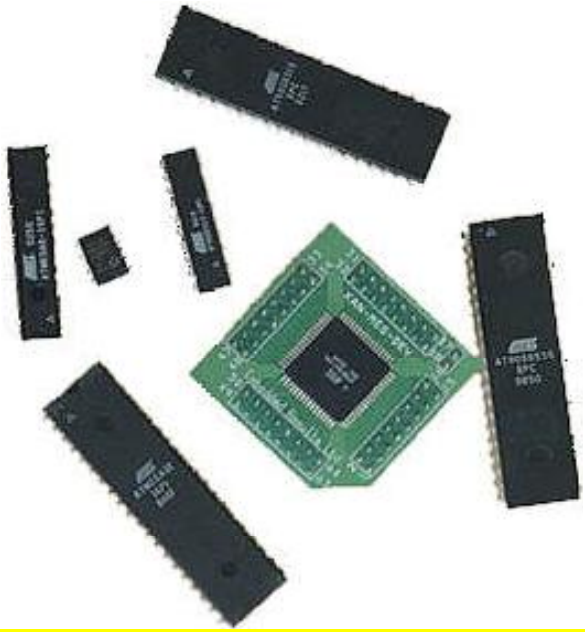
BASIC des microcontroleurs

Jean-Pierre Duval

1ere partie : Le BASIC

2eme Partie :
Le système de développement BASIC
BASCOM-AVR



3eme Partie : Le dictionnaire

4eme partie :
les microcontrôleurs
R.I.S.C. ATMEL

Cet ouvrage est un boite à outil qui nous l'espérons, vous rendra de nombreux services.
La première et la quatrième partie, le livre 1- 4 à été rédigé par l'auteur.

La seconde et troisième partie Livre 2-3 sont des adaptations du manuel de Mark Alberts, ce n'est pas une traduction mot à mot.

Nous avons cherché à faire facile, facile ne veut pas dire simpliste :
- ⇨ Le Basic, démontrer par des exemples,
- ⇨ homogénéïser la présentation,
- ⇨ donner un sens aux sigles et abréviations quand nous avons pu en trouver la signification[1] !
- ⇨ Plugin, add-on sont des mots difficilement traduisibles (ajouts ?) notre propos n'est pas d'être un élève studieux de l'académie mais d'aider les francophones à aller plus vite en apprentissage, nous garderons donc les mots classiques du «jargon électronicien »

Nous avons cherché à faire utile :
- ⇨ en rassemblant les fonctions par famille,
- ⇨ en allant à l'essentiel, les fonctions importantes sont en **gras** pour retenir l'attention.
- ⇨ Notre challenge : qu'un débutant puisse réaliser sa propre application à la fin de cet ouvrage.

Ce document se compose de trois livrets:

Livre 1	Les premiers pas et la découverte du Basic. La programmation avancée et les annexes.
Livre 2	Découverte de L'IDE[2] BASCOM
Livre 3	Le dictionnaire des instructions Basic BASCOM
Livre 4	Les microcontrôleurs ATMEL (les choix de l'auteur)

Suivant le niveau atteint, le lecteur passera de l'un à l'autre

Il n'engage en rien les auteurs et les erreurs ou oublis qui auraient pu subsister malgré une relecture attentive ne pourraient être opposées de quelques façons que ce soit.

Personne n'est infaillible, donc si vous constatez une (des) erreur(s), une (des) fautes n'hésitez pas à nous prévenir.

---

[1] Voir le site : http://www.linux-france.org/prj/jargonf/general/bgfrm.html

[2] IDE :Integrated Development Environment : éditeur, programmeur, simulateur, débugeur, compilateur, émulateur de terminal, etc..)

GENERALITES

Qu'est qu'un MICROCONTROLEUR ?

C'est un circuit intégrant une certaine quantité de mémoire de différents types où se trouvent bien rangés, les instructions exécutables (le programme), les variables, les constantes, une ALU[3] , des ports d'entrées et sorties, des registres de traitement de programmes.
Il sert à piloter, des automatismes, à être « l'intelligence » d'appareils.
On passera sur les avantages et inconvénients des différents types d'architectures, à savoir Von Neuman et Harward. Voir :
http://www.bibmath.net/bios/index.php3?action=affiche&quoi=neumann
sur les microcontrôleurs en général :http://www.commentcamarche.net/pc/cpu.php3

Par paresse et par habitude microcontrôleur sera souvent remplacé par µP ou µ-contrôleur.

Qu'est-ce qu'un PROGRAMME ?

Ou Logiciel ? C'est une suite d'opérations logiques, définie par des instructions.

Assembleur, C ou BASIC ?

Les premiers µ-contrôleurs possédaient si peu de mémoire, et leur vitesse d'exécution si lente qu'on ne pensait même pas possible qu'un jour on puisse les programmer en langages évolués. Ces langages sont, maintenant, assistés par des puissants compilateurs, qui transforment les instructions lisibles en programmes assemblés.

Basic ou C, pourquoi faire simple quand on peut faire compliqué, c'est une réponse pragmatique.
Le C à un énorme avantage il est normalisé : Ansi[4]-C
Le Basic à une référence que tout Basic se doit de respecter au plus près : Quick Basic® de Microsoft® qui à été décliné en Visual basic comme C à été décliné en C++ ou Visual C pour répondre aux besoins des interfaces graphiques : Windows®
Basic-Bascom est si proche de Quick-Basic que l'auteur a recopié des parties entières de programme par simple couper-coller.
En C ou en Basic, des utilitaires permettent de décompiler le Basic-Bascom pour avoir l'équivalent assembleur.

Pourquoi un AVR ?

La gamme AVR est très homogène, un programme écrit pour un « petit ATtiny» µP tournera sur un « ATmega », elle est basée sur l'architecture originale RISC[5] mais avec un jeu d'instructions très complet. De plus ce sont, actuellement, les µP qui disposent d'une quantité de RAM[6] suffisante pour ne pas se poser de questions lors du développement.

---

3 Unité Arithmétique et Logique.
4 American National Standards Institute
5 Reduced Instruction Set Computer
6 RAM random access memory (mémoire à accès aléatoire, volatile(pour les variables)

Pourquoi BASCOM ?

Quand on dispose d'un µp de 128 Ko de programme il faut un langage très évolué, très proche des dernières versions de QuickBasic® de Microsoft® c'est à dire du type procédural (une procédure est un morceau de programme qui peut être assimilé à une nouvelle instruction) et possédant un jeu d'instructions mathématiques complet, un débogueur[7], un compilateur[8] et un simulateur de haut niveau.

De plus BASCOM-BASIC-AVR possèdent des add-on professionnels, les librairies qui permettent entre autre de piloter les cartes-mémoire, des bus industriels comme Modbus, de fabriquer ces propres interfaces I2C…

Enfin c'est un produit SIMPLE et le compilateur produit un code-source très rapide, quasiment équivalent à l'assembleur.

7 Programme permettant de déceler les erreurs de syntaxe.

8 Programme transformant les instructions Basic en langage « machine »

LE BASIC

Pour afficher 50 fois « bonjour » on peut écrire 50 fois Print « bonjour » c'est fatigant et c'est très consommateur de mémoire, on peut aussi écrire un programme qui fera ce travail pour nous :

```
For j=1 to 50
        Print « bonjour »
Next
```

3 lignes contre 50, pas mal ! je vous sens intéressé par la programmation…

Détaillons ces trois lignes :

For (en français POUR)[9] J une variable qui prendra la valeur de 1 à 50
Print « Bonjour » dans une boucle For…Next, pour une question de lisibilité du programme, on indente la partie concernée qui peut tenir sur plusieurs pages de programme !
Print est une instruction
Next (J suivant)

Ces trois lignes viennent de nous apprendre trois notions importantes
1) **L'instruction** qui va par exemple signaler au µP qu'il doit imprimer « bonjour »
2) La boucle FOR…NEXT , un instruction **inconditionnelle** qui sera étudiée plus complétement dans le chapitre consacré à ce sujet.
3) La **variable** (ici représentée par la lettre J)

Nous allons démarrer l'apprentissage du Basic par l'étude des variables, mais il faut commencer par le commencement.

Le calcul binaire

les bases

Un µ-contrôleur fonctionne seulement avec des 0 et 1, on ne va pas entrer dans les détails de l'algébre de Boole[10] mais la question est : comment faire tourner un µP avec seulement un 1 et un 0 ?
Ici intervient le calcul binaire = base 2

Un nombre décimal (base 10) peut se décomposer en puissance de 10 :
$235=(2*10^2)+(3*10^1)+(5*10^0)$ ou encore 200+30+5

Le même nombre en binaire s'écrira décomposé en puissance de 2 :
$(1*2^7)+(1*2^6)+(1*2^5)+(0*2^4)+(1*2^3)+(0+*2^2)+(1*2^1)+(1*2^0)$

128	+	64	+	32	+	0	+	8	+	0	+	2	+	1	= 235
1		1		1		0		1		0		1		1	= 235

Pour exprimer ce nombre on a utilisé 8 poids ou puissance de 2 (7-6-5-4-3-2-1-0)
Un poids = un bit, un bit=1 si sa valeur multipliée par son poids est ≠ 0
8 bits = un byte (un octet)
Avec 8 bits on peu compter 256 valeurs (de zéro à 255)

---

9 nous conseillons à chacun un petit ditionnaire d'anglais en ligne genre 7 dicos de Microapplication®

10 George Boole, anglais 1815-1864

Les µ-contrôleurs ATMEL travaillent avec des registres, des adresses, et des emplacements mémoires sur 8 bits.

Les PC travaillent en 32 bits, les consoles de jeux en 64 bits .

Depuis l'origine des ordinateurs la place mémoire est chèrement payée, alors on ruse. Il existe une technique de compactage des données, puisque 4 bits permettent de compter jusqu'à 15, De 0 à 15 il y a 16 nombres (on travaille alors en **hexadécimal**. = base 16) : 0,1,2,3,4,5,6,7,8,9,A,B,C,D,F suivant la table ci-dessous :

Poids →	**Hexa ↓**	**8**	**4**	**2**	**1**
Nombre ↓	**Puissance 2 →**	**3**	**2**	**1**	**0**
0	0	0	0	0	0
1	1	0	0	0	1
2	2	0	0	1	0
3	3	0	0	1	1
4	4	0	1	0	0
5	5	0	1	0	1
6	6	0	1	1	0
7	7	0	1	1	1
8	8	1	0	0	0
9	9	1	0	0	1
10	A	1	0	1	0
11	B	1	0	1	1
12	C	1	1	0	0
13	D	1	1	0	1
14	E	1	1	1	0
15	F	1	1	1	1

Notre nombre décimal 235 n'occupe que 1 octet (2 x 4bits) qui s'écrit : &HEB
(E)+(B)=1110+1011 **E** étant l'expression des MSB[11] et **B** des LSB
&H ne sert qu'à indiquer que la variable est un nombre Hexa, à comparer au nombre binaire. **&B**11101011

il est temps d'aller boire un bon 51966…ou si vous préférez **&H**CAFE qui sera moins indigeste qu'un **&B**1100101011111110, je crains fort que tout cela ait le même goût !

les Basic évolués ont des instructions pour passer les variables d'une base à l'autre.

Dans la littérature électronique en général, et donc dans les exemples proposés par BASCOM, on ne compte pas (en binaire) de 1 à 8 mais de 0 (poids le moins signifiant qui peut prendre la valeur 1 ou 0) à 7 (poids le plus signifiant qui peut prendre la valeur 127 ou 0)
Les ports de microcontrôleurs sont nommés d'une manière identique:
PA0, PA7 …

Pour les nombres plus grands que 255 on utilisera des combinaisons de 16 ou 32 bits.

[11] MSB Most significant bits(Plus représentatif) - LSB less significant (moins représentatif) abbrev. très utilisées.

Les opérateurs de l'algèbre de Boole

1+1=2 c'est vrai en base 10 mais en base 2 on ne peut pas dépasser 1 ! On découvre donc de nouveaux opérateurs.
→ L'addition logique :
1+1=1 en logique Booléenne, on dit 1 ou 1 égale 1
1+0=1
0+1=1
0+0=0 Cette suite d'opération est appelé une **table de vérité**

si A=1 et B=1
A+B=1
Pour être plus clair en Basic on remplace le + par le mot **Or**
si A=1010 et B=0101
A **Or** B =1111 : les bits appartenant à A **ou** à B se retrouvent dans le résultat
si A=0011 et B=0110
A **Or** B= 0111

La multiplication logique :
1 . 1 =1
1 . 0 =0
0 . 1 =0
0 . 0 =0
En basic on remplace le . par le mot **And**
si A=0100 (4) et B= 0101(5)
A **And** B =0100 seuls les bits **en commun** se retrouvent dans le résultat.

Deux opérateurs qui n'existent que dans l'algèbre de Boole :
Not : complément logique
si le byte A = 00000111 (7)
B = **not** A
B = 11111000 (248) et : 248+7=255

XOR : (**Ou exclusif**)
si A=1010 et B=1001 A **Xor** B=0011

```
  1010
XOR 1001
  0011
```

Je suis né le 03-04-50
XOR 03-04-06 mon anniversaire
00 00 56 mon âge !

cela ne marche pas pour tout le monde !

les bits identiques sont mis à 0, les bits 0 identiques restent à 0, à la différence du NOT.
Ce dernier opérateur étant particulièrement ardu à utiliser par les débutants.

Il est évident que ce n'est pas en deux pages que l'on peut résumé le travail de George Boole, mais ces quelques notions sont suffisantes pour aborder la programmation.

Ne pas confondre l'addition logique avec l'addition mathématique, les langages évolués ont des outils de traductions et on sait faire 1+1=2 avec un µP, étonnant non !

Les opérateurs

Voici les opérateurs que nous rencontrerons et utiliserons dans nos programmes :

Arithmétiques

+ addition, - soustraction, * multiplication, \ division entière, MOD modulo(reste de la division entière), ^ élève à la puissance. / division. (ce dernier est à utilisé avec les variables de type Single [voir le chapitre qu'est qu'une variables])

Mathématiques

Les fonctions logarithmiques et trigonométriques sont étudiées dans le cadre du dictionnaire et de la "Programmation avancée".

Relationnels

= égal, <>différent, >plus grand que, <plus petit que, <=plus petit ou égal, >=plus grand ou égal.
Exemples : J >= K, K <= J

Booléen

Or, And, Not, Xor, (voir plus haut)

Spécifiques

J étant une variable quelconque

J=J+1	ce qui signifie que	J prend la valeur de J incrémenté de 1
Si J=12	J=J+1 J=13	
J=J x 12	si J=12 J=144	J prend la valeur de J multiplié par 12

Basic Bascom-AVR

Comme tout langage, le Basic possède une syntaxe, des mots, des patois…et des habitudes. Nous nous intéresserons plus spécifiquement au BASIC BASCOM-AVR.

Les bonnes habitudes

- Commercer les programme par une entête de remarques avec la date, les modifications et leur date, et le sujet du programme.
- Commencer par définir vos variables avec des mots explicites, en français, ce qui évite d'utiliser des mots réservés : pression, couleur, plutôt que P et C.
- Les variables génériques comme A, J, K, utilisées dans les boucles sont à remettre à 0 avant utilisation.
- Prendre le temps de décrire l'application, de rechercher les points difficiles, et commencer par résoudre ceux-ci.
- Indenter les boucles, les programmes sont plus lisibles et le débogage plus facile.

- Nettoyer le programme, quand il tourne.

- Construire des procédures et fonctions pour limiter le corps du programme.

- Faire court : Si le nom des variables doit être explicite, la longueur du nom ne rentre pas en ligne de compte dans la compilation, leur dimension doit être au plus juste, ne pas utiliser des "integer" quand on peut utiliser des "byte", par exemple.

Attention aux variables tableaux très gourmandes de place.

- Les variables génériques sont déclarées une seules fois donc elles utilisent une seule places en mémoire RAM, en revanche leur utilisation est liée à des règles impératives,

Remettre à zéro avant utilisation, Vérifier qu'elle ne sont pas utilisées deux fois (ou plus) dans une même boucle.

- Utiliser les mémoires à bon escient, la mémoire EEPROM peut accueillir les constantes, les data ou des variables de données qui ne changent que très rarement.

- Mettre des « Rem[12] » régulièrement, pour expliquer ce que vous faites et pourquoi. Les REMarques ne rentrent pas dans le programme compilé.

- Reprendre un programme 10 ans, 15 ans après sa création n'est pas impossible (cela est arrivé souvent à l'auteur de ces lignes).

- Reprendre un programme écrit avec un langage pour le réécrire avec un autre, cela aussi arrive de temps en temps.

Et enfin le pire:
- Reprendre un programme écrit avec un langage qui est moins familier (Pascal ou C) écrit par un autre programmeur… l'horreur!

Une seule solution : **expliquer**

Les rem ou remarques sont donc très utiles pour l'avenir.
C'est aussi un outil fantastique de programmation, pendant la mise au point (le débogage) en éludant une partie du programme qui ne fonctionne pas on retrouve la partie déficiente.
BASCOM est très riche pour cette non-instruction puisque les REM peuvent être de trois types:

```
REM ceci est une remarque
' ceci est aussi une remarque
'(   ceci est le début du bloc que je mets en remarque
        instruction
        instruction   ces deux instructions ne seront pas exécutées
et ceci en est la fin :
')
```

Une commande permet de mettre en rem tout un bloc surligné puis de le réinstaller d'un coup de souris (dernière option de Edit dans la ligne de menu déroulant de Bascom)
Toutes les instructions ou commandes Basic, exposées ci-dessous, sont reprises dans la partie dictionnaire.
Un piège à éviter : ne pas mettre de parenthèses dans les rem, cela perturbe le compilateur.

---

[12] Rem=remarque voir ce mot dans le dictionnaire des instructions et au chapitre "struture logique d'une application"

Qu'est ce qu'une variable ?

Une variable est un objet défini par le programmeur, cet objet numérique (même les mots sont des valeurs numériques!) change d'état pendant le déroulement du programme.
Un petit exemple :
Un bouton qui change d'état fait varier un port d'entrée de 0 à 1. On va associer ce bouton à une variable et déclencher un process. (par exemple la fermeture d'un relais).

```
Dim Boutonstop as bit          rem déclare la variable Boutonstop en tant que bit
Do                             rem boucle do loop until (faire  tourner jusqu'à)
      Boutonstop=pind.5        rem pind.5 est la 5° patte du port D
loop until Boutonstop=0        rem boucle jusqu'à ce que pind.5 soit mise à 0
```

---

les types de variables

Attention le mot « Type » est une notion importante on doit **TYPER** les variables avec les instructions **Dim**, **Local et Function**

- **BIT** : dans l'exemple ci-dessus, la boucle attend que le boutonstop soit basculé, la variable peut prendre 2 états : 1 et 0.
- **BYTE** : Avec 8 bits on compte jusqu'à &H**FF** soit 255 la variable associée est du type Byte.
- **INTEGER, WORD** : Avec 16 bits on compte jusqu'à &H**FFFF** soit 65535. Suivant l'utilisation on a les variables signées du type **Integer** de –32768 à+32767 ou du type **Word** de 0 à 65535
- **LONG** : Avec 32 bits (&H**FFFFFFFF**) on compte de –2.147.483.648 à +2.147.483.647 et on aura les **long** (correspondant aux Integer)
- **SINGLE** ; ces variables signées et stockées sur 32 bits, permettent de calculer de $1.5X10^{-45}$ à3.4×10^{38} mais la quantité de chiffres significatifs est limité !
  Les variables de types Single permettent de travailler en virgules flottantes (voir opérateurs mathématiques)
- **DOUBLE** : Variables signées sur 64 bits comme les Single mais de –E308 à E308 !

Nécessite l'implémentation de la librairie double.lbx fournie.

