

Systemes Numériques

Implémentation logicielle d'un fréquencesmètre
sur un HC11

2015

Sommaire

Introduction.....	2
I. Mesure de la période d'un signal	3
A. Le Polling	3
B. Mesure de la période	5
C. Les interruptions	8
II. Programmation d'un LCD	10
A. Initialisation	10
B. Notre premier affichage	11
C. Affichage plus complexe.....	12
D. Conversion d'un entier en octet pour l'affichage	12
III. Communication par liaison série asynchrone	14
A. Communication avec le port série en mode Polling.....	14
B. Communication avec le port série en mode Interruptions	15

Introduction

Tout au long de ce projet, nous allons travailler sur le HC11 qui est un microcontrôleur développé par Motorola. Nous allons nous servir de ce microcontrôleur à la fois pour apprendre à utiliser l'assembleur que nous avons vu rapidement en Architecture des Ordinateurs, mais également à l'appliquer dans des cas réels.

L'objectif final de notre projet est l'implémentation logicielle d'un fréquencemètre sur un 68HC11. Pour cela, nous allons étudier 3 éléments importants qui nous permettront d'effectuer ce fréquencemètre : La mesure de la période d'un signal, la programmation de fonctionnalités d'entrée / sortie et la communication par liaison série asynchrone.

Nous allons ainsi travailler sur 2 méthodes différentes de programmation en Assembleur :

- **La méthode de Polling** : Elle consiste à scruter l'arrivée du premier front du signal, puis du second de même nature, pour en déduire sa période,
- **La méthode par Interruption** : Le programme principal s'interrompt à l'arrivée du premier front, ce qui lui permet de déduire la période du signal à l'arrivée du second front de même nature.

Nous allons également avoir besoin d'un périphérique LCD qui nous permettra d'afficher nos résultats, et apprendrons donc faire communiquer le microcontrôleur et cet écran. Pour finir, nous verrons comment effectuer un système de saisie de caractères qui seront transférés et affichés sur l'écran LCD.

I. Mesure de la période d'un signal

La partie mesure de la période d'un signal est très importante dans l'optique de l'objectif final de notre projet, à savoir : Implémenter un fréquencemètre sur un HC11. Pour cela, nous allons utiliser la minuterie du microcontrôleur, le 68HC11. Nous disposerons en effet d'un compteur de 16 bits (Le TCNT) qui nous permettra de connaître la période du signal grâce au nombre de cycles machines écoulés durant le temps voulu en étudiant les dates d'arrivées de fronts.

Nous allons pouvoir compter ces arrivées sur les fronts grâce à la méthode de Polling.

La méthode de Polling est assez simple : Le programme attend, note le temps d'arrivée sur le premier front, et celui d'arrivée sur le second front. On soustrait ces 2 données, on obtient alors la période, et ainsi la fréquence.

A. Le Polling

Pour comprendre précisément le fonctionnement, nous allons l'étudier avec divers exemples. En tout premier lieu, prenons les comptages de cycles CPU et Timer. Nous utiliserons le code assembleur fourni dans le sujet du projet :

```
TCNT equ $100E
      org $2000
      ldx TCNT
      nop
      ldy TCNT
      rts
```

Nous obtenons alors la sortie suivante :

0005	2000	fe 10 0e	[5]	ldx	TCNT
0006	2003	01	[2]	nop	
0007	2004	18 fe 10 0e	[6]	ldy	TCNT

On compte alors le nombre de cycles séparant la fin des 2 chargements ldx et ldy :

$$2 + 6 = 8 \text{ cycles}$$

On vérifie ensuite ces données en étudiant les états des registres X et Y. Pour cela on lance le programme, on a alors cette sortie :

```
>CALL 2000
P-2000 Y-458B X-4583 A-FF B-FF C-DO S-0041
```

On a donc :

- Dans le registre X se trouve le temps du premier front,
- Dans le registre Y celui du second front.

La différence est alors assez simple à effectuer :

Contenu du registre à Y	458B
Contenu du registre à X	4583
Y – X	0008

Ce résultat justifie bien le calcul théorique qu'on a effectué précédemment.

