

01/06/2012

EFREI
2010

Projet systèmes numériques

Implémentation logicielle d'un fréquencemètre sur un HC11

Minh-hoang Nguyen | Rouley David

Projet systèmes numériques

Implémentation logicielle d'un fréquencemètre sur un HC11

Sommaire

I. Mesure de la période d'un signal.....	4
1. Première méthode : La méthode de polling ou de scrutation	4
l'horloge du CPU est celui du TCNT	4
Durée d'un processus	5
Mesure de la période.....	6
2. Deuxième méthode : Méthode optimisée par interruption	9
Interruption d'un programme.....	9
Conclusion (A FAIRE).....	11
II. Programmation de fonctionnalités d'entrée/sortie.....	12
Exemple d'appel de sous programme « courte » :.....	12
Exemple d'appel du sous programme « longue » :	13
Fonction réalisée par le programme	13
Affichage du contenu d'une variable sur l'écran LCD.....	16
Conclusion à faire	17

I. Mesure de la période d'un signal

Nous allons dans un premier temps étudier deux méthodes de programmation de mesure de fréquence : la méthode de polling et la méthode optimisée par interruption.

1. Première méthode : La méthode de polling ou de scrutation

Le programme scrute en permanence l'arrivée du premier front du signal puis du deuxième front de même nature, une fois ce front arrivé, on en déduit la période.

Avec cette méthode, nous allons devoir compter les fronts à l'aide d'un compteur 16 bits appelé TCNT contenu dans la minuterie du microcontrôleur 68HC11.

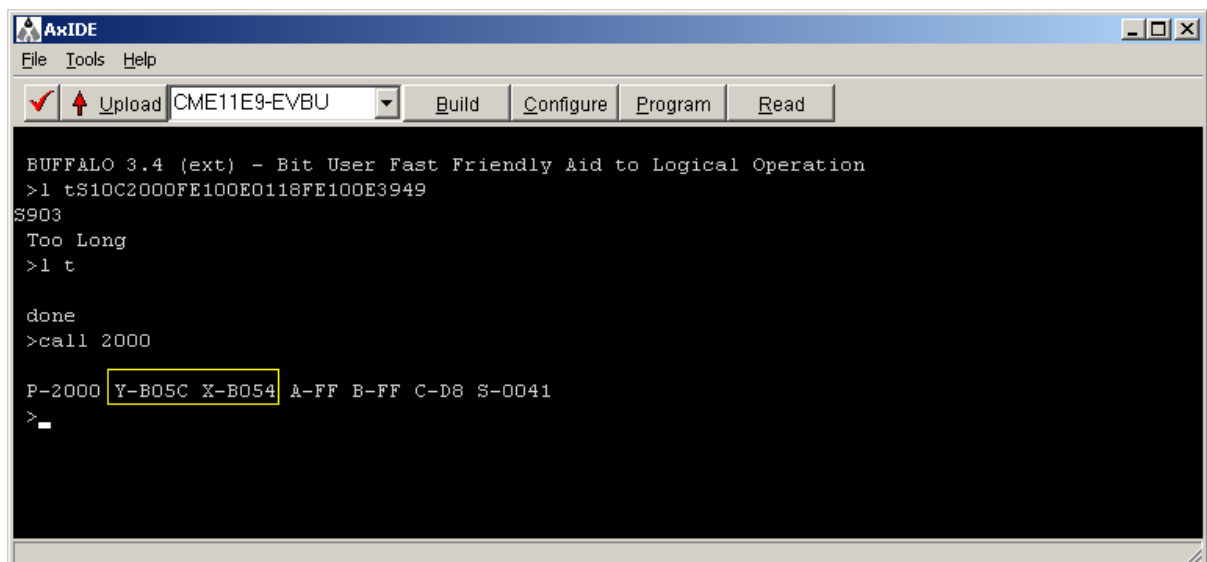
l'horloge du CPU est celui du TCNT

Tout d'abord, nous allons mesurer le nombre de cycles entre deux chargements. On fait commencer le programme ci dessous à l'adresse \$2000 (call 2000).

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	

Comparons ce résultat en réalisant la différence Y-X, on calcule le nombre de cycles s'écoulant entre les deux chargements.

Y-BO5C et X-BO54



Entre Y-BO5C et X-BO54, on a 8 fronts montants sur l'horloge du CPU (de 4 à C)

Il y a 8 cycles. Ainsi, le rapport entre le Timer Count Register (TCNT) et le CPU est de $8/8 = 1$

Ainsi, on remarque que le cycle du TCNT est égal au cycle du CPU. De plus, la durée d'un cycle est de 0,5 nanosecondes (voir doc du constructeur). Nous savons donc mesurer la durée d'une instruction.

Nous allons maintenant nous intéresser à la durée d'un processus.

Durée d'un processus

La durée d'un processus peut donc être mesurée au moyen d'un encadrement avec deux lectures du TCNT.

