

Présentation du microcontrôleur

La carte Explorer 16 et sa carte d'extension

Lors des laboratoires, nous utiliserons la carte à microcontrôleur Explorer 16 de Microchip. Cette carte, dont le schéma de principe est tracé en Figure 1 contient un microcontrôleur DSPIC33 du même constructeur (voir plus loin) ainsi que diverses interfaces vers le monde extérieur.

Un circuit de programmation permet d'écrire un programme dans la ROM du contrôleur.

Nous y avons adjoint une carte d'extension comprenant des boutons, un clavier, des conditionneurs pour les signaux analogiques (permettant notamment de sortir du son) ainsi que des connecteurs. Cette carte nous permet d'interagir aisément avec l'Explorer 16.

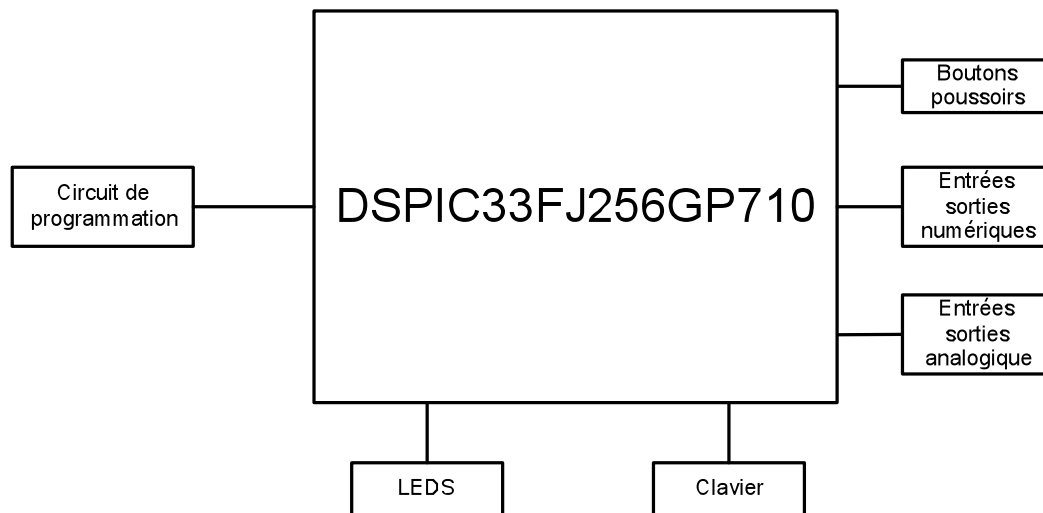


Figure 1 Architecture de la carte

Le microcontrôleur DSPIC33FJ256GP710

Au cœur de l'architecture se trouve le microcontrôleur DSPIC33.

Pour rappel, un microcontrôleur se différencie d'un microprocesseur par le fait qu'il intègre également de la mémoire (ROM et RAM) ainsi que différents périphériques lui permettant de remplir des fonctions multiples. Il s'agit donc d'un système pouvant fonctionner de manière presque autonome (à l'inverse d'un processeur de PC par exemple).

Comme indiqué sur le schéma de la Figure 2, ce contrôleur contient notamment

- Un microprocesseur, cerveau du système,
- De la mémoire ROM non volatile, accueillant le programme devant être exécuté par le processeur ainsi que certaines constantes. Elle ne peut être réécrite en fonctionnement, cette opération doit se faire via un circuit de programmation présent sur la carte,
- De la mémoire RAM volatile contenant toutes les variables temporaires,

- Des timers (horloges) fournissant une base de temps permettant entre autres d'exécuter des actions de manière périodique,
- Des ports d'entrée – sortie numériques, sur lesquels sont notamment connectées des LEDS ainsi que le clavier et des boutons poussoirs,
- Un convertisseur analogique – numérique

