Jean-Michel FRIEDT, Mai 1999 (3 avril 2001)

Introduction au 68HC11(F1)

1 Introduction

Le choix du 68HC11F1 a été fait selon le critère d'avoir un micro-contrôleur ne nécessitant pas de programmeur, pouvant être programmé au vol à partir d'un port série[1] et comportant des convertisseurs analogique-numérique (A/D) tout en permettant l'interfaçage avec une RAM externe.

L'assembleur utilisé ici est disponible gratuitement par anonymous ftp à `ftp.cmf.nrl.navy.mil` dans `/pub/kenh/` (il s'agit de `asxxxx-v1.51.tar.gz` en Novembre 1999, bien que la version puisse changer ultérieurement). Cet assembleur a l'avantage d'être facilement adaptable à n'importe quel micro-contrôleur CISC (et inclut notamment des versions pour le 6809 et le Z80).

La méthode de compilation utilisée, que ce soit sous DOS ou Linux, est la suivante :

- écrire le programme en assembleur dans un éditeur de texte (le fichier sera ici nommé `prg.asm`). Par rapport aux exemples donnés dans le texte, il ne faut taper que la partie droite (par exemple pour une ligne du type
  `0000 8E 01 FF lds #0h01FF ; setup : stack`
  il ne faut entrer que `lds #0h01FF` suivi éventuellement des commentaires (le texte après le '`;`'). Le début de la ligne est l'adresse où se situe la ligne de code (ici début du programme donc adresse `0000`), puis les opcodes correspondant à cette ligne après assemblage (ici `8E 01 FF`). C'est la suite de ces opcodes que nous enverrons via la liaison série au 68HC11.

- l'assembler par `as6811 -o prg.asm` sous DOS ou `as6811 -o prg.rel prg.asm` sous Linux pour générer le fichier `prg.rel` qui contient le code en hexadécimal et d'autres informations.

- retirer les lignes ne commençant pas par un '`T`' (`grep ^T`) et les 8 premiers caractères des lignes restantes (qui sont le '`T`' suivi de l'adresse de début de ligne : `cut -c9-60`) et mettre le résultat dans `prg.out` : `grep T %1.rel | cut -c9-60 >%1.out` sous DOS ou `grep T $1.rel | cut -c9-60 >$1.out` sous Linux.

- enfin, programmer le 68HC11F1 en exécutant le programme DOS `hc11.exe prg.out`, résultat de la compilation de `hc11.pas` fourni en annexe 13 de ce document. À noter que le numéro du port série est en hard dans le programme sous le nom de variable `PortId`.

Les programmes présentés dans ce document ont été générés par as6811 -l prg.asm sous Linux (après édition des lignes inutiles). La même fonction est disponible sous DOS par as6811 -l prg.asm qui génère automatiquement le fichier `.lst`.

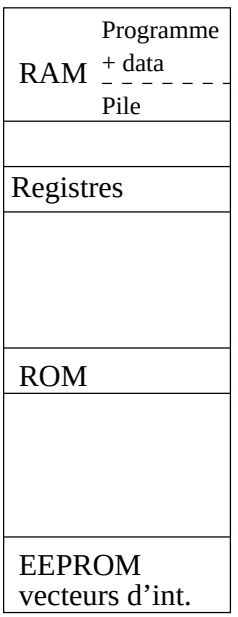
2 Généralités

Les instructions de base.

1. La restriction sur le port RS232 vient de l'utilisation d'un palmtop HP200LX comme programmeur. Cet ordinateur ne possède qu'un port série comme extension (absence de port parallèle).

2 registres 16 bits utilisés pour les accès mémoires : X, Y

2 registres généraux 8 bits : A, B qui peuvent se concaténer en D=AB.



- Les fonctions logiques habituelles : AND(A/B), LSL(A/B) (shift left), LSR(A/B) (shift right), ORA(A/B)
- Les opérations sur la pile : PSH(A/B/X/Y), PUL(A/B/X/Y)
- Les fonctions sur des bits : BCLR (bit clear), BSET (bit set)
- Les fonctions de saut relatif (court) : BRA (toujours), BNE ($\neq 0$), BSR (branch to subroutine), RTS
- Les fonctions de saut long : JMP, JSR,
- Les fonctions de chargement de valeur : LDA(A/B), LDD, LDS, LD(X/Y), ST(X/Y), STA(A/B), STD
- Les sauts avec test : BRCLR (branche si après masquage avec les bits à 0, le résultat est nul), BRSET (réalise un ET logique avec le masque fourni, et branche si le résultat est 0)
- Interruptions : CLI (autorise les interruptions), RTI, SEI (arrête les interruptions)
- Comparaison : CP(X/Y), CMP(A/B)
- Arithmétique : DEC(A/B), DE(X/Y), INC(A/B), IN(X/Y), SUB(A/B), SUBD, ADD(A/B), ADDD.

Organisation de la mémoire :

1024 premiers octets (0000-03FF) : RAM. L'exécution commence à l'adresse 0000 par un opcode fetch. Dans cette mémoire, nous trouvons : la pile (stack), positionnée par l'utilisateur au moyen de l'instruction `lds`, les pseudo-vecteurs d'interruptions (si les interruptions matérielles sont utilisées) de 00C4 à 00FF par paquets de 3 octets (JMP @).

Attention : la pile décroît lors de son utilisation (*i.e.* mettre un élément sur la pile par une instruction `psh` implique une **dé**croissance du pointeur de pile : $sp \leftarrow sp - 1$). Il faut donc placer la pile suffisamment loin du programme pour ne pas risquer une collision de la pile et des opcodes placés en RAM lors des empilements successifs liés aux appels de fonctions (`bsr` ou `psh`).

Les registres de configurations de 1000 à 105F (96 registres).

La ROM contenant le code à exécuter au boot, de BF00 à BFFF

L'EEPROM de FE00 à FFC0 (256 octets)

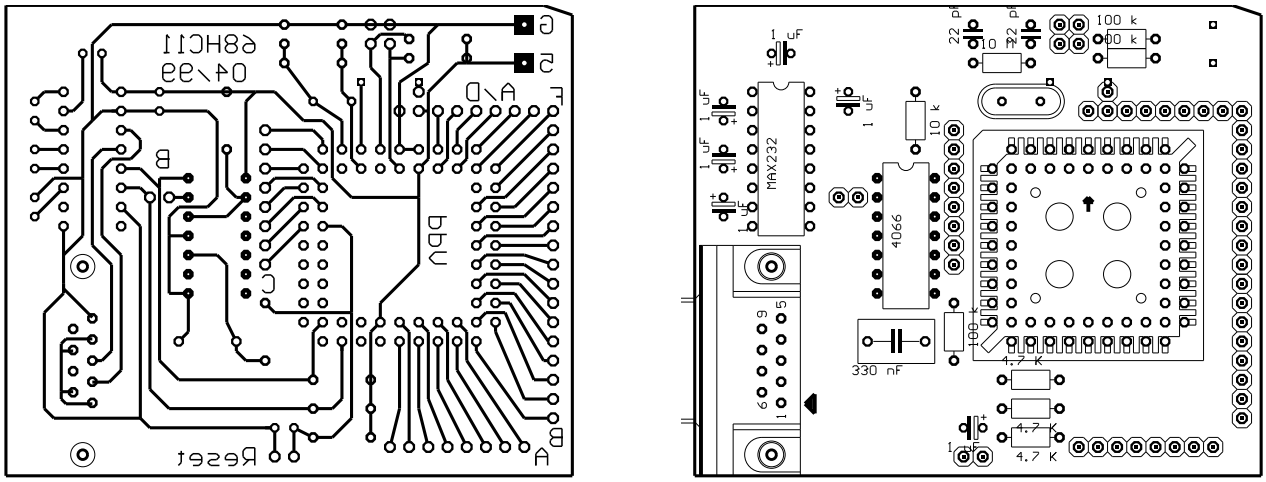
Il est à noter que l'organisation de la mémoire peut être optimisée en fonction des besoins par la possibilité de déplacer les registres de configuration, la RAM et l'EEPROM à n'importe quel emplacement de la forme x000 (x=0..F). Dans tous les cas nous démarrons en bootstrap mode, mais il est possible ensuite de passer en mode étendu et ainsi accéder à une mémoire extérieure.