Le problème est que le 68HC11 ne prévoit pas de soustraction pour les registres X et Y. Cette opération ne reste possible que sur l'accumulateur D qui est la concaténation des registres A et B.

/* Il va donc falloir réaliser un petit programme qui va stocker la valeur du TCNT puis charger de nouveau la valeur du TCNT pour la comparer à la valeur précédente */

Assemblons le programme suivant afin de compter le nombre de cycles d'un processus quelconque.

0001 100e		TCNT equ	\$100E
0002 2000		org	\$2000
0003 2000		initial	rmb 2
0004 2002		delta	rmb 2
0005 2050		org	\$2050
0006 2050 fc 10 0e	[5]	ldd	TCNT
0007 2053 fd 20 00	[5]	std	initial
0008	10 cycles	Processus quelconque	
0009 2056 fc 10 0e	[5]	ldd	TCNT
0010 2059 b3 20 00	[6]	subd	initial
0011 205c fd 20 02	[5]	std	delta
0012 205f 39	[5]	rts	

Entre deux opérations de chargement 0007 et 0009 il y a 10 cycles machines (5 + 5).

Nous allons maintenant vérifier le contenu de la mémoire à l'adresse \$2002 après exécution du programme.

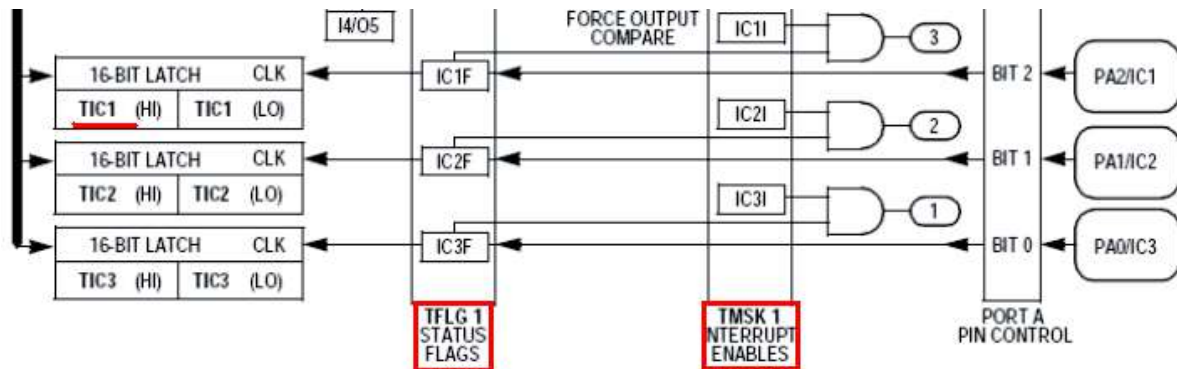
```
>md 2002
2000 B9 96 00 0A 18 FE 10 0E 39 FF DF 1F F9 3F EC FD          9  ?
2010 FF 3B 77 D9 F9 CF 5F BF 9B F5 6B 7F 7F E4 DD CB      ;w  _  k
2020 CD FB 7E 78 AD F7 FB BE F7 DE 7B FC F2 76 D9 FD      x      v
2030 BF FE EE E9 EC BF 93 D6 EA FF FA 3D ED 3A 3F AB              = :?
2040 95 33 17 E4 67 C7 D4 3F 56 F7 D6 BF 71 ED F7 EF      3  g  ?V  q
2050 FC 10 0E FD 20 00 FC 10 0E B3 20 00 FD 20 02 39              9
2060 E5 FE B3 E7 BF 7F DD FA FF F4 63 CF FD C8 FF FB              c
2070 CF 7F DF F7 DD BE 41 2D 2F E7 11 76 CF D7 FF DB      A-/  v
2080 67 98 ED BB BE BF 7D 94 DD EF EF E9 FF 6D FE 7F g              m
>
```

On obtient 10 cycles. On obtient bien le résultat désiré.

Enfin, l'erreur commise dans le calcul de la durée du processus quelconque est que le comptage a lieu entre les deux appels de TCNT (ldd TCNT), il ne faut donc pas prendre en compte le nombre de cycles pour l'instruction std initial (ici égal à 5).

Mesure de la période

Nous allons maintenant essayer de mesurer une période. Tout d'abord, notons les registres acteurs d'un scénario utilisant comme entrées du signal test PA2, PA1 ou encore PA0.



