

Ingénierie en microcontrôleur

Projet : Montre connectée



Encadré par : M. DELEBARRE

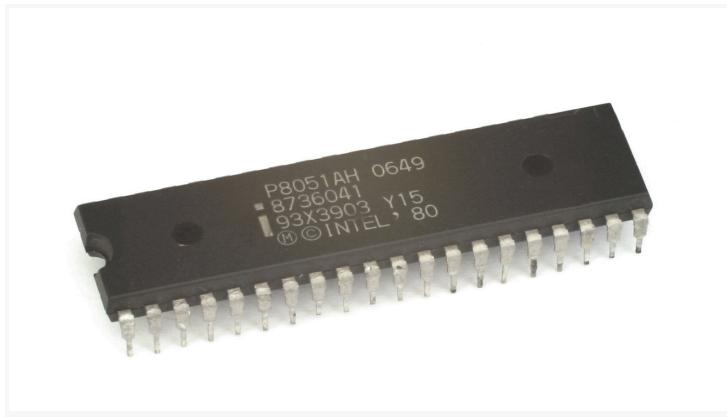
Écrit par : TOURE Mouhamadou
PAWLOWSKI Jimmy
TAKFA Abdelfatah

Introduction:

Depuis de nombreuses années, il existe des micro-ordinateurs de poignet. En fait, la première montre avec un calculateur intégré a été développée en 1970 par la société Hamilton Watch Company. Aujourd'hui, il existe de nombreux dérivés de ces montres, y compris des modèles haut de gamme qui peuvent lire les notifications du téléphone, prendre des photos et surveiller la fréquence cardiaque. Dans cette étude, nous allons nous intéresser à la programmation d'un système plus simple, mais toujours connecté à un smartphone. Cette montre sera équipée d'un processeur de type 8051 et devra être capable de compter les pas, de localiser son porteur et de chronométrier ses efforts physiques. En parallèle, elle devra également transmettre ces informations au smartphone pour être utilisées dans diverses applications.

I. Les composants

1) Microprocesseur 8051



Le microcontrôleur 8051 est un circuit intégré programmable qui est largement utilisé pour les applications de contrôle et d'automatisation. Il a été développé par Intel dans les années 1980 et est toujours populaire aujourd'hui. Le microcontrôleur 8051 est doté d'une architecture à 8 bits, d'une horloge interne, de plusieurs ports d'entrée/sortie, de timers et de compteurs, ainsi que d'une unité centrale de traitement (CPU) puissante qui prend en charge de nombreux types d'instructions. Il est également équipé d'une mémoire flash reprogrammable qui permet de modifier le code du programme en cours d'exécution. Le microcontrôleur 8051 est utilisé dans une large gamme d'applications, notamment l'automobile, les systèmes de contrôle

d'accès, les systèmes de sécurité, les dispositifs médicaux, les appareils ménagers, les télécommunications et les systèmes de contrôle industriel.

2) ADXL345



L'ADXL345 est un capteur d'accélération à trois axes développé par Analog Devices, Inc. Il mesure les forces d'accélération dans les trois axes X, Y et Z et fournit des données numériques précises à un microcontrôleur. Le capteur est équipé d'un large éventail de gammes de mesure, de résolutions et de fréquences d'échantillonnage, ce qui le rend adapté à une variété d'applications. Il dispose également d'une interface numérique I2C ou SPI pour la communication avec d'autres composants électroniques. Dans notre application, nous optons pour l'I2C. Le capteur ADXL345 est utilisé dans des applications telles que les systèmes de stabilisation de caméra, les appareils portables, les systèmes de navigation et les consoles de jeux. Il est apprécié pour sa petite taille, sa consommation d'énergie faible et sa haute précision.

3) Bluetooth BLE112



Ce module Bluetooth va permettre la communication entre la montre et le téléphone. La communication entre le microcontrôleur peut se faire par SPI ou par UART.

4) Ecran LCD



L'afficheur LCD va envoyer l'information à l'utilisateur, heure, nombre de pas, chronomètre... L'afficheur reçoit des données logiques, sur 4 ou 8 bits.

