Electromag1 Circuits électroniques et Picaxes

Petit site pour ceux qui bricolent en électronique en général et sur les PICAXES en particulier.
Les PICAXES sont des microprocesseurs (PIC de Microship) programmables en BASIC. Pleins de qualités, pas chers, ultra faciles à programmer.
Tout (ou presque) est sur le site du distributeur Gotronic (voir les liens).
Les pages seront ajoutées ou modifiées petit à petit.
Dans le but d'améliorer ce blog, communiquez moi vos remarques et les erreurs que vous pouvez constater.

Pour me contacter: Cliquez sur "Contact",ici ou en haut de la colonne de gauche

Pour retourner au menu, Cliquer sur "Accueil", ici ou en haut de la colonne de gauche

Bonne visite

ACCUEIL
CONTACT
Livre d'or

Les picaxes

et leur basic

Présentation:

Les picaxes ne sont pas destinés aux professionnels de la programmation, mais aux élèves débutants et aux bricoleurs voulant développer **simplement et facilement** une petite application dans les domaines de la robotique, domotique, animation de leds, temporisations et toutes les applications ne demandant pas une vitesse de traitement élevée.

Prérequis recommandés pour continuer la lecture:

Cette page ne remplace pas la lecture de la doc officielle.

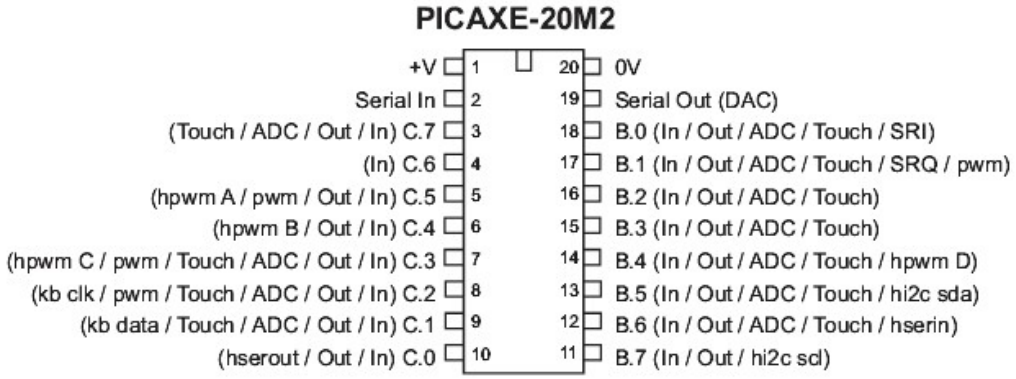
Lire les trois tomes de la documentation picaxe en téléchargement ci dessous:
http://www.picaxe.com/Getting-Started/PICAXE-Manuals/
Le tome 2 sur les commandes vient d'être traduit en français.
http://www.picaxe.com/docs/picaxe_manual2_fr.pdf
Il n'est pas question d'appendre toute cette doc par coeur, mais d'avoir une idée du contenu.

Je recommande aussi la lecture de ce tutoriel, écrit par BESQUEUT, intervenant actif sur le forum français:
PDF en téléchargement ICI

Installer le logiciel gratuit "Programming Editor 5":
http://www.picaxe.com/Software/PICAXE/PICAXE-Programming-Editor/
Ce logiciel est un éditeur de texte, un simulateur et charge le programme dans le picaxe.
Il permettra de manipuler les programmes d'exemple.
Un nouvel éditeur, PE6 est sorti, plus puissant, avec de nouvelles fonctionnalités (par ex les macros), et intégrant le logiciel de programmation graphique FLOWCHART (pour ceux qui aiment...).
Voici un excellent tuto d'Alain sur PE6 ICI
Il est aussi beaucoup plus "lourd" et les exemples ci dessous utilisent PE5

Les picaxes sont classés en deux séries
La série M2, du 08M2 au 20M2
La série X2, du 20X2 au 40X2, plus performante que la série M2.

Les exemples s'appuieront sur le 20M2, le plus gros de la série M2.



Voyons un peu...

- Broches 1 et 20, c'est l'alimentation.

En général fixée à 5 v, mais la plage de fonctionnement descend aux environs de 2 v.
Fonctionnement sur pile sans problème. Ne jamais dépasser 5,5 volts!

- Broches 2 et 19, pour la programmation

C'est ici que se connecte le câble de programmation série: RX sur Serial IN et TX sur Serial Out.
Sur le 20M2, la broche 19 est aussi une sortie DAC (Convertisseur Numérique Analogique), la valeur "analoqique" comporte 32 niveaux, c'est déjà ça...

- Les broches B et les broches C.

Il y a deux groupes de broches:
D'un côté : de 3 à 10, les broches C, nommées C.0 à C.7
De l'autre côté, de 11 à 18, les broches B, nommées B.0 à B.7.

Toutes ces broches sont configurables en entrée (In) ou en sortie (Out), sauf C.6, qui ne peut être qu'une entrée (il faut bien une exception), on retrouve cette distinction sur les broches C.3 ou C.5 des autres picaxes M2 (je vous laisse vérifier).

De plus, ces broches sont utilisées par des commandes basic plus évoluées:
Des ADC, convertisseur Analogique - Digital, qui transforment des tensions en nombres, (lecture de capteurs, etc)
Des PWM, modulation de largeur d'impulsion, qui donnent en tache de fond des signaux de fréquences et de rapports cycliques variables, (variation de vitesse moteur, dimmage d'éclairage, etc)

Des liaisons séries protocole 12C.
Pour le reste, lisez un peu la doc.

Détail technique: le courant max pour une sortie est de 20 mA. Mais le courant total délivré par les broches est limité à 90 mA.
Il faut tenir compte de ces limitations pour l'application réelle des exercises ci dessous.
Au delà des limitations, le courant devra être amplifié. Plusieurs possibilités: transistor, mosfet, uln2803...
Voir les détails dans le tome 3: http://www.picaxe.com/docs/picaxe_manual3.pdf

Les mémoires disponibles, et quelques spécifications:

Feature	PICAXE Command	08M2	18M2	18M2+	14M2	20M2
Voltage Range (V)		2.3-5.5	1.8-5.5	1.8-5.5	1.8-5.5	1.8-5.5
Memory Capacity (bytes)		2048	2048	2048	2048	2048
Parallel Tasks (starts)	resume, suspend	4	4	8	8	8
Max Internal Freq (MHz)	setfreq	32	32	32	32	32
Variables RAM (bytes)	peek, poke @bptr	128	256	512	512	512
Table data (bytes)	table, readtable tablecopy	-	-	512	512	512
I2C master support	hi2cin, hi2cout hi2csetup	Yes	Yes	Yes	Yes	Yes
Pwmout channels	pwmout	1	2	2	4	4
Hpwm support	hpwm	No	No	No	Yes	Yes
Keyboard support	kbin, kbled	No	No	Yes	Yes	Yes
RF radio support	rfin, rfout	No	No	Yes	Yes	Yes
Internal temp. sensor	readinternal-temp	Yes	No	Yes	Yes	Yes
Configurable input type	inputtype	No	No	No	Yes	Yes

**18M2 has now been replaced by 18M2+*

Dans le système picaxe, le terme byte représente un octet (groupe de 8 bits).

Capacité mémoire : 2048 octets. C'est l'espace mémoire recevant le code du programme. Pour les X2, il y a 4096 octets.

Table de données: 512 octets. Non disponibles sur le 08M2. Ce sont des valeurs fixes, introduites par le code du programme, non modifiables par le programme, par ex: l'enregistrement d'un texte ou la description point par point d'une courbe, etc.

mémoires EEPROM: 256 octets, accessibles en lecture et écriture à tout moment, pour sauvegarder et relire des données sur 8 ou 16 bits par

ex: des résultats de calcul, des paramètres, des données, etc.

Variables RAM : 512 octets. Accessibles par les commandes "poke", "peek", et en adressage indirecte par le pointeur "@bptr" (ça, c'est pour plus tard).
28 de ces octets sont manipulables directement, numérotés de b0 à b27 (b comme byte). Ils permettent d'enregistrer des nombres de 0 à 255

Les bytes b0, b1, b2, b3 sont découpables en "bit" pour les variables binaires, 0 ou 1
Ex : b0 = { bit7;bit6;bit5;...;bit1;bit0}

La concaténation de deux bytes donne une variable de type "word" de 16 bits, les 28 bytes donnent alors 14 variables numérotées de w0 à w13, (w comme word), pour enregister des nombres de 0 à 65535.
w0= {b1;b0}

ATTENTION:
Rien n'empêche d'utiliser la variable b1 d'une part et la variable w0 d'autre part, comme les deux variables se partagent le même octet b1, cela entrainera des surprises aléatoires.
Pour éviter les mésaventures, on peut:
Réserver b0 pour un découpage éventuel en variables bits
Prendre les variables bytes dans le sens croissant : b1,b2,b3...etc
Prendre les variables word dans le sens décroissant : w13,w12,w11...etc

Toutes ces variables ne contiennent que des nombres entiers positifs (et pas tous)
l'incrémentation d'une variable "byte" donne une boucle de 256 nombres .
255+1 = 0 (sans avertissement). Les variables "word" bouclent sur 65536 nombres.

Les particularités du basic (picaxe)

Le signe =

En mathématiques, le signe = signifie que, ce qu'il y a de chaque côté, représente exactement la même entité. Si a=b on peut écrire b=a, c'est pareil.

