1 Configuration

- Installation du compilateur XC16
- Génération des configurations par défaut : dans la fenêtre *Configurations Bits*, cliquer sur *Generate Source Code to Output* et copier la sortie au début du main
- Désactivation du Watchdog Timer

1.1 Oscillateur

Utilisation de l'oscillateur interne (primary) :

- `#pragma config FNOSC = PRI`
- `#pragma config IESO = OFF`
- `#pragma config POSCMD = HS`

Oscillateur du dsPIC à 20 MHz (peut aller jusqu'à 40) -> fréquence de cycle à $\frac{20e6}{2} = 10$ MHz. Fréquence vérifiée à l'oscilloscope.

1.2 Bus SPI

Configuration PPS (à faire avant toute autre instruction) : déverrouiller l'accès, mapper les PIN du bus SPI puis reverrouiller l'accès.

- `__builtin_write_OSCCONL(OSCCON & ~(1<<6));`
- `RPINR20bits.SCK1R = 20;` (SDI)
- `RPOR10bits.RP21R = 7;` (SDO)
- `RPOR11bits.RP23R = 8;` (SCK)
- `__builtin_write_OSCCONL(OSCCON | (1<<6));`

La fonction d'initialisation `init_SPI()` reste à peu près la même que pour le projet précédent, il suffit de changer les variables préprocesseur du header et quelques noms de registres : `SSPBUF` devient `SPI1BUF`, `SSP1STAT` devient `SPI1STAT`, les bits `CKE` (clock edge) et `SSPEN` (qui devient `SPIEN`) changent de registre de configuration. Le bus SPI est utilisé en mode 8 bits car le LCD ne supporte pas plus mais il peut être utilisé en mode 16 bits, ce qui n'était pas possible avec le `PIC18F8722`, en settant le bit `SPI1CON1bits.MODE16`.

Par rapport à l'initialisation du projet précédent, il faut ajouter les instructions :

- `SPI1CON1bits.MSTEN = 1;` pour activer le bus SPI en mode principal
- `SPI1CON1bits.SPRE = 6;` (prescaler secondaire)
- `SPI1CON1bits.PPRE = 2;` (prescaler primaire)

1.3 MCP4922

En 12 bits et sans le mode x2, le MCP4922 a une plage de sortie de 0 à 4.9988 V ($V_{OUT} = \frac{V_{REF} \times D_n}{2^n} G$). C'est la même qu'en entrée -> pas besoin d'amplification en sortie comme dans le projet précédent qui utilisait le MCP4922 en 8 bits.

1.4 I/O

Le programme n'est pas interactif -> il y a peu d'entrées et sorties à configurer.

- `TRISAbits.TRISA0 = 1;` lecture du signal sur `AN0`
- `TRISCbits.TRISC9 = 0;` `TICK` en output
- `TICK = 0;` `TICK` initialisé à 0

Les entrées et sorties utilisées par le bus SPI et les périphériques qui y sont associés sont configurées dans la fonction `init_SPI()`.

1.5 ADC

La conversion analogique-numérique est plus complexe sur le dsPIC : elle se configure sur 10 registres. Mais le principe reste le même qu'avec le `PIC18F8722`.

- `AD1CON1bits.ADON = 0;` désactivation de la conversion (sinon la configuration ne peut pas se faire)
- `AD1CON1bits.AD12B = 1;` résultat sur un seul canal en 12bits
- `AD1CON3bits.SAMC = 4;` fréquence de conversion à 4 T_{AD}
- `AD1CON3bits.ADCS = 3;` clock de conversion : 1 $T_{AD} = 3T_{CY}$
- `AD1PCFGLbits.PCFG0 = 0;` `AN0` en analogique
- `AD1CON1bits.ADON = 1;` réactivation de la conversion

Le reste des paramètres sont les paramètres par défaut : tensions de références à V_{DD} et V_{SS}, conversion sur le channel 0, représentation du résultat en integer (peut aussi être représenté en Qx).

Fonctionnement de la conversion : armer la conversion, attendre qu'elle soit finie (bit `DONE`) et lire le résultat dans le buffer `ADC1BUF0`. C'est le même principe qu'avec le `PIC18F8722` mais la conversion doit être armée manuellement car pas de special event trigger.

- `AD1CON1bits.SAMP = 1;`
- `AD1CON1bits.SAMP = 0;`
- `while(!AD1CON1bits.DONE);`

1.6 Interruption par le Timer1

Les interruptions sont plus complexes que sur le `PIC18F8722` avec notamment 7 niveaux d'interruptions programmables. Il y a donc plus de registres liés aux interruptions (voir l'annexe sur les interruptions). La valeur du timer est initialisée à 64923 pour 16kHz avec correction (nécessaire car le timer n'est pas réinitialisé automatiquement comme dans le premier dossier), fréquence vérifiée à l'oscilloscope avec le `TICK`. Le programme n'implémente pas la fréquence d'échantillonnage de 8 kHz. Attention : l'activation de l'interruption doit se faire en dernier.

- `T1CON = 0x8000;` activation du timer1 sur la clock interne avec un prescaler `1:1`
- `TMR1 = 64923;` pour du 16 kHz
- `_T1IP = 7;` priorité de l'interruption par le timer à 7, la plus haute (programmable entre 1 et 7)
- `_T1IF = 0;` interrupt flag effacé
- `_T1IE = 1;` interruption activée

2 Fonctionnement du programme

Pas d'interaction dans Proteus : les paramètres du filtres doivent être initialisés dans le code et le code recompilé pour être testé (de la même manière que les tests ont été faits pour le premier dossier).

3 Conclusion

- la résolution du signal passe de 8 à 12 bits -> le signal de sortie est de meilleure qualité
- pour le filtre par moyenne glissante, l'arithmétique 16 bits du `dsPIC` offre plus de possibilités de traitement du signal que le `PIC18F` : le filtre peut être implémenté sans rotation de bits jusqu'à 16 pas
- en filtre passe-haut et passe-bas la résolution est meilleure : les coefficients sont en Q16 ici contre du Q7 dans le premier projet, les calculs intermédiaires sont faits sur des `long`
- pas d'implémentation du filtre écho car la mémoire du `dsPIC` est trop petite pour que ce soit intéressant (2048 bytes)