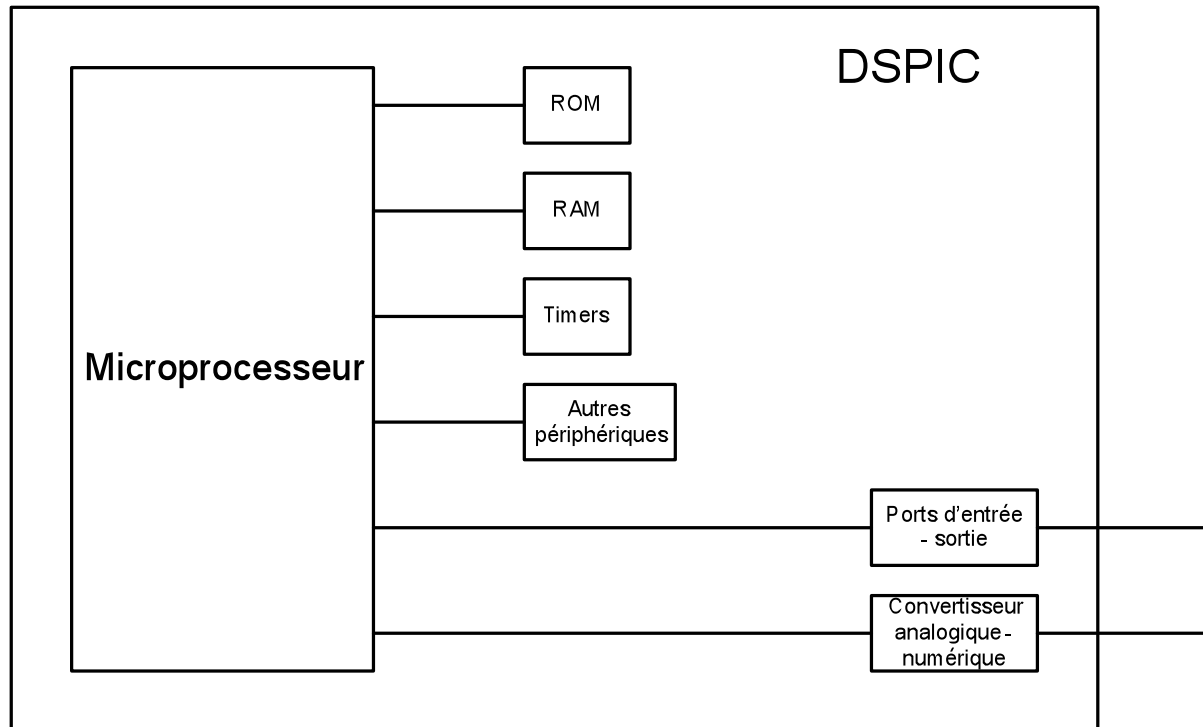


Figure 2 Structure interne du microcontrôleur

Les registres

Un registre est une case mémoire contenue dans le microcontrôleur et pouvant être accédée par le programme. Suivant les cas, un registre peut avoir diverses utilités :

- Interface avec l'extérieur : on peut y lire la valeur présente sur une patte d'entrée numérique ou y écrire la valeur à imposer sur une patte.
- Configuration du processeur ou des périphériques : choisir si une patte donnée est une entrée ou une sortie, choisir la période d'un timer, ...
- Statut : indique l'état du processeur ou d'un périphérique. On peut par exemple y voir si une conversion analogique numérique est terminée ou si un timer est arrivé à son terme.

Les registres du DSPIC peuvent tous être accédés de deux manières

- Par mot : on y écrit ou lit en une fois un mot de 16 bits. Lorsque l'on désire écrire une constante, il est possible de la mettre sous forme hexadécimale en lui ajoutant le préfixe 0x

Exemples

```
PR1 = 5000 ; //écrit la valeur 5000 dans le registre PR1
```

```
TRISA = 0x4FFF ; //écrit le nombre hexadécimal 4FFF (20479 en
décimal) dans le registre TRISA
Int myVar = PORTA ; // écrit le contenu du registre PORTA dans
la variable myVar
```

- Par bit : il est également possible de ne modifier ou de ne lire qu'un seul bit d'un registre. Cette méthode est souvent avantageuse pour lire l'état d'un périphérique. Les valeurs lues et écrites sont forcément binaires. Cet accès est réalisé en faisant suivre le nom du registre par le suffixe « bits », puis d'un point suivi du nom du bit à modifier (à trouver dans la notice ou le présent document)

Exemples

```
TRISAbits.TRISA1 = 1 ; //écrit la valeur 1 dans le bit TRISA1
du registre TRISA
If(IFS0bits.T2IF==1) a=1 : //met la variable a à 1 si le bit
T2IF du registre IFS0 contient la valeur 1
```

Les ports d'entrée-sortie

Les pattes d'entrée-sortie (I/O pour input-output) constituent le moyen de communication avec l'extérieur le plus basique.

Le principe d'envoi de données est simple : écrire un '1' dans le bit correspondant à une patte indique au processeur qu'il doit imposer une tension de 3.3V sur cette patte. A l'inverse, écrire un '0' dans le registre forcera le processeur à imposer 0V sur la sortie.