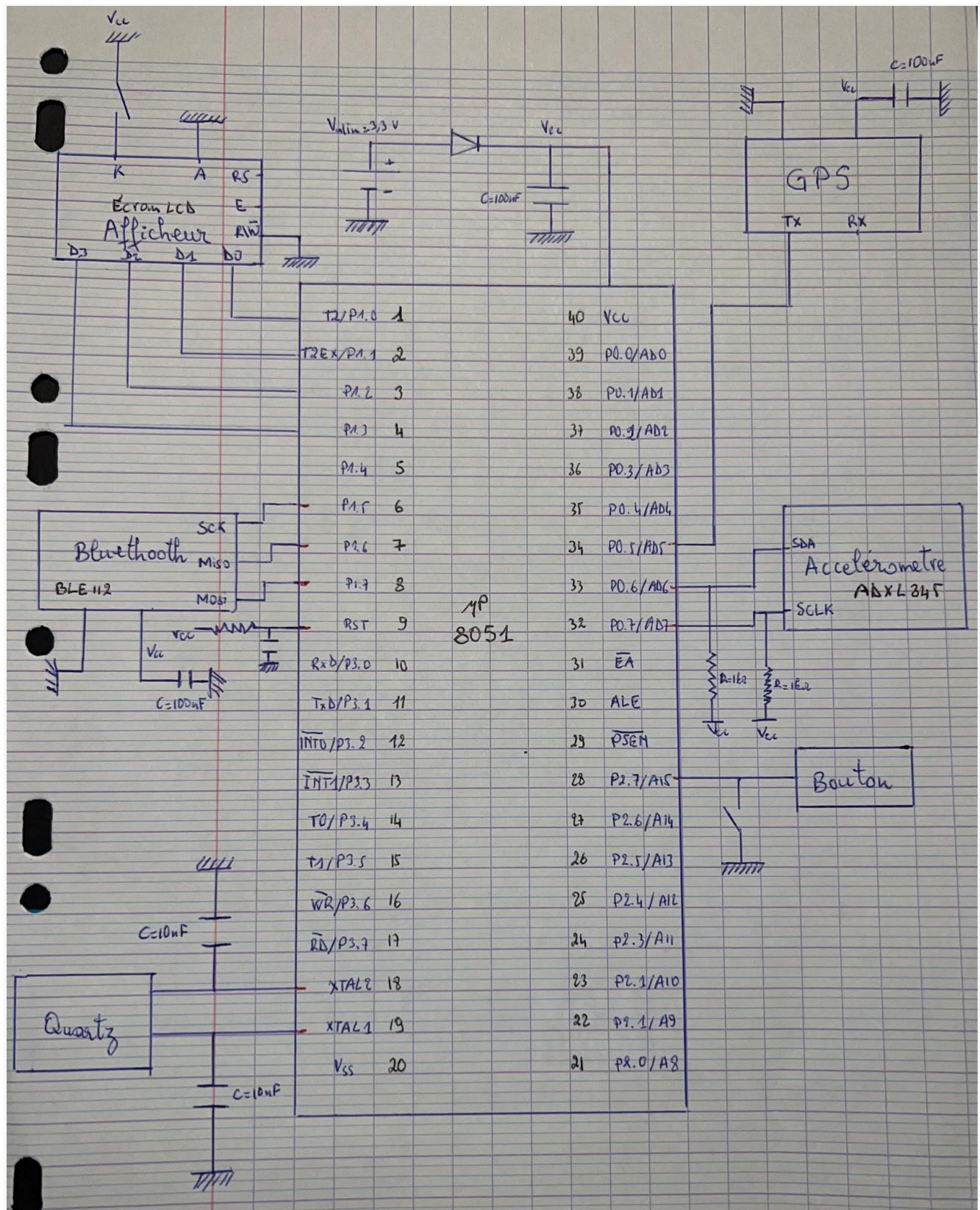
5) Le GPS



Le GPS dans une montre connectée est un système de positionnement global qui permet de suivre les activités physiques, de naviguer dans des zones inconnues et de surveiller la santé. Il offre des fonctionnalités variées et est un outil très pratique pour les utilisateurs qui cherchent à améliorer leur condition physique et leur santé.

