

Projet Systèmes Numériques

Implémentation logicielle d'un fréquencemètre sur un HC11

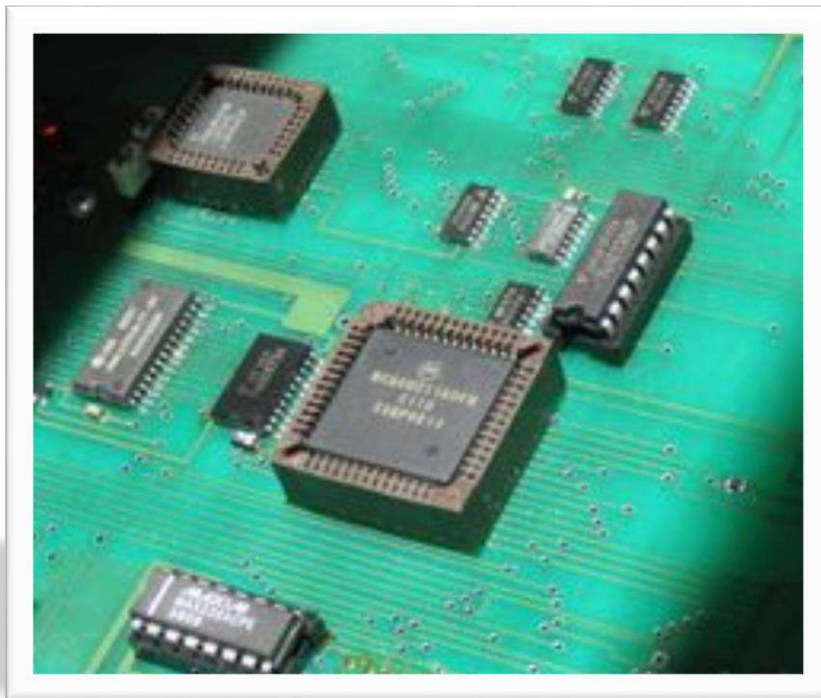


Table des matières

Introduction.....	2
PROGRAMMATION D'UNE E/S PAR SCRUTATION : LE POLLING	3
Le Polling ?.....	3
But de nos manipulations.....	3
Applications et mesures	3
Cycles.....	3
Durée d'un processus.....	4
Mesure d'une période.....	5
PROGRAMMATION D'UNE E/S PAR INTERRUPTION.....	11
L'interruption.....	11
Application de la méthode et mesures	11
Interruption du programme	11
Programmation d'un LCD	14
Procédure : Initialisation du LCD	14
Affichage d'un caractère	15
Affichage d'une chaîne de caractères	15
Conversion d'un entier sur 1 ou 2 octets	17
Conversion d'un entier sur 1 octet	17
Communication par liaison série asynchrone	19
Conclusion	22

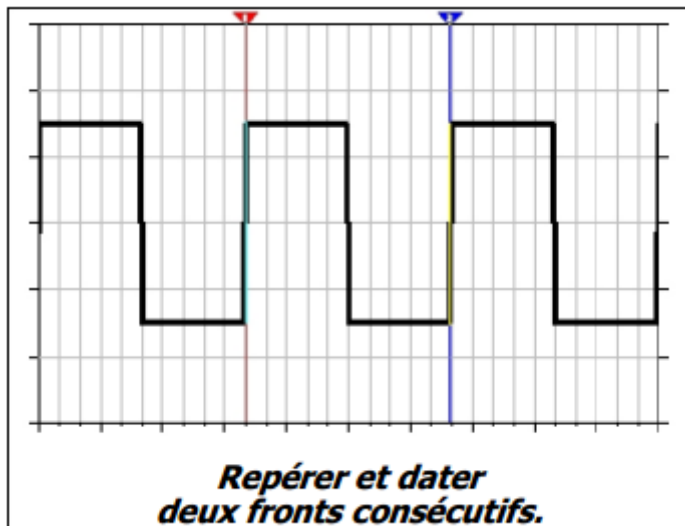
Introduction

- Implémentation logicielle d'un fréquencemètre sur un HC11 (compteur 16 bits, TCNT)
- Comparaison des différentes méthodes de programmation d'un fréquencemètre, avec affichage LCD
- Utilisation du port série ordinateur
- Logiciel AxIDE
- Langage assembleur

PROGRAMMATION D'UNE E/S PAR SCRUTATION : LE POLLING

Le Polling ?

En anglais, **poll** signifie « sondage ». Dans notre cas, le principe est assez simple, on scrute en permanence l'arrivée du premier front du signal. On attend un second de même nature, et une fois ceci fait, on en déduit la période. En voici un exemple :



But de nos manipulations

Avec la méthode de Polling, on cherche à compter les fronts à l'aide du TCNT, qui est le compteur 16 bits HC11. On va donc chercher à savoir si les cycles de l'horloge du CPU et celle du TCNT sont synchronisés. Ainsi, on pourra définir la durée d'une période.

Applications et mesures

Cycles

On cherche tout d'abord dans la documentation, qui nous indique qu'un **cycle dure 0,5 µs** et que la fréquence du TCNT est définie en fonction du CPU. À l'appel du TCNT on choisit si l'on divise la fréquence par 2, 4, 8, ou plus en modifiant les valeurs de PR1 et PR0 dans le registre.

Dans notre cas on utilise la valeur de la division par défaut : 1. Le **cycle du TCNT est donc égal à celui du CPU**.

Après avoir assemblé le programme avec l'option de comptage des cycles machines et exécuté le programme, notre fichier LST nous donne :

```
0001 100e                TCNT equ    $100E
0002 2000                org      $2000
0003 2000 fe 10 0e      [ 5 ]    ldx    TCNT
0004 2003 01            [ 2 ]    nop
0005 2004 18 fe 10 0e    [ 6 ]    ldy    TCNT
0006 2008 39            [ 5 ]    rts
```

Fichier LST ?

Le fichier LST est celui qui nous donne les adresses, le code machine, les opérandes, et la source. Il indique les éventuelles erreurs.

On remarque que seulement 1 cycle sépare les fins de chargements de ldx et ldy. Le nombre de cycle est inscrit entre les crochets.

On fait une mesure du nombre de cycles entre 2 chargements. Le programme commence à l'adresse 2000 et l'on fait deux « load » de la valeur du TCNT. Le compteur va alors s'incrémenter entre la mémorisation du registre x et y. En faisant la différence, on trouve le nombre d'incrémentations.

Entre la fin des deux chargements, il y a **8 cycles** (2 pour « nop » et 6 pour « ldy »), soit environ 4 µs.

Vérifions à présent ce résultat en comparant les valeurs des registres X et Y.

En faisant CALL 2000, on relève l'état des registres X et Y. Le registre Y donne **489A** et le registre X donne **4892**.

```
>call 2000  
P-2000 Y-489A X-4892 A-FF B-FF C-DO S-0041
```

En décimal, le registre Y : $489A_{16}=18586_{10}$

En décimal, le registre X : $4892_{16}=18578_{10}$

La différence d'Y-X vaut 8.

Nous avons donc **8 cycles**.

Le TCNT a une période par horloge de 500 ns, d'où une fréquence de **2MHz** également.(celle du CPU)

Ainsi, le rapport entre l'horloge du CPU et celui du TCNT est de **1**.

Durée d'un processus

On sait maintenant mesurer la durée d'une instruction sachant que le cycle du TCNT est égal à celui du CPU, et qu'un cycle a une durée de 0,5µs. Intéressons-nous maintenant à la durée d'un processus.

