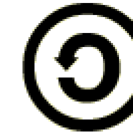
Cours micro pic18
IUT GEII

cours micro pic 18 v0.91.odp

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Licence



Paternité - Pas d'utilisation Commerciale -
Partage des Conditions Initiales à l'identique 2.0 France

Vous êtes libres :
* de reproduire, distribuer et communiquer cette création au public
* de modifier cette création, selon les conditions suivantes :

Paternité. Vous devez citer le nom de l'auteur original.

Pas d'utilisation Commerciale.
Vous n'avez pas le droit d'utiliser cette création à des fins commerciales.

Partage des Conditions Initiales à l'identique
Si vous modifiez, transformez ou adaptez cette création,
vous n'avez le droit de distribuer la création qui en résulte
que sous un contrat identique à celui-ci.
* A chaque réutilisation ou distribution, vous devez faire apparaître clairement aux autres les conditions contractuelles de mise à disposition de cette création.
* Chacune de ces conditions peut être levée si vous obtenez l'autorisation du titulaire des droits
Ce qui précède n'affecte en rien vos droits en tant qu'utilisateur (exceptions au droit d'auteur : copies réservées à l'usage privé du copiste, courtes citations, parodie...)
voir le contrat complet sous : http://fr.creativecommons.org/contrats.htm

Plan du cours

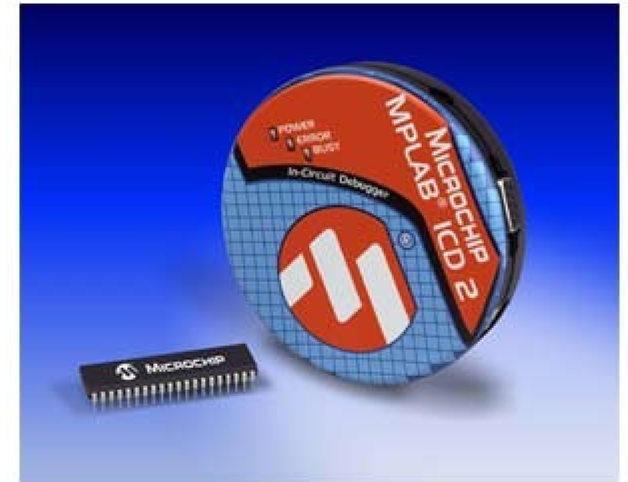
- Matériels et logiciels nécessaires
- Hello world pic18
- Utiliser un écran lcd à base de contrôleur HD44780
- Structure logicielle d'une application / Machine d'états
- Les entrées/sorties logiques
- Le module convertisseur analogique/numérique (adc)
- Les interruptions
- Gestion du temps
- Générer un signal PWM avec un module CCP
- Liaison série asynchrone module UART

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Matériel et logiciels nécessaires

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Matériels nécessaires



- **ICD2 : debugger / programmateur**
  Il permet de charger un programme dans le micro (programmeur) et de contrôler son exécution, de voir et de modifier des variables, de placer des points d'arrêt (debugger).
  (vérifier que le pic utilisé est compatible icd2)

- On peut également utiliser un simple programmateur (picstart plus) mais la mise au point sera fastidieuse !

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Connections ICD2/PC/Cible

- Liaison ICD2<>PC par le bus USB

  l'ICD est alimenté par l'USB , la cible doit posséder sa propre alimentation.

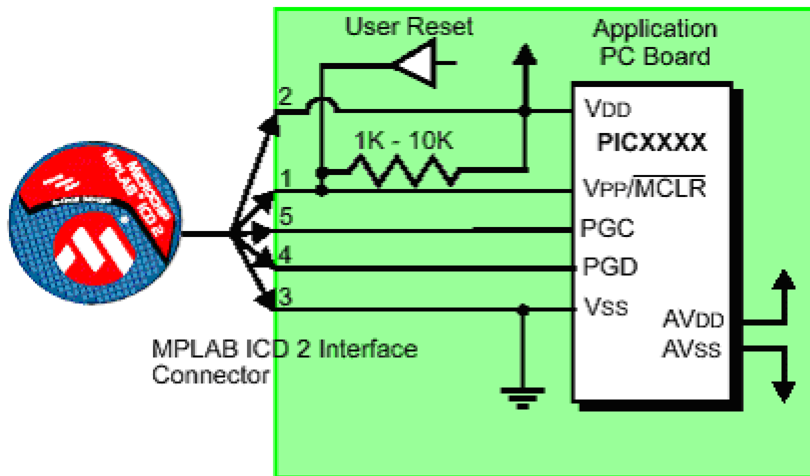
- Laison ICD2<>PC par le port série RS232

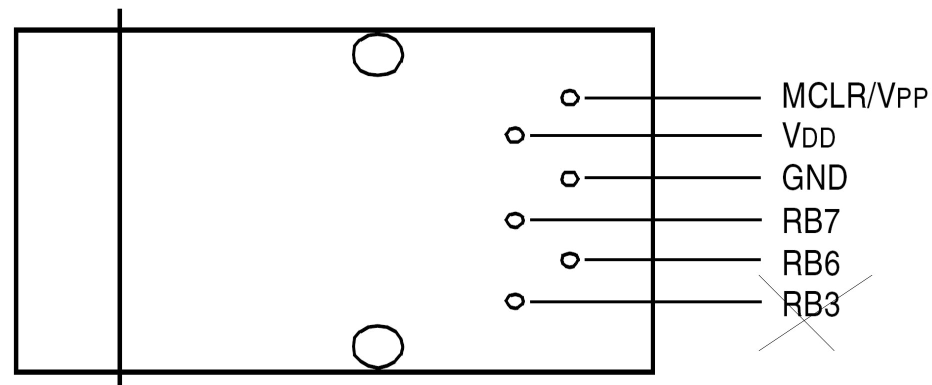
  L'ICD doit être alimenté par l'alimentation fournie.
  L'ICD peut alimenter la cible en validant une option dans le menu :Debugger->setting->Power
  (pour une faible consommation seulement)

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Matériels nécessaires

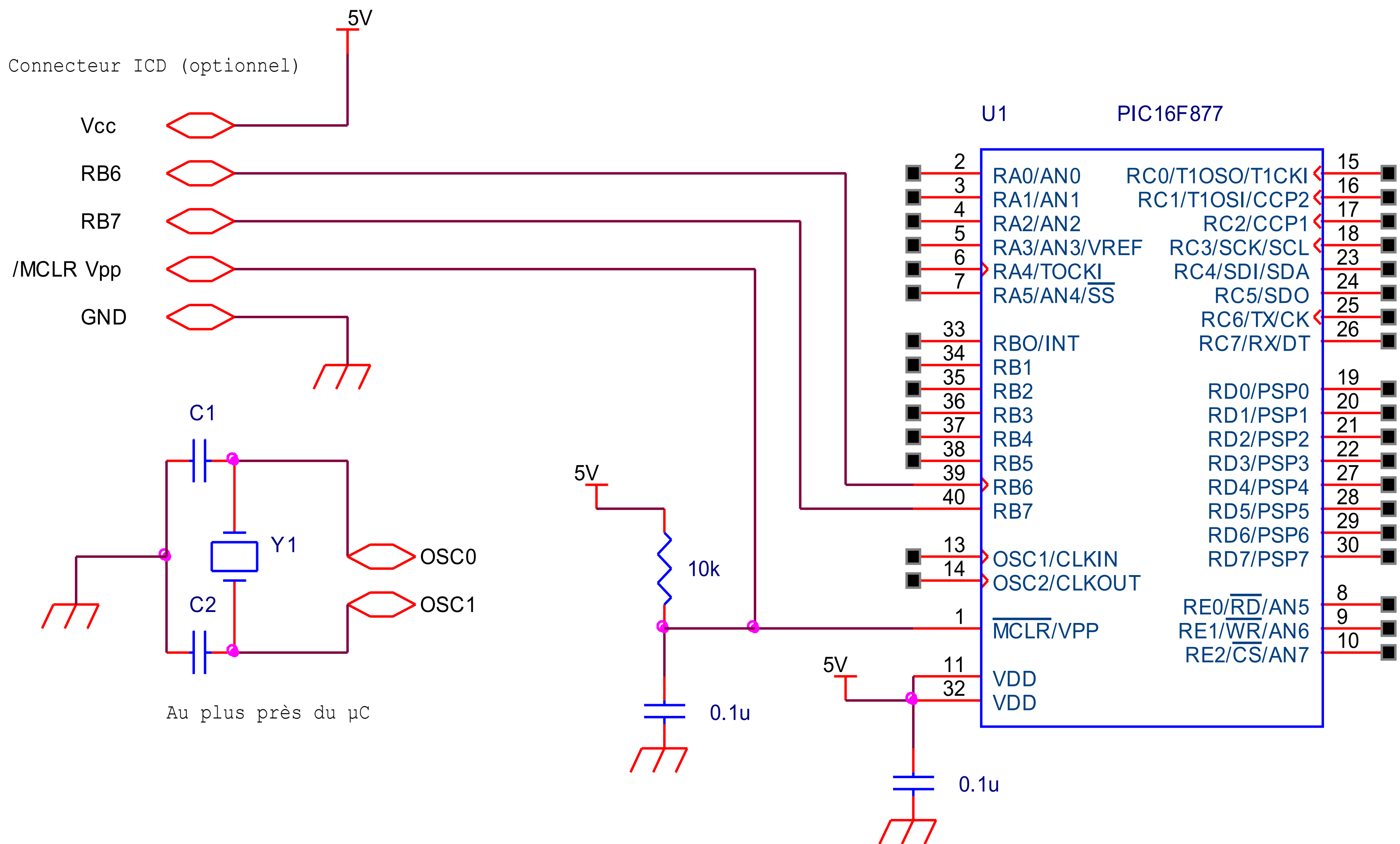
- Une carte cible munie d'un connecteur icd2 (rj12).





© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Schéma minimum de démarrage



© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Schéma minimum de démarrage

- Si la tension Vdd s'établit assez rapidement,on peut se passer du circuit RC de Reset et relier directement la broche /Reset à Vdd (si on utilise pas l'icd).

- Pour une application où la précision sur les durées n'est pas critique, on peut utiliser un circuit RC pour générer l'horloge.

- L'utilisation du programateur/débugger ICD2 fait perdre l'usage de :

2 broches E/S (RB6 et RB7)
2 niveaux de pile (sur 31 possibles)
512 octets de mémoire programme
10 octets de mémoire data.

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

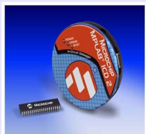
Documentations

PIC 18FXX2 Data Sheet DS39564b.pdf

MPLAB-C18-Getting-Started_51295f.pdf

MPLAB-C18-Users-Guide_51288j.pdf

MPLAB-C18-Libraries_51297f.pdf



© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Logiciels nécessaires

- L'environnement MPLAB IDE (www.microchip.com)
- Le compilateur C18 (www.microchip.com)
- Pour le lcd de la carte picdem2+ la librairie xcldvvv.lib et son fichier entête xcldvvv.h correspondant. (arlotto.univ-tln.fr)

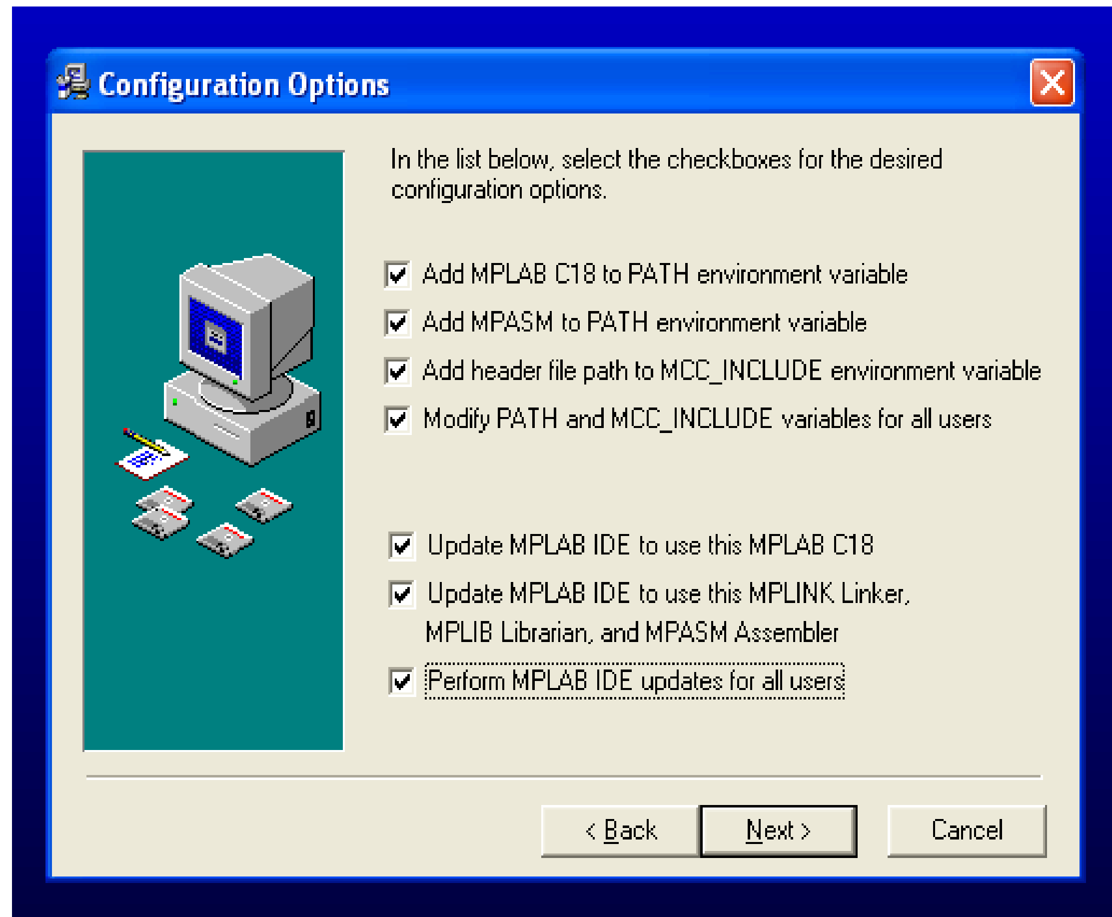
  N'hésitez pas à télécharger la version la plus récente :
  Les mises en jours sont fréquentes et corrigent
  de nombreux bugs.
- Pour ce cours on utilise :

  mplab **v7.31** c18 **v3.02** (linker 4.02) xlcd**100**.lib

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Installation

- MPLAB : pré-installer les drivers usb
- C18 : cocher toutes les options et respecter le répertoire par défaut



© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Hello world pic18

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Hello world pic18

- Créer un projet avec Project->Project Wizard...
- Choisir le bon pic (18f452)
- Choisir le langage : Microchip C18 Toolsuite
- Créer un répertoire pour chaque projet
- Ajouter en copiant un linker script au projet.
  le script sample du pic correspondant convient la plupart du temps si vous n'utilisez pas de librairie supplémentaire
  (ici 18f452.lkr)
- Terminez

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Hello World pic18

- Créez le fichier HelloWorld.c et ajoutez-le au projet

```
#include <p18cxxx.h>

// bits de configuration
#pragma config OSC = HS // dépend de l'oscillateur utilisé
#pragma config WDT = OFF // pas de wd pour debug
#pragma config LVP = OFF // pas de lvp pour l'icd

void main(void)
 {
PORTBbits.RB3 = 0 ;
TRISBbits.TRISB3 = 0 ;  // RB3 en sortie
```

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Hello World pic18

```
for(;;) {
if(PORTAbits.RA4==0) {
                    PORTBbits.RB3 = 1 ;
                  }
          else  {
                   PORTBbits.RB3 = 0 ;
                  }
         } // fin for
} // fin main
```

- Compilez et linkez : Projet->Build All
- Corrigez vos erreurs éventuelles (puis Build All à nouveau)

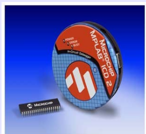
© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Hello World pic18

- Debugger->Select Tool->MPLAB ICD2
  connecter l'icd (Reset and connect icd)
- Chargez le programme sur la cible : Debugger->Program
- Lancez le programme : Debugger->Run
- Essayez le !!

