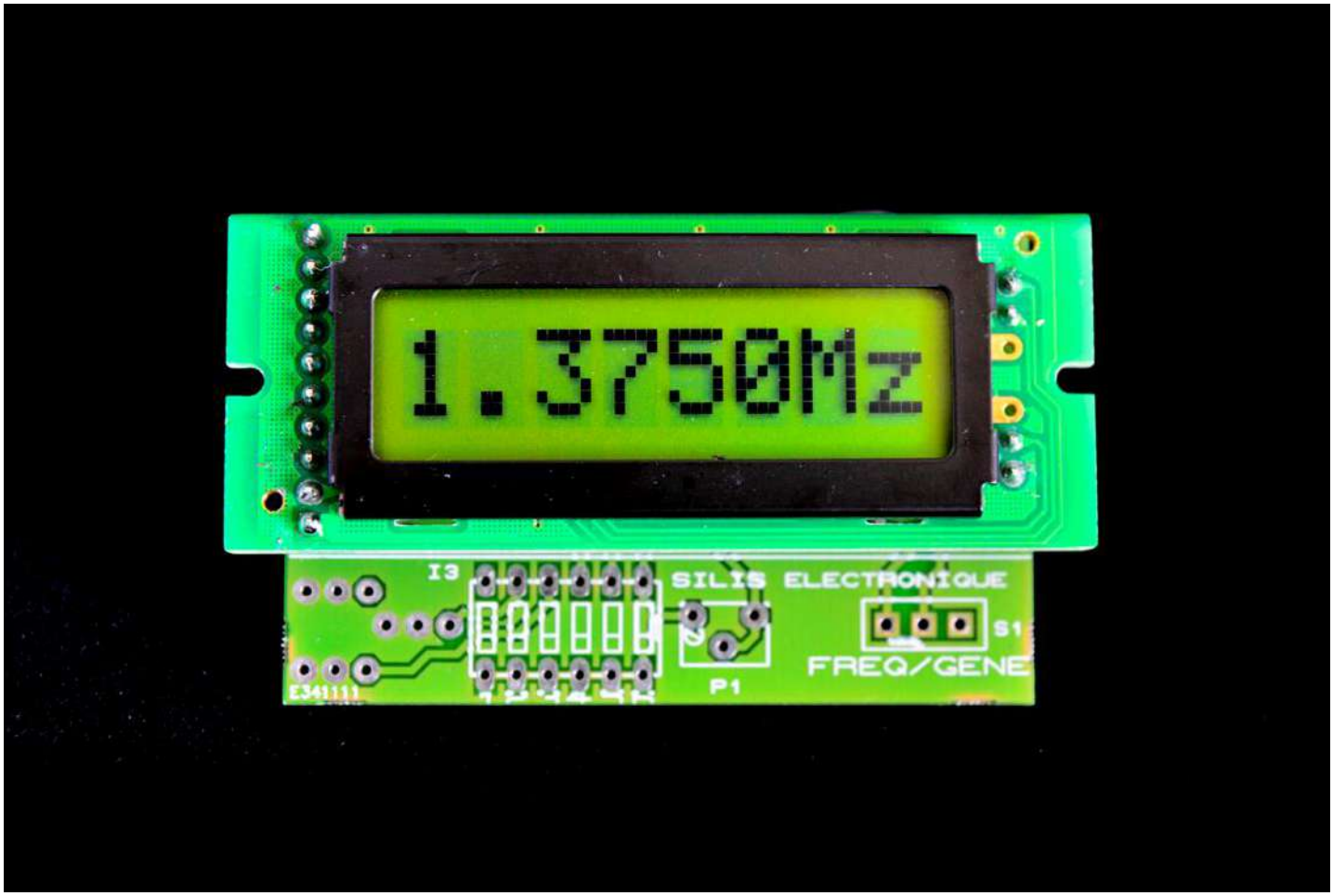
LE GROS Clara
GAUTIER Arthur
Groupe C

Le 12/06/2015



Rapport de Projet : Système Numérique

Microcontrôleur et Fréquencemètre

Table des matières

LE GROS Clara
GAUTIER Arthur
Groupe C

Le 12/06/2015

Introduction

Dans le cadre de notre seconde année à l'EFREI, il nous a été demandé de réaliser un projet mettant en œuvre des compétences obtenues nos cours de système numérique et d'architecture des ordinateurs, en manipulant une carte de développement CME11E9-EVBU en langage machine assembleur et réalisée autour d'un microcontrôleur 68HC11 de Motorola. Cette carte de développement nous a permis de manipuler des registres et des processeurs, comme ce que nous avons pu voir pendant nos cours. La finalité du projet était l'affichage d'une fréquence en hexadécimal sur un écran LCD. Nous avons donc, à travers ces trois séances de projet, réalisés un fréquencemètre à l'aide de la carte de développement.

Ainsi, le projet c'est articulé autour de trois séances. La première était consacrée à la prise en main de la carte de développement, la seconde à la réalisation du fréquencemètre et la troisième à la manipulation de l'écran LCD. Pour cela, nous avons utilisé la carte de développement CME11E9-EVBU que l'on manipule en langage assembleur. Cette carte était reliée à l'ordinateur à l'aide d'un câble DB9. Afin de manipuler la carte, nous sommes passés par une machine virtuelle et le moniteur BUFFALO 3.4. Enfin, l'interface avec la carte nous était fournie par le logiciel AxIDE, logiciel grâce auquel on pouvait compiler nos programmes rédigés à l'aide du bloc-notes (programmes en .asm à partir desquels on obtient un fichier .lst et un fichier .s19) puis exécuter les programmes (en .s19). Nous avons également eu besoin d'un générateur afin de produire les signaux.