II. Câblage

Voici le câblage proposé pour notre montre connectée :



III. Programmation

1. Initialisation du microcontrôleur:

1.1 Initialisation 8051:

```
OGR 0H: MOV T2XCLK, #00H // 12 oscillations = 1 top horloge  
MOV CKCON, 0 // Choix de l'horloge externe  
MOV TR2, 1 // Activation du Timer 2  
MOV TMR2RL, #B1E0H // -20000 en complément à 2
```

1.2 Déclaration des ports d'entrée/sortie:

XTAL1	EQU	P0.2
XTAL2	EQU	P0.3
TxD	EQU	P0.4
RxD	EQU	P0.5
SDA	EQU	P0.6
SCLK	EQU	P0.7
RS	EQU	P1.4
SCK	EQU	P1.5
MISO	EQU	P1.6
MOSI	EQU	P1.7

2. Programme principal (affichage)

- Envoyer à l'afficheur les caractères à afficher en fonction du mode.
- Choisir une fréquence de rafraîchissement de l'écran pas trop élevée pour éviter la persistance rétinienne.
- L'afficheur reçoit des caractères ASCII donc il faut convertir les caractères à afficher en code ASCII.

2.1 Sous-routine de conversion d'un nombre code ASCII sa représentation BCD:

- Réalise la division du nombre par 10 pour le convertir en BCD
- Ajout de 0x30 aux unités et aux dizaines pour obtenir leur code ASCII respectif.
- Retourne le code ASCII des unités dans R2 et celui des dizaines dans R1.

```
B_TO_BCD: MOV B, #0AH
              DIV A, B
              ADD A, #30H
              MOV R1, A
              MOV A, B
              ADD A, #30H
              MOV R2, A
              RET
```

2.2 Sous-routine d'affichage de l'heure:

Transmet les codes ASCII des dizaines et des unités de l'heure, des minutes, des secondes et des centièmes de seconde à l'afficheur et les affiche sur la ligne supérieure de l'écran au format HH:MM:SS

```
AFFICHE_HEURE: MOV A, HEURE
                   LCALL B_TO_BCD
                   MOV CHAR1_1, R1
                   MOV CHAR1_2, R2
                   MOV A, MINUTE
                   LCALL B_TO_BCD
                   MOV CHAR1_4, R1
                   MOV CHAR1_5, R2
                   MOV A, SECONDE
                   LCALL B_TO_BCD
                   MOV CHAR1_7, R1
                   MOV CHAR1_8, R2
```

2.3 Sous-routine d'affichage du nombre de pas:

```
AFFICHE_PAS: PUSH A
    JZ FLAG_MATIN, APREM
    MOV A, PAS_AM_L
    JMP END
    MOV A, PAS_PM_L

APREM:
    LCALL B_TO_BCD
END:   MOV CHAR2_1, R1
        MOV CHAR2_2, R2
        MOV CHAR2_6, #70H
        MOV CHAR2_7, #61H
        MOV CHAR2_8, #73H
```

```
CLR RS
MOV P1, #40H
MOV P1, #00H
MOV A, CHAR2_1
SETB RS
MOV P1, A
SWAP A
MOV P1, A
CLR RS
MOV P1, #40H
MOV P1, #10H
```

```
MOV A, CHAR2_2
SETB RS
MOV P1, A
SWAP A
MOV P1, A
CLR RS
MOV P1, #40H
MOV P1, #50H
MOV A, CHAR2_6
```

```
SETB RS
MOV P1, A
SWAP A
MOV P1, A
CLR RS
MOV P1, #40H
MOV P1, #60H
MOV A, CHAR2_7
SETB RS
```

```
MOV P1, A
SWAP A
MOV P1, A
CLR RS
MOV P1, #40H
MOV P1, #70H
```

```
MOV A, CHAR2_8
SETB RS
MOV P1, A
SWAP A
MOV P1, A
POP A
RET
```