En basic, ce n'est pas le cas: Ce qu'il y a à gauche est une mémoire, comme b0.
Ce qu'il y a à droite est son contenu, un nombre. Comme si on disait qu'une bouteille est égale au liquide qu'elle contient.

Le signe = n'est pas un signe d'égalité, mais d'attribution. Cette confusion des genres entre contenant et contenu agace les esprits les plus rigoureux, d'autres langages dissocient les signes d'égalité et d'attribution (= et ==).

En basic:
b2 = 5 à une signification.
5 = b2 ne veut rien dire et donne une erreur.

La hiérarchie des opérateurs:

"Normalement": 3 + 2 x 6 = 15 On fait la multiplication avant l'addition.
En basic picaxe: 3 + 2 x 6 = 30 Les opérations se font de gauche à droite, stritement.

La vitesse horloge:

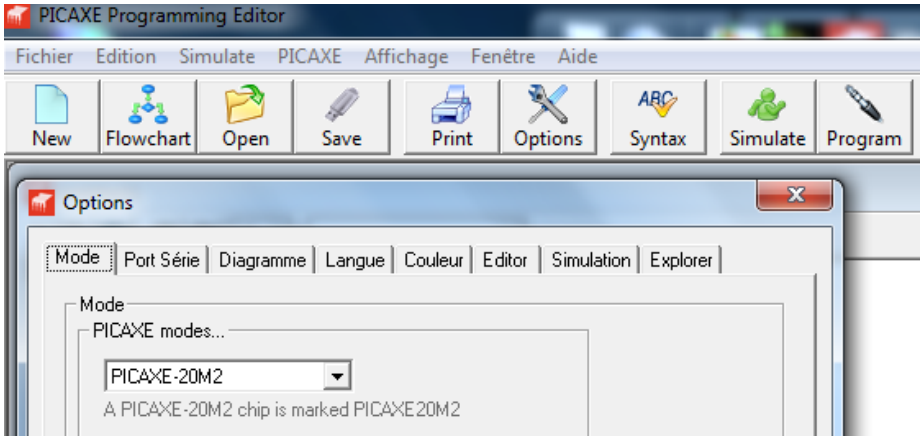
Sur la série M2, elle est de 4MHz par défaut.
Elle est modifiable par la commande "setfreq" et grimpe jusquà 32 MHz sur cette série.
Sur les X2, la fréquence max est de 64MHz, avec ou sans quartz, suivant le modèle.
Attention, les commandes liées au temps sont affectées par le changement de fréquence.
Ex: "pause 1000" correspond à une pause de 1000 ms pour une fréquence de 4 MHz, mais à 8 MHz, ce n'est plus que 500 ms, etc...

Les signes particuliers:

Les caractères " ; " et " ' " indiquent que le texte qui suit n'est pas du code à exécuter, mais un commentaire.
Le caractère " : " permet de séparer plusieurs commandes sur une même ligne: Ex: high led1 : low led2

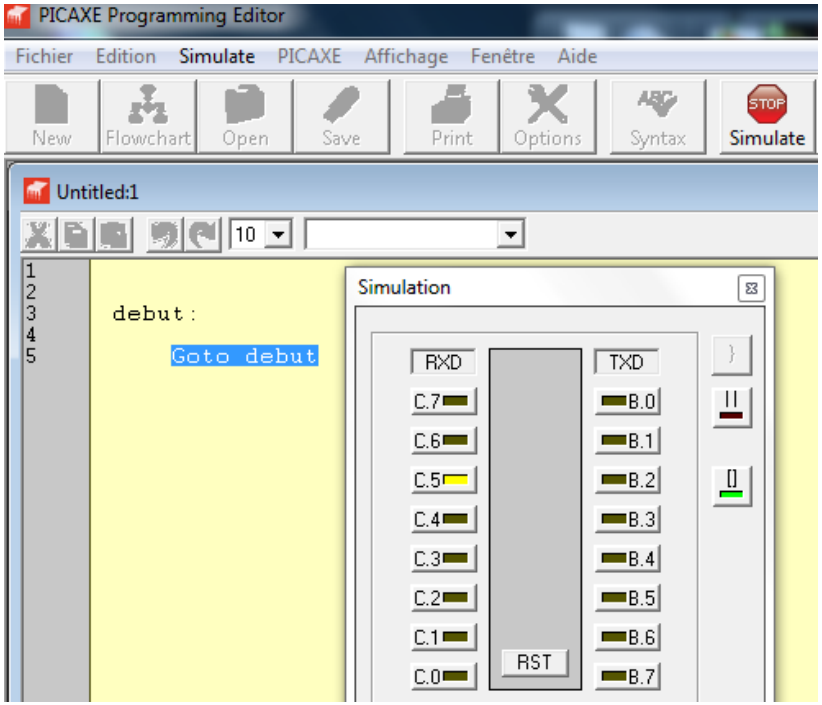
Bon, c'est la récré, on va jouer au simulateur !

Ouvrez le Picaxe Programming Editor 5.5.5 ou 5.5.6



Les icones carrés donnent accès aux fonctions, dans les options, sélectionnez Picaxe-20M2

On dit d'un programme "qu'il tourne". La fenêtre du simulateur ne reste ouverte que si il y a des instructions à simuler, (ou si on a décocher une case dans les options de simulation).
La commande la plus connue du basic est le GOTO, associé à son étiquette, "ça tourne".



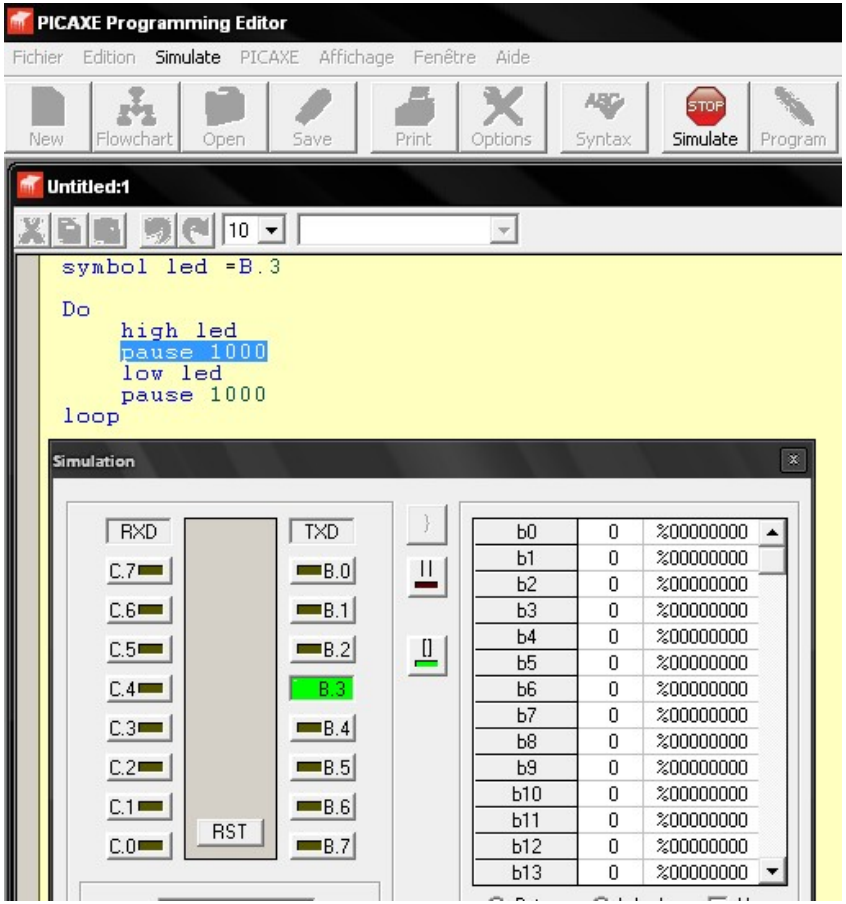
Le goto et son étiquette "debut:" sont écrits dans l'éditeur.
Remarque: le nom de l'étiquette est suivi de ":" (pas dans le goto).
La simulation est lancée en cliquant sur l'icône "Simulate", et la fenêtre s'ouvre.
Nous voyons les 16 broches du 20M2. Par défaut, toutes les Entrées/Sorties sont configurées en Entrées, au niveau bas (0) . Les entrées passent au niveau haut (1) en cliquant sur le bouton (Ex entrée C.5). Pour repasser au niveau bas, il faut cliquer une deuxième fois.
Seul problème, ce programme ne fait rien.

Trois façons de dire la même chose:
Niveau haut = + alimentation = niveau logique " 1"
Niveau bas = 0 volt = niveau logique "0"
En fait, ce n'est pas tout à fait aussi simple, les niveaux logiques 0 et 1 sont définis par des normes:
Norme TTL: Pour une alimentation > 4,5 v , une tension < 0,8 v est un 0 et > 2 v , c'est un 1
Pour plus d'infos, voir la commande Inputtype du manuel

Trois façons d'écrire le même nombre:
En décimal : 1252 = 1252 (normal...)
En binaire : 1252 = %10011100100 (% pour binaire)
En hexadécimal : 1252 = $4E4 ($ pour hexadécimal)
On peut écrire un nombre sous une forme ou une autre, aucune importance.