Le projet

Expériences et Résultats

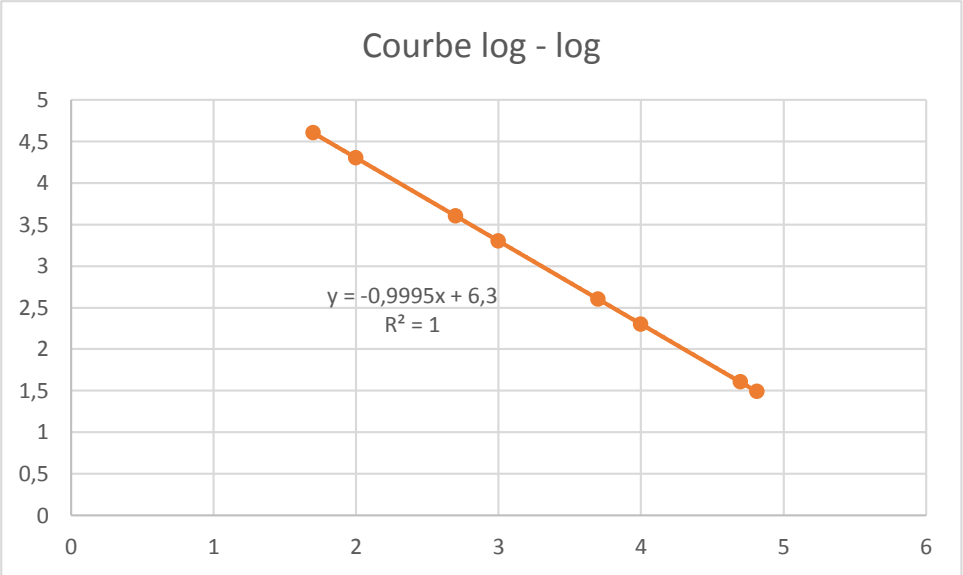
Les principales expériences que nous avons réalisées ont été la mesure de la fréquence à l'aide d'un programme ainsi que son affichage sur l'écran LCD.

Pour la mesure de la fréquence, nous avons effectué les mesures avec une méthode de polling (ou de scrutation) que nous détaillerons avec les réponses aux questions. Le polling correspond au nombre de cycles effectués par la carte (et va donc nous permettre de calculer la période).

Voilà les résultats que nous avons obtenus :

F HZ	Polling Hexa	Polling Deci
30	0468	1128
50	9C42	40 002
100	4E22	20 002
500	0FA5	4 005
1k	07D0	2 000
5k	0190	400
10k	00C8	200
50k	0028	40
65k	001F	31

Ces valeurs nous donnent la courbe suivante, une fois projetée dans un repère log-log ayant la fréquence d'entrée en ordonnée et le nombre de cycles en abscisse.



On remarque que bien le nombre de cycle (et donc la période) est inversement proportionnelle à la fréquence, ce qui est logique vu que nombre de cycle et période sont liés.

Pour l'affichage, nous nous sommes heurtés à un problème, le programme nous étant fourni (détaillé plus tard) ne réussissant à afficher les lettres. Voilà ce que l'on obtient pour une fréquence d'entrée d'environ (en réalité légèrement supérieure) 50 Hz :

Passons désormais au détail des questions qui nous étaient posées ainsi qu'à leur réponse.

LE GROS Clara
GAUTIER Arthur
Groupe C

Le 12/06/2015

Séance 1 : Prise en main

Nous avons connecté la carte à l'ordinateur à l'aide du câble DB9. Nous avons configuré la connexion à l'aide de AxIDE. L'affichage du prompteur nous a confirmé le bon fonctionnement de cette connexion.

QV1 : Nous avons compilé le programme hello.asm qui nous était fourni avec la carte puis nous avons exécuté le programme hello.S19 à l'aide de la commande call 2400.

```
>call 2400
Hello World
P-2400 Y-FFFF X-2451 A-0D B-90 C-D4 S-0041
>
```

Nous utilisons « call 2400 » car c'est l'adresse à laquelle est chargé le programme, comme son code nous l'indique :

```
**********************
* Program starts here *
**********************
        ORG   $2400
```

Il nous était ensuite demandé d'écrire une donnée en mémoire, la valeur $DA.

Pour cela, voilà le fichier .asm que nous avons rédigé :

```
        ORG     $2000
        FCB     $DA
```

Ici, le programme commence à l'adresse $2000 (org indiquant l'adresse à laquelle commence le programme ou la partie du programme), on utilisera donc la commande « call 2000 » pour l'exécuter. La commande fcb va écrire en mémoire à l'adresse actuelle la valeur qu'on lui indique (ici $DA, le $ servant à indiquer que l'on travaille sur une valeur en hexadécimal).

LE GROS Clara
GAUTIER Arthur
Groupe C

Le 12/06/2015

Nous avons relevé l'état de la mémoire après l'exécution du programme :

```
>md 2000

2000 DA F5 68 DF EE BE CD FF 7F F6 FF 5F 1F AF FD 76   h          _    v
2010 FE FB 6B D1 EB FD FE FE CE C2 F1 BF DA 72 AE DF   k            r
2020 FD 7B B2 E7 B4 CD ED EA 7D CF OB 2F FE 7F FB 7E              /
2030 F6 77 DA D9 F5 6F E6 9E 95 79 3F 5D 4E 5F AD F9  w   o    y?]N_
2040 7F BB B2 BE AO 5F F9 ED 9B CF F4 D8 CE 2C 3F 36        _         ,?6
2050 AF FB F9 O4 5A 67 63 77 EF 2F E7 B1 7D 30 B9 7D      Zgcw /    O
2060 EF FF A7 D7 F1 8D FF 56 F5 74 5B BB F9 49 71 B6         V t[   Iq
2070 BE 33 E9 FD E3 3F 3F 1F 1E 6D FD FD FB BD DD 86  3    ??  m
2080 DC 3E F4 F7 F9 E6 AE FD 6A FF 58 26 FB 3F FD BA  >       j X& ?
```

On retrouve bien la valeur $DA à l'adresse $2000.