- **STRING** : Ce sont les lettres, mots et phrases jusqu'à 254 bytes, leurs longueurs sont déclarées : ex : Discours as string * 10. (les strings occupent 1 byte de plus que leur déclaration.) Voir chapitre "Conversions de variables"

Il reste un type de variable spéciale :

- **TABLEAUX (ARRAYS)** ce sont des groupes de variables indexées ;

Elles regroupent sous un même nom des variables qui seront appelées par un index.
Elles sont déclarées par un nom suivi par le nombre d'index entre ( )
Le nombre d'éléments(index) doit être inférieur ou égal à 65535
Attention l'index 0 n'existe pas. (voir le programme Array.bas) (contrairement au QuickBasic)
Par exemple la température en fonction de l'heure :

```
Dim temperature(24) as integer  ' les températures peuvent être négatives
Dim J as byte
Temperature(1)=10
Temperature(2)=15
Temperature(3)=portA
Temperature(J)=20   ' J (l'index) est une variable pouvant aller de 1 à 65535
                    ' mais dans cet exemple J ne peut dépasser 24 !
```

Les tableaux peuvent être de tout type, **string** y compris. (Sauf bit)

Ex : Dim Mot(5) as **string * 10** 'déclare un tableau de 5 mots de 10 caractères

Les tableaux ne peuvent pas être dimensionnés en tant que **Local**.

Nom de variables

La déclaration des variables est obligatoire. **Dim** déclare les variables globales du programme principal, **Local** les variables des SUB et fonctions.

Le nom peut avoir de 1 à 32 caractères, le premier caractère doit être une lettre, l'usage des mots réservés n'est pas autorisé sauf quand ils sont incorporés dans le nom de variable : oper**AND**e est accepté. (donc utiliser le français, mais sans accent !)
Pour mes variables génériques (celles qui servent dans les boucles) j'utilise J,K,Y et i (En BASCOM pour cette dernière c'est une mauvaise habitude car I (i) = l (L))

Il n'y a pas de différence entre les minuscules et les majuscules : PinStop = pinstop
(dans les dernières versions de Bascom les variables sont automatiquement réécrites avec une majuscule en entête, suivie de minuscules – voir option dans environnement

Certains BASIC utilisent des suffixes pour caractériser les variables : $,%, &
Dans le Basic Bascom, il vaut mieux éviter. Par exemple : mot$, permet seulement de se souvenir que mot$ est une variable **string**. Varinteger% n'est pas autorisé, il est utilisé dans les déclarations de data[13].

Imputer une valeur à une variable

Une variable peut prendre :

- La valeur d'un port :

```
Config portD = input        rem configuration des ports
Pression=portD              rem la valeur de la pression est lue sur le portD
```

- Une valeur de départ

```
Pression=23
```

- Une valeur calculée

```
Pression=pressotat1+pressotat2
```

- Une valeur incrémentée

```
Pression=pression+pressostat1
```

On peut imputer une valeur de variable à une autre variable d'un type différent en respectant la logique de type (la valeur de la variable qui donne doit être plus petite que la portée de la variable qui reçoit) Exemple :

```
Si Var_long =256
Var_byte=var_long    'résultat var_byte =1 car on dépasse la capacité de la variable. (de 0 à 255)
```

Mais on peut utiliser cette propriété pour assigner les LSB d'une var_long

---

[13] voir plus loin (constantes et data) et à ce mot dans le dictionnaire.

Autre restriction :
Resultat=Var_long * var_byte ne fonctionne pas, les opérations (voir ci-dessous) ne fonctionnent qu'avec des opérateurs du même type.
resultat=var_long1* var_long2 rem est correct

varlong2=var_byte 'assigne la valeur de var_byte à var_long2
resultat=varlong1*varlong2 'est correct

Jusqu'à présent les variables présentées sont du type **entier**, nous aborderons les variables à **virgules flottantes** dans le chapitre « Programmation avancée »

De même nous apprendrons ce qu'est une **constante** et une **data** en fin de cours

---

Questions : il peut y a voir plusieurs réponses mais une seule est bien adaptée.

1) La température de fusion d'un alliage est de 1227° quel va être le type de variable utilisé pour les calculs
A String
B Single
C Integer

2) l'eau de mer gèle à –9° quel va être le type de variable utilisé pour les calculs ?
A byte
B Word
C Integer

3) l'eau distillée gèle exactement à 0.0°C quel va être le type de variable utilisé pour les calculs ?
A Single
B Word
C Integer

4) un seul nom de variable est correct pour chaque proposition expliquer pourquoi:
A 0tello
B Otéllo
C Otello

5) temperature consomme plus de Sram que T (ces deux variables ayant le même type)
A non c'est pareil
B temperature est plus gourmande
C temperature n'est pas un nom de variable accepté

Réponses :

Question 1 : integer, single est possible mais consomme 2 fois plus de mémoire

Question 2 :Word, byte et integer n'acceptent pas les nombres négatifs

Question 3 : single, puisque on demande une précision avec au moins 1 chiffre après la virgule

Question 4 : Otello, 0tello commence par un zéro ! et Otéllo à un accent !

Question 5 : A = c'est pareil en effet, une variable est une représentation virtuelle d'un emplacement mémoire, défini par son adresse et son type.

Les instructions de boucles, conditionnelles et inconditionnelles

Un programme est en général, composé d'une boucle principale, et de n boucles.
Nous sommes gâtés ! il y a l'embarras du choix.

Les boucles conditionnelles

- **Do … loop until** Faire… jusqu'à (une condition soit remplie)
- **While….Wend** Tant que(la condition n'est pas remplie) fin

Ces instructions très communes dans tous les langages, exécutent une série d'instructions jusqu'à ce que la condition soit remplie (vraie)

La boucle **Do loop until** exécute **d'abord** les instructions avant de vérifier les conditions.
Exemple :

```
Do
        Touche=Pind          'la valeur lue sur les broches du port D est transmise à la
                             'variable Touche
Loop until touche=4          'vérification de la condition
```

La boucle **While Wend** contrôle d'abord si une instruction est vraie avant de réaliser les instructions et avant de sortir seulement si la condition est vraie.

```
A=0                          'A mise à 0
While A <= 10                'si A est plus petit ou égal à 0 (la condition)
        Print A              'imprimer A
        Incr A               'Incrémenter A
Wend                         ' fin de while (while-end)
```

Les boucles inconditionnelles

- **For..to…Next** pour une valeur… jusqu'à une autre…suivant

elle réalise la même série d'instruction autant de fois que demandée.

```
For  J=0 to 15
        PortB=J      'donne au port B la valeur de J  (exemple : allumage des 4  'premières
                     diodes)
 Next  J             'l'indication J n'est pas obligatoire (sauf dans le cas de boucles
                     'imbriquées)
```

On peut varier les plaisirs :
1) Faire varier le pas de J (dans ce cas il faut donner la longueur de ce pas)

```
For J=0 to 16 Step 2
        PortB=J
Next
```

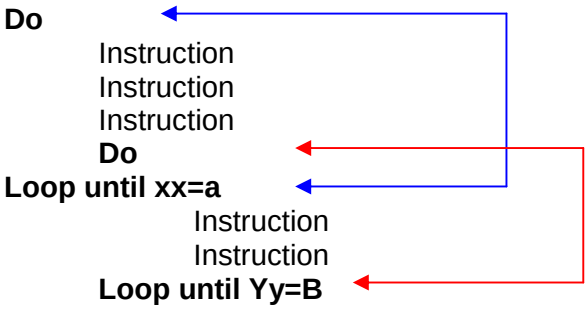
2) La variation peut être négative (**step** -5)
3) Les valeurs de début ou de fin peuvent être des variables

```
For J=temperature to j=tropchaud step 3
        PortB=J
next
```

La pratique de la programmation conduit à utiliser l'une ou l'autre de ces boucles.

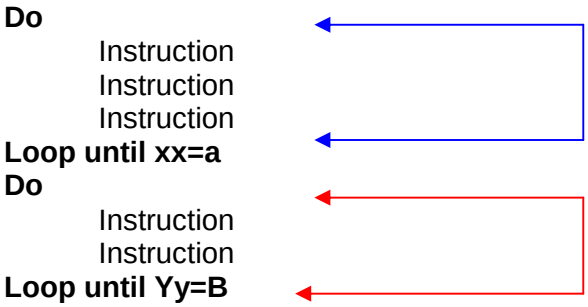
Remarques sur l'utilisations des boucles

1) L'imbrication de trop de boucles rend illisible le programme,
2) Un boucle ne peut pas se terminer hors de son champs, exemple :

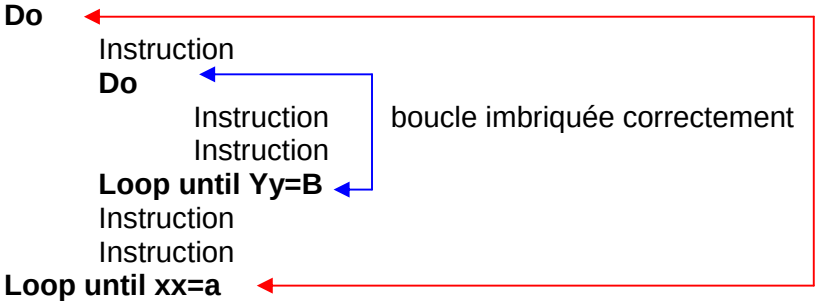


Ce programme est faux, il risque de donner des résultats…bizarres

---



Est correct.



Ces exemples données avec **Do…loop until** peuvent bien sûr, exister avec n'importe quelle combinaisons de boucles.

3) la boucle **do…Loop** peut être sans fin, c'est souvent le cas dans le cas de la boucle principale, on oublie le test de condition.

```
Do
        Instruction
        Instruction
Loop
```

- **Exit** permet de sortir de ces boucles. Il faut le faire suivre du type de boucle ;

Exit For, Exit Do, Exit While.

Les instructions conditionnelles

Encore des instructions fondamentales et classiques.

- **IF…then…Elseif..then… Else…end If**
  Si alors.. sinon-si… sinon…fin de condition Exemple :

```
Pression=portD
IF pression>200 then
        Print « Alerte »
        PortD=128                       'fermeture des vannes
Elseif pression<50 then
        PortD=96                        'démarrage chaudière
Elseif pression<190 and pression >100 then
        PortD=15                        ' régulation
Else                                    'pression inf. à 100
        Print « tout va bien »
End if
```

Ne pas oublier **Then** et **End If**

On peut aussi écrire cette instruction sur une seule ligne quand elle ne comporte qu'un choix :
If pression>200 **then** Stopmachine=0 **endif**
Les choix suivant sont facultatifs
Elseif doit être suivi d'une autre condition et de **then**
Else

- **Select …Case…End Select** Si la variable=Case(variable) réalise…fin

Exécute une ou un groupe d'instruction quand la variable sélectionnée correspond à un Choix (CASE) , exemple :

```
Pression=portd
SELECT CASE pression
CASE =50
        Print « démarrage chaudière)
        instruction
CASE =100
        Print « parfait »
CASE =150
        Print « régulation »
        instruction
CASE=200
        Print « alerte »
        instruction
END SELECT
```

On peut aussi écrire
CASE 50 TO 100 de *50 à 100*
et
CASE IS > 50 *si case est >50*

Les branchements inconditionnels

La différence entre ces branchements et les précédents est expliquée dans le titre !
Si les deux premiers sont un héritage du Basic ligné : numéro de ligne avant instruction (basica d'IBM®, GWBASIC de Microsoft®) la première instruction est à fuir ; en effet, c'est un joyeux « bor..el » quand plusieurs goto s'entremêlent. On appelle cela de la programmation-spaghetti, du nom des coups de stylos tracés sur les listings des programmes de nos ancêtres.
Pour la petite histoire, Goto n'existait pas dans les premières versions de Pascal, donc on peut éviter…

- **GOTO**

Quand le programme rencontre cette Instruction, il saute à l'étiquette indiquée.

```
Dim J as byte
J=0
encore:                    une étiquette doit se terminer par «  : »
J=J+1
IF J<10 then
        Goto encore
endif
```

Ce programme peut s'écrire

```
J=0
Do
        J=J+1
Loop until J=10
```

Imaginez la multiplication des **Goto**, appelant des étiquettes dans tous les sens !
De plus quand on fait un goto pour sortir d'une routine (Gosub sub ou function) on ne vide pas la pile du micro-controleur, cela fonctionne un peu, puis le programme se plante : Dépassement de Pile (stack overflow) ce qui ne génère pas d'erreur de compilation.

- **GOSUB**

La différence entre Gosub et les sous-routines est infime, nous approfondirons les sous-routine plus loin, un gosub ne passe pas d'argument.

Une **Routine** s'écrit à la fin du programme, on y accède par **Gosub** étiquette et on retourne dans le programme par **Return**

```
J=0
Do
        J=J+1
        If J=5 then
        GOSUB imprime
        endif
Loop until J=10
End                     rem fin du programme
'----------------début de rédaction des gosub-------------------------------
Imprime : 'une routine se termine par :
Print « 5 »
Return
```

Les étiquettes sont affichées dans une fenêtre du compilateur BASCOM.

Autres instructions conditionnelles ou inconditionnelles

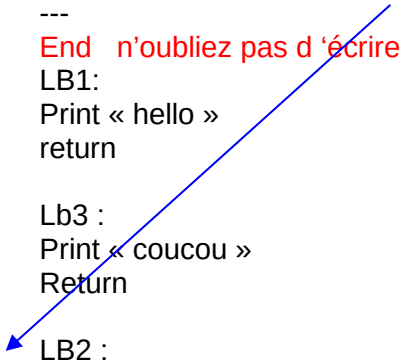
- **Call**
  Cette instruction permet d'appeler une **SUBROUTINE** nous la décrivons au chapitre suivant.

- **On….Value [Goto] [Gosub]**
  Une instruction très performante proche de Select Case, elle mélange instructions conditionnelles et inconditionnelles. Exemple :
  Dans cette instruction La variable démarre à 0 donc si J=2 le Gosub concerné est LB3. (0,1,2)

```
Dim J as Byte
J=2
On J gosub LB1,   LB2,  LB3
---'              ^^0    ^^1   ^^2
---
End   n'oubliez pas d 'écrire les étiquettes de sous-programmes après le END
LB1:
Print « hello »
return

Lb3 :
Print « coucou »
Return

LB2 :
        Print « bonjour »
Return
```



- **On…Interrupt (voir programmation avancée)**

- **Return**
  Cette instruction est obligatoire à la fin d'une SUBROUTINE (CALL SUB, CALL FUNTION) ou d'une ROUTINE (ON…GOTO, ON…GOSUB, GOSUB)

- **Start et Stop**
  Instruction de démarrage et d'arrêt d'exécution de « device » (composants du µ-contrôleur comme Timer, Counter, ADC) voir programmation avancée.

- **END**
  Arrête le programme et désactive les éventuelles interruptions.

Questions

1) Dans le programme suivant, il y a une erreur, expliquer :

```
Dim Resultat as byte          'noter la déclaration de variables
Dim J as Byte
Resultat = 1              'resultat doit être au minimum à 1
For J=1 to 10
        Resultat = Resultat * J
        Print Resultat
Next
End
```

---

2) cette boucle ne se termine jamais pourquoi ?

```
Dim Resultat as byte, J as Byte 'noter la déclaration de variables plus concise
J=12
Resultat=144
while J <>resultat
        resultat=resultat/2
        print resultat
wend
End
```

---

3) dans l'exemple ci-dessus quelle valeur maximum peut prendre *resultat* pour que la boucle s'arrête ?

---

4) un piège classique…

```
Dim resultat as single, mesure as word, canal as byte, J as byte
Config Adc = Single , Prescaler = Auto      'les Config sont des outils extraordinaires
Start Adc                                   'pour alimenter le Port analogique

canal=3                                     'le canal analogique à lire
Do
        For J=1 to 10
                mesure =getadc(canal)
                Resultat=resultat+mesure    'en additionnant 10 mesures on augmente la
                                            'précision
        next
        resultat=resultat/10                'moyennage des 10 mesures
        if resultat=1000.0 then exit do
loop
```

Dites pourquoi, malgré que la valeur de *resultat* soit proche de 1000, il y a peu de chances qu'on sorte de cette boucle ?

Réponses :

Question 1 : résultat s'accumule et dépasse la contenance d'un Byte.

Question 2 : 144/2=72, 72/2=36, 36/2=18, 18/2=9 donc la variable resultat n'est jamais égale à 12

Question 3 : 192/2=96, 96/2=48, 48/2=24, 24/2=12….192 CQFD

Question 4 : suivant la valeur attendu, par excès ou par défaut, il faut utiliser:
If resultat >=1000.0 then exit do ou <= car avec une single il y a n chiffres après la virgule !

Les Procédures et Fonctions

Ce sont des « bouts » de programmes indépendants qui fonctionnent comme des instructions paramétrables :
Elles permettent de créer ses propres instructions, liées ou non à des paramètres.

- **La procédure**

  Imaginons que toutes les minutes, votre programme modifie la pression, et que suivant l'heure, on modifie les paramètres de celle-ci :
  La création d'une procédure « verification » avec comme variables les paramètres à modifier serait bien utile.

  Une procédure (**SUB**) doit être déclarée de la façon suivante :

```
Dim Pression As Integer , heure as byte              ' déclaration des variables
Declare Sub Verification(byval Pression as Integer, heure as byte)
      Pression = 255                                 'déroulement du programme
Print Pression                  'imprime 255
Call Verification(pression)             'appel de la procédure
Print Pression                  'imprime 255
End                 'end program  attention les sub et fonction sont écrite après cette
ligne !

Sub Verification(pression As Integer, heure as byte)       'procédure
    Pression = Pression - 10
    Print Pression                      'imprime 245
   Waitms  200                                  ' etc…
End Sub
```

  Attention :
  Byval transmet à la SUB la valeur de la variable, au retour la variable n'a pas changée.
  Byref transmet à la SUB l'adresse de la variable, au retour la variable a été transformée par la SUB, elle est contenue à l'adresse !
  Si **Byval/byref** sont omis **Byref** est utilisé par défaut.

- La fonction (**Function**)

  Comme la Sub, elle doit être déclarée dans l'entête du programme, **Byval/byref** fonctionne de la même façon.
  Ce qui diffère :
  La déclaration doit être suivie du type retourné :

Declare function lumiere (byval candela **as integer**, mot$ **as string) as integer**

  L'appel de fonction se fait directement :

```
J=lumiere(K , Y$)
```

En fin de rédaction ne pas oublier le **END FUNCTION**

Remarques sur les Sub et Fonctions

- Les SUB et Function peuvent posséder leurs propres variables qui seront déclarées en **Local** (elles n'affecteront la SRAM que pendant l'appel de la procédure ou fonction)

- Ni les bits, ni les tableaux ne peuvent être déclarés en local.

- Call n'est pas obligatoire pour appeler une sub, dans ce cas on omet les parenthèses
Verification Pression**,** heure
- Exit est utilisable dans les Sub et fonctions de manière identique aux boucles :

Exit Sub
Exit Function.

- Les constantes doivent être passées ByVal.

Qu'est-ce q'une DATA ?

C'est un nombre, un mot, qui ne sera pas utilisé comme variable, voir le programme READDATA.BAS qui est très explicite.
En voici une partie.

```
'-----------------------------------------------------------
'                   READDATA.BAS
'           Copyright 1999-2002 MCS Electronics
'-----------------------------------------------------------
Dim A As Integer , B1 As Byte , Count As Byte
Restore Dta1                              'pointe le marqueur sur la data
For Count = 1 To 3                        'pour le nombre d'items en DATA
  Read B1                                         'lit la data
  Print Count ; "  " ; B1                         'imprime la data
Next                                              'suivant

End                                               'fin du programme
'début d'écriture des data.

Dta1 :
Data &B10 , &HFF , 10
Dta2 :
Data –1000%, 525%
```

remarque : les valeurs du type INTEGER (>256 ou>0) sont exprimé avec le signe % en suffixe (-1000%,525%). Les LONG avec &
Les variables attachées au data doivent être du même type
Ex : B1(byte) pour data &B10(2 en binaire), &HFF(255 en Hexa), 10 en décimal.