Nous allons maintenant mesurer la durée d'un processus en l'encadrant de 2 lectures du TCNT. Nous effectuons la même procédure que précédemment, nous assemblons le programme et obtenons les sorties suivantes :

```
0009 2050 fc 10 0e      [ 5 ]      ldd    TCNT
0010 2053 fd 20 00      [ 5 ]      std    initial
0011                      |
0012 2056 fc 10 0e      [ 5 ]      ldd    TCNT
0013 2059 b3 20 00      [ 6 ]      subd   initial
0014 205c fd 20 02      [ 5 ]      std    delta
0015
0016 205f 39            [ 5 ]      rts
```

Le nombre de cycle qui séparent les 2 opérations de chargement est de $5 + 5 = 10$ cycles.

```
>call 2050
P-2050 Y-458B X-4583 A-00 B-0A C-D0 S-0041
```

On sait qu'aux adresses \$2002 et \$2004 on a les valeurs des registres A et B respectivement.

Les registres contiennent les valeurs :

Pour A	00
Pour B	0A

On cherche le contenu de la mémoire à l'adresse \$2002 après exécution du programme.

```
>md 2002
2000 47 19 00 0A 18 AC 70 04 37 F9 1F 7A 4B D8 46 34 G      p 7  zK F4
```

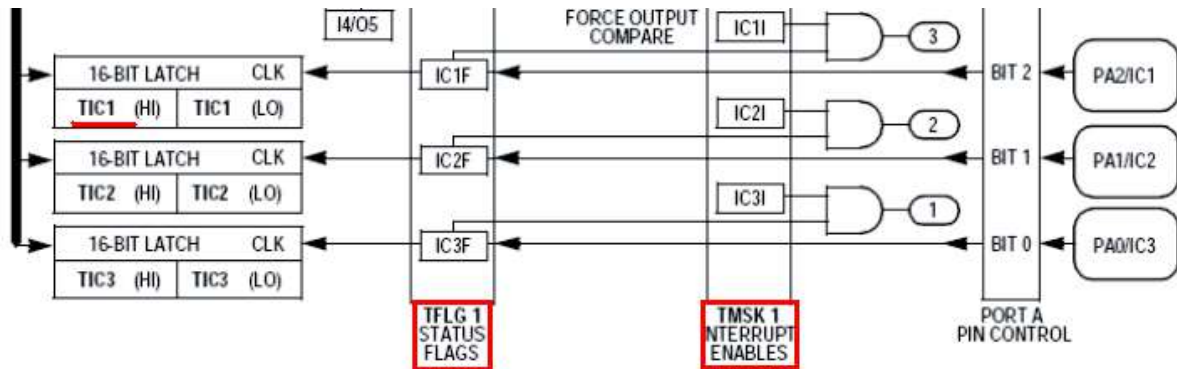
On obtient également nos 10 cycles, le résultat est bien celui désiré.

Toutefois, il y a une erreur qui est commise dans le calcul de la durée du processus : Le comptage a lieu entre les 2 appels de TCNT, il ne faut donc pas prendre en compte les cycles pour l'instruction std initial, qui n'intervient pas dans l'opération que nous cherchons à calculer.

B. Mesure de la période

Nous allons désormais mesurer une période. Pour cela nous commençons par comprendre quels sont les registres acteurs d'un scénario qui va utiliser comme entrées du signal : PA2, PA1 ou PA0.

En nous servant du schéma donné, nous obtenons ceci :



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																										

On peut donc en déduire que les acteurs d'un scénario utilisant :

- **PA2 :**
 - TFLG1 Status flags : \$1023
 - TPMSK1 interrupt enables : \$1022
 - TIC1 : \$1010 et \$1011
- **PA1 :**
 - TFLG1 Status flags : \$1023
 - TPMSK1 interrupt enables : \$1022
 - TIC2 : \$1012 et \$1013
- **PA0 :**
 - TFLG1 Status flags : \$1023
 - TPMSK1 interrupt enables : \$1022
 - TIC3 : \$1014 et \$1015