  A ce stade le programme est chargé dans la cible mais elle n'est pas autonome : le pic attend l'ordre de l'icd pour démarrer.

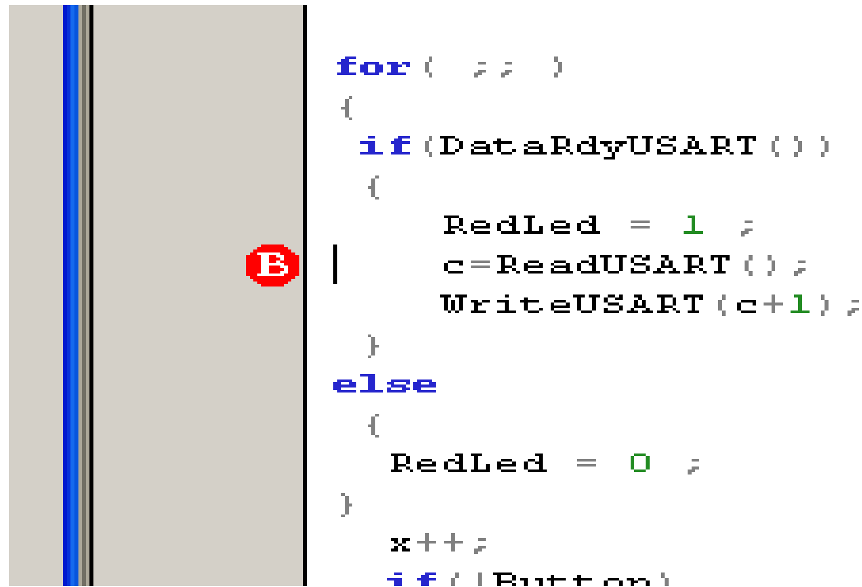
  Pour rendre la cible autonome :
- Choisir l'icd en tant que "programmer" (Programmer->Select Programmer)
- Recompiler et recharger le programme.



© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Debugger avec l'icd

- Point d'arrêt : double-clic à gauche d'une ligne de programme

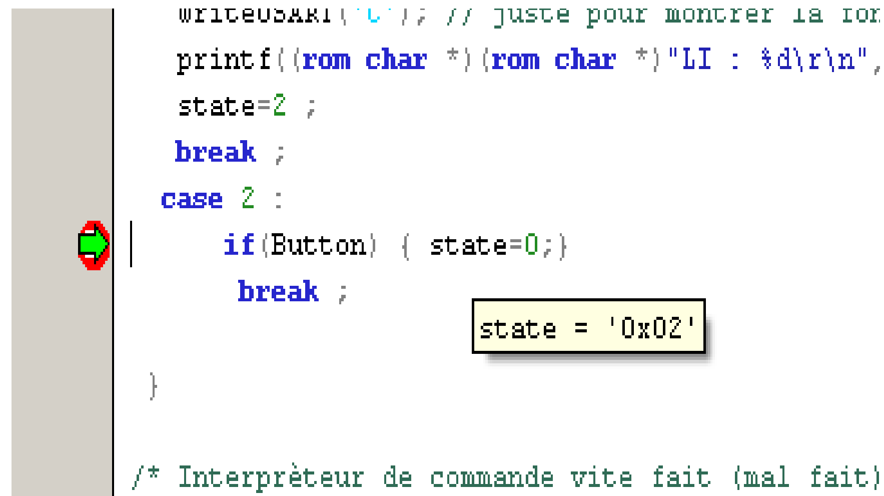


Attention certaines les lignes du source ne se retrouvent pas dans le programme !
L'arrêt ou non du programme est une indication du passage par la ligne où est placé le point d'arrêt

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Debugger avec l'icd

- Visualisation de la valeur d'une variable :
  A l'arrêt passer la souris sur une variable montre sa valeur.

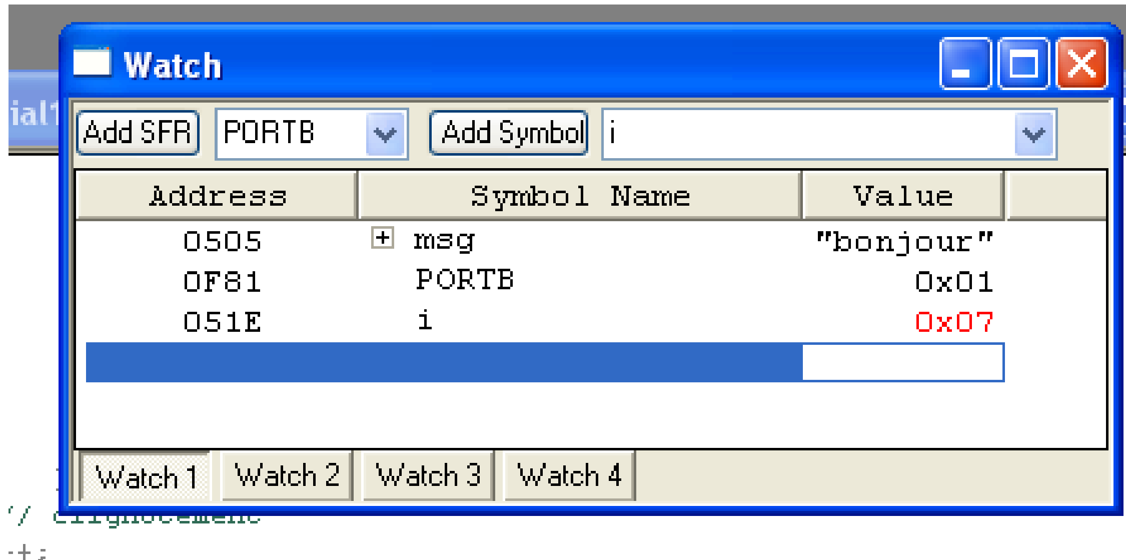


Attention, une variable n'a qu'une valeur à un moment donné. La valeur montrée est donc indépendante de la position pointée dans le programme.

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Debugger avec l'icd

- Voir et modifier les variables et les registres avec une "watch window"



la valeur en rouge indique :
soit un changement depuis le dernier affichage
soit une valeur que l'on a modifiée et qui sera chargée au prochain démarrage.

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Utiliser un écran lcd à base d'un contrôleur HD44780

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Ecran lcd HD44780

- C18 est fournit avec une bibliothèque de fonctions de pilotage d'un écran lcd HD44780 appelée xlcd par microchip.
- Les sources sont dans : C:\mcc18\src\traditional\pmc\XLCD
- Il suffit de modifier le fichier xlcd.h pour adapter les définitions aux lignes de commande utilisées dans votre câblage. Puis de recompiler la librairie de c18.
- Ce travail est fait pour le câblage de la carte picdem2+ (voir http://arlotto.univ-tln.fr/pic/pic18/sources_xlcdlib100/)
- Attention le câblage particulier de la carte picdem2+, interdit l'utilisation des entrées analogiques autre que AN0 !
  Il vaut mieux mettre le lcd sur le port D d'un 18F4520 si on souhaite utiliser plus d'une entrée analogique.

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Utiliser la librairie xlcd100

- Recopier les fichiers xlcd100.lib , xlcd100.h et 18F452xlcd100.lkr dans le répertoire de votre projet et ajoutez les au projet.
- Vérifier que : Dans Project->Build Option->Project :
  1/ Library Path contient seulement : c\mcc18\lib
  Les autres champ sont vides
  2/ Dans l'onglet MPLAB C18 Categories Memory Model
  Les modèles mémoires Code et RAM utilisés sont "Large"
- Inclure l'entête de la librairie par #include "xlcd100.h"
- Pour printf inclure stdio.h par #include <stdio.h>
  et changer la sortie standard par stdout =_H_USER ;

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Utiliser la librairie xlcd100

- Initialiser le lcd par :

```
OpenXLCD(OPEN_PICDEM_LCD);
```

- Ensuite les fonctions de stdio ainsi que les fonctions décrites dans la documentation microchip sont utilisables
- Pour positionner le curseur, une fonction supplémentaire est disponible : gotoXLCD

```
gotoXLCD(LCD_LINE_ONE);// ligne 1 pos 0

unsigned char i =4 ;
gotoXLCD(LCD_LINE_TWO+i); // ligne 2 pos i (4)
```

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Utiliser la librairie xlcd100

- Voir un exemple sur :

  http://arlotto.univ-tln.fr/pic/pic18/xlcd100_demo/

- Les fichiers à récupérer sont à :

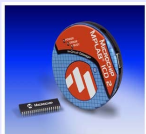
  http://arlotto.univ-tln.fr/pic/pic18/lib_xlcd100/

- Pour correction et adaptation on peut partir du projet :

  http://arlotto.univ-tln.fr/pic/pic18/sources_xlcdlib100/

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Modèle de programmation



© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Structure d'un programme

- Le programme embarqué dans le micro a la particularité de démarrer à la mise sous tension du système et de ne s'arrêter qu'à la coupure de l'alimentation. C'est donc un programme qui après une séquence d'initialisation tourne en *une boucle infinie*.
- Cette boucle peut être interrompue par des *interruptions*
- Les *tâches* de l'application sont exécutées en séquence les unes après les autres dans un ordre prédéfinit.

  Les tâches sont ici des séquences d'instructions qui répondent aux différentes fonctionnalités de l'application. Ce ne sont pas des boucles infinies.

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Interruption

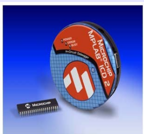
Une interruption est un évènement matériel* qui déclenche *très rapidement* l'exécution d'une partie de programme (appelée programme d'interruption). Après l'exécution du programme d'interruption, le µP reprend l'exécution normale du programme interrompu.

* : sur certains µP, il existe aussi des évènements logiciels (division par 0, erreur d'adresse,instruction spéciale,...) qui peuvent déclencher une interruption : on parle alors d'interruption logicielle (ou synchrone).

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Modèle boucle infinie/ interruptions (superloop background/foreground)

```
void main(void)
{
    séquence d'initialisation ;
    ....;
    autorisation éventuelle des interruptions ;
    for( ;;)
    {
        Tâche n°1 ;
        Tâche n°2 ;
        ....;
        Tâche n° N ;
    }
}
```



© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

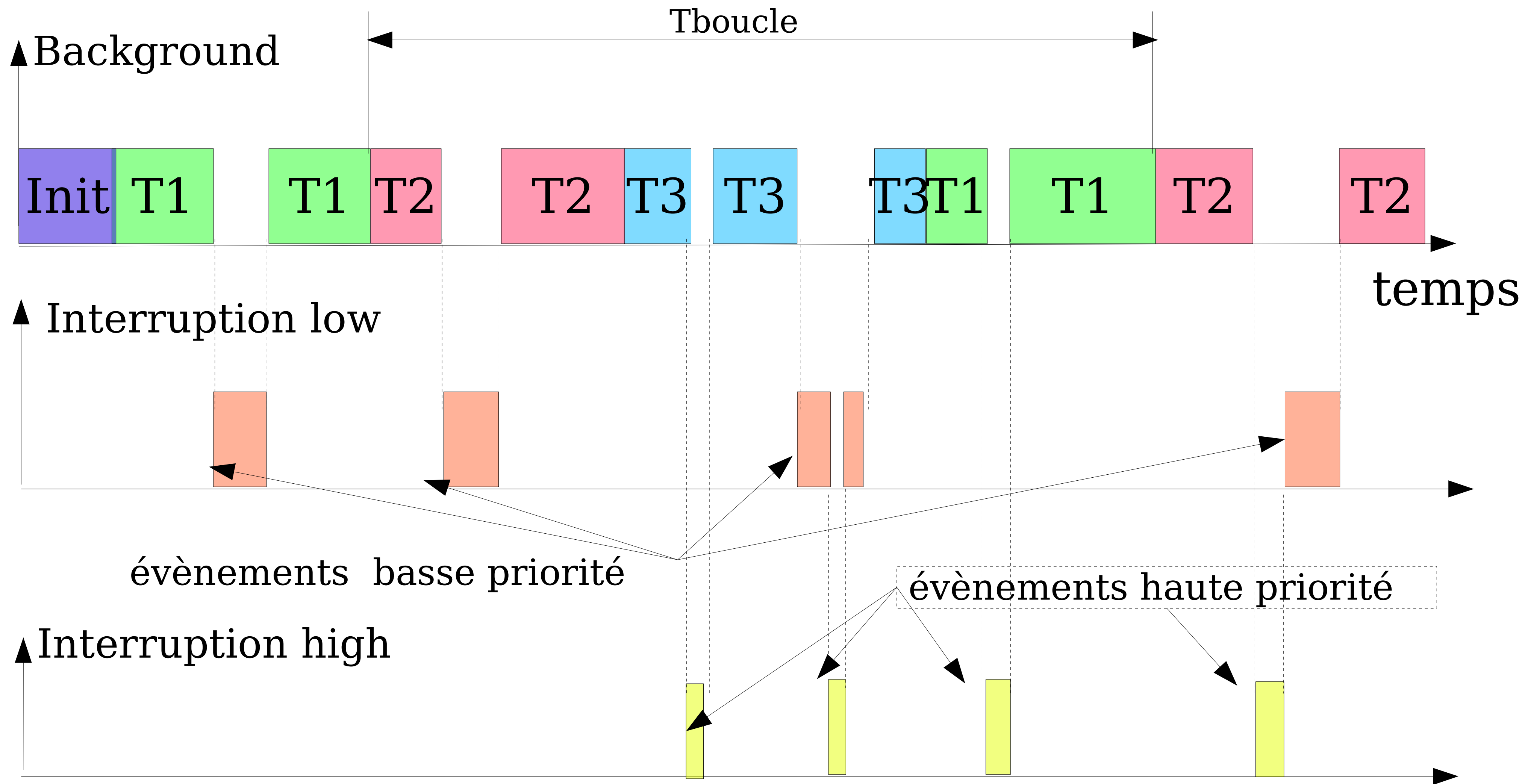
Modèle boucle infinie/ interruptions

```
#pragma interrupt InterruptHandlerHigh
void InterruptHandlerHigh ()
{
....;
}

#pragma interrupt InterruptHandlerLow
void InterruptHandlerLow ()
{
....;
}
```

Sur un pic18, il ne peut y avoir que deux fonctions d'interruptions

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Timing boucle infinie / interruptions



(Voir cours temps reél)

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Timing boucle infinie/interruption

- Un tâche doit être effectuée le plus rapidement possible car son exécution retarde l'exécution des autres tâches.