3 Mise en RAM locale d'un programme et exécution

L'objectif dans un premier temps est de tester le montage de base comprenant uniquement le micro-contrôleur 68HC11 et la logique de communication par port RS232[2] ainsi que les logiciels associés (programme de communication du côté du PC pour envoyer le programme à exécuter par le micro-contrôleur, et programme à exécuter lui même).

2. La mémoire utilisée pour stocker le programme est incluse dans le micro-contrôleur (contrairement à plus loin où nous nous efforcerons d'interfacer une RAM externe avec le 68HC11).

Le montage proposé est le suivant :



Le circuit est composé de 3 éléments principaux : le 68HC11F1, le MAX232, le 4066. Quelques composants discrets supplémentaires définissent les niveaux de quelques broches du HC11.

Le *MAX232* a pour rôle de convertir les signaux TTL provenant des deux broches de communication série (TxD et RxD) à des niveaux ±12 V requis par le port RS232 du PC.

Le *4066* a pour rôle de passer du mode "chargement du programme du port série" au mode "exécution du programme stocké en EEPROM". En effet, le HC11 émet au boot (cf Annexe B, p.25) le caractère 0h00 sur la broche TxD, puis se met en écoute de ce qui se passe sur la broche RxD. Si le caractère 0h00 est reçu, l'exécution passe tout de suite au début de l'EEPROM. Si le caractère n'est pas 0h00 mais pas non plus 0hFF, alors le HC11 tente de changer de baud rate (fréquence de réception des données). Enfin, la réception d'un caractère 0hFF signifie qu'il faut recevoir les octets et les placer en RAM (adresse commençant en 0h0000), jusqu'à la fin de la communication. Ainsi, nous constatons que court-circuiter RxD et TxD au boot permet de passer tout de suite à l'EEPROM. Cependant, court-circuiter définitivement RxD et TxD fait perdre la possibilité d'utiliser le port série ultérieurement. Le rôle des switchs analogiques du 4066 est de commander le court circuit pendant seulement les 10 premières ms après le RESET en commandant les interrupteurs par un circuit RC (sous le 4066 : la capacité de 330 nF et la résistance de 100 kΩ. La résistance de 10 kΩ au-dessus du 4066 sert de résistance de pull-up). Ce court circuit est de plus permis par un strap : si ce strap est ouvert, il n'y a pas court circuit et le programme est chargé par le port série. Si ce strap est fermé, il y a court circuit et donc saut en début d'EEPROM.

Enfin, pour le *68HC11*, seulement quelques composants discrets externes sont nécessaires : le quartz et ses deux capacités de quelques pF pour le faire osciller (avec un résistance optionnelle de 10 MΩ), des résistances de pull up (en bas du HC11) sur les fils reliés à TxD, RxD et RESET. Il sera de plus nécessaire, en cas d'utilisation des interruptions, d'ajouter deux résistances de pull up entre les broches IRQ et Vdd et XIRQ et Vdd (sinon le HC11 est continuellement interrompu et le programme se bloque après exécution de l'instruction `cli`).

Les straps disponibles sont donc : sous le HC11 (horizontal) pour le RESET (il s'agit en fait plutôt de court-circuiter ces deux broches avec un objet conducteur pour effectuer le RESET), le strap (horizontal) à gauche du 4066 qui détermine s'il faut charger le programme sur le port série (ouvert) ou exécuté

celui stocké en EEPROM (fermé), et enfin les deux straps (verticaux) en haut du HC11 et qui doivent normalement être fermés pour lancer le HC11 en bootstrap mode.

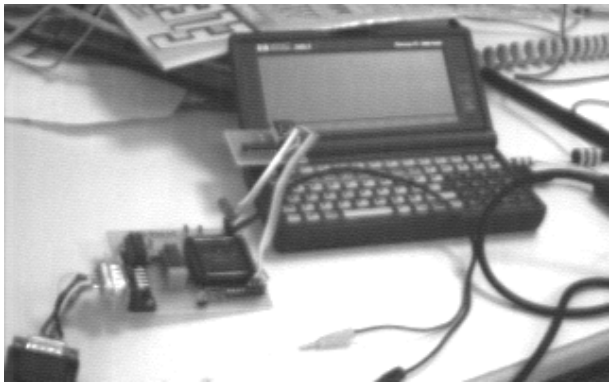
Nous prendrons soin, lors de la réalisation, de placer deux supports de jumpers de façon *verticale* en haut du circuit (sélection du mode de fonctionnement : les deux jumpers doivent être mis en place pour booter en bootstrap mode) ainsi qu'un support de jumper en bas du circuit pour le Reset (réalisé en court-circuitant temporairement ce support de jumper avec un objet conducteur électriquement).

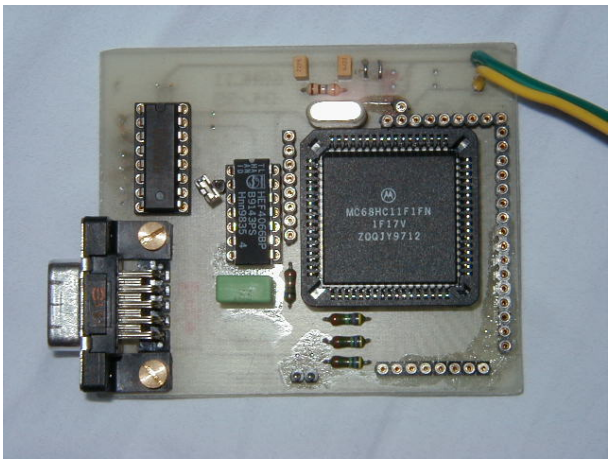
```
0000 8E 00 FF    lds    #0h00FF
0003 CE 10 00    ldx    #0h1000
0006 86 FF       ldaa   #0hFF
0008 A7 01       staa   0h01,x
```

```
000A 86 00       ldaa   #0h00
000C A7 00     boucle:staa 0h00,x
000E 18 CE FF FF ldy    #0hFFFF
0012 18 09     wait:    dey
```

```
0014 26 FC       bne    wait
0016 4C          inca
0017 20 F3       bra boucle
```

Ce petit programme de test fait défiler sur le port A (horizontal, en bas du circuit) la représentation binaire des nombres de 0 à 255. Le résultat est visualisable en connectant un afficheur 7 segments ou une barrette de LEDs sur le port A.