Qu'est qu'une CONSTANTE ?

Plus souple à utiliser que la DATA, elle fonctionne d'une façon identique SANS CONSOMMER DE LA MEMOIRE PROGRAMME

```
Const S = "test"
Const A = 5                    'déclare les constantes
Const B1 = &B1001
Const X = (b1 * 3) + 2         'ou utilise une expression pour assigner une constante
Const Ssingle = Sin(1)
```

Quand une valeur est assignée à une constante, cette valeur sert ensuite de référence, PI, Euro…

Qu'est qu'un ALIAS ?

Alias fonctionne un peu comme une constante mais pour les ports.

```
Dim direction as byte
Config Pinb.1 = Output
Direction Alias Portb.1        'maintenant vous pouvez agir sur Pinb.1 avec la variable
                               'direction
Do
  Set Direction                'fonctionne comme SET PORTB.1
  Waitms 1
  Reset Direction
Loop
End
```

Direction est plus parlant et plus clair que Pinb.1

Ce que BASCOM-BASIC ne sait pas faire

Bascom-Basic a un gros défaut il ne sait pas résoudre des opérations multiples
Exemple :
Distance = Sqr((x_cible - X_robot) ^ 2 +(y_cible - Y_robot) ^ 2)

ne fonctionnera pas
il va falloir décomposer la ligne:
J= *x_cible – X_robot*
J=J^2
K= y_cible - Y_robot
K=k^2
L= j+K, on peut utiliser les « fausses lignes » avec :
J= *x_cible – X_robot : J=J^2 : K= y_cible – Y_robot : K=k^2 : L=J+K : Distance=Sqr(L)*

Quand on a compris ces fondamentaux on a compris 70% du Basic et de la programmation.

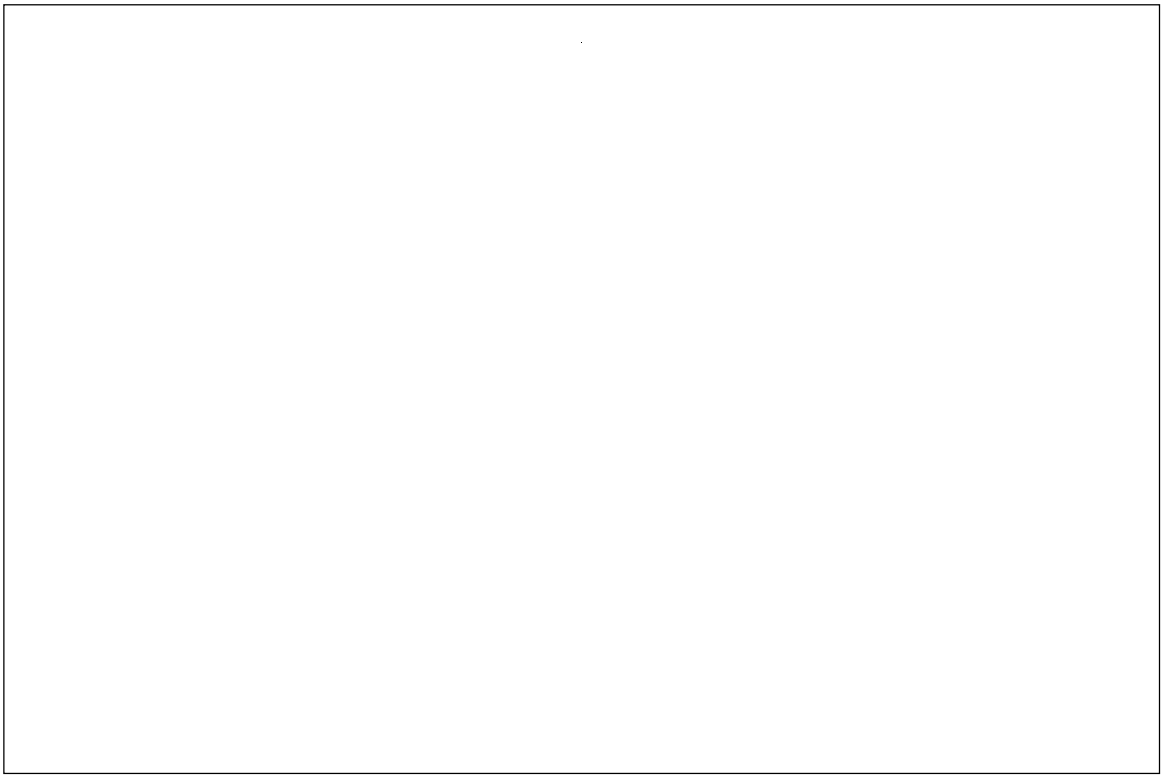
STRUCTURE LOGIQUE D'UNE APPLICATION

Analyse

Même si votre tête est bien faite, sûrement puisque vous avez acheté Bascom ! Il est vivement recommandé de passer par cette étape.

- ⇨ Ecrire les tâches à effectuer.
- ⇨ Décrire la partie Hardware nécessaire : choix du µp, de la mémoire, du clavier, de l'afficheur, du quartz, des communications (SPI, UART, I2C, 1Wire) et de l'alimentation.
- ⇨ Analyser l'application:
  - Quels rôles attribuer aux "petites pattes"? Ce point est capital, des broches sont naturellement dévolues à des fonctions spécifiques. Exemple : RTX et DTX pour l'UART[14] ces broches sont aussi PD0 et PD1 de portD. On évitera donc d'utiliser PD0 et PD1 dans une commande d'afficheur si on désire un transfert série !
  - Dessiner le circuit imprimé en fonction de ce qui précède et essayer de marier au mieux les broches et les fonctions.
- ⇨ Analyser le programme:

  Faire un logigramme. En voici un très académique, l'auteur l'utilise pour des fonctions complexes et sur un tableau d'écolier… à l'ancienne.



Une autre technique d'analyse consiste à répertorier toutes les fonctions que devra effectuer le µP, écrire le programme de ces fonctions, les essayer une à une, puis les rassembler.

[14] Universal Asynchronous Receiver Transmitter, liaison série appelée aussi RS232.

Cette technique n'est pas antinomique de la précédente, mais un complément. C'est aussi un moyen de se créer des bibliothèques de programmes réutilisables.

Premier programme

```
print "Bonjour"  'remarquer les " "
stop     'pour la simulation  le stop est préférable avant le END
end
```

On enregistre et on donne un nom de programme: ne pas utiliser "test.bas" cette appellation prête a confusion mais MonPremierProg.bas par exemple.
Lancer une simulation :

On attend un petit peu et ça fonctionne !
utilisation d'une variable:

```
dim phrase as string *10
phrase="bonjour"
print phrase
stop
end
```

---

pas de différence ! alors les variables ne servent à rien ! Mais si attendez !

```
dim nombre as byte, j as byte
for j=1 to 10      'une boucle inconditionnelle
        print "mon nombre est:" ; nombre          'remarquez l'indentation et le « ; »
        nombre=nombre+1
next
end
```

la variable *nombre* est passée de 0 à 9. On a utilisé une boucle for..next

---

```
dim nombre as byte, j as byte
for j=1 to 10
        print "mon nombre est:" ; nombre
        nombre=nombre+1
        if nombre=6 then  'une instruction conditionnelle
                exit for
        endif
next
print "mon nombre =" ; nombre
end
```

ici on a un exemple de l'utilisation de l'instruction conditionnelle if… then
Le simulateur montre que le programme est hiérarchisé, la valeur 6 est affichée à la sortie de la boucle for--next conditionnée par l'instruction If …then; non pas dans la boucle elle-même!
Vous pouvez jouer avec if..then, if then…elseif…if then …else….endif puis avec do…loop until etc.
Bascom-AVR est très riche, il y a plus de 200 instructions, l'utilisateur les découvrira dans le dictionnaire
Maintenant que nous avons quelques connaissances de bases nous allons les appliquer : il faut découvrir le compilateur Bascom AVR sujet du livre 2, le mettre en route, brancher la carte de développement et revenir ensuite sur l'étude du Basic.

A Suivre …

SIGNES, SYMBOLES ETC…

Dans les exemples (Samples) nous allons retrouvé des signes et symboles peut nombreux mais qu'il faut connaître :

Signe ou Symbole	signification
Espace	Séparateur
‘	Apostrophe pour démarrer une remarque
*	Multiplication
+	Addition
-	Soustraction
,	Virgule (séparateur ex : Locate Y,X)
.	Point décimal ex : 123.45
/	Division universel
\	Division (integer ou Word)
:	Séparateur de commande ex : **Locate** 1,2 **: LCD** " coucou "
;	Sépare les variables à afficher ex : **LCD** j ; k ; "coucou "
"	Entoure les mots à afficher ou à transferer (string)
<	Plus petit que
>	Plus grand que
=	Egal
^	Exposant : s'affiche directement avec l'accent circonflexe
>=	Plus grand ou égal
<=	Plus petit ou égal
#	Directive de compilation (voir l'aide Bascom #if #else …)
&	Précède B ou H pour spécifier la base utilisée (ex :&HC0C0) Utiliser aussi comme suffixe pour signaler des data type long
%	Suffixe pour l'utilisation des data integer
!	Utiliser pour démarrer une ligne assembleur dans le Basic
$	Directive de compilation ex : $crystal = 4000000 Attention $ est quelque fois le préfixe des nombres hexadécimaux (voir registre interne)

LES CONVERSIONS DE VARIABLES

Conversions de variables numériques

Traitement des variables numériques

Nous avons déjà abordé ces questions au chapitre **Qu'est ce qu'une variable ?**
Nous avons vu que la conversion d'un type de variable dans un autre devait respecter la capacité de réception.
En clair : si on dispose de trois bouteilles pour contenir un tonneau on aura quelques problèmes…
Mais on a vu aussi que la programmation nécessitait de jongler avec :
l'hexadécimal (les adresses,…), les octets (les ports, les registres…)
Les valeurs ASCII[15] des caractères du type **string**, on vous l'avait dit que les mots c'était aussi du numérique!
Un petit programme, en annexe Conversions.BAS, reprend les principales instructions de conversions du BASIC et un exemple d'application pour chacune. Le programme fonctionne sans l'aide de la carte de programmation, sur le simulateur.

INSTRUCTION	DESCRIPTION
ABS	Renvoie la valeur absolue d'une variable signée
ASC	Renvoie le code ASCII d'un caractère ou de la première lettre d'un mot. Voir CHR
BCD	Transforme une variable au format BCD en String
BIN	Transforme une variable en représentation binaire (string)
BINVAL	Transforme une représentation binaire en variable
CHR	Renvoie le caractère équivalent au code ASCII. Voir ASC
DECR	Décrémente une variable de 1
FIX	Renvoie une valeur approchée arrondie à 0 plus petite si supérieure à 0, plus grande si inférieure à 0
FRAC	Renvoie la partie décimale d'un nombre, Y=FRAC(1.256), Y=256
HEX	Transforme une variable décimale en représentation hexadécimale (string)
HEXVAL	Transforme une représentation Hexadécimale (string) en variable décimale
HIGH	Récupère le MSB d'une variable integer
HIGHW	Récupère le MSB d'une variable long
INCR	Incrémente une variable de 1
LOW	Récupère le LSB d'une variable integer
MAKEBCD	Transforme une variable décimale en valeur BCD
MAKEDEC	Transforme une variable BCD en variable décimale
MAKEINT	Compacte deux bytes en une Integer ou Long
ROTATE	Fait tourner dans le sens demandé, du nombre de bits demandé.
ROUND	Renvoie une valeur approchée arrondie à 0 plus petite si inférieure à 0, plus grande si supérieure à 0
SHIFT	Fait avancer les bits dans le sens demandé et supprime les bits qui ne sont plus représentés dans l'octet.
SWAP	Echange deux variables de même type
TOGGLE	Fait varier l'état d'un bit.
VAL	Transforme une représentation(string) d'une variable en variable.

[15] ASCII= American standard code for Information Interchange

Conversions de variables alphanumériques

D'une façon identique, il existe un jeu d'outils très pratique pour manipuler les variables STRING.
Mais tout d'abord il est temps d'aborder une notion nouvelle : la chaîne de caractères.
Nous avons évoqué la représentation (string) d'une variable comme étant la transformation d'une variable numérique en variable string. (BIN, HEX)
Vous ne trouverez jamais cette appellation « représentation(string) » dans la littérature, en revanche on vous parlera de chaîne de caractère, cette notion étant un peu absconse, nous avons repoussé l'explication qui suit :

Notion de chaîne de caractères

On appelle chaîne une suite d'objets contigus de même type.

- Un mot est une chaîne de caractères.
- Une suite de mot (une phrase par exemple) est une chaîne de caractères.
- Par extension on accepte la chaîne nulle : "" et la chaîne d'un seul caractère: "a".
- La chaîne de caractères est dimensionnée par le mot **string** qui réserve la place pour autant de caractères que nécessaire plus un. (voir le chapitre **type de variable**)
- Les valeurs de variables sont entre « » ou entre " ":

Mot = « coucou » ou mot ="coucou"

- Chaque caractère est défini par sa valeur ASCII, la table de correspondance est donnée en annexe. Exemple A=65, B=66, C=67…a=97, b=98,c=99.

On peut afficher un caractère par son code, voir l'instruction CHR.
Attention, les chiffres (en tant que caractères) ont aussi une valeur ASCII : 0=48, 1=49…

Traitement des variables « chaînes »

Le programme « chaine_de_caractere.bas » démontre quelques instructions.

INSTRUCTION	DESCRIPTION
CHR(N)	Retourne un caractère ayant la valeur ASCII N
FORMAT	Formatage d'une variable STRING, présentation, signe, virgule.
FUSING	Formatage d'une variable SINGLE avec arrondi
INSTR	Recherche une chaîne incrustée dans une autre chaîne
LCASE	Transforme la casse d'un texte en minuscule.
LEFT	Sélectionne, de x caractères, la partie gauche d'un texte.
LEN	Donne la longueur d'une variable STRING
LTRIM	Supprime les espaces à gauche d'une variable STRING
MID	Sélectionne Y caractères d'une STRING à partir du $X^{ème}$ caractère.
RIGHT	Sélectionne, de x caractères, la partie droite d'un texte.
RTRIM	Supprime les espaces à droite d'une variable STRING
SPACE	Assigne à une variable STRING la valeur de x espaces
SPC	Affiche des espaces (print ou LCD)
STR	Transforme une variable numérique en variable STRING
STRING	Assigne à une variable STRING x fois le même caractère.
TRIM	Supprime les espace en tête et en queue de variable STRING
UCASE	Transforme la casse d'un texte en majuscule.

LES COMMANDES LCD

Bien que beaucoup d'applications réalisées avec un µ-contrôleur se passent d'afficheur, celui-ci est bien souvent indispensable, BASCOM a développé les outils nécessaires à son utilisation soit sur BUS soit sur PORT.
Le simulateur nous permettra de contrôler la plupart des instructions mais un contrôle final sur la carte de programmation sera le bienvenu.
Les instructions de manipulation de chaînes sont étroitement liées aux instructions LCD.

INSTRUCTION	DESCRIPTION
CHR	Pour envoyer le caractères ASCII entre ( )
CIRCLE	Pour afficher un cercle sur un afficheur graphique
CLS	Efface l'afficheur
CONFIG GRAPHLCD	Pour désigner au compilateur, le type de LCD graphique utilisé.
CONFIG LCD	Pour désigner au compilateur, le type de LCD utilisé.
CONFIG LCDBUS	Pour désigner au compilateur, les adresses à utiliser
CONFIG LCDMOD	Pour désigner au compilateur, le mode de travail BUS ou PORT En utilisation PORT on utilise le mode 4 bits
CONFIG LCDPIN	Pour désigner au compilateur, les 6 ports utilisés
CONFIG PORT	Les broches utilisées doivent être en sortie.
CURSOR	Règle l'état du curseur (visible, clignotement)
DEFLCDCHAR	Définitions des caractères utilisateurs
DISPLAY	Active ou désactive le LCD.
FOURTHLINE	Place le curseur sur la 4° ligne (pour les Afficheurs 4lignes)
GLCDCMD	Pour envoyer des commandes à un afficheur graphic (SED)
GLCDDATA	Pour envoyer des data à un afficheur graphic (SED)
HOME	Place le curseur en position 1 sur la ligne spécifiée
HOME LOWER	Place le curseur en position 1 sur la ligne spécifiée
HOME UPPER	Place le curseur en position 1 sur la ligne spécifiée
LCD	Envoie les caractères au LCD.
LCDAT	Pour afficher une constante ou variable sur un afficheur Graphic SED
LCDCONTRAST	Pour changer le contraste sur certains afficheurs
LINE	Pour afficher une ligne sur un afficheur graphique
LOCATE	Place le curseur en position spécifiée ligne/colonne
LOWERLINE	Place le curseur sur la 2° ligne
PSET	Pour afficher un point sur un afficheur graphique
SHIFTCURSOR	Déplace le curseur à droite ou à gauche d'une position
SHIFTLCD	Déplace les caractères à droite ou à gauche d'une position
SHOWPIC	Pour afficher un dessin (BMP) sur un afficheur graphique
SHOWPICE	Pour afficher un dessin (BMP) en EEPROM sur un afficheur graphique
SPC	Imprime des espaces : LCD SPC(x)
THIRDLINE	Place le curseur sur la 3° ligne (pour les Afficheurs 4lignes)
UPPERLINE	Place le curseur sur la 1° ligne

Nous avons réalisé un programme démontrant les principales : afficheur.bas
Les commandes spécifiques aux afficheurs graphiques ne sont pas accessibles par le simulateur. Il y a 3 types d'afficheurs graphiques supportés par BASCOM, voir le dictionnaire à l'instruction Config GRAPHLCD et la police de caractères SED disponible sur le site www.mcselec.com

UTILISATION DES AFFICHEURS

les afficheurs LCD classique sont assez bien normalisés, ils respectent tous les mêmes branchements :

Broche	connexion
1	Gnd
2	VCC(5V)
3	Contraste
4	RS
5	R/W (gnd)
6	E (enable)
7	DBO (Gnd)
8	DB1 (Gnd)
9	DB2 (Gnd)
10	DB3 (Gnd)
11	DB4
12	DB5
13	DB6
14	DB7
15 A	Anode du rétro éclairage (4.2V)
16 K	Gnd

Remarques sur les LCD

- Sous Bascom, par expérience, il est conseillé de travailler en mode 4 bits c'est rapide et cela sauve 4 broches du microcontrôleur.

- La broche contraste est le curseur d'un potentiomètre de 10K à 25KΩ branché entre le VCC et le GND
- Si l'afficheur possède un rétro éclairage, K est à la masse (GND) et A est à VCC par l'intermédiaire d'une résistance en série de 20 ou 30Ω pour limiter le courant à environ 200 mA. Attention à la puissance de la résistance.

- Les afficheurs graphiques sont plus « méchants », ils nécessitent 2 ports complets de microcontrôleur et certains demandent une tension négative pour le contraste. Veuillez vérifier avant de choisir que le contrôleur est bien reconnu par Bascom. Par sécurité on choisira les afficheurs équipés d'un T6963C.

L'exemple d'horloge sur le site montre la facilité d'utilisation des G-LCD avec Bascom.

LA GESTION DES PORTS

Les petites (de plus en plus petites) pattes des µ-contrôleurs obéissent aux ordres donnés par le programme. Il faut être très attentif à ne pas « fusiller » ces petites bêtes, par exemple contrarier un port mis en sortie et à 1 en le court-circuitant : 5V sur 0V !

