctions de délai logiciel                                     ***
 ***     - clignotement led D1                                             ***
 ***  Auteur  : CREMMEL Marcel                                             ***
 ***  Version : V1.0                                                       ***
 ***  Date de création     : 12/10/2011                                    ***
 ***  Dernière mise à jour : 12/10/2011                                    ***
 *****************************************************************************/

/*----------------------------------------------------------------------------
                   Librairies et fichiers inclus
----------------------------------------------------------------------------*/
#include "Hardware.h"
```

```
/*-------------------------------------------------------------------------
                  Déclarations des variables globales en RAM
-------------------------------------------------------------------------*/
uint CptTimer1;       // Compteur interruptions Timer 1
uint CptTimer1Cligno; // Pour faire clignoter la led D1

/*-------------------------------------------------------------------------
                      Fonctions en mémoire flash
-------------------------------------------------------------------------*/

/*-----------------------------------------------------
  Function   : void InitTimer1(void)
  Description: Initialisation du Timer 1 (Timer A) pour créer des délais logiciels
               et une RTI de fréquence 1000Hz
     - interruptions validées
-----------------------------------------------------*/
void InitTimer1(uint Frti)
{
 T1CON=0x0000;    // Fonctionnement continu, horloge=FCY
 PR1=FCY/1000-1;  // Pour obtenir une fréquence d'interruption de 1000Hz
 T1CONbits.TON=1; // Validation Timer 1
 _T1IF=0;         // Raz indicateur T1IF du Timer 1
 _T1IE=1;         // Validation interruption Timer 1
}

/*-----------------------------------------------------
  Function   : void Delay(uint16 Delai)
  Description: Délai "logiciel"
               Utilise le Timer1
  Paramètre  : Delai : résolution 1mS
                       délai maximum : 65535*1mS=65,5s
-----------------------------------------------------*/
void Delay_mS(uint Delai)
{
 uint Temps_debut,Temps_actuel;
 Temps_debut=CptTimer1;     // Echantillonnage temps début
 do Temps_actuel=CptTimer1; // Echantillonnage temps actuel
 while ((Temps_actuel-Temps_debut)<=Delai);
}

/*-------------------------------------------------------------------
  Function   : void _ISR _T1Interrupt (void)
  Descrition : Fréquence d'activation = 1kHz
  Traitement :
   - incrémentation du compteur "CptTimer1"
   - incrémentation du compteur "CptTimer1Cligno" modulo 500 pour faire
     clignoter la led D1 au rythme de 2Hz
-------------------------------------------------------------------*/
void _ISR _T1Interrupt (void)
{
 TEST2=1;
 _T1IF=0;               // Acquittement interruption
 CptTimer1++;
 CptTimer1Cligno++;
 if (CptTimer1Cligno==500)
 {
  CptTimer1Cligno=0;
  LedD1=!LedD1;       // D1 clignote
 }
 TEST2=0;
}
```

Pour contrôler la fréquence des interruptions. Peut être supprimé

Pour contrôler la fréquence des interruptions. Peut être supprimé

## 3.5 "UART2_ECG.c"

La configuration du module et les fonctions de transmission sont celles de la librairie fournie par Microchip.
Par contre, des caractères peuvent être reçus du module Bluetooth à tout moment. Il est donc nécessaire d'utiliser les interruptions et un tampon circulaire. Ces tâches ne sont pas proposées par la librairie Microchip, il faut donc les écrire spécifiquement pour cette application.

```
     /*********************************************************************
      ***                Projet ECG sur Android                         ***
      ***              Fonctions liées à l'UART2                        ***
      ***                                                               ***
      ***  Fichier "source" : UART2.c                                   ***
      ***  Auteur : CREMMEL Marcel                                      ***
      ***  Date   : 28/04/2008                                          ***
      *********************************************************************/

#include "UART2_ECG.h"

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

/* Buffers RX et TX pour l'UART
*******************************/
#define RX2_BufSize 1024        // Taille du Buffer_RX2
char Buffer_RX2[RX2_BufSize];   // Buffer circulaire de réception
char *ptrRX2_WRdata=Buffer_RX2; // Pointeur d'écriture ds Buffer_RX2
char *ptrRX2_RDdata=Buffer_RX2; // Pointeur de lecture ds Buffer_RX2