  Temps de réponse d'une tâche <= Temps de parcours de la boucle générale. (Tboucle)

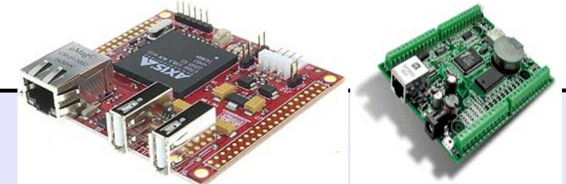
  Tboucle n'est pas constant.

- Dans la boucle infinie, on peut manquer des évènements dont la durée est inférieure au temps de boucle !

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

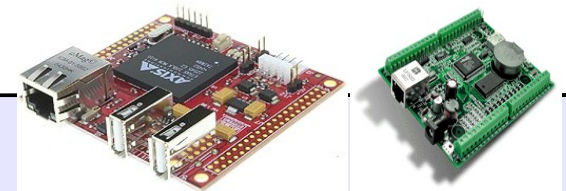
Programmation boucle infinie/interruptions

- Les actions "urgentes" sont traitées par interruption.
- Les routines d'interruptions sont les plus courtes possibles.
- Les traitements dans la boucle infinie (tâches) sont les plus courts possibles car une tâche "longue" retarde toutes les autres.
- Lorsqu'une tâche se bloque, elle bloque toutes les autres tâches : Il faut un mécanisme logiciel pour attendre des événements sans boucle while (ni for) : *machine d'états*.
- Si une tâche boucle accidentellement, le système se fige. Un *watch dog* matériel permet de resetter le système.
- L'initialisation doit également être très courte en cas de reset par un watch dog.



© Copyright 2005, Philippe Arlotto
Creative Commons Attribution-ShareAlike 2.0 license

Attendre sans boucler



© Copyright 2005, Philippe Arlotto
Creative Commons Attribution-ShareAlike 2.0 license

Attendre sans boucler

Dans beaucoup de systèmes on doit *attendre* un événement avant de passer à la suite des opérations.

Souvent les spécifications s'énoncent ainsi :

Mettre le four en marche jusqu'à ce que la température atteigne 180°C

Attendre l'arrêt complet (vitesse nulle) avant d'ouvrir la porte

Mettre le moteur en marche
Attendre deux minutes
Arrêter le moteur

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Attendre sans boucler

Attendre l'arrêt complet (vitesse nulle) avant d'ouvrir la porte

```
while(vitesse()!=0);
PORTBbits.RB3=1;    // ouvrir porte
```

Mettre le four en marche tant que la température n'a pas atteint 180°C

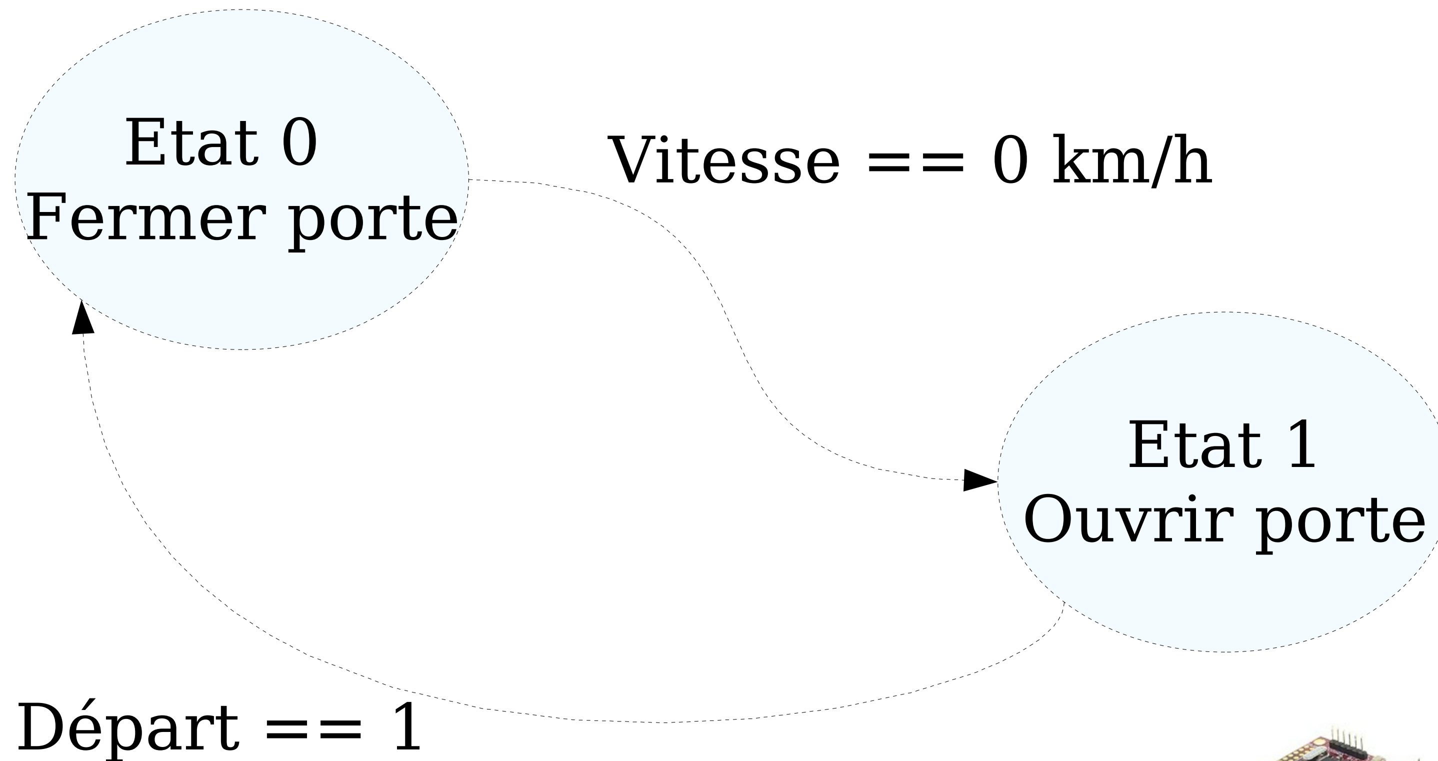
```
PORTBbits.RB2=1;  // four en marche
while(temperature()<180);
PORTBbits.RB3=0;  // arrêt four
```

Avec ces lignes de programme les autres tâches seraient bloquées !

© Copyright 2005, Philippe Arlotto
Creative Commons Attribution-ShareAlike 2.0 license

Attendre sans boucler

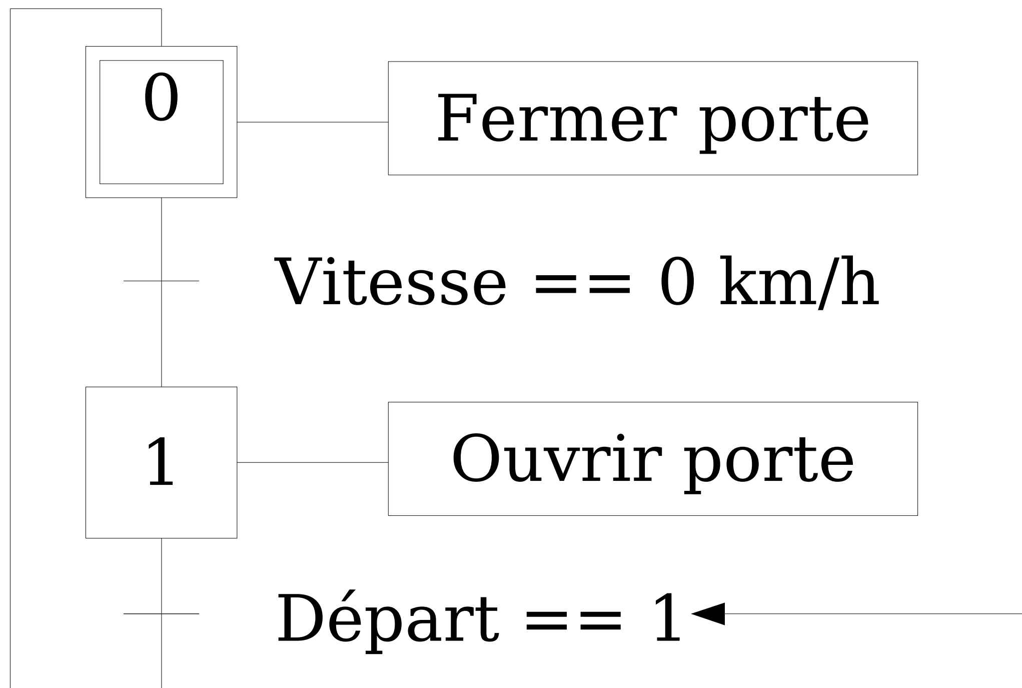
Une solution est de créer une *machine d'état* que ne passe à l'état suivant que lorsque la condition est réalisée



© Copyright 2005, Philippe Arlotto
Creative Commons Attribution-ShareAlike 2.0 license
http://arlotto.univ-tln.fr

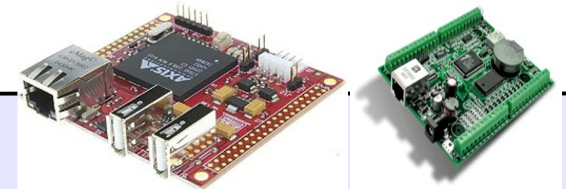
Attendre sans boucler

En france, on a coutume d'utiliser la représentation GRAFCET.



Un Etat est appelé ici étape

Une transition est une équation logique placée entre deux étapes



© Copyright 2005, Philippe Arlotto
Creative Commons Attribution-ShareAlike 2.0 license

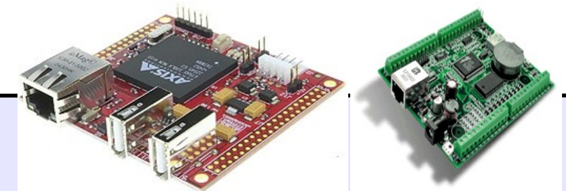
Règles d'évolution d'un grafcet

Lorsqu'une étape possède plusieurs étapes suivantes,
on s'imposera la condition :

Produit des transitions = 0

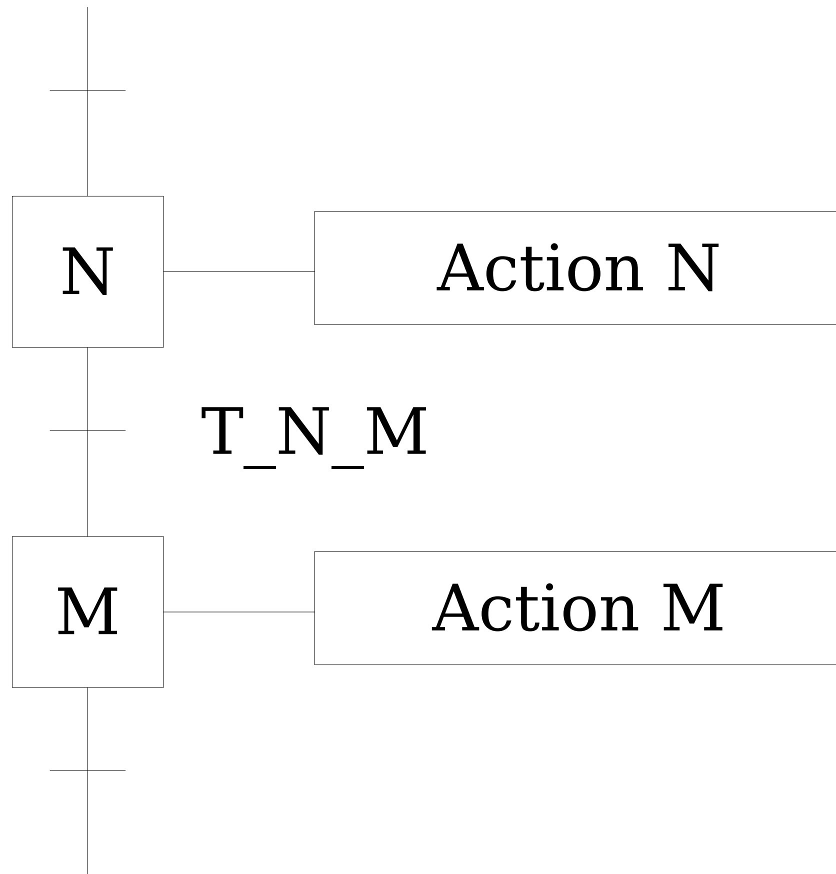
Ainsi :
on est forcé à examiner tous les cas lors de la
conception du grafcet.

Lors de la programmation l'ordre d'évaluation des transitions
n'aura pas d'importance.

