

Programmation basic pic et picaxe

Ce document est simplement le reflet de ma façon de programmer il n'a pas la prétention d'un cours complet .L'unique but est de partager ce que je sais .(Toute remarque est bien venue .)

Ce document est structuré de la façon suivante

1 /Du français au langage des pic

Une présentation globale sur la traduction du français vers le langage machine .

2/ La programmation structurée

Une méthode de programmation utilisable pour tous les langages

3/ Les premières étapes avant le programme

Avant la programmation proprement dite il est nécessaire de configurer le PIC afin qu'il soit adapté a ce que l'on lui demande de faire .cette configuration est particulière pour chaque pic .

C'est ici que l'on précisera si on utilise un l'oscillateur interne du pic ,que tel ou tel broche sera une entrée tout ou rien ou une entrée analogique ou une sortie .

Pour les **picaxe** cette étape est grandement facilité

4/ Exemple de programme

Un exemple complet détaillé d'un minuteur (de l'idée jusqu'à la réalisation)

4/ L'électronique autour du pic

1/ Du français au langage des pic

Si nous savons écrire avec les 26 signes dits « lettres de l'alphabet » et les 10 signes dits « chiffres », les PIC, comme tous les microcontrôleurs, ne comprennent que le langage binaire. Le courant passe ou le courant ne passe pas.

Le symbole utilisé est 0 : le courant ne passe pas,

Le symbole utilisé est 1 : le courant passe.

C'est l'association de ces 2 symboles, 0 ou 1, qui permet de construire un langage compris par le PIC.

L'association des lettres permet de fabriquer des mots, l'association des 0 et des 1 permet de fabriquer des instructions.

Les mots de notre vocabulaire ne contiennent pas tous le même nombre de lettre mais les instructions de notre pic sont toujours composés du même nombre de signes (disons 8).

Un programme chargé dans le PIC est donc une succession de 0 et de 1 que le PIC sait interpréter.

Soit un programme chargé dans le pic Ce programme est sous la forme 010111110001010110 etc... A cette série de 0 et de 1 correspond, par exemple : j'ouvre ce circuit, je ferme le suivant, etc ...

Ce langage n'est pas très clair pour nous mais c'est le seul compris par le PIC, c'est le langage **binaire**.

On a donc créé un langage un peu plus clair :
On regarde les 8 premiers signes et on associe un symbole :

On associe 00 à 00000000
On associe 01 à 00000001
.....
On associe 09 à 00001001
On associe 0A à 00000101
.....
On associe FF à 11111111

Le programme prend donc la forme 9F A1, c'est le langage **hexadécimal**.

Ce listing est beaucoup plus court que le listing de 01 mais tout aussi incompréhensible pour nous.

A chaque code (code binaire ou hexa) correspond une action. Pour faciliter la programmation, on a fait correspondre un mnémonique à chaque code. Par exemple :

Les broches du pic sont regroupées en port .

Le portA par exemple est le nom donné aux 8 broches nommées portA.0 ,portA.1portA.7

Par exemple la datasheet nous apprend que la broche 18 est le portA.1

Dans le langage assembleur bsf veut dire mettre à 1

L'instruction **bsf porta.1** mettra donc la broche 18 (correspondant à portA1) à 1 (dsoit 5Volts)

Donc : 01011111 est équivalent à 9F, qui est équivalent à **bsf portA.1**

(nota /Le langage assembleur comporte 35 instructions)

Cette façon de programmer n'est pas évidente aussi nous allons utiliser le langage **basic** c'est une 'couche 'de langage qui facilite la vie puisqu'il suffit d'écrire en debut de programme

Symbole led1=portA.1 signifie : on affecte le mot Led1 à la patte 18 du circuit.

Dans le programme la ligne **led1=1** signifiera : on met 5V sur la broche 18 (si une led est connectée elle s'allumera donc).ces 2 instructions seront converties par le compilateur en succession de 0 et de 1

Le BASIC est un langage plus évolué que le langage machine

La compilation transformera ce texte BASIC en programme .hex (qu'il suffira de charger dans le pic qui saura comprendre ce langage.

Les mots du basic permettront de programmer de façon structurée .

Symbole led1=portA.1 est le vocabulaire employé pour les pic

Symbole led1=pinA.1 est le vocabulaire employé pour les picaxe

2/ La programmation structurée : (utilisable pour tous le langages)

En s'imposant certaine contrainte (l'utilisation de structure de base) cela facilite la compréhension des programmes, leur mise au point et leur évolution.

