INSTITUT d'OPTIQUE GRADUATE SCHOOL ParisTech

EITI 1A

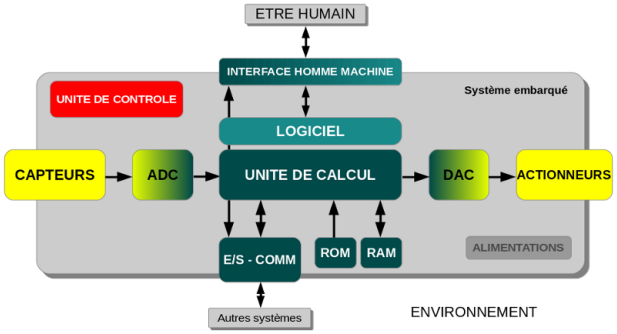
Microcontroleurs PIC16F

2016-2017

Système embarqué

Un **système embarqué** :

— est le regroupement d'un **système matériel** et d'un **logiciel**
— repose sur une **architecture spécifique** dédiée à l'exécution d'un ensemble de **tâches particulières**
— doit être **très réactif**, en parfaite **autonomie** et en contact permanent avec son environnement

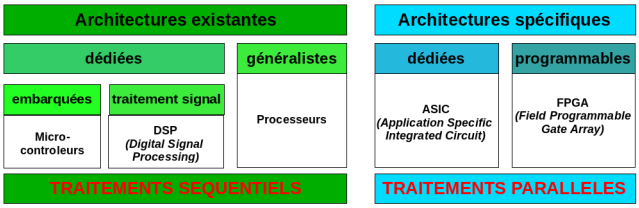


Caractéristiques

— Principalement **numérique** pouvant intégrer une partie analogique (conditionnement, modulation, filtrage...)
— Exécution d'une **application dédiée**
— **Système matériel simplifié**
  — Meilleure fiabilité
  — Réduction de la consommation électrique
  — Réduction des couts de fabrication

Constitution

La partie numérique du traitement peut être réalisée de plusieurs façons différentes :



Lors de votre cursus à l'IOGS, vous serez amené à développer des applications autour de processeurs généralistes et de microcontroleurs.

Microcontroleur

Le **microcontrôleur** est un composant numérique très utilisé dans le monde industriel pour piloter des systèmes de manière **autonome** et en **temps réel**, grâce notamment à ses interactions avec l'environnement, via des **broches de communication** spécifiques.

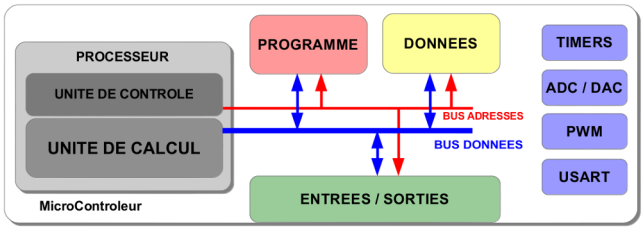
Différences avec un processeur

Un microcontroleur est un composant autonome, qui intègre l'ensemble des **ressources** nécessaires à l'exécution d'une **tâche particulière**, contrairement à un processeur auquel il faut ajouter des espaces mémoires externes.

Les fréquences d'exécution des calculs sont souvent plus lentes que sur un processeur, mais globalement les quelques tâches qu'il a à accomplir sont exécutées plus rapidement que sur un processeur généraliste, souvent associé à un système d'exploitation multitache.

Architecture

Un microcontroleur est constitué : d'une unité de calcul (intégré au processeur), de mémoire de données, de mémoire de programme et d'entrées/sorties.



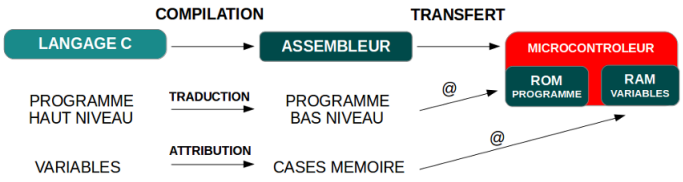
La plupart des microcontroleurs intègrent également d'autres périphériques simplifiant la mise en oeuvre de certains traitements ou l'échange d'informations inter-systèmes.

Ressources

Les microcontroleurs ont toutes les ressources nécessaires en interne pour pouvoir faire du traitement numérique de l'informatique. Cependant les quantités de mémoires sont limitées par rapport à un ordinateur standard. Mais les systèmes embarqués sont cependant connus pour avoir une très grande réactivité aux sollicitations et évènements extérieurs.