2.4 Sous-routine d'affichage générale:

- redirige vers les autres programmes d'affichage en fonction du mode sélectionné.
- affiche la première ligne de l'écran (heure ou chronomètre en fonction du mode).
- Affiche le nombre de pas sur la seconde ligne de l'écran.
- Va toujours se répéter.

```
AFFICHE_PPAL: JB FLAG_CHN,  
    SJMP AFFICHE_CHN  
        LIGNE_1  
        LCALL AFFICHE_HEURE  
        CLR  
LIGNE_1:    MOV RS  
        MOV P1, #00H  
        SETB A, CHAR1_1  
        MOV RS  
        SWAP P1, A  
        MOV A
```

```
MOV P1,A  
CLR  
MOV RS  
MOV P1,#00H  
MOV P1,#10H  
SETB A,CHAR1_2  
MOV RS  
SWAP P1,A  
MOV P1,A  
CLR  
MOV RS
```

```
MOV P1,#00H  
MOV P2,#20H  
SETB  
MOV RS  
MOV P1,#30H  
MOV P1,#10H  
CLR  
MOV RS  
MOV P1,#00H  
MOV P1,#30H  
SETB A,CHAR1_4  
MOV RS
```

```
SWAP P1,A  
MOV P1,A  
CLR  
MOV RS  
MOV P1,#00H  
MOV P1,#40H  
SETB CHAR1_5  
MOV RS  
SWAP P1,A  
MOV P1,A  
CLR  
MOV RS
```

```
MOV P1,#00H  
MOV P1,#50H  
SETB  
MOV RS  
MOV P1,#30H  
MOV P1,#10H  
CLR  
MOV RS  
MOV P1,#00H  
MOV P1,#30H  
SETB A,CHAR1_7  
MOV RS
```

```

SWAP P1,A
MOV P1,A
CLR
MOV RS
MOV P1,#00H
MOV P1,#40H
SETB A,CHAR1_8
MOV RS
SWAP P1,A
MOV P1,A
LCALL AFFICHE_PAS
LIGNE2: JMP AFFICHE_PPAL

```

3. Horloge, chronomètre et boutons:

3.1 Horloge:

- Reçoit un top par centième de seconde grâce à l'initialisation
- Incrémente les registres HEURE, MINUTE, SECONDE, DIZ-MS
- Tient compte du format 12h ou 24h
- Gère le flag FLAG_MATIN
- RAZ et sauvegarde du nombre de pas à minuit

```

INT_MS:      PUSH A
             INC DIZ_MS
             MOV A, DIZ_MS
             CJNE A ,#100,Sortie
             INC SECOND
             MOV DIZ_MS, #00H
             MOV A, SECOND
             CJNE A, #60, Sortie
             INC MINUTE
             MOV SECOND,#00H
             MOV A, MINUTE

```

```
CJNE A, #60, Sortie
INC HEURE
MOV MINUTE, #00H
JB Mode_24, Suite
MOV A, HEURE
Suite1: CJNE A, #12, Sortie
CPL FLAG_MATIN
MOV PAS_AM_L_S, PAS_AM_L
MOV PAS_AM_L, #00H
MOV PAS_AM_H_S, PAS_AM_H
MOV PAS_AM_H, #00H
CJNE A, #13, Sortie
```

```
MOV HEURE, #01
suite: POP A
RETI
suite: MOV A, HEURE
CJNE A, #12, Suite2
CPL FLAG_MATIN
MOV PAS_AM_L_S, PAS_AM_L
MOV PAS_AM_L, #00H
MOV PAS_AM_H_S, PAS_AM_H
MOV PAS_AM_H, #00H
SJMP Sortie
```

```
CJNE A, #24, Sortie
MOV HEURE, #00
CPL FLAG_MATIN
MOV PAS_PM_L_S, PAS_PM_L
MOV PAS_PM_L, #00H
MOV PAS_PM_H_S, PAS_PM_H
MOV PAS_PM_H, #00H
SJMP Sortie
```