Puis, nous avons éteint l'alimentation et rallumer la carte avant de revérifier l'état de la mémoire à l'adresse $2000 :

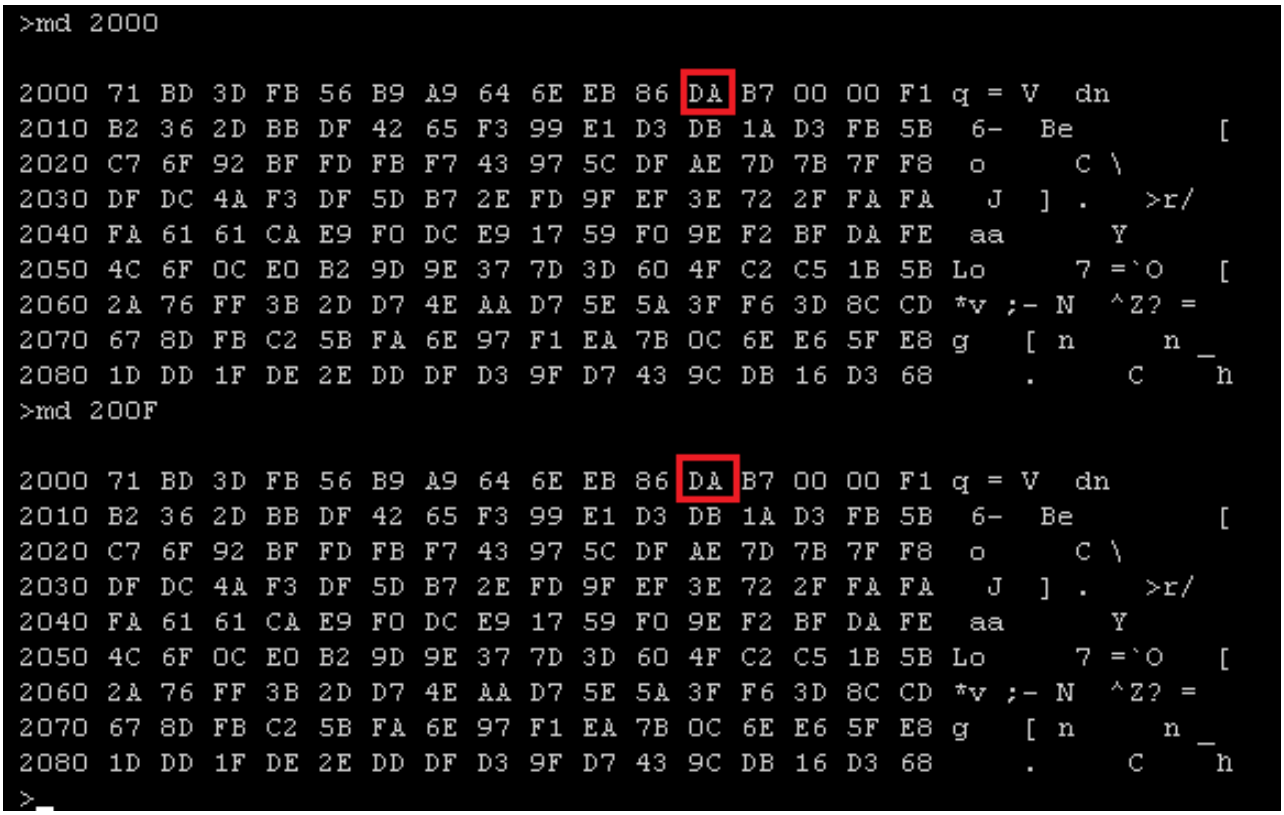
```
>md 2000

2000 6F F5 68 DF EE BE CC FF 7F F6 FF 57 1B AF FD 36 o h         W    6
2010 7E FA FB D1 AB 75 FE FA CE 82 F1 FF DA 72 AF DF       u         r
2020 FD 7B B2 E7 A4 CD ED 2A 7D CD OB AF BC 7F FB 7E          *
2030 E6 77 DA D1 E7 6F E2 9E 91 79 3F 5D OF 5F A9 E9  w   o    y?] _
2040 1F BB B2 BE AO 5F F1 E5 9B CF F4 D8 CA 2C 3F 36        _         ,?6
2050 AF B3 F9 O4 5A 63 63 77 ED 27 E5 B1 3D 20 A9 7D      Zccw '   =
2060 EF FB A7 CF B1 89 FF 56 F4 74 49 BB F9 49 71 36         V tI   Iq6
2070 96 A3 E8 BD E3 3E 3E 1F 1E 6D BD F9 7B BD 59 86       >>  m     Y
2080 5E 3E F4 F5 F9 E6 EE 39 6A F7 48 O6 FB 2F FD BA ^>      9j H  /
```

On remarque la valeur $DA a disparu de la mémoire.

Nous avons ensuite exécuté le programme suivant :

```
        ORG     $2000
VAL     RMB     1
        ORG     $200A
        LDAA    #$DA
        STAA    VAL
```

On va lancer le programme à l'aide d'un « call 2000 ». Ce programme va ensuite réserver un octet appelé « val » à l'aide de la commande « rmb ». Ensuite, il va se déplacer à l'adresse $200A où il va écrire $DA ainsi que le stocker dans l'accumulateur A, qu'il stockera ensuite dans la variable val. Nous avons relevé l'état de la mémoire en $2000 et en $200F et on remarque bien l'apparition de notre DA à l'adresse $200A :



Q2 : Voilà le fichier .lst généré par la compilation du code précédent :

```
0001 2000                                  ORG     $2000
0002 2000                          VAL     RMB     1
0003 200A                                  ORG     $200A
0004 200A 86 da              [ 2 ]         LDAA    #$DA
0005 200C b7 20 00           [ 4 ]         STAA    VAL
```

Le code mnémonique associé à l'instruction LDAA est 86 et celui associé à staa est b7.