Le 68HC11 n'effectue pas par défaut de soustraction entre les registres X et Y, il va falloir alors réaliser un programme qui va stocker la valeur initiale du TCNT dans les deux premiers octets de 2000, et dans un second temps charger de nouveau la valeur du TCNT afin de la comparer avec la valeur stockée précédemment. Le résultat sera alors mit dans les deux octets suivants en \$2000.

Après assemblage du programme, nous obtenons le fichier .lst suivant :

0001	100e		TCNT equ	\$100E
0002				
0003	2000		org	\$2000
0004	2000		initial	rmb 2
0005	2002		delta	rmb 2
0006				
0007				
0008	2050		org	\$2050
0009				
0010	2050 fc 10 0e	[5]	ldd	TCNT
0011	2053 fd 20 00	[5]	std	initial
0012				
0013	2056 fc 10 0e	[5]	ldd	TCNT
0014	2059 b3 20 00	[6]	subd	initial
0015	205c fd 20 02	[5]	std	delta

```
0016
0017 205f 39          [ 5 ]      rts
```

Les deux opérations de chargement nous donnent 5 comme cycle, ainsi 10 cycles entre les deux chargements. On va déterminer si cette valeur est bien présente dans le résultat de la soustraction, qui, comme rappelé ci-dessus, est stocké sur 2 octets aux adresses 2002 et 2003.

Différence

```
>md 2000
2000 B7 E6 00 0A 18 FE 10 0E 39 F8 53 FB 3E 37 9B A7          9 S >7
2010 1D 37 F9 EF F9 FF FF AD E5 E6 F5 FB 7F FB ED AB 7
2020 9A 9E 57 ED D6 9F 53 61 F9 FE FF E7 FF C1 67 FF W Sa g
```

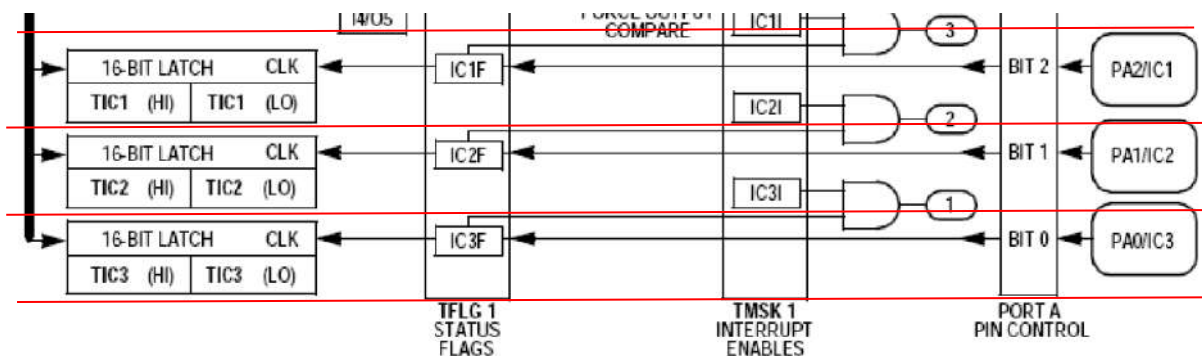
On trouve alors $000A]_{16}=10]_{10}$

On obtient bien le résultat souhaité. Par la même occasion, on observe une erreur de 5 cycles. Le comptage se déroule entre les deux appels de TCNT, et ainsi le temps de l'instruction « std initial » est en trop, ce qui nous ramène à cette erreur de 5 cycles.

Mesure d'une période

Il s'agit maintenant de réaliser la mesure. Pour cela il faut rechercher les **registres acteurs pour les entrées PA0, PA1 et PA2**.

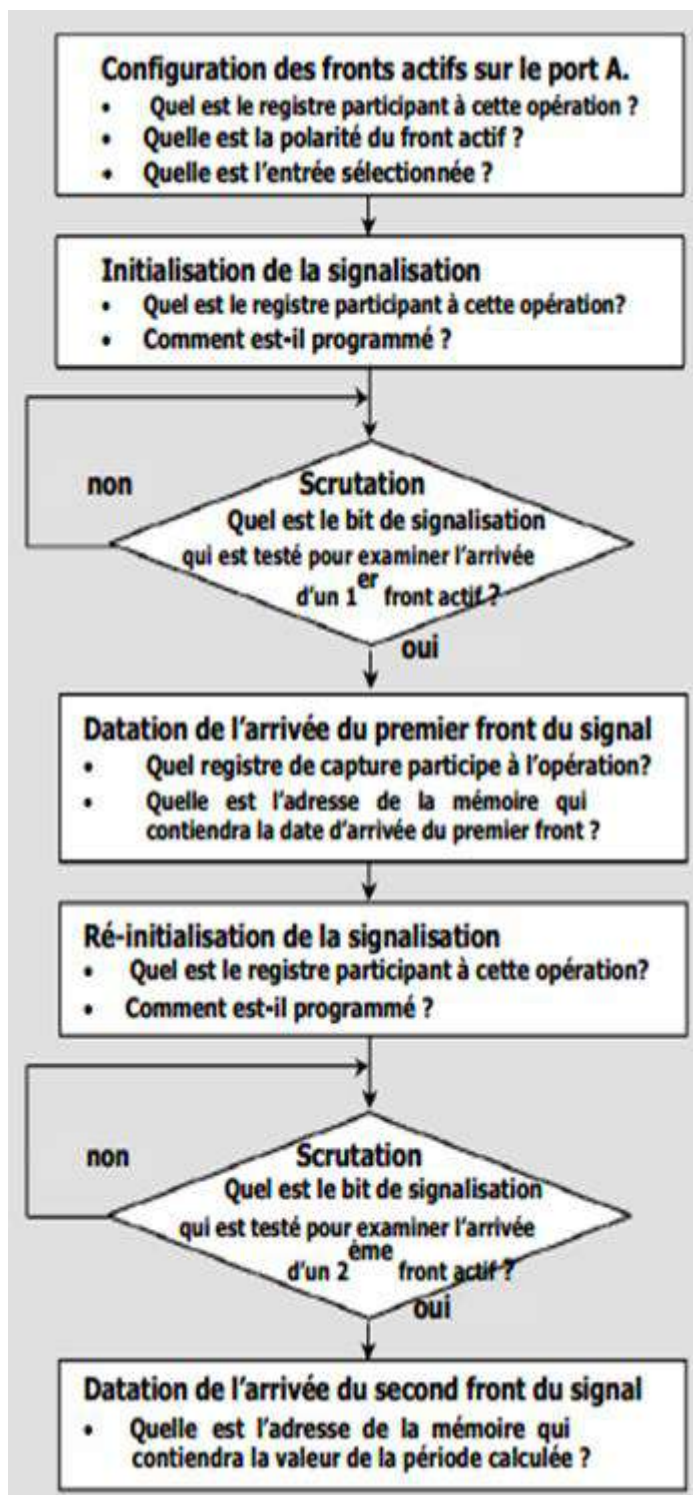
On va alors chercher dans la documentation du timer :



On distingue bien dans le schéma fonctionnel du timer trois registres : PA2, PA1 et PA0 dont voici les caractéristiques.