3.2 Interruption des boutons, choix MODE

```
INT0: JNZ FLAG_BT, AFFICHAGE_HEURE  
      JNZ FLAG_CHRONO, AFFICHAGE_HEURE  
      JNZ FLAG_HEURE, MODE2  
AFFICHAGE_HEURE: MOV FLAG_CHRONO, #00H  
                  MOV FLAG_BT, #00H  
                  CPL FLAG_HEURE  
                  RETI
```

```
MODE2:    CPL FLAG_HEURE  
          JNZ BT_CONNECT, AFF_CHN  
          CPL FLAG_BT  
          JUMP sortie  
sortie:   CPL FLAG_CHRONO  
          RETI
```

3.3 Interruption bouton 2

Le second bouton aura un effet différent en fonction du mode dans lequel la montre se trouve. Ce programme permet donc de renvoyer au programme correspondant lorsque le bouton 2 crée une interruption.

```
INT1  JNZ FLAG_HEURE, Bouton_LUM  
      JNZ FLAG_CHRONO, Bouton_Chrono  
      JNZ FLAG_BT, Bouton_BT
```

```
Bouton_Chrono: JNZ FLAG_RAZ, START
                JNZ FLAG_START, STOP
                JNZ FLAG_STOP, RAZ
Start:          CPL FLAG_RAZ
                CPL FLAG_START
Stop:          CPL FLAG_STOP
                CPL FLAG_START
RAZ:           CPL FLAG_STOP
                CPL FLAG_RAZ
```

3.4 Chronomètre

```
INT_CHN_MS: PUSH A
            INC Tmes_GPS
            CJNE A, #50, TEST-CHN
            LJUMP SEND_DATA_BLE
TEST-CHN:    JNZ FLAG_STOP, Sortie
            JNZ FLAG_START, CHN
            JNZ FLAG_RAZ, INIT_CHN
CHN:         INC CHN_DIZ_MS
            MOV A, CHN_DIZ_MS
            CJNE A ,#100,Sortie
```

```
INC CHN_SECOND
MOV CHN_DIZ_MS, #00H
MOV A, CHN_SECOND
CJNE A, #60, Sortie
INC CHN_MINUTE
MOV CHN_SECOND,#00H
MOV A, CHN_MINUTE
CJNE A, #60, Sortie
INC CHN_HEURE
MOV CHN_MINUTE, #00H
```

```
MOV A, CHN_HEURE
CJNE A, #24, Sortie
MOV CHN_HEURE, #00
Sortie: POP A
RETI
```

3.5 Remise à zéro du chronomètre

```
INIT_CHN: MOV CHN_HEURE, #00H
MOV CHN_MINUTE, #00H
MOV CHN_SECONDE, #00H
MOV CHN_SECONDE, #00H
RETI
```

4. Accéléromètre

4.1 Interface I2C:

L'accéléromètre communique par I2C. Les programmes suivants contiennent l'ensemble des sous-programmes permettant l'envoie et la réception de données par I2C.

- **Sous-routine MASTER_CONTROLLER**

Cette sous-routine envoie une condition de START et l'adresse de l'esclave pour commencer une communication.

```
CLR BUS_FAULT
JNB SCL_PIN, FAULT
JNB SDA_PIN, FAULT
CLR SDA_PIN
%DELAY_3_CYCLES
```

```
CLR SCL_PIN  
%DELAY_3_CYCLES  
MOV A, SLV_ADDR  
ACALL SEND_BYTE  
RET  
FAULT:  
SETB BUS_FAULT  
RET
```

- **Sous-routine MASTER_TRANSMIT - SEND_BYTE:**

Cette sous-routine envoie l'octet stocké dans l'accumulateur.

```
SEND_BYTE: MOV BIT_CNT, #8  
SB_LOOP:   RLC A  
           MOV SDA_PIN, C  
           %RELEASE_SLC_HIGH  
           %DELAY_3_CYCLES  
           CLR SCL_PIN  
           %DELAY_3_CYCLES
```

```
DJNZ BIT_CNT,  
SB_LOOP  
SETB SDA_PIN  
%RELEASE_SLC_HIGH  
%DELAY_4_CYCLES  
JNB SDA_PIN, SB_EX  
SETB NO_ACK  
SB_EX:    CLR SCL_PIN  
           %DELAY_3_CYCLES  
           RET
```

- **Sous-routine MASTER_TRANSMIT - SEND_DATA**

Cette sous-routine envoie plusieurs octets par le fil SDA.