Faire attention aux ESD[16] potentielles, donc réaliser des environnements sains, avec mise à la masse etc…

Veuillez à ne pas dépasser 20mA par broche et un total de 100mA par port.
Une bonne habitude, si possible, un port sortant sera équipé d'une résistance de 1KΩ, un port entrant d'une résistance de 10KΩ
20mA sous 5volts R=U/I R=5/0.02 R=250Ω… on est tranquille avec 1KΩ

Les ports simples

Nous avons décrit très sommairement le µ-contrôleur ATMEGA8 dans le livre II. Nous allons étudier maintenant les instructions d'asservissement et de lecture des ports.
Les µ-contrôleurs sont en général pourvus de ports multifonctions, la première étape est l'étude des ports en Entrée et Sortie. Voir description des autres fonctions dans le livre 4.

Notion de registres

ATTENTION ce qui suit n'est qu'une explication très simpliste du fonctionnement d'un µ-contrôleur, pour de plus amples informations nous vous conseillons de vous plonger dans l'excellent livre **de Christian Tavernier « Microcontrôleur AVR description et mise en œuvre » Dunod**.

Registres, nous avons déjà cité ce nom dans les généralités. Voici ce qu'il faut savoir :
Le µP fonctionne avec des classeurs appelés registres dans lesquels sont rangés les données (valeur de variables), les data, les constantes, les instructions, le pointeur de mémoire.
Votre programme est rangé en mémoire, un outil demande à l'ALU d'exécuter les instructions de celui-ci, c'est le stack pointer ou pointeur de pile qui va rythmer l'avancement du programme en fonction des instructions, de l'horloge et des interruptions.
Les ports, eux-mêmes, possèdent trois registres :

- Le registre **DDRx** où x est le nom du port, est le registre de sens (entrée ou sortie)
- Le registre **PORTx** est le registre de données soit en lecture soit en écriture suivant le **DDRx**
- Le registre **PINx** est le registre de lecture et seulement en lecture.

La lecture par le registre **PORTx** lit le dernier changement d'état autrement dit le verrou (**latch**).
La lecture **PINx** donne l'état actuel du port.

[16] ElectroStatic Discharge

Au démarrage ou à un **RESET** les ports sont mis en entrée et en haute impédance.
Le simulateur BASCOM permet de visualiser l'état des ports et les modifications des registres.

Attention l'instruction **RESET** n'a rien à voir avec le **RESET HARD** (mise à 0 de la patte RESET)

INSTRUCTION	DEFINITION
Config Portx	Equivalent à DDRx pour l'ensemble du port (entrée= INPUT) ou (sortie=OUTPUT)
Config Pinx.y	Equivalent à DDRx pour une broche du port. (entrée= INPUT) ou (sortie=OUTPUT) x étant le port, Y la broche.
DDRx	Pour configurer un port x. Exemple : DDRB =&B11110000 les broches 7.6.5.4 sont en sortie, les broches 3.2.1.0 sont en entrée.
PINx	Lecture du port x. PORTx peut être en INPUT ou en OUTPUT
RESET PortA.0-7	Si le port B a été configuré en sortie, met la patte concernée (0 à 7) à 0 (0V)
SET PortB.0-7	Si le port B a été configuré en sortie, met la patte concernée (0 à 7) à 1 (5V)
X=PORTB	X prend la valeur des verrous (latches) du port B
PORTB=X	Prend la valeur de la variable X. Port B doit être en OUTPUT

- On doit configurer les ports (ou broche) avec CONFIG PORT, CONFIG PIN ou DDRx avant de les utiliser. (utilisation du **1° registre** des ports)
- Alors **si et seulement si** la broche est en sortie (OUTPUT, DDRx.1), pour positionner une broche dans un état haut ou dans un état bas, on utilisera SET et RESET. Ou l'instruction PortX = n (utilisation du **2° registre** des ports).

Pour positionner l'ensemble d'un port on préférera la fonction PORTx= Y où Y peut être un variable type binaire ou byte. Ex Portb=&B01010101, Portb=85

- Ou lire l'état des verrous d'un port : X=PortB, cette lecture ne donne pas forcément l'état actuel du port, elle donne l'image de sa dernière configuration.
- Ou lire un port en utilisant le **3° registre** des ports : Y=PINx.

Par exemple :

```
DDRB = &B1111_0000 ' le quartet[17] supérieur est en sortie, le quartet inférieur est en entrée
SET PORTB.7 'Positionne le MS bit à +5V le Most Significant bit
RESET PORTB.7 'positionne le MS bit to 0 V
'Pour lire une broche :
Print PINB.0 'lit la broche 0 (LSB) et l'envoi à la RS232
```

Au démarrage

Les ports sont mis en « trois-états », ils peuvent être utilisé en lecture. En revanche ils doivent impérativement être configurés en OUTUT pour être utilisés en sortie.

[17] nibble =quartet.

UTILISATION DE LA MEMOIRE

FLASH

C'est la mémoire de stockage du programme, Atmel autorise 10.000 cycles de programmation. En fin de cycle de vie il peut de y avoir des problèmes d'effacement de celle-ci. Nous conseillons vérifier le contenu de celle-ci par rapport au Buffer.

IRAM

Certains petits microcontrôleurs n'ont pas de SRAM, on utilise alors les premiers registres pour stocker les valeurs.
Attention Bascom n'est pas optimisé pour cela. Voir $tiny. Cela ne concerne pas tous les tiny.

SRAM ou RAM (par défaut)

Chaque variable utilise de la mémoire SRAM, cette mémoire dépend du µP choisi.
Nous utilisons souvent le terme RAM pour la désigner.

32 bytes (adresse 0 à 31) sont utilisés par les registres.

Utilisez des variables bits quand c'est possible. Souvenez-vous que :

- 1 byte occupe 8 bits
- 1 integer/word utilise 2 Bytes
- 1 long/single utilise 4 bytes
- 1 string utilise au moins 2 bytes
- 1 string de 10 caractères occupe 11 bytes

La pile soft (STACK) est utilisée pour stocker l'adresse des variables LOCAL et pour les variables qui sont passées par les SUB. Pour chacune de ces variables 2 bytes sont nécessaires.

La taille de la pile soft peut être calculée en prenant le maximum de paramètres des subs, en ajoutant le nombre de variables LOCAL et en multipliant par 2.
Pour être tranquille ajoutez 4 au total précédent pour les fonctions internes.

Les **valeurs** des LOCAL sont stockées dans un emplacement appelé FRAME,
Par exemple : une string de 40 bytes et 1 long= (41+1)+4=45 bytes de FRAME

Les instructions de conversions comme VAL( ) STR( ) nécessitent de la FRAME (16 bytes)
Les instructions de transfert comme Input, LCD, PRINT idem.

XRAM

Sur les µP de haut niveau il est facile d'augmenter la taille de la RAM en ajoutant une mémoire supplémentaire appelé XRAM
Cette mémoire ne peut stocker des données qu'après la SRAM, la partie basse de la XRAM n'est pas utilisable.

ERAM ou EEPROM

La plupart des µP AVR ont un espace EEPROM qui peut conserver des data, des constantes ou des valeurs qui n’évoluent que lentement, en effet cet espace a une durée de vie limitée à 100.000 cycles donc attention de ne pas "boucler" sur ces valeurs.

CONSTRUCTION D'UN PROGRAMME

Création ou utilisation d'un fichier « .bas »

Le bureau d'étude nous demande de créer un programme (découverte.bas) qui va :

- Lire un bouton,
- Mettre 8 phrases en Data,
- Afficher une phrase correspondante à chaque bouton,
- Allumer une diode correspondante au bouton,
- Faire une pause,
- Faire une subroutine d'allumage des diodes clignotantes à chaque changement d'état des boutons (autant de fois plus 1 que le numéro de bouton)

A chaque étape les outils Basic, et Bascom seront utilisés au mieux.

On lance « Bascom » et suivant le cas, on ouvre un programme existant (File-Open) ou on crée un nouveau programme (File-New).
Bascom permet d'ouvrir plusieurs programmes qui seront affichés, à la demande (voir le signet Window), en cascade ou superposés.
Le programme tourne sous windows >=95 donc, SVP, oubliez les noms de 8 lettres, utilisez des mots explicites pour enregistrer vos programmes (Save as) !

Choix des ports

On se sert de la carte STK200 les 8 diodes sont connectées au port B et à VCC par des résistances de 1.1KΩ tandis que les 8 boutons sont connectés au port D et tirés à VCC par des R10KΩ
Le port A et le port C sont disponibles pour le LCD que nous préférons brancher en externe.
Les afficheurs LCD fonctionnent quasiment tous suivant la même logique :
Choix du mode, 8 bits ou 4 bits - Envoi des commandes - Envoi des caractères.
Le compilateur comporte des instructions qui simplifient ces tâches (voir config LCD)

Choix du µ-contrôleur

Il nous faut 3 ports (2 de 8 bits et 1 de 6) soit 22 broches minimum, nous n'avons pas besoin de ports A/D (Analogique-Digital). Le choix doit être fait entre les
AT90S2313, ATMEGA 8-16PC, ATMEGA8515, ATMEGA32
Le AT90S2313, est trop léger en nombre de ports (15)
LE ATMEGA8 à 23 broches de port I/O, le 8515 en a 28 donc le choix est ouvert. Par soucis de standardisation de nos programmes, et de compatibilité directe avec les exemples Bascom nous avons choisi le ATMEGA8, le ATMEGA32 n'apporte rien de plus.
Le PortC pilotera le LCD

Décomposition du programme decouverte.bas

L'objet du programme

```
' découverte bascom ---------------------------------------------
' Jean-Pierre Duval le 14-11-2002
'modifié le 08-09-2004
'carte de développement STK200
'ATMEGA8 réglé en oscillateur interne 8MHz
'AVCC à VCC
```

Déclaration et Création des sub-routines.

Le quickBasic® nous a donné de bonnes habitudes :
Déclarer les variables et constantes puis les sub-routines et fonctions en tête de programme, Ici aussi c'est conseillé.

```
'directive de compilation----------------------------------------
$regfile = "M8def.dat"       ' on retrouve ces directives dans options/compiler/ chip
$crystal = 8000000       'donc ces déclarations ne sont pas obligatoire
'---------Déclaration des variables--------------------------------
Dim Phrase(8) As String * 10
'déclare un tableau de 8 phrases de 10 caractères
'declare les variables génériques
Dim J As Byte
Dim Y As Byte
Dim A As Byte
'déclare les variables string
Dim Result As String * 8
Dim Endroit As String * 1
'--------Déclaration des sub et fonctions-------------------------
'la déclaration des sub et fonction se fait en début de programme
'la rédaction des sub et fonction se fait après le END
Declare Sub Clignote(byval Diode As Byte)
```

Configuration du système

Il faut configurer le port D en entrée (les boutons), le port B (les diodes), et le port C en sortie (L'afficheur LCD) avec les outils de configuration.
Il faut aussi configurer et câbler l'afficheur LCD tel que nous l'avons proposé.
Les outils de configurations sont une particularité remarquable du Basic BASCOM - voir le dictionnaire.

```
' --------configuration des ports---------------------------------
Config Portd = Input              'le port des boutons sur la carte STK200
Config Portc = Output             'le port que nous utiliserons dans ce programme pour l'afficheur LCD
Config Portb = Output             'le port des LED sur la carte STK200
' configuration du LCD--------------------------
Config Lcdmode = Port       ' config  4 bit
Config Lcdpin = Pin , Db4 = Portc.3 , Db5 = Portc.2 , Db6 = Portc.1 , Db7 = Portc.0 , E = Portc.4 , Rs = Portc.5
'par defaut c'est un afficheur 16 X 2 qui est utilisé dans Bascom donc cette ligne est inutile
Config Lcd = 16 * 2
'---------Chargement du tableau de mots avec les data--------------
Restore Donnees
J = 1
Do
Read Phrase(j)
J = J + 1
Loop Until J = 9
'netoyage écran
Cls
'--------Boucle d'attente d'un bouton-----------------------------
Do
    J = Pind              'Permet de trouver la position de la diode à allumer
    'initialisation des variables
    Result = ""
    'transformation Decimal/string
    Result = Bin(j)
  ' allumage de la diode : copie du port D--------------------------
    Portb = Binval(result)       'transformation string/binaire
  '---envoi des données au LCD-------------------------------------
Result = ""              'initialisation des variables
Y = 9 : A = 8
```

```
   'lecture du Port B
     J = Portb
   'transformation décimal/string
   Result = Bin(j)
   'inversion de la lecture à la recherche du seul bit "0" qui a allumé la diode
  Do
      Endroit = Mid(result , A , 1)
      If Endroit = "0" Then Exit Do       'En simulation on recherche un bit 1
      A = A - 1
  Loop Until A = 0
' petite attente pour visualiser la diode
  Waitms 300
  Y = Y - A
  If Y <> 9 Then
      ' affichage des data
      Cls
      Locate 2 , 1 : Lcd Phrase(y)
      Call Clignote(y)              'lancement de la sub CLIGNOTE
  End If
Loop      'boucle sans fin

End       'end program
```

Ce qui suit doit être mis en fin de programme c'est pourquoi le END doit être taper dès le début car un programme sans End fonctionne !
Ceci concerne Les DATA , les SUB ou FUNTION

```
'-------sub de clignotement-----------------------------------------
'fait clignoter toutes les diodes autant de fois que la variable "diode"
Sub Clignote(diode As Byte)
Local Led As Byte
Led = Diode + 1       'minimum 1
Do
     Portb = 85
     Waitms 200
     Portb = 170
     Waitms 200
     Led = Led - 1
Loop Until Led = 0
Portb = 255
End Sub

Donnees:
Data "Bouton 0" , "Bouton 1" , "Bouton 2" , "Bouton 3" , "Bouton 4" , "Bouton 5" , "Bouton 6" , "Bouton 7"
```

Ce programme se retrouve dans les exemples.

BASIC BASCOM PROGRAMMATION AVANCEE

Le basic BASCOM dispose de nombreux outils permettant de réaliser plus vite des fonctions compliquées. Pour ce faire on dispose de directives de compilation, de librairies, de fonctions de configuration.
A la différence de l'aide BASCOM qui respecte l'ordre alphabétique, nous suivons un ordre d'usage pour expliquer les fonctions avancées.
A ce stade nous ne réalisons pas d'exemples simples, ils existent dans les SAMPLES et c'est au lecteur de faire ses propres expériences.

Les librairies

On trouve dans le menu TOOLS un intitulé LIB MANAGER. En ouvrant cet onglet on découvre un grand nombre de librairies permettant de gérer les composants externes au µP et/ou de réduire la taille du code.
BASCOM édite des nouvelles librairies régulièrement, nous en citerons donc que quelques-unes :

BCD. Lib	Conversion BCD
EUROTIMEDATE.lib	Conversion de la date en jj/mm/YY
FP_trig.lib	Gestion des virgules flottantes
LCD4, LCDbuzy, LCD4E2,	Gestion des LCD
GLCD	Gestion des LCD graphiques
I2C	Gestion des I2C
MCS	Librairie de base, prise par défaut.
MCSBYTE	Permet de réduire le code pour les conversions String-numérique quand, dans l'ensemble du programme, elles ne concernent que des Bytes.
MCSBYTEINT	Idem pour les Bytes, Integer, Word seulement.

Librairie	Utilisation	Option
At_emulator	configure un émulateur de clavier-souris PS2	X
AVR-DOS	permet d'utiliser les commandes DOS pour la gestion de fichiers, accès libre pour une utilisation « hobbie »	X
BCCARD	configure un lecteur de carte	X
BCD	conversion BCD	
Datetime	ajoute des fonctions utiles pour traiter les dates	
Eurotimedate	permet d'utiliser le format jour/mois/année	
Flashcarddrive	configure un lecteur de flash card	
Flashcarddrive-el-pin	configure un lecteur de flash card (elektor interface)	
fp-trig	fonctions mathématiques	
Glcd	pilote des afficheurs graphiques utilisant un T6963C	
Glcdsed	pilote des afficheurs graphiques utilisant un SED	
GlcdKS0108	pilote des afficheurs graphiques utilisant un KS0108	
i2c	pilote des composants I2C	

Librairie	Utilisation	Option
i2cslave	outil de fabrication de composants I2C	X
lcd4	pilote des LCD 4 lignes	
lcd4buzy	pilote des LCD 4 lignes, utilisation de la commande W/R	
lcd4e2	pilote des LCD 4 lignes avc utilisation des 2 Enable	
Mcs	librairie par défaut	
Mcsbyte	librairie à préferer quand on n'utilise que des Bytes	
Mcsbyteint	librairie à preferer quand on n'utilise que des Bytes et des Integers	
Modbus	pilote d'un bus MODBUS	X
Mylib	exemple de librairie personnelle	
Spislave	outil pour les composants qui n'ont pas de SPI	
SQR	pour calculer les racines carrés sans utiliser la lib:Fp-trig	
sqr_it	pour calculer les racines carrés sans utiliser la lib:Fp-trig	
Xramdrive	Pour piloter des Ram externes, voir AVR-DOS	

Utilisation des librairies

Il suffit d'inclure dans le début du programme la directive de compilation :
$lib « nom_de_la_librairie.lbx »

MCS.LBX est cherchée en dernier et est toujours incluse dans la compilation donc il n'est pas nécessaire de la prévoir.

Il est possible de modifier les librairies, par exemple, en rajoutant ou en supprimant des fonctions ou encore en changeant les ports concernés (voir les librairies LCDxx). Il suffit alors de les éditer avec un éditeur Texte comme Notepad

Exemple : Un gros programme qui n'utilise que des variables Bytes ou String aura avantage à spécifier $lib "MCSBYTE.LBX", ce qui permettra de réduire sérieusement la taille de la compilation.

Options

les librairies marquées d'une croix sont vendues séparément.

Les directives de compilations

BASCOM-IDE est, comme nous l'avons signalé au début de cet ouvrage, un Integrated Development Environment : éditeur, simulateur, compilateur, etc..
L'environnement vous permet de signaler au compilateur le type de µP, la mémoire, la vitesse du quartz, etc…
On peut aussi signaler au compilateur qu'il est nécessaire de modifier certaines conditions de travail ou de tenir compte de directives particulières propres à l 'environnement à l'aide des directives de compilations.
Les directives commencent toutes par un $

DIRECTIVES	**FONCTION :** Indique au compilateur qu'il faut
$ASM	Inclure un bloc de codes en assembleur.
$BAUD	Changer la vitesse de transmission.
$BAUD1	Donner la vitesse de transmission pour la 2° voie série
$BGF	Inclure un fichier graphique.
$BOOT	Pour introduire un boot loader (programme de démarrage)
$CRYSTAL	Changer la fréquence du quartz.
$DATA	Démarrer le stockage des Data à la ligne suivant le $data en mémoire. Voir $EEPROM
$DEFAULT	Déterminer l'emplacement de stockage des variables
$DBG	Debugger un transfert UART
$EEPROM	Démarrer le stockage des Data en EEPROM
$EEPLEAVE	Ne pas recréer ou effacer le ficher EEP (voir
$EEPROMHEX	Nécessaire avec une STK500
$EXTERNAL	Inclure une routine ASM (assembleur) dans le programme
$FRAMESIZE	Configure la taille de la « frame » utilisée pour les conversions
$HWSTACK	Configure la pile hardware du µp
$INCLUDE	Inclure un fichier ASCII à la position courante. Un programme répétitif par exemple.
$LCD	Générer un code sur 8 bits pour le LCD
$LCDPUTCTRL	Rediriger les contrôles LCD
$LCDPUTDATA	Rediriger les Data du LCD.
$LCDRS	Générer du code LCD sur 8 bits et sur le bus.
$LCDVFO	Gérére un Enable très court pour certains LCD
$LIB	Inclure une librairie.
$MAP	Générer un label dans le rapport de compilation.
$NOINIT	Créer du code sans initialisation.
$NORAMCLEAR	Ne pas détruire le contenu de la RAM.
$REGFILE	Utiliser le fichier de registre indiqué (chaque type de µP a son propre REGFILE)
$ROMSTART	Utiliser une adresse particulière pour enregistrer le code.
$SERIALINPUT (1)	Rediriger l'INPUT sur une autre entrée que COMM.
$SERIALINPUT(2)LCD	Rediriger les entrées séries vers le LCD pour l'écho.
$SERIALOUTPUT(1)	Rediriger l'OUTPUT sur une autre sortie que COMM.
$SIM	Supprimer les WAIT et WAITMS pendant la simulation.
$SWSTACK	Configure la pile software du µp
$TIMEOUT	Un temps max. pour recevoir des données de l'UART 0 (voir exemple)
$TINY	Ne pas ajuster les espaces RAM réservés.
$WAITSTATE	Augmenter un temps d'attente pour la manipulation des XRAM.
$XRAMSIZE	Spécifier la taille des mémoires extérieures

DIRECTIVES	FONCTION : Indique au compilateur qu'il faut
$XRAMSTART	Spécifier l'adresse de la mémoire externe.