Compilation / Transfert

Le langage initial du microcontrôleur (et de tout processeur) est le **langage machine**, souvent associé au langage **assembleur**. L'utilisation d'un compilateur C facilite la production de code pour les applications embarquées et permet de s'affranchir de la connaissance de ce langage de très bas niveau.



Une fois le **programme compilé**, il est **transféré au microcontroleur** pour une exécution en autonomie.

Les **zones mémoires** utilisées sont transparentes pour le programmeur par l'utilisation du compilateur (et de l'éditeur de liens associé). Cependant, il faut s'assurer que la quantité maximale de mémoire n'est pas atteinte. Sinon, il faut changer de microcontroleur.

Programme type

Programme testé sous MPLABX avec le compilateur XC8.

```
#include <xc.h>

void initPIC(void);

void main(void){
  initPIC();
  while(1){
    // programme principal
  }
}

void initPIC(void){
  // fonction d'initialisation
}
```

Registres internes

Les **registres** sont des espaces mémoire de 8 bits pour les familles `PIC16F` et `PIC18F` et de 16 bits pour la famille `dsPIC30F`. Ces espaces mémoires sont **adressables indépendamment**.

Certains sont à **usage général**, permettant ainsi de stocker des données (résultats de calculs par exemple) et d'autres à **usage plus spécifique**, permettant la gestion de certains modules du microcontroleur (ADC, PWM, Timers...) ou l'interaction avec l'environnement extérieur (port d'entrées-sorties).

Registres spécifiques

Pour pouvoir écrire sur le port A par exemple (nommé `PORTA` par le constructeur), il suffira d'écrire :

```
PORTA = 0b00101001; // en binaire
PORTA = 0x29;    // en hexadecimal
PORTA = 41;    // en decimal
```

Pour pouvoir récupérer la valeur du registre `STATUS`, par exemple, il suffira d'écrire :

```
int c = STATUS;
```

Il est également possible d'**affecter** une valeur à **un seul bit** (ou à un ensemble de bits d'un registre).

Pour pouvoir mettre à un le bit 2 du port A par exemple, il suffira d'écrire :

```
PORTAbits.RA2 = 1;
```

Pour pouvoir modifier les 4 bits `IRCF` du registre `OSCCON`, il faudra écrire :

```
OSCCONbits.IRCF = 0b1101; // FOSC = 4 MHz
```

Registres généraux

Une zone de l'espace mémoire est à **usage général**, c'est là où seront stockées les variables. Il est possible de déclarer des variables de type **char** ou **int**.

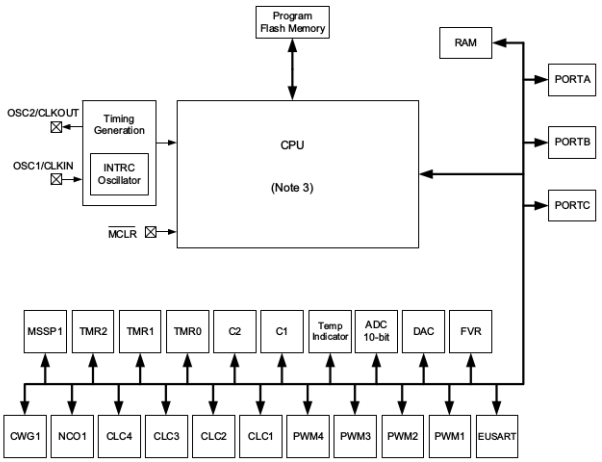
```
char a = 10;  // 1 octet
int b = 34000;  // 2 octets
```

L'instruction suivante permet de faire un décalage d'un bit vers la droite de b et de le stocker dans la variable a. En binaire, cela revient à faire une division par 2.

```
char a, b = 10;
a = b >> 1;
```

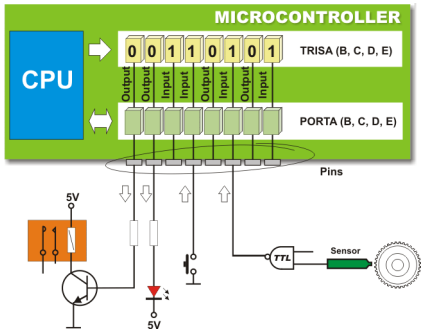
PIC16F1503/9 Voir aussi documentation technique

Structure



Entrées/Sorties numériques

Les broches d'entrées/sorties peuvent avoir plusieurs fonctions sur ces microcontroleurs. Cependant, par défaut, elles peuvent toutes être utilisées en entrées ou sorties numériques. Ces entrées/sorties sont regroupées dans des **ports** nommés de A à F (selon les modèles de composants).