Les registres suivants doivent être renseignés avant la transmission:

- BYTE_CNT = nombre d'octets à transmettre.

- SLV_ADDR = adresse de l'esclave.
- @R0 = Data à transmettre.

```

SEND_DATA: ACALL MASTER_CONTROLLER
            JB NO_ACK, SDEX
            MOV A, R0
SD_LOOP: ACALL SEND_BYTE
            INC R0
            JB NO_ACK, SDEX
            DJNZ BYTE_CNT, SD_LOOP
            ACALL SEND_STOP
SDEX:     RET

```

• Sous-routine MASTER RECEIVE - RECEIVE DATA BYTES

Cette sous-routine permet de recevoir plusieurs octets d'un esclave dans le registre adressé par R0.

Les registres suivants doivent être renseignés avant la transmission:

- BYTE_CNT = nombre d'octets à transmettre.
- SLV_ADDR = adresse de l'esclave.
- @R0 = Data à transmettre.

```

RCV_DATA: INC SLV_ADDR
            ACALL
            MASTER_CONTROLLER
            JB NO_ACK, RDEX
            ACALL RECV_BYTE
RDLOOP:   MOV @R0, A
            INC R0
            DJNZ BYTE_CNT, RD_LOOP
            ACALL SEND_STOP
RDEX:     RET

```

• Sous-routine SEND_STOP:

Cette sous-routine transmet une condition de STOP pour libérer le bus.

```

SEND_STOP: CLR SDA_PIN
    %RELEASE_SLC_HIGH
    %DELAY_3_CYCLES
    SETB SDA_PIN
    CLR I2C_BUSY
    RET

```

4.2 Sous programme d'initialisation de l'accéléromètre

- Initialise l'accéléromètre en mode 'single tap' en forçant D6 à '1' et D5 à '0' dans le registre INT-ENABLE (0x2E).
- Assigne des valeurs aux registres THRESH_TAP (0x1D) et DUR (0x21) pour définir le seuil de détection d'un pas.

```

MOV BYTE_CNT, #01H
MOV SLV_ADDR, #2EH
LCALL RCV_DATA
ANL R0, #11011111
ORL R0, #01000000
MOV BYTE_CNT, #01H
MOV SLV_ADDR, #2EH
LCALL SEND_DATA
MOV BYTE_CNT, #01H
MOV SLV_ADDR, #1DH
MOV R0, #00010000

```

```

LCALL SEND_DATA
MOV BYTE_CNT, #01H
MOV SLV_ADDR, #21H
MOV R0, #00100000
LCALL SEND_DATA
RET

```

- THRESH_TAP (0x 1D) contient la valeur du seuil d'accélération au-dessus duquel un pas est détecté (facteur d'échelle = 62.5 mg/LSB).
- On initialise THRESH_TAP à 2 g = #00010000.

- DUR (0x 21) contient la durée durant laquelle l'accélération doit être supérieure à THRESH_TAP pour que le pas soit détecté (facteur d'échelle = 625 µs/LSB).

- On initialise DUR à 0.04 s = #00100000.

4.3 Sous programme d'interruption de l'accéléromètre:

Compte les pas lors d'une interruption de pas détectée par l'accéléromètre.

- Incrémente les registres PAS_AM et PAS_PM lors d'une interruption de l'ADXL en fonction de la valeur du flag FLAG_MATIN.