Utilisation les directives

La longue liste précédente n'est pas destinée à effrayer l'utilisateur débutant, en réalité, seule la nécessité, vous conduira à utiliser les directives de compilation.
Les directives de compilation qui correspondent aux options n'ont pas été reprise ici.

Exemples :

$INCLUDE

Pour éviter de surcharger un programme et de rendre la lecture aléatoire, on peut utiliser les routines déjà développées
En voici l'exemple :
Un premier programme , testinclude.bas contient des routines utilisées par le second testincludemain.bas. On notera l'emplacement du $include du second.

- Il faut compiler les deux programmes
- Le programme à inclure n'a pas de End
- Les data sont dans le programme principal ce qui génére une erreur de compilation dans l'include.

```
'----programme inclus  TESTINCLUDE-------------------------------------------------------------------
$regfile = "M32def.dat"       ' the used chip
 $crystal = 11052000
'i2C bus for the XAmega32 module from Sidena
Config Sda = Portb.0
Config Scl = Portb.1
Config Portb.3 = Output
 Led Alias Portb.3

 'config LCD-----------------------------------------------------------------------------
Config Lcdmode = Port       ' config  4 bit
Config Lcdpin = Pin , Db4 = Portc.3 , Db5 = Portc.2 , Db6 = Portc.1 , Db7 = Portc.0 , E = Portc.4 , Rs = Portc.5
'---sub and  functions---------------------------------
 Declare Function Key() As Byte 'keyboard
 Declare Sub Calcul(byval Somme As Integer , Byval Multiple As Integer) 'sub for the fun
' ----variables for the include file --------------------
Dim Jled As Byte
Dim Buz As Byte
Dim Addition As Integer
' --module ok-----------------------------------------
For Jled = 1 To 3       'blinking of the led
    Set Led
    Waitms 200
    Reset Led
    Waitms 200
Next
Reset Led
'------------ sub, gosub  et function-------------------------------------------------
Function Key() As Byte
Local Cle As Byte , Ligne As Byte , Memoire_clavier As Byte
Do
  Encore:
  Ddra = &B00000111       'column in output
   Porta = 7       'send 1 to the column
   Cle = Pina       'read the line
Select Case Cle
    Case 23       'row 1
      Ligne = 1
     Case 39       'row 2
        Ligne = 4
     Case 71       'row 3
```

```
      Ligne = 7
    Case 135      'row 4
      Ligne = 10
    Case Else
      Goto Encore
 End Select
 Waitms 50
 Ddra = &B11110000      'line in output
 Porta = 240      'send 1 to the line
 Cle = Pina
 Memoire_clavier = Cle      'for the bounce
  Select Case Cle
   Case 241      'Column 1
     Key = Ligne + 1
    Case 242      'Column 2
      Key = Ligne + 2
    Case 244      'Column 3
      Key = Ligne + 3
    Case Else
      Goto Encore
 End Select
      Do
        Cle = Pina      ‘if the key is on, loop again
      Loop Until Cle <> Memoire_clavier
Loop Until Key => 1
Key = Key - 1
Cle = Lookup(key , Clavier)
Key = Cle
End Function

Buzzer:
Config Porta.3 = Output
 For Buz = 1 To 8 : Porta.3 = 1 : Waitms 20 : Porta.3 = 0 : Waitms 20 : Next
 Porta.3 = 0
Config Porta.3 = Input
 Return
```

Et maintenant le programme principal

```
'------TESTINCLUDEMAIN----------------------
' Programme principal, il fait appel à un programme inclus dans lequel on peut placer les sub et fonctions ce qui améliore la lisibilité. Dans le programme inclus on peut aussi placer les directives, les ports utilisés les config etc...
'en revanche les déclarations de variables seront placés dans chaque partie.

$include "testinclude.bas"

 '----variables for this program-----------------------
Dim Resultat As Integer , Total As Integer
'-------------------------------------------------------
Cls
' use of a sub --------------------------
Resultat = 3 : Total = 2
Calcul Resultat , Total ‘ call of the sub
Locate 1 , 1 : Lcd "resultat:" ; Addition
‘use of a function----------------------
Do
    Locate 2 , 1 : Lcd Spc(16)      'clear the line
    Jled = Key()
    Locate 2 , 1 : Lcd Jled
    Waitms 300
'    Gosub Buzzer   'if we want
Loop

End      'end program
'-----------------------------------------------------------------------------------------
Clavier:  ' must be after the end
Data 0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 100 , 0 , 200      'first 0 because the data start to 0
```

$CRYSTAL
Permet d'ajuster la fréquence d'un quartz qui n'existe pas dans la liste du menu Option/compiler/chip/communication.

$EEPROM
qui signale au compilateur de stocker les données (DATA) en EEPROM

$LIB
qui est indispensable pour certaines applications comme LCD graphique ou très pratique pour réduire la taille du code (MCSBYTE.lib)
Exemple : $LIB = "MCSBYTE.LBX" (l'extension LBX est la version compilée des librairies)

$REGFILE
un programme peut être développé sur un type de µP, puis pour une raison quelconque, évoluer vers un autre type. Avec cette directive placée en tête de programme, avant tout autre commande on peut effectuer ces changements sans problème. Les fichiers sont tous sous la forme xxxxDEF.dat
exemple : $REGFILE= « m8DEF.dat »

$SIM
A la mise au point des programmes, cette directive permet de supprimer les temps d'attente WAIT et WAITMS de la compilation.

Important :

$HWSTARK, $SWSTARK et $FRAMESIZE

Dimensionner les piles est très difficile, la plupart du temps on y va un peu au hasard, alors qu'il existe une méthode : l'utilisation de la fonction STCHECK. Voir l'aide.
On peut aussi utiliser les recommandations données par les messages quand on clique sur Option/ compiler/ Chip/ framesize, HWStack…

La Framesize :
Quand a t'on besoin d'espace Frame ?

- Quand on fait une conversion Num=>string comme Print B

où B est une variable numérique
Si B est un Byte on utilisera 4 bytes
Integer on utilisera 7 bytes
Long on utilisera 16 byte
Single on utilisera 24 byte

- Quand on ajoute des String comme :

Texte= « ABCD » + texte
On enregistre deux fois la variable Texte
Si Texte est dimensionné à 20, la Frame utilisée sera de 20+1

- Quand on passe une variable byVal on utilise la copie de la variable donc de la frame

un byte passé = 1 byte de frame , un long = 4 bytes.
Quand on utilise des local, par exemple un LOCAL long utilise 4 byte de frame.

Heureusement les espaces Piles et frame sont réutilisables au cours du programme, cela fonctionne un peu comme un indicateur de niveau sonore, en fonction de la demande.
Aussi le plus difficile est de trouver le bon compromis.

Risques des dépassements
Le programme utilise deux fois les mêmes espaces en même temps, il y a fonctionnement aléatoire.

- Il est donc important d'utiliser ces 3 directives de compilations, en effet, une fois que votre programme est débuggé, lire, ces espaces dans les options, puis, entête de programme rajouter les trois lignes :

$HWSTACK= nn
$SWSTACK=xx
$FRAMESIZE=yy

Watchdog

Le watchdog est un timer spécial, présent sur toute la gamme AVR, il est utilisé comme sa traduction l'indique comme gardien interne du programme.

Les instructions concernées :

FONCTION	DESCRIPTION
Config Watchdog	Configuration du timer watchdog
Reset watchdog	Relance le watchdog
Start watchdog	Démarre le timer WD
Stop watchdog	Arrête le timer WD

On placera l'instruction RESET de manière à relancer WD avant le temps donné dans le Config watchdog.

---

la liaison série - UART

Attention : ne pas confondre avec le protocole SPI ou avec le protocole USI
Tout d'abord le tableau des fonctions utilisables par la liaison série :

FONCTION	DESCRIPTION
BAUD	Pour régler la vitesse de transmission
BUFSPACE()	Pour connaître la place disponible dans le tampon(buffer) de sortie
CONFIG SERIALIN	Configuration de l'UART pour utiliser un tampon d'entrée.
CONFIG SERIALOUT	Configuration de l'UART pour utiliser un tampon de sortie.
$CRYSTAL	Pour ajuster finement la fréquence du quartz et éviter les pertes de données.
$DBG	Pour debugger un transfert (ne plus utiliser)
DBG	Pour debugger un transfert (ne plus utiliser)
DTMFOUT	Envoie les codes tonals téléphoniques
ECHO	Renvoie l'écho de la réception.
INKEY	Renvoie la valeur ASCII du 1° caractère du tampon.
INPUT	Renvoie la valeur de la variable associée.
INPUTBIN, INPUTHEX	Lit des variables bin ou Hex sur un port
ISCHARWAITING	Renvoie un 1 tant qu'un caractère est présent dans le tampon même si c'est un CHR(0).
OPEN / CLOSE	Pour ouvrir une x° liaison série software,
PRINT	Pour envoyer des données par la liaison série
PRINTBIN	Equ : Print chr(var)
SERIN	Pour configurer une liaison série soft en entrée
SEROUT	Pour configurer une liaison série soft en sortie
WAITKEY	Attend un caractère provenant de la liaison série.

En annexe vous trouverez le tableau des codes ASCII et une méthode de travail utilisée par l'auteur, pour le formatage de transferts simples.

Le logiciel de réception doit tenir compte de tous les caractères y compris les CR+LF pour son calcul avant de retourner l'ACK.

Les exemples RS232BUFFER.bas et RS232BUFFEROUT.bas sont clairs et fonctionnent parfaitement sur la carte STK200. Attention au cavalier JP3 sur les anciens modèles!

DBG et $DBG permettaient de debugger un transfert en contrôlant l'espace utilisé en Frame, en Hard stack et en Soft stack, ce qui permettait de surveiller les éventuels dépassements, cela n'est plus vrai dans la version 8.2, il y avait interférence quand la liaison était contrôlée par une interruption.

En liaison USB avec un câble il fait mettre 2 stop bit.

TIMERS, Compteurs et PWM

Attention, ce qui suit dépend énormément du µP utilisé, en effet, vous avez 1,2 ou 3 timers-compteurs disponibles appelés T0, T1 et T2

T0 sur 8 bits,
T1 sur 16 bits.
T2 sur 8 bits asynchrone particulièrement bien adapté aux fonctions de dates et heures.

Travailler avec les timers va permettre d'utiliser des nouveaux registres des µP, TCNT0. TCNT1L, TCNT1H etc…

Voici les fonctions qui les pilotent :

FONCTION	DESCRIPTION
CONFIG CLOCK	Pour configurer le timer utilisé par Date$ et Time$
CONFIG TIMER0	Pour configurer la sortie concernée en timer ou compteur
CONFIG TIMER1	Pour configurer la sortie concernée en timer, compteur, driver PWM, comparateur ou réaliser une capture.
CONFIG TIMER2	Pour configurer la sortie concernée en timer, compteur, driver PWM, comparateur ou réaliser une capture.
DATE$	Instruction / variables pour la configuration de la date
TIME$	Instruction / variables pour la configuration de la date
COUNTER0	Instruction de réglage ou variable de récupération des Timers ou compteurs
COUNTER1	Instruction de réglage ou variable de récupération des Timers ou compteurs
PWM	Sous-Instruction de TIMER1,TIMER2
CAPTURE	Sous-Instruction de TIMER1,TIMER2
COMPARE	Sous-Instruction de TIMER1,TIMER2
LOAD	Charge une valeur dans un timer.
START TIMERn	Démarre le timer.
STOP TIMERn	Arrête le timer.

Voir les exemples CAPTURE.bas (mesure de temps), Timer0.bas (compteur d'impulsion), Timer1.bas (utilisation de timer, counter, compare, pwm)
Et les notes d'applications dans www.mcselec.com

Il faut quelquefois utiliser l'assembleur pour « titiler » les interruptions par exemple sur un dépassement (overflow) d'un timer exemple :

```
If Tifr1.2 = 1 Then       '---- si timer echu
  If Showtime = 1 Then    '---- si montrer heure
    Stop Timer1
    Set Tifr1.2           '---- remise à 0 de l'overflow
```

Attention ce qui précède est valable pour un ATMEGA644 on peut jouer sur les 3 overflow des 3 timers sur un atmega 32 on écrira :

```
If Tifr.2 = 1 Then        '---- si timer echu
  If Showtime = 1 Then    '---- si montrer heure
    Stop Timer1
    Set Tifr.2            '---- remise à 0 de l'overflow
```

On notera la différence d'écriture de « tifr »

Timer0

Timer 8 bits. On peut choisir la source Clock de ce Compteur/Timer0 depuis CK(l'horloge interne), le prédiviseur CK ou une broche externe. En plus il peut être arrêté.

Le "flag" de dépassement se trouve dans le registre TIFR (TIMER/COUNTER Interrupt Flag Register)
Les signaux de contrôle se trouvent dans le registre TCCRO (TIMER/COUNTER Control Register)
Les interruptions correspondantes se trouvent dans le registre TIMSK (TIMER/COUNTER Interrupt Mask Register)

Quand Timer/counter0 est piloté par une broche externe, celle-ci est synchronisée avec la fréquence de l'oscillateur du µP. Pour que cela fonctionne correctement, le temps minimum entre deux impulsions d'horloge externe doit être au moins égal à une transition de l'horloge interne du µP.
Le signal d'horloge externe est échantillonné sur le front montant de l'horloge interne du µP.

TIMER1

Timer 16bits. On peut choisir la source Clock de ce Compteur/Timer0 depuis CK(l'horloge interne), le prédiviseur CK ou une broche externe. En plus il peut être arrêté.
Les "flags" de dépassement, de comparaison-égale et de comparaison d'événement ainsi que les signaux de contrôle se trouvent dans les registres TCCR1A et TCCR1B
Les interruptions correspondantes se trouvent dans le registre TIMSK (TIMER/COUNTER Interrupt Mask Register)
Quand Timer/counter1 est piloté par une broche externe, celle-ci est synchronisée avec la fréquence de l'oscillateur du µP. Pour que cela fonctionne correctement, le temps minimum entre deux impulsions d'horloge externe doit être au moins égal à une transition de l'horloge interne du µP.
Le signal d'horloge externe est échantillonné sur le front montant de l'horloge interne du µP.
Ce Timer comporte en plus deux fonctions de comparaisons utilisant les registres OCR1A et OCR1B
Ce Timer peut aussi être utilisé comme générateur d'impulsion programmable sur 8,9 ou 10 bits (fonction PWM)
Il possède, enfin une fonction de capture d'événement, qui peut être le cas échéant piloté par le comparateur analogique.

TIMER2

Timer 8bits. Surtout utilisé pour le contrôle de l'heure, il présente des fonctions proches du Timer1.

Les interruptions

Comme le nom l'indique les interruptions sont des fonctions permettant de suspendre le déroulement du programme à un point particulier ou quand une (des) condition(s) particulière(s) est(sont) réunie(s).

Les interruptions sont soit des entrées de données sur broche, soit des résultats de composants du µP comme les timers.

Les µP ATMEL sont particulièrement riches en interruptions : de 5 (+5 avec les wake-up) pour le modeste TINY11 à 42 ! pour le MEGA128

Voici celles prises en compte par BASIC BASCOM-AVR:

INTERRUPTION	DESCRIPTION
INTO	Interruption externe 0
INT1	Interruption externe 1
OVF0, TIMER0, COUNTER0	Interruption TIMER overflow 0
OVF1, TIMER1, COUNTER1	Interruption TIMER overflow 1
CAPTURE, ICP1	Interruption INPUT CAPTURE TIMER1
COMPARE1A, OC1A	Interruption TIMER1 OUTPUT COMPAREA
COMPARE1B, OC1B	Interruption TIMER1 OUTPUT COMPAREB
SPI	Interruption SPI
URXC	Interruption transmission série RX complète
UDRE	Interruption registre des données série vide
UTXC	Interruption transmission série TX complète
SERIAL	Invalide URXC, UDRE, UTXC
ACI	Interruption Comparateur Analogique
ADC	Interruption convertisseur Analogique/digital
INT2, INT3, etc..	Interruptions dépendantes des µp.

Les instructions concernées :

FONCTION	DESCRIPTION
Config INTx	Configuration des interruptions
Desable	Suspend une/toutes interruption(s)
Enable	Autorise une/toutes interruption(s)
On interrupt	Branchement sur l'interruption
Return	Retour après branchement

Certains composants déclenchent des interruptions (timers, UART, Watchdog)
La lecture des datasheets est conseillée pour la programmation des interruptions.
Les programmes interruptions sur le site et spi-softslave.bas sont des très bons exemples.

(le site http://perso.orange.fr/bils-instruments/ et dans les Samples)

Fonctions mathématiques et Virgules flottantes

Les fonctions mathématiques suivantes sont supportées, elles sont décrites dans le dictionnaire:
Les résultats sont approchés à 8 chiffres significatifs pour les valeurs avant la virgule et à 7 pour les valeurs après la virgule.

FONCTION	DESCRIPTIONS
ACOS	Arc cosinus
ASIN	Arc sinus
ATN	Arc tangente
ATN2	Arc tangente (quelque soit le quadrant du cercle trigo.)
COS	Cosinus
COSH	Cosinus hyperbolique
DEG2RAD	Conversion des valeurs en degrés vers radians
EXP	Exponentielle
LOG	Logarithme naturel
LOG10	Logarithme base 10
POWER	Elévation à la puissance d'un nombre quelconque positif à une puissance quelconque - voir aussi ^
SIN	Sinus
SINH	Sinus hyperbolique
SQR	Racine carrée
RAD2DEG	Conversion des valeurs en radians vers degrés
TAN	Tangente
TANH	Tangente hyperbolique

Comme les valeurs des variables Singles sont approchées, il faut être prudent dans l'utilisation du signe " = "

En effet il est prudent d'arrondir les valeurs calculées avant leur utilisation
Exemple :
Sin d'un angle de 30° (ne pas oublier de convertir en radians)=0.5 sera rendu comme étant 0.499993915 donc l'usage de round( ) est conseillé !

1WIRE et I2C

Nous ne ferons que citer les fonctions à utiliser pour faire fonctionner ceux-ci.
La lecture des datasheets des composants est recommandée.