4 Lecture des valeurs du convertisseur analogique-numérique

Il faut :

1. mettre en route les convertisseurs A/D en écrivant dans le registre $0x1039$.[3].
2. pour chaque nouvelle conversion, écrire le numéro du convertisseur (entre 0 et 7) qui doit effectuer la conversion dans le registre $0x1030$ (dans l'exemple ci-dessous, nous utilisons le convertisseur numéro 3)
3. attendre le signal de fin de conversion (par polling du registre $0x1030$).
4. lire la valeur obtenue dans le registre $0x1034$.

```
0000 8E 00 FF   lds #0h00ff
0003 86 FF      ldaa #0hff
0005 B7 10 01   staa 0h1001
0008 86 90      ldaa #0h90
000A B7 10 39   staa 0h1039
000D          start:
```

```
000D 86 23      ldaa #0h23
000F B7 10 30   staa 0h1030
0012 8D 05      bsr ad_read
0014 B7 10 00   staa 0h1000
0017 20 F4      bra start
```

```
0019 B6 10 30 ad_read:ldaa 0h1030
001C 2A FB      bpl ad_read
001E B6 10 34   ldaa 0h1034
0021 39         rts
```

Cet exemple lit la valeur analogique sur une des voies du port E (au dessus du 68HC11) - le canal 3 - et renvoie la valeur sur le port A (auquel on aura relié un afficheur 7 segments par exemple). Une

3. Nous utiliserons indifféremment la notation $0xyyyy$ (notation C), $0hyyyy$ (notation assembleur non-Motorola 68HC11) ou $\$yyyy$ (notation Pascal) pour noter le nombre hexadécimal $yyyy$

variation de la tension appliquée sur la broche du port E (entre 0 et 5 V) doit se traduire par un variation de la valeur affichée.

5 La communication série

Il faut :

1. mettre le port D en mode communication
2. déterminer la vitesse et le protocole de transmission
3. une fois ces initialisations effectuées, attendre la transmission d'un caractère et le lire lorsqu'il a été reçu (par polling).

Dans les programmes qui suivent, le 68HC11 doit être relié par un câble direct (non-croisé) à un PC sur lequel tourne un logiciel d'émulation de terminal (type Terminal sous Windows 3.11 ou Procom sous DOS). Dans le premier exemple, le 68HC11 affiche sur le port A (auquel on aura relié un afficheur 7 segments par exemple) la valeur ASCII du caractère envoyé par le PC (par appui d'une touche du clavier du PC par exemple). Dans le second exemple, le 68HC11 envoie en continu les caractères ASCII affichables : le terminal doit continuellement afficher les signes de ponctuation, les chiffres et l'alphabet en minuscules et majuscules. Enfin, le dernier exemple combine ces deux applications pour attendre l'envoi d'un caractère du PC et le lui renvoyer.

Exemple de lecture des données venant sur le port série du 68HC11 :

```
0000 8E 00 FF     lds #0h00FF
0003 CE 10 00     ldx #0h1000
0006 86 FF        ldaa #0hFF
0008 A7 01        staa 0h01,x
000A 1C 28 20     bset 0h28,x,#0h20
```

```
000D 6F 2C        clr 0h2c,x
000F CC 33 0C     ldd #0h330C
0012 A7 2B        staa 0h2B,x
0014 E7 2D        stab 0h2D,x
0016 1F 2E 20 FC  rc: brclr 0h2E,x,#0h20,rc
```

```
001A E6 2F        ldab 0h2F,x
001C E7 00        stab 0h00,x
001E 20 F6        bra rc
```

Exemple d'émission de données sur le port série du 68HC11 :

```
0000 8E 00 FF            lds #0h00FF ; stack
0003 CE 10 00            ldx #0h1000
; config registers
0006 86 FF               ldaa #0hFF
; port A for output
0008 A7 01               staa 0h01,x ; |
000A 1C 28 20            bset 0h28,x,#0h20
```

```
; port D OR mode
000D 6F 2C               clr 0h2C,x ; set 2400 N81
000F CC 33 0C            ldd #0h330C ; |
0012 A7 2B               staa 0h2B,x ; |
0014 E7 2D               stab 0h2D,x ; |
0016 86 20               ldaa #0h20
; init value at ' '
```

```
0018 1F 2E 80 FC         boucle: brclr 0h2E,x,#0h80,boucle
001C A7 2F               staa 0h2F,x ; send char
001E 4C                  inca ; next char
001F 81 7B               cmpa #0h7B ; > 'z' ?
0021 26 F5               bne boucle
0023 86 20               ldaa #0h20 ; restart at ' '
0025 20 F1               bra boucle
```

Combinaison des deux programmes précédents pour lire une valeur sur le port série et la renvoyer au PC :

```
0000 8E 00 FF     lds #0h00FF
0003 CE 10 00     ldx #0h1000
0006 86 FF        ldaa #0hFF
0008 A7 01        staa 0h01,x
000A 1C 28 20     bset 0h28,x,#0h20
```

```
000D 6F 2C        clr 0h2C,x
000F CC 33 0C     ldd #0h330C
0012 A7 2B        staa 0h2B,x
0014 E7 2D        stab 0h2D,x
0016 1F 2E 20 FC  xm: brclr 0h2E,x,#0h20,xm
```

```
001A E6 2F        ldab 0h2F,x
001C E7 00        stab 0h00,x
001E 1F 2E 80 FC  rc: brclr 0h2E,x,#0h80,rc
0022 E7 2F        stab 0h2F,x
0024 20 F0        bra xm
```

6 Mise en EEPROM locale et exécution d'un programme

6.1 Aspect logiciel

L'EEPROM commence en $0xFE00$. Il nous faut donc, en bootloader mode, charger un programme en RAM dont la fonction est d'attendre le programme à stocker en EEPROM et exécuter la procédure de stockage. Nous avons choisi de réaliser la conversion hexadécimal-ASCII vers une valeur numérique