On retrouve dans le fichier .S19, l'exécutable traduit de notre programme en Assembleur pour la carte de développement, des éléments du fichier .lst, notamment « 200A86DAB7200090 », c'est-à-dire l'adresse $200A et les instructions 86 da et b7 20 00 :

```
S108200A86DAB7200090
S9030000FC
```

QV3 & Q4 : On s'intéresse maintenant aux boucles et à leur fonctionnement. On compile le fichier .asm suivant :

```
head    equ     $2000
tail    equ     $2009
        org     $2010
        ldaa    #$01
        ldx     #head
loop    staa    0, x
        inca
        inx
        cpx     #tail
        bne     loop
        rts
        end
```

Le programme commence à l'adresse $2010. Ici, on va initialiser deux variables head et tail, respectivement à l'adresse $2000 et $2009. Puis, on charger la valeur $01 dans l'accumulateur a et l'adresse de head dans l'accumulateur x. On va ensuite incrémenter l'accumulateur x jusqu'à qu'il vaille la valeur de tail. Une fois cette valeur atteinte, il va sortir de la boucle.

Q5 : On va exécuter le programme suivant :

```
porta   equ     $1000
        org     $2000
        ldaa    #$10
        staa    porta
        rts
```

Le programme initialise la variable « porta » à l'adresse $1000 et débute à $2000. On charge la valeur $10 dans l'accumulateur A, puis on la range dans « porta ».

QV6 : PA2 est une entrée du port A (les bits d'entrée du port A sont PA0, PA1 et PA2).

On compile le programme suivant afin d'obtenir sa durée (à l'aide de son nombre de cycle et de la cadence interne du microcontrôleur) :

```
porta   equ     $1000
        org     $2000
start   ldaa    porta
        asla
        asla
        staa porta
        bra start
        rts
```

Ce programme va initialiser la variable « porta » à l'adresse $1000 et commence à l'adresse $2000. On va charger la valeur de porta dans l'accumulateur puis effectuer deux décalages à gauche dans cet accumulateur. Ensuite, on va stocker la nouvelle valeur de a dans porta et on recommence au « start » c'est-à-dire au chargement de porta dans a : on boucle.

Le fichier .lst que nous donne la compilation est le suivant. Les nombres entre crochet indiquent le nombre de cycles :

```
0001 1000                          porta       equ     $1000
0002 2000                               org     $2000
0003 2000 b6 10 00          [ 4 ] start       ldaa    porta
0004 2003 48                [ 2 ]       asla
0005 2004 48                [ 2 ]       asla
0006 2005 b7 10 00          [ 4 ]       staa porta
0007 2008 20 f6             [ 3 ]       bra start
0008 200a 39                [ 5 ]       rts start
```

Une boucle du programme représente donc : 4 + 2 + 2 + 4 + 3 = 17 cycles.

Or, un cycle représente $0,5*10^{-6}$s. Donc une boucle dure $17 * 0,5*10^{-6} = 8,5 * 10^{-6}$s.

QV8 : On a ici choisi de reprendre le programme vu en Q6 mais en déplaçant les décalages dans une sous-routine, appelée sspro. Le programme en .asm est donc le suivant :

```
porta   equ     $1000
        org     $2000
start   ldaa    porta
        jsr     sspro
        staa    porta
        bra     start
sspro   asla
        asla
        rts
```

On obtient le fichier .lst suivant :

```
0001 1000                          porta        equ     $1000
0002 2000                               org     $2000
0003 2000 b6 10 00          [ 4 ] start        ldaa    porta
0004 2003 bd 20 0b          [ 6 ]      jsr     sspro
0005 2006 b7 10 00          [ 4 ]      staa    porta
0006 2009 20 f5             [ 3 ]      bra     start
0007 200b 48                [ 2 ] sspro        asla
0008 200c 48                [ 2 ]      asla
0009 200d 39                [ 5 ]      rts
```

On a donc 4 + 6 + 4 + 3 + 2 + 2 = 21 cycles soit $10,5*10^{-6}$s.

Cette première séance aura été l'occasion de se familiariser avec la carte de développement, son microcontrôleur et le logiciel AxIDE. On manipule la carte à l'aide de programme en .asm, qui une fois compilés nous donne deux programmes : l'un en .lst (nous permettant de mesurer la durée d'un programme ou d'une partie de ce programme) et le second en .S19 qui sera le fichier exécuté par la carte. De plus, nous avons réussi à communiquer avec un port entrée/sortie de la carte et nous avons réussi à écrire en mémoire. C'était une partie obligatoire afin de pouvoir réaliser le but de ce projet : le fréquencemètre.

LE GROS Clara
GAUTIER Arthur
Groupe C

Le 12/06/2015

Séance 2 : Fréquencemètre

La mesure d'une fréquence par la carte de développement peut être faite à l'aide de deux méthodes : la méthode de polling (ou de scrutation), dans laquelle le programme scrute en permanence l'arrivée du premier front du signal, puis du second, de même nature, et en déduit sa période, ou la méthode optimisée par interruption, où le programme principal est interrompu par l'arrivée d'un premier front et déduit la période du signal à l'arrivée du second front de même nature.

On va d'abord étudier la méthode de polling, avant de s'intéresser brièvement à la méthode par interruption. La méthode du polling va consister à relever la valeur du timer interne de la carte (TCNT) à l'arrivée du premier et du second front, et de faire la soustraction des deux valeurs pour déterminer la période.

Q0 : Nous avons compilé le programme fourni par le sujet, afin d'obtenir le nombre de cycles séparant les deux chargements du TCNT dans les accumulateurs x et y :

```
0001 100e                          TCNT equ     $100E
0002
0003 2000                               org     $2000
0004
0005 2000 fe 10 0e           [ 5 ]      ldx     TCNT
0006 2003 01                 [ 2 ]      nop
0007 2004 18 fe 10 0e        [ 6 ]      ldy     TCNT
0008
0009 2008 39                 [ 5 ]      rts TCNT
```

On compte donc 2 + 6 = 8 cycles entre la fin du chargement du TCNT dans x et le chargement du TCNT dans y.

Voilà ce que l'on obtient en exécutant le programme :

```
>call 2000

P-2000 Y-9055 X-904D A-FF B-FF C-D8 S-0041
>
```

En faisant la soustraction X – Y, on obtient 8, qui représente le nombre de cycles séparant les deux chargements. Un cycle de l'horloge du CPU va incrémenter le TCNT de 1.