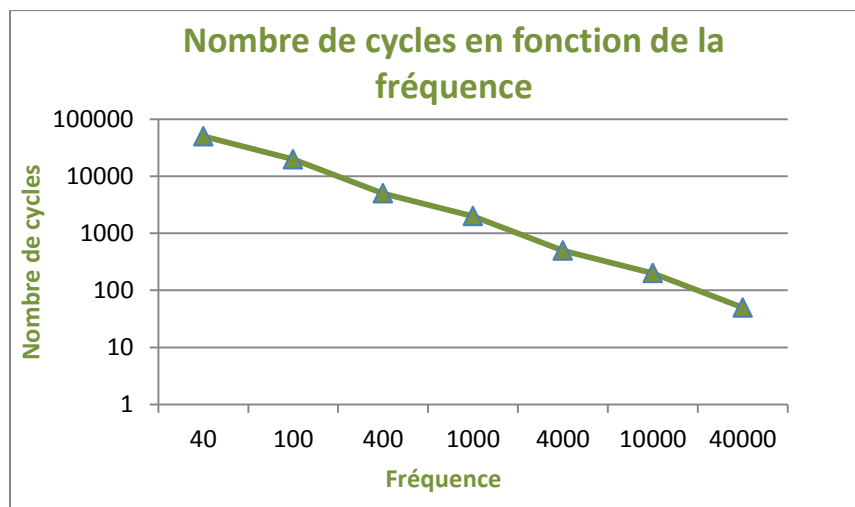
On utilise le programme suivant :

TIC1 equ	\$1010	On définit TIC1 à l'adresse 1010
TFLG1 equ	\$1023	On définit TFLG1 à l'adresse 1023
TCTL2 equ	\$1021	On définit TCTL2 à l'adresse 1021
org	\$2050	On se place à l'adresse \$2050
front rmb	2	On réserve 2 octets pour le front
period rmb	2	On réserve 2 octets pour la période
ED1ba fcb	\$10	On définit le front descendant
org	\$2100	Début du programme à l'adresse \$2100
ldab ED1ba		Initialisation de ED1ba
stab TCTL2		On stocke dans le registre
ldaa TFLG1		Initialisation de TFLG1
anda #\$04		ET Logique entre la valeur de l'accumulateur A
staa TFLG1		et la valeur 04 ; TFLG1 devient un temps
lire1 ldaa	TFLG1	
anda	#\$04	
beq	lire1	Si égal, aller à lire1
suite1 ldd		On stocke le front
TIC1		
std	front	
ldaa TFLG1		
anda	#\$04	On réinitialise le temps
staa	TFLG1	
lire2 ldaa	TFLG1	
anda	#\$04	
beq	lire2	
suite2 ldd		On a le front actuel
TIC1		On soustrait le front stocké
subd	front	On obtient la période
std	period	
rts		

Suite à l'assemblage du programme, on peut voir qu'il y a 29 cycles machines qui séparent la fin des opérations de chargement dans l'accumulateur D.

On utilise alors ce programme en faisant varier notre fréquence de 50 Hz à 50 kHz. On obtient les résultats contenus dans le tableau suivant :

Fréquence	Polling (Base 16)	Polling (Base 10)
50 Hz	9C4F	40015
100 Hz	432A	17194
400 Hz	1389	5001
1 kHz	07D1	2001
4 kHz	01F4	500
10 kHz	00C8	216
40 kHz	0032	50
50 kHz	0028	40



On remarque dans cette expérience que plus la période augmente, plus la fréquence diminue. Toutefois, le comptage augmente aussi !

Nous devons maintenant déterminer les fréquences minimales et maximales permettant le fonctionnement correct de notre programme.

Nous avons précédemment vu que le programme effectue au minimum 29 cycles, prenant chacun 500 ns comme le précise la documentation. On calcule alors la fréquence maximale et on obtient :

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

Passé cette fréquence, le Timer calcule trop lentement par rapport à la fréquence de mesure, les résultats seront donc faux.

Pour calculer le maximum, rappelons-nous que nous utilisons un compteur qui est un registre de 16 bits, on a donc 65 535 cycles maximum. On peut, de la même façon que précédemment, obtenir la fréquence minimale :

$$f_{min} = \frac{1}{T} = \frac{1}{65535 * 5.10^{-7}} = 30 \text{ Hz}$$

C. Les interruptions

Il existe une autre méthode différente de celle du Polling, c'est la méthode de l'interruption.

Avec cette méthode, le programme est interrompu lorsque le premier front arrive. La période du signal est alors automatiquement déduite à l'arrivée du second front de même nature.

Commençons par décrire le phénomène de boucle.

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

Cette boucle va placer 0000 à l'adresse \$2000.