Allumons des leds

Traditionnellement, le premier programme fait clignoter une led, le voici:



Remarques:
Toutes les broches et les variables peuvent être nommées par une commande "symbol", le programme est beaucoup plus lisible, le premier bénéfiaire est l'auteur lui-même.
Ici, la broche B.3 s'appelle "led".
Parfois utile, le goto a mauvaise réputation. La boucle goto et son étiquette est remplacée par une boucle do : loop. De plus, la possibilité d'ajouter des conditions (while , until) au début ou en fin de boucle est très intéressante.
Les commandes "high" et "low" sont, sous leur aspect anodin, un concentré d'informations, "high led" signifie:

Que la broche concernée est B.3
Que cette broche sera configurée en sortie (automatiquement).
Que cette sortie sera au niveau haut (+ alimentation)

Rem:C.6 ne peut être qu'une entrée, high C.6 provoque (généralement) une erreur de syntaxe.

Le dessin de la broche B.3 a changé, c'est un rectangle noir pour le niveau bas ou vert pour le niveau haut, il suffit de relier une led avec sa

résitance de limitation (max 20 mA) à cette broche pour la faire clignoter, une seconde on, une seconde off.

La fonction spéciale "dir"
Cette fonction spéciale permet de configurer les broches en Entrées ou en Sorties, en fonction des circuits branchés sur les broches.
Le picaxe 20M2 distingue les broches B et les broches C. Il y a une fonction spéciale pour chaque groupe de broches : dirsB et dirsC.
Ex: Pour que la broche B.3 soit configurée en sortie, il faut écrire:

DirB.3=1
ou:
dirsB= %00001000
Les 8 broches sont représentées dans un octet, de B.0 à droite à B.7, à gauche

Rem: C.6 ne peut être qu'une entrée, mais, dirC.6 =1 ne provoque aucune erreur, la commande est simplement ignorée, idem pour dirsC = %01000000

La fonction spéciale "pin"
Le basic fait la distinction entre une broche, et sa valeur logique (0 ou 1) .
Ex : C.6 représente la broche ; pinC.6 représente sa valeur logique.
Dans un test, il ne faut pas considérer une broche, mais sa valeur.
Les groupes des 8 broches B et C sont nommés pinsB et pinsC
Ex : Les 8 broches B sont représentées par un octet, de B.0 à droite à B.7, à gauche.
Pour fixer la valeur de la broche B.3 au niveau logique 1, il faut écrire:
pinB.3 = 1
ou
pinsB= %00001000 (c'est un octet, en binaire)

pinsB est donc un octet, mais la commande pinsB a deux fonctions distinctes:

- A gauche du signe "=", l'octet représente respectivement le niveau des broches "B" en sorties, mais uniquement pour les broches définies précédemment comme une sortie par une commande "dir".
- A droite du signe "=", l'octet représente respectivement les niveaux des entrées lus sur les broches "B".

Résumé:
dirsC = %11111111 (les huit broches C sont définies en SORTIES)
pinsC = pinsB (les broches C prendront broche à broche les niveaux lus sur les broches B)
Voir les exemples dans la suite

Voir suite page n°2.

Electromag1 Circuits électroniques et Picaxes

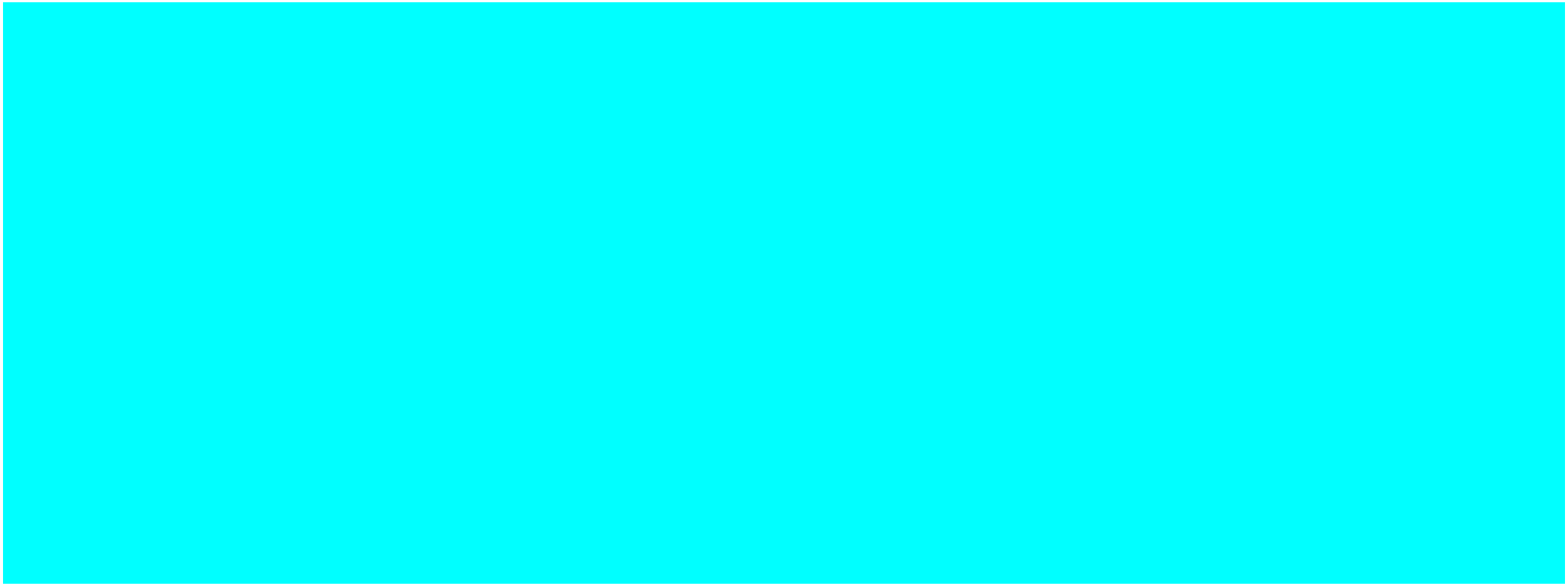
Petit site pour ceux qui bricolent en électronique en général et sur les PICAXES en particulier.
Les PICAXES sont des microprocesseurs (PIC de Microship) programmables en BASIC. Pleins de qualités, pas chers, ultra faciles à programmer.
Tout (ou presque) est sur le site du distributeur Gotronic (voir les liens).
Les pages seront ajoutées ou modifiées petit à petit.
Dans le but d'améliorer ce blog, communiquez moi vos remarques et les erreurs que vous pouvez constater.

Pour me contacter: Cliquez sur "Contact",ici ou en haut de la colonne de gauche

Pour retourner au menu, Cliquer sur "Accueil", ici ou en haut de la colonne de gauche