au niveau du PC (contrairement à l'AN1010 de Motorola qui réalise cette fonction au niveau du micro-contrôleur) et de n'envoyer que les octets correspondant au programme (pas de checksum, pas d'adresse de début ...).

```
0000 8E 00 FF     lds  #0h00FF ; stack
0003 CE 10 00     ldx #0h1000 ; config registers offset
0006 18 CE FE 00  ldy #0hFE00 ; start of EEPROM (cf 4.12 tech man)
000A 86 00        ldaa #0h00 ; bprot - block protection disabled
000C A7 35        staa 0h35,x ; |
000E 1C 28 20     bset 0h28,x,#0h20 ; port D wired OR mode : set port D
0011 6F 2C        clr 0h2C,x ; set 2400 N81
0013 CC 33 0C     ldd #0h330C ; |
0016 A7 2B        staa 0h2B,x ; |
0018 E7 2D        stab 0h2D,x ; |
001A 86 FF        ldaa #0hFF ; port A as output
001C A7 01        staa 0h01,x ; |
001E 86 22        ldaa #0h22 ; show '22' on display
0020 A7 00        staa 0h00,x ; |
0022      boucle:
0022 1F 2E 20 FC  brclr 0h2E,x,#0h20,boucle ; wait for datum
0026 E6 2F        ldab 0h2F,x ; store it in B
0028 E7 00        stab 0h00,x ; echo on port A
002A 86 16        ldaa #0h16 ; reset 1 byte
002C 8D 12        bsr program ; write to location in EEPROM
002E 86 02        ldaa #0h02 ; store 1 byte
```

```
0030 8D 0E        bsr program ; write to location in EEPROM
0032 18 08        iny ; goto next location in EEPROM
0034 18 8C FF BF  cpy #0hFFBF ; until we get to 0hFFBF
0038 26 E8        bne boucle ; get next datum if not finished
003A 86 08        ldaa #0h08 ; show '8' on display
003C A7 00        staa 0h00,x
003E 20 FE        fin: bra fin

0040      program: ; cf procedure p.4.16 of tech manual
0040 A7 3B        staa 0h3B,x
0042 18 E7 00     stab 0,y ; erase/program byte
0045 6C 3B        inc  0h3B,x
0047 3C   pshx
0048 CE 1A 0A     ldx #0h1A0A ; 2x0h0D05 (16 MHz Q) => 2xdelay_AN1010
004B 09   wait: dex
004C 26 FD        bne wait
004E 38   pulx ; put X=0h1000 back
004F 6A 3B        dec 0h3B,x
0051 6F 3B        clr 0h3B,x
0053 39   rts
```

Le programme `ee.asm` dont le rôle est de recevoir des données sur le port série (2400, N81) et de les stocker en EEPROM.

6.2 Aspect matériel

Nous avons vu plus haut qu'une série d'interrupteurs analogiques avaient été inclus sur la carte. Leur fonction est d'éventuellement permettre de court-circuiter pendant un temps bref les lignes `RxD` et `TxD` de communication de micro-contrôleur. En effet, si à l'allumage le 68HC11 est en bootstrap mode, il attend de recevoir un signal sur la broche `RxD` et agit en conséquence. Si ce signal est $0xFF$, le baudrate est correct et l'acquisition du programme à stocker commence. Si ce signal est $0x00$, l'exécution saute en début d'EEPROM : le programme qui avait été stocké là antérieurement est exécuté. Dans les autres cas, le micro-contrôleur change le baud rate.

Or à l'allumage, la première opération du programme du bootloader est d'envoyer $0x00$ sur la broche `TxD`. Ainsi, en court-circuitant temporairement `RxD` et `TxD`, nous lançons le programme en EEPROM plutôt qu'attendre l'acquisition d'un programme à exécuter par le port série.

Le court-circuit se fait simplement en connectant un circuit RC sur la commande d'un des interrupteurs analogiques elle même connectant la commande d'un autre interrupteur analogique à la masse. La commande de cette seconde porte logique est cependant maintenue à l'état haut par une résistance de pull-up de 10 kΩ. Au reset, le niveau de la broche `RESET` est bas. Cette broche est reliée à la commande du premier interrupteur analogique qui est donc ouvert, et la résistance de pull-up maintient le second interrupteur fermé, connectant `TxD` et `RxD`. Le circuit RC se charge, fermant le premier interrupteur qui relie la commande du second interrupteur à la masse et ouvre donc le contact entre `TxD` et `RxD`.

6.3 Aspect pratique

Le programme `hc11.pas` a été prévu pour éventuellement accepter deux arguments. Nous avons jusque là fourni un seul argument, le programme en hexadécimal à charger dans la RAM du 68HC11. Si nous donnons deux arguments, qui sont dans l'ordre le programme `ee.out` vu plus haut, suivi du nom du programme à charger en EEPROM (nommé ici `prg.out`), hc11.exe ee.out prg.out chargera dans un premier temps `ee.asm` dans la RAM du HC11 `ee` dont le rôle est d'attendre la réception de `prg.out` sur le port série et de le stocker en EEPROM. Ceci se fait par un délai d'une seconde entre la fin de

l'émission de `ee.out` et le début de l'émission de `prg.out` (délai suffisant pour démarrer `ee`). Il faudra de plus penser à relocaliser, à la compilation, `prg.asm` à l'adresse de début de l'EEPROM (0hFE00, puisque l'EEPROM va de 0hFE00 à 0hFFBF) par l'ajout des deux lignes suivantes en début de programme : `.area code (ABS)` et `.org 0hFE00`.

7 Branchement d'une RAM externe

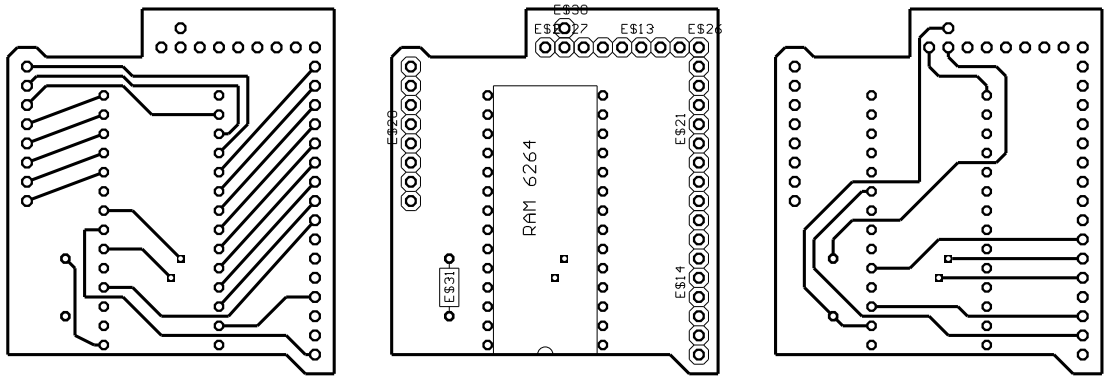
Dans le cas de la connexion d'un composant externe, il faut faire passer le 68HC11 en mode étendu de façon à ce que les ports F, B et C soient utilisés comme bus d'adresses (octets de poids faible et fort) et bus de données (sur 8 bits), et pour faire fonctionner le signal d'écriture `R/W#`. Cette opération se fait de façon logicielle : en effet, il nous faut tout d'abord être en bootstrap mode pour charger le programme en RAM interne du 68HC11 avant de pouvoir accéder à la RAM externe.