J'utilise 4 structures

L'**itération** (faire ça ,puis ça ,puis ça)

La **boucle** (faire ça N fois)

Le **conditionnel** (faire ça si la condition est vraie)

Le **tant que** (faire ça tant que la condition est vraie

Le ça peut être une instruction ou une suite d'instruction (**sous programme**)

La programmation structurée peut s'appliquer à tout les langages

Programmer structurer c'est donc écrire en français en s'imposant les structures ci dessus puis le traduire dans le langage de programmation choisi.

Les exemples ci dessous utilise le Basic

Programmer en Basic

Le basic est le langage que nous nous proposons d'utiliser pour parler au pic nous allons donc regarder comment l'utiliser et apprendre à manipuler quelques mots et structures essentiels qui nous permettrons de parler ce langage .

Ce langage n'est pas universel ,suivant le basic utilisé (de tel ou tel fabricant) le mots sont légèrement différent comme le vocabulaire de notre langue. On dit un gamin au nord et un gone à Lyon) ,mais rassurer vous les différences sont minimales .

2/Les premières étapes avant le programme sont :

La configuration du pic :

Configurer c'est renseigner les registres du pic (des zones mémoires spécifiques)

En fonction de ce que l'on souhaite faire.

La **définition des variables** utilisées

On donne un nom aux pattes du pic pour que le programme soit plus clair .(il est plus lisible de dire (si l'on a connecté une led la la patte 1 du port B) LED1=1 que de dire porB.0=1 .

(La définition des variables utilisées permet de réserver des emplacements mémoire)

' **configuration du PIC** : le détail des lignes ci dessous est repris en annexe (ces registres permettent de configurer le pic en fonction de notre besoin ,allons nous utiliser l'oscillateur interne ou pas ? etc....)

```
Define CONF_WORD = 0x2f70
```

```
OSCCON = %01100100
```

```
ADCON1 = %00001110
```

```
ADCON0 = %01000001
```

```
OPTION_REG = %00000000
```

'configuration des pattes 'il est nécessaire de dire quelle patte fait quoi

```
TRIS A = %00100011 ' pattes 4 ,17 et 18 en entrée
```

```
TRIS B = %00000000 ' toutes les pattes en sortie
```

La configuration est particulière pour chaque pic ,c'est la data sheet qui précise comment l'établir .

' **donner un nom aux pattes du pic** que nous allons utiliser pour faciliter la programmation

```
Symbol led0 = PORTB.0
```

```
Symbol led1= PORTB.1
```

```
Symbol poussoir = PORTA.1
```

'**definition** des variables utilisées

```
Dim I as byte
```

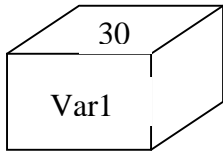
Les lignes ci dessus que j'appelle CONFIGURATION seront placées avant le programme proprement dit (le programme est une suite d'instructions ,de boucles ,etc...)

les variables : (elles sont utilisées pour stocker)

une variable est une boîte qui contient un élément .Elle est définie par son nom est ce qu'elle peut contenir .

Dim var1 as byte signifie que la boîte nommée var1 peut contenir un nombre entre 0 et 255

Var1=30 signifie que var1 contient le nombre 30



si on sait que cette boîte ne peut contenir que 0 ou 1 on écrira **Dim var1 as bit** .
Dim var1 as word permet de stocker une valeur comprise entre 0 65535 et
Dim var1 as long permet de stocker une valeur comprise entre 0 4294967295.

Une variable permet donc de stocker (des nombres dans notre cas)

On peut aussi modifier son contenu

Var1=30 'on initialise la variable

Var1=var1+1 ' on ajoute 1 ,c'est à dire que si la boîte nommée var1 contenait 30 après l'opération (Var1=var1+1)
la boîte contient 31 ,on dit que la variable vaut maintenant 31

Après avoir défini les variables qui seront utilisées dans le programme on peut donc écrire le programme (qui sera une suite d'instructions ou de structures .)

La configuration du picaxe :(elle est beaucoup plus simple)

Il faut préciser quel picaxe on utilise

#picaxe 20m2 'on précise que l'on va utiliser un 20m2

puis on précise quelles sont les entrées et les sorties

dirsb=%11111111 est l'équivalent de TRIS B = %00000000 ' toutes les pattes en sortie
vous pouvez remarquer qu'un 1 correspond à une sortie ,c'est l'inverse des pic (mais c'est pas ma faute !)