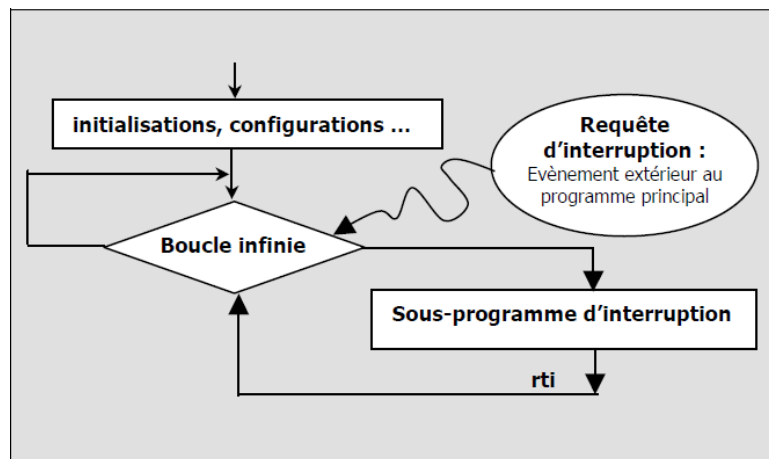
On a ensuite deux nouvelles instructions :

- **sei** : Instruction qui interdit les interruptions
- **cli** : Instruction qui autorise les interruptions

Le programme va fonctionner de la manière suivante : A l'arrivée d'un signal d'entrée (PA2), le programme quitte la boucle, c'est l'interruption, on a alors l'écriture à l'adresse \$00E8 et l'exécution d'issprog, qui place ACDC à l'adresse \$2000 :

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

Le programme est représenté sous la forme d'un organigramme:



Passons maintenant à la pratique. Nous lançons le programme, et nous allons relever l'état de la mémoire aux adresses 2000 et 00E8.

(Note de Florian LEPOT : Mon binôme n'ayant pas été présent à la séance, j'ai travaillé pour la suite avec Aurélien DAVID et Adrien RAMOS comme convenu avec vous durant la séance, aussi, j'aurai pour la suite, les mêmes relevés qu'eux).

Relevés initiaux :

>md 2000

```
2000 F2 55 18 F7 F9 E6 EF F9 B2 E4 7B AB B2 EF E3 63 U c
2010 E6 77 BE AA F1 E1 FE 64 CF 77 B2 DE AE D4 76 6F w d w vo
2020 BB BB 79 C7 BF FB FB E4 FF 37 CF DF F8 F3 F5 7A y 7 z
2030 BC 5F 74 6D F3 96 5B F7 E7 DA D1 99 FF 74 CF 77 _tm [ t w
2040 6C CD 45 19 BE FF 6E 15 FO DD BF 3C EF B8 C4 EE l E n <
2050 B7 FD 3B FF 10 EE DF 41 77 76 CB 4F EO FA 7F 57 AwvwO W
2060 7D D7 FC B2 7D 6D ED 7E 9A E6 69 44 96 FF F3 B6 m F
2070 79 2E 74 EF 73 DB FB 9F 1D 79 FC F2 7D ED 73 7A y. s y sz
2080 89 7A F8 58 7B 5E F5 41 F7 BA FE 57 75 EO BC DB z Z ^ A Wu
>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 E3 71 7E E3 71 7E E3 71 7E E3 71 q q q q 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
```

Relevés en cas d'absence du signal :

>md 2000

```
2000 25 00 00 FF FF 00 FA 31 25 00 00 FF FF 00 FA 31 % 1% 1
2010 25 00 00 FF FF 00 FA 31 25 00 00 FF FF 00 FA 31 % 1% 1
2020 25 00 00 FF FF 00 FA 31 25 00 00 FF FF 00 FA 31 % 1% 1
2030 25 00 00 FF FF 00 FA 31 25 00 00 FF FF 00 FA 31 % 1% 1
2040 25 00 00 FF FF 00 FA 31 25 00 00 FF FF 00 FA 31 % 1% 1
2050 25 00 00 FF FF 00 FA 31 25 00 00 FF FF 00 FA 31 % 1% 1
2060 25 00 00 FF FF 00 FA 31 25 00 00 FF FF 00 FA 31 % 1% 1
2070 25 00 00 FF FF 00 FA 31 25 00 00 FF FF 00 FA 31 % 1% 1
2080 25 00 00 FF FF 00 FA 31 25 00 00 FF FF 00 FA 31 % 1% 1
```

Relevés en cas de présence du signal :