FONCTION	DESCRIPTION
1WIRE	
CONFIG 1WIRE	Configuration du µP
1wirecount	Lit le nombre de composants 1WIRE reliés au bus.
1wread	Lit les Data du bus 1Wire.
1wreset	Met la broche 1WIRE dans un état de fonctionnement correct en envoyant un RESET au bus.
1wsearchfirst	Attribue le premier ID du bus 1WIRE dans un tableau.
1wsearchnext	Attribue le prochain ID du bus 1WIRE dans le tableau.
1wverify	Vérifie si un ID est disponible sur le bus 1WIRE
1wwrite	Ecrit la valeur d'une variable sur le bus 1Wire.
I2C	
CONFIG I2CDELAY	Configuration l'horloge I2C
Config SCL	Configuration de la broche SCL
Config SDA	Configuration de la broche SDA
I2CRECEIVE	Reçoit des données depuis un composant I2C série.
I2CSEND	Envoie des données à un composant I2C série.
I2CSTART	Génère une condition de démarrage I2C
I2CSTOP	Génère une condition d'arrêt I2C
I2CRBYTE	Reçoit un Byte depuis un composant I2C
I2CWBYTE	Envoie un Byte à un composant I2C

Le protocole SPI

Le protocole SPI(Serial Peripherical Interface) permet de réaliser des transferts à très grande vitesse entre composants ou entre µP.
Il est possible de placer plusieurs µP dans la boucle SPI (voir l'aide en ligne AVR[18] ou l'aide Bascom-Avr)

Les instructions concernées :

FONCTION	DESCRIPTION
CONFIG SPI	Configuration du protocole
SPIINIT	Pour initialiser le bus SPI
SPIIN	Pour lire des données sur le Bus SPI
SPIOUT	Pour envoyer des données sur le Bus SPI
SPIMOVE	Envoie et reçoit une valeur ou une variable du bus SPI

[18] http://www.atmel.com/atmel/products/

LES FICHIERS

Un microcontrôleur n'est pas fait pour gérer des grands fichiers car sa capacité de stockage n'est pas suffisante. 4Kbytes pour les plus performants de la gamme AVR et réinscriptible 100.000 fois seulement donc impossible à utiliser pour des phénomènes très changeants. Josef Franz Vögel a développé un DOS[19] spécifique pour enregistrer ce genre de données en carte Compact-Flash ou sur Eeprom le AVR-DOS. Cet add-on est inclus dans le compilateur Basic mais si vous l'utilisez à des fins professionnels vous devez acheter une licence.

L'auteur utilise une autre méthode utilisant des Serial-EEPROM de la famille 24Cxx :

Soit une mémoire de 8Kbyte (24C64) = 8 X 8bit=64
Attention les mémoires de cette série sont livrées avec &hFF pour chaque valeur d'octet
La première opération à réaliser sera donc un formatage à 0.

Supposons que l'on veuille enregistrer des fiches de 60 octets, et que l'on veuille enregistrer 100 fiches soit 6000 octets
On considère que sur les 8192 octets on en utilisera que 6000 pour la mémoire.

La zone 8010 à 8112 sera utilisée comme marqueur de fiche.

On crée la variable Valdata qui sera le marqueur de mémoire.

En début de programme on lit pas à pas la zone 8010 à 8110 à la recherche du premier 0
Par exemple 8013=0, on a tourné 4 fois donc on écrira la fiche à partir de la mémoire :
60 x 4 = 240
puis on écrira 1 à la valeur 8013

Les fonctions I2C commandant la mémoire

```
'--- configuration pour le bus I2C-------------------------
Portb.6 = 0 : Portb.7 = 0
Ddrb.6 = 0 : Ddrb.7 = 0
Config Sda = Portb.6       ' I2C Data.
Config Scl = Portb.7       ' I2C Clock.

'------Ecriture I2C EEPROM-----------------------------------------------------
Sub I2c_ecrit(adresse_prom As Word , Valeur_en _prom As Byte)
   Msb_adresse = High(adresse_prom)      'le byte haut de l'adresse où écrire
   Lsb_adresse = Low(adresse_prom)       'le byte bas de l'adresse où écrire

   I2cstart      ' Envoi I2C Start.
   I2cwbyte Commande_ecrit       'Adresse du composant I2C+Adresse+1.
   I2cwbyte Msb_adresse       ' Envoi la partie haute de l'adresse
   I2cwbyte Lsb_adresse       ' Envoi la partie basse de l'adresse
   I2cwbyte Valeur_en_prom       ' la valeur du byte à écrire.
   If Err <> 0 Then       ' controle le  status de l'operation.
      Locate 3 , 5 : Lcd "Erreur écriture: Adresse: " ; Adresse_prom
   End If
   I2cstop      '  Fin de la sub d'écriture
   Waitms 10       ' Atente pour la fin de l'écriture
End Sub
```

[19] Disk Operating System = gestionnaire de disque

```
'-----lecture I2C EPROM-----------------------------------------------------
Sub I2c_lit(adresse_en_prom As Word)
   Msb_adresse = High(adresse_prom)       'le byte haut de l'adresse où lire
   Lsb_adresse = Low(adresse_prom)       'le byte bas de l'adresse où lire
   I2cstart       ' envoi un I2C Start..
   I2cwbyte Commande_ecrit       ' Selection du composant en ecriture
   I2cwbyte Msb_adresse       ' Envoi la partie haute de l'adresse
   I2cwbyte Lsb_adresse       ' Envoi la partie basse de l'adresse
   I2cstart       ' envoi un autre Start
   I2cwbyte Commande_lit       ' Selection du composant en lecture
   I2crbyte Valeur_en_prom , Nack       ' lecture...
   If Err <> 0 Then       ' controle le  status de l'operation.
     Locate 3 , 5 : Lcd "Erreur Lecture : " ;       ' Report failure.0
   End If
   I2cstop       ' Fin de la sub de lecture
End Sub
```

Exemple de lecture non fonctionnel, provenant d'un programme volumineux !

```
Lecture:
Adresse_prom = 0       'initialisation
Call I2c_lit(adresse_prom)
'lecture du tableau image-memoire
Adresse_prom = 8010
For Valdata = 1 To 100
 Copymem = Adresse_prom       'pour se souvenir de l'adresse en mémoire de l'image
 Call I2c_lit(adresse_prom)
 If Valeur_en_prom = 1 Then       'mémoire écrite
   Adresse_prom = 60 * Valdata
   Sublecture
   Gosub Affichememoire
 Elseif Valeur_en_prom = 0 Then
   Locate 16 , 31 : Lcd "Mem. Vide"       '+beep
   Valdata = 0
   Exit For
 End If
  Adresse_prom = Copymem + 1       'incrémentation
Next
Return
```

TRAITEMENT DES ERREURS

Pendant la compilation, pour aider le programmeur à trouver ses erreurs de syntaxe ou de structure. Dans le déroulement du programme que ce soit à la suite d'une erreur de code ou d'une erreur hardware ou encore logique.

Un grand nombre d'erreurs peuvent se produire en voici la liste.

ERREUR	DESCRIPTION
1	Instruction inconnue
2	EXIT sans référence à une instruction connue (do, while…)
3	WHILE attendu
4	Plus de place pour IRAM BIT (mémoire RAM intégrée)
5	Plus de place pour BIT
6	. attendu dans le nom de fichier
7	IF THEN attendu
8	Fichier source BASIC non trouvé
9	128 alias autorisés au maximum
10	Type de LCD inconnu
11	INPUT, OUTPUT, 0 ou 1 attendu
12	Paramètre CONFIG inconnu
13	CONST déjà spécifié
14	Seul les Bytes IRAM sont autorisés.
15	Mauvais type de DATA
16	Définition inconnue
17	9 paramètres attendus
18	BIT seulement avec SRAM ou IRAM
19	Longueur de STRING attendu dim mot as string *12 par exemple
20	Byte DATA inconnu
21	Plus de place en IRAM
22	Plus de place en SRAM (Ram statique)
23	Plus de place en XRAM (Ram externe)
24	Plus de place en EPROM
25	Variable déjà déclarée
26	AS attendu
27	Paramètre attendu
28	IF THEN attendu
29	SELECT CASE attendu
30	BITs GLOBAL ne peuvent pas être effacés
31	Type de data non valide
32	Variable non déclarée
33	Variables GLOBAL ne peuvent pas être effacées
34	Nombre de paramètres invalide
35	3 paramètres attendus

ERREUR	DESCRIPTION
36	THEN attendu
37	Opérateur de comparaison invalide
38	Opération impossible sur les BITS
39	FOR attendu
40	Variable ne peut pas être utilisée avec RESET
41	Variable ne peut pas être utilisée avec SET
42	Paramètre numérique attendu
43	Fichier non trouvé
44	2 variables attendues
45	DO attendu
46	Erreur d'affectation
47	UNTIL attendu
50	Valeur ne rentrant pas dans un INTEGER
51	Valeur ne rentrant pas dans un WORD
52	Valeur ne rentrant pas dans un LONG
60	Etiquette (label) dupliquée
61	Etiquette non trouvée
62	SUB ou FUNCTION attendu en premier
63	INTEGER ou LONG attendu pour ABS( )
64	, attendu
65	Composant non ouvert
66	Composant déjà ouvert
68	Canal attendu
70	Vitesse de transmission (BAUD) impossible
71	Les paramètres passés sont différents de ceux déclarés
72	Erreur interne (GETCLASS)
73	- erreur non encore validée -
74	3 paramètres attendus
80	Le code ne rentre pas dans le µP cible
81	Utiliser HEX(var) à la place de PRINTHEX
82	Utiliser HEX(var) à la place de LCDHEX
85	Source d'interruption inconnue
86	Paramètre invalide pour la configuration TIMER
87	ALIAS déjà utilisé
88	0 ou 1 attendu
89	Dépassement de portée, doit être entre 1 et 4
90	Adresse hors limite
91	INPUT, OUTPUT, BINARY ou RANDOM (aléatoire) attendu
92	LEFT(gauche) ou RIGHT(droit) attendu

ERREUR	DESCRIPTION
93	Variable non dimensionnée
94	Trop de bits spécifiés
95	Front montant (rising) ou descendant (falling) attendu
96	Valeur de division doit être 1,8,64,256, 1024
97	SUB et FUNCTION doivent être déclarés
98	SET ou RESET attendu
99	TYPE attendu
100	Pas de tableau pour les variables IRAM
101	Ne trouve pas le registre HARDWARE
102	Erreur dans une routine interne
103	= attendu
104	LoadReg erreur
105	StroreBit erreur
106	Registre inconnu
107	Erreur de chargement de valeur numérique
108	Directive inconnue dans un fichier de composant
109	= attendu dans un fichier "Include" pour .EQU
110	Fichier "Include" non trouvé
111	SUB ou FUNCTION non déclaré
112	Nom de SUB/FUNCTION attendu
113	SUB ou FUNCTION déjà déclaré
114	LOCAL autorisé seulement dans SUB ou FUNCTION
115	Numéro de canal (#Channel) attendu dans gestion de fichier
116	Fichier registre invalide
117	Interruption inconnue
200	.DEF non trouvé
201	Pointeur bas de registre attendu
202	.EQU non trouvé, probablement une fonction non supportée par le µP
203	Erreur dans une instruction LD ou LDD
204	Erreur dans une instruction ST ou STD
205	} attendu
206	Fichier librairie non trouvé
207	Fichier librairie déjà enregistré
210	Définition d'un bit non trouvée
211	Routine externe non trouvée
212	LOW LEVEL, RISING ou FALLING attendu
213	STRING attendu pour l'affectation
214	Taille de la string XRAM à 0
215	Mnémonique ASM inconnue
216	CONST non définie
217	Tableaux non permis avec les bits et booléens
218	Les registres doivent êtres compris entre R16 et R31
219	INTO-INT3 sont toujours ajustées au niveau bas dans les µP MEGA
220	Saut avant trop grand

ERREUR	DESCRIPTION
221	Saut arrière trop grand
222	Caractère non reconnu
223	* attendu
224	Dépassement d'index
225	( ) ne doivent pas être utilisées avec des constantes
226	Constante numérique ou chaîne attendue
227	SRAM commence plus haut que la fin
228	Les lignes de DATA doivent commencer après le END
229	END SUB ou END FUNCTION attendu
230	On ne peut pas écrire dans un registre PINx
231	TO attendu
232	Non supporté par le µP choisi
233	READ fonctionne seulement pour les DATA, pas pour les EPROM data
234	') attendu en dernier
235	'(attendu en premier
236	Cette valeur ne tient pas dans une variable BYTE
238	Cette variable n'est pas dimensionnée comme un tableau
239	Séquence de code invalide à cause d'un bug AVR
240	END FUNCTION attendu
241	END SUB attendu
242	La variable source ne rencontre pas la variable cible
243	Dépassement de capacité pour l'index
244	Ne pas utiliser le pointeur Y
245	Les tableaux ne sont pas supportés avec des variables IRAM
246	Plus de place pour définition d'une librairie .DEF
247	. attendu
248	BYVAL doit être utilisé dans la déclaration
249	ISR déjà défini
250	GOSUB attendu
251	L'étiquette doit être nommée SECTIC
252	Integer ou Word attendu
253	Variable ERAM ne peut pas être utilisée
254	Z ou Z+ attendu
256	Single attendu
257	" " attendu
258	Chaîne de caractère SRAM attendue
259	- non permis pour un BYTE
260	Valeur Chaîne de caractère trop grande
261	Tableau attendu
262	ON ou OFF attendu
263	Dépassement d'index de tableau
264	Utiliser ECHO ON et ECHO OFF en place de…
265	Un offset est attendu en LDD ou STD comme Z+1
266	Timer0 ou Timer1 ou timer2 attendu
267	Constante numérique attendue
268	Les paramètres doivent être rangés de 0-3

ERREUR	DESCRIPTION
269	End select attendu
270	Adresse déjà occupée
322	Les data ne sont pas acceptées avec une instruction
323	Etiquette trop longue
324	Le µP ne supporte pas la librairie I2C slave
325	Les valeurs du diviseur doivent être 1, 8, 32, 128, 256, ou 1024
326	#endif attendu
327	La taille maximum est 255
328	Pas utilisable pour l'UART soft
329	Les champs Dates et heures ne peuvent être que des variables
330	Valeur Maximale pour OUT est &H3F
332	$END ASM attendu
334	') attendu
335	A utiliser avant l'instruction DIM
336	Ne peut régler Clock à cette valeur
999	Les versions DEMO/BETA ne supportent que 4096 lignes de code

LES REGISTRES INTERNES

Attention ce qui suit ne s'adresse pas au débutants, il faut des connaissances en assembleur pour manipuler les registres internes directement depuis le BASIC.
Les registres internes sont des mots réservés.

Les registres et leurs adresses sont définis dans les fichiers .DAT placés dans le répertoire BASCAVR. L'appel d'une fonction se traduit par l'appel à différents registres.

Les noms de registres peuvent être utilisés directement comme variables Basic BASCOM

Ex: DDRA=&B11110000.

Codes ASCII de 0 à 127

Code ASCII	symbole	Hexa	Code ASCII	symbole	Hexa	Code ASCII	symbole	Hexa	Code ASCII	symbole	Hexa
0	NULL	0	33	!	21	66	B	42	99	c	63
1	SOH	1	34	"	22	67	C	43	100	d	64
2	STX	2	35	#	23	68	D	44	101	e	65
3	ETX	3	36	$	24	69	E	45	102	f	66
4	EOT	4	37	%	25	70	F	46	103	g	67
5	ENQ	5	38	&	26	71	G	47	104	h	68
6	ACK	6	39	'	27	72	H	48	105	i	69
7	BEL	7	40	(	28	73	I	49	106	j	6A
8	BS	8	41	)	29	74	J	4A	107	k	6B
9	HT	9	42	*	2A	75	H	4B	108	l	6C
10	LF	A	43	+	2B	76	L	4C	109	m	6D
11	VT	B	44	,	2C	77	M	4D	110	n	6E
12	FF	C	45	-	2D	78	N	4E	111	o	6F
13	CR	D	46	.	2E	79	O	4F	112	p	70
14	SO	E	47	/	2F	80	P	50	113	q	71
15	SI	F	48	0	30	81	Q	51	114	r	72
16	DLE	10	49	1	31	82	R	52	115	s	73
17	DC1	11	50	2	32	83	S	53	116	t	74
18	DC2	12	51	3	33	84	T	54	117	u	75
19	DC3	13	52	4	34	85	U	55	118	v	76
20	DC4	14	53	5	35	86	V	56	119	w	77
21	NAK	15	54	6	36	87	W	57	120	x	78
22	SYN	16	55	7	37	88	X	58	121	y	79
23	ETB	17	56	8	38	89	Y	59	122	z	7A
24	CAN	18	57	9	39	90	Z	5A	123	{	7B
25	EM	19	58	:	3A	91	[	5B	124	|	7C
26	SUB	1A	59	;	3B	92	\	5C	125	}	7D
27	ESC	1B	60	<	3C	93	]	5D	126	~	7E
28	FS	1C	61	=	3D	94	^	5E	127	DEL	7F
29	GS	1D	62	>	3E	95	_	5F			
30	RS	1E	63	?	3F	96	'	60			
31	US	1F	64	@	40	97	a	61			
32		20	65	A	41	98	b	62			

Codes ASCII de 128 à 255

Code ASCII	symbole	Hexa	Code ASCII	symbole	Hexa	Code ASCII	symbole	Hexa	Code ASCII	symbole	Hexa
128	Grand ç	80	161	à	A1	194		C2	227		E3
129	ü	81	162		A2	195		C3	228		E4
130	é	82	163		A3	196		C4	229		E5
131	â	83	164		A4	197		C5	230		E6
132	ä	84	165		A5	198		C6	231		E7
133	à	85	166		A6	199		C7	232		E8
134	å	86	167		A7	200		C8	233		E9
135	ç	87	168		A8	201		C9	234		EA
136	ê	88	169		A9	202		CA	235		EB
137	ë	89	170		AA	203		CB	236		EC
138	è	8A	171		AB	204		CC	237		ED
139	ï	8B	172		AC	205		CD	238		EE
140	î	8C	173		AD	206		CE	239		EF
141	ì	8D	174	<<	AE	207		CF	240		F0
142	Ä	8E	175	>>	AF	208		D0	241		F1
143	Å	8F	176		B0	209		D1	242		F2
144	É	90	177		B1	210		D2	243		F3
145	æ	91	178		B2	211		D3	244		F4
146		92	179		B3	212		D4	245		F5
147	ô	93	180		B4	213		D5	246		F6
148	ö	94	181		B5	214		D6	247		F7
149	ò	95	182		B6	215		D7	248		F8
150	û	96	183		B7	216		D8	249		F9
151	ù	97	184		B8	217		D9	250		FA
152	ÿ	98	185		B9	218		DA	251		FB
153	Ö	99	186		BA	219		DB	252		FC
154	Ü	9A	187		BB	220		DC	253		FD
155		9B	188		BC	221		DD	254		FE
156	£	9C	189		BD	222		DE	255		FF
157	Ø	9D	190		BE	223		DF			
158		9E	191		BF	224		E0			
159		9F	192		C0	225		E1			
160		A0	193		C1	226		E2			

Des caractères "exotiques" compris entre 127 et 255 existent, si nécessaire. Se reporter à une table officielle.

Attention les afficheurs ne respectent pas toujours cette table !