*les variables sont prédéfinies elles portent d'origine les noms b0,b1,b2 ... ou w1,w2w3..... et se chevauchent !
explication .*

la variable w0 est composé de b0 et b1

si b1= 0000010 et b0 = 0000111aors w0= 1100000000000111

vous remarquez que modifier b0 ou b1 modifie w0 (et inversement)

si vous voulez stocker un nombre supérieur à 255 vous devrez donc utiliser les variables w

pour stocker le nombre 30 on peut donc écrire b0=30 (on aurait pu écrire aussi w0=30)

pour que le programme soit plus lisible on donne un synonyme à b0 par exemple

symbol var1 =b0 (pour les pic on avait écrit Dim var1 as et Var1=30)

la configuration est terminée !

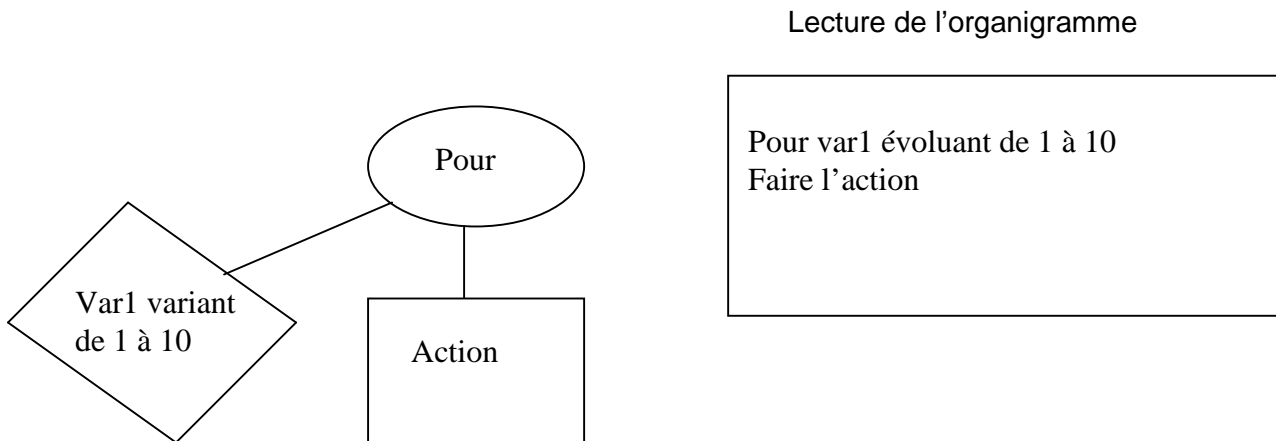
Les structures

(les exemples sont écrits en basic pour les pic)

La structure conditionnelle FOR NEXT

Si l'on souhaite réaliser 10 fois la même chose **en français** on écrit ' **Faire 10 fois l'action** '

Sous forme d'**organigramme** on peut décrire cette boucle de la façon suivant



Pour un pic

...

...

....

Action

Next var1

Pour un picaxe

symbol var1 =b0

...

....

for var1) 1 to 10

Action

Next var1

Ce que fait cette boucle

La variable var1 contient la valeur 1

L'action est réalisée (c'est par exemple allumer une led attendre 1 seconde puis l'éteindre 1seconde)

Le mot next signifie que l'on repart à la ligne 'for Mais var1=2 var1 =2

On réalise la même action action

Puis var1 =3

On réalise la même action action

Puis

Puis var1=10

On réalise la même action action

Fin de la boucle car var1=10

(la led à clignoté 10 fois)

On passe à la suite du programme

Nota :dans cette exemple on à utilisé une variable que l'on a nommé var1 mais très souvent on nomme cette variable i (par convention mais ça ne change rien au programme)

Le programme devient

Exemple de programme utilisant la structure **FOR NEXT**

'pour un pic

```
for l=1 to 10  
led1=1  
waitms1000  
led1=0  
waitms 1000  
next l
```

pour un picaxe

```
for l=1 to 10  
led1=1  
wait 1  
led1=0  
wait 1  
next l
```

Que fait ce programme programme
Ce programme fera clignoter la led 10 fois
(nous en avons profité pour utiliser ne nouvelle
instruction waitms 1000 .Le programme se fige
1000millisecondes donc 1 s .)