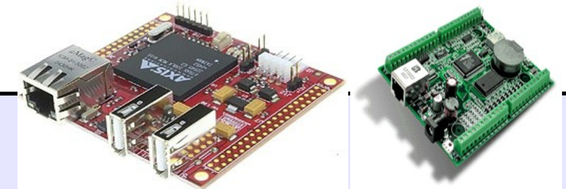


© Copyright 2005, Philippe Arlotto
Creative Commons Attribution-ShareAlike 2.0 license
http://arlotto.univ-tln.fr

Règles d'évolution d'un grafcet



Lorsque l'étape N est active et que la transition T_N_M est vraie :
L'étape N est désactivée et l'étape M devient active



© Copyright 2005, Philippe Arlotto
Creative Commons Attribution-ShareAlike 2.0 license

Divergence en OU

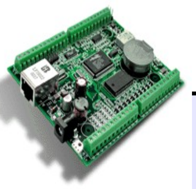
N — Action N

T_N_M . T_N_P = 0

Après l'étape N une seule des deux étapes M ou P peut être active.

T_N_M — M — Action M

T_N_P — P — Action P



© Copyright 2005, Philippe Arlotto
Creative Commons Attribution-ShareAlike 2.0 license

Divergence en ET

La condition T_N_M . T_N_P = 0
n'est plus respectée.

N — Action N

T_N_M — T_N_P

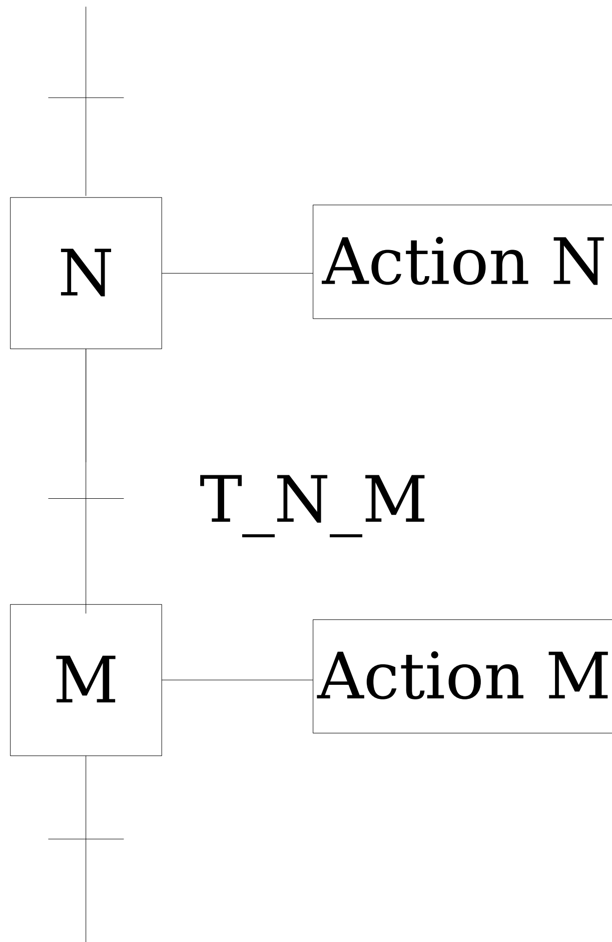
M — Action M

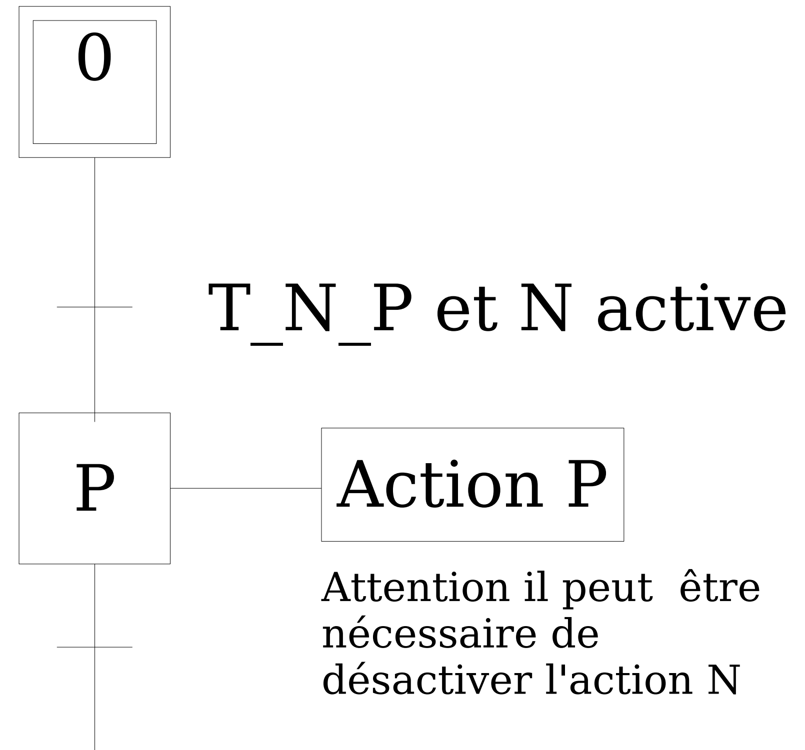
P — Action P

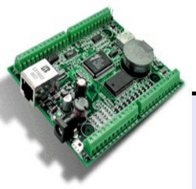
Après l'étape N les deux étapes M ou P peuvent être actives.
Ce cas n'est pas traitée par la méthode suivante.
On préférera faire deux grafcets.

© Copyright 2005, Philippe Arlotto
Creative Commons Attribution-ShareAlike 2.0 license

Divergence en ET simulée



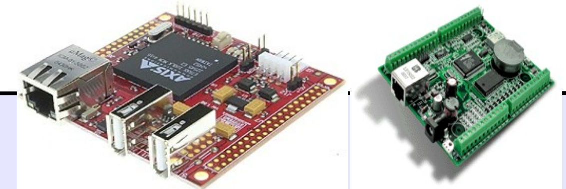




© Copyright 2005, Philippe Arlotto
Creative Commons Attribution-ShareAlike 2.0 license

Codage d'un grafcet en C

- Une variable (entière) contient le numéro de l'étape courante (en général initialisée à 0)
- Le grafcet est codé dans une structure switch...case
  Un cas par étape + un clause default en cas de problème logiciel.
- Le traitement d'une étape est effectué par un cas de la structure switch..case.
- Le grafcet "avance" éventuellement d'une étape à chaque exécution du switch...case (une fois par tour de boucle générale).



© Copyright 2005, Philippe Arlotto
Creative Commons Attribution-ShareAlike 2.0 license
http://arlotto.univ-tln.fr

Codage d'un grafcet en C

```
int etape = 0 ; //initialisation
....;
for( ; ; ) {
....;
switch (etape) {
  case 0 :
           // traiter étape 0 ;
            break ;
  case 1 :
           // traiter étape 1 ;
            break ;
  case N :
            .......;
            break ;
  default : // erreur !!
            .......;
}
....;
}
```

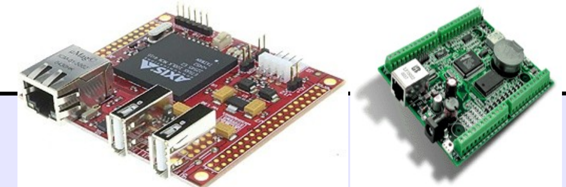
© Copyright 2005, Philippe Arlotto
Creative Commons Attribution-ShareAlike 2.0 license

Codage d'un grafcet en C

Traitement d'une étape :

```
case N :
     action N ;
     if ( T_N_M ) {
       etape = M ; }
     else {
       if ( T_N_P ) {
          etape = P ; }
     }
     break ;
```

On exécute l'action de l'étape courante et ensuite on évalue les transitions vers les étapes suivantes. Si une transition est vraie on change simplement la valeur de la variable qui contient le numéro de l'étape courante. L'étape suivante sera alors active au prochain tour de boucle générale.

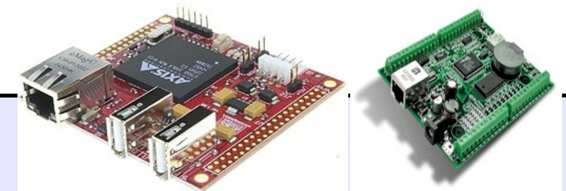


© Copyright 2005, Philippe Arlotto
Creative Commons Attribution-ShareAlike 2.0 license

clause default

Normalement on ne peut pas arriver dans la clause default.
Si on y arrive c'est que la variable etape est corrompue !
C'est alors est une défaillance critique du logiciel.

Il faut :
mettre le système en sécurité, prévenir, redémarrer si nécessaire.
Ce cas doit être prévu dès la conception.



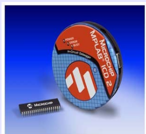
© Copyright 2005, Philippe Arlotto
Creative Commons Attribution-ShareAlike 2.0 license

Les Périphériques du 18F452

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Périphériques du 18F452

- 5 ports d'entrées sorties logiques (jusqu'à 34 broches E/S)
- 4 timers (2 de 8bits , 2 de 16bits)
- 2 modules CCP : Capture , Compare, Pwm (génération et mesures de signaux)
- 1 port série synchrone (I2C ou SPI)
- 1 port série asynchrone (UART)
(utilisable en USB, RS232 , RS485 avec un adaptateur externe)
- 1 convertisseur analogique numérique 10 bits (jusqu'à 8 canaux d'entrées)
- watch dog , détection tension alimentation basse
- fréquence horloge modifiable par programme
- 256 octets de mémoire EEPROM



© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Entrées/sorties logiques

- Toutes les broches peuvent être configurées indépendamment en entrées ou sorties logiques.
- Au Reset les broches sont toujours configurées en entrées
- Attention le PORTA est partagé avec le module adc.
  Toutes les combinaisons Analogique/Digital ne sont pas possibles. Il est configuré en analogique au Reset.
  Sauf RA4 qui est toujours digitale.
  RA4 : drain ouvert en sortie / trigger de smith en entrée
- On peut activer des résistances de pullup internes sur le PORTB.
  RB6 et RB7 sont dédiées à l'ICD

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Entrées/sorties logiques

- Trois registres importants :
  PORTx : bits représentants l'état réel de lignes d'E/S
  LATx : bits représentants l'état forcé des sorties
  TRISx : bits de configurations :
  1 : entrée (valeur au reset)
  0 : sortie

  Normalement pour une sortie LATx = PORTx sauf dans quelques cas particulier :
  court-circuit entre broches
  drain ouvert
  surconsommation

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Entrées/sorties logiques

- **Entrée**

Configuration :

```
TRISBbits.TRISB0 = 1 ; // inutile au reset
```

Lecture :

```
if(PORTBbits.RB0 == 0 ) {
    ...... ; // action si 1
} else {
    ..... ; // action si 0
```

Ecriture : sans effet
(PORTBbits.RB0 = 1 ;)

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Entrées/sorties logiques

- **Sorties**

  Configuration :

  PORTBbits.RB3 = 0 ;
  TRISBbits.TRISB3 = 0 ; // il vaut mieux fixer l'état avant
  // que la broche soit une sortie

  Ecriture :

  PORTBbits.RB3 = 0 ; ou LATBbits.LATB3 = 0 ;

  Relecture :
  if (PORTBbits.RB3==1) {....
  ou
  if (LATBbits.LATB3==1) {....

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Convertisseur analogique/numérique

- Convertisseur 10 bits à approximations successives
- Jusqu'à 8 canaux d'entrées
- Références :
  haute : Vdd ou externe par l'entrée Vref+ (AN3)
  basse : Vss ou externe par l'entrée Vref- (AN2)
- Valeur obtenue :

  N = E[1023 x (Ve - Vref-) / (Vref+ - Vref-)]

  Avec références 0/5V :   N = E[1023.Ve/5]   (E[x]:partie entière de x)
- Durée d 'une conversion : 12.Tad   (Tad = k . Tosc)
- Attention à respecter le temps de charge de CHOLD après un changement de canal

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Convertisseur analogique/numérique

- Initialisation

  Choisir la configuration des broches :
  Analogique/Digital/Référence voir ADCON1
  (chap. 17 doc 18FXX2)
  En déduire la constante à passer à la fonction OpenADC :
  Par exemple : ADC_1ANA_0REF

  Calculer la fréquence de l'horloge du convertisseur :
  il faut avoir TAD >= 1.6µs avec TAD = k.Tosc
  Choisir le plus faible k parmi 1,2,4,8,16,32 ou 64
  En déduire la constante à passer à la fonction OpenADC
  Par exemple : ADC_FOSC_8

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Convertisseur analogique numérique

- Sélectionner le canal
  Soit à l'initialisation par OpenADC()
  Par la suite par la fonction SetChanADC()
- Attendre le temps d'acquisition (charge de CHOLD)
  Par exemple : Delay1TCY(30); (30µs à Fosc=4Mhz)
- Lancer la conversion : ConvertADC();
- Attendre la fin : while(BusyADC()); (ou 12TAD)
- Lire le résultat : int Result ;
  Result = ReadAC();
- Pour économiser l'énergie on peut l'arrêter par CloseADC();
  (attention il faudra refaire OpenADC() pour l'utiliser à nouveau)

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Convertisseur Analogique/Numérique

Voir un exemple pour la carte picdem2+ à :

http://arlotto.univ-tln.fr/pic/pic18/adc/

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Timers

Un timer est un compteur 8 ou 16 bits préchargeable
dont l'horloge Th peut être :

Soit dérivée de l'horloge principale du PIC
(Tosc ou Kp.Tosc si on utilise un prédiviseur)

Soit fournie par un signal externe
(broche RA4/T0CKI pour le timer0)

Soit fournie par un oscillateur interne supplémentaire
(oscillateur du Timer1 (ou Timer3) avec quartz externe)

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Timers

Le passage de la valeur maximum (0xFF ou 0xFFFF) à 0 est appelé overflow.
L'overflow provoque le passage à 1 du bit TMRxIF.

Une interruption est disponible sur l'overflow.

On peut lire/écrire une valeur dans le compteur.
(Précautions nécessaires pour les compteurs 16bits)

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Timers

La durée entre deux overflow est :

Toverflow = $(2^n$ – valeur initiale). Th (n = 8 ou 16)

On peut donc :
produire des interruptions à intervalle régulier pour réaliser des temporisations (cf chapitre Gestion du Temps)

produire une interruption un certain temps après un événement.

mesurer le temps entre deux événements 1 et 2.
A t1 on lit C1 dans le compteur, à t2 on lit C2
Après la lecture à t1 on compte les overflow et on a :
t2 -t1 = (C2 -C1).Th + (nombre d'overflow).Toverflow

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Exemple

Le timer 1 est un timer 16 bits. Avec une horloge externe à 32.768 kHz et sans prédiviseur on a :

Toverflow = 2^{16} . 1/32768 = 2s

En rechargeant la valeur 32768 après chaque overflow on a :

Toverlfow = (2^{16} – 32768). 1/32768 = 1s

En rechargeant la valeur 49152 après chaque overflow on a :

Toverlfow = (2^{16} – 49152). 1/32768 = 0.5s

Voir la mise en oeuvre à :
http://arlotto.univ-tln.fr/pic/pic18/timers/tempo_timer1.c

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Générer un signal PWM

Deux modules CCP (capture, Compare, Pwm) peuvent être utilisés pour générer deux signaux PWM de même période et de rapport cyclique indépendant.