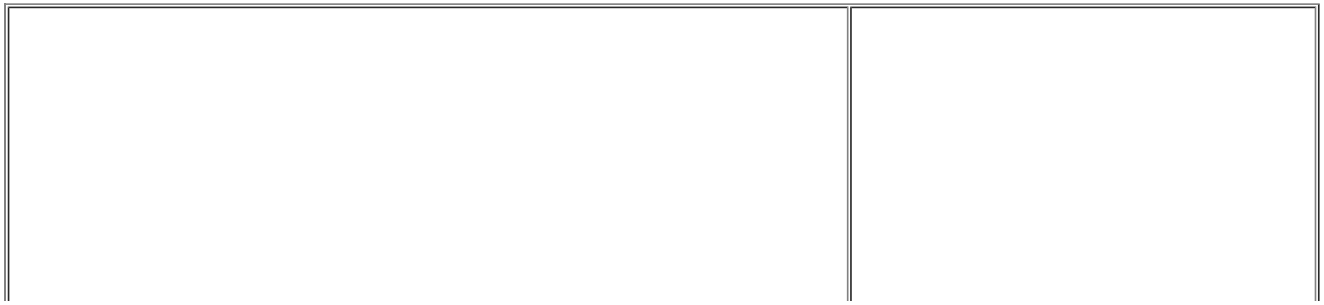
Bonne visite

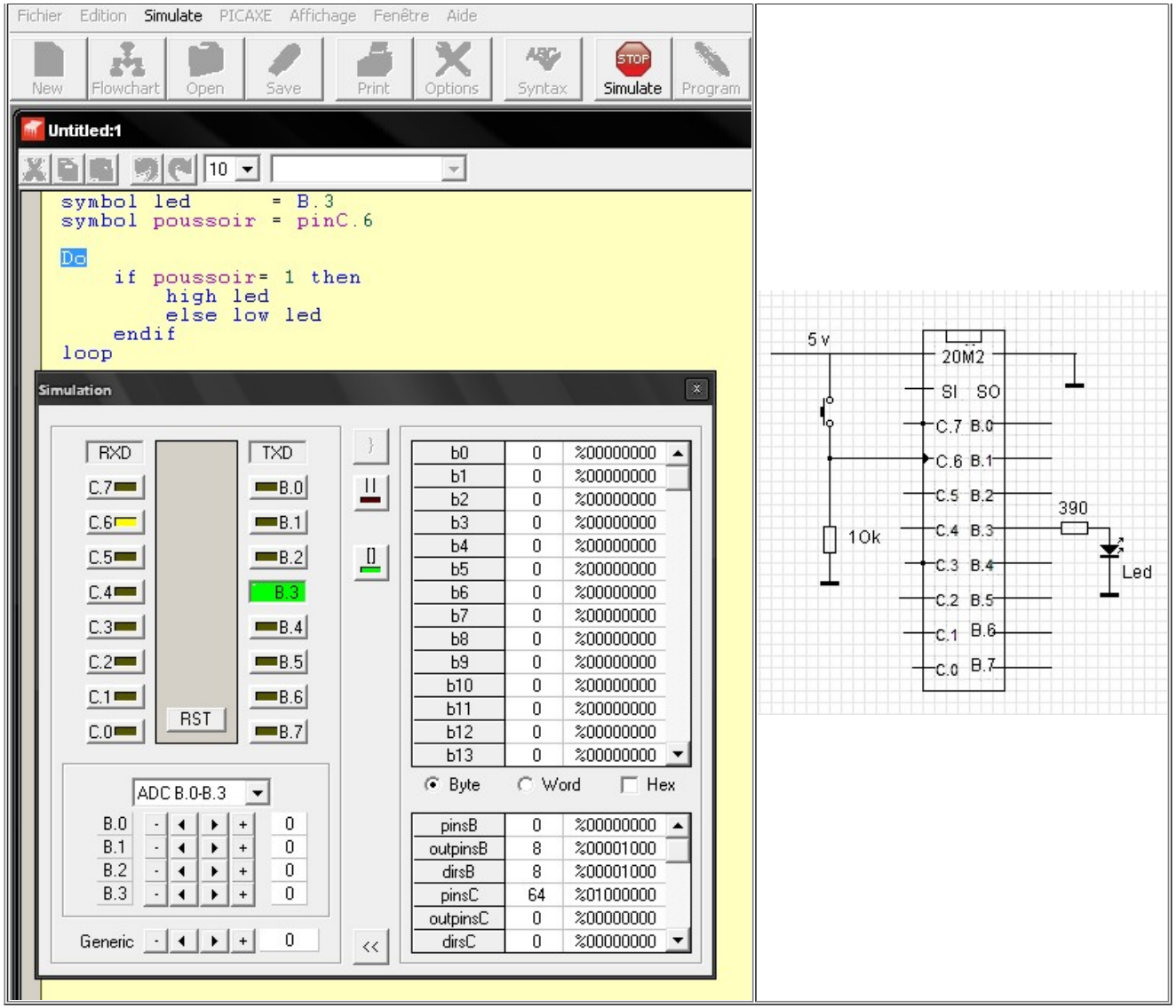
ACCUEIL
CONTACT
Livre d'or



Allumer une led avec un poussoir

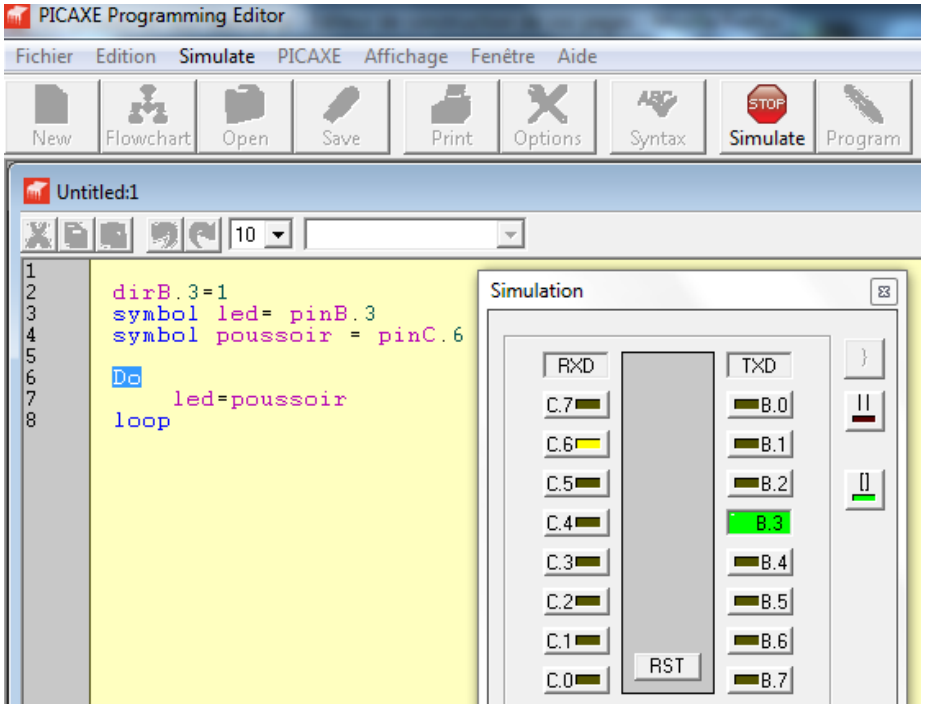
Les rieurs diront qu'on peut le faire sans µC, ils auront raison. Toutefois, l'exercice n'est pas sans intérêt.
Il va falloir tester la valeur de la broche entrée, alimentée par le poussoir.
En premier lieu, on pense à un programme de ce genre:





Si le poussoir positionne C.6 au niveau haut (jaune) , alors, la sortie B.3 passe au niveau haut (vert) et la diode s'allume, sinon, elle passe au niveau bas (noir)

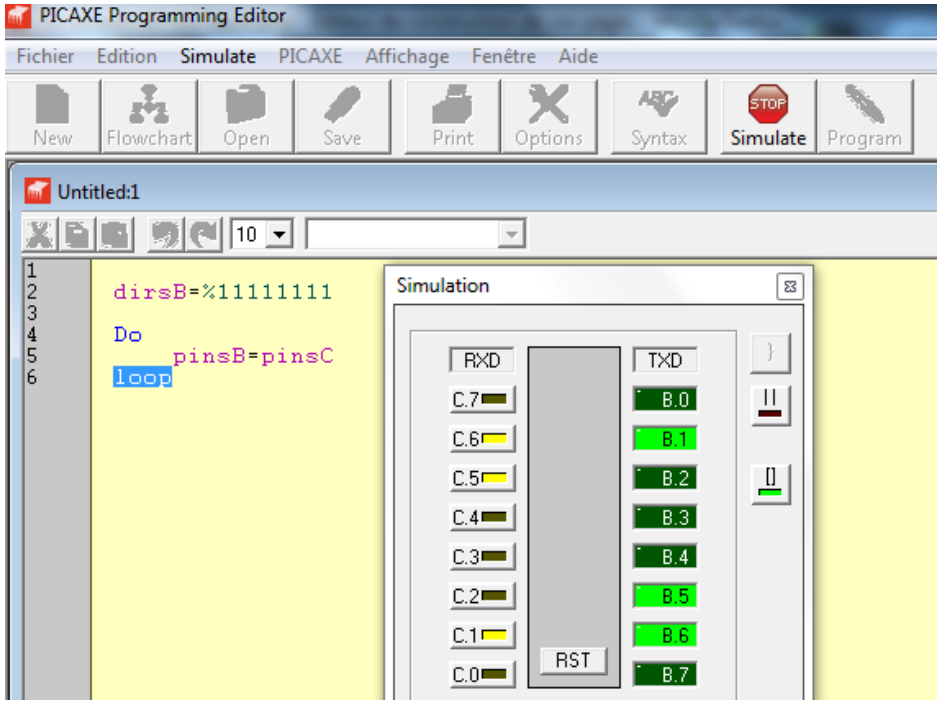
Autre méthode:
Comme pour les entrées, les valeurs des broches configurées en sorties peuvent être désignées par pin (ou outpin, si on veut faire la distinction entre entrées et sorties).
Ex : **Aprés avoir configurer la broche B.3 en sortie**, on peut écrire que la valeur d'une sortie est égale à la valeur d'une entrée.



Le "if then" a disparu et il n'y a plus qu'une seule ligne, la diode en B.3 s'allume si on appuie sur le poussoir en C.6

Plus fort: 8 poussoirs et 8 leds

dirsB = %11111111 configure les 8 broches B en sorties.
pinsB (ou outpinsB) désigne les 8 sorties sous la forme d'un octet
pinsC fait de même avec les entrées C



En appuyant sur les poussoirs C.1, C.5,C.6 les leds correspondantes B.1, B.5, B.6, s'allument.

Le chenillard

En plus des dénominations visibles sur cette simulation en B.x ou C.x, chaque broche est représentée par un simple numéro.

Ainsi, les broches :