Explication

```
i=1  
allumage de la led1  
attendre 1S  
eteindre la led1  
attendre 1S  
i=2  
..
```

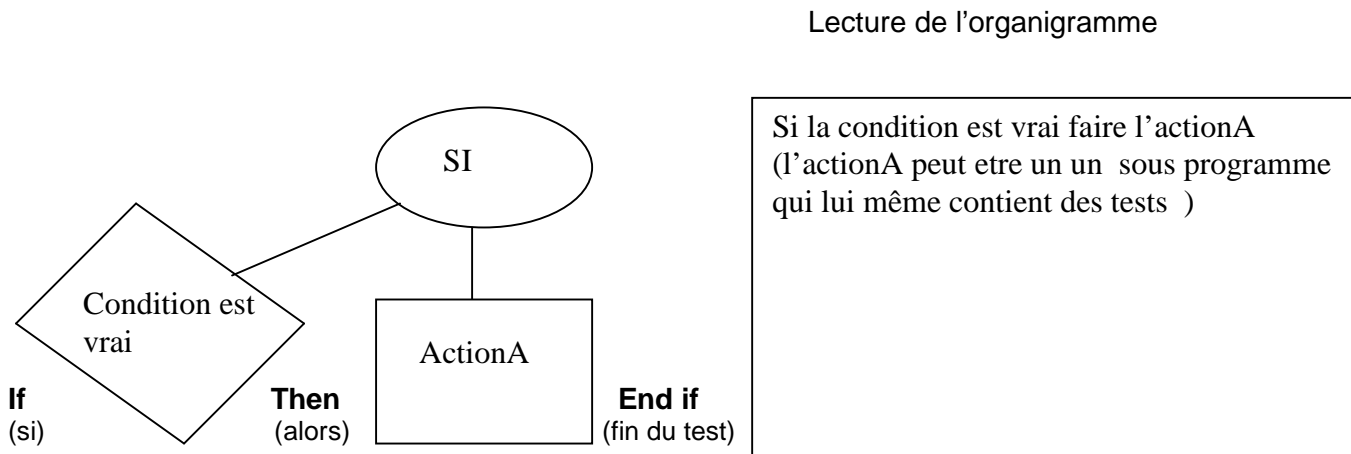
Vérification de la programmation:

Il suffit de verifier qu'il y a bien un next qui cloture la boucle

La structure conditionnelle IF THEN

Si l'on souhaite réaliser une action à condition qu'une condition soit vraie **en français** on écrit ' **Faire cette action si la condition est vraie l'action** '

Sous forme d'**organigramme** on peut décrire cette condition de la façon suivante



En basic

CONFIGURATION

```
..  
....  
....  
'Programme  
pour pic et picaxe
```

```
If var1=2 then  
Action  
End if  
...
```

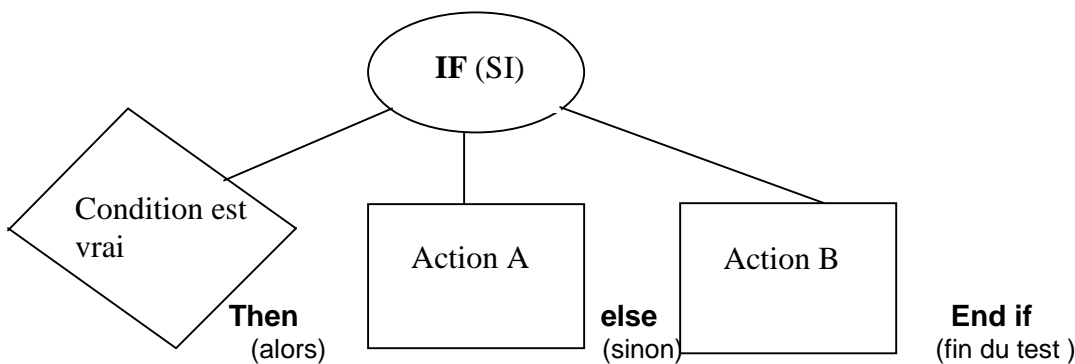
Que fait cette condition

La variable var1 contient une valeur renseignée en cours de programme
Quand le programme arrive à la ligne if
Si var1 contient 2 alors l'Action est réalisée
Sinon on poursuit le programme sans réaliser l'action

La structure conditionnelle IF THEN ELSE

Nous pouvons décider de réaliser une action A si la condition est vraie ou une action B si la condition est fautive. Dans ce cas il faut compléter la structure

Si l'on souhaite réaliser une action A à condition qu'une condition soit vraie et réaliser l'action B si l'action est fautive en français on écrit ' **Faire cette action A si la condition est vraie sinon faire l'action B**



Si la condition est Vraie faire l'action A
Autrement faire l'action B
(nota :une seule des 2 actions sera faite)

En basic
pour pic et *picaxe*

....
....