Le fonctionnement est semblable lorsqu'il s'agit de lire une valeur : relier une source de tension proche de 3.3V à une patte forcera le processeur à écrire un '1' au niveau du bit correspondant à cette patte dans un registre associé.

Le DSPIC contient différentes séries d'I/O, nommées PORTA à PORTG et contenant chacune de 8 à 16 pattes distinctes. Les I/O du port A sont nommées RA0 à RA15, le principe est le même pour les autres ports (RB0 à RB15, RE0 à RE7, ...)

Avant de lire/écrire sur une patte, il faut d'abord indiquer au processeur s'il s'agit d'une entrée ou d'une sortie. Ceci se fait par l'intermédiaire des registres TRIS. Si l'on souhaite, par exemple, indiquer que la patte RA3 du port A est une entrée, il suffit d'écrire la ligne :

```
TRISAbits.TRISA3=1 ;
```

A l'inverse, si la patte est une sortie :

```
TRISAbits.TRISA3=0 ;
```

Il est également possible de configurer tout un port en une fois :

```
TRISB = 0x1E5C ; // Configure tout le port B en une seule fois.
```

Le nombre hexadécimal 1E5C correspond au nombre binaire 0001111001011100. Sachant que le bit de poids faible correspond à RB0, on en déduit que les pattes RB0, RB1, RB5,

RB7, RB8, RB13, RB14 et RB15 sont configurées en sortie et que les autres sont configurées en entrée.

Une fois la patte configurée, il est possible d'interagir avec elle via les registres PORT :

```
PORTAbits.RA4 = 1 ; //écrit un '1' sur la patte RA4
```

```
Int myVar = PORTB ; //lit la valeur de toutes les pattes du PORTB  
comme un mot de 16 bits et l'écrit dans la variable myVar. A  
nouveau, RB0 est le bit de poids faible
```

Les LEDS

Huit LEDS sont présentes sur la carte Explorer 16. Comme le montre la Figure 3, elles sont connectées aux pattes RA0 à RA7.

Pour allumer/éteindre une LED, il suffit d'écrire un '1' ou un '0' sur la patte correspondante

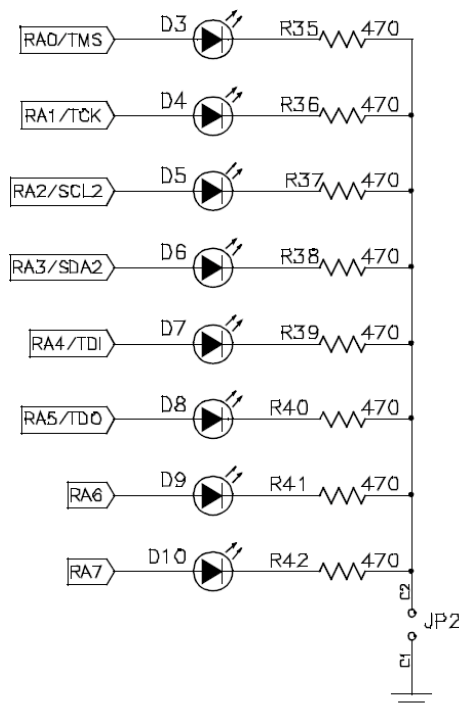


Figure 3 Connexion des LEDS

Les boutons poussoirs

Quatre boutons sont connectés sur la carte d'extension. En regardant la carte d'extension et en partant du haut, ceux-ci sont connectés aux pattes RD12, RD7, RD13 et RD6.

Le schéma de principe est donné sur la Figure 4. Lorsque l'on n'appuie pas sur un bouton, la patte correspondante mise à 3.3V via la résistance (nommée pull-up car elle « tire » la tension vers le haut lorsque l'on n'impose pas volontairement la tension). Lorsque l'on appuie sur le bouton, la patte est reliée à la masse, c'est-à-dire 0V.

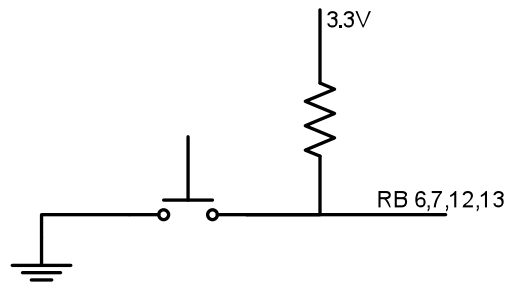


Figure 4 Boutons poussoirs

Les timers (horloges)

Les timers permettent d'exécuter une fonction à intervalle fixe, de mesurer une durée ou d'obtenir une base de temps.