Module CCP1 : sortie pwm sur RC2.

Module CCP2 : sortie pwm sur RC1 (ou RB3 en fonction de l'état d'un bit de configuration).

La période est fixée par les overflow du timer2.

Le temps à l'état haut est modifiable par la fonction SetDCPWMx

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Générer un signal PWM

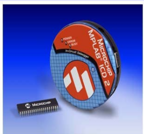
On souhaite générer un signal pwm de période **T = 1ms**
et de **rapport cyclique 35,8%** sur RC2.
Le pic à un quartz de **4Mhz**.

RC2 => Module CCP1 => fonctions indicées 1

Calcul de la valeur à passer à la fonction OpenPWM1 :

T = (period+1).4.Tosc.K2 (K2=prédivision du timer2 1,4 ou 16)

Avec K2 = 1 => period+1 = 1ms/1µs = 1000 > 256 impossible
Avec K2 = 4 => period + 1 = 1ms/4us = 250 < 256 possible
Avec K2 =16 ce sera donc possible mais résolution plus faible
on choisit donc **K2=4, period = 249**



© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Générer un signal PWM

Rapport cyclique : 35,8 % => Temps à l'état haut = 0.358ms

0.358 ms = dutycyle . Tosc . **K2** = dutycyle . 0.25us . **4**

Soit dutycyle = 358 (<= 1023 donc possible)

Pour 100% on aurait 1000us = dutycyle . 0.25us . 4
soit dutycycle = 1000

Programmation :

```
OpenPWM1(249);
OpenTimer2(TIMER_INT_OFF &  T2_PS_1_4 );
SetDCPWM1(358);
```

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Générer un signal PWM

Par la suite on peut modifier le rapport cyclique par un nouvel appel à SetDCPWM1()

Voir un exemple plus simple (sans prédiviseur) à :

http://arlotto.univ-tln.fr/pic/pic18/pwm/pwm.c

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Module USART

Associé à un circuit MAX232 ou FDTI FT232R par exemple, le module UART du Pic permet la communication RS232 ou USB.

Utilisé seul il permet la communication série asynchrone ou synchrone avec d'autres circuits alimentés en 5V.

Il permet d'envoyer et de recevoir des **octets**.

Nous n'utiliserons que le mode **asynchrone** (RS232)

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Module USART

Initialisation :

Vitesse : deux possibilités :

High Speed
Vitesse en bit/s = Fosc / (16 spbrg + 1)

Low Speed
Vitesse en bit/s = Fosc / (64 spbrg + 1)

Faire les deux calculs de spbrg. Si deux valeurs sont possibles (0-255) choisir celle qui donne le moins d'erreur.

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Module USART

Exemple : 9600 bit/s à Fosc = 4MHz

High Speed : USART_BRGH_HIGH

spbrg = 25 erreur = +0.16%

Low Speed : USART_BRGH_LOW

spbrg = 6   erreur = -6.99%

On choisit donc High Speed

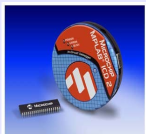
© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Module USART

Choisir USART_CONT_RX pour recevoir des caractères en continu.

Les autres possibilités sont évidentes :

```
OpenUSART( USART_TX_INT_OFF &
USART_RX_INT_ON &
USART_ASYNCH_MODE &
USART_EIGHT_BIT &
USART_CONT_RX &
USART_BRGH_HIGH,
25 );
```



© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Module USART

émettre un caractère (octet quelconque):

```
  char c ;

  c = 'A' ;

WriteUSART(c);
```

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Module USART

Pour utiliser printf, il suffit que la sortie standard soit dirigée sur le module USART.

```
stdout = _H_USART ;
printf("Coucou !\r");
```

C'est le cas au reset.

Si on a redirigé la sortie sur le lcd par exemple (stdout = _H_USER;) il suffit de la rediriger vers l'USART pour que printf sorte à nouveau sur le module USART.

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Module USART

Réception (souvent réalisée sous interruption) :

```
char c ;


if(DataRdyUSART() ) {

            c = ReadUSART() ;
}
```

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Module USART

Voir deux exemples sur :

http://arlotto.univ-tln.fr/pic/pic18/liaison_serie

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

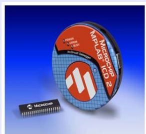
Les chaînes de caractères

En C une chaîne de caractères est une suite de char dont la fin est marquée par le caractère de valeur 0x00 (noté'\0')

```
char Ch[12]="Bonjour !" ;
```

Il existent de nombreuses fonctions C ANSI permettant de manipuler les chaînes de caractères : affichage, recopie, comparaison, concaténation, recherche de motif , etc...

MPLAB C18 fournit la plupart des fonctions standards mais du fait de la structure mémoire du pic il y a plusieurs fonctions C18 pour une même fonction C ANSI.



© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Fonctions C18 sur les chaînes de caractères

Il faut utiliser la bonne version d'une fonction selon
la mémoire dans la laquelle sont stockés les arguments :

strcpy : de mémoire data à data

strcpypgm : de mémoire programme à programme

strcpypgm2ram : de mémoire programme à data

strcpyram2pgm : de mémoire data à programme

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Fonctions C18 sur les chaînes de caractères

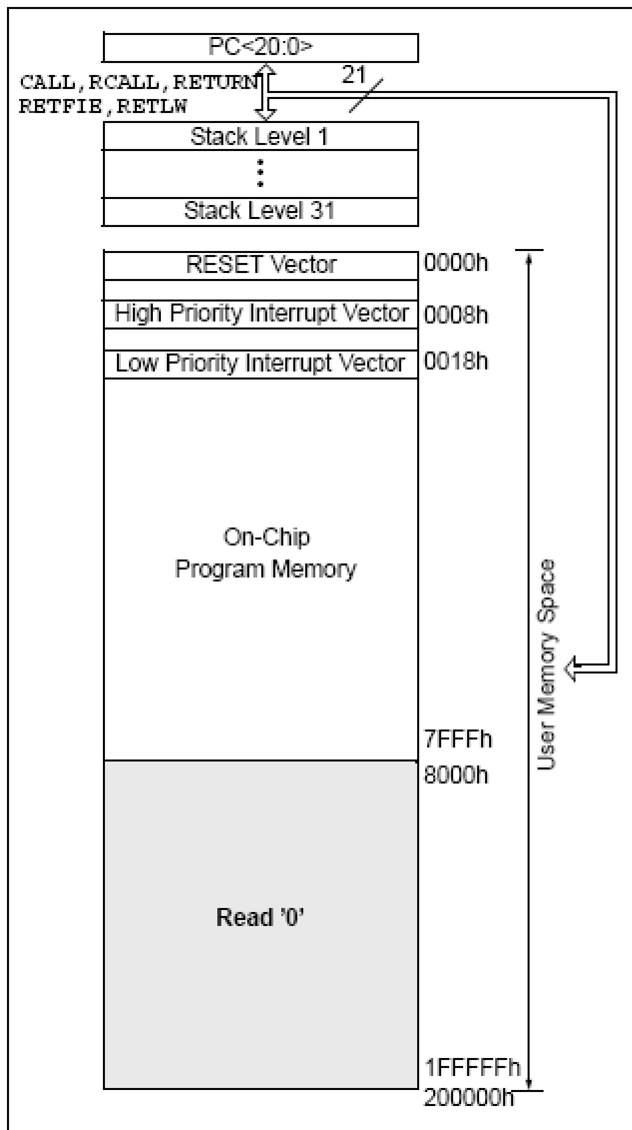
```
// Chaîne stockée en mémoire programme :
const rom char Ch[]="bonjour !" ;

void main(void) {
// Chaîne en mémoire data
 char RamCh[15] ;
  ....;
// Copie de Ch dans RamCh
strcpypgm2ram(RamCh,Ch);

}
```

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Fonctionnement Interne

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

La mémoire programme (Flash)



21 bits d'adresses =
2 Mo d'espace adressable

Sur 18F452 seulement
32ko implémentés
(de 0x0000 à 0x7FFF)

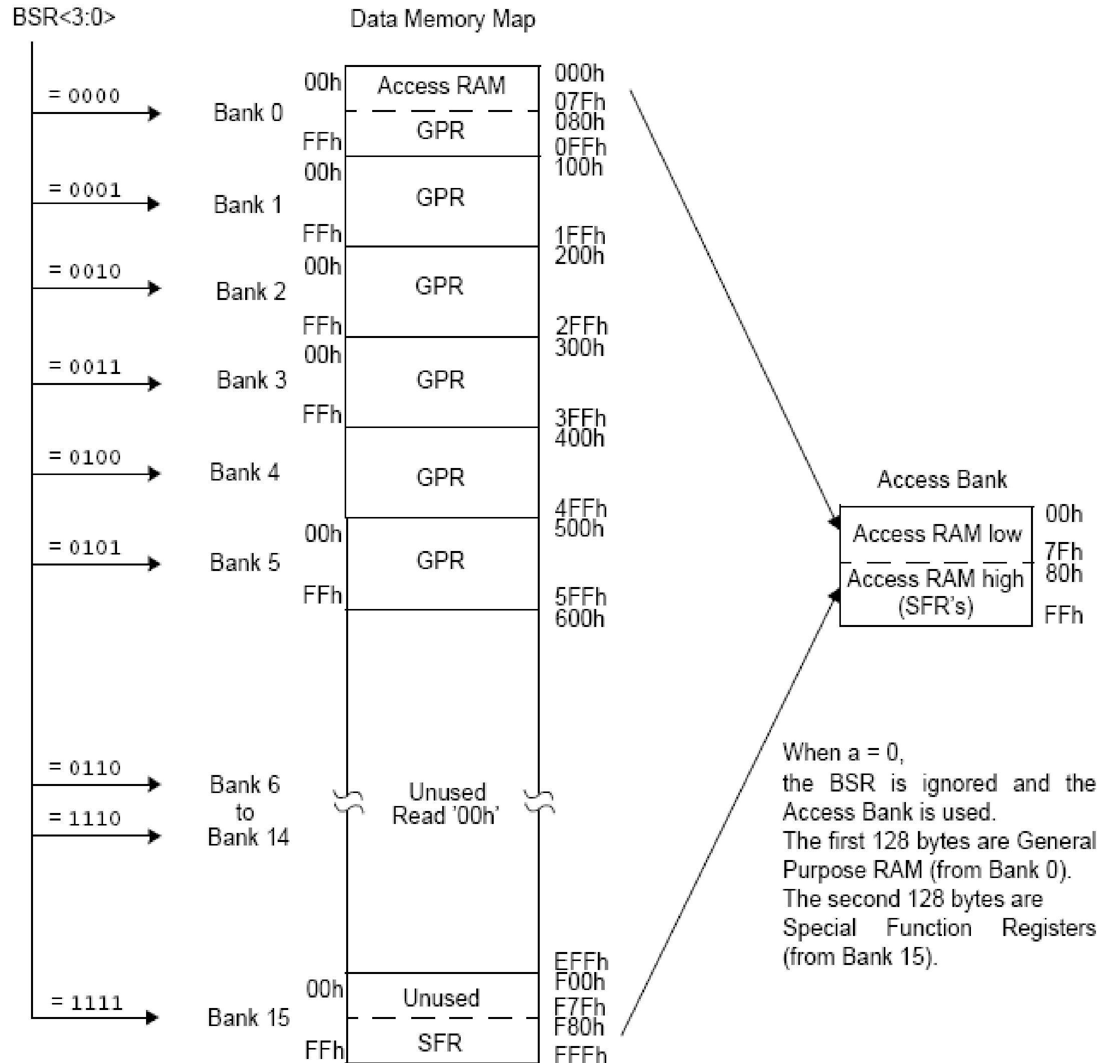
Au Reset PC = 0x000000

Les instructions sont codées sur
16 bits (2 cases).
Les instructions commencent donc
toujours à une adresse paire.

32ko => 16k instructions

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

La mémoire de données (Ram data memory)



12 bits d'adresses
0x000 à 0xFFF

8 bits de données

soit 4ko possibles

Seulement 1536o existent sur le 18F452

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

La mémoire de données (Ram data memory)

Elle est organisée en 16 banques de 256o: bank0 à bank15.
On change de banque lorsque le chiffre héxa de poids fort change :

bank0	0x000 – 0x0FF (256o)	Seules les banques 0 à 5 existent sur le 18F452 6x256o=1536o=1.5ko
bank1	0x100 – 0x1FF	
bank2	0x200 – 0x2FF	
….		
bank15	0xF00 – 0xFFF	

La zone 0xF80 – 0xFFF est allouée aux SFR : registre dédiés à une fonction (port E/S, configuration périphériques, ...)

SFR : special fonction register

GP : General Purpose (case mémoire à usage général)

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

La mémoire de données (Ram data memory)

Le BSR Bank Select Register est un SFR qui contient le poids fort de l'adresse RAM utilisée (la banque).

Un zone particulière de 256o constituée des 128 premiers octets de la banque 0 et des 128 SFR
est appelée ACCESS BANK.
Cette zone permet d'accéder à des registres GP et SFR sans devoir changer de banque
(pas de modification du BSR)

Pour optimiser la durée et la taille des programmes, les compilateurs C essayent de placer les variables le plus souvent utilisées et les résultats de calculs intermédiaires dans la zone ACCESS BANK.

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

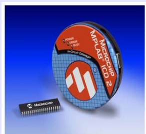
Fonctionnement

Une instruction est d'abord lue en mémoire puis exécutée.

Ce processus prend deux cycles :

- un cycle pour la lecture en mémoire programme (fetch)
- un cycle pour l'exécution et la sauvegarde du résultat.

L'architecture de harvard permettant l'accès simultané aux deux mémoires, le pic exécute un cycle fetch sur la prochaine instruction pendant le cycle d'exécution de l'instruction en cours.
Ainsi la plupart des instructions sont exécutées <u>en un cycle</u>.



© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Relation Cycle / Horloge

Un cycle dure 4 périodes de l'horloge dénommées : Q1,Q2,Q3,Q4.