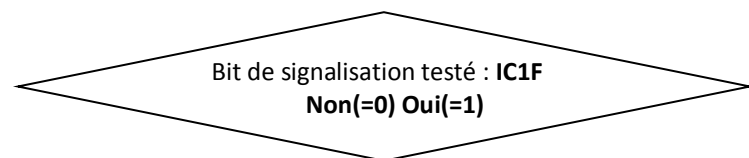
- PA2 : TFLG avec le bit IC1F, TIC1, TMSK1 avec le bit IC1I
- PA1 : TFLG avec le bit IC2F, TIC2, TMSK1 avec le bit IC2I
- PA0 : TFLG avec le bit IC3F, TIC3, TMSK1 avec le bit IC3I

Voici l'organigramme du programme à compléter :



- ✓ Le registre participant à l'opération est le **registre de contrôle TCTL2**
- ✓ Le front actif est initialisé avec &10 donc c'est un **front descendant**
- ✓ L'entrée sélectionnée est **PA2**

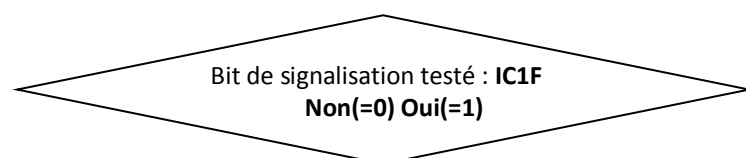
- ✓ Le registre participant à l'opération est le **registre de signalisation TFLG1**
- ✓ Il est **chargé dans l'accumulateur A** pour servir de masque au ET logique



INTERRUPTION

- ✓ Le registre de capture qui participe à l'opération est **TOCX**
- ✓ La date d'arrivée sera stockée aux adresses **2050 et 2051**

- ✓ Le registre participant à l'opération est **TFLG1**
- ✓ Il est **chargé dans l'accumulateur A** pour servir de masque au ET logique.



- ✓ La valeur de la période calculée sera aux adresses **2052 et 2053**

Voici le programme que l'on va exécuter, commentons le ligne par ligne :

```
TIC1 equ    $1010          //On donne le nom « TIC1 » à l'adresse 1010
TFLG1 equ    $1023          //On donne le nom « TGLG1 » à l'adresse 1023
TCTL2 equ    $1021          //On donne le nom « TCTL2 » à l'adresse 1021

    org      $2050          //Le programme commence en RAM à l'adresse 2050
front rmb     2             //On réserve 2 octets en mémoire pour front
period rmb    2             //On réserve 2 octets en mémoire pour period
ED1ba fcb     $10           //On crée une variable ED1ba de valeur $10

    org      $2100          //On se place à l'adresse 2100

ldab  ED1ba    //On charge ED1ba dans le registre b
stab  TCTL2    //On copie la valeur de l'accumulateur b dans TCTL2

ldaa  TFLG1    //On charge TFLG1 dans le registre a
anda  #$04     //ET logique entre l'accu. A et la valeur argument
staa  TFLG1    //On copie la valeur de l'accumulateur a dans TFLG1

lire1 ldaa  TFLG1 //On charge TFLG1 dans le registre a
      anda  #$04 //ET logique entre l'accu. A et la valeur argument
      beq   lire1 //Branch Equal: branche dans le cas où le résultat de
l'opération précédente est égal à zéro

suite1 ldd   TIC1
      std   front

      ldaa  TFLG1 //On charge dans l'accu. a TFLG1
      anda  #$04 //ET logique entre l'accu. a et la valeur argument
      staa  TFLG1 //On copie la valeur de l'accumulateur a dans TFLG1

lire2 ldaa  TFLG1 //On charge dans l'accu. a TFLG1
      anda  #$04 //ET logique entre l'accu. a et la valeur argument
      beq   lire2 //Branch Equal: branche dans le cas où le résultat de
l'opération précédente est égal à zéro

suite2 ldd   TIC1
      subd  front

      std   period

      rts
```

Il y a 29 cycles machines qui séparent les deux opérations de chargement dans l'accumulateur D.

On trouve ci-dessus le résultat de la compilation, soit le fichier LST, qui nous affiche le nombre de cycles.

```

0001 1010          TIC1 equ    $1010
0002 1023          TFLG1     equ    $1023
0003 1021          TCTL2     equ    $1021
0004
0005 2050          org      $2050
0006 2050          front     rmb      2
0007 2052          period    rmb      2
0008 2054 10       ED1ba     fcb     $10
0009
0010 2100          org      $2100
0011
0012 2100 f6 20 54   [ 4 ]     ldab     ED1ba
0013 2103 f7 10 21   [ 4 ]     stab     TCTL2
0014
0015 2106 b6 10 23   [ 4 ]     ldaa     TFLG1
0016 2109 84 04      [ 2 ]     anda     #$04
0017 210b b7 10 23   [ 4 ]     staa     TFLG1
0018
0019 210e b6 10 23   [ 4 ]     lire1     ldaa     TFLG1
0020 2111 84 04      [ 2 ]     anda     #$04
0021 2113 27 f9      [ 3 ]     beq      lire1
0022
0023
0024 2115 fc 10 10   [ 5 ]     suite1     ldd      TIC1
0025 2118 fd 20 50   [ 5 ]             std      front
0026
0027
0028 211b b6 10 23   [ 4 ]     ldaa     TFLG1
0029 211e 84 04      [ 2 ]     anda     #$04
0030 2120 b7 10 23   [ 4 ]     staa     TFLG1
0031
0032 2123 b6 10 23   [ 4 ]     lire2     ldaa     TFLG1
0033 2126 84 04      [ 2 ]     anda     #$04
0034 2128 27 f9      [ 3 ]     beq      lire2
0035
0036
0037 212a fc 10 10   [ 5 ]     suite2     ldd      TIC1
0038 212d b3 20 50   [ 6 ]             subd     front
0039
0040 2130 fd 20 52   [ 5 ]             std      period
0041
0042 2133 39         [ 5 ]             rts

```

Il s'agit maintenant de **tester** la méthode avec **différentes fréquences** en cherchant les limites, pour des fréquences de 40 à 40 kHz.

On branche la sonde sur le fréquencesmètre. (PA2 et GND)

F(Hz)	Polling (nombre de cycles)	Période(s)	Fréquence(Hz)
40	C35F = 50015	0,0250075	39,9880036
100	4E18 = 19992	0,009996	100,040016
400	1388 = 5000	0,0025	400
1000	07CF = 1999	0,0009995	1000,50025
4000	01F4 = 500	0,00025	4000
10000	00C8 = 200	0,0001	10000
40000	0032 = 50	0,000025	40000

$$Période = Cycles * 0.5 \mu s$$

$$Fréquence = 1/Periode$$

Afin de connaître le Polling, on relève les registres A et B.

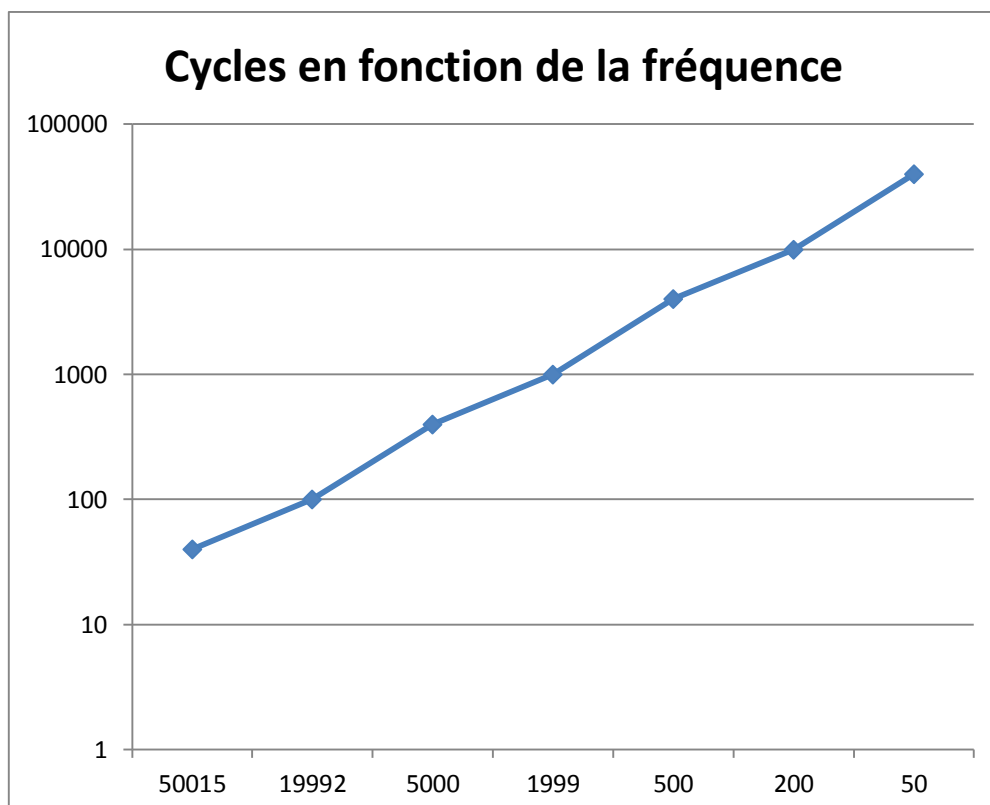
Un exemple d'appel « CALL » à l'adresse 2100 pour 40Hz :

```
>CALL 2100  
P-2100 Y-FFFF X-FFFF A-C3 B-5F C-D8 S-0041
```