Le branchement de la RAM se fait facilement en reliant les bus d'adresses et de données et, pour éviter les risques de conflit avec les mémoires internes, nous avons relié les signaux d'adresse 13 et 14 aux signaux d'activation de la RAM pour faire commencer l'espace adressable de celle-ci à $0x2000$.

```
0000 8E 00 FF     lds #0h00FF
0003 CE 10 00     ldx #0h1000 ; config regs
0006 18 CE 20 00  ldy #0h2000 ; RAM offset
000A 86 FF        ldaa #0hFF ; set A port out
000C A7 01        staa 0h01,x ; |
000E 1C 28 20     bset 0h28,x,#0h20
; set SPI clock rate
0011 6F 2C        clr 0h2C,x ; 2400 bauds, N81
```

```
0013 CC 33 0C     ldd #0h330C ; |
0016 A7 2B        staa 0h2B,x ; |
0018 E7 2D        stab 0h2D,x ; |
001A 86 E5        ldaa #0hE5 ; set xtend mode
001C A7 3C        staa 0h3C,x ; |
001E 1F 2E 20 FC  xmi:  brclr 0h2E,x,#0h20,xmi
; read char
0022 E6 2F        ldab 0h2F,x ; |
```

```
0024 E7 00        stab 0h00,x ; store on port A
0026 18 E7 00     stab 0h00,y ; store in RAM
0029 18 E6 00     ldab 0h00,y ; read from RAM
002C 18 08        iny ; goto next RAM addr
002E 1F 2E 80 FC  rec:  brclr 0h2E,x,#0h80,rec
; echo char to rs232
0032 E7 2F        stab 0h2F,x ; |
0034 7E 00 1E     jmp xmi
```



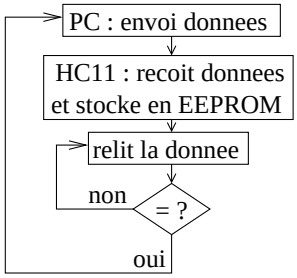
Ce circuit peut facilement être modifié pour y adapter une RAM de 32K 62256 : relever les broches 1, 20 et 22 de la RAM et relier 1 à la broche d'adresse 14 du 68HC11 tandis que les broches 20 et 22 sont reliées à la masse pour toujours activer la RAM.

Utilisation de ce circuit pour la programmation d'une EEPROM externe

Une EEPROM, la 28C64, comporte des branchements identiques à deux de la RAM que nous avons utilisé dans ce montage (6264). Il est ainsi aisé d'utiliser cette carte d'extension pour programmer de façon permanente (*i.e.* conservation des données en cas de coupure de courant) ce type d'EEPROM. La seule distinction que nous ayons identifié entre la 2864 et la 6264 est, lors du cycle d'écriture, la fonction de la broche `G#` (ou `OE#`) : cette broche semble pouvoir être à un niveau quelconque (bas dans notre cas) lors de l'écriture dans une RAM 6264, alors qu'il *doit* être au niveau haut lors de la phase d'écriture dans une EEPROM 2864. Il se trouve que cette broche, `G#` ou `OE#`, est connectée ici dans une fonction de décodage d'adresse au bit de poids le plus élevé d'adresse du 68HC11 (adresse 15). Ainsi, pour faire passer `OE#` au

niveau haut, il ne faut pas écrire à un offset de $0x2000$ (pour lequel `OE#` est bas) mais $0x2000 + 0x8000$ ($0x8000$ s'écrit 1000 0000 0000 0000 en binaire, *i.e.* met `OE#` au niveau haut). Nous distinguons donc deux phases d'accès au registe d'adresse X de l'EEPROM : l'écriture en $0x8000 + 0x2000 + X = 0xA000 + X$ et une lecture en $0x2000 + X$ (comme dans le cas de la RAM). Ce même circuit a été testé avec une EEPROM 2817 (2 KB), courante en récupération de composants : la programmation est possible mais nécessite un temps plus long que la 2864 (modèle ancien de 2817A utilisé ici : Intel '83). `hc11_ee64wri` fonctionne correctement après avoir remplacé le délai (fonction `delai`) de 0h0FFF (pour la 2864) en 0hFFFF (pour la 2817). `hc11_ee64prg` n'a pas fonctionné avec la 2817 pour des raisons que nous n'avons pas cherché à analyser (peut être la vitesse de transfert de données par le port série – 2400 bauds – est trop rapide par rapport au temps d'écriture dans l'EEPROM et les données transmises pour stockage sont perdues).

Un algorithme légèrement plus complexe a été testé avec succès lors de la programmation d'une EEPROM 28C256 (voir plus bas pour l'aspect matériel lié à cette EEPROM) : après avoir écrit dans l'EEPROM l'octet reçu du PC par le port série, le programme relit la valeur stockée dans l'EEPROM jusqu'à obtenir le résultat désiré, et ne passe à l'octet suivant qu'à ce moment là (la synchronisation du PC se fait en attendant la confirmation de l'écriture de l'octet en EEPROM avant d'envoyer le caractère suivant). Ce principle est illustré dans la figure ci-contre.



Ce dernier algorithme est celui implémenté dans les programmes `hc11_ee64.c` (sous Linux) et `hc11_ee64prg.asm` (sur le HC11) pour la programmation des EEPROM. Attention : étant donné que `hc11_ee64.c` lit les valeurs écrites dans l'EEPROM avant d'envoyer une nouvelle donnée, le port série doit être libre pour la lecture. Il ne faut donc *pas* lancer `hc11rec` en même temps que `hc11_ee64` sous peine de bloquer ce dernier, et donc l'écriture en EEPROM.

Nous avons développé deux applications : d'une part le stockage par le 68HC11 de données dans l'EEPROM pour une lecture ultérieure, ces données étant conservées même si une coupure de courant se produit entre le stockage et la lecture, et d'autre part la lecture de données sur le port série (émise par un PC) pour être stockées dans l'EEPROM. Ce second exemple permet d'étendre les 512 octets d'EEPROM interne du 68HC11F1 (en introduisant un `jmp 0x2000` dans le code en EEPROM interne pour passer à l'exécution d'opcodes 68HC11 stockés dans la 2864), ou de programmer l'EEPROM externe avec des opcodes pour un autre processeur (x86 ou Z80 par exemple) pour remplacer une EPROM (type 2764).

Ce programme, `hc11_ee64wri.asm`, génère une rampe décroissante et la stocke dans l'EEPROM :

```
8E 00 FF    lds     #0h00FF
CE 10 00    ldx     #0h1000         ; config registers offset
1C 01 FF    bset    0h01,x,#0hff    ; set A port for output
1C 28 20    bset    0h28,x,#0h20
6F 2C       clr     0h2C,x          ; 2400 bauds, N81
CC 33 0C    ldd     #0h330C         ; |
A7 2B       staa    0h2B,x          ; |
E7 2D       stab    0h2D,x          ; |

86 E5       ldaa    #0hE5           ; set extended mode
A7 3C       staa    0h3C,x          ; |

18 CE 20 00 ldy     #0h2000         ; RAM offset : 001x yyyy yyyy yyyy
C6 FF       ldab    #0hff
18 8F  boucle:xgdy
C3 80 00    addd    #0h8000         ; OE# MUST be hi => set 1xxx xxxx => @ += 8000
18 8F       xgdy
18 E7 00    stab    0h00,y
8D 1E       bsr     delai
18 8F       xgdy

83 80 00    subd    #0h8000
18 8F       xgdy
18 A6 00    ldaa    0h00,y
8D 0B       bsr     snd
5A          decb
18 08       iny
18 8C 40 30 cmpy    #0h4030 ; ED00+10 pour verifier que les 16 last vals=FF
26 DF       bne     boucle
20 FE   fin:bra     fin

1F 2E 80 FC snd:brclr   0h2E,x,#0h80,snd
A7 2F       staa    0h2F,x          ; echo char to serial port
39          rts

18 3C delai:pshy
18 CE 0F FF ldy     #0h0FFF
18 09   del:dey
26 FC       bne     del
18 38       puly
39          rts
```