Tcy = 4 . Tosc

Tosc = 1/Fosc   Fosc : fréquence de l'horloge

Exemple :

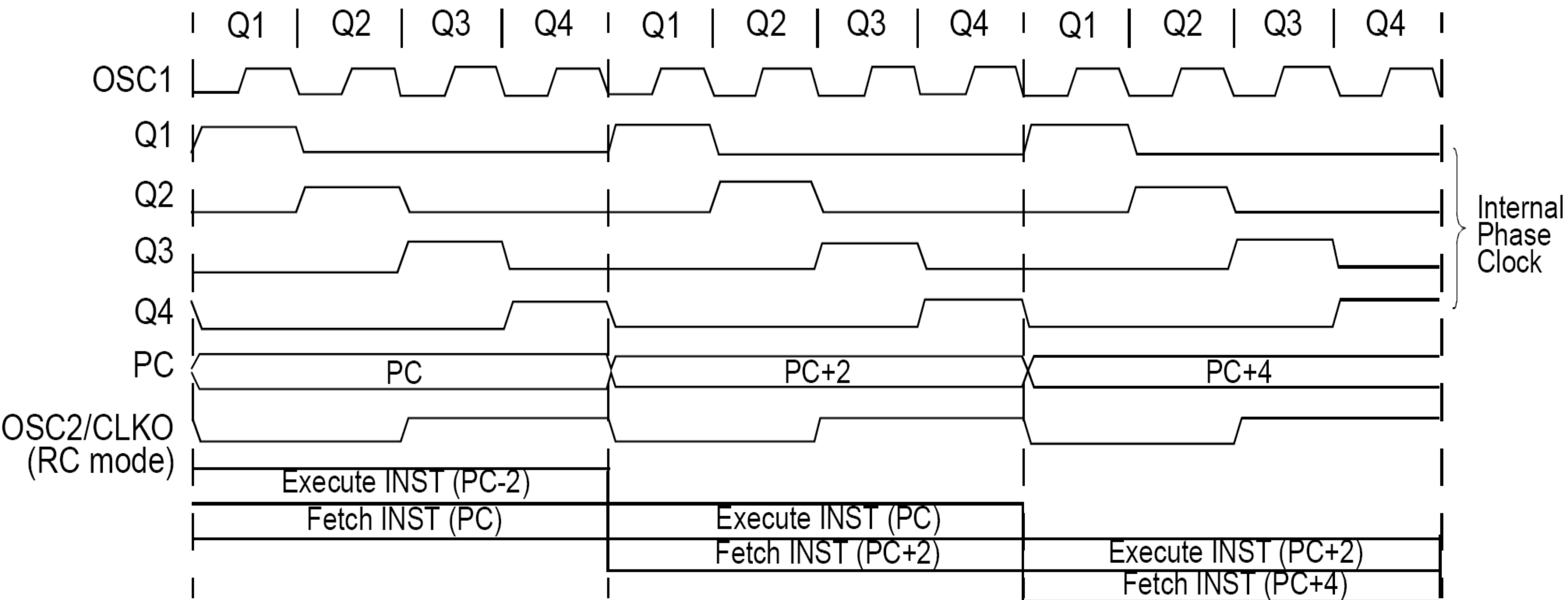
avec un quartz à 4 Mhz , un cycle dure 1µs

Fosc=4Mhz d'où Tosc=0.125µs

et Tcy = 4 x 0.125µs = 1µs

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

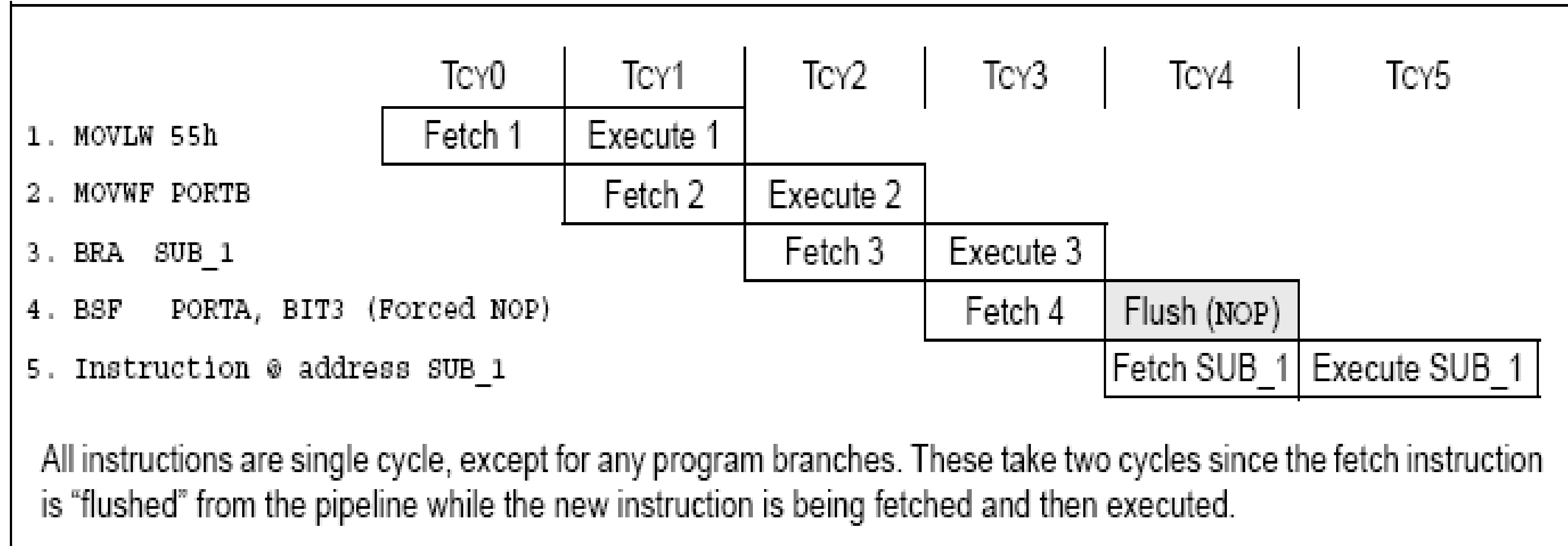
Cycles machines



Un cycle "Fetch" : le PC est incrémenté sur Q1 puis l'instruction est lue (Q2,Q3) et placée dans le registre d'intructions (Q4). A la fin d'un cycle fetch le PC est incrémenté de 2 unités (instruction sur 16bits) et pointe l'adresse de la prochaine instruction.

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Cycles machines


EXAMPLE 4-2: INSTRUCTION PIPELINE FLOW

	TCY0	TCY1	TCY2	TCY3	TCY4	TCY5
1. MOVLW 55h	Fetch 1	Execute 1				
2. MOVWF PORTB		Fetch 2	Execute 2			
3. BRA SUB_1			Fetch 3	Execute 3		
4. BSF PORTA, BIT3 (Forced NOP)				Fetch 4	Flush (NOP)	
5. Instruction @ address SUB_1					Fetch SUB_1	Execute SUB_1

All instructions are single cycle, except for any program branches. These take two cycles since the fetch instruction is "flushed" from the pipeline while the new instruction is being fetched and then executed.

Un cycle d'exécution :
Q1 configuration par le registre d'instruction (décodage)
Q2 lecture de l'opérande (data memory read)
Q3 calcul
Q4 écriture du résultat (destination write)

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

pipeline

Grace à l'architecture de harvard, pendant l'exécution d'une instruction, on lit l'instruction suivante. Ainsi la plupart des instructions s'exécute en un seul cycle.

Rupture de séquence :

Les instructions de branchement (goto, call)
et les instructions de branchement conditionnel provoquent des modifications du PC qui fait que l'instruction suivante qui avait été lue n'est plus valable.
Un cycle (fetch) est nécessaire pour aller lire l'instruction à la nouvelle adresse avant de l'exécuter.
Une instruction vide (nop) est exécutée à la place de l'instruction lue en avance dans le pipeline.

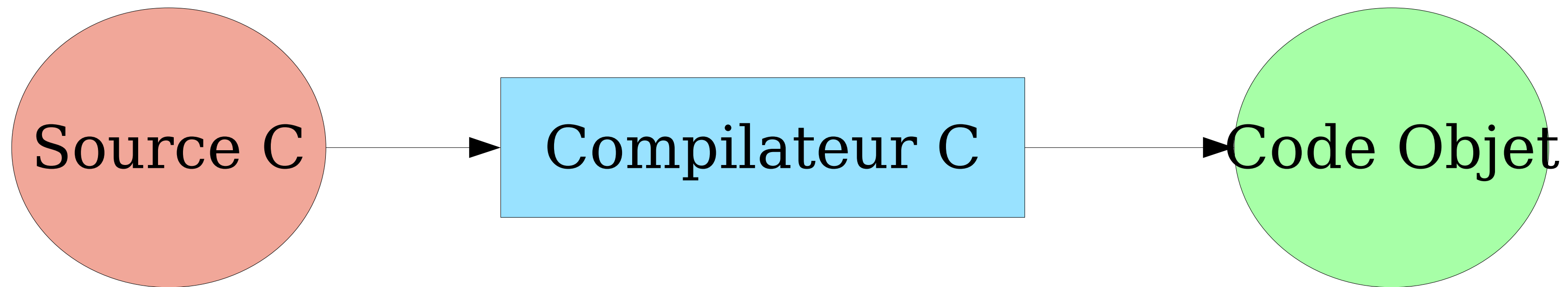
Les instructions qui modifient le PC nécessitent donc 2 cycles.

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Chaîne de développement

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Chaîne de développement

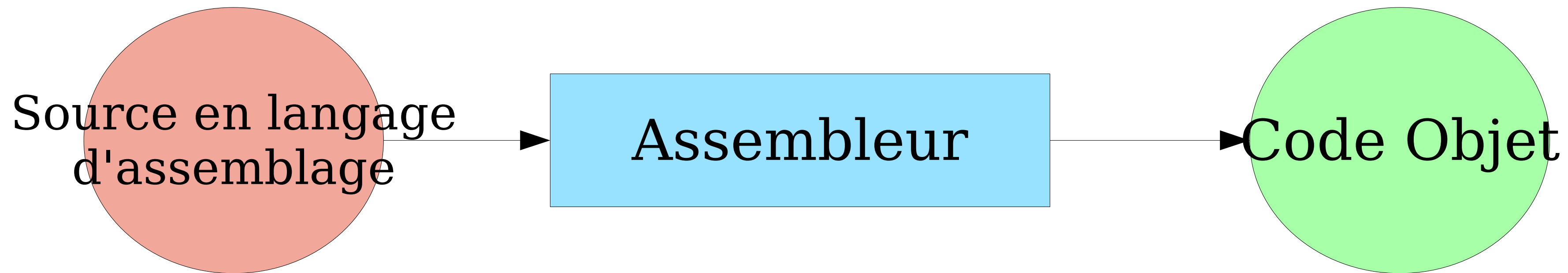



```
int x,y ;
....
if(x!=0){
y=x++;
}
```

Le code *objet* est le code source traduit en instructions de processeur. Ce code n'est pas exécutable car il manque les adresses des différentes zone mémoires (sections). C'est un code dit *relogeable*. (Les adresses réellement utilisées seront fixées lors de l'édition de liens)

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Chaîne de développement




```
Deb :
movlw 5
goto Loop
....
addwf X
subf Y,w
Loop:
```

Le langage d'assemblage est un langage composé d'instructions du processeur écrites en *mnémonique*.

Chaque type de processeurs possède son propre langage d'assemblage.

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Compilateur C

Le compilateur transforme le code source C en code source assembleur avant de produire le code objet.

Source C → Source Assembleur → Code Objet

On peut généralement examiner le code source assembleur généré par un compilateur par l'option -S (stop).

Ex : gcc -S test.c

On estime qu'un compilateur C produit un programme 5 à 10% plus gros et moins rapide qu'un programme écrit en langage d'assemblage par un bon programmeur.

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Editeur de liens

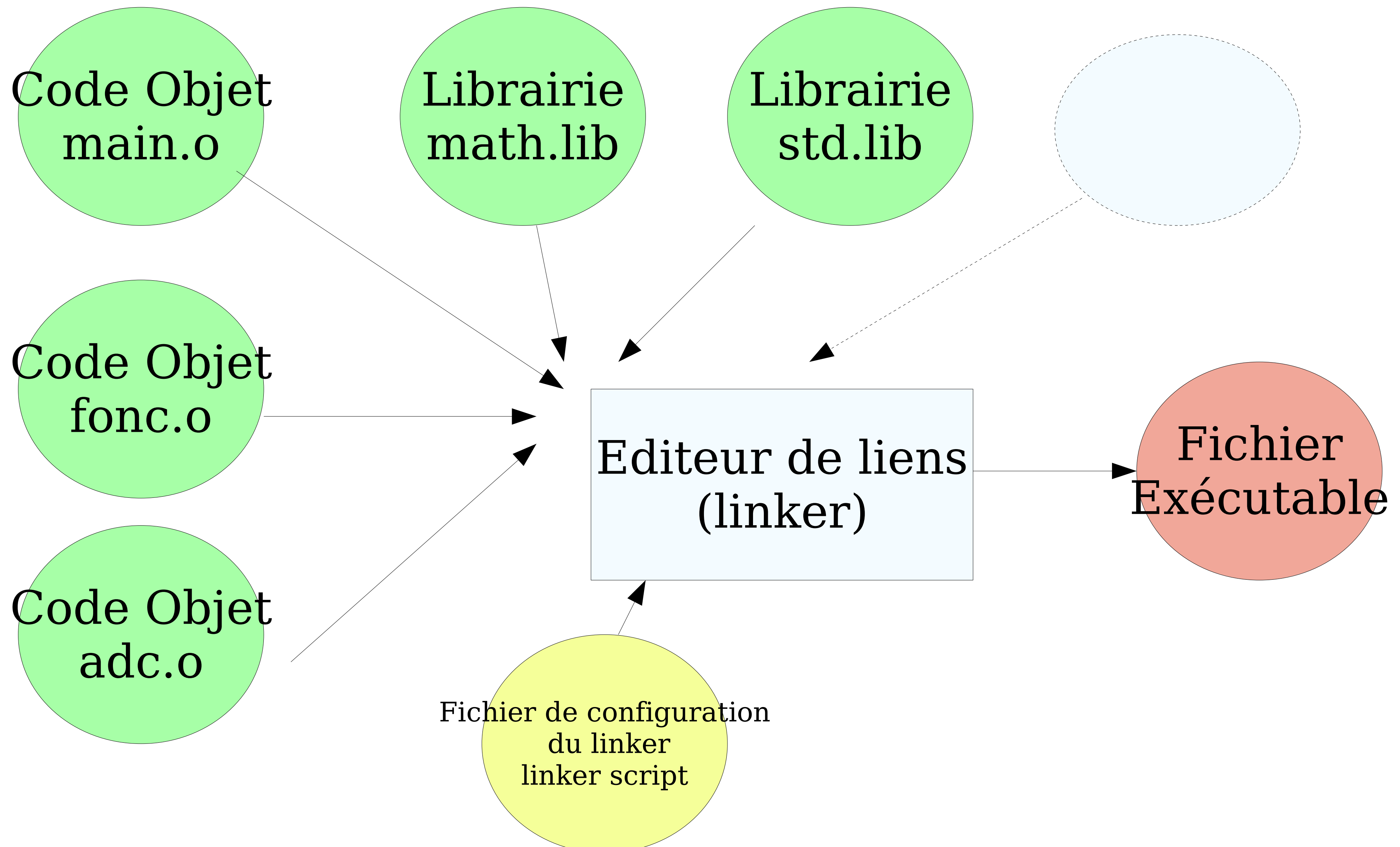
L'éditeur de liens (linker) rassemble les fichiers objets issus des fichiers sources (C , assembleur ou autre) ainsi que des fichiers de librairies (standard ou spécifiques) pour former le fichier exécutable.