Et pour les autres...

```
>CALL 2100  
P-2100 Y-FFFF X-FFFF A-4E B-18 C-D0 S-0041  
>CALL 2100  
P-2100 Y-FFFF X-FFFF A-13 B-88 C-D0 S-0041  
>CALL 2100  
P-2100 Y-FFFF X-FFFF A-07 B-CF C-D0 S-0041  
>CALL 2100  
P-2100 Y-FFFF X-FFFF A-01 B-F4 C-D0 S-0041  
>CALL  
P-2100 Y-FFFF X-FFFF A-00 B-C8 C-D0 S-0041  
>CALL 2100  
P-2100 Y-FFFF X-FFFF A-00 B-C8 C-D0 S-0041  
>CALL 2100  
P-2100 Y-FFFF X-FFFF A-00 B-32 C-D0 S-0041
```

Pour le tableau de données précédent, en échelle logarithmique (log-log), on trace de graphique suivant, qui représente l'évolution du nombre de cycle par rapport à la fréquence.



La notice nous donne un temps de cycle machine de 0,5 us.

La courbe obtenue est en fait décroissante. En effet, la période étant inversement proportionnelle à la fréquence, lorsque la fréquence augmente, la période diminue.

On retrouve bien les fréquences, le fonctionnement du fréquencemètre est ainsi vérifié.

La limite de HF est 69kHz correspondant au nombre de cycle inférieur au nombre de cycle machine. Au minimum, le programme exécute 29 cycles, et nous savons qu'un cycle est de 500 ns, on obtiendra alors le temps d'exécution minimal, et d'où la fréquence maximale possible en entrée :

$$f_{\max} = \frac{1}{T} = \frac{1}{29 \cdot (1 \text{ cycle})} = \frac{1}{29 * 5.10^{-7}} = \mathbf{69 \text{ kHz}}$$

Au delà de 69 kHz, le Timer émet trop d'erreur et calcule trop lentement.

Au maximum, sur 16 bits, on ne peut dépasser FFFF, ainsi on aura 65535 cycles :

Ainsi, pour FFFF]₁₆ cycles :

$$f_{\min} = \frac{1}{T} = \frac{1}{FFFF \cdot (1 \text{ cycle})} = \frac{1}{65\,535 * 5.10^{-7}} \approx \mathbf{31 \text{ Hz}}$$

Au-dessous de cette valeur, la valeur codée sur 16 bits sera erronée (overflow).

PROGRAMMATION D'UNE E/S PAR INTERRUPTION

On teste ici notre deuxième méthode : l'interruption.

L'interruption

Le principe de cette méthode est de créer une notion d'« événementiel », d'un événement qui « interrompt » le fonctionnement normal. Le programme est interrompu lors de l'arrivée d'un premier front et en déduit directement la période à l'arrivée du second front.

Application de la méthode et mesures

Pour réaliser les mesures, nous utilisons les registres TIC1, TFLG1, TMSK1 et TCTL2, soit au final l'entrée PA2. On va dans un premier temps commencer par apprendre à utiliser l'interruption pour ensuite l'appliquer.

Interruption du programme

Notre programme tourne indéfiniment dans la boucle infinie suivante :

```
ldd    #$0000 ;
std    $2000
bra    boucle ;
```

On réitère ici le chargement du contenu à l'adresse 0.

Cependant, on crée un événement qui va interrompre cette boucle :

```
sei    ;
lds    #$0200
ldab   ED1ba
stab   TCTL2
ldaa   TFLG1
anda   #$04
staa   TFLG1
ldaa   #$04 ;
staa   TMSK1 ;
cli    ;
```

On appelle cela **l'événement d'interruption**. Les fonctions « sei » et « cli » sont des fonctions permettant l'activation et la désactivation de l'option d'interruption. Elles modifient la valeur du bit 1 dans le registre CCR.

Au moment où il y a un signal en entrée de PA2, on écrit à l'adresse \$00E8 ; le programme va alors quitter sa boucle infinie et à l'aide de la commande « jump » on va exécuter la partie « issprog ».

Le vecteur d'interruption : Le FLAG passe à 1, on sort alors de la boucle infinie et on se rend en une adresse prédéfinie