Ce programme, `hc11_ee64prg.asm`, reçoit des données d'un PC par le port série et les stocke en EE-

PROM :

```
8E 00 FF    lds #0h00FF
CE 10 00    ldx #0h1000      ; config registers offset
1C 01 FF    bset 0h01,x,#0hff ; set A port for output
1C 28 20    bset 0h28,x,#0h20
6F 2C       clr 0h2C,x       ; 2400 bauds, N81
CC 33 0C    ldd #0h330C      ; <- changed to 1200 bauds
A7 2B       staa 0h2B,x       ; |
E7 2D       stab 0h2D,x       ; |
86 E5       ldaa #0hE5        ; set extended mode
A7 3C       staa 0h3C,x       ; |
18 CE 20 00 ldy #0h2000      ; RAM offset : 001x yyyy yyyy yyyy
18 8F  boucle: xgdy
C3 80 00    addd #0h8000 ; OE# MUST be hi => set 1xxx xxxx => @ += 8000
18 8F       xgdy
8D 24       bsr  rc           ; recoit la donnee a stocker
18 E7 00    stab 0h00,y
8D 26       bsr delai
18 8F       xgdy
83 80 00    subd #0h8000
18 8F       xgdy
```

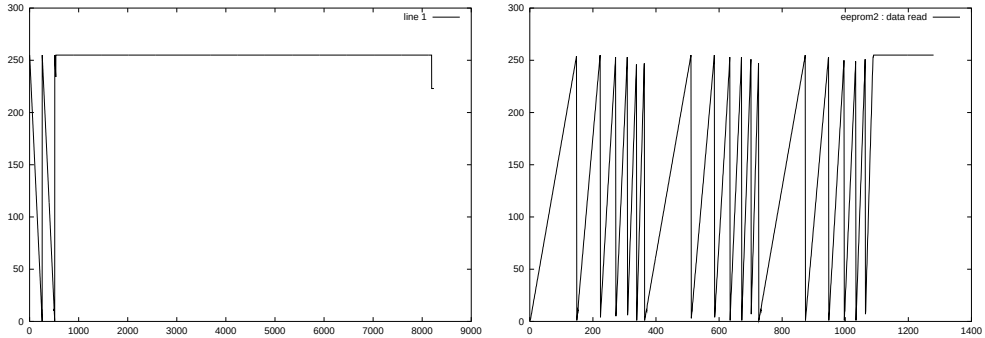
```
18 E1 00 bcl:cmpb 0h00,y      ; B=donnee a stocker, cmp a donnee en RAM
26 FB       bne bcl          ; si different : rate :(
8D 0A       bsr snd
18 08       iny
18 8C 40 00 cmpy #0h4000      ; taille de l'EEPROM
26 DC       bne boucle
20 FE   fin:bra fin
1F 2E 80 FC snd:brclr 0h2E,x,#0h80,snd
E7 2F       stab 0h2F,x       ; echo char to serial port
39          rts
1F 2E 20 FC rc: brclr 0h2E,x,#0h20,rc ; recoit donnee et la stocke dans B
E6 2F       ldab 0h2F,x
39          rts
18 3C delai:pshy
18 CE 2F FF ldy #0h2FFF
18 09   del:dey
26 FC       bne del
18 38       puly
39          rts
```

Ce programme, `hc11_ee64rea.asm`, lit les données dans l'EEPROM (stockées par un des deux programmes précédents) et les envoie sur le port série pour affichage et traitement par un PC :

```
8E 00 FF    lds #0h00FF
CE 10 00    ldx #0h1000             ; config registers offset
86 FF       ldaa #0hFF              ; set A port for output
A7 01       staa 0h01,x             ; |
1C 28 20    bset 0h28,x,#0h20       ; set SPI clock rate ?!
6F 2C       clr 0h2C,x              ; 2400 bauds, N81
CC 33 0C    ldd #0h330C             ; |
A7 2B       staa 0h2B,x             ; |
E7 2D       stab 0h2D,x             ; |

86 E5       ldaa #0hE5              ; set extended mode
A7 3C       staa 0h3C,x             ; |
```

```
18 CE 20 00 ldy #0h2000             ; RAM offset : 1yyy yyyy yyyy yyyy
18 E6 00  boucle2:ldab 0h00,y
8D 0C       bsr snd
E7 00       stab    0h00,x
18 08       iny
18 8C 40 30 cmpy #0h4030
26 F1       bne boucle2
20 FE   fin:bra fin

1F 2E 80 FC snd:brclr 0h2E,x,#0h80,snd  ; echo char to serial port
E7 2F       stab 0h2F,x             ; |
39          rts
```

Les données ainsi transmises peuvent être lues sous Linux au moyen de `hc11rec` présenté au chapitre 14.



À gauche : stockage de données générées par le 68HC11 (rampes) dans l'EEPROM. Noter que nous n'avons généré que deux rampe : la valeur par défaut des registres de l'EEPROM est 255, et nous avons lu quelques valeurs en plus de la taille de l'EEPROM pour vérifier un changement des valeurs lues (et vérifier la fin de la zone d'adressage de l'EEPROM), d'où la variation de niveau de la courbe sur la fin. À droite : stockage de données issues du PC (transmises par RS232) dans l'EEPROM.

Le logiciel `hc11_ee64` de programmation de l'EEPROM externe à partir de Linux (utilise `rs232.c` et `rs232.h` présentés à la fin de ce document, en chapitre 14). Il s'utilise par `./hc11_ee64 hc11_ee64prg.out programme.out` pour stocker `programme.out` dans l'EEPROM externe :

```
#include "rs232.h"

char to_ne2000(char a)          // xchange bits 1/2, 3/4, 5/6 & keep bits 0 & 7
{char tmp6,tmp5,tmp4,tmp3,tmp2,tmp1,tmp;
 tmp1=(a & 0x02)/0x02;
 tmp2=(a & 0x04)/0x04;
 tmp3=(a & 0x08)/0x08;
 tmp4=(a & 0x10)/0x10;
 tmp5=(a & 0x20)/0x20;
```

```
 tmp6=(a & 0x40)/0x40;
 tmp=(a&0x01)+(a&0x80) + tmp1*0x04+tmp2*0x02 + tmp3*0x10+tmp4*0x08
                       +tmp5*0x40+tmp6*0x20;
 return(tmp);
}

void send_hc11(FILE *f,int fd)
{char buf[1024];int i=0,status;unsigned int j;
 buf[0]=255;write(fd,buf,1);                          /* start with 'FF' */
```

```
 do {do {status=fscanf(f,"%x",&j);buf[i]=(char)j;i++;} /* read while !EOF */
         while ((status!=EOF) && (i<1024));     /* && less than 255 chars */
     if (i<1024) i--;
     write(fd,buf,i);printf ("%d bytes sent\n",i);i=0;}
 while (status!=EOF);
}

void send_data(FILE *f,int fd)
{char buf;int status;unsigned int j;
 do {status=fscanf(f,"%x",&j);buf=to_ne2000((char)j);write(fd,&buf,1);
    printf ("%d=%d => ",j&0x000000FF,((int)buf)&0x000000FF);fflush(stdout);
read(fd,&buf,1);printf ("%d=$%x\n",((int)buf)&0x000000FF,((int)buf)&0x000000FF);
    }
 while (status!=EOF);                                  /* read while !EOF */
}

int main(int argc,char **argv)
{int fd;FILE *f;
 if (argc<3) printf("%s filename[hc11_ee64prg.out] program\n",argv[0]); else {
    fd=init_rs232(BAUDRATE);
    f=fopen(argv[1],"r");send_hc11(f,fd);fclose(f);
    close(fd); fd=init_rs232(BAUDRATE);   //  <- clear RS232 buffer
    sleep(1);
    f=fopen(argv[2],"r");send_data(f,fd);fclose(f);
 /* free_rs232(); */
 }
 return(0);
}
```