B0 à B7 sont numérotées de 0 à 7

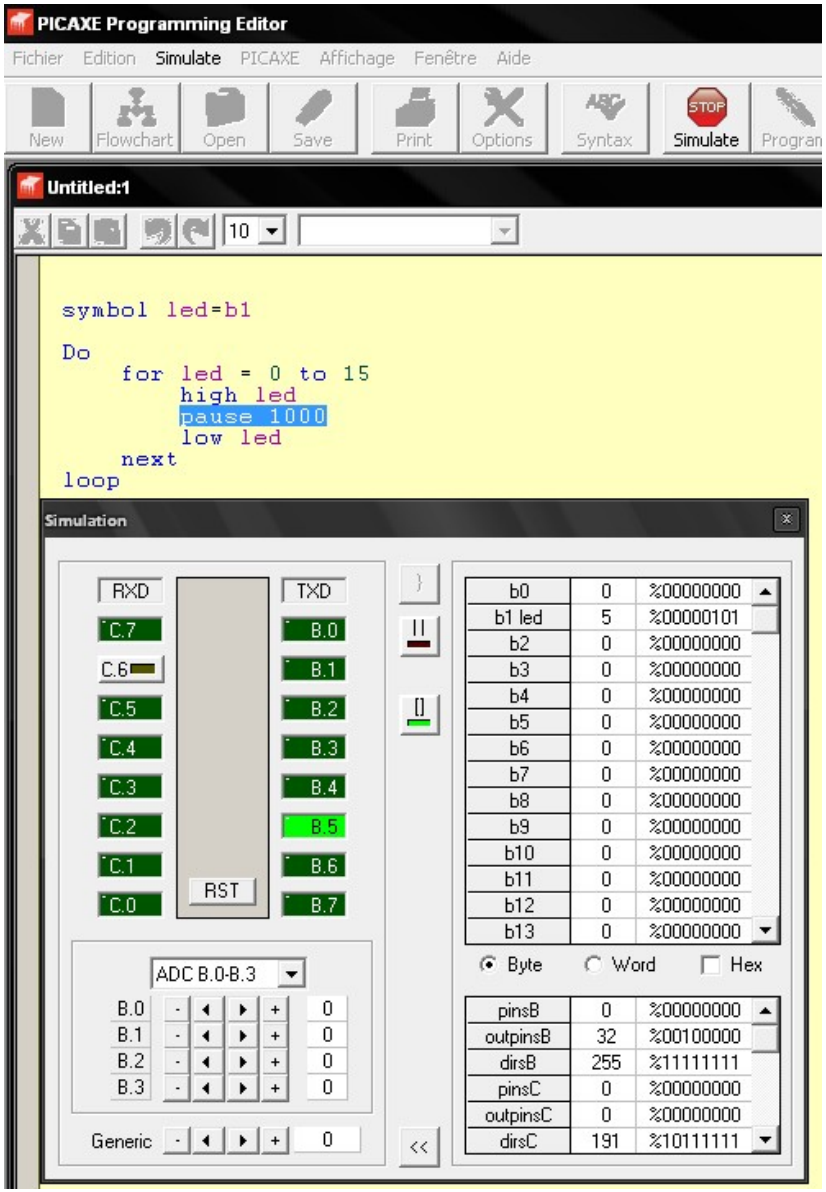
C0 à C7 sont numérotées de 8 à 15

Pour les picaxes de la série X2:

A0 à A7sont numérotées de 16 à 23

D0 à D7 sont numérotées de 24 à 31

Cela facilite la programmation d'un chenillard

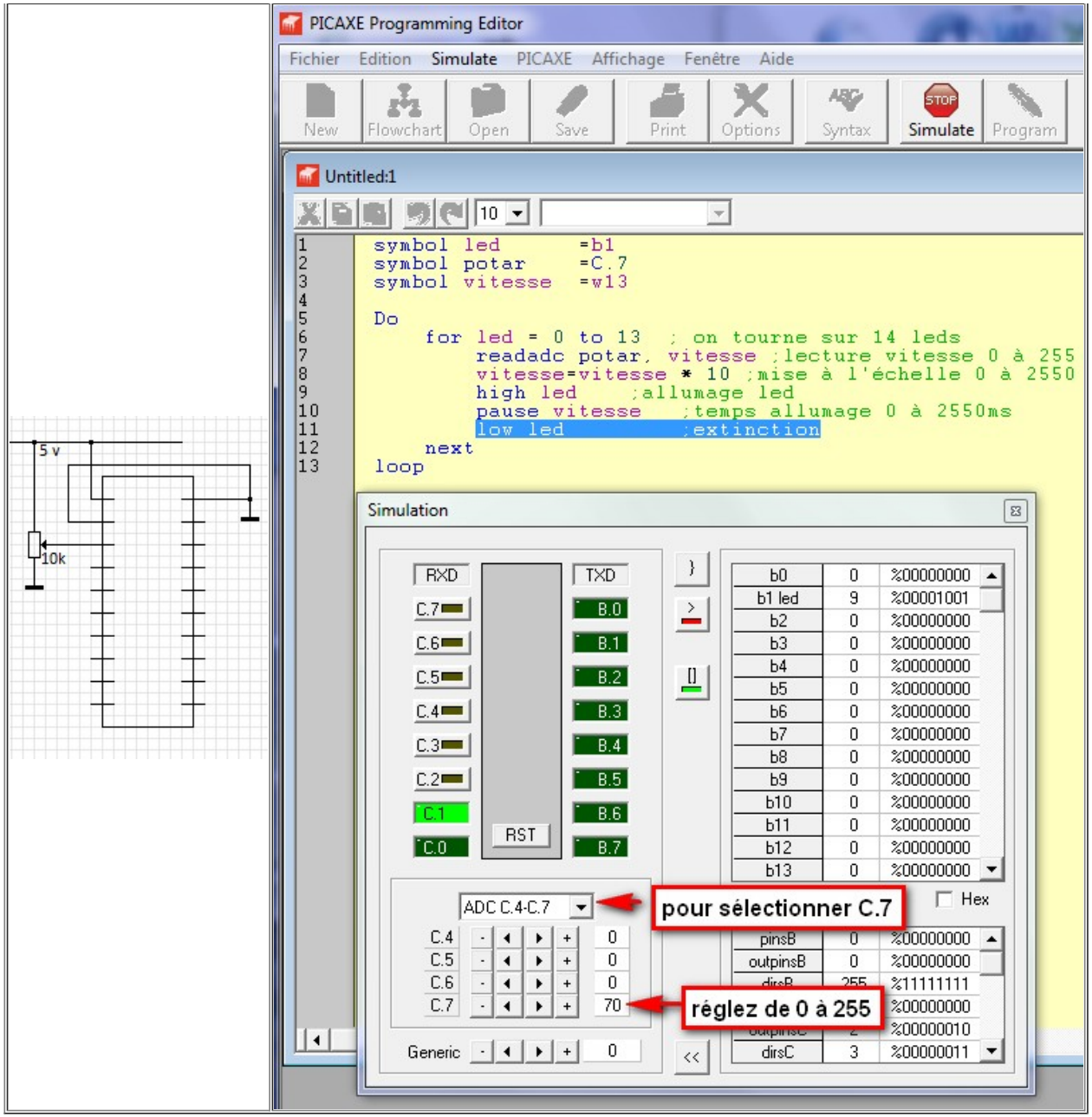


Une bouche For / next incrémente le compteur "led" de 0 à 15, allume chaque broche pendant une seconde, jusqu'à la dernière, et le cycle recommence.
Ici, toutes les broches ont été positionnées en sorties pendant le premier cycle de la simulation (par les commandes HIGH et LOW), sauf C.6.
Aucune erreur à la vérification de syntaxe. Lorsque le compteur est arrivé sur C.6, il y a eu une pause de 1 seconde, la broche a été ignorée et le compteur a continué, ...tranquillement.
Pour éviter la pause de trop, il faut faire un test sur led=14 et avancer d'un cran.

```
symbol led=b1

Do
    for led = 0 to 15
        if led=14 then
            inc led
        endif
        high led
        pause 1000
        low led
    next
loop
```

Faire varier la vitesse:
"pause 1000" allume les leds pendant 1000 ms. Cette valeur peut varier entre 0 et 65535.
Avec un potentiomètre et un ADC, il est facile de modifier la valeur de cette pause.
l'ADC ou "Analogique to Didital Converter" ou CAD en français, convertit une tension variable,
par ex entre 0 à 5 v (tension d'alimentation), en un nombre
Il y a deux commandes ADC:
ADC 8 bits: "readadc", convertit une tension de 0 à 5 v en un nombre de 0 à 255
ADC 10 bits: "readadc10", convertit une tension de 0 à 5 v en un nombre de 0 à 1023
Il faut sacrifier une broche (qui supporte la commande ADC, voir le 20M2 au début) pour brancher ce potentiomètre. La broche C.7 semble toute désignée, elle est la dernière, après C.6, nous tournerons sur 14 broches et le test n'est plus utile.
Les commentaires: Les caractères suivant ";" ou "'" sont des commentaires, expliquant le programme, l'idéal est de commenter chaque ligne. Le grand bénéficiaire est encore l'auteur lui même. Le texte est de couleur différente, en vert ici.
Les commentaires et la désignation de "symbol" n'occupent pas de place en mémoire. L'abus ne nuit pas.
Si le µC n'est pas programmé sur site, la broche "serial in " doit être reliée au 0 Volt.