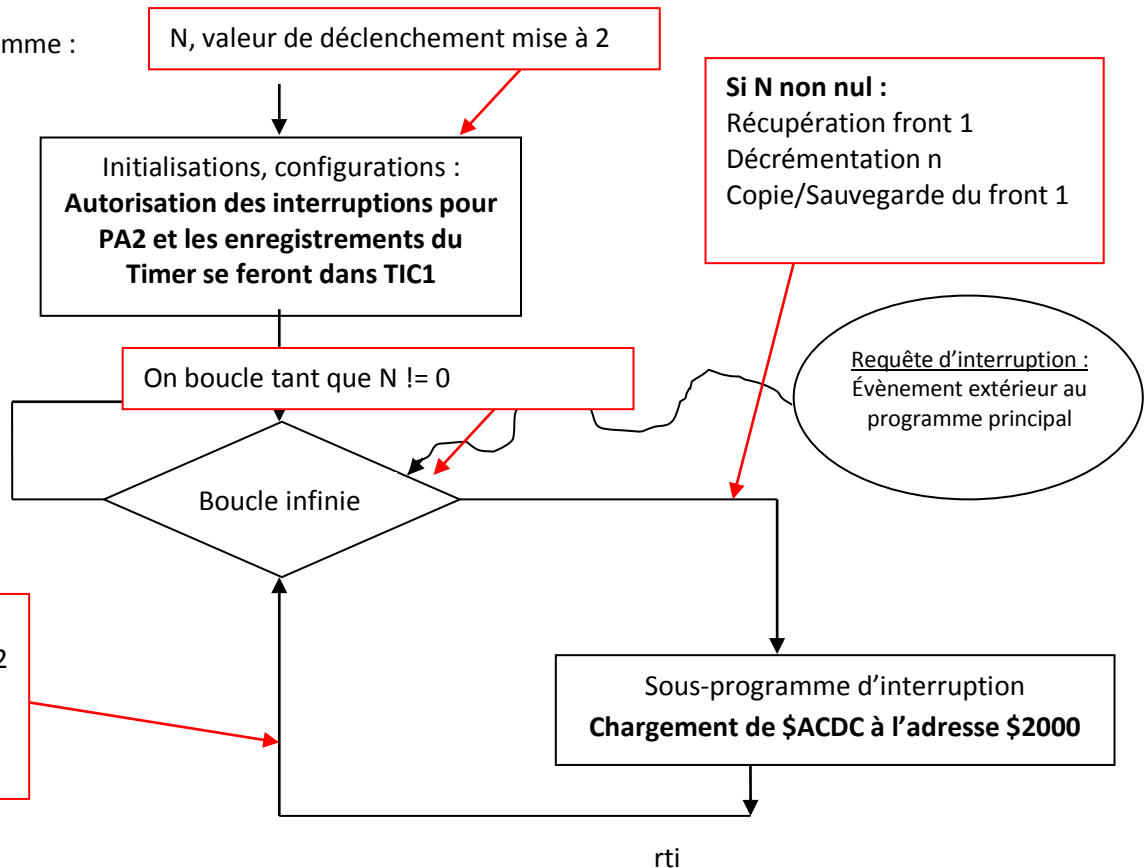
```
org    $00E8
jmp    issprog
```

Voici le sous-programme d'interruption, qui est exécuté à l'appel d'« issprog ».

```
issprog    ldd    #$ACDC ;
            std    $2000 ;
```

Complétons maintenant l'organigramme :

Organigramme :



On va relever l'état de la mémoire aux adresses 2000 et 00E8 avant et après l'exécution :

Adresses	Avant interruption	Après interruption
00E8	<pre> >md 00E8 00E0 E3 71 7E E3 71 7E E3 71 7E 21 20 7E E3 71 7E E3 q q q ! q 00F0 71 7E E3 71 7E FA 7A 7E E3 71 7E E3 00 7E E3 71 q q z q q 0100 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 0110 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 0120 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 0130 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 0140 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 0150 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 0160 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF > </pre>	<pre> >md 00E8 00E0 E3 71 7E E3 71 7E E3 71 7E 21 20 7E E3 71 7E E3 q q q ! q 00F0 71 7E E3 71 7E FA 7A 7E E3 71 7E E3 00 7E E3 71 q q z q q 0100 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 0110 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 0120 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 0130 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 0140 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 0150 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 0160 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF > </pre>
2000	<pre> >md 2000 2000 6F F5 E8 DF EE BE ED FF 7B F6 FF 5F 3B AF FD 76 o _; v 2010 7E FB FB DD EB FD FE FE CE 87 F1 BF DA 72 AF DF r 2020 FD 7F B2 E7 A4 CD ED EA 7D CF OF 2F BE 7F FB 7E / 2030 F6 F7 DA D9 F7 EF E6 9E 95 79 3F 5D 4F 5F AD EB y?]0_ 2040 7F BB B2 BE A0 5F FD ED 9B EF F4 F8 CE 2C 7F B6 / 2050 AF FB F9 44 10 6F E3 77 EF 2F E7 B1 7D 38 B9 7D D o w / 8 2060 EF FF E7 D7 F1 CD FF 56 F5 74 5D BB F9 49 71 B6 V t] Iq 2070 BE B3 E8 FF A3 3E 3F 1F 1E 6D FD F9 FB BD DD 86 >? m 2080 DE 3E F4 F5 F9 E6 AF FD 6A F7 58 26 FB 3F FD BA > j X& ? </pre>	<pre> done >md 2000 2000 AC DC E8 DF EE BE ED FF 7F F6 FF 5F 1B AF FD 76 _ v 2010 7E FB 7B D1 EB FD FE FE CE 87 F1 BF DA 72 AF DF r 2020 FD 7B B2 E7 A4 CD ED EA 7D CF 0B AF FE 7F FB 7E 2030 E6 77 DA D9 F7 6F E6 9E 95 79 3F 5D 4E 5F AD EB w o y?]N_ 2040 5F BB B2 BE A0 5F FD ED 9B EF F4 F8 CE 2C 7F 36 _ , 6 2050 DD FD 00 32 10 67 63 77 EF 2F E7 B9 3D 30 A9 7D 2 gcw / =0 2060 EF FB E7 D7 B1 8D FF 56 F5 75 5F 3B F9 49 71 B6 V u; Iq 2070 36 B3 E9 FF E3 3F 1F 1E 6D FD F9 FB BD DD 86 6 ?? m 2080 DE 3E F4 F7 F9 E6 AE 7D 6A F7 58 26 FB 3F 7D BA > j X& ? </pre>

On lance une première fois le programme, sans brancher l'alimentation sur l'entrée PA2, on obtient les valeurs 6F et F5 aux adresses 2000/2001, ainsi il n'y a pas d'interruption. À contrario, si on applique une tension, la valeur change et l'on obtient bien ACDC, on a donc réussi à faire une interruption ! La valeur contenue à l'adresse \$00E8 elle ne change pas ; elle vaut E3 et 71 dans les deux cas, ce qui est normal.



/!\ Messieurs ACDC, plus d'interruption SVP !!

Voici le fameux programme.

Le principe du programme est le suivant : le programme commence par des initialisations/configurations, il rentre ensuite dans une boucle infinie qui ne s'arrête que s'il y a une interruption. Le programme exécute alors un sous-programme, puis de nouveau on retourne dans cette boucle infinie.

Le programme est le suivant :

```

TIC1 equ    $1010      //TIC1 se situe à l'@ 1010
TFLG1 equ    $1023      //TFLG1 se situe à l'@ 1023
TCTL2 equ    $1021      //TCTL2 se situe à l'@ 1021
TMSK1 equ    $1022 ;    //TMSK1 se situe à l'@ 1022
      org    $2050      //On se positionne à l'@ 2050
front rmb    2          //On réserve 2 octets pour front
period rmb   2          //On réserve 2 octets pour period
ED1ba fcb    $10        //Activation fronts descendants

      org    $00E8
      jmp    issprog

      org    $2100      //On se positionne à l'@ 2100

sei          ;          //début des interruptions
lds          #$0200

ldab ED1ba
stab TCTL2

ldaa TFLG1
anda #$04
staa TFLG1

ldaa #$04 ;
staa TMSK1 ;
cli          ;          //Fin des interruptions

ldd          #$0000 ;    //Place "0"
std          $2000      //à l'adresse 2000
boucle bra   boucle ;    //en boucle

issprog ldd   #$ACDC ;    //place "ACDC"
std       $2000 ;        //à l'adresse 2000

rti          //retourne au programme

```

Programmation d'un LCD

Procédure : Initialisation du LCD

Afin d'afficher les caractères, on nous donne deux sous programmes « longue » et « courte » qui permettent une **temporisation pour la mise à jour de l'affichage**. On s'intéresse ici au nombre de cycles qu'on obtient avec l'option de comptage.