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

/*-----------------------------------------------------------------
  Nom          : int ReadRXD2(char *c)
  Description  : Lecture d'un octet dans le buffer de réception
  Arguments       : c : pointeur vers le caractère reçu
  Valeur renvoyée : 0 si aucun caractère reçu
                    1 si caractère reçu
  Traitements :
    - pointeur circulaire ptrRX2_RDdata incrémenté
-----------------------------------------------------------------*/
int ReadRXD2(char *c)
{
 char c_bis;
 if (ptrRX2_RDdata==ptrRX2_WRdata) return(0); // Pas de caractère reçu
 else
 {
  *c=*ptrRX2_RDdata++; // Affecter "c" avec le caractère reçu puis
                       // incrémenter le pointeur ptrRX2_RDdata
  c_bis=*c;
  // Pointeur circulaire
  if (ptrRX2_RDdata==Buffer_RX2+RX2_BufSize) ptrRX2_RDdata=Buffer_RX2;
  return(1);
 }
}
```

Taille adaptée à la plus grande longueur des messages reçus.

```
/*-----------------------------------------------------------------
  Nom          : int ReadStringRXD2(char *String)
  Description  : Lecture d'une chaîne de caractères dans le buffer de réception
  Arguments       : String : pointeur vers la chaîne à affecter
  Valeur renvoyée : 0 si aucun caractère reçu ou chaîne incomplète
                    1 si chaîne complète reçue (séquence terminale = CR-LF)
-----------------------------------------------------------------*/
int ReadStringRXD2(char *String)
{
 char c;
 char *ptrWR;
 ptrWR=ptrRX2_WRdata-1;
 if (ptrWR<Buffer_RX2) ptrWR=Buffer_RX2+RX2_BufSize-1;
 if (*ptrWR!=LF) return (0);
 do
 {
  ReadRXD2(&c);
  *String++=c;
 }
 while (ptrRX2_RDdata!=ptrRX2_WRdata);
 return (1);
}

/*-----------------------------------------------------------------
  Nom          : uint NbCharBufRX2(void)
  Description  : Calcul du nombre de caractères non lus dans le buffer RX2
  Arguments       : aucun
  Valeur renvoyée : nombre de caractères non lus dans le buffer RX2
-----------------------------------------------------------------*/
uint NbCharBufRX2(void)
{
 uint Nb;
 Nb=ptrRX2_WRdata-ptrRX2_RDdata; // Ecart entre les pointeurs
 if (Nb>RX2_BufSize) Nb=Nb-RX2_BufSize;
 return (Nb);
}

/*-----------------------------------------------------------------
  Nom         : void AnnulBufRX2(void)
  Description : Vider le buffer RX2
  Arguments   : aucun
  Retour      : nombre de caractères non lus dans le buffer RX2
-----------------------------------------------------------------*/
void AnnulBufRX2(void)
{
 ptrRX2_RDdata=ptrRX2_WRdata;
}

/*-----------------------------------------------------------------
  Nom          : void _ISR _U2RXInterrupt (void)
  Description :  Programme d'interruption déclenché à la réception
                 d'un caractère
  Arguments    : aucun
  Valeur renvoyée : aucune
-----------------------------------------------------------------*/
void _ISR _U2RXInterrupt (void)
{
 char c;
 IFS1bits.U2RXIF=0;      // Clear interrupt flag
 while (U2STAbits.URXDA) // Boucle pour lire ts les car. ds le buf. de l'UART
 {
  c=U2RXREG;
  *ptrRX2_WRdata++=c; // Lecture du caractère suivant et incrémentation
                      // du pointeur circulaire de lecture
  if (ptrRX2_WRdata==Buffer_RX2+RX2_BufSize) ptrRX2_WRdata=Buffer_RX2;
 }
}
```

Retour prématuré si le dernier caractère reçu n'est pas "LF"

## 3.6 "CAN10bits.c"

Le module "CAN 10 bits" du PIC24 est utilisé pour numériser le signal analogique ECG. Celui-ci a les caractéristiques suivantes :

- Valeur moyenne = VDD/2 = 1,65V
- Amplitude max = 1,65V

Le CAN 10 bits est configuré comme suit :

- VREF+ = VDD = 3,3V et VREF- = VSS = 0V
- Autoconversion
- Une seule entrée analogique
- Fréquence d'échantillonnage = 4000Hz
- Fréquence des échantillons moyens transmis au module Bluetooth = 4000/16 = 250Hz

La conversion est initialement bloquée. Elle ne sera libérée qu'après la connexion Bluetooth avec la tablette Android.