If var1=2 then

Action A

Else

Action B

End if

Que fait cette condition

La variable var1 contient une valeur renseignée en cours de programme

Quand le programme arrive à la ligne if
Si var1 contient 2 alors l'Action A est réalisée
Sinon c'est l'action B qui est réalisé

Puis le programme se poursuit

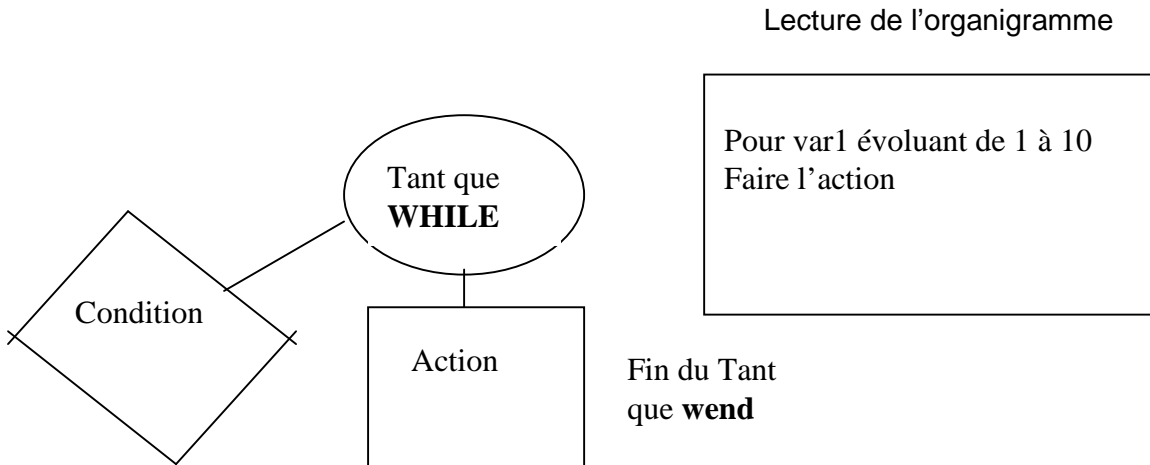
Vérification de la programmation:

Il faut vérifier que la condition a bien été lue avant le if

La structure tant que **WHILE WEND**

Si l'on souhaite réaliser une action tant qu'une condition est vraie **en français** on écrira
Faire l'action tant que la condition est vraie

Sous forme d'**organigramme** on peut décrire cette structure de la façon suivant



Pour un pic

Lire la condition

While condition vraie

Action

Lire la condition

Wend

Pour un picaxe

Lire la condition

do

While condition vraie

Action

Lire la condition

loop

Que fait cette structure

La condition est lue

Si la condition est vraie (le poussoir est actionné par exemple)

Alors l'action sera exécutée (mettre une sortie à 1 ce qui allumera une LED)

On relie la condition pour savoir s'il faut refaire l'action ou quitter la structure

On notera bien que la condition doit être lu **AVANT** le while et **DANS** la structure ,entre le while et le wend

Vérification de la programmation:

Il faut vérifier que la condition a bien été lu avant le while

Et quelle est à nouveau lue dans la procédure « entre le while et le wend) sinon on ne sort pas de la boucle

Certain langage ne comporte pas la structure du tant que (l'assembleur par exemple)

Il faut donc la recréer (avec la structure conditionnelle qui est présente dans tous les langages)

Debut :
Lire la condition
Si condition vrai
Action
Goto debut
Sinon
Goto fin :
Fin de si
Fin :

Début et fin sont des repères (on les appelle souvent étiquette)
On utilise la fonction goto qui permet de se placer à un endroit précis du programme . Cette fonction est à bannir car elle est 'anti structure'
En effet les programmes construits avec cette fonction sont difficiles à faire évoluer.(on sait ou on va ,goto mais on ne sait pas de où ?). Dans les structures (for next ,while wend ..)on sait toujours où on est .
On n'utilise le goto que dans ce cas précis .