Nous avons deux boucles avec de nombreuses itérations : 50000 et 1000 ; il faut donc modifier leur durée. Pour afficher octet par octet, on se restreint à un passage par boucle. A l'aide de l'exécution pas à pas, on en déduit les étapes :

- Initialisation écran (LCDCMD)
- Rafraichissement (CLRCH)
- Affichage curseur (DCBON)
- Resteindre son déplacement (CINSHOF)
- Avancer sa position de 1 vers la droite (SCHR)
- Déclarer position actuelle comme initiale (CSRHOM)

Voici le fichier LST correspondant

```
0001 b5f0          LCDCMD    equ    $b5f0
0002 b5f1          LCDDAT    equ    $b5f1
0003
0004 003c          L4xC20     equ    $3c * 4x20
0005 0001          CLRCH      equ    $01
0006 000f          DCBON      equ    $0f
0007 0006          CINSHOF    equ    $06
0008 0014          SCSHR      equ    $14
0009 0002          CSRHOM     equ    $02
0010
0011 2100          ORG        $2100
0012
0013 2100 86 3c     [ 2 ] init    ldaa    #L4xC20
0014 2102 b7 b5 f0 [ 4 ] staa    LCDCMD
0015 2105 bd 21 30 [ 6 ] jsr     longue
0016 2108 86 01     [ 2 ] ldaa    #CLRCH
0017 210a b7 b5 f0 [ 4 ] staa    LCDCMD
0018 210d bd 21 30 [ 6 ] jsr     longue
0019 2110 86 0f     [ 2 ] ldaa    #DCBON
0020 2112 b7 b5 f0 [ 4 ] staa    LCDCMD
0021 2115 bd 21 37 [ 6 ] jsr     courte
0022 2118 86 06     [ 2 ] ldaa    #CINSHOF
0023 211a b7 b5 f0 [ 4 ] staa    LCDCMD
0024 211d bd 21 37 [ 6 ] jsr     courte
0025 2120 86 14     [ 2 ] ldaa    #SCSHR
0026 2122 b7 b5 f0 [ 4 ] staa    LCDCMD
0027 2125 bd 21 37 [ 6 ] jsr     courte
0028 2128 86 02     [ 2 ] ldaa    #CSRHOM
0029 212a b7 b5 f0 [ 4 ] staa    LCDCMD
0030 212d bd 21 37 [ 6 ] jsr     courte
0031
0032 2130 ce c3 50   [ 3 ] longue    ldx     #50000
0033 2133 09         [ 3 ] loop_l    dex
0034 2134 26 fd      [ 3 ] bne     loop_l
0035 2136 39         [ 5 ] rts
0036
0037 2137 ce 03 e8   [ 3 ] courte    ldx     #1000
0038 213a 09         [ 3 ] loop_c    dex
0039 213b 26 fd      [ 3 ] bne     loop_c
0040 213d 39         [ 5 ] rts
```

Affichage d'un caractère

```
Init //Initialisation Q11

suite ldaa #'A' //On charge le caractère 'A' dans l'accumulateur
      staa LCDDAT //On applique le caractère
      jsr courte //Temporisation courte (caractères)
      ldaa #'B' //IDEM...
      staa LCDDAT
      jsr courte
      ldaa #'C'
      staa LCDDAT
      jsr courte
      ldaa #'D'
      staa LCDDAT
      jsr courte
```

On obtient bien les lettres A, B, C, et D à la suite, comme montré ci-dessous.



En effet, après l'exécution du programme compilé correctement, on a réussi à obtenir l'affichage des caractères sur notre écran LCD programmable. Pour cela, on a utilisé la commande « CALL 2100 » qui exécute la procédure présente à l'adresse \$2100.

Affichage d'une chaîne de caractères

On souhaite à présent afficher une chaîne de caractères. Pour cela, on se sert du programme donné et on le complète. La directive « **fcc** » permet de créer une variable qui contient les caractères ASCII. Elle réserve la place nécessaire en octets pour la chaîne.

```
//INITIALISATION

      org      $2000 //On se place à l'adresse 2000
      texto fcc 'La periode =' //La variable texto réserve l'espace nécessaire pour contenir la
chaîne indiquée, à savoir « La periode = »
      fin fcb #0 //Pointeur de « fin » : Déclaration d'une constante sur un octet de
valeur 0

      org      $2100 //On se place à l'adresse 2100
```

```

//INIT Q11

    ldy    #texto // On charge dans le registre y la valeur de texto

chaîne ldaa    0,y    //Pointeur « chaîne » : On charge dans l'accumulateur la valeur contenue
    beq    suite //On branche à suite si le résultat est nul→fin de la chaîne
    staa   LCDDAT //Enregistrement du contenu de A dans la variable de l'écran LCD
    jsr    courte //Saut au pointeur courte
    iny    //Incrémementation y
    bra    chaîne //On retourne à chaîne

suite ...    //CF Q12

```

Le programme affiche bien « La periode =ABCD », comme spécifié dans le code source. La preuve en image...