LE GROS Clara
GAUTIER Arthur
Groupe C

Le 12/06/2015

Q1 : On a compilé le programme donné par le sujet, afin de compter le nombre de cycles séparant le chargement de l'accumulateur d avec le TCNT.

```
0001 100e                              TCNT equ     $100E
0002
0003 2000                                   org     $2000
0004 2000                              initial      rmb     2
0005 2002                              delta        rmb     2
0006
0007 2050                                   org     $2050
0008
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 delta
```

On peut ici voir que 10 cycles séparent les deux chargements.

On relève ensuite les états des registres après l'exécution :

```
P-2050 Y-779F X-7797 A-00 B-0A C-D0 S-0041
```

Le registre D étant représenté par la concaténation des registres A et B, il contient bien la valeur A (10 en décimal), ce qui représente le nombre de cycles entre les deux chargements de d. Cependant, la mesure effectuée par ce programme pour un processus ne sera pas correct, puisque les 5 cycles du stockage de d dans la variable initial et le chargement de d après le processus se rajouteront à la durée du processus.

Si on vérifie la mémoire à l'adresse $2002 :

```
2000 14 28 00 0A 18 FE 10 0E 39 FB 1F 7A EB D8 46 34
```

On retrouve bien notre valeur A et de B aux adresses $2002 et $2003.

Q2 : Le registre TIC1 utilise PA2 comme entrée du signal, TIC2 utilise PA1 et TIC3 utilise PA0.

Q3 :

```
TIC1    equ     $1010
TFLG1   equ     $1023
TCTL2   equ     $1021

        org     $2050
front   rmb     2
period  rmb     2
ED1ba   fcb     $10 ; On ecrit une constant ED1ba à l'adresse 10

        org     $2100

        ldab    ED1ba ; On charge B avec ED1ba
        stab    TCTL2 ; On stocke B dans TCTL2

        ldaa    TFLG1 ; On charge A avec TFLG1
        anda    #$04

        staa    TFLG1 ; On stocke A dans TFLG1

lire1   ldaa    TFLG1 ; On charge TFLG1 dans A
        anda    #$04 ; On vérifie A avec la valeur $04
        beq     lire1 ; Si A est différent de $04, alors, on execute lire1

suite1  ldd     TIC1 ; On charge TIC1 dans D
        std     front ; On stocke D dans front

        ldaa    TFLG1 ; On charge A avec TFLG1
        anda    #$04 ; On vérifie A avec la valeur $04 : on vérifie juste le 3ème bit
        staa    TFLG1 ; On stocke A dans TFLG1

lire2   ldaa    TFLG1 ; On charge A avec TFLG1
        anda    #$04 ; On vérifie A avec la valeur de $04
        beq     lire2 ; Si A est différent de $04 alors on execute lire2

suite2  ldd     TIC1 ; On charge TIC1 dans D
        subd    front ; On compare D avec front

        std period ; On stocke D dans period

        rts
```

On initialise les valeurs TIC1, TFLG1 et TCTL2 à $1010, $1023 et $1021. On réserve 2 octets pour « front » et « period » à partir de l'adresse $2050, et on définit ED1ba (une constante) avec la valeur $10.

On indique ensuite que le programme commence à l'adresse $2100. On charge la constante ED1ba dans l'accumulateur b et on le range dans TCTL2. On charge TFLG1 dans l'accumulateur a et on le compare à la valeur $04 (0100 en binaire), on stocke le résultat dans a, puis on range la valeur nouvelle dans TFLG1.

On a ensuite le sous-programme « lire1 », qui charge le contenu de TFLG1 dans l'accumulateur a, le compare à la valeur $04, c'est-à-dire que l'on vérifie le 3^{e} bit. S'il vaut 0, on réappelle le sous-programme « lire1 ».

Le sous-programme « suite1 », charge le contenu de TIC1 dans l'accumulateur d et le range dans « front ». On charge le contenu de TFLG1 dans a, on le compare à $04 et on le range dans TFLG1.

Le sous-programme « lire2 », charge le contenu de TFLG1 dans l'accumulateur a, le compare à la valeur $04. Si on a un zéro, on réappelle le sous-programme « lire2 ».

Enfin, le sous-programme « suite2 » charge le contenu de TIC1 dans l'accumulateur d puis soustrait à ce dernier la valeur contenue dans « front », c'est-à-dire le temps retenu lors de la détection du premier front. On range ensuite le contenu de l'accumulateur d, qui contient résultat de la soustraction, dans « period ».

Pour déterminer le nombre de cycles séparant les deux chargements de d, on utilise le fichier .lst :

```
0001 1010                          TIC1 equ     $1010
0002 1023                          TFLG1        equ     $102
0003 1021                          TCTL2        equ     $102
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 ; On c
0013 2103 f7 10 21          [ 4 ]       stab    TCTL2 ; On s
0014
0015 2106 b6 10 23          [ 4 ]       ldaa    TFLG1 ; On c
0016 2109 84 04             [ 2 ]       anda    #$04
0017
0018 210b b7 10 23          [ 4 ]       staa    TFLG1 ; On s
0019
0020 210e b6 10 23          [ 4 ] lire1         ldaa    TFLG
0021 2111 84 04             [ 2 ]       anda    #$04 ; On vé
0022 2113 27 f9             [ 3 ]       beq     lire1 ; Si A
0023
0024 2115 fc 10 10          [ 5 ] suite1        ldd     TIC1
0025 2118 fd 20 50          [ 5 ]       std     front ; On s
0026
0027 211b b6 10 23          [ 4 ]       ldaa    TFLG1 ; On c
0028 211e 84 04             [ 2 ]       anda    #$04 ; On vé
0029 2120 b7 10 23          [ 4 ]       staa    TFLG1 ; On s
0030
0031 2123 b6 10 23          [ 4 ] lire2         ldaa    TFLG
0032 2126 84 04             [ 2 ]       anda    #$04 ; On vé
0033 2128 27 f9             [ 3 ]       beq     lire2 ; Si A
0034
0035 212a fc 10 10          [ 5 ] suite2        ldd     TIC1
0036 212d b3 20 50          [ 6 ]       subd    front ; On c
0037
0038 2130 fd 20 52          [ 5 ]       std period ; On stoc
0039
0040 2133 39                [ 5 ]       rts period ; On stoc
```

On peut voir qu'il y a 4 + 2 + 4 + 5 + 5 + n*(4 + 2 + 3) = 20 + 9n cycles entre les deux chargements de d.