```
/************************************************************************
 ***             Fonctions de gestion du CAN 10 bits d'un PIC24          ***
 ***                                                                     ***
 ***  Auteur  : CREMMEL Marcel                                           ***
 ***  Version : V1.0                                                     ***
 ***  Date de création     : 12/10/2011                                  ***
 ***  Dernière mise à jour : 12/10/2011                                  ***
 ************************************************************************/

/*-----------------------------------------------------------------------
                   Librairies et fichiers inclus
-----------------------------------------------------------------------*/
#include "Hardware.h"
#include <UART.h>

/*-----------------------------------------------------------------------
                  Déclarations des variables globales en RAM
-----------------------------------------------------------------------*/
// Variables pour le calcul de la valeur moyenne sur 16 échantillons
unsigned int CptSample;   // Compteur des échantillons modulo 16
unsigned int TotalSample; // Pour sommer les  échantillons
unsigned int MoySample;   // Moyenne de 16 échantillons

/*-----------------------------------------------------------------------
                    Déclarations des variables externes
-----------------------------------------------------------------------*/

/* Indicateurs de fonctionnement
 ******************************/
extern struct
{
 unsigned FB755_OK     : 1; // Module Bluetooth présent et opérationnel
 unsigned BT_Scan      : 1; // Scan Bluetooth en cours
 unsigned BT_Connected : 1; // Connexion Bluetooth établie
 unsigned ECG_Run      : 1; // Acquisition ECG Run (1) ou Stop(0)
 unsigned unused       : 12;
}Flags;

/*-----------------------------------------------------------------------
                     Fonctions en mémoire flash
-----------------------------------------------------------------------*/

/*****************************************************************
  Function:     void InitADC10(int FE, int Input)
  Description : Initialisation de l'ADC 10 bits :
                   Auto-conversion
                   Interruption EOC toutes les 16 conversions
                   Durée cycle Tad = FCY/(40*FE)
                   Durée échantillonnage = 28*Tad
                   Durée échantillonnage+conversion = 40*Tad
  Arguments : FE    : fréquence d'échantillonnage en Hz
              Input : entrée à convertir
*****************************************************************/
void InitADC10bits(int FE, int Input)
{
 AD1PCFG&=~(1<<Input); // L'entrée "Input" est configurée en entrée analogique
 AD1CON1bits.ADSIDL=0; // Le module est arrêté en mode "Idle"
 AD1CON1bits.FORM=0;   // Conversion en binaire naturel
```

```
 AD1CON1bits.SSRC=7;   // Autoconversion
 AD1CON1bits.ASAM=1;   // L'échantillonnage commence immédiatement à la fin de la conversion.
 AD1CON2bits.VCFG=0;   // VREF+=VDD et VREF-=VSS
 AD1CON2bits.CSCNA=0;  // Pas de scan des entrées analogiques
 AD1CON2bits.BUFS=0;   // Affectation des buffers 0 à 7 par le CAN
 AD1CON2bits.SMPI=0;   // Interruption à chaque conversion
 AD1CON2bits.BUFM=0;   // Buffer = 1 mot de 16 bits
 AD1CON2bits.ALTS=0;   // MUXA sélectionné
 AD1CON3bits.ADRC=0;   // Horloge = FCY
 AD1CON3bits.SAMC=28;  // Durée d'échantillonnage: 28*Tad
 AD1CON3bits.ADCS=(FCY/FE)/40-1; // Durée de Tad: (N+1)*TCY. Conversion en 12 cycles Tad
 AD1CHSbits.CH0NA=0;     // MUXA : entrée - canal 0 = VR-
 AD1CHSbits.CH0SA=Input; // MUXA : sélection de l'entrée ANx (

 _AD1IF=0;             // Raz de l'indicateur d'interruption
 _AD1IE=1;             // Validation des interruptions
 //AD1CON1bits.ADON=1;   // Démarrer le CAN
}