- Valide l'interruption de l'ADXL avant de retourner au programme principal.

```

INT_PAS:    PUSH A
                JZ   FLAG_MATIN, PM
                INC  PAS_AM_L
                JNC  SUITE
                INC  PAS_AM_H
                JUMP SUITE
                INC  PAS_PM_L
PM:        JNC  SUITE
                INC  PAS_AM_H

```

```

SUITE:      MOV  BYTE_CNT, #01H
                MOV  SLV_ADDR, #2EH
                LCALL RCV_DATA
                MOV  A, R0
                ANL  A, #11101111
                MOV  BYTE_CNT, #01H
                MOV  SLV_ADDR, #2EH
                MOV  R0, A
                LCALL SEND_DATA
                RETI

```

Validation interruption (I2C)

- Dans INT_ENABLE (0x2E), mettre D4 à 0 pour arrêter l'interruption et pouvoir retourner dans le programme principal.
- Reviens à réaliser l'opération de masquage ANL A, #11101111 (ANL = ET logique).

5. Le GPS:

5.1 Initialisation du GPS:

Ce programme permet d'initialiser le GPS

```
INIT_GPS: MOV TMOD, #20H //Timer 1 8bit Mode 2  
          MOV SCON,#50H //8bit, 1bit start, 1bit stop  
          MOV TH1,#FAH //vitesse de communication  
          SETB TR1 //Démarrage TIMER1  
          RET
```

5.2 Lecture d'un caractère

```
READ_CHAR: JNB RI,READ_CHAR //attente d'une donnée  
           CLR RI //Nettoyage du flag  
           MOV A,SBUF //lecture d'un octet en ASCII  
           RET
```

5.3 Lecture des données du GPS

Ce programme va lire les données reçues par le GPS et les stocker dans des variables qu'on affichera plus tard sur l'écran.

```
MESURE_POS_GPS: PUSH A  
                 LCALL READ_CHAR  
                 CNJE A,#24H,GPS_DATA  
                 LCALL READ_CHAR  
DEBUT_GPS_DATA1 CNJE A,#2CH,DEBUT_GPS_DATA1  
                  LCALL READ_CHAR  
DEBUT_GPS_DATA2 CNJE A,#2CH,DEBUT_GPS_DATA2  
                  LCALL READ_CHAR  
                  MOV LAT0,A  
                  LCALL READ_CHAR  
                  MOV LAT1,A
```

```
LCALL READ_CHAR  
MOV LAT2,A  
LCALL READ_CHAR  
MOV LAT3,A  
LCALL READ_CHAR  
MOV LAT4,A  
LCALL READ_CHAR  
MOV LAT5,A  
LCALL READ_CHAR  
MOV LAT6,A  
LCALL READ_CHAR  
MOV LAT7,A
```

```
LCALL READ_CHAR  
MOV LAT8,A  
LCALL READ_CHAR  
LCALL READ_CHAR  
MOV LON0,A  
LCALL READ_CHAR  
MOV LON1,A  
LCALL READ_CHAR  
MOV LON2,A  
LCALL READ_CHAR  
MOV LON3,A  
LCALL READ_CHAR
```

```
LCALL READ_CHAR  
MOV LON5,A  
LCALL READ_CHAR  
MOV LON6,A  
LCALL READ_CHAR  
MOV LON7,A  
LCALL READ_CHAR  
MOV LON8,A  
LCALL READ_CHAR  
MOV LON9,A  
POP A  
RET
```

6. Communication : Liaison UART et SPI

On initialise le 8051 pour qu'il envoie les données par Bluetooth

```
INIT_PORT: MOV XBR0,#03H  
MOV P0SKIP,#F3H  
MOV P1SKIP,#F8H  
SETB MSTEN  
MOV SPI0CN,#03H  
MOV SPI0CKR,#H  
RET
```

Conclusion:

A travers ce projet, nous avons pu réaliser un schéma électrique d'une montre connectée en utilisant un microcontrôleur 8051, un module Bluetooth, un GPS et un écran LCD.

A partir des différentes datasheet, nous avons pu réaliser le programme en langage Assembleur permettant de faire communiquer les différents composants de la montre connectée avec le microprocesseur 8051.

D'un point de vue personnel, ce sujet nous a permis d'améliorer nos compétences en langage Assembleur mais aussi en gestion de projet.

Nous tenons à remercier Monsieur Delebarre pour l'accompagnement durant les séances de cours magistraux et de travaux dirigés.