Conversion d'un entier sur 1 ou 2 octets

Conversion d'un entier sur 1 octet

On commence par compléter les lignes manquantes de l'algorithme qui servira à convertir un entier de 1 octet(deux quartets) en deux caractères(un quartet haut et un quartet bas) à partir du label conv.

```
org    $2000           //on déclare les variables à l'adresse 2000
texto fcc 'La periode=' //la variable texto réserve l'espace nécessaire pour contenir la
chaîne indiquée, à savoir 'La période='

fin    fcb    #0        //pointeur "fin" : déclaration d'une constante sur un octet de
valeur 0

ORG    $2020           //début du programme à l'adresse 2020
table fcc '0'          //réservation du caractère ASCII '0'
      fcc '1'          //réservation du caractère ASCII '1'
      fcc '2'          //réservation du caractère ASCII '2'
      fcc '3'          //réservation du caractère ASCII '3'
      fcc '4'          //réservation du caractère ASCII '4'
      fcc '5'          //réservation du caractère ASCII '5'
      fcc '6'          //réservation du caractère ASCII '6'
      fcc '7'          //réservation du caractère ASCII '7'
      fcc '8'          //réservation du caractère ASCII '8'
      fcc '9'          //réservation du caractère ASCII '9'
      fcc 'A'          //réservation du caractère ASCII 'A'
      fcc 'B'          //réservation du caractère ASCII 'B'
      fcc 'C'          //réservation du caractère ASCII 'C'
      fcc 'D'          //réservation du caractère ASCII 'D'
      fcc 'E'          //réservation du caractère ASCII 'E'
      fcc 'F'          //réservation du caractère ASCII 'F'

octet  fcb    #$36      //octet de test
q_haut rmb    1         //réservation d'un octet au label q_haut
q_bas  rmb    1         //réservation d'un octet au label q_bas

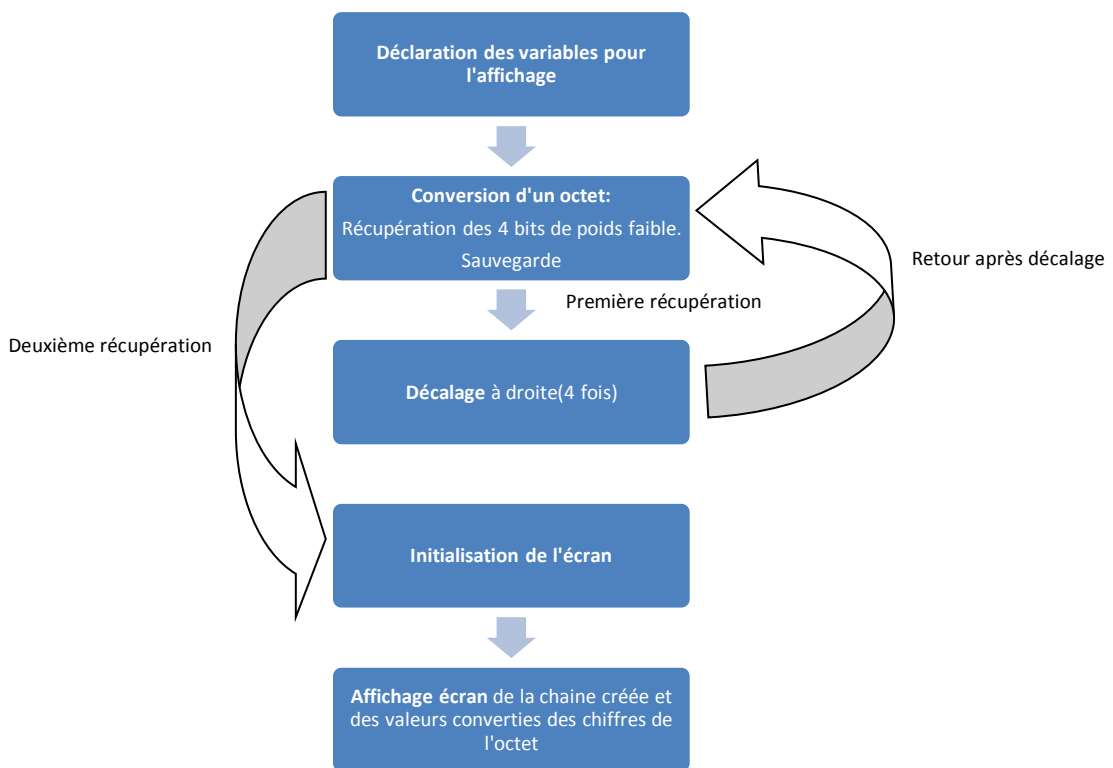
conv   ORG    $2100      //début du programme à l'adresse 2100
      ldax   #table     //label conv : on charge dans le registre x la valeur de table
      ldaa   octet      //on charge le contenu de octet dans a
      anda   #$0F        //on fait un ET logique avec a et 00001111 (intérêt : 4 bits de
                        //poids faible)
      adda   0,x         //on ajoute la valeur a dans le registre x
      staa   q_bas       //enregistrement du contenu de a dans la variable q_bas
      ldaa   octet      //on re-charge le contenu de octet dans a
      asra                   //décalage à droite
      asra                   //décalage à droite
      asra                   //décalage à droite
      asra                   //décalage à droite
      anda   #$0F        //on fait un ET logique avec a (récupération des 4 bits de poids
                        //fort)
      anda   0,x         //on cherche la valeur du symbole qui correspond pour LCD
      staa   q_haut       //on enregistre la 2ème valeur, bit de poids fort
init   .....           // (cf q11) partie du programme d'initialisation
      ldy    #texto      //on charge dans le registre y la valeur de texto
chaîne  ldaa   0,y        //pointeur "chaîne" : on charge dans l'accumulateur a la valeur
                        //contenue à y+0
      beq    suite       //si la valeur précédente = 0 , on saute à l'instruction "suite"
      staa   LCDDAT       //enregistrement du contenu de a dans la variable donnée de
                        //l'écran LCD
      jsr    courte      //saut au pointeur d'instructions "courte"
      iny                   //on incrémente y
      bra    chaîne      //on retourne à "chaîne" quelle que soit l'instruction
suite   ldaa   q_haut      //on charge le contenu de q_haut dans a
      staa   LCDDAT       //enregistrement du contenu de a dans la variable donnée de
                        //l'écran LCD
      jsr    courte      //saut au pointeur d'instructions "courte"
      ldaa   q_bas        //on charge le contenu de q_bas dans a
      staa   LCDDAT       //enregistrement du contenu de a dans la variable donnée de
                        //l'écran LCD
      jsr    courte      //saut au pointeur d'instructions "courte"
      ldaa   # '.'        //on charge la valeur '.' dans a
      staa   LCDDAT       //enregistrement du contenu de a dans la variable donnée de
                        //l'écran LCD
      jsr    courte      //saut au pointeur d'instructions "courte"
```

```
ldaa    #'.'          //on charge la valeur '.' dans a
staa    LCDDAT        //enregistrement du contenu de a dans la variable donnée de
                        //l'écran LCD
jsr     courte        //saut au pointeur d'instructions "courte"
```

Pour pouvoir afficher sur un LCD un caractère saisi au clavier par un utilisateur, il est nécessaire de faire une conversion car le code ASCII est un entier. Nous rappelons qu'un caractère pèse un octet.

Illustrons l'algorithme par un organigramme (ci-dessous). Dans un premier temps, on charge la table qui sert de référence. Ensuite on récupère la valeur totale. Puis, on effectue un ET logique entre l'accumulateur et \$0F, puis on ajoute 0x à l'accumulateur, et enfin on a le stockage du bit de poids faible.

Par la suite, on reprend le même principe pour récupérer le bit de poids fort, mais on devra se décaler de 4 bits.



Nous avons alors réalisé un test grâce au programme et nous pouvons lire 'La periode=36..'



Communication par liaison série asynchrone

Voici le programme que nous avons utilisé

```
BAUD    equ    $102b    //On assigne l'adresse 102b au label BAUD
SCDR    equ    $102f    //On assigne l'adresse 102f au label SCDR
SCSR    equ    $102e    //On assigne l'adresse 102e au label SCSR
SCCR1    equ    $102c    //On assigne l'adresse 102c au label SCCR1
SCCR2    equ    $102d    //On assigne l'adresse 102d au label SCCR2

        org    $2200    //Debut
        clr    SCCR1    //on met à 0 le contenu de SCCR1 (clear)
        ldaa   #$30     //on charge la valeur 30 (en hexadécimal) dans l'accu a
        staa   BAUD     //on stocke la valeur de l'accu a à BAUD

        ldab   #$0C     //on charge la valeur 12 dans l'accu b
        stab   SCCR2    //on stocke la valeur de l'accu b à SCCR2

        clr    $2000    //On met à 0 ce que l'on a en 2000

loop    ldaa   SCSR      //Début boucle, on stocke dans l'accu a la valeur de SCSR
        anda   #$20     //on fait un ET entre a et valeur 20
        beq    loop     //Si le test précédent a mis le flag Z à 1, alors on boucle

        ldaa   SCDR     //On stocke dans a la valeur de SCDR
        staa   $2000    //On stocke a à l'adresse 2000

        rts            //fin
```

Lorsque l'on effectue ce programme, on demande en fait à l'utilisateur de faire une saisie que l'on va stocker à l'adresse 2000. On souhaite réaliser le déclenchement de l'affichage à partir de la saisie d'un caractère dans AxIDE. Afin de tester les méthodes de communication avec le port série, nous avons tenté d'effectuer des saisies pendant une scrutation du programme. Le choix du Polling s'est tout de suite imposé à nous car il est bien plus facile de tester si le programme reçoit bien notre signal.

Les méthodes de Polling et d'interruption présentées sont toutes deux très différentes sauf qu'on relève un inconvénient majeur dans la méthode de Polling car on ne peut obtenir directement l'affichage du caractère tapé au clavier par l'utilisateur. Contrairement à la méthode d'interruption, l'affichage est quasi-instantané et il n'est pas nécessaire de réinitialiser la carte électronique.