Q4 : Voir la partie Expérience et Résultat

Q5 : On va maintenant essayer de déterminer les fréquences limites pour lesquelles cette méthode ne fonctionne plus. En passant par une méthode expérimentale, on a déterminé que la méthode ne fonctionnait plus aux alentours et en dessous de 30 Hz et au-dessus de 65 kHz.

Voyons maintenant comment on peut confirmer ces résultats par le calcul.

On a déterminé que le nombre de cycles était de 20 + 9n. Donc la période est de (20 + 9n) * 0,5 * 10^{-6}. Et la fréquence : f = $\frac{1}{(20+9n)*0,5*10^{-6}}$.

La fréquence maximale est donc pour le nombre de cycles minimal soit 29. On trouve alors f_{max} = $\frac{1}{29*0,5*10^{-6}} \approx 69\ \text{kHz}$.

Pour déterminer la fréquence minimale, on doit utiliser le plus grand nombre de cycles possibles donc la plus grande valeur possible de l'accumulateur : $FFFF (65535 en décimal) d'où f_{min} = $\frac{1}{65535*0,5*10^{-6}} \approx 30\ \text{Hz}$.

Ainsi, pour s'intéresser à des fréquences inférieures à 30Hz ou supérieures à 69 kHz, il faudra utiliser la méthode optimisée par interruption.

Q6 : On compile le programme suivant :

```
TIC1    equ     $1010
TFLG1   equ     $1023
TCTL2   equ     $1021
TMSK1   equ     $1022

        org     $2050
front   rmb     2
period  rmb     2
ED1ba   fcb     $10

        org     $00E8
        jmp     issprog

        org     $2100

        sei             ;

        lds     #$0200

        ldab    ED1ba
        stab    TCTL2

        ldaa    TFLG1
        anda    #$04
        staa    TFLG1

        ldaa    #$04
        staa    TMSK1
        cli             ;

        ldd     #$0000
        std     $2000
boucle  bra     boucle

issprog ldd     #$ACDC
        std     $2000

        rts
```

Le programme va réaliser ses initialisations sans interruption (grâce aux commandes sei et cli). Une fois les initialisations réalisées, le programme va s'enfermer dans une boucle infinie (« boucle bra boucle »). Si il est interrompu par un vecteur d'interruption, il ira en $00E8, qui l'amènera au sous-programme issprog qui écrira la valeur $ACDC à l'adresse 2000. Il retourne ensuite dans le boucle infinie.

QV7 : On relève l'état de la mémoire en $2000 et en $00E8 avant l'execution :

```
>md 2000

2000 4C 17 6F 4E DA 6D 5B FC 02 1F C8 7F D5 4B 67 EF L oN m[      Kg
2010 BC 3B 9D 2F F4 AB 5D F7 68 FD EC 7D 77 95 FB 79  ; /  ] h   w  y
2020 16 BB 7F BF DB 9A BB FA B6 EA DF 77 7B 97 7A ED            w  z
2030 CC F3 A4 9D 38 F7 BF E4 9F 8F F3 EC DB 6D EF 2E     8        m .
2040 FC 9B B5 DD 3F AE DA FD D5 FF FF 0F 6B E2 7F DE     ?        k
2050 EF F7 59 44 10 E1 66 5F 53 FB 21 9C FB 0B 7F 77   YD  f_S !     w
2060 E8 DE 12 01 EE BA 5F 76 67 C9 3B 5F E7 EB E9 FD       _vg ;_
2070 91 7C 2B 6B DD EF 3F C7 2F C1 C7 5F FA 56 FE DF   +k  ? /  _ V
2080 EF DF 1F 88 B6 F9 76 4D 3F AC FF E9 2E FE 2D FB       vM?   . -
```

```
>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
```

Puis, on exécute le programme sans signal et on relève l'état de la mémoire en $2000 :

```
>md 2000

2000 00 00 6F 4E DA 6D 5B FC 02 1F C8 7F D5 4B 67 EF   oN m[      Kg
2010 BC 3B 9D 2F F4 AB 5D F7 68 FD EC 7D 77 95 FB 79  ; /  ] h   w  y
2020 16 BB 7F BF DB 9A BB FA B6 EA DF 77 7B 97 7A ED            w  z
2030 CC F3 A4 9D 38 F7 BF E4 9F 8F F3 EC DB 6D EF 2E     8        m .
2040 FC 9B B5 DD 3F AE DA FD D5 FF FF 0F 6B E2 7F DE     ?        k
2050 EF F7 59 44 10 E1 66 5F 53 FB 21 9C FB 0B 7F 77   YD  f_S !     w
2060 E8 DE 12 01 EE BA 5F 76 67 C9 3B 5F E7 EB E9 FD       _vg ;_
2070 91 7C 2B 6B DD EF 3F C7 2F C1 C7 5F FA 56 FE DF   +k  ? /  _ V
2080 EF DF 1F 88 B6 F9 76 4D 3F AC FF E9 2E FE 2D FB       vM?   . -
```

Comme il n'y a pas de signal, on n'exécute jamais le sous-programme issprog et $ACDC ne s'écrit pas en 2000.

Avec un signal :

```
>md 2000

2000 AC DC 6F 4E DA 6D 5B FC O2 1F C8 7F D5 4B 67 EF   oN m[      Kg
2010 BC 3B 9D 2F F4 AB 5D F7 68 FD EC 7D 77 95 FB 79  ; /  ] h   w  y
2020 16 BB 7F BF DB 9A BB FA B6 EA DF 77 7B 97 7A ED            w  z
2030 CC F3 A4 9D 38 F7 BF E4 9F 8F F3 EC DB 6D EF 2E     8        m .
2040 FC 9B B5 DD 3F AE DA FD D5 FF FF OF 6B E2 7F DE     ?       k
2050 EF F7 59 44 10 E1 66 5F 53 FB 21 9C FB OB 7F 77   YD  f_S !     w
2060 E8 DE 12 O1 EE BA 5F 76 67 C9 3B 5F E7 EB E9 FD       _vg ;_
2070 91 7C 2B 6B DD EF 3F C7 2F C1 C7 5F FA 56 FE DF   +k  ? /  _ V
2080 EF DF 1F 88 B6 F9 76 4D 3F AC FF E9 2E FE 2D FB       vM?   . -
```

On remarque l'écriture de $ACDC en $2000.