TIC1 Registre 16b	<table><tr><td>B₁₅</td><td>B₁₄</td><td>B₁₃</td><td>B₁₂</td><td>B₁₁</td><td>B₁₀</td><td>B₉</td><td>B₈</td><td>B₇</td><td>B₆</td><td>B₅</td><td>B₄</td><td>B₃</td><td>B₂</td><td>B₁</td><td>B₀</td></tr><tr><td colspan="8">MSB à l'adresse 1010</td><td colspan="8">LSB à l'adresse 1011</td></tr></table>	B ₁₅	B ₁₄	B ₁₃	B ₁₂	B ₁₁	B ₁₀	B ₉	B ₈	B ₇	B ₆	B ₅	B ₄	B ₃	B ₂	B ₁	B ₀	MSB à l'adresse 1010								LSB à l'adresse 1011								Adresse : \$1010-1011
B ₁₅	B ₁₄	B ₁₃	B ₁₂	B ₁₁	B ₁₀	B ₉	B ₈	B ₇	B ₆	B ₅	B ₄	B ₃	B ₂	B ₁	B ₀																			
MSB à l'adresse 1010								LSB à l'adresse 1011																										
TIC2 Registre 16b	<table><tr><td>B₁₅</td><td>B₁₄</td><td>B₁₃</td><td>B₁₂</td><td>B₁₁</td><td>B₁₀</td><td>B₉</td><td>B₈</td><td>B₇</td><td>B₆</td><td>B₅</td><td>B₄</td><td>B₃</td><td>B₂</td><td>B₁</td><td>B₀</td></tr><tr><td colspan="8">MSB à l'adresse 1012</td><td colspan="8">LSB à l'adresse 1013</td></tr></table>	B ₁₅	B ₁₄	B ₁₃	B ₁₂	B ₁₁	B ₁₀	B ₉	B ₈	B ₇	B ₆	B ₅	B ₄	B ₃	B ₂	B ₁	B ₀	MSB à l'adresse 1012								LSB à l'adresse 1013								Adresse : \$1012-1013
B ₁₅	B ₁₄	B ₁₃	B ₁₂	B ₁₁	B ₁₀	B ₉	B ₈	B ₇	B ₆	B ₅	B ₄	B ₃	B ₂	B ₁	B ₀																			
MSB à l'adresse 1012								LSB à l'adresse 1013																										
TIC3 Registre 16b	<table><tr><td>B₁₅</td><td>B₁₄</td><td>B₁₃</td><td>B₁₂</td><td>B₁₁</td><td>B₁₀</td><td>B₉</td><td>B₈</td><td>B₇</td><td>B₆</td><td>B₅</td><td>B₄</td><td>B₃</td><td>B₂</td><td>B₁</td><td>B₀</td></tr><tr><td colspan="8">MSB à l'adresse 1014</td><td colspan="8">LSB à l'adresse 1015</td></tr></table>	B ₁₅	B ₁₄	B ₁₃	B ₁₂	B ₁₁	B ₁₀	B ₉	B ₈	B ₇	B ₆	B ₅	B ₄	B ₃	B ₂	B ₁	B ₀	MSB à l'adresse 1014								LSB à l'adresse 1015								Adresse : \$1014-1015
B ₁₅	B ₁₄	B ₁₃	B ₁₂	B ₁₁	B ₁₀	B ₉	B ₈	B ₇	B ₆	B ₅	B ₄	B ₃	B ₂	B ₁	B ₀																			
MSB à l'adresse 1014								LSB à l'adresse 1015																										

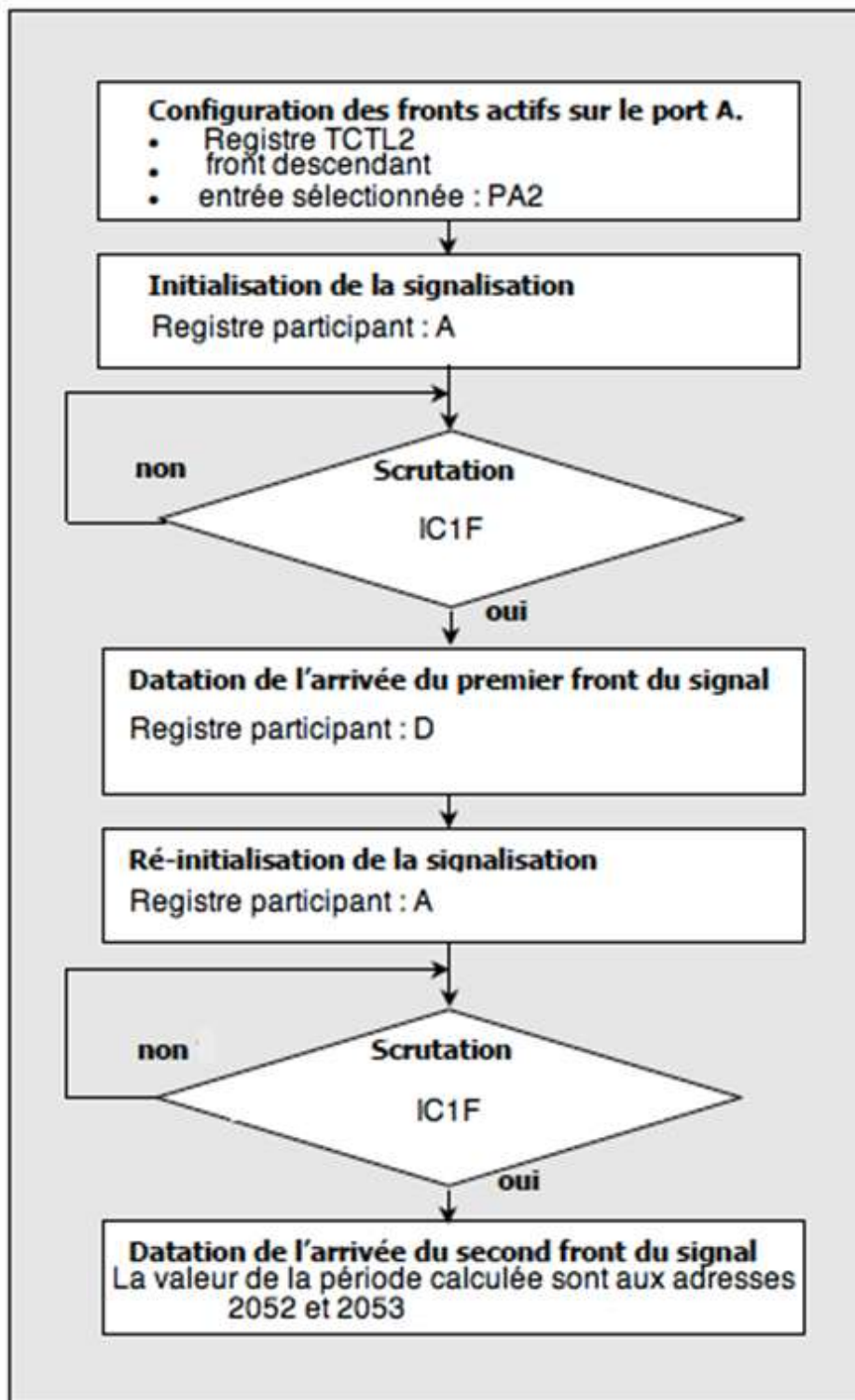
En se reportant au schéma fonctionnel de la minuterie :