>md 2000

```
2000 AC DC 00 FF FF 00 FA 31 25 00 00 FF FF 00 FA 31 1% 1
2010 25 00 00 FF FF 00 FA 31 25 00 00 FF FF 00 FA 31 % 1% 1
2020 25 00 00 FF FF 00 FA 31 25 00 00 FF FF 00 FA 31 % 1% 1
2030 25 00 00 FF FF 00 FA 31 25 00 00 FF FF 00 FA 31 % 1% 1
2040 25 00 00 FF FF 00 FA 31 25 00 00 FF FF 00 FA 31 % 1% 1
2050 25 00 00 FF 10 00 FA 31 25 00 00 FF FF 00 FA 31 % 1% 1
2060 25 00 00 FF FF 00 FA 31 25 00 00 FF FF 00 FA 31 % 1% 1
2070 25 00 00 FF FF 00 FA 31 25 00 00 FF FF 00 FA 31 % 1% 1
2080 25 00 00 FF FF 00 FA 31 25 00 00 FF FF 00 FA 31 % 1% 1
```

On note bien lorsqu'on a un signal, qu'à l'adresse \$2000 et \$2001 on a pour valeur AC et DC. Il y a donc eu interruption, le programme a bien fonctionné.

II. Programmation d'un LCD

Dans la première partie nous avons appris à compter les cycles d'un programme, l'interrompre, et en déduire la période.

Nous voulons maintenant coupler le 68HC11 avec un écran LCD pour afficher les résultats lorsqu'on exécute un programme. Durant toute cette partie nous allons donc étudier le fonctionnement d'un LCD pour pouvoir l'utiliser dans un objectif plus complexe.

A. Initialisation

La première chose à faire est d'initialiser le périphérique LCD, pour cela nous disposons d'un programme. En étudiant ce programme on peut voir que l'on a 2 sous programmes nommés « courte » et « longue ». Ces 2 sous programmes sont des sous routines qui vont nous servir en fonction des registres utilisés.

Pour être plus précis, on a 2 types de commandes :

- « Clear Display, Cursor to Home » et « Cursor to Home » qui ont un délai de 1,65ms
- Toutes les autres, avec un délai de 40µs.

Pour cela, nous avons donc besoin de 2 sous programmes de temporisation de durée différente, on a donc la temporisation longue qui va être appelée pour les commandes ayant un délai de 1,65ms, et la courte qui est appelée pour les commandes avec un délai de 40µs.

Nous exécutons alors le programme donné pas à pas pour observer les différentes étapes de l'initialisation du programme d'affichage de l'écran LCD.

LCDCMD	equ	\$b5f0	*
LCDDAT	equ	\$b5f1	*
L4xC20	equ	\$3c * 4x20	*
CLRCH	equ	\$01	*
DCBON	equ	\$0f	* On définit tous les labels
CINSHOF	equ	\$06	*
SCSHR	equ	\$14	*
CSRHOM	equ	\$02	*
ORG	\$2100		* Début du programme à \$2100
init	ldaa	#L4xC20	*On définit la taille
	staa	LCDCMD	* de l'écran LCD
	jsr	longue	* (4x20)
	ldaa	#CLRCH	*
	staa	LCDCMD	* On le clear
	jsr	longue	*
	ldaa	#DCBON	*
	staa	LCDCMD	* On désactive le clignotement
	jsr	courte	*

ldaa #CINSHOF	*
staa LCDCMD	* Shit Off
jsr courte	*
ldaa #SCSHR	*
staa LCDCMD	* Shift On
jsr courte	*
ldaa #CSRHOM	*
staa LCDCMD	* On replace le curseur au début
jsr courte	*
* longue ldx #50000	
loop_l dex	
bne loop_l	
rts	
* courte ldx #1000	
loop_c dex	
bne loop_c	
rts	

B. Notre premier affichage

Maintenant que l'écran est initialisé, nous pouvons commencer à entrer dans le concret ! Pour cela, nous allons afficher les caractères A, B, C et D sur l'écran.

Nous utilisons alors le bout de programme pour afficher un caractère, et on le répète pour en afficher d'autres différents :

suite ldaa #'A'	*
staa LCDDAT	* On affiche le caractère A
jsr courte	*

On assemble le programme, on l'exécute, ça y est ! Notre écran LCD nous affiche bien ABCD.

C. Affichage plus complexe

Maintenant que nous avons affiché des caractères un par un, nous allons voir comment afficher une chaîne de caractères.