Il utilise pour cela les valeurs des adresses des différentes sections (code,data,....) qui sont spécifiques à la carte ou au microcontrôleur utilisé.

Ces adresses lui sont fournies par un fichier de configuration (linker script)

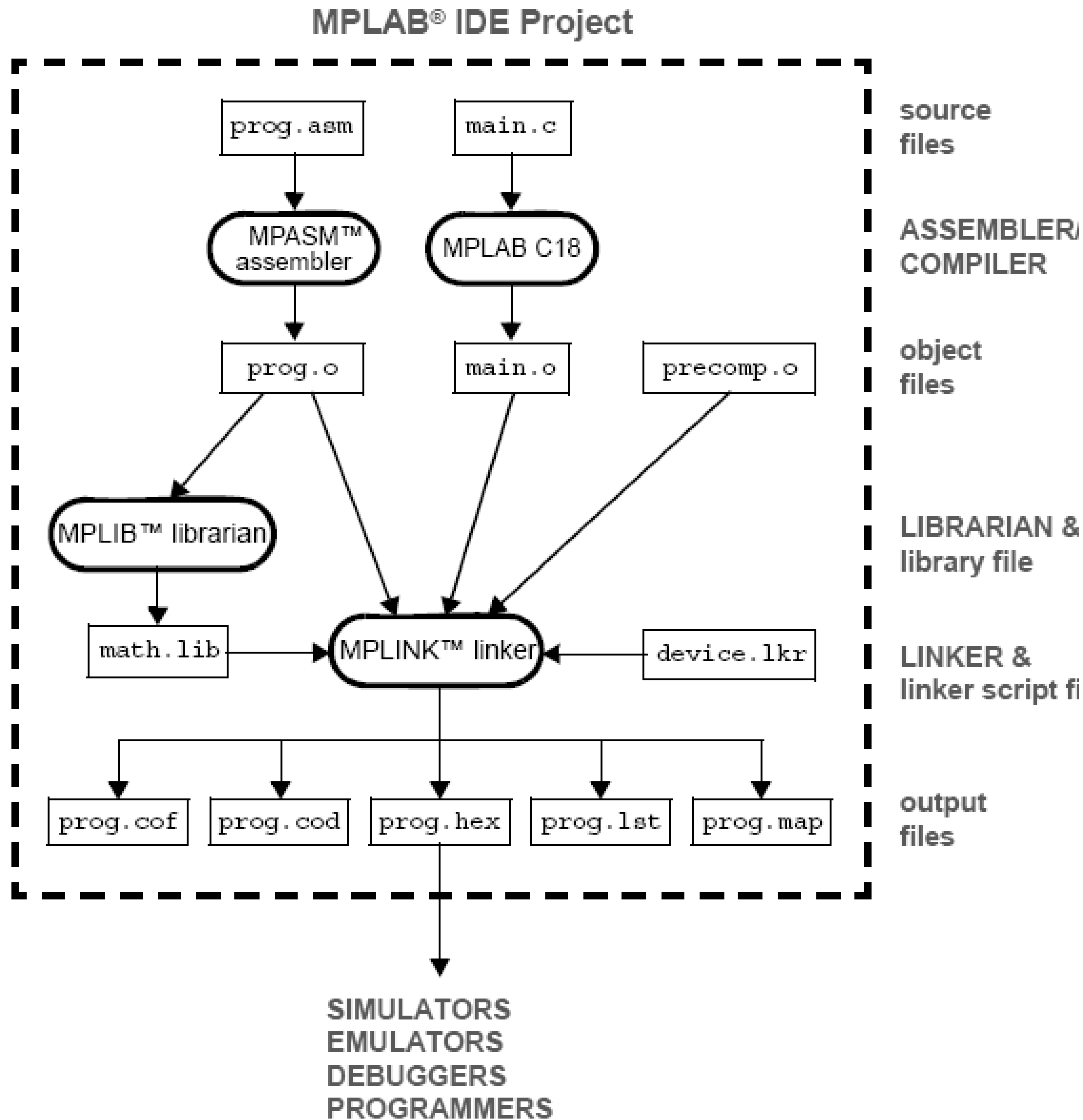
© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Chaîne de développement



© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Chaîne de développement



© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Configuration linker pour 18f452

```
LIBPATH .

FILES c018i.o //code de démarrage
FILES clib.lib  //librairie standard
FILES p18f452.lib //lib librairie spécifique f452

CODEPAGE   NAME=vectors   START=0x0          END=0x29         PROTECTED
CODEPAGE   NAME=page      START=0x2A         END=0x7FFF
CODEPAGE   NAME=idlocs    START=0x200000     END=0x200007     PROTECTED
CODEPAGE   NAME=config    START=0x300000     END=0x30000D     PROTECTED
CODEPAGE   NAME=devid     START=0x3FFFFE     END=0x3FFFFF     PROTECTED
CODEPAGE   NAME=eedata    START=0xF00000     END=0xF000FF     PROTECTED

ACCESSBANK NAME=accessram  START=0x0         END=0x7F
DATABANK   NAME=gpr0      START=0x80         END=0xFF
DATABANK   NAME=gpr1      START=0x100        END=0x1FF
DATABANK   NAME=gpr2      START=0x200        END=0x2FF
DATABANK   NAME=gpr3      START=0x300        END=0x3FF
DATABANK   NAME=gpr4      START=0x400        END=0x4FF
DATABANK   NAME=gpr5      START=0x500        END=0x5FF
ACCESSBANK NAME=accesssfr  START=0xF80       END=0xFFF        PROTECTED

SECTION    NAME=CONFIG    ROM=config

STACK SIZE=0x100 RAM=gpr5
```

section programme (→ CODEPAGE NAME=page)

section Ram data memory (DATABANK gpr1 – gpr4)

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Du C à l'assembleur puis dans le µP

Portions de programme C :

```
char x , y ;
....;
x=3;
y=x + 2 ;
....;
```

Instructions µP

```
movlw   3
movwf   0x000

movf    0x000 , w
addlw    2
movwf  0x001
```

Comme les véritables adresses sont inconnues à ce stade, on prend 0 comme base. par exemple :
0x000 pour x , 0x001 pour y

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Du C à l'assembleur puis dans le µP

```
 x = 3 ;
```

```
movlw    3       déplace (move) la constante (literal) 3 dans w
movwf    0x000   déplace w dans la mémoire (file) d'@ 0x000
```

```
 y = x + 2 ;
```

```
movf 0x000 , w      move file 0x000 (x) dans w    (w=x)
addlw    2          add literal 2 to w result in w  (w=w+2)
movwf   0x001       move w to file 0x001 (y)        (y=w)
```

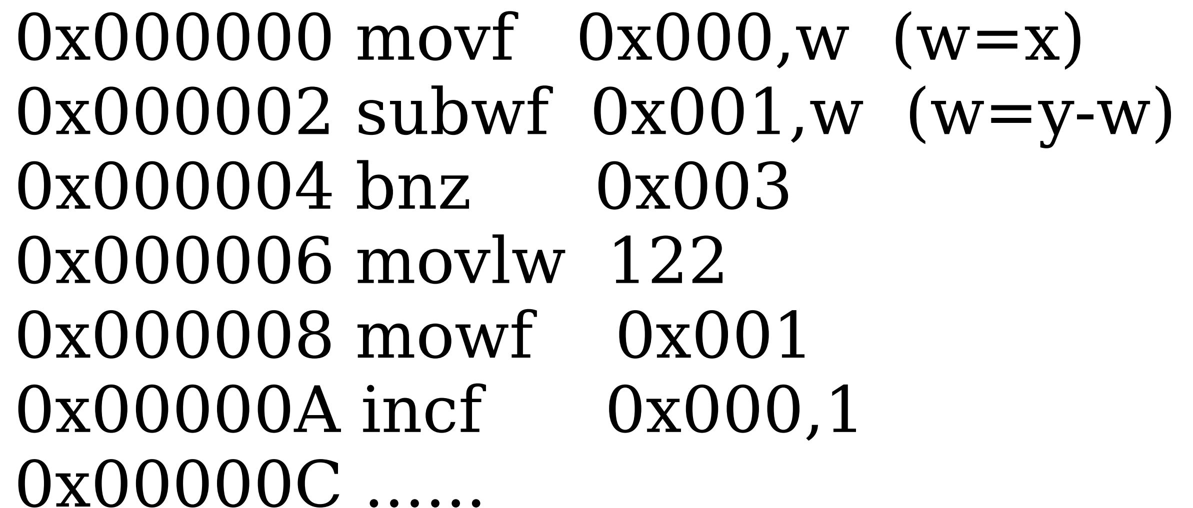
NB : si ces deux instructions C se suivent, l'instruction movf 0x000,w est inutile. C'est le rôle de l'optimiseur de s'en apercevoir et de la supprimer pour rendre le programme plus court et plus rapide.

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Du C à l'assembleur puis dans le µP

```
if( x == y ){
y=122;
x++;
}
```

On va soustraire x de y.
si le bit n°2 (Z) du registre d'état (0xFD8) ne vaut pas 1 on saute les instructions du if (bnz : branch if not Zero)



© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

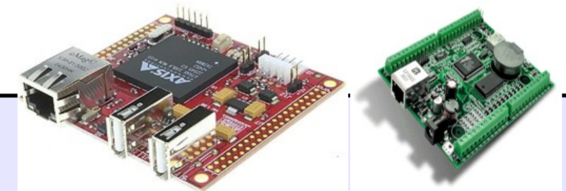
Et dans le µP

Address	Opcode	Disassembly
00E6	**0000**	NOP
00E8	**5100**	MOVF x, W, BANKED
00EA	**5D01**	SUBWF y, W, BANKED
00EC	**E103**	BNZ 0xf4
00EE	**0E7A**	MOVLW 0x7a
00F0	**6F01**	MOVWF y, BANKED
00F2	**2B00**	INCF x, F, BANKED
00F4	**0000**	NOP
00F6	**0000**	NOP

Contenu de la mémoire programme

© Copyright 2005, Philippe Arlotto http://arlotto.univ-tln.fr
Creative Commons Attribution-ShareAlike 2.0 license

Les interruptions



© Copyright 2005, Philippe Arlotto
Creative Commons Attribution-ShareAlike 2.0 license
http://arlotto.univ-tln.fr

Programmation boucle infinie/interruptions

- Les actions "urgentes" sont traitées par interruption.
- Les routines d'interruptions sont les plus courtes possibles.
- Les traitements dans la boucle infinie (tâches) sont les plus courts possibles car une tâche "longue" retarde toutes les autres.
- Lorsqu'une tâche se bloque, elle bloque toutes les autres tâches : Il faut un mécanisme logiciel pour attendre des événements sans boucle while (ni for) : *machine d'états*.
- Si un tâche boucle accidentellement, le système se fige. Un *watch dog* matériel permet de resetter le système.
- L'initialisation doit également être très courte en cas de reset par un watch dog.

© Copyright 2005, Philippe Arlotto
Creative Commons Attribution-ShareAlike 2.0 license

Les interruptions

- Traitement immédiat
  Le traitement associé à l'interruption est effectué directement dans la routine d'interruption.
  Utilisé pour des événements urgents dont le traitement est court.
  Ex : arrêt d'urgence, mesure de temps, temporisations....
- Traitement différé
  La routine d'interruption positionne simplement une variable (globale) qui sera utilisée par une tâche pour effectuer le traitement associé. Le temps de réponse est alors le temps de réponse de la tâche mais l'évènement ne peut être manqué.
  Utilisé pour des évènements à ne pas manquer mais dont le traitement est plus long.
- Traitement mixte (une partie immédiate, une partie différée)

© Copyright 2005, Philippe Arlotto
Creative Commons Attribution-ShareAlike 2.0 license
http://arlotto.univ-tln.fr

Interruptions : Traitement différé

```
unsigned char it;
void main(void) {
    ......;
    for( ;;) {
        ...... ;
        if ( it )
        {
           traitement ;
           it=0;
        }
       ...... ;
           }
}
```

```
#pragma code
#pragma interrupt isr
void isr(void)
{
   if(bit F){
     it = 1 ;
     RAZ bit F ; }
}
```

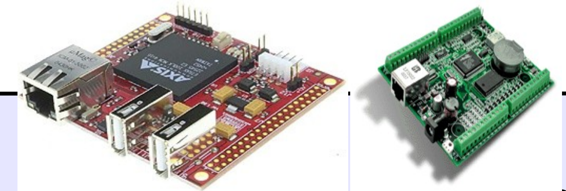
Rq : Si it vaut 1 lorsque, on arrive dans la routine c'est qu'on a manqué le traitement d'une interruption (overflow)
On peut tester it et signaler une erreur.