Deux légères modifications sont nécessaires pour la programmation des 8 premiers kilooctets d'une EEPROM 28C256 (il est éventuellement possible de programmer l'intégralité d'une 28C256 en la programmant par bancs, les broches d'adresse A14 et A13 étant alors reliées au port A par exemple) : la broche 1 du support d'EEPROM doit être reliée à la masse (présente sur la broche 14 du support par exemple) par une résistance (de 5,6 kΩ par exemple) et un fil doit relier la broche 26 du support à la masse (1 et 26 sont, dans le cas de la 28C256, les lignes d'adresses A13 et A14, que nous relions ainsi à la masse tout en préservant la compatibilité avec les autres composants – RAM 6264 ou EEPROM 28C64 – utilisés jusqu'ici. Pour une programmation par bancs, relier ces broches au port A au lieu de la masse).

Résumé des divers types de composants utilisés et des différences dans les assignements de broches (les EPROM n'ont été utilisées que en lecture) :

Type de composant	RAM	RAM	EEPROM	EEPROM	EPROM	EPROM
Référence	6264	62256	2864	28256	2764	27256
Broche 1	NC	A14=GND	RDY=GND	A14=GND	Vpp=+5V	Vpp=+5V
Broche 26	CS=+5V	A13=GND	NC	A13=GND	NC	A13=GND
Broche 27	W#		W#	W#	PGM#=+5V	A14=GND

En conclusion, nous pouvons au moyen du circuit d'ajout d'une RAM externe :
– augmenter la quantité de RAM disponible de 8 KB (puisque les lignes d'adresse 0 à 12 servent à l'adressage, les lignes 13 à 15 étant utilisées pour le décodage d'adresse)
– programmer des EEPROM de taille allant jusqu'à 8 KB en reliant éventuellement la broche 1 à la masse *via* une résistance mais surtout en mettant la broche 26 à la masse
– lire une EPROM 27256 en mettant la broche 1 à +5 V et les broches 26 et 27 à la masse.

Ce logiciel a été utilisé avec succès pour programmer des EEPROM utilisés dans un module autonome à base de TMP-Z84 (version microcontroleur du Z80) et une carte de programme pour le Gameboy.

8 Résultat final : acquisition de données, stockage en RAM puis envoie par port série

L'objectif du travail présenté jusqu'ici était de réaliser un système capable de faire seul un grand nombre (> 1000) d'acquisitions pendant une durée déterminée à l'avance (de l'ordre de quelques heures), et une fois les données obtenues d'attendre un signal extérieur (sur le port série) pour envoyer ces données à un PC qui s'occupera du traitement des données.

Le programme suivant :

– initialise le port de communication et les convertisseurs A/D puis passe en mode étendu pour pouvoir accéder à la RAM externe.

– lit périodiquement la valeur analogique sur le convertisseur 3 et stocke la valeur observée en RAM externe tout en affichant la valeur lue sur le port A.

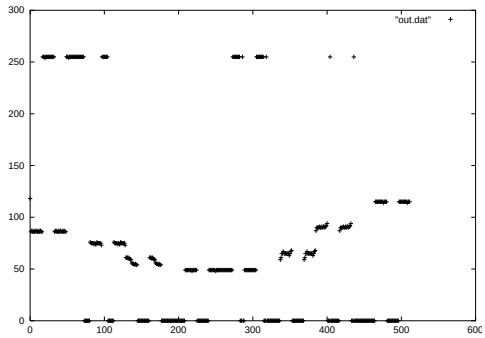
– une fois la RAM externe pleine, attend un caractère (quelconque) sur le port série, et affiche “00” sur le port A pour indiquer l’attente.

– envoie les données sur le port série lorsque la requête a été faite et affiche la valeur du caractère reçu sur le port A pour indiquer que la transmission est en cours.

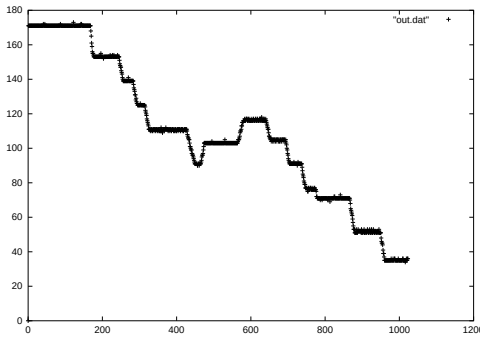
Il faudra se rapporter à l’annexe 15 pour le programme en Turbo Pascal de réception des données de la RAM liée au HC11 vers l’IBM PC une fois l’acquisition finie.

```
0000 8E 01 50    lds #0h0150      ; stack
0003 CE 10 00    ldx #0h1000      ; config regis
0006 18 CE 20 00 ldy #0h2000      ; RAM offset
000A 86 FF       ldaa #0hFF       ; port A: output
000C A7 01       staa 0h01,x      ; |
000E 1C 28 20    bset 0h28,x,#0h20; port D OR mode
0011 6F 2C       clr 0h2C,x       ; 2400 bauds, N81
0013 CC 33 0C    ldd #0h330C      ; |
0016 A7 2B       staa 0h2B,x      ; |
0018 E7 2D       stab 0h2D,x      ; |
001A 86 E5       ldaa #0hE5       ; set extended mode
001C A7 3C       staa 0h3C,x      ; |
001E 86 90       ldaa #0h90       ; enable A/DC
0020 A7 39       staa 0h39,x      ; store OPTION reg

0022 CE 3F FF  ad: ldx #0h3FFF
0025 09        loop:dex           ; wait before next conv
0026 26 FD       bne loop
0028 CE 10 00    ldx #0h1000
002B 86 23       ldaa #0h23       ; start conversion on AD3
002D A7 30       staa 0h30,x      ; |
002F 8D 30       bsr ad_read
0031 18 A7 00    staa 0h00,y      ; store in RAM
0034 A7 00       staa 0h00,x      ; echo on port A
```

```
0036 18 08       iny              ; goto next RAM address
0038 18 8C 24 00 cpy #0h2400      ; 256 values
003C 26 E4       bne ad           ; get next datum
003E 86 00       ldaa #0h00       ; displ. result on A
0040 A7 00       staa 0h00,x      ; |

0042 1F 2E 20 FC rcv: brclr 0h2E,x,#0h20,rcv; wait for 1 char
0046 A6 2F       ldaa 0h2F,x      ; display on portA
0048 A7 00       staa 0h00,x

004A 18 CE 20 00 ldy #0h2000      ; RAM offset
004E 18 A6 00  read_ra:ldaa 0h00,y; read from RAM
0051 1F 2E 80 FC snd:brclr 0h2E,x,#0h80,snd
0055 A7 2F       staa 0h2F,x      ; echo char to serial port
0057 18 08       iny              ; inc RAM @
0059 18 8C 24 00 cpy #0h2400      ; end of filled RAM ?
005D 26 EF       bne read_ra      ; get next datum
005F 20 FE     fin:       bra fin

0061 A6 30       ad_read:ldaa 0h30,x
0063 2A FC       bpl ad_read      ; wait convert end
0065 A6 34       ldaa 0h34,x      ; return res in A
0067 39          rts
```