Dans cette séance, nous avons donc réussi à réaliser un fréquencemètre à l'aide de la carte de développement.

Voyons maintenant comment afficher cette fréquence directement sur un écran LCD.

Séance 3 : Affichage sur un écran LCD

Q1 : Le sujet nous a fourni un programme pour permettre l'initialisation de l'écran. Dans ce programme, il fait appel à deux boucles de temporisation, appelées « longue » et « courte ». Ces boucles permettent à l'écran d'effectuer les commandes, qui ont un délai variable, soit court ($40*10^{-6}$s) soit long ($1,65*10^{-3}$s).

On compile le programme afin de récupérer le nombre de cycle de chaque boucle :

```
0037 2130 ce c3 50          [ 3 ] longue      ldx     #50000
0038 2133 09                [ 3 ] loop_l      dex
0039 2134 26 fd             [ 3 ]     bne     loop_l
0040 2136 39                [ 5 ]     rts
0041
0042 2137 ce 03 e8          [ 3 ] courte      ldx     #1000
0043 213a 09                [ 3 ] loop_c      dex
0044 213b 26 fd             [ 3 ]     bne     loop_c
0045 213d 39                [ 5 ]     rts loop_c
```

On peut voir que les deux boucles durent 9 cycles, la différence entre les deux étant le nombre de passage dans la boucle. La première fera 50000 tours, et la seconde seulement 1000. Ainsi, la première boucle durera $9*10^{-6}*5*10^{5} = 45*10^{-1}$s et la seconde $9*10^{-6}*10^{3} = 9*10^{-3}$s.

Q2 : On ajoute au programme précédent une partie permettant l'affichage de caractère sur l'écran :

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

        ldaa    #'B'
        staa    LCDDAT
        jsr     courte

        ldaa    #'C'
        staa    LCDDAT
        jsr     courte

        ldaa    #'D'
        staa    LCDDAT
        jsr     courte

        rts
```

Une fois exécuté, voilà ce qu'affiche le programme sur l'écran LCD :

Q3 : On va maintenant chercher à afficher un texte en plus d'une simple chaîne de caractère. Pour cela, on ajoute encore une partie au programme précédent :

```
org  $2000
texto       fcc     'La periode ='

fin  fcb     #0
           ldy     #texto
      chaine       ldaa    0,y
           beq     suite
           staa    LCDDAT
           jsr     courte
           iny
           bra     chaine
```

La directive fcc permet donc de stocker une chaîne de caractère en mémoire.

Ce programme va charger l'adresse de texto dans l'accumulateur y, puis va récupérer ce vers quoi pointe y dans l'accumulateur (au premier tour de boucle, y pointe donc vers « L »). Si jamais toute la chaîne a été affichée, on passe au sous-programme « suite » qui est l'affichage des caractères ABCD. Sinon, on stocke l'accumulateur a dans le registre donnée du LCD (LCDDAT) ce qui va afficher le caractère contenu par l'accumulateur a sur l'écran. On incrémente ensuite y afin de passer au caractère suivant et on repasse au chargement de a.

Le résultat sur l'écran est le suivant :

Q4 : Dans cette question, on veut, à partir d'un octet (donc 2 caractère), le convertir en 2 quartets (quartet haut et quartet bas) afin de pouvoir l'afficher. Pour cela, on va exécuter le programme suivant :

```
LCDCMD  equ     $b5f0
LCDDAT  equ     $b5f1

L4xC20  equ     $3c * 4x20
CLRCH   equ     $01
DCBON   equ     $0f
CINSHOF equ     $06
SCSHR   equ     $14
CSRHOM  equ     $02

        org     #2000
texto   fcc     'La periode ='
fin     fcb     #0

        ORG     $2020

table   fcc     '0'
        fcc     '1'
        fcc     '2'
        fcc     '3'
        fcc     '4'
        fcc     '5'
        fcc     '6'
        fcc     '7'
        fcc     '8'
        fcc     '9'
        fcc     'A'
        fcc     'B'
        fcc     'C'
        fcc     'D'
        fcc     'E'
        fcc     'F'

octet   fcb     #$36
q_haut  rmb     1
q_bas   rmb     1
```

```
          ORG     $2100

conv      ldx     #table
          ldaa    octet
          anda    #$0F
          adda    0,x
          staa    q_bas

          ldaa    octet
          asra
          asra
          asra
          asra
          anda    #$0F
          adda    0,x
          staa    q_haut

init      ldaa    #L4xC20
          staa    LCDCMD
          jsr     longue

          ldaa    #CLRCH
          staa    LCDCMD
          jsr     longue

          ldaa    #DCBON
          staa    LCDCMD
          jsr     courte

          ldaa    #CINSHOF
          staa    LCDCMD
          jsr     courte

          ldaa    #SCSHR
          staa    LCDCMD
          jsr     courte

          ldaa    #CSRHOM
          staa    LCDCMD
          jsr     courte


           ldy     #texto
chaine     ldaa    0,y
           beq     suite
           staa    LCDDAT
           jsr     courte
           iny
           bra     chaine
```