La direction est configurable à l'aide des registres `TRISx` (où x représente le nom du port - A à F).

```
TRISCbits.RC1 = 1;
TRISAbits.RA4 = 0;
```

Les deux lignes précédentes permettent d'initialiser la broche 1 du port C en entrée et la broche 4 du port A en sortie.

Le registre `PORTx` (où x représente le nom du port - A à F) permet de connaître l'état d'une entrée ou d'affecter une valeur sur une sortie.

```
char a = PORTCbits.RC1;
PORTAbits.RA4 = 1;
```

La première ligne permet de stocker dans la variable `a` *la valeur de la broche 1 du port C. La seconde ligne permet d'affecter la valeur '1' à la broche 4 du port A.*

ATTENTION : Par défaut, les broches de communication vers l'extérieur sont configurées en entrée analogique et non en numérique. Il est donc également important de modifier le registre `ANSELx` en conséquence.

Entrées analogiques

La plupart des microcontroleurs intègrent des convertisseurs analogiques-numériques (CAN). Pour pouvoir les utiliser, il faut configurer les entrées associées en mode analogique à l'aide du registre `ANSELx`.

```
ANSELAbits.ANS1 = 1;
ANSELBbits.ANS3 = 0;
```

La première ligne permet de configurer la broche 1 du port A en mode analogique. La seconde ligne permet de configurer la broche 1 du port A en mode numérique.

Le PIC16F1503/9 possède un CAN qui convertit sur 10 bits. On peut sélectionner la voie d'acquisition à l'aide des bits `CHS` du registre `ADCON0`.

La conversion peut se lancer en passant le bit `ADGO` du registre `ADCON0` à '1'.

Exemple d'une conversion sur la broche RA4 (AN3) en continu toutes les 50 ms et affichage sur un écran LCD.

```
void initADC(void){
    TRISAbits.TRISA4 = 1;
    ANSELAbits.ANSA4 = 0;

    ADCON1bits.ADFM = 0;
    ADCON1bits.ADCS = 0b111;
    ADCON1bits.ADPREF = 0;
    ADCON0bits.CHS = 3;
    ADCON0bits.ADON = 1;
}

void main(void) {
    initADC();
    initLCD_DOG();
    while(1){
        ADCON0bits.ADGO = 1;
        while(ADCON0bits.ADGO == 1);
        res = (ADRESH << 2) + (ADRESL >> 6);
        clearLCD();
        sprintf(ligne, "Val = %d", res);
        writeStrLCD(ligne, 1, 1);
        __delay_ms(50);
    }
    return;
}
```

Registres importants

STATUS

Le registre `STATUS` contient des informations sur l'état de l'**unité de calcul**.

REGISTER 3-1: STATUS: STATUS REGISTER

U-0	U-0	U-0	R-1/q	R-1/q	R/W-0/u	R/W-0/u	R/W-0/u
—	—	—	$\overline{TO}$	$\overline{PD}$	Z	DC(1)	C(1)
bit 7							bit 0

Z Zero bit - 0 si résultat différent de 0, 1 sinon

C Carry/Borrow bit - 0 si pas de retenu sur 8 bits, 1 sinon

TRISx / PORTx / ANSELx

Les registres `TRISx`, `PORTx` et `ANSELx` sont liés aux broches d'entrées/sorties du port X. Tous les ports fonctionnent sur le même principe.

Un bit de chacun de ces registres est alors associé à chaque broche du composant.

OSCCON

Le registre `OSCCON` permet de configurer l'**horloge du système**.

REGISTER 5-1: OSCCON: OSCILLATOR CONTROL REGISTER

U-0	R/W-0/0	R/W-1/1	R/W-1/1	R/W-1/1	U-0	R/W-0/0	R/W-0/0
—	IRCF<3:0>				—	SCS<1:0>	
bit 7							bit 0

ADCONx / ADRES

Les registres `ADCONx` permettent de configurer le **convertisseur analogique-numérique**.

REGISTER 15-1: ADCON0: ADC CONTROL REGISTER 0

U-0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0
—	CHS<4:0>					GO/$\overline{DONE}$	ADON
bit 7							bit 0

CHS Permet de sélectionner la voie à acquérir
- 00000 AN0
- 00001 AN1
- 01011 AN11

ADGO Permet de connaître l'état de la conversion et de la lancer : 1 si conversion en cours, 0 sinon

ADON Permet d'alimenter le CAN : 1 pour valider, 0 sinon

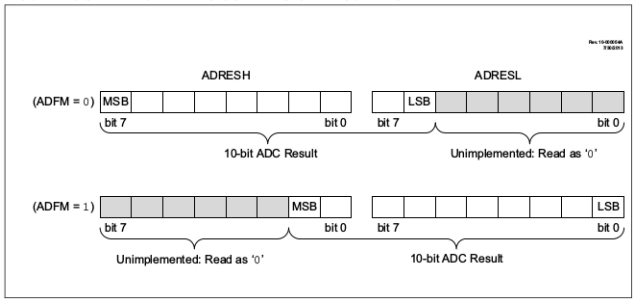
REGISTER 15-2: ADCON1: ADC CONTROL REGISTER 1

R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	U-0	U-0	R/W-0/0	R/W-0/0
ADFM	ADCS<2:0>			—	—	ADPREF<1:0>	
bit 7							bit 0

ADFM Permet de choisir la justification du résultat : 0 à gauche, 1 à droite.