/*****************************************************************
  Function    : void _ISR _ADCInterrupt (void)
  Description : Programme d'interruption déclenché à chaque EOC du CAN 10 bits
  Résultat des conversions :
      - de ANx en binaire utilisé pour calculer "MoySample"
*****************************************************************/
void _ISR _ADC1Interrupt (void)
{
 unsigned int Resultat;
 TEST1=1;
 _AD1IF=0;             // Raz indicateur interruption
 Resultat=ADC1BUF0;    // Résultat de la conversion A/N
 TotalSample+=Resultat; // Addition des échantillons successifs
 CptSample++;          // Compter les échantillons
 if (CptSample==16)    // 16 échantillons successifs additionnés ?
 {// Oui : calcul moyenne des 16 derniers échantillons
  CptSample=0;         // Raz du compteurs
  MoySample=TotalSample>>4; // Calcul de la valeur moyenne
  TotalSample=0;            // Raz de la somme pour calcul suivant
  if (Flags.BT_Connected)   // Connexion Bluetooth établie ?
      WriteUART2((char)(MoySample>>2)); // Transmission de l'échantillon moyen
 }
 TEST1=0;
}
```

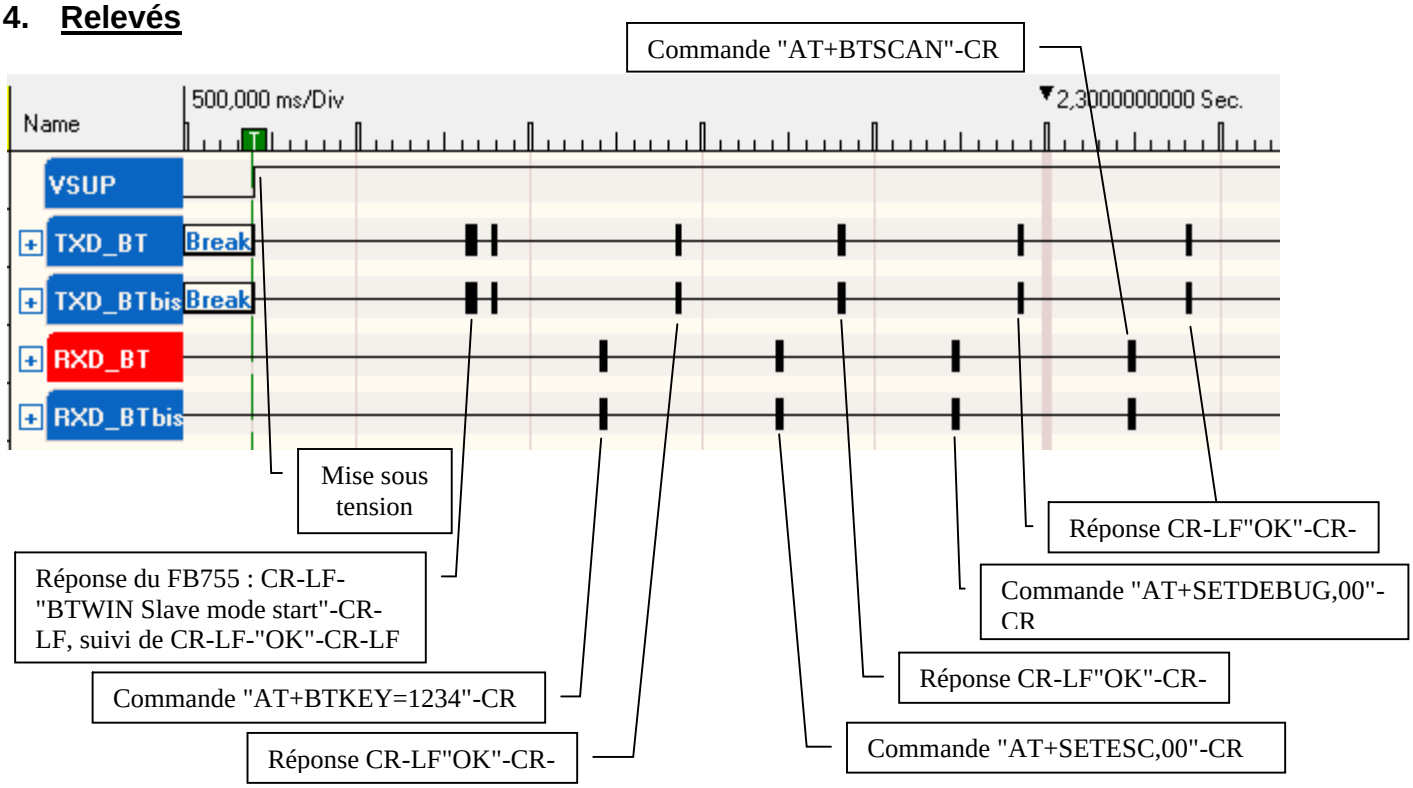
Attention : (FCY/FE)/40 doit donner un résultat entier pour obtenir une fréquence d'échantillonnage précise

Le CAN sera démarré à l'établissement de la connexion Bluetooth avec la tablette

Pour contrôler la fréquence d'échantillonnage à l'oscilloscope

Pour contrôler la fréquence d'échantillonnage à l'oscilloscope

## 4. Relevés




AJOUTER RELEVE AVEC LE DELAI DE CONNEXION ET LES ECHANTILLONS A 250Hz