Nota :

Quand certaines actions doivent se répéter on les écrits sous forme de sous programmes

Un sous programme s'écrit entre 2 instructions

Première instruction le nom du programme suivi de :

Deuxième instruction **return**

Exemple :

La boucle for next (dans l'exemple du début) peut faire l'objet d'un sous programme si l'on souhaite plusieurs fois dans le programme faire clignoter une LED on créera un sous programme clignotant .

Le sous programme se place en fin de programme on écrira

Pour un pic

Clignotant : (ne pas oublier les 2 points)

for l=1 to 10

led1=1

waitms1000

led1=0

waitms 1000

next l

return

pour un picaxe

clignotant :

for l= 1 to 10

led1=1

wait 1

led1=0

wait 1

next l

return

A chaque fois que l'on souhaitera faire clignoter la led on écrira

.....

Gosub clignotant

Gosub clignotant

Arrivé à cette instruction (gosub) le programme executera le sous programme (clignotant) .

Après son exécution il executera la suite des instructions du programme .

Exemple de programme

Ecriture du programme en français courant

Soit à réaliser une minuterie programmable .

On souhaite par poussoirs mémoriser le temps de la minuterie puis déclencher un buzzer quand le temps est écoulé .

A la mise sous tension on allumera brièvement la LED pour vérifier que le programme fonctionne

Pour mémoriser le temps

on va créer 6 poussoirs

Un poussoir va incrémenter des minutes tant qu'il sera poussé

Un autre va incrémenter des « 5minutes » tant qu'il sera poussé

Un autre va incrémenter des « 10minutes » tant qu'il sera poussé

Un autre va incrémenter des « 30minutes » tant qu'il sera poussé

Un autre va incrémenter des « heures » tant qu'il sera poussé

A chaque incrémentation on allumera la led

Un poussoir de départ lancera le comptage et émettra un bip pour vérification

Quand le temps sera écoulé le buzzer retentira quelques secondes puis une led restera allumée

On retourne au début (le comptage reste mémorisé ainsi on repartira sur la même temporisation si

On appuie sur départ , on ajoutera du temps si l'on appuie sur les autres poussoirs)

Ecriture du programme en langage structuré (on écrit le programme en utilisant les structures définies plus haut)

Dès la mise sous tension

Faire clignoter la LED

Debut du programme

Lire l'état des poussoirs

tant que le poussoir 1 est appuyé incrémenter le compteur de temps de 1

tant que le poussoir 5 est appuyé incrémenter le compteur de temps de 5

tant que le poussoir 10 est appuyé incrémenter le compteur de temps de 10

tant que le poussoir 30 est appuyé incrémenter le compteur de temps de 30

tant que le poussoir 60 est appuyé incrémenter le compteur de temps de 60

si le poussoir départ est appuyé

mettre en route le buzzer (avec plusieurs tonalités)

arrêter le buzzer

allumer la LED

puis retourné au debut

si le poussoir départ n'est pas appuyé

retourner au debut du programme

Le programme est écrit pour un pic **'configuration du pic**

```
Define CONF_WORD = 0x2f70
ADCON1 = %00001110 'configuration en I/O et ra0 en analogique
OPTION_REG = %00000000
OSCCON = %01100100 '4 Mhz
ADCON0 = %01000001 'A/D en ra0
TRISA = %00100011 'RA0 en entrée
TRISB = %00111111 'pattte 12 et 13 en sortie
```

'affectation des pattes du PIC **'et définition des variables**

```
Symbol pun = PORTB.0
Symbol cinq = PORTB.1
Symbol pdix = PORTB.2
Symbol ptrente = PORTB.3
Symbol psoix = PORTB.4
Symbol pdepart = PORTB.5
Symbol buzzer = PORTB.6
Symbol led1 = PORTB.7
Dim i As Word 'permet de compter jusque 32000
Dim j As Byte 'byte car ne dépasse pas 254
Dim un As Bit 'binaire des entrée tout ou rien du pic donc un bit suffira
Dim cinq As Bit
Dim dix As Bit
Dim trente As Bit
Dim soix As Bit
Dim total As Word 'permet de compter jusque 32000
```

'initialisation on donne une valeur par défaut au variables

```
un = 1 ' un est mis à zero quand le poussoir est activé
cinq = 1
dix = 1
trente = 1
soix = 1

total = 0 'nombre de minutes
```