ADCS Permet de choisir l'horloge de conversion. Par défaut, il faut choisir $F_{RC} = 011$.

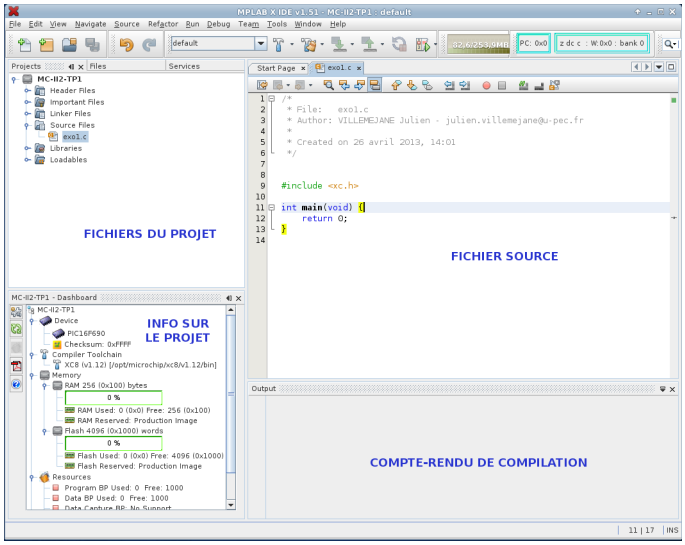
FIGURE 15-3: 10-BIT ADC CONVERSION RESULT FORMAT



MPLABX - XC8

Interface

L'environnement de développement est `MPLABX`.



Créer un projet

Vous devez au préalable lancer `MPLABX IDE`.

1. Lancer ensuite l'assistant de création de projet (`File` → `New Project`).
2. A la première page, il faut choisir la catégorie `Microchip Embedded` et un projet du type `Standalone project`. Puis cliquer sur `Next`.
3. Choisir ensuite le microcontrôleur sur lequel vous voulez travailler dans `Device` (`PIC16F1503`). Puis cliquer sur `Next`.
4. Laisser l'option `None` dans la page suivante. Puis cliquer sur `Next`.
5. A la page `Select Tool`, sélectionner l'option `Simulator`, si vous n'avez pas de carte d'étude, ou l'option `ICD3` si une carte d'étude est connectée au PC. Puis cliquer sur `Next`.
6. A la page `Select Compiler`, sélectionner l'option `XC8` pour développer en C. Puis cliquer sur `Next`.
7. A la page `Select Project Name and Folder`, donner un nom à votre projet (TPx par exemple) et entrer votre répertoire de travail (dans U :/). Puis cliquer sur `Finish`.

Attention : indiquer un chemin du type : `u:/`
et non pas `://domain.iogs/`

Associer au projet un code source existant

1. Faire un *clic droit* sur le dossier `Source Files`.
2. Sélectionner `Add existing file`.

Écrire un nouveau code source

Si le fichier source n'est pas donné, le créer peut être réalisé par la procédure suivante :

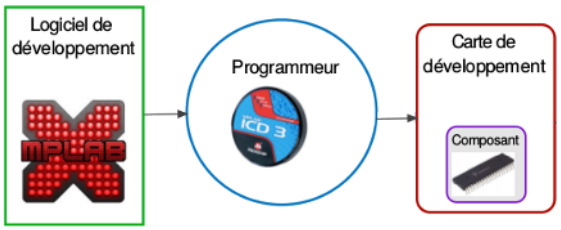
1. Faire un *clic droit* sur le dossier `Source Files`.
2. Sélectionner ensuite `New` et `Others`.
3. Choisir ensuite la catégorie `Microchip Embedded / XC8 Compiler` et le type de fichier `main.c`. Puis cliquer sur `Next`.
4. Donner un nom à votre fichier source et cliquer sur `Finish`.

Vous devriez obtenir un fichier `*.c` ressemblant à ceci :

```
#include <xc.h>

int main(void)
{
  return 0;
}
```

Chaine de développement



Connecteur ICD3 / Brochage PIC16F1503