Le schéma correspondant	



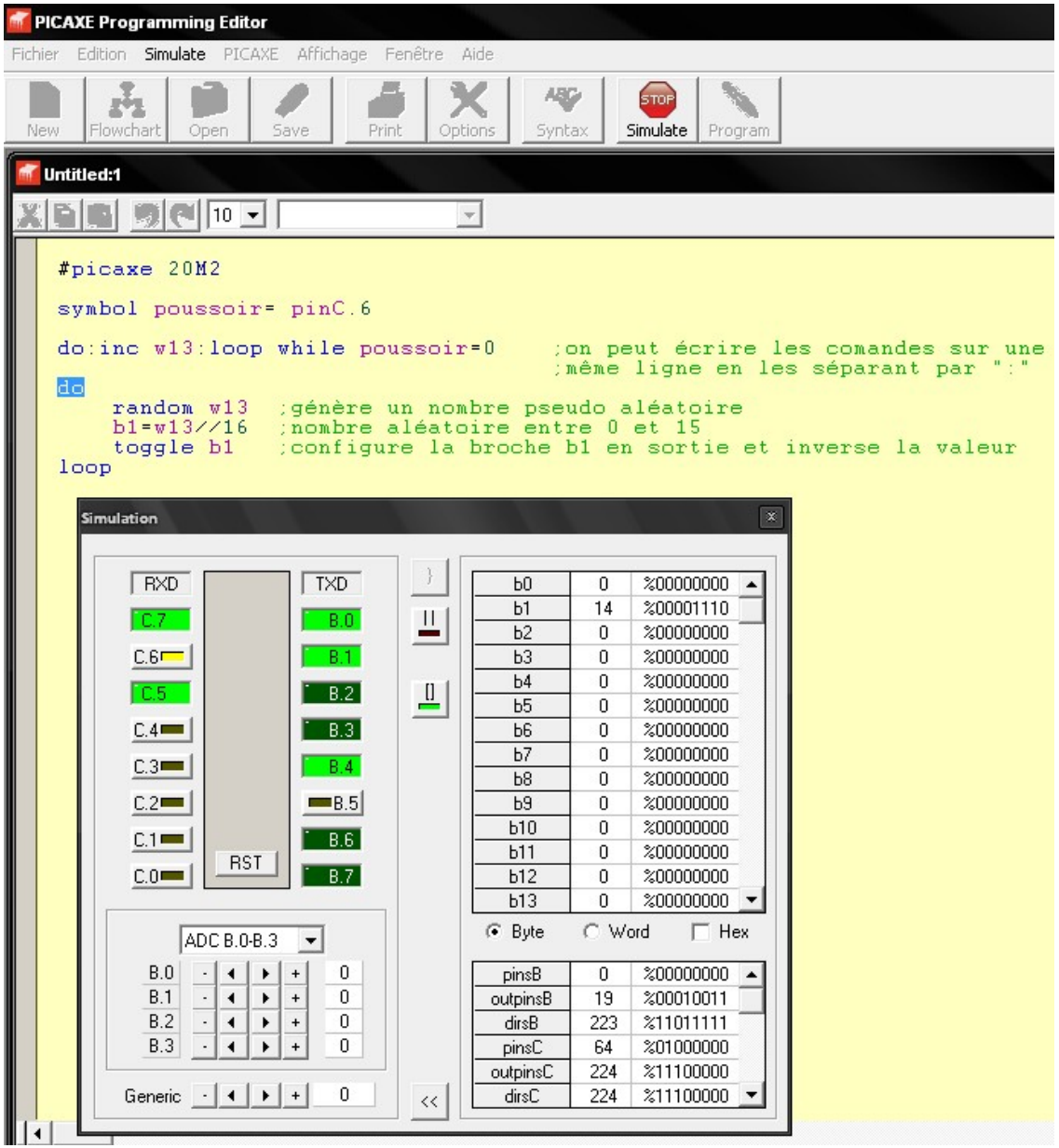
Un peu de hasard

A la longue, le chenillard est un peu lassant, et on connait exactement la suite des évènements. La commande **random** permet de générer des nombres au hasard, enfin presque.

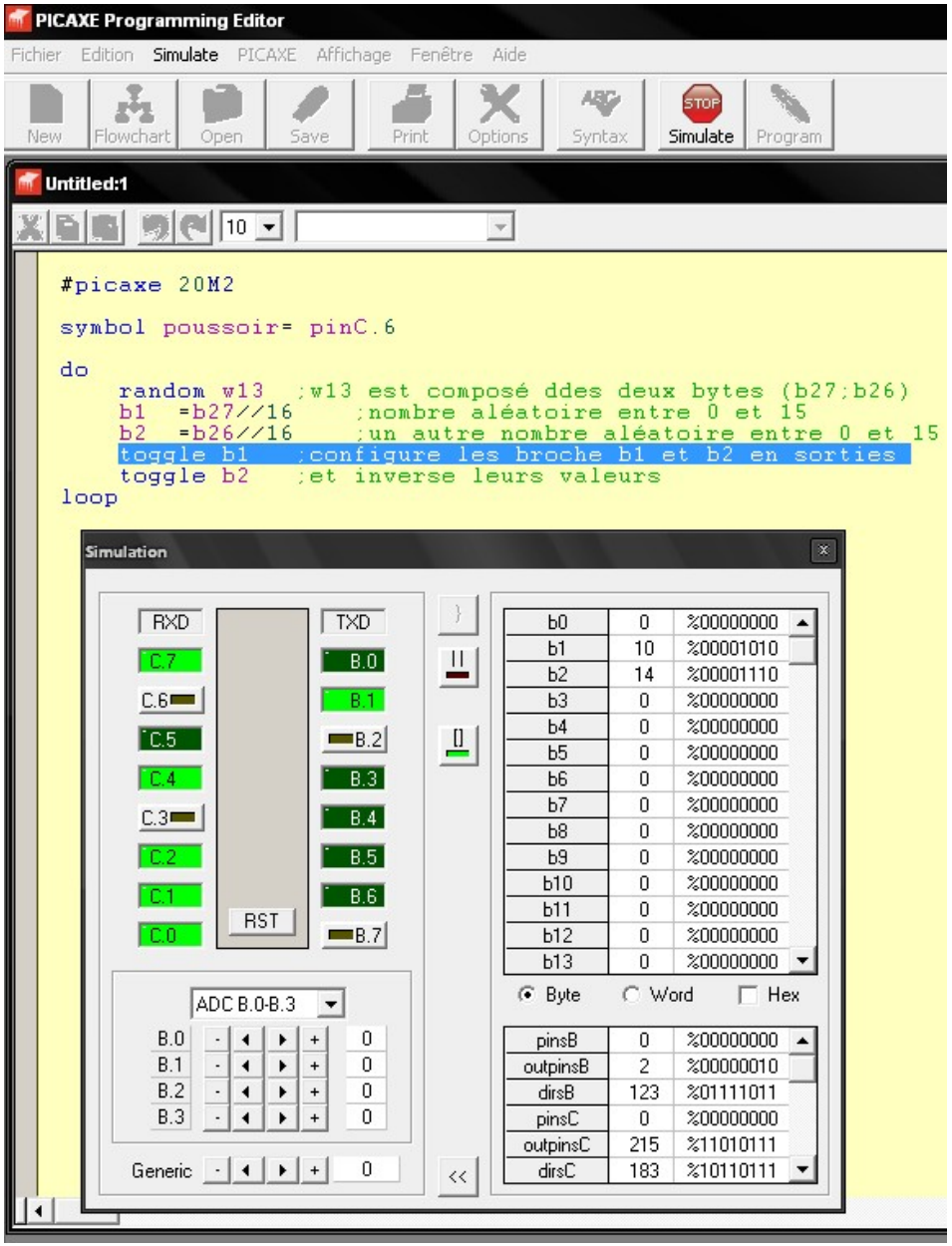
Cette fonction donne toujours la même suite de nombres si elle est initialisée toujours de la même façon. La suite n'est pas aléatoire , mais "pseudo" aléatoire. La simulation n'est pas très réaliste, elle est meilleure que la réalité, les PC peuvent initialiser les fonctions aléatoires avec la date et l'heure en cours qui est en perpétuelle évolution.
Si on veut améliorer le côté aléatoire de random (ce qui n'est pas toujours utile), l'initialisation peut se faire, avec la variable système **time** ou avec une boucle que l'on interrompt manuellement.
L'argument de random doit être une variable word, qui peut donner deux variables aléatoires de type byte.
Pour allumer nos leds, il nous faut un nombre compris entre 0 et 15. Pour cela nous utiliserons l'opérateur modulo, "//", qui n'est autre que le reste (r) de la division euclidienne a = bq + r
où a est le dividente, b le diviseur, q le quotient. Le quotient est donné par l'opérateur "/".

La commande "toggle" configure la broche en sortie et inverse sa valeur

Un exemple pour illustrer celà:



Deux fois plus d'animation avec cette variante:
La boucle d'initialisation est supprimée. Vous pouvez vérifier que en simulation sur PC, random fournit des animations différentes à chaque démarrage,

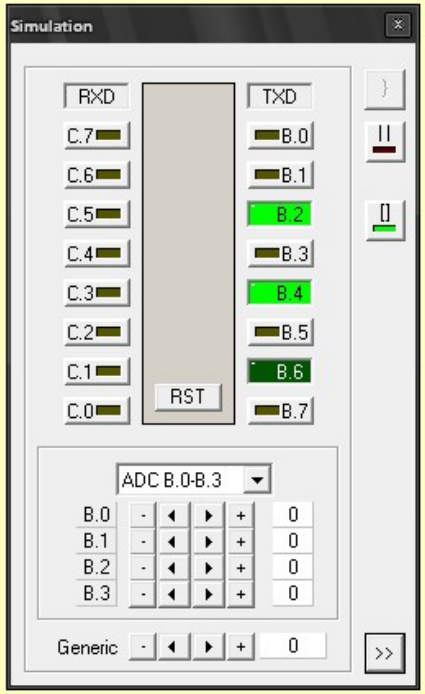


Le pseudo multitâches

Ces µcontrôleurs ne savent faire qu'une chose à la fois et lorsque le programme arrive sur une commande "pause", il s'arrête, le temps de la pause. Le pseudo multitaches, présent sur la série M2, offre une solution à ce blocage, en sautant d'une tâche à une autre, assez rapidement pour nous

donner l'illusion de les faire en même temps.
En contre partie, la vitesse de l'horloge interne est gérée par le µC, et la commande "setfreq "est rejettée. L'arrêt d'un seul des programmes arrête tout.
Chaque tâche commence par une étiquette "startx:", de start0 à start7
Les picaxes M2 gèrent jusqu'à 8 tâches (4 seulement pour le 08M2).
Voici un exemple:

```
#picaxe 20M2

symbol flag =bit0
symbol led1 =B.2
symbol led2 =B.4
symbol led3 =B.6

start0:
     do
          flag=1
          high led1
          pause 10000
          flag=0
          low led1
          pause 5000
     loop

start1:
     do
          do while flag=1
               high led2
               pause 500
               low led2
               pause 500
          loop
     loop

start2:
     do
          if flag=0 then
               high led3
               pause 250
               low led3
               pause 250
          endif
     loop
```



Trois tâches se partagent les broches et les variables.
start0 modifie la variable binaire flag et allume la led 1.
En fonction de la valeur de flag, start1 allume la led 2 et start 2 allume la led 3.
start1 et start2 donnent deux façons différentes de faire la même chose.