```
suite   ldaa    q_haut
        staa    LCDDAT
        jsr     courte

        ldaa    q_bas
        staa    LCDDAT
        jsr     courte


longue  ldx     #50000
loop_l  dex
        bne     loop_l
        rts


courte  ldx     #1000
loop_c  dex
        bne     loop_c
        rts
```

Voyons maintenant ce que fait ce programme. Il commence par réserver 1 octet pour chaque quartet. Ensuite, le sous-programme « conv » va faire pointer l'accumulateur x vers le premier élément de table. Puis il va charger octet dans a. On ne va garder que les 4 bits de poids faible en premier lieu (d'où la comparaison avec $0F). Une fois que l'on a la valeur en hexadécimal des 4 bits de poids faible, on va l'ajouter à x, et stocker le résultat dans a, puis on stocke a dans le quartet bas. Le fait d'ajouter la valeur en hexadécimal à x permet de nous placer sur le bon caractère dans la table. Il faut maintenant initialiser le quartet haut. On va de nouveau charger l'octet dans l'accumulateur a. On va ensuite décaler le registre 4 fois à droite afin que les 4 bits de poids fort deviennent les 4 bits de poids fort. Puis on applique la même méthode que pour le quartet bas, si ce n'est qu'on va stocker le résultat dans le quartet haut.

Ici, l'écran affiche, après exécution : « La période est =36 ».

QV5 : Maintenant, comme la mesure de fréquence atteint 2 octets de précision, on va convertir 2 octets, chacun en 2 quartets.

Pour cela, nous avons rajouté 2 quartets dans le programme, que l'on initialise de la même manière que les deux quartets du premier programme.

Q6 : Enfin, la finalité de ce projet, l'affichage de la période sur un écran LCD. Pour cela, on va combiner le programme de la Q5 de la séance 3 avec celui de la Q3 de la séance 2. Voilà ce que cela donne :

```
TIC1    equ     $1010
TFLG1   equ     $1023
TCTL2   equ     $1021

LCDCMD  equ     $b5f0
LCDDAT  equ     $b5f1

L4xC20  equ     $3c * 4x20
CLRCH   equ     $01
DCBON   equ     $0f
CINSHOF equ     $06
SCSHR   equ     $14
CSRHOM  equ     $02

        org     $2050
front   rmb     2
period  rmb     2
ED1ba   fcb     $10

        org     #2000
texto   fcc     'La periode ='
fin     fcb     #0
```

```
          ORG     $2020

table     fcc     '0'
          fcc     '1'
          fcc     '2'
          fcc     '3'
          fcc     '4'
          fcc     '5'
          fcc     '6'
          fcc     '7'
          fcc     '8'
          fcc     '9'
          fcc     'A'
          fcc     'B'
          fcc     'C'
          fcc     'D'
          fcc     'E'
          fcc     'F'

octeta    rmb     1
octetb    rmb     1
q_haut    rmb     1
q_bas     rmb     1
q2_haut   rmb     1
q2_bas    rmb     1
```

```
          org     $2100

          ldab    ED1ba
          stab    TCTL2

          ldaa    TFLG1
          anda    #$04

          staa    TFLG1

lire1     ldaa    TFLG1
          anda    #$04
          beq     lire1

suite1    ldd     TIC1
          std     front

          ldaa    TFLG1
          anda    #$04
          staa    TFLG1

lire2     ldaa    TFLG1
          anda    #$04
          beq     lire2

suite2    ldd     TIC1
          subd    front

          std period
```

```
        staa    octeta
        stab    octetb

conv    ldx     #table
        ldaa    octeta
        anda    #$0F
        adda    0,x
        staa    q_bas

        ldaa    octeta
        asra
        asra
        asra
        asra
        anda    #$0F
        adda    0,x
        staa    q_haut

        ldab    octetb
        andb    #$0F
        addb    0,x
        stab    q2_bas

        ldab    octetb
        asrb
        asrb
        asrb
        asrb
        andb    #$0F
        addb    0,x
        stab    q2_haut
```

```
init    ldaa    #L4xC20
        staa    LCDCMD
        jsr     longue

        ldaa    #CLRCH
        staa    LCDCMD
        jsr     longue

        ldaa    #DCBON
        staa    LCDCMD
        jsr     courte

        ldaa    #CINSHOF
        staa    LCDCMD
        jsr     courte

        ldaa    #SCSHR
        staa    LCDCMD
        jsr     courte

        ldaa    #CSRHOM
        staa    LCDCMD
        jsr     courte
```

```
          ldy     #texto
chaine    ldaa    0,y
          beq     suite
          staa    LCDDAT
          jsr     courte
          iny
          bra     chaine

suite     ldaa    q_haut
          staa    LCDDAT
          jsr     courte

          ldaa    q_bas
          staa    LCDDAT
          jsr     courte

          ldab    q2_haut
          stab    LCDDAT
          jsr     courte

          ldab    q2_bas
          stab    LCDDAT
          jsr     courte

*

longue    ldx     #50000
loop_l    dex
          bne     loop_l
          rts

*

courte    ldx     #1000
loop_c    dex
          bne     loop_c
          rts
```

Ainsi, comme le nombre de cycles est stocké dans le registre d, qui est la concaténation des registres a et b, on peut récupérer le premier octet grâce au registre a et le second grâce au registre b. Une fois cela fait, on va réutiliser le sous-programme « conv » afin de séparer les octets en quartet. Puis on réalise l'affichage. Cependant, le programme, à cause de la table, ne permet pas d'afficher les lettres. Pour l'affichage, il est dans la partie Expérience et Résultats.

Conclusion

Ainsi, ce projet qui nous demandait de manipuler une carte de développement et le microcontrôleur qui lui est associée, nous a permis de réaliser un fréquencemètre et de l'afficher sur un écran LCD.

Ce projet, aura été l'occasion d'appliquer de façon très concrète ce que nous avons pu voir à travers nos cours de Systèmes Numérique et d'Architecture des Ordinateurs. C'aura été un projet très intéressant, et qui a touché autant à la physique qu'à l'informatique, allant même jusqu'à, parfois, confondre la frontière entre les deux. Bien que nous avons pu avoir, parfois, du mal sur certaines questions, ou à interpréter certains programmes, nous pensons que ce projet nous a permis de grandement progresser dans la compréhension de ces deux matières.

Nous avons beaucoup appris de ce projet et nous n'en apprécions que mieux les matières concernées, puisqu'elles sont véritablement devenues concrètes à nos yeux.