On a pris un exemple pour illustrer le fonctionnement de ce programme. Dans l'exemple ci-dessous, on appelle le programme situé à l'adresse 2200 et on fait la saisie de la lettre 'a' (son code est 61). Le programme s'arrête alors. On recommence une seconde fois l'opération, et on saisit cette fois-ci la lettre 'b' (son code est 62).

```

P-2200 Y-FFFF X-FFFF A-62 B-0C C-D0 S-0041
>

P-2200 Y-FFFF X-FFFF A-6D B-0C C-D0 S-0041
>call 2200

P-2200 Y-FFFF X-FFFF A-61 B-0C C-D0 S-0041
>md 2000

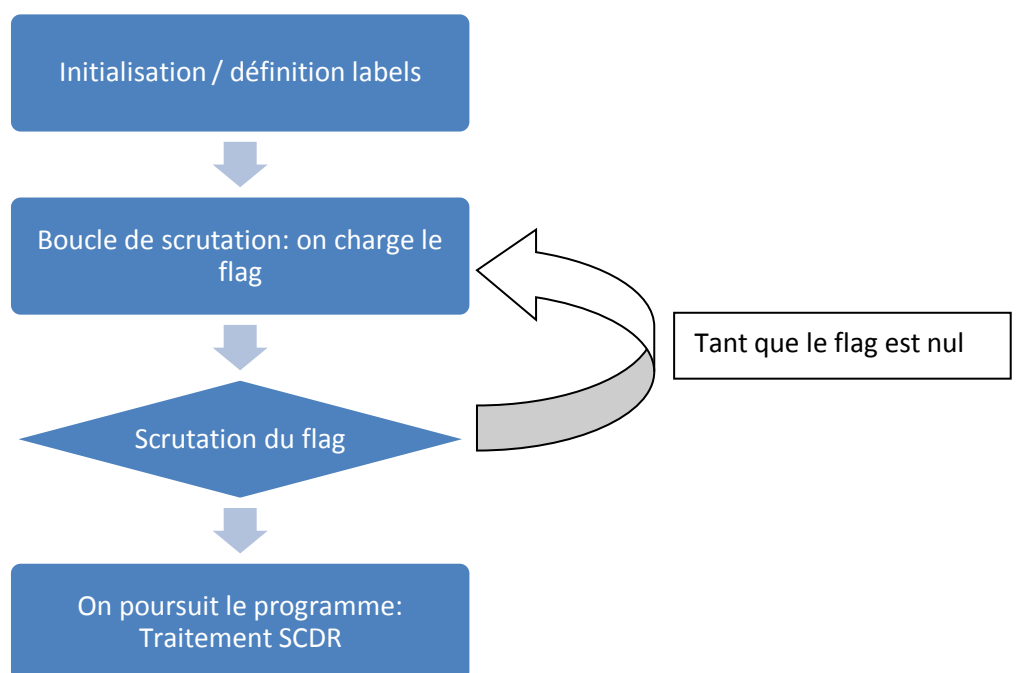
2000 61 CE 3E 1A FB 1C FF 7C 7D 79 80 7F 01 F9 9D 36 a > y 6
2010 BC 9A F5 9A D8 8F 78 47 3C 0E ED 3F DD D3 F8 45 xG< ? E
2020 C9 AD 9C 9F 3D D1 DF C0 83 71 FF 9F 36 37 93 7F = q 67
2030 AE F3 B3 FF 70 5E F1 0F BA BE E4 F5 BB B6 B1 73 p+ s
2040 D5 4D DF F0 9D F9 6F 39 5C C6 3F 4F 5C FA F6 F1 H o9\ 20\
2050 3F A3 C5 16 75 5F 6F 53 DF 96 EE E7 EF EA 3E 26 ? u_oS >4
2060 FB E7 68 BB F7 FD 39 F0 2E A5 FE D6 DF E7 3C 06 h 9 . <
2070 77 C7 8B DF 5A CF E9 E7 69 DB 16 FE 1D 7B 9F BD w Z i
2080 F7 37 D9 3C C7 C0 5B 6C CB 1A A6 EF DA EF CB F9 7 < [1
>call 2200

P-2200 Y-FFFF X-FFFF A-62 B-0C C-D0 S-0041
>md 2000

2000 62 CE 3E 1A FB 1C FF 7C 7D 79 80 7F 01 F9 9D 36 b > y 6
2010 BC 9A F5 9A D8 8F 78 47 3C 0E ED 3F DD D3 F8 45 xG< ? E
2020 C9 AD 9C 9F 3D D1 DF C0 83 71 FF 9F 36 37 93 7F = q 67
2030 AE F3 B3 FF 70 5E F1 0F BA BE E4 F5 BB B6 B1 73 p+ s
2040 D5 4D DF F0 9D F9 6F 39 5C C6 3F 4F 5C FA F6 F1 H o9\ 20\
2050 3F A3 C5 16 75 5F 6F 53 DF 96 EE E7 EF EA 3E 26 ? u_oS >4
2060 FB E7 68 BB F7 FD 39 F0 2E A5 FE D6 DF E7 3C 06 h 9 . <
2070 77 C7 8B DF 5A CF E9 E7 69 DB 16 FE 1D 7B 9F BD w Z i
2080 F7 37 D9 3C C7 C0 5B 6C CB 1A A6 EF DA EF CB F9 7 < [1

```

Maintenant le programme réalisé, établissons l'organigramme :



Il convient, pour terminer, d'effectuer le programme en mode interruption.

```

BAUD    equ    $102b    //On assigne l'adresse 102b au label BAUD
SCDR    equ    $102f    //On assigne l'adresse 102f au label SCDR
SCSR    equ    $102e    //On assigne l'adresse 102e au label SCSR
SCCR1   equ    $102c    //On assigne l'adresse 102c au label SCCR1
SCCR2   equ    $102d    //On assigne l'adresse 102d au label SCCR2

        org    $00C4    //Début : adresse correspondant à la requête d'interruption
        jmp    issprog   //Saut au sous-programme qui traite l'interruption

```

```

org      $2200

sei                      //On met le bit 1 de CCR à 1 pour interdire les interruptions
                        //masquées

clr      SCCR1           //on met à 0 le contenu de SCCR1 (clear)
clr      SCSR            //On met à 0 le SCSR
ldaa     #$30            //on charge la valeur 30 (en hexadécimal) dans l'accu a
staa     BAUD            //on stocke la valeur de l'accu a à BAUD

ldaa     #$2C            //on charge la valeur 44 dans l'accu a
staa     SCCR2           //on stocke la valeur de l'accu a à SCCR2

clr      $2000           //On met à 0 ce que l'on a en 2000
cli                      //On met le bit 1 de CCR à 0 ; fin déclaration interruption

loop     bra      loop   //Début boucle, on stocke dans l'accu a la valeur de SCSR

issprog  clr      SCSR   //Sous-programme d'interruption
          ldab     SCDR   //On stocke dans l'accumulateur b la valeur de SCSR
          stab     $2200  //On stocke cette valeur à l'adresse 2200

          rti          //On stoppe l'exécution

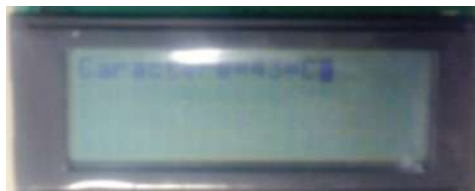
```

Pour sortir de la boucle infinie qui est sollicitée dans notre programme, nous avons alors tapé sur la touche 'c' du clavier qui a fait réagir un sous-programme signalant qu'une information est présente. Cette information envoyée est directement logée dans le registre appelé le SCSR et qui renvoie le code ASCII au registre SCDR. Afin de savoir ce qui s'est déroulé, nous avons interrogé la mémoire. L'interrogation mémoire est effectuée par la commande "MD2000" du logiciel AxIDE, et on observe d'après la requête la valeur contenue dans l'accumulateur A.

Pourquoi ?

Tout simplement parce que le programme sauvegarde dans l'accumulateur A et à l'adresse 2000 le contenu de la variable SCDR. Ce résultat que nous montre l'écran LCD est bien correct car en effet, nous avons réussi d'une part à convertir le caractère saisi au clavier, son code ASCII en tant que caractère affiché, mais également nous obtenons son code ASCII. Comme le montre la photo ci-contre, le caractère 'c' a bien pour code ASCII 63 dans la table universelle.

Et le caractère tapé au clavier est bien un 'c'. La réponse logicielle affichée à l'écran est le résultat d'un assemblage d'instructions que sont l'initialisation du LCD, la conversion des octets, l'affichage etc... mais surtout d'une exécution fonctionnelle.



Conclusion

- Langage assembleur
- Application concrète à un micro controleur
- blablalballaal