'Faire clignoter la LED

```
led1 = 1
WaitMs 400
led1 = 0
WaitMs 300
led1 = 1
WaitMs 400
led1 = 0
debut:
```

'lire les poussoirs

'lire l'état du poussoir 1

un = pun 'quand le poussoir est appuyé la LED est à la masse donc =0

```
While un = 0 'tant que le poussoir est appuyé
led1 = 1 'allumer la LED
total = total + 1 'incrémenter le compteur de temps
WaitMs 500 'attendre 1/2 seconde
led1 = 0 'éteindre la LED
WaitMs 500 'attendre 1/2 seconde
un = pun 'relire la condition du tant que
Wend 'fin du tant que
```

'notez bien que la condition (valeur de **un**) est lu 2 fois, avant et dans le tant que) c'est souvent une source d'erreur car on oublie de lire dans la boucle et donc on n'en sort pas !!)

'lire l'état du poussoir 5

```
cinq = pcinq
While cinq = 0
led1 = 1
total = total + 5
WaitMs 500
led1 = 0
WaitMs 500
cinq = pcinq
Wend
```

'lire l'état du poussoir 10

```
dix = pdix
While dix = 0
led1 = 1
total = total + 10
WaitMs 500
led1 = 0
WaitMs 500
dix = pdix
Wend
```

'lire l'état du poussoir 30

```
trente = ptrente
While trente = 0
led1 = 1
total = total + 30
WaitMs 500
led1 = 0
WaitMs 500
trente = ptrente
Wend
```

'lire l'état du poussoir 60

```
soix = psoix
While soix = 0
led1 = 1
total = total + 60
WaitMs 500
led1 = 0
WaitMs 500
```

```
soix = psoix
Wend
```

```
If pdepart = 0 Then 'si l'on pousse le poussoir départ
led1 = 0          'allumane de la LED
buzzer = 1       'mise en route du buzzer
WaitMs 200       'pendant 0,2 secondes
buzzer = 0       'arreter le buzzer
For i = 1 To total 'comptage de minutes
For j = 1 To 60  'attendre 60 fois
WaitMs 1000     'attendre 1 minute
Next j
Next i
```

```
'mise en route du buzzer avec des fréquences différentes
For i = 1 To 300
buzzer = 1
WaitUs 1550
buzzer = 0
WaitUs 1550
Next i
```

```
For i = 1 To 475
buzzer = 1
WaitUs 800
buzzer = 0
WaitUs 800
Next i
```

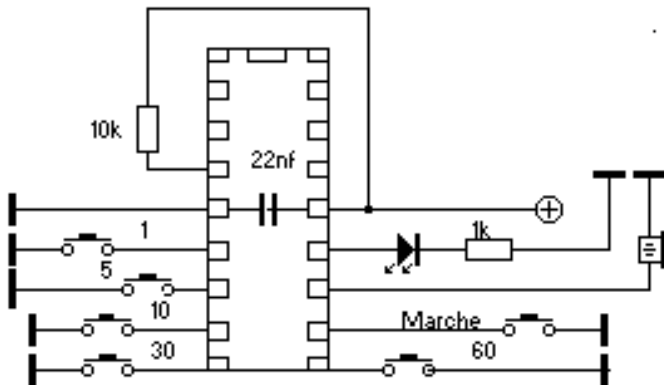
```
For i = 1 To 700
buzzer = 1
WaitUs 1150
buzzer = 0
WaitUs 1150
Next I
```

```
' fin d'émission du buzzer
led1 = 1          'allumer la LED
Endif 'fin de l'action provoqué par le bouton départ
```

```
Goto debut 'retour au début
End
```

(je vous laisse le soin d'écrire le programme pour un picaxe , il faudra modifier l'affectation des poussoir suivant le picaxe choisi ,traduire certain mots ,(par exemple waitUS n'existe pas il faudra le remplacer par pause)
Bon courage !!

Schémas pour un pic



Electronique autour du pic

Ci dessous figures des généralités qu'il faut affiner avec les datasheets (en particulier les tensions puissance intensités utilisable).