Pour cela, on utilise une nouvelle directive : fcc. Celle-ci nous permet de déclarer une chaîne de caractères de type constante.

texto fcc 'La periode ='	* On initialise la chaîne de caractères
fin fcb #0	* Marqueur de fin de chaîne de caractères
(...)	
ldy #texto	* On charge le registre Y avec l'adresse
	* du premier caractère de texto
Chaine ldaa 0,y	* On charge l'accumulateur A de façon
	indirecte (Adresse contenue dans Y + 0)
beq suite	* Branche à suite si l'accumulateur A
	contient le caractère null (Chaîne
	transférée entièrement vers le LCD)
staa LCDDAT	* Sinon, on écrit le contenu dans les
	datas du LCD
jsr courte	* Délai
iny	* On incrémente Y (Caractère suivant de
	texto)
bra chaine	* Branche à chaîne pour charger le
(...)	caractère suivant

En assemblant et en exécutant le programme, on affiche bien sur l'écran LCD : « La periode = ».

L'affichage d'une chaîne de caractère fonctionne donc parfaitement !

D. Conversion d'un entier en octet pour l'affichage

Pour pouvoir afficher notre période, qui est stockée sous la forme d'un octet (2 octets maximum), il va donc falloir la convertir.

On va commencer par réserver un tableau correspondant à tous les caractères affichables en hexadécimal (donc de 0 à 9 et de A à F). On a donc des données de \$2020 à \$2035.

On utilise donc le programme suivant, que l'on détaille :

Conv ldx #table	* Charge le registre X avec l'adresse du tableau
	de caractères
Ldaa octet	* Octet de test = \$36 en hexa dans
	l'accumulateur A
Anda #\$0F	* On fait un AND (immédiat (#)) pour mettre à 0
	les 4 bits de poids fort : \$0F = 00001111 ₂

Adda 0,X	* On ajoute à cette valeur le premier caractère du tableau (dont l'adresse est dans X) cela doit donner '6'
Staa q_bas	* Le résultat est stocké dans l'octet réservé à l'affichage du quartet bas
Ldaa octet	* On charge de nouveau A avec \$36
Asra Asra Asra Asra	* Décalage de 1 bit vers la droite * Décalage de 4 bits vers la droite, maintenant il y a \$03 dans A
Anda #\$0F	* Au cas où les poids forts seraient différents de 0 (lors du décalage le bit de poids le plus fort reste dans le bit de poids 7)
Adda 0,X	* On ajoute au résultat la valeur du contenu du premier octet du tableau. Il y a maintenant '3' dans A
Staa q_haut	* Le résultat est stocké dans l'octet réservé à l'affichage du quartet haut

La conversion est désormais faite, on pourrait donc l'afficher sur notre écran LCD !

III. Communication par liaison série asynchrone

Nous savons désormais utiliser le Timer et l'écran LCD, nous allons pouvoir passer au déclenchement de l'affichage au moment où nous avons la saisie d'un caractère. On va donc utiliser la liaison série SCI pour communiquer avec le processeur.

A. Communication avec le port série en mode Polling

Nous allons commencer par noter la vitesse de la communication. Pour cela, nous prenons la valeur que nous voulons charger dans l'accumulateur (Ici 30 en hexadécimal).

On convertit cette valeur en binaire, et on étudie les correspondances avec le « BAUD Rate Control Register » de la documentation du HC11, ce qui nous donne pour 30 :

$$30 = 0110000|_2$$

Baud Rate Control Register

TCLR	0	SCP1	SCP0	RCKB	SCR2	SCR1	SCR0
0	0	1	1	0	0	0	0

On obtient SCP0 et SCP1 pour la valeur entrée, on peut ainsi aller chercher dans le BAUD Rate Prescale Selects de la documentation, la vitesse de communication recherchée.