Les registres acteurs d'un scénario utilisant PA2 sont les registres TFLG1 Status flags \$1023, TPMSK1 interrupt enables \$1022 et TIC1 aux adresses \$1010-1011

Les registres acteurs d'un scénario utilisant PA1 sont les registres TFLG1 Status flags, TPMSK1 interrupt enables et TIC2 aux adresses \$1012-1013

Les registres acteurs d'un scénario utilisant PA0 sont les registres TFLG1 Status flags, TPMSK1 interrupt enables et TIC3 aux adresses \$1014-1015

Organigramme



Relève des résultats

0001	1010	TIC1	equ	\$1010
0002	1023	TFLG1	equ	\$1023
0003	1021	TCTL2	equ	\$1021
0004				
0005	2050		org	\$2050
0006	2050	front	rrb	?

29 cycles machines séparent la fin des deux opérations de chargement dans l'accumulateur D.

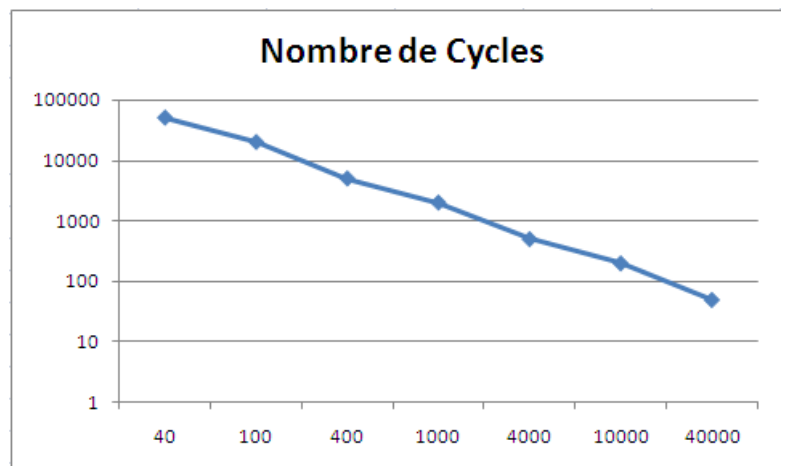
Nous allons maintenant exécuter le programme pour différentes valeurs de fréquences. Le tableau suivant contient le nombre de cycles entre deux fronts pour des fréquences allant de 40Hz à 40 kHz.

```

front2 front1 periode 07CF=AB0A - A33B
2050 A3 3B AB 0A D7 CF 10 00 DE 9E 18 29 88 87 8D A1
2060 D0 A9 AD 57 DB 7F 9A 6E 2C 82 12 EF 32 98 8B 9C
2070 50 F5 E8 07 54 95 7B F4 88 AC 06 9C BB 43 DB BE
2080 F5 12 3B 1E FD EE D9 C1 C6 FA 06 04 AC DA FE 9B
2090 B3 9E 3F C4 FC 66 1F 3C 3B 37 45 03 E3 1F 1C 3D
20A0 87 67 36 CE F8 24 77 64 3C 5F C6 84 D4 01 2F BA
20B0 D2 6D 06 AA 99 5D EF FC 93 EC D1 BD 8F 4E 6B 37
20C0 76 B9 3F FF 03 4C 79 FA 49 FE BE AA 56 D4 C4 4E
20D0 F1 BE 4E 74 87 EC DE CF 85 EC E7 BB AC EE 53 59
>

```

Fréquence des mesures (Hz)	Nombre de cycles (Polling)
40	50001
100	19905
400	4999
1000	2001
4000	500
10000	200
40000	50



Echelle log-log

Remarques :

Plus on augmente la période, plus la fréquence diminue mais le comptage augmente.
(Période plus grande, plus de nombre, plus à compter).