Afin de communiquer avec l'extérieur il est souhaitable de connaître le fonctionnement de certain composants La tension classique d'alimentation est de 5V (il existe cependant des pic fonctionnant à d'autres tensions)

Les entrées :

L'entrée tout ou rien

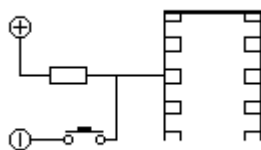
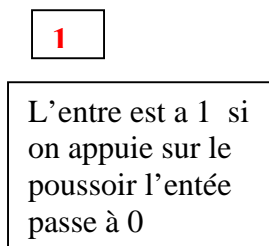
Un poussoir ou un interrupteur permet de mettre à 0 ou 1 une entrée , cependant quand l'interrupteur ou le poussoir r ne sont pas actionné il faut les forcée a un etat (0 ou 1)

Il y a 2 solutions pour mettre une entrée à 1

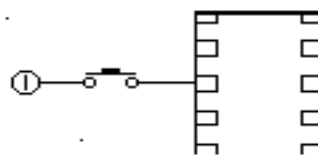
- 1/ on relie la patte au 5 v avec une resistance de 10K
- 2/ la configuration de certaines entrée est possible par soft (configuration activation pull up)

la mise à 0 se fait par mise à a masse (avec une resistance de 10K

exemple :



2



L'entre est a 1 si on appuie sur le poussoir l'entée passe à 0

entrée analogique

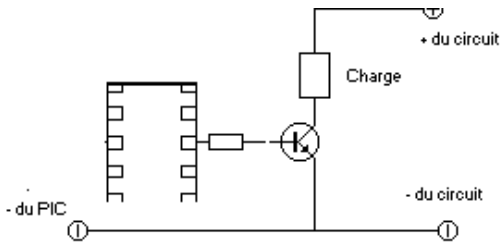
Une entée configurée « analogique » fournie au programme une valeur (entre 0 et 255 ou entre 0 et 1024) Suivant la tension (il ne faut pas dépasser la tension d'alimentation)

Sorties

Les sorties ne peuvent délivrer que quelques milliamperes et le pic quelques dizaines (voir les datasheet) on ne peut utiliser les sorties que pour alimenter quelques diodes ou buzzer .

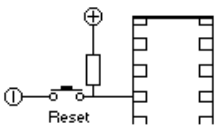
sinon on utilise un transistor en commutation (il se comporte alors en interrupteur dès que sa base est excitée. (cet interrupteur provoque une chute de tension de 0,6V).

Lorsque la sortie passe à 1 la base du transistor (alimenté en série avec une résistance de 1K) permet au transistor de devenir passant et donc d'alimenter la charge. Le schéma montre un transistor NPN . si l'on veut alimenter la charge avec un 0 il faut monter un transistor PNP.



Certaines sorties (dans le pic)sont en drain ouvert (la broche RA4 du 16F628 par exemple) elle ne peut donc pas fournir du 0V (on ne l'utilisera pas pour des liaisons type RS232...

Par sécurité je configure la broche MCLR ON (et non pas en entrée) .cela me permet d'utiliser cette broche en reset et de ne pas avoir de soucis de programmation d'un pic qui utilise l'oscillateur interne .



Annexe

Configuration du PIC

C'est vraiment la partie la plus « pénible »

Cependant elle est faite une fois pour toute par type de programme

Par exemple si on utilise un pic avec une entrée analogique et le reste en tout ou rien

La configuration restera valable pour tout les programme qui n'auront que cette exigence .

Ci dessous vous trouverez un début d'explication du contenu es registres

```
Define CONF_WORD = 0x2f70
```

Ce registre configure 11 parametres

Faut il protéger le code (pour le rendre incopiable)

Faut il utilise la sortie 5 du port A en reset

....

.....

```
OSCCON = %01100100
```

Ce registre précise la valeur de l'oscillateur interne

Les bits 654 (110) precisent par exemple la fréquence à 4MHZ

```
ADCON1 = %00001110
```

```
ADCON0 = %01000001
```

Ces registres se positionnent suivant l'utilisation qui sera faite du convertisseur Analogique digital (quelle entrée est utilisée)

```
OPTION_REG = %00000000 (par défaut ce registre n'est pas à %00000000 )
```

C'est dans ce registre que l'on positionne le pull up de l'entrée du portB

(on met de façon interne une résistance entre les entrées et le +)

pour ne pas avoir une valeur de tension aléatoire .
(sur le port A c'est pas possible on doit mettre une vraie résistance)
(tout n'est pas détaillé dans ce tutorial puisque le but est seulement de monter la démarche)
Vous n'y coupez donc pas de consulter la datasheet du pIC concerné.
Si vous programmez un **picaxe** vous remplacez cette configuration par
#picaxe 20m2 'c'est quand même beaucoup plus simple !

bon amusement