Table 9-1 Baud-Rate Prescale Selects

SCP1	SCP0	Division Factor	Crystal Frequency				
			2 ²³ Hz	8 MHz	4.9152 MHz	4 MHz	3.6864 MH
			Highest Baud Rate				
0	0	1	131.072K Baud	125.000K Baud	76.80K Baud	62.50K Baud	57.60K Baud
0	1	3	43.691K Baud	41.667K Baud	25.60K Baud	20.833K Baud	19.20K Baud
1	0	4	32.768K Baud	31.250K Baud	19.20K Baud	15.625K Baud	14.40K Baud
1	1	13	10.082K Baud	9600 Baud	5.908K Baud	4800 Baud	4431K Baud
			2.1 MHz	2 MHz	1.2288 MHz	1 MHz	921.6 kHz
Bus Frequency (E clock)							

Comme vu plus haut, pour 30, SCP1 et SCP0 sont à 1, on est donc dans la 4eme ligne du tableau. Par ailleurs, on sait que nous sommes à 8 MHz dans notre cas, nous obtenons donc une vitesse de communication de 9600 Baud.

Ce mode de programmation a cependant un désavantage notable et important. En effet, en mode Polling, le programme ne fait rien d'autre que d'attendre une donnée. Le CPU est donc utilisé à 100% pour vérifier l'arrivée d'une donnée potentielle. A 9600 Baud, nous sommes à un peu moins de 960 données par seconde (Car sur 10 bits, nous avons 1 bit de start, 8 bits de data et 1 bit de stop), ce qui nous fait beaucoup de pertes pour un CPU tournant à 8 MHz.

Passons maintenant à la pratique, avec le programme fournit, que nous complétons, nous obtenons ceci :

Clr	SCCR1	* Reset du registre de contrôle de com SCI (\$102C) 8 bits de data, 1 de start, 1 de stop
Ldaa	#\$30	* Charge l'accumulateur A avec \$30
Staa	BAUD	* Stocke le contenu de A dans le registre BAUD (\$102B). SCP0 et SCP1 à « 1 ». Donc prédivison par 13, soit une vitesse de transmission de 9600 Baud pour une fréquence de 8 MHz.
Ldab	#\$0C	* Charge l'accumulateur B avec la valeur hexa \$0C
Stab	SCCR2	* Stocke le contenu de B dans le registre SCCR2 (\$102D). Emission et réception activées (bits TE et RE à 1), pas d'interruption (bits TIE et RIE à 0)
Clr	\$2000	* Efface l'adresse mémoire \$2000.
Loop	ldaa	SCSR
	Anda	#\$20
	Beq	loop
	Ldaa	SCDR
	Staa	\$2000

On assemble le programme, qui va créer une boucle infinie en attendant de recevoir une donnée. On appuie alors sur « b ». Rien ne se passe visuellement évidemment puisqu'on n'a aucune sortie de boucle, toutefois, si on coupe le programme et qu'on vérifie la mémoire à l'adresse \$2000, on obtient « 62 », ce qui correspond au caractère ASCII de « b ».

On a donc notre saisie qui fonctionne et qui est enregistrée en mémoire.

B. Communication avec le port série en mode Interruptions

Le vecteur d'interruption se situe aux adresses \$00C4 et \$00C6. On l'implémente donc dans le programme final en mode d'interruptions, et on obtient le programme suivant :

Org	\$00C4	* Saut à la routine d'interruption
Jmp	issprog	
Org	\$2200	
Sei		* Interdit les interruptions
Clr	SCCR1	* 8 bits de datas + 1 start et 1 stop
Clr	SCSR	* Clear le registre d'état de la liaison SCI
Ldaa	\$30	* 0011 0000 ₂
Staa	BAUD	

Ldaa #2C	*0010 1100 ₂
Staa SCCR2	*Emission Réception activées, mais Interruption uniquement sur Réception
Clr \$2000	* Efface la donnée en mémoire adresse \$2000
Cli	*Autorise les interruptions
Issprog clr SCSR	* Clear l'interruption
Ldab SCDR	* Charge le registre B avec la donnée reçue dans le registre Réception data de la liaison SCI
Stab \$2000	* Et la stocke en adresse mémoire \$2000

Durant la dernière séance, après avoir assemblé et exécuté le programme, nous avons pu remarquer que les caractères saisis étaient enregistrés dans la mémoire, toutefois, l'affichage sur l'écran LCD ne fonctionnait pas.

Sur le moment, nous n'avions pas eu le temps de corriger cela, toutefois après réflexions, cela paraît normal puisque nous n'avons pas chargé le registre LCDDAT avec les données saisies, et de ce fait, cela nous pouvait les afficher. En ajoutant donc ces données également dans le registre LCDDAT, l'affichage sur l'écran LCD devrait fonctionner, ainsi les caractères saisis s'afficheraient.