De plus, on sait qu'un cycle dur 0,5ms

Or $2001 * 0.5\text{ms} \approx 1000$ qui est la période qui correspond à 1kHz

Nos résultats sont donc cohérents.

Nous allons donc chercher à déterminer les fréquences minimale et maximale de fonctionnement correct du programme.

Nous savons qu'au minimum, le programme exécute 29 cycles, puisque un cycle prend 500ns (voir doc constructeur), on obtient le temps d'exécution minimal, on peut en déduire la fréquence maximale qu'on peut lui fournir en entrée :

Au minimum, on a 29 cycles :

$$f_{\max} = \frac{1}{T} = \frac{1}{29.1 \text{ cycle}} = \frac{1}{29 * 5.10^{-7}} = \mathbf{69 \text{ kHz}}$$

Au-delà de cette fréquence, le Timer calcule beaucoup trop lentement par rapport à la fréquence de mesure. Les valeurs sont erronées.

Au maximum, nous ne pourrions pas dépasser ($[FFFF]_{16} \cdot 1 \text{ cycle}$) puisque notre compteur est un registre de *16bits* : soit **65 535 cycles**.

Au maximum, on a $[FFFF]_{16}$ cycles :

$$f_{\min} = \frac{1}{T} = \frac{1}{FFFF \cdot 1 \text{ cycle}} = \frac{1}{65\,535 \cdot 5 \cdot 10^{-7}} \approx \mathbf{30 \text{ Hz}}$$

Au-dessous de cette valeur, la valeur codée sur 16bits est erronée.

En nous référant à nos mesures et notre graphique, nous constatons en effet qu'en dessous de 30Hz et au dessus de 70kHz, le programme ne fonctionne plus.

2. Deuxième méthode : Méthode optimisée par interruption

Le programme est interrompu par l'arrivée d'un premier front. La période du signal est déduite automatiquement à l'arrivée du second front de même nature.

Interruption d'un programme

Commençons par prendre en main les processus d'interruptions.

```

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

Cette boucle charge le contenu de l'adresse 0000, et la stocke dans l'adresse 2000.

```

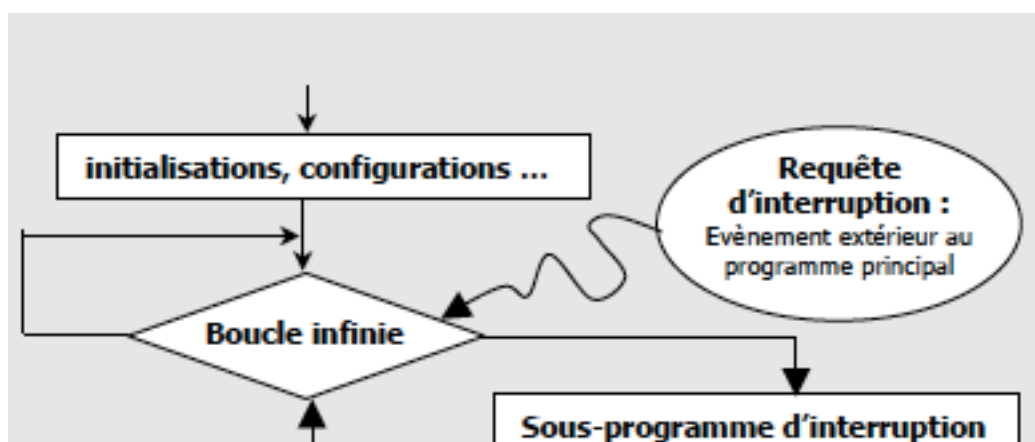
sei
lds      #$0200
ldab     ED1ba
stab     TCTL2
ldaa     TFLG1
anda     #$04
staa     TFLG1
ldaa     #$04
staa     TMSK1
cli
  
```

Les fonctions sei et cli sont des fonctions qui interdit / Autorise les interruptions.

Quand il y aura un signal en entrée (PA2), le programme va quitter la boucle (il sera interrompu) il y aura une écriture à l'adresse \$00E8. et cela va exécuter issprog (place ACDC à l'adresse \$2000).

```

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



Nous lançons ce programme et relevons de l'état de la mémoire aux adresses 2000 et 00E8.