Le principe est le suivant : on commence par entrer une période dans le registre PRx (ou x est le numéro du timer, de 1 à 9). Cette période est exprimée en nombre de cycles d'exécution du processeur. Par défaut, le processeur exécute des instructions à une fréquence de 23,03125 MHz.

```
PR2 = 10000 ; //indique au timer 2 que sa période est de 1000023.03
106=434.2µs
```

Attention : la fréquence de l'horloge du Timer 1 est prédivisée par 256. Ecrire PR1 = 10 correspond donc à $1023.03 \cdot 106 \cdot 256$. Cela rend le timer moins précis (intervalle de temps plus grand) mais permet en contre partie de compter des durées plus longues.

Le processeur va alors incrémenter un compteur jusqu'à arriver à la valeur de PR2. A ce moment, le bit TxF est mis à '1' par le processeur. Pour les timers 1 à 3, ce bit se trouve dans le registre IFS0. Attention : ce bit doit être remis à '0' manuellement dans le programme. Dans le même temps, le compteur est automatiquement remis à zéro.

Pour lancer le timer, il suffit de mettre à '1' le bit TON du registre TxCON correspondant au timer voulu. Mettre ce même bit à '0' gèle le compteur à sa valeur actuelle

```
T1CONbits.TON = 1 ; //lance le timer1
```

Le convertisseur analogique-numérique

Note : il s'agit ici d'une explication fortement simplifiée : le fonctionnement complet du convertisseur est hautement plus complexe.

Le contrôleur contient un convertisseur analogique – numérique permettant, comme son nom l'indique, de numériser une tension présente à son entrée pour s'en servir dans le cadre de traitement numérique. La grandeur numérique comprise entre 0V et 3.3V obtenue est codée sur 12 bits. Une tension de 0V donnera un résultat de 0 et une tension de 3.3V donnera un résultat de 4095

Le convertisseur possède 32 entrées analogiques nommées AN0 à AN31, dont seules 4 sont accessibles sur la carte d'extension (AN0, AN1, AN3 et AN4).

Le lancement d'une conversion se fait en mettant à '0' le bit SAMP du registre AD1CON1. Cette conversion n'est pas instantanée : elle prend quelques μ s à être réalisée.

```
AD1CON1bits.SAMP=0 ; //lance une conversion
```

Une fois la conversion terminée, son résultat est écrit dans le registre ADC1BUF0. En même temps, le bit DONE du registre AD1CON1 est mis à '1'. Regarder la valeur de ce bit permet donc de vérifier si la conversion est terminée.

La carte est configurée pour faire l'acquisition de la tension présente sur la patte AN0. Un potentiomètre est relié à cette patte : en le faisant tourner, on fait varier la tension appliquée à la patte de 0V à 3.3V.

L'interface numérique-analogique et l'ampli de puissance

Le DSPIC ne contient pas de convertisseur numérique analogique, un convertisseur externe a donc été ajouté. Le principe est semblable à celui du convertisseur analogique – numérique : un nombre de 12 bits est converti en une tension comprise entre 0V et 3.3V.

A la sortie du convertisseur se trouve l'étage de sortie utilisé lors du montage de la radio : un filtre passe-haut suivi de l'ampli de puissance NJM2113. Un connecteur permet de brancher un baffle.

La fonction `ecrit_dac_signal(int val)` permet de convertir le nombre *val* en grandeur analogique. Le convertisseur numérique-analogique fonctionnant sur 12 bits, *val* ne peut valoir plus de 4095.

Les types de variables

Tout comme dans le C classique, plusieurs types de variables sont disponibles. Les types suivants sont les plus utilisés:

char : variable stockée sur 8 bits

int : variable stockée sur 16 bits

long : variable stockée sur 32 bits

Il est très important de faire attention au type de chaque variable. Par exemple, tous les registres sont des mots de 16 bits ; en conséquence, l'instruction

```
myVar = PORTB ;
```

ne fonctionnera correctement que si *myVar* est une variable d'au moins 16 bits.

De même, soit l'instruction suivante où *a* et *b* sont des variables de type *int*:

```
c = a*b ;
```

Le produit de deux nombres de 16 bits est un nombre de 32 bits. En conséquence, *c* doit être une variable de type *long*