Voir suite page 3

Electromag1 Circuits électroniques et Picaxes

Petit site pour ceux qui bricolent en électronique en général et sur les PICAXES en particulier.
Les PICAXES sont des microprocesseurs (PIC de Microship) programmables en BASIC. Pleins de qualités, pas chers, ultra faciles à programmer.
Tout (ou presque) est sur le site du distributeur Gotronic (voir les liens).
Les pages seront ajoutées ou modifiées petit à petit.
Dans le but d'améliorer ce blog, communiquez moi vos remarques et les erreurs que vous pouvez constater.

Pour me contacter: Cliquez sur "Contact",ici ou en haut de la colonne de gauche

Pour retourner au menu, Cliquer sur "Accueil", ici ou en haut de la colonne de gauche

Bonne visite

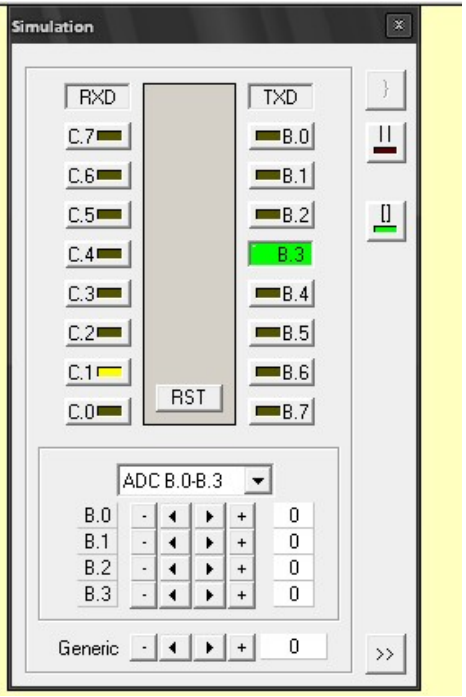
ACCUEIL
CONTACT
Livre d'or

Les interruptions

Une autre solution pour sortir d'une pause avant la fin est d'utiliser une interruption.
Lorsque les conditions de l'interruption sont réunies, le programme quitte la tâche en cours (sauf pour quelques commandes bloquantes) pour exécuter le sous programme d'interruption nommé "interrupt:"
Si l'interruption intervient pendant une pause, le programme sort de la pause, exécute le sous programme d'interruption et le "return" de ce sous programme renvoit à l'instruction SUIVANT la pause, ce qui fait sortir de cette pause..

Un exemple:

```
#picaxe 20M2

;setint place une interuuption
;quand le niveau=1, sur la broche C.1
   setint %00000010,%00000010
do
    pause 60000
    high B.3
    pause 60000
    low B.3
loop


interrupt:
do loop while pinC.1=1   ;blocage
setint %00000010,%00000010
return
```



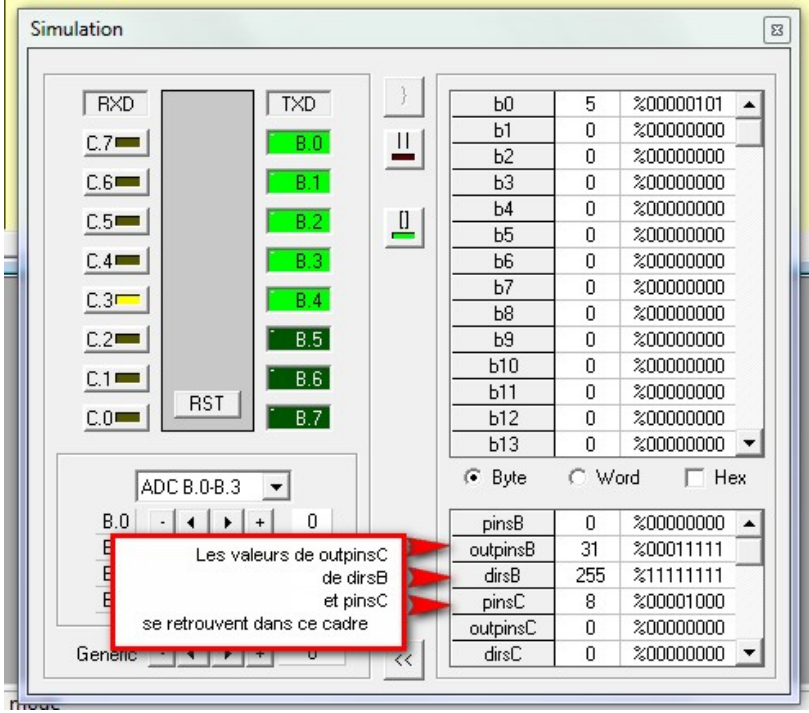
Explications:
La commande setint pose l'interruption. Elle est décrite dans les deux octets qui suivent.
Le second octet donne la broche utilisée pour déclencher l'interruption, ici : C.1
Pour le 20M2, seules les broches C.1 à C.5 peuvent être utilisées (voir doc setint).
Le premier octet donne la valeur de la broche qui déclenchera l'interruption, ici:le niveau haut.

La bouche do loop positionne la broche B.3 60 s au niveau haut puis 60 s au niveau bas.
Si la valeur de C.1 passe de 0 à 1, la pause en cours est interrompue et lorsque cette valeur repasse à 0, le programme passe à l'instruction suivante.
Chaque commande setint ne fonctionne qu'une fois. Pour pouvoir recommancer l'opération, il faut que le programme passe par une nouvelle commande setint.
En général, cette réactivation est effectuée par un setint placé à la fin du sous programme interrupt.

Un phénomène apparaît avec notre exemple:
C.1 passe au niveau 1, l'interruption envoie le programme dans le sous programme, setint réactive l'interruption, le return renvoie le programme dans la boucle principale, très bien.
Mais si C.1 est toujours au niveau haut, l'interruption se redéclenche immédiatement et le programme joue au ping pong entre la boucle principale et le sous programme, tant que C.1 est à l'état haut.
Pour cet exemple, il n'y a pas de conséquences, mais si il s'agit de compter des impulsions, il y aura certainement des erreurs.
Dans cet exemple, une boucle de blocage donne une solution (faites un essai en plaçant un ";" devant la ligne do loop).

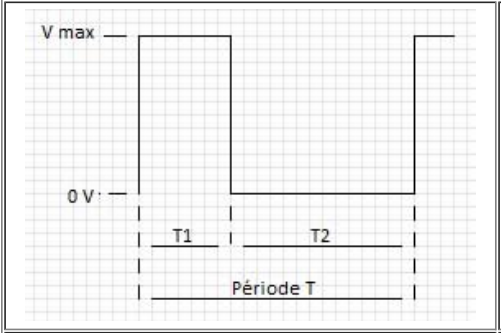
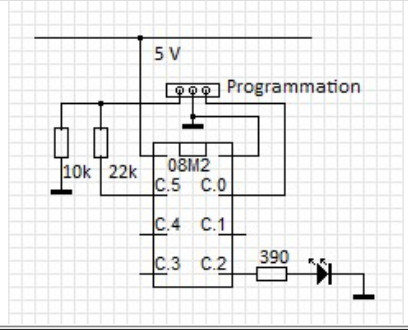
Un autre exemple, fourni par dje8269, un "débutant" venant de découvrir les picaxes:
Petit chenillard bidirectionnel avec blocage par poussoir "télérupteur".

```
#PICAXE 20M2

'Interruption sur C.3 niveau haut


setint %00001000,%00001000  'Armement de l'interruption

'############################
'Programme principal
'############################
do
    for b0 = 0 to 7
        high b0
    next
    for b0 = 7 to 0 step -1
        low b0
    next
loop

'############################
'Interruption fonction télérupteur
'############################

 interrupt:
do while pinC.3 =1 loop      ;blocage niveau haut
do while pinC.3 = 0 loop     ;blocage niveau bas
do while pinC.3 =1 loop      ;blocage niveau haut

setint %00001000,%00001000  'Re-armement de l'interruption
return
```



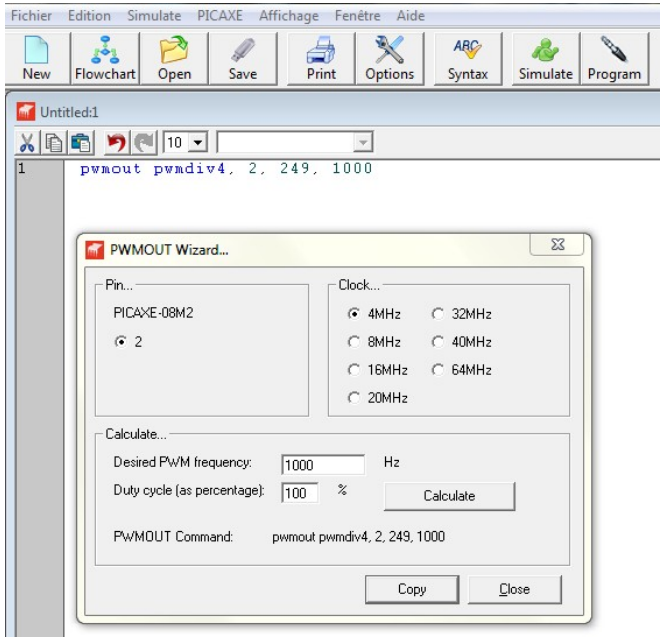
Faire varier la luminosité d'une LED (ou la vitesse d'un moteur à courant continu).