Avant la compilation : On remarque qu'à 00E8 on obtient 7 E et à l'adresse 2000, on obtient une valeur aléatoire.

```
>md 2000
2000 7C FE E7 9D DD C7 D5 FA 7E 7E DF 2E B3 77 73 EE      . ws
2010 79 42 FD B9 17 F3 FD B0 7F 58 FF 3D CF 53 BA FE yB      X = S
2020 FE 67 EF 7B E5 BF 05 C7 FD EF DF FB 18 AF 8F 36 g      6
2030 F9 FF D3 47 67 5F 1E DF 8F 79 C4 0B F8 DB FC 7D      Gg_ y
2040 BA DC 71 9C 6E 5C 5E 78 BC 77 6A 9A 8E FB 4E 89 q n\^x wj N
2050 1E 2B 00 1D 10 FD 2E 68 DA 7B 3C C6 73 92 D7 BC + .h < s
2060 BB FC EC 7A 6E 6F 3C DF FE 7F A3 FC B3 DF FC B7 zno<
2070 75 CD ED 6F 67 BF 91 8F FE 2A 07 E9 13 51 76 B8 u og * Qv
2080 FF 17 D7 2E BF F7 A9 EF B1 7F A6 FD DA FF 8A DF .
>md 00E8
00E0 E3 71 7E E3 71 7E E3 71 7E E3 71 7E E3 71 7E E3 q q q q q
00F0 71 7E E3 71 7E E3 71 7E E3 71 7E E3 71 7E E3 71 q q q q q
0100 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
0120 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
0140 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
0160 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
>
```

Voici le contenu de la mémoire après compilation :

- **GBF éteint (Sans interruption):**

```
BUFFALO 3.4 (ext) - Bit User Fast Friendly Aid to Log
>md 2000      Chargement de #$0000 : Boucle
              effectuée sans interruption
2000 00 00 9C 51 EB FE FF 7E BF F8 53 FB 3E 37 9B A7
2010 1D 37 F9 EF 79 FF FF AD E5 E6 F1 FF 7F FB ED AB
2020 01 0F 5F 5B 0C 0E 53 41 7B FF FF FF FF FF FF FF
```

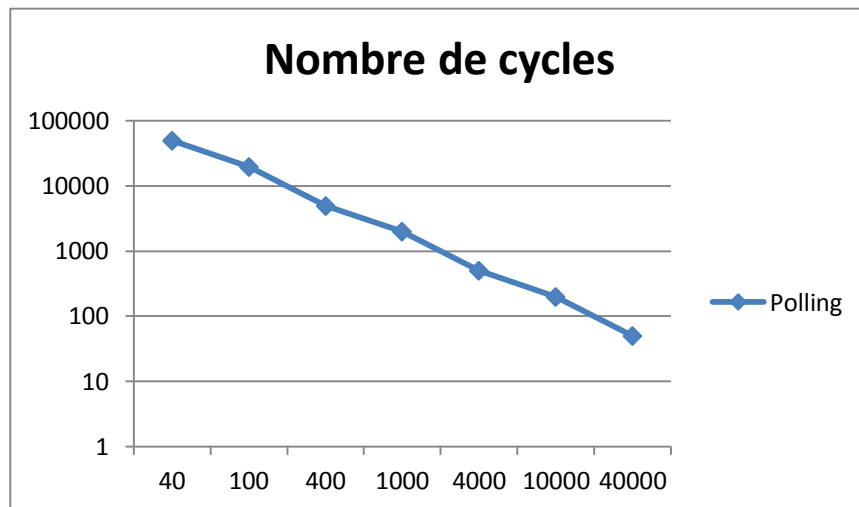
On voit bien qu'à l'adresse 2000 et 2001, on a 00 00. Il n'y a donc pas d'interruption.

- GBF allumé (Avec interruption):

```
>md 2000  Chargement de #ACDC : Boucle
          effectuée avec interruption
2000 AC DC 9C 51 EB FE FF 7E BF F8 53 FB 3E 37 9B A7
2010 1D 37 F9 EF 79 FF FF AD E5 E6 F1 FF 7F FB ED AB
```

A l'adresse 2000 et 2001, on a AC DC. On a donc fait une interruption

Fréquence (Hz)	Interruptions (base 10)	Interruption (base 16)
40	49980	c33c
100	19781	4d45
400	4980	1374
1000	2000	07d0
4000	499	01f3
10000	200	00c8
40000	50	32



On trouve une courbe similaire à la précédente.

Echelle log-log

----- voir limites -----

Conclusion (A FAIRE)

II. Programmation de fonctionnalités d'entrée/sortie

Jusqu'à présent nous avons effectués des manipulations uniquement au niveau matériel. Cela n'a que peu d'intérêt si ce matériel a pour but d'être inclus dans un système plus complexe. C'est pourquoi nous nous intéressons maintenant au contrôle de la maquette 68HC11 au niveau du pc, à l'aide d'un écran LCD affichant nos résultats dès lors que l'on exécute un programme.

Toutefois, comme pour tous les périphériques, l'écran LCD nécessite une initialisation avant d'être utilisé.