Utilisation des codes ASCII 0 à 31 pour la liaison série

Les caractères officiels (les importants sont en gras)						
Code ASCII						
0	Null	Utilisé par certains systèmes pour l'initialisation				
1	**SOH**	**Début d'entète**				
2	**STX**	**Debut de texte**				
3	**ETX**	**fin de texte**				
4	**EOT**	**Fin transmission**				
5	ENQ	Qui êtes vous ?				
6	**ACK**	**Bien reçu**				
7	BEL	sonnerie				
8	BS	1 lettre en arrière				
9	HT	1 lettre en avant				
10	**LF**	**Line feed (avance d'une ligne)**				
11	VT	Vertical Tabulation				
12	FF	début prochaine page				
13	**CR**	**(Carriage Return) retour chariot -à la ligne**				
14	SO					
15	SI					
16	DLE					
17	DC1	XON				
18	DC2	Contrôle dispo 2				
19	DC3	XOFF				
20	DC4	Contrôle dispo 4				
21	**NAK**	**Mal reçu**				
22	SYN					
23	**ETB**	**Reçu? J'attends**				
24	CAN	Annulé				
25	EM					
26	SUB	Fin de fichier				
27	**ESC**	**Echappement**				
28	FS					
29	GS					
30	RS					
31	RUS					
	Utilisation personnelle de ces codes:					
	Les micro-controleurs ont peu de ressources je me suis donc construit un protocole maison:					
	Print Chr(1)		attention j'envoi			
	Print Chr(2)		Début de texte			
	Print "mon texte 1" (le CR+LF est automatique)					
	Print "mon texte 1 "					
	Print "mon texte 1 "					
	Print Chr(3)		Fin de la partie de texte N1			
	Print Chr(2)		Début de texte			
	Print "mon texte 2 "					
	Print Chr(3)		Fin de la partie de texte N2			
	Print chr(5); x =(nombre de caractères envoyés)					
	Print Chr(4)		J'ai fini			
	Rem je me mets alors en écoute d'un ACK (chr(6)) ou d'un NAK (chr(26))					

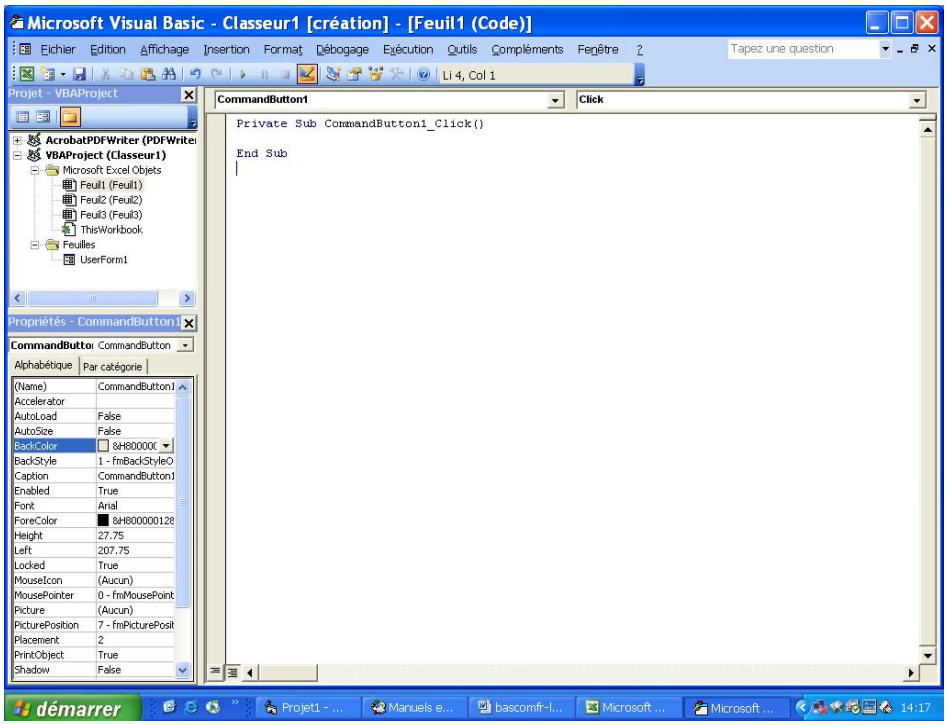
BASCOM ET VBA POUR EXCEL

VBA® veut dire Visual-Basic® pour application (nous mettons des ® partout car même si on fait vendre des produits Windows® Bill est quand même étatsunien, vaut mieux se méfier. A part cela je considère Windows® comme un standard qui a l'avantage de faire dialoguer des milliards de gens avec les mêmes outils, donc merci Mr Gates). Fin des considérations vaseuses.

Avec Visual-Basic pro, il y a un outil de communication qui s'appelle Mscomm32.ocx.
Il se trouve dans c:\windows\system32\mscomm32.ocx donc normalement vous ne pouvez pas développer ce genre d'application si vous ne possédez pas ce fichier sauf si vous téléchargez un exemple non compilé et que par chance dans le .Zip il y ait ce fichier à mettre dans ce fameux répertoire.

C'est ce qu'on va trouver en suivant ce lien http:/cherbe.free.fr/ qui est tout simplement formidable de simplicité et de facilité. Télécharger les exemples pesées.xls.
Ou encore en téléchargeant notre exemple de communication que nous vous proposons sur le site. (Nous utilisons VB5 pro, Mscomm32.ocx est livré avec, nous espérons que les exemples fournis fonctionnent sans obligation d'acheter un langage aussi cher.)

Dans Excel vous devrez activer la barre d'outil Visual Basic (onglet affichage)
Elle est composé de 3 icones : un écritoire, une boite à outils, une règle et une équerre.
- Cliquer sur la boite à outil
- prendre un bouton de commande, il se dessine avec le bouton gauche de la souris et flotte sur la grille Excel, il s'appelle commandBouton1 si vous cliquez dessus un écran VB s'affiche



Sur la gauche en haut, la fenêtre de projet, dessous la fenêtre de propriété, à droite la fenêtre de code qui affiche la première sub.

ATTENTION, ATTENTION, ATTENTION, ATTENTION, ATTENTION

VB autorise la déclaration implicite de variable ce qui est aberrant, dangereux, stupide !
Aussi dès l'ouverture du programme il faut écrire :

```
Option Explicit ' écrire cette phrase avant la première sub, elle veut dire que les variables doivent être déclarés pour ce module
--------------------------------
Private Sub CommandButton1_Click()

End Sub
```

Puis il faut demandé à VBA d'ajouter un userform (dans la page d'écriture du code) cliquer sur l'onglet insertion puis userform.
Revenir sur la grille Excel, recliquer sur le bouton, on revient à l'éditeur ajouter :

```
Option Explicit
Private Sub CommandButton1_Click()
UserForm1.Show
End Sub
```

Dans les propriétés du bouton, modifiez le Caption (texte du bouton) : « lancer le transfert »

Cliquer sur userform1 dans la fenêtre de projet.
Ouvrir la boite à outils 🛠 si le ☎ n'est pas présent cliquer dans la boite à outils dans l'onglet outils cliquet sur contrôles supplémentaires et rechercher « Microsoft® communication control, version… » cocher la case, le téléphone doit s'afficher.

L'exemple

Transférer des données Excel à un microcontrôleur, lire ces données, renvoyer un message à Excel

Comme toujours notre démarche est de montrer quelque chose qui fonctionne avec les outils Bascom soit pour cet exemple:
- Une carte XAmega32 de Sidena (une STK200 ou autres ferait l'affaire, il lui suffit d'un port série)
- D'un ATmega 32
- d'un afficheur 2x16

Le but du programme :
- Transférer le contenu d'une cellule vers le Atmega 32. La cellule transmise est A3 seulement soit colonne A ligne 3 donc il faut qu'il y ait quelque chose en A3 .

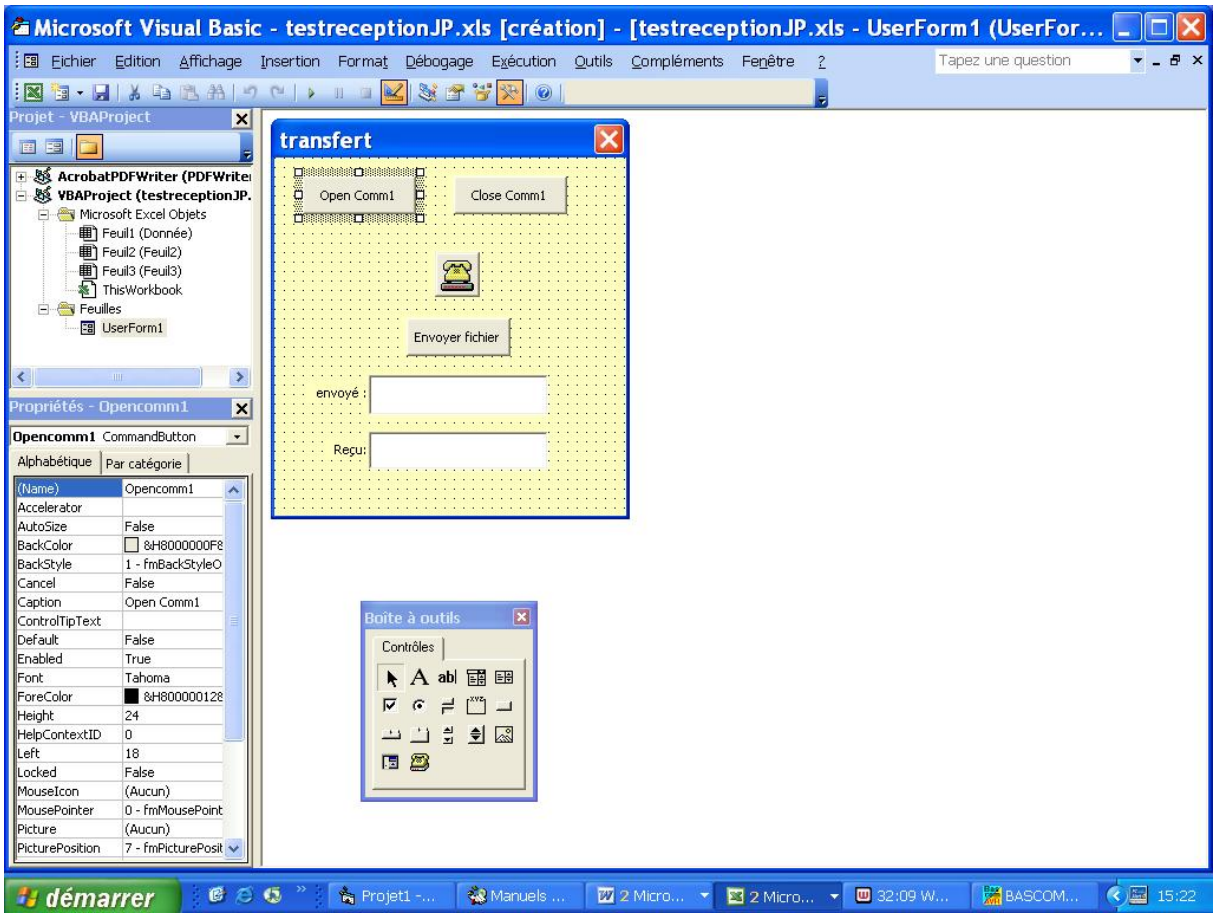
- Affichage de cette cellule sur l'afficheur

- Retour d'un message vers Excel

- Affichage de ce message dans une cellule de réception B3.

Nous considérerons que la cellule envoyée peut contenir autant de caractères que le buffer peut en recevoir soit 32.

Nous retournons un mot de 20 caractères, s'il fait moins de 20, il est complété par des espaces chr(32).



L'userForm contient
3 boutons : ouvrir Comm1 ; Fermer Comm1 ; Envoyer
2 Fenêtres : Envoyé ; Reçu
1 Téléphone
Dans les propriétés de Userform on pourra changer la couleur, la taille, le titre…

La propriété name des boutons Open Comm1 et Close Comm1 sera changée respectivement en Opencomm1 et Closecomm1

L'objet communication ☎ qui doit être visible dans la boite à outils possède un certain nombre de propriétés :

CommPort le numéro du port série du PC en général=1

DTREnable Data Transmission Ready Enable doit être « true » pour autoriser la communication.

EOFEnable End Of File Enable nous n'utilisons pas pour nos applications donc « false »

Handshaking définit le protocole de communication utilisé, 0=pas de protocole

InBufferSize Taille du buffer d'entrée 1024 par défaut

InputLen Définit et renvoie le nombre de caractères lus dans le tampon de réception par la propriété **Input**. Si=0 lit la totalité du Buffer

Inputmode Définit ou renvoie le type de données extrait par la propriété **Input**. En mode texte

NullDiscard Détermine si les caractères nuls sont transférés du port vers le tampon de réception.

OutBufferSize Définit la taille du tampon de transmission en octets.

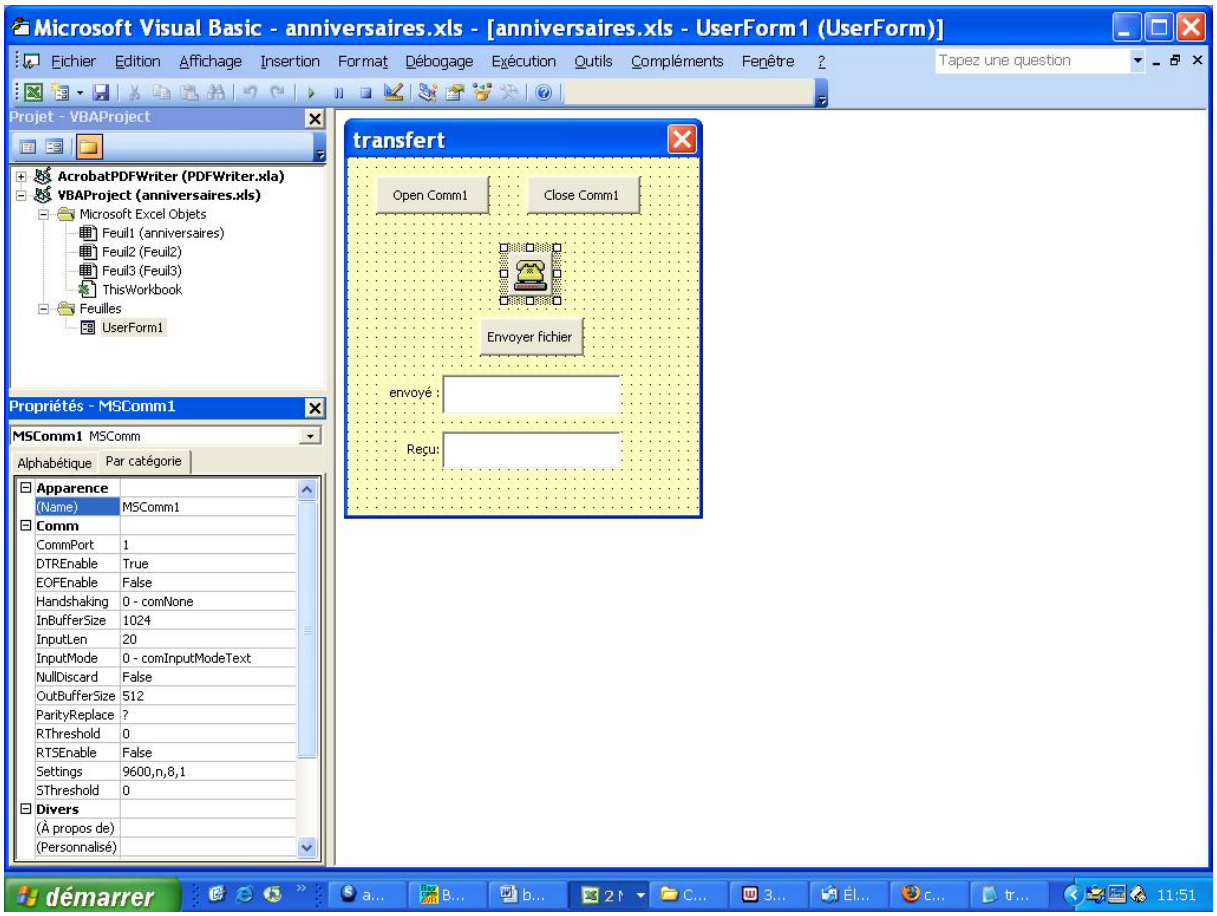
ParityReplace Définit et renvoie le caractère qui remplace un caractère non valide dans le flux de données en cas d'erreur de parité.

Rthreshold pas utilisé doit être laissé à 0

RTSEnable Demande pour émettre « false » ici

Settings Réglage du port série par défaut 9600 baud, Pas de Parité, 1 bit de stop, mot de 8 bits

Sthreshold utilisé doit être laissé à 0



Code VBA :

```
Option Explicit
---------------------------------------------------------
Private Sub closecomm1_Click()
' ferme le port s'il est ouvert------------
If MSComm1.PortOpen = True Then MSComm1.PortOpen = False
  'ferme le userform en fin de réception
  UserForm1.Hide
End Sub
---------------------------------------------------------

Private Sub Opencomm1_Click()
'vider le buffer---------------------------
MSComm1.InBufferCount = 0
'ouvrir la liaison série-------------------
MSComm1.Settings = "9600,n,8,1"
' on attend 20 caractères------------------
MSComm1.InputLen = 20
' ouverture du port série------------------
If MSComm1.PortOpen = False Then MSComm1.PortOpen = True
End Sub

---------------------------------------------------------

Private Sub CommandButton1_Click()
Dim ligne As Byte, colonne As Byte
Dim received As String * 20
Dim send As String * 20
If MSComm1.PortOpen = True Then
     ' envoie des données------------------
   ligne = 3
   colonne = 1
    send = Worksheets("Données").Cells(ligne, colonne)
    MSComm1.Output = send
    TextBox1 = send
    ' fin de transfert--------------------
    MSComm1.Output = Chr(3)
    ' reception des données----------------

    CommandButton1.Caption = "reception"
   ligne = 3
   colonne = 2
    'boucle d'attente de reception--------
    Do
       DoEvents
    Loop Until MSComm1.InBufferCount >= 20
    received = MSComm1.Input
    TextBox2 = received
    'suppression des char(32) et affichage-----
   Worksheets("données").Cells(ligne, colonne) = Trim(received)
 Else
    ' message si le port n'est pas ouvert---
    MsgBox ("attention, le port série n'est pas ouvert")
 End If
 ' mise à 0 du buffer-----------------------
 MSComm1.InBufferCount = 0
CommandButton1.Caption = "bien reçu"

End Sub
```

Code BASCOM

```
' ------Jean-Pierre Duval 26-01-2006----------------
'           test la liaison série avec Excel
'     utilisation d'un module Xamega32 SIDENA
'-------------------------------------------------------------
$regfile = "M32def.dat"       ' the used chip
 $crystal = 11052000
 '----[LCD configuration---- REMarks if not used]-----
Config Lcdmode = Port       ' config  4 bit
Config Lcdpin = Pin , Db4 = Portc.3 , Db5 = Portc.2 , Db6 = Portc.1 , Db7 = Portc.0 , E = Portc.4 , Rs = Portc.5
Config Lcd = 16 * 2
'i2C bus for the XAmega32 module from Sidena
Config Sda = Portb.0
Config Scl = Portb.1
Config Portb.3 = Output
Led Alias Portb.3
'--RS232 configuration -------------------------------
$baud = 9600
Config Serialin = Buffered , Size = 32
Config Serialout = Buffered , Size = 32
Enable Interrupts
'---sub and  functions---------------------------------
 Declare Sub Rsread()       'read the data from the excel file
 Declare Sub Rswrite()      'write the data from the excel file
' ----variables ---------------------------------------
Dim Jbyte As Byte
Dim Bytereceived(30) As Byte
Dim Send As String * 20
' --chek of the module ok----------------------------
For Jbyte = 1 To 3       'blinking of the led
    Set Led
    Waitms 200
    Reset Led
    Waitms 200
Next
Reset Led
'-------------------------------------------------------------
Main:
Cls
Locate 2 , 1 : Lcd " wait"       'pour lancer Excel Etc....
Wait 5
Locate 2 , 1 : Lcd "lecture"
Call Rsread

Locate 2 , 1 : Lcd "transfert:"
    Call Rswrite
   Waitms 500       ' trés important de ne pas terminer immédiatement

End

'------rswrite----RS232-------------------------
Sub Rswrite()
Local Kbyte As Byte , Lbyte As Byte

Send = "ABCDEFGHIJK"
Kbyte = Len(send)
Kbyte = 20 - Kbyte
For Lbyte = 1 To Kbyte
    Send = Send + " "
Next
Kbyte = Len(send)
 Locate 2 , 1 : Lcd Send
Print Send       ' envoi par RS232
End Sub
```

```
'------rsread----RS232------------------------
Sub Rsread()
Local Rs_index As Byte
Jbyte = 0
Rs_index = 1       ' les tableaux commencent à 1 en Bascom
Locate 1 , 1 : Lcd ""
Do
   If Ischarwaiting() = 1 Then           ' si 1 char est dans la file d'attente
      Jbyte = Inkey()                    'reçoit la valeur de la liaison série
          If Jbyte = 3 Then              'dernier caractère envoyé par Excel
           Exit Do
          End If
        Lcd Chr(jbyte);
          Bytereceived(rs_index) = Jbyte
          Rs_index = Rs_index + 1
   End If
 Loop
End Sub
```