Le principe généralement utilisé est la modulation de largeur d'impulsion (MLI), plus connu sous le nom de PWM (Pulse Width Modulation in english).
On délivre sur une sortie un signal carré de période fixe T et on modifie le rapport cyclique entre T1, temps valeur haute et T2, temps valeur basse:

<table>
<tr>
<td> 
</td>
<td> 
</td>
</tr>
<tr>
<td>La modulation
est une modification du rapport T1/T
entre 0 et 100%</td>
<td>Shéma du circuit de démo
pour la vidéo ci dessous.</td>
</tr>
</table>

Ici, c'est un 08M2, le plus petit des picaxes, il était déjà sur la platine d'essais. Il n'y a qu'une sortie compatible avec la commande PWM, la broche C.2. Chaque picaxe a une ou plusieurs broches compatibles (voir doc).
Deux commandes pour gérer le PWM:
PWMOUT: Initialise la génération du signal (voir doc)
PWMDUTY: Fait varier le rapport cyclique sans interruption du signal

Pour obtenir la commande, l'éditeur fournit un utilitaire.
Cliquez sur PICAXE (bandeau)
puis Wizard, pwmout et le panneau ci dessous apparait:



Réglez la vitesse horloge
Choisissez la broche (si possible)
Entrez la fréquence (ici 1000Hz)
Le rapport cyclique (ici 100%)
et cliquez "Calculate":
La commande pwmout apparait: pwmout pwmdiv4,2,249,1000
pwmdiv4= prédiviseur de la fréquence horloge pour obtenir la fréquence choisie.
2, c'est le n° la broche
249 donnera la fréquence 1000Hz
Le dernier paramètre est le rapport cyclique. Pour 100%, il est toujours égal au premier paramètre x 4,(bon, ici l'utilitare donne 1000).
En faisant varier ce paramètre entre 0 et 1000, on obtient une variation du rapport cyclique entre 0 et 100%

Le code démo:

\\PRIMAX_CLOUD\Maison_Dossier partagé\Documents Michel\Michel\PIC\Encodeur\demo pwm.bas

```
1
2   symbol rapcycl=w13 ;nomination des variables
3
4   pwmout pwmdiv4, C.2, 249, 0 ;initialisation pwmout
5
6   do
7   ;boucle allumage
8   for rapcycl=0 to 1000 step 10    ;augmentation par pas de 10
9       pwmduty C.2, rapcycl         ;variation du rapport cyclique
10      pause 10
11  next
12  pause 2000                       ;pause 2 s au maximum 100%
13  ;boucle extinction
14  for rapcycl=1000 to 0 step -10   ;diminution par pas de 10
15      pwmduty C.2, rapcycl         ;variation du rapport cyclique
16      pause 10
17  next
18  pause 2000                       ;pause 2 s à 0
19  loop                             ; on recommence...
```

Résultat:
La simulation du PWM n'est pas possible, voici donc une vidéo:

La variable système "time":

La variable "time" est spécifique de la série M2, c'est une simplification du timer présent sur les picaxes de la série X2.

Elle compte les secondes en tâche de fond dès le lancement du programme, sans qu'on ait besoin de le lui demander. Plus précisément, elle compte les secondes aux fréquences horloge de 4 et 16 MHz, pour les autres fréquences horloge, les unités time correspondent à ce tableau:

32 MHz	16 MHz	8 MHz	4 MHz	2 MHz	1 MHz	500 kHz	250 kHz	125 kHz	31 kHz
0,5 s	1 s	2 s	1 s	2 s	16 s	32 s	64 s	32 s	128 s

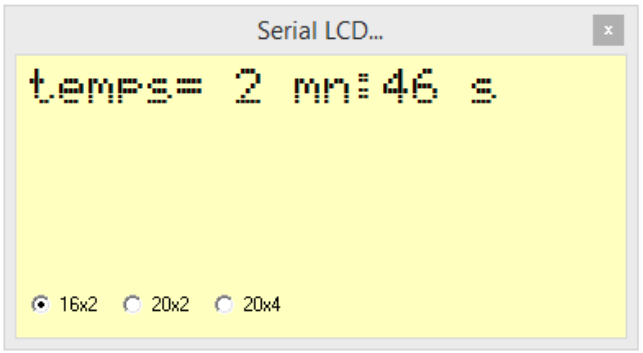
C'est une variable de type word, elle compte jusqu'à 65535 et repasse 0 sans prévenir.

L'utilisation est très simple, on peut lire (ex: W13 = time) ou remettre ce compteur à 0 (ex: time=0) à tout moment. La programmation de compteur, minuteur, et horloge en tout genre est très facilitée, pour les valeurs compatibles avec le tableau évidemment.

Le piège:

Si on veut afficher le temps qui passe sur un afficheur série on fait un petit programme tout simple, comme ceci:

```
1  #picaxe 20M2
2  symbol minut=w13
3  symbol second=b1
4        do
5              minut=time/60
6              second=time//60
7              serout C.4,N2400,(254,128,"temps= ",#minut," mn:",#second," s   ")
8        loop
```



Commmentaires:
L'opérateur / donne le quotient de la division par 60 , soit les minutes.
L'opérateur // (modulo) donne le reste de cette division , soit les secondes.
décompose la valeur qui le suit en caractères ascii pour l'affichage.
Pour afficher la fenêtre Serial LCD, il faut cocher la case dans les options de simulation et choisir la broche de sortie utilisée dans la commande.

Donc, on est content,... pas très longtemps, on s'apperçoit très vite que la variable time prend un retard croissant.
Cela vient du fait que serout et time partagent le même timer et serout perturbe d'autant plus time qu'il est appelé plus souvent. Et dans cette

boucle, la dégradation est très rapide.

Une solution boiteuse consiste à ne passer par serout qu'au changement de seconde.
Mais une meilleure solution est d'utiliser la version hardware de serout : hserout (qu'il faut configurer par hsersetup). Il n'y a plus le choix de la broche, mais il n'y a plus de perturbation.

Le code adapté:

```
#picaxe 20M2
symbol minut=w13
symbol second=b1
      hsersetup B2400_16, %10 ; 2400 baud, inverted TXD pour afficheur
                              ;hserout uniquement sur C.0
      do
            minut=time/60
            second=time//60
            hserout B2400_4,(254,128,"temps= ",#minut," mn:",#second," s   ")
      loop
```

Il n'y a pas de simulation de hserout.

Les sous programmes:

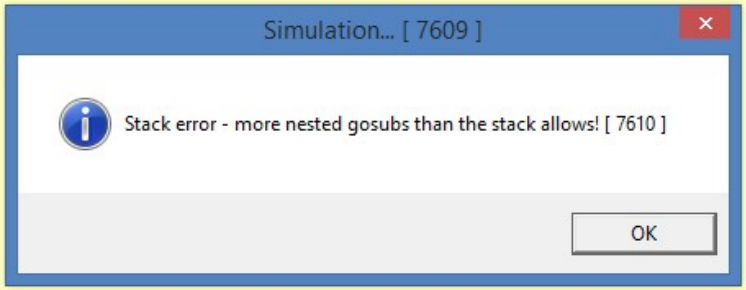
Les sous programmes permettent de structurer un programme pour le rendre plus lisible.
A l'analyse , on a intéret à découper un projet en modules intégrables dans un sous programme et exécuté sur demande du programme principale.
Les sous programmes évitent de réécrire les mêmes lignes de codes à des endroits différents du programme.

Le gosub et son return
Le sous programme est appelé par un gosub ex: subcalcul
Le sous programme commence par une étiquette se terminant par ":" ex: subcalcul:
Le sous programme se termine par un "return"

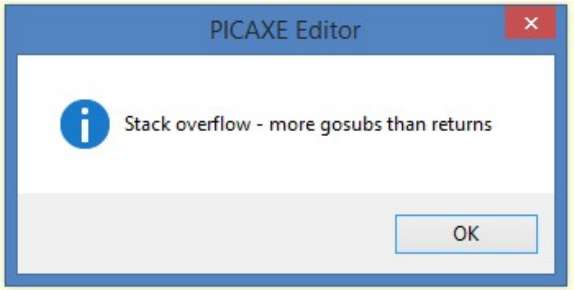
Le sous programme est un déroutement temporaire du programme, à la fin du sous programme, le programme reprend à la ligne suivant le gosub, l'adresse de retour et mémorisée dans une pile, chaque gosub a une adresse de retour, effacée en passant par le return correspondant. Si on ne passe pas par ce return, les adresses de retours font "déborder" la pile et plante le programme.
Et cela vient assez vite, le nombre de gosub imbriqués est limité à 8.
Le problème est que cette erreur n'est pas détectée lors du contrôle de syntaxe et ne le sera peut être pas pendant une simulation, mais le plantage arrivera un jour.

Ce n'est pas toujours aussi évident que dans cet exemple:

```
do
debut:
gosub test
loop


test:
goto debut

return
```



Méme punition avec l'éditeur PE6:



1 2 3