On constate que cette procédure fait appel à deux sous programmes nommés « longue » et « courte ».

```
***sous routine longue***
longue ldx    #50000
loop_l dex
      bne    loop_l
      rts

***sous routine courte***
courte ldx    #1000
loop_c dex
      bne    loop_c
      rts
```

Ces sous programmes servent en fait de temporisation nécessaire à la mise à jour de l'affichage. Il y'en a donc une qui est **courte**, et une qui est **longue**.

Les temporisations sont nécessaires pour laisser à l'écran LCD le temps d'initialiser ou rafraichir les différentes valeurs à afficher.

Exemple d'appel de sous programme « courte » :

La commande CINSHOF se trouvant à l'adresse \$06 utilise une temporisation courte, d'après le LCD command Codes

CINSHOF equ \$06

Code	Delay
\$01	1.65ms
\$02	1.65ms
\$04	40us
\$05	40us
\$06	40us

(Extrait du LCD command Codes)

En analysant le code du programme, nous constatons que l'utilisation du sous programme « courte » fait une décrémentation de la valeur **x = 10 000** jusqu'à ce qu'elle soit égale à 0.

courte ldx #1000

Exemple d'appel du sous programme « longue » :

La commande CSRHOM, qui est disposée à l'adresse \$02, utilise une temporisation longue, toujours d'après le LCD command Codes :

CSRHOM equ \$02

Code	Delay
\$01	1.65ms
\$02	1.65ms

(Extrait du LCD command Codes)

De même, le code du programme nous informe que l'utilisation du sous programme « longue » fait une décrémentation de la valeur **x=50000** jusqu'à ce qu'elle soit égale à 0 :

longue ldx #50000

Fonction réalisée par le programme

En exécutant le programme pas à pas nous observons que les différentes étapes de l'initialisation du programme d'affichage de l'écran LCD sont les suivantes :

Maintenant que nous avons analysés comment fonctionnait l'initialisation de l'écran LCD, nous allons nous intéresser à sa fonctionnalité première : l'affichage.

Pour l'affichage d'1 caractère, il faut envoyer le caractère à afficher à l'emplacement mémoire de LCDDAT puis ensuite temporiser le programme, le temps qu'il puisse effectuer l'affichage.

Tentons donc de rendre notre tout premier caractère, à la suite de l'initialisation, on insert le code suivant :

```
ldaa #'A'      * on garde le caractère 'A'
staa LCDDAT    * il s'agit de l'application d'un caractère
jsr courte     * puisque c'est un caractère
```

Nous allons maintenant tenter de produire un affichage correcte avec une phrase et, bien évidemment une place pour notre période que nous (ne perdons pas de vue) devons récupérer!

```
org $2000      * A partir de l'adresse $2000
texto fcc 'La periode =' * Réserver la place nécessaire en
                                * octets pour cette chaine de caractères
fin fcb #0      * Réserve d'octets multiples : ici pour 0
[...]
ldy #texto      * stockage de la chaine de caractères
chaine ldaa 0,y  * observation du registre Y
staa LCDDAT     * commande de données
jsr courte     * temporisation courte (caractères)
iny            * incrémente Y (caractère suivant)
bra chaine      * retourner à Y (tant que la fin n'est pas
[...]           * atteinte.
```

Conversion d'un entier en octet

Le plus simple est passé, fini de faire joujou avec l'écran, il va falloir l'afficher cette période! Et ce n'est pas si simple qu'on ne pourrait le penser : la période est stockée sous forme d'un octet (hexadécimal)! On sait que la période n'atteindra jamais plus de 2 octets...

Voici le –très long– programme nous permettant de convertir (cela vaut toutes les explications !) :

```
*** [Initialisation]
*** [Réservation de "texto"]

                org     $2020

table          fcc     '0'          *
              fcc     '1'          *  Réservation d'un table
              fcc     '2'          *  correspondant à tous
              fcc     '3'          *  les caractères affichables
              fcc     '4'          *  (en hexadécimal).
              fcc     '5'          *
              fcc     '6'          *  Chaque caractère est
              fcc     '7'          *  implanté de façon à ce
              fcc     '8'          *  que sa valeur corresponde
              fcc     '9'          *  à (l'adresse de départ) $2020
              fcc     'A'          *  PLUS sa valeur.
              fcc     'B'          *
              fcc     'C'          *  Exemple :
              fcc     'D'          *  Pour afficher "9", on se placera
              fcc     'E'          *  à l'adresse $2020 + 9
              fcc     'F'          *  = $2029 !

octet          fcb     #$36          *  à partir de l'adresse $2036, on
q_haut         rmb     1             *  réserve par exemple la valeur "36",
q_bas         rmb     1             *  1 octet pour le quartet bas et haut

                ORG     $2100          *
conv          ldx     #table          *  stockage valeur de la table : $2020
              ldaa    octet           *  stockage valeur de l'octet [36]16
              anda    #$0F           *  ET (0000 1111) [0F]16
              adda    0,x             *  donnera toujours q.bas : [06]16
              staa    q_bas           *  on additionne à $2020 : $2026 : '6'
                                      *  cela nous donne le caractère q.bas

              ldaa    octet           *  stockage valeur de l'octet [36]16
              asra    4              *  décalage 4 fois d'un bit :
              asra    4              *  36 = 0011 0110
              asra    4              *  0011 0110
              asra    4              *  0011 0110
              anda    #$0f           *  on obtient [03]16, ET [0F]16 = [03]16
              adda    0,x             *  on additionne à $2020 : $2023 : '3'
              staa    q_haut          *  cela nous donne le caractère q.haut

*** [Initialisation des commandes]

                ldy     #texto          *  on retient le texte
              jsr     chaine          *  on passe à la sous-routine "chaine"

*** [Sous-routine "longue"]
*** [Sous-routine "courte"]
*** [Sous-routine "chaine"]

suite         ldaa    q_haut          *
              staa    LCDDAT          *  affichage du quartet haut
              jsr     courte          *
```

```
ldaa q_bas      *  
staa LCDDAT     * affichage du quartet bas  
jsr  courte     *
```