PROGRAMMES EXEMPLES

Conversions numériques

```
'conversions-------------------------------------------------------
'programme de conversion numérique avec  les outils BASCOM
'par ordre alphabétique
'..ABS..ASC..BCD..MakeBCD..MakeDEC..MAKEINT...BIN..BINVAL..CHR....HEX..
'..HEXVAL..HIGH...HIGHW...LOW...ROTATE...SHIFT...SWAP...TOGGLE..VAL
'pour visualiser une conversion, supprimer le '(REM) avant le WAIT
'Ce programme n'utilise pas la carte de programmation.
'JPDuval le 2/10 novembre 2002
'-----------------------------------------------------------------
Dim Nombre As Integer
Dim Secondnombre As Integer
Dim Nombreword As Word
Dim Nombrelong As Long
Dim Nouveausingle As Single
Dim Nombrebyte As Byte
Dim Seconbyte As Byte
Dim Mot As String * 8
'------ ABS ( ) valeur absolue d'une variable négative-------------
Nombre = -25852
Secondnombre = Abs(nombre)
Print "ABS d'integer: " ; Secondnombre
'Wait 1

' ---ce qui suit ne fonctionne pas---------------------------------
'Nombrelong = Nombre
'Nouveausingle = Abs(nombrelong)
'Print "ABS de long : " ; Nouveausingle
'Wait 1

'ce qui suit fonctionne-------------------------------------------
Nombrelong = Nombre
Nombrelong = Abs(nombrelong)
Print "ABS de long : " ; Nombrelong
'Wait 1

' -----ASC--------valeur ASCII d'une lettre ou de la première lettre d'un
mot--
 Mot = "Bonjour"
Nombrebyte = Asc(mot)
 Print "ASII : " ; Nombrebyte
'Wait 1

'-----BCD------MakeBCD----MakeDEC---------------------------------
For Nombre = 1 To 20       'amusez vous à faire varier ce nombre
Secondnombre = Makebcd(nombre) 'contrôlez les variables avec le simulateur.
Print Secondnombre ; "      " ; Bcd(secondnombre)
'MakeDEC converti une variable BCD en  variable décimale (voir
dictionnaire)
'Waitms 50
Next
'Wait 1
```

```
'-----MAKEINT-------------------------------------------------------
Nombre = 2
Secondnombre = Makeint(nombre , 1)
Print "secondnombre :" ; Secondnombre       'dites pourquoi 258 ?
'Wait 1
'-----BIN-----BINVAL------------------------------------------------
Nombre = 85
Mot = Bin(nombre)
Secondnombre = Binval(mot)
Print Mot ; "       " ; Secondnombre
'Wait 1
'-----CHR-----------------------------------------------------------
Nombre = 85
Print Nombre ; "  Chr(nombre): " ; Chr(nombre)
'Wait 1
'-----HEX-----HEXVAL------------------------------------------------
 Nombrelong = 43690
 Mot = Hex(nombrelong)
 Nombrelong = Hexval(mot)
 Print "MOT hexa: " ; Mot ; "    nombrelong: " ; Nombrelong
' Wait 1
'----HIGH----LOW----------------------------------------------------
Nombre = 515
Secondnombre = High(nombre)
Nombrebyte = Low(nombre)
Print "High de nombre: " ; Secondnombre       'la réponse est 1, 256=&h100
Print "low de nombre: " ; Nombrebyte       'pourquoi 3 ? cherchez un peu ?
'Wait 1
'----HIGHW----------------------------------------------------------
Nombrelong = &H12345678
Nombreword = High(nombrelong)
Print "highw de nombrelong: " ; Hex(nombreword)
'Wait 1
'----ROTATE------SHIFT----------------------------------------------
Nombrebyte = 186
Print Nombrebyte ; "=" ; Bin(nombrebyte)
Rotate Nombrebyte , Left , 2
Print "rotate: " ; Bin(nombrebyte) ; "     " ; Nombrebyte
Shift Nombrebyte , Left , 2
Print "shift: " ; Bin(nombrebyte) ; "     " ; Nombrebyte
'Wait 1
'-----SWAP-----------TOGGLE-----------------------------------------
Nombre = 185
Secondnombre = 581
Print "nb 1     nb2 :" ; Nombre ; "    " ; Secondnombre
Swap Nombre , Secondnombre
Print "nb 1     nb2 :" ; Nombre ; "    " ; Secondnombre
'TOGGLE change d'état une variable BIT, cette commande est aussi utilisée
'dans les commandes de ports, voir Dictionnaire.
'Wait 1
'-----VAL-----------------------------------------------------------
Mot = "2857"
Nombre = Val(mot)
Nombre = Nombre * 2
Print "Mot X 2 " ; Nombre
Wait 1
End
```

Chaîne de caractères

```
'---chaine de caractères-------------------------------------
'Les outils de conversions et de traitement des chaines
'Par ordre alphabétique et repris dans le dictionnaire.
' pour contrôler une instructions, supprimer le REM(') avant le WAIT
' et lancer le simulateur. ce programme n'utilise pas la carte de
'programmation.
'...format...fusing...INSTR..Lcase...LEN...Ucase...LEFT...RIGHT...MID
'LTRIM...RTRIM...TRIM...SPACE...SPC...STR...STRING
'Jp Duval Le 13 -11 -2002
'------------------------------------------------------------
Dim Mot As String * 20
Dim Secondmot As String * 20
Dim Nombre As Integer
Dim Nombresingle As Single

'----FORMAT----------------------------------------------------
'formatage d'une variable string numérique
Nombre = 1234
Mot = "1234"
Mot = Format(mot , "+")
Print "mot: " ; Mot
Mot = Format(mot , "  00.00")
Print "mot: " ; Mot
'Wait 1

'----FUSING----------------------------------------------------
'formatage d'une valeur SINGLE avec #
Nombresingle = 123.45678
'conversion du nombre en chaine
Mot = Str(nombresingle)
Print "mot entier: " ; Mot       'print 123.456779477
Mot = Fusing(nombresingle , "#.##")
Print "mot arrondi: " ; Mot
Mot = Fusing(nombresingle , "#.####")
Print "mot arrondi: " ; Mot
' Wait 1

'---INSTR------------------------------------------------------
'initialise la phrase 1
Mot = "abcdeabcd"
'initialise la phrase à trouver
Secondmot = "ab"
'returne la 1° position dans mot
Nombre = Instr(mot , Secondmot)
Print Nombre
'Wait 1
'Maintenant recherche d'une autre localisation (ab se trouve 2 fois)
Nombre = Instr(2 , Mot , Secondmot)
Print Nombre
'Wait 1
'doit retourner 6
Nombre = Instr(mot , "xx")
'xx  n'est pas dans "mot" donc : retourne 0
Print Nombre
'Wait 1
```

```
'---LCASE---UCASE-----------------------------------------------
Mot = "coucou"
Mot = Ucase(mot) : Print Mot
Mot = Lcase(mot) : Print Mot
'Wait 1
'---LEFT--RIGHT-MID----------------------------------------------
Mot = "bonjour ou coucou"
Secondmot = Left(mot , 7)
Print Secondmot;
Secondmot = Right(mot , 6)
Print Secondmot
'Waitms 300
For Nombre = 1 To 17
Mot = "bonjour ou coucou"
Secondmot = Mid(mot , Nombre , 1)
    Print Secondmot;
Waitms 50
Next Nombre
'Wait 1
Print ""

'---LEN----------------------------------------------------------
Mot = "bonjour ou coucou"
Nombre = Len(mot)
Print "longueur de mot: " ; Nombre
'Wait 1
'---LTRIM---RTRIM---TRIM-----------------------------------------
Mot = "    bonjour    "
Print "Ltrim" ; Ltrim(mot);
Print "fin"
Print "Rtrim" ; Rtrim(mot);
Print "fin"
Print "trim" ; Trim(mot);
Print "fin"
'Wait 1

'---SPACE--SPC---------------------------------------------------
Nombre = 15
Mot = Space(nombre)       'assigne à "mot" la valeur de 15 espaces
Print Mot ; "fin"
Print Spc(nombre) ; "Fin"       ' ne fait qu'afficher des espaces,
'peut être utilisé dans une commande LCD, ne consomme pas d'espace mémoire.
'Wait 1

'--STR-----------------------------------------------------------
Nombre = 15527       'jouons un peu avec les nombres
Mot = Str(nombre)
Mot = Mot + " coucou"       'mélangeons numérique et caractères
Print Mot
'Wait 1
'---STRING-------------------------------------------------------
Mot = String(15 , 64)
Print Mot
Wait 1
End
```

Afficheurs

```
'----------afficheur-------'jp duval le 13-11-2002------
'Les outils de manipulation des LCD
'Par ordre alphabétique et repris dans le dictionnaire.
'-----------------------------------------------------------
'les variables
Dim Y As Byte
' --------configuration du port LCD---------------------------------
Config Portd = Output
'---------configuration LCD---------------------------------------
Config Lcdmode = Port    ' config le LCD en 4 bit
Config Lcdpin = Pin , Db4 = Portd.3 , Db5 = Portd.2 , Db6 = Portd.1 , Db7 =
Portd.0 , E = Portd.5 , Rs = Portd.6
' cette ligne change le réglage du compilateur. A écrire sur 1 ligne.
Config Lcd = 16 * 2
' cette ligne n'est pas obligatoire quand c'est un 16*2
Cls  '---démarrer par un effacement---------------------------
Home Upper
Lcd "clignote"
Cursor On
Cursor Blink       'clignote
Wait 3
Cls
Cursor Noblink
Lcd "ne clignote plus"
Waitms 300
Cls
Home
Cursor On Blink
Lcd "shift est facile"
For Y = 1 To 5
   Shiftcursor Right , 1
    Wait 1
Next
Lcd "shift est facile"
For Y = 1 To 5
    Shiftcursor Left , 1
    Wait 1
Next
Cls
Home Upper
Lcd "shift est facile"
For Y = 1 To 15
    Shiftlcd Right , 1
    Waitms 100
Next
For Y = 1 To 15
    Shiftlcd Left , 1
    Waitms 100
Next
Cls : Locate 2 , 8
Deflcdchar 1 , 31 , 16 , 20 , 21 , 26 , 20 , 16 , 30       ' replace ? avec
un nombre (0-7)
Cls
Lcd Chr(1)
```

TRUCS ET ASTUCES

Les détails

On oublie toujours de rechanger la vitesse du µP entre la carte de développement et l'application, donc tant que l'application n'est pas fonctionnelle, j'utilise la même fréquence de quartz.

Les Fusibles

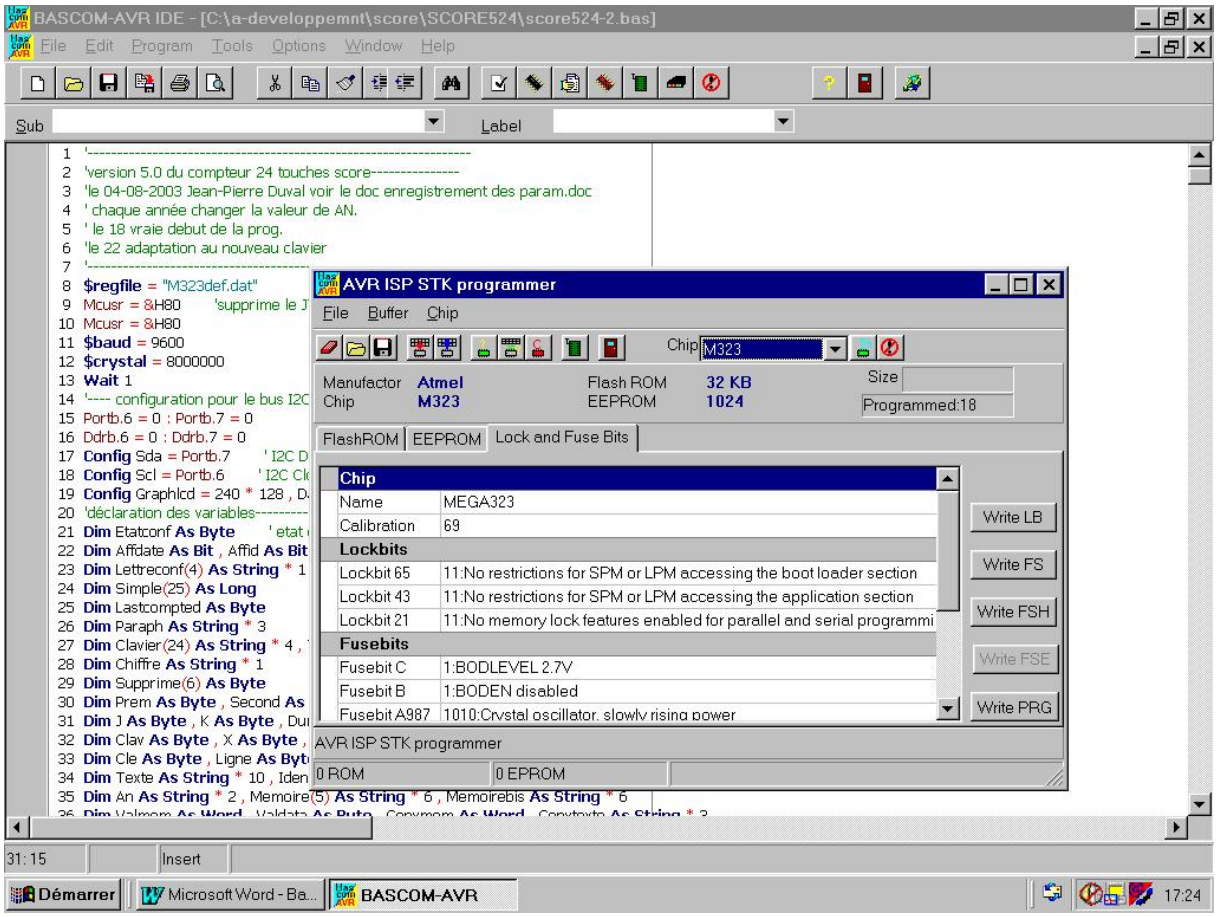
Un chapitre complet est consacré aux fusibles dans le livre 4, le mot Fusible ne doit pas affoler, puisque la plupart de ceux-ci sont réarmable.

Sur les circuits équipés d'un JTAG[20], il est conseillé de désactiver le JTAG pour utiliser complètement le port C en I/O. Le programme de transfert fournit par KANDA permet de le désactiver par fusible, De même que le BASCOM. En revanche il est toujours possible de lancer la routine assembleur donner dans l'exemple M323.BAS :

```
'-----------------------------------------------------------
'                M323.BAS
'-----------------------------------------------------------
$regfile = "M323def.dat"
'This file is intended to test the M323
'The M323 has the JTAG enabled by default so you can not use
'pins PORTC.2-PORTC.5
'Use the following code to disable JTAG
Mcusr = &H80
Mcusr = &H80                    Ce qui fonctionne très bien.
```

[20] J.T.A.G. : Join Test Action Group ou BST Boundary_Scan_Test Méthode de contrôle in situ d'un composant.

Vitesse d'horloge interne des circuits MEGA



Les circuits Mega peuvent être utilisés sans quartz. L'horloge interne est d'une bonne précision sans avoir celle d'un quartz. Ce qui peut être gênant pour une liaison série. Pour remédier à ceci on règle par soft la valeur de calibrage donné dans l'écran fuse: Ici 69 (chip calibration) celle-ci peut-être approximative.

Ce petit programme permet de bien régler cette dernière:

```
'-----------------------------------------------------------
'            mega8-32.bas
'    mega8 sample file   reglage du coeff de correction
'-----------------------------------------------------------
$regfile = "m8def.dat"
Osccal = &HA0
$baud = 9600
Dim J As Byte , X As Byte , S As String * 10 , Y As Byte
'---------configuration LCD---------------------------------
Config Lcdmode = Port
' config le LCD en 4 bit
Config Lcdpin = Pin , Db4 = Portc.0 , Db5 = Portc.1 , Db6 = Portc.2 , Db7 = Portc.3 , E = Portc.4 , Rs = Portc.5
' cette ligne change le réglage du compilateur.
Config Lcd = 20 * 2
'config uart---------------------------------------------------
Config Com1 = Y , Synchrone = 0 , Parity = None , Stopbits = 2 , Databits = 8 , Clockpol = 1
Config Serialin = Buffered , Size = 20
Config Serialout = Buffered , Size = 20
 Wait 5
 For J = 1 To 20
 Osccal = Osccal + 1
 S = Hex(osccal)
 Print "OSCCAL " ; S
```

```
Lcd "OSCCAL " ; S
Wait 1
Cls
Next
End
```

On donne au début la valeur OSCCAL-10 (valeur de calibrage-10) et on lit sur le terminal emulator les valeurs envoyées par print, il se peut que ce soit des valeurs bizarres au début, mais petit à petit les données seront correctes, on prendra une valeur moyenne.
Ceci est à faire pour chaque µP.
Ici le programme tourne 20 fois, si la réponse est toujours incorrecte, augmenter le nombre de tour.

Securité contre la copie ou la malveillance

Un fois le programme terminé, s'il s'agit d'un programme personnel, il est toujours bon de se protéger contre les petits malins copieurs et s'il s'agit d'un programme professionnel contre le piratage d'invention.
Les lockbits sont fait pour cela.
Il ne bloque pas définitivement le µP, ils peuvent étre remis à 0(1 dans la logique ATMEL !) par un effacement général.
Ils sont au nombre de 3 et chacun avec 2 bits

21

= 11 Pas de bloquage
= 10 programmation future interdite
= 00 programmation future interdite, lecture interdite

43 et 65

Ces fusibles sont critiques pour l'utilisation des interruptions, suivant que ces dernières sont placées ou non dans le Boot-loader (partie de la mémoire écrite en assembleur qui permet de lancer plusieurs programmes ou options de programme différents)
Nous vous conseillons soit de les laisser à 11 soit à 10 (impossibilité d'écrire)

Les bits 21 étant les plus important.
Au fur et à mesure des découvertes que je fais ou que vous faites ces trucs et astuces sont mis à jours et envoyés aux lecteurs.

A suivre…
Mon site http://perso.wanadoo.fr/bils-instruments/ ou vous trouverez deux circuits de programmation
L'un dédié au circuit 28 pattes(M8) et l'autre aux 40 pattes (analogiques 8535 atmega 32 /323…

Un site indispensable : www.grote.net/bascom c'est une mailing-list Bascom très active.

REMERCIEMENTS

A Mark pour sa gentillesse.

REFERENCES BIBLIOGRAPHIQUES

Mark Alberts - Bascom-avr (version 1.11.7.7 et logiciels d'aides du programme)2004

Christian Tavernier - Microcontrôleurs AVR - DUNOD ® 2001

Psion® - Organiser II comms-links 1989 (pour les tableaux de codes ASCII des transferts RS232)

Microsoft® guide de programmations Quick Basic 4.5 (pas de date ?) pour les comparaisons.

Et mes cours d'informatiques au CNAM et à la CCIP électronique et microcontrôleurs.