© Copyright 2005, Philippe Arlotto
Creative Commons Attribution-ShareAlike 2.0 license

Interruptions sur PIC18

Les différentes sources d'interruptions peuvent être soit :

- Désactivées
- Activées et assignées au vecteur de priorité haute (0x000008)
- Activées et assignées au vecteur de priorité basse (0x000018)



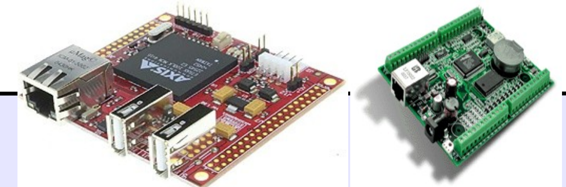
© Copyright 2005, Philippe Arlotto
Creative Commons Attribution-ShareAlike 2.0 license

Interruptions sur pic18

Chaque source d'interruption* possède trois bits qui contrôlent son fonctionnement :

- Un bit F (Flag) qui indique que l'événement s'est produit.
- Un bit E (Enable) qui autorise le programme à se dérouter vers le vecteur d'interruption (qui est fonction du bit P) lorsque F passe à 1.
- Un bit P (Priority) qui sélectionne le vecteur utilisé par la source correspondante (1 : High vector / 0 Low vector)

*sauf INT0 qui n'a pas de bit P (priorité toujours High)



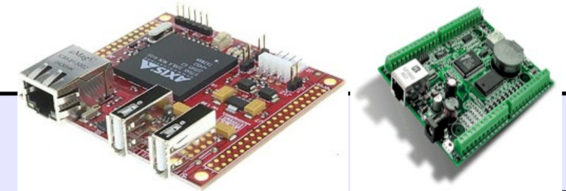
© Copyright 2005, Philippe Arlotto
Creative Commons Attribution-ShareAlike 2.0 license

Interruptions sur pic18

Attention, le bit F ne repasse pas à 0 lorsque l'événement disparaît ou lorsque on rentre dans la routine d'interruption.

C'est au programmeur de penser à le remettre à 0 dans la routine d'interruption.

Si ce n'est pas fait la condition d'interruption est toujours considérée active et il n'y a pas de retour dans la boucle générale.



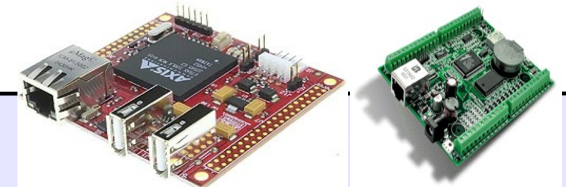
© Copyright 2005, Philippe Arlotto
Creative Commons Attribution-ShareAlike 2.0 license

Interruptions sur pic18

Le pic possède deux modes de fonctionnement :

RCONbits.IPEN=0 ;
Mode par défaut compatible Mid-Range (PIC16) :
Les priorités ne sont pas utilisées (bits P sans effet).
Seul le vecteur High (0x000008) est utilisé.
Les interruptions sont autorisées par INTCONbits.GIE et INTCONbits.PEIE (cf DS39564B ch 8.0 fig 8-1)

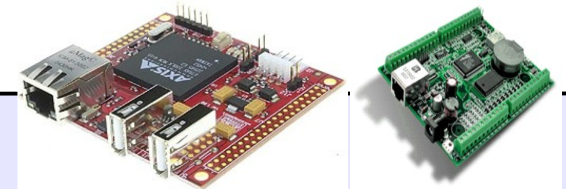
RCONbits.IPEN=1 ;
Les bits P fixent le vecteur utilisé par chaque source.
INTCONbits.GIEH autorise les sources assignées au vecteur haute priorité.
INTCONbits.GIEL autorise les sources assignées au vecteur basse priorité.



© Copyright 2005, Philippe Arlotto
Creative Commons Attribution-ShareAlike 2.0 license
http://arlotto.univ-tln.fr

Mise en oeuvre avec C18

- 1/ Dans la séquence d'initialisation, autoriser la source a produire une interruption : Soit en mettant le bit E à 1, soit en passant des paramètres d'activation à la fonction Open du périphérique. Fixer la priorité par le bit P.
- 2/ En fin de séquence d'initialisation, autoriser globalement les interruptions selon le mode par GIE/PEIE ou par GIEH/GIEL.
- 3/ Ecrire une ou deux fonctions d'interruption.
  void isr_high_vector(void);
  void isr_low_vector(void);
- 4/ Placer un branchement (goto) vers les fonctions d'interruption aux adresses des vecteurs 8 et 18.



© Copyright 2005, Philippe Arlotto
Creative Commons Attribution-ShareAlike 2.0 license
http://arlotto.univ-tln.fr

Mise en oeuvre avec C18

La fonction gestionnaire d'interruption doit :

- tester chaque bit F pouvant déclencher l'interruption
- Effectuer le traitement correspondant (rapidement!)
- Remettre à zéro le bit F.

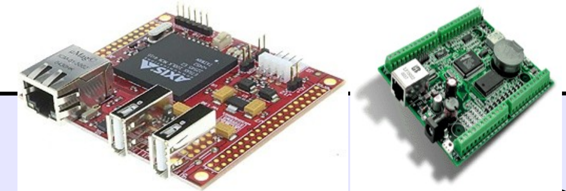
```
#pragma code
#pragma interrupt InterruptHandlerHigh
void InterruptHandlerHigh (){
if (bit F source x ){
    Traitement correspondant à la source x ;
    bit F source x = 0 ; }
if (bit F source y ){
    Traitement correspondant à la source y ;
    bit F source y = 0 ; }
....;
}
```

© Copyright 2005, Philippe Arlotto
Creative Commons Attribution-ShareAlike 2.0 license
http://arlotto.univ-tln.fr

Placement des branchements

```
#pragma code InterruptVectorHigh = 0x08
void
InterruptVectorHigh (void)
{
  _asm
    goto InterruptHandlerHigh //jump to interrupt routine
  _endasm
}
```

Idem pour le vecteur 0x18

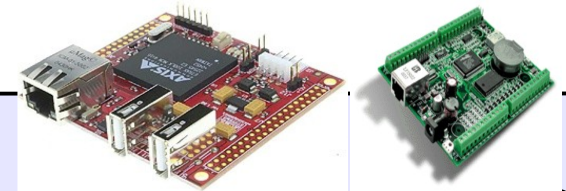


© Copyright 2005, Philippe Arlotto
Creative Commons Attribution-ShareAlike 2.0 license

Exemples

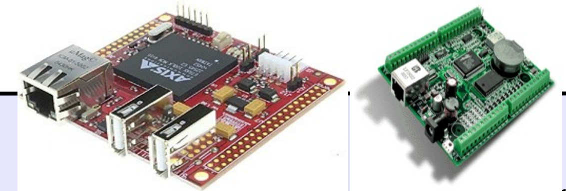
http://arlotto.univ-tln.fr/pic/pic18/interruptions/

http://arlotto.univ-tln.fr/pic/pic18/liaison_serie/



© Copyright 2005, Philippe Arlotto
Creative Commons Attribution-ShareAlike 2.0 license

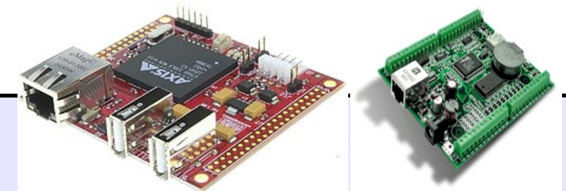
Gestion du temps



© Copyright 2005, Philippe Arlotto
Creative Commons Attribution-ShareAlike 2.0 license

Temporisations

- Temporisations basées sur la durée des instructions
- Temporisations basées sur la durée de la boucle générale
- Temporisations basées sur une interruption périodique



© Copyright 2005, Philippe Arlotto
Creative Commons Attribution-ShareAlike 2.0 license

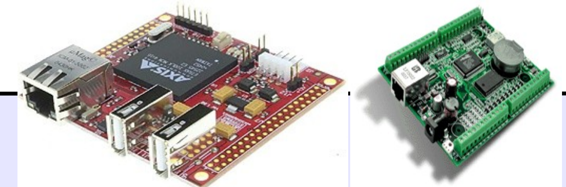
Temporisations basées sur la durée des instructions

Ces temporisations consistent à répéter des instructions jusqu'à obtenir la durée voulue. On peut les réaliser à partir d'une boucle for (ou plusieurs imbriqués pour des durées plus longues) :

```
for (i=0; i <3000; i++) ;
```

Les durées obtenues sont précises et stables (basées sur l'horloge du µP : quartz).

L'inconvénient majeur est de bloquer la suite du programme pendant la durée de la temporisation.
A réserver à des durées <u>très faibles</u> devant le temps de réponse souhaité.



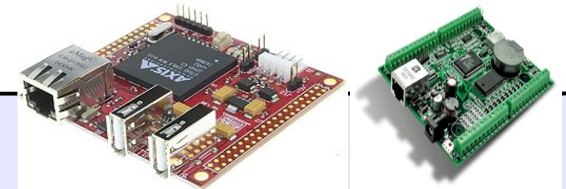
© Copyright 2005, Philippe Arlotto
Creative Commons Attribution-ShareAlike 2.0 license

Temporisation basée sur la durée de la boucle générale

Il suffit de déclencher un traitement une fois tous les N tours de la boucle générale. La durée obtenue est N.Tboucle.

```
unsigned int n ;
n=0;
.....;
for(; ;) {
....;
n++;
if(n>=N) {
   Action périodique ;
   n = 0 ;
}
....;
}
```

C'est simple à mettre en oeuvre mais c'est peu précis car Tboucle varie.



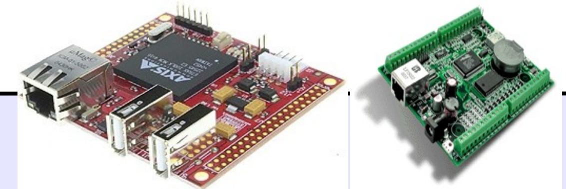
© Copyright 2005, Philippe Arlotto
Creative Commons Attribution-ShareAlike 2.0 license

Temporisations basées sur une interruption périodique

Principe:

Un timer génère un interruption périodique tous les dT.
Dans la routine d'interruption, on incrémente
une variable.
Quand cette variable a variée de N, c'est qu'il s'est écoulé
la durée N.dT.
Ce principe peut être implémenté de différentes manières
selon que l'on souhaite effectuer : des actions périodiques,
des actions à des dates déterminées, des actions
de durées déterminés, ...

C'est simple, précis et non bloquant.



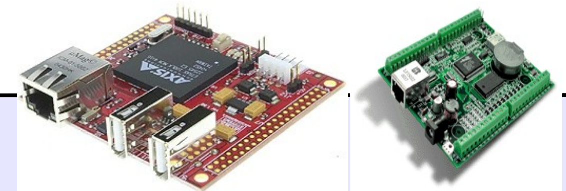
© Copyright 2005, Philippe Arlotto
Creative Commons Attribution-ShareAlike 2.0 license

Temporisations basées sur une interruption périodique

Sur la carte picdem2, un quartz 32.768Khz peut être utilisé avec l'oscillateur du timer 1 pour produire une interruption chaque seconde. En décrémentant un compteur dans la routine d'interruption associée, il est facile de produire des temporisations de durée variable.

Voir un exemple à :

http://arlotto.univ-tln.fr/pic/pic18/timers/



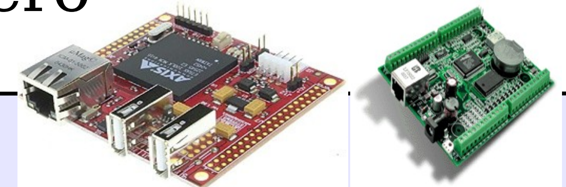
© Copyright 2005, Philippe Arlotto
Creative Commons Attribution-ShareAlike 2.0 license

Une bibliothèque de temporisation

Un temporisation est vue comme l'association d'un *compteur* et d'un *état* (on peut utiliser une structure en C).

l'état peut prendre les valeurs suivantes :
STOP : la temporisation est arrêtée
START : la temporisation démarre
RUNNING : la temporisation est en cours
DONE : la temporisation est échue

Le compteur est une variable entière qui :
est chargée à sa valeur maximale (durée de la temporisation) lorsque la temporisation est dans l'état START.
est décrementée à intervalle régulier dans l'état RUNNING
fait passer l'état à DONE lorsqu'elle vaut zéro



© Copyright 2005, Philippe Arlotto
Creative Commons Attribution-ShareAlike 2.0 license

Une bibliothèque de temporisation

La temporisation est régie par le grafcet suivant qui est exécuté à intervalle régulier :

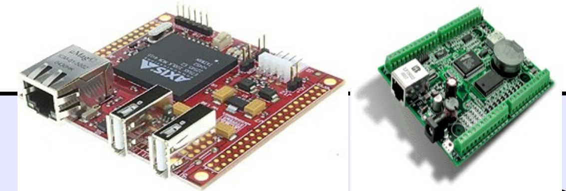
0 STOP
1 START
2 RUNNING
3 DONE

© Copyright 2005, Philippe Arlotto
Creative Commons Attribution-ShareAlike 2.0 license

Une bibliothèque de temporisation

Le grafcet est écrit dans une fonction timer_tick()
qui est appelée régulièrement dans une fonction d'IT,

La condition start est simplement réalisée par l'appel
d'une fonction qui place le grafcet dans l'état START et charge
la valeur maximale dans le compteur.



© Copyright 2005, Philippe Arlotto
Creative Commons Attribution-ShareAlike 2.0 license

Une bibliothèque de temporisation

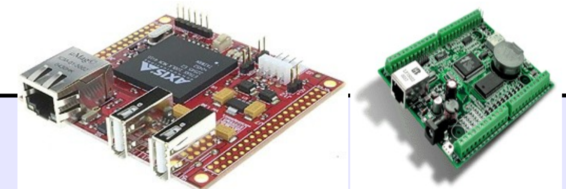
L'interface est réalisée par les fonctions suivantes :

```
void timer_start(int time); // lance la tempo pour time sec
int timer_done(void) ; // retourne vrai si la tempo est échue
```

on peut rajouter les fonctions suivantes :

```
void timer_stop(void) ; // interrompt le décompte de la temporisation.
void timer_restart(void) ; // relance la temporisation
void timer_more(unsigned long time); // ajoute time secondes à une temporisation en cours.
```

En utilisant un tableau de structures et en ajoutant un numéro de temporisation en paramètre de chaque fonction, on peut aisément créer de nombreux temporisateurs indépendants à partir d'une base de temps fournie par un seul timer matériel.



© Copyright 2005, Philippe Arlotto
Creative Commons Attribution-ShareAlike 2.0 license

Une bibliothèque de temporisation

Ces fonctions se révèlent très pratiques :

```
void main(void){
unsigned long time;
int x=0 ;
OpenTimer1(.......);
TRISBbits.TRISB2=0;PORTBbits.RB2=0;
timer_start(5);
for( ;; )
{
....;

 if (timer_done())
 {
   x=1-x ;
   PORTBbits.RB2=x;
   timer_start(5-2*x);
 }
....;  // rien n'est bloquant !!
} }
```

La led 2 clignote 5 sec éteinte / 3 sec allumée

```
void isr(void){
Timer_tick(); }
```

© Copyright 2005, Philippe Arlotto
Creative Commons Attribution-ShareAlike 2.0 license