Exemple : affichage de la lettre A

```
ldaa    #'A'
staa    LCDDAT
jsr     courte
```

- 1) On charge dans 'a', la lettre 'A'
- 2) On stocke le caractère contenu dans a dans LCDDAT
- 3) On temporise

Nous savons désormais comment afficher sur l'écran LCD, 1 caractère. L'intérêt est ici limité, aussi, nous allons voir comment afficher une chaîne de caractère.

Pour cela, nous devons utiliser la commande fcc qui alloue de la mémoire permettant d'écrire une chaîne de caractère dans une variable. Il est ainsi possible de la parcourir grâce à l'adresse indexée.

Exemple d'affichage de chaîne de caractère:

Nous voulons afficher « la periode= »

Voici le code nécessaire à l'affichage de cette chaîne de caractère :

```
texto    fcc    'la periode='
          ldy    #texto
chaîne    ldaa    0,y
```

- 1) On stocke la chaîne de caractère «la periode= » dans 'texto'
- 2) On charge cette chaîne de caractère dans 'y'
- 3) L'accumulateur 'a' reçoit le premier caractère de y[0].

Il est nécessaire de boucler le programme pour tout afficher.

Soit le code complet :

```

texto    org    $2000          *Le programme commence à partir de l'adresse $2000
fin       fcc    'La periode=' *On alloue une place mémoire en octet égale à  cette chaîne de char
          fcb    #0

          ORG    $2100          *L'initialisation débute à l'adresse $2100
//initialisation du programme
//...
//Fin d'initialisation

chaîne    ldy    #texto          *On stocke dans 'y' la chaîne de caractère stockée dans 'texto'
          ldaa    0,y            *On stocke dans 'a' le premier caractère de y[0]
          beq     suite          *On saute à 'suite' s'il n'y a rien dans 'a' (donc qu'il n'y a pas
                                *ou plus de caractère)
          staa    LCDDAT          *On copie la valeur contenue dans 'a' dans LCDDAT
          jsr     courte          *Saut long à la sous routine "courte"
          iny     *Incrémentation de 'y'
          bra     chaîne          *On retourne à 'chaîne', pour afficher le reste des caractères&

//Affichage|
```

Exemple : Stockons une valeur dans une variable et affichons la à la suite de la chaîne de caractère « la période= ».

Le problème ici rencontré est que les valeurs contenues dans les variables sont stockées sous forme d'octets dans les accumulateurs, les valeurs écrites en mémoire étant en hexadécimale. Ces accumulateurs ne pourront donc stocker qu'un nombre maximal de 2^8 bits soit en Hexadécimal, un nombre compris entre 00 et FF.

On est donc limité ici à afficher des chiffres n'excédant pas la dizaine (00 à 99) **(A vérifier)**.

Il va donc falloir séparer les valeurs en deux, respectivement en le premier chiffre et deuxième chiffre du nombre.

suite	ldaa	q haut
	staa	LCDDAT
	jsr	courte
	ldaa	q bas
	staa	LCDDAT
	jsr	courte

Nous avons vu comment effectuer une conversion d'un entier sur 1 octet, ce qui est relativement limité au niveau de l'affichage.

Passons à la conversion d'un entier sur 2 octets

L'affichage d'un nombre sur 1 octet passe, comme on l'a vu précédemment par un nombre assez conséquent d'étapes, alors qu'il s'agit d'un affichage relativement basique. L'affichage d'un nombre sur 2 octets est encore plus compliqué si l'on essaie d'appliquer telle quelle la méthode d'affichage sur un octet sur l'affichage sur 2 octets.

Pour l'affichage sur 2 octets, il suffit en fait, de reprendre la même méthode mais de stocker la valeur dans 2 variables différentes au lieu d'une. La première partie du nombre dans la variable 1, et la deuxième partie, dans la variable 2.

Conclusion à faire