Exemple d’acquisition avec un fil du bus d’adresses de la RAM déconnecté (@5)



Exemple d’acquisition correcte

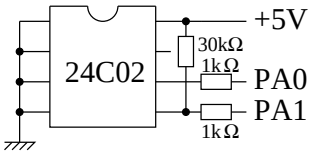
9 Communication avec un composant I2C (EEPROM)

Le protocole I2C développé par Phillips permet la communication entre divers composants via seulement 2 fils (horloge et données). Ce protocole est relativement complexe et permet par exemple d’avoir plusieurs maîtres sur le bus. Nous ne nous intéresserons ici que à un sous-ensemble de ce protocole où le 68HC11 est maître et un composant I2C est esclave : dans notre cas il s’agit d’une EEPROM de 256 octets 24C02.

Le protocole I2C contient 3 étapes : le début d’une séquence (START) défini par un front montant sur le fil de données tandis que le fil d’horloge est au niveau haut, la fin d’une séquence (STOP) défini par un front descendant sur le fil de données tandis que le fil d’horloge est au niveau haut, et finalement le transfert d’une donnée défini par un niveau stable sur le fil de données pendant que l’horloge est au niveau haut. Les transitions sur le fil de données pendant que le fil d’horloge est au niveau bas n’ont

aucune importance (du point de vue du protocole I2C) et permettent de replacer la valeur sur le fil de données en prévision de la prochaine opération à effectuer.

Les deux exemples qui suivent permettent, dans un premier temps, d'écrire en série jusqu'à 16 octets consécutifs en EEPROM (celle-ci est divisée en pages de 16 octets pour lequelles il est possible de ne définir que la première adresse de stockage, puis d'envoyer en série jusqu'à 16 données sans avoir à redéfinir le type d'opération en cours (écriture) ou l'adresse).



Circuit électronique utilisé pour tester le programme de communication avec l'EEPROM (PA est le port A du 68HC11F1)

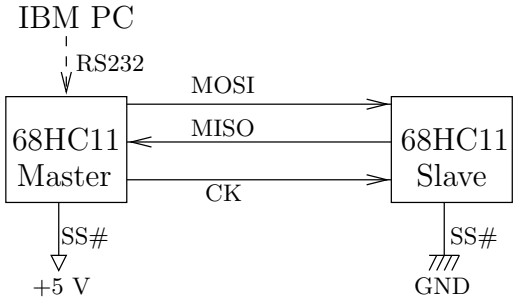
```
; RAM : programm uses           : 0->d241=F1
;       temporary datum storage : FF
;       stack                   : ...<-2FF
; I2C : PA0=CK, PA1=DATA  (ctrl packet format : 1010 000 R/W#), ck active hi

8E 02 FF     lds #0h02FF       ; stack decreases ...
CE 10 00     ldx #0h1000
86 FF        ldaa #0hff
A7 01        staa 0h01,x       ; port A output
1D 00 FF     bclr 0h00,x,#0hff ; port A=lo all <- ck=data=lo
1C 28 20     bset 0h28,x,#0h20 ; setup serial port
6F 2C        clr 0h2C,x        ; |
CC 33 0C     ldd #0h330C       ; |
A7 2B        staa 0h2B,x       ; |
E7 2D        stab 0h2D,x       ; |
;---------------------------------------
8D 44  begin:bsr   i2c_start
86 A0        ldaa  #0hA0     ; 1010 000W#=$A0
8D 71        bsr   i2csnd
86 00        ldaa  #0h00     ; word address
8D 6D        bsr   i2csnd

86 0F        ldaa  #0h0f     ; on va ecrire 15 donnees (on peut ecrire max
;            ldaa  #0h00     ;              16 donnees d'affilee = 1 page)
    bclsn:  ;bsr rc           ; read datum from serial port
8D 69        bsr   i2csnd
4A           deca
           ; inca
           ; cmpa  #0h0F
26 FB        bne bclsn
8D 51        bsr   i2c_stop        ; SEND STOP
;-------------------------------------------------------------
8D 46        bsr     delai
           ; bsr   delai
           ; bsr   delai
8D 2F        bsr   i2c_start
86 A0        ldaa  #0hA0   ; 1010 000R=$A0 : write adress to start reading from
8D 5C        bsr   i2csnd
86 00        ldaa    #0h00   ; word address
8D 58        bsr     i2csnd
8D 25        bsr     i2c_start
86 A1        ldaa    #0hA1   ; 1010 000R=$A1 : start reading
8D 52        bsr     i2csnd
C6 0F        ldab    #0h0F   ; on va lire 16 donnees
86 00  bclrc:ldaa    #0h00   ; NOT LAST BYTE read ... => ACK
BD 00 C7     jsr     i2crcv  ; modifies A
96 FF        ldaa    0hFF
8D 0F        bsr     xm      ; verification de la lecture ...
5A           decb
26 F4        bne     bclrc

86 02        ldaa    #0h02   ; LAST BYTE read ... => DONT ACK
8D 77        bsr     i2crcv  ; modifies A
96 FF        ldaa    0hFF
8D 04        bsr     xm      ; verification de la lecture ...

8D 27        bsr     i2c_stop
20 FE    fin:bra fin
;-------------------------------------------------------------
1F 2E 80 FC xm:brclr 0h2E,x,#0h80,xm
A7 2F          staa 0h2F,x
39             rts

             ;rc:    brclr 0h2E,x,#0h20,rc
             ;       ldaa 0h2f,x
             ;       rts

     i2c_start:          ; SEND START
1C 00 02       bset    0h00,x,#0h02    ; set DATA hi
8D 10          bsr     delai
1C 00 01       bset    0h00,x,#0h01    ; set ck hi
8D 0B          bsr     delai
1D 00 02       bclr    0h00,x,#0h02    ; lower DATA = START
8D 06          bsr     delai
1D 00 01       bclr    0h00,x,#0h01    ; lower CK
8D 01          bsr     delai
39             rts

18 CE 02 FF delai:ldy #0h02FF
18 09         del:    dey
26 FC                 bne del
39                    rts

      i2c_stop:          ; SEND STOP
1D 00 02       bclr    0h00,x,#0h02    ; data=lo
8D F2          bsr     delai
1C 00 01       bset    0h00,x,#0h01    ; ck=hi
8D ED          bsr     delai
1C 00 02       bset    0h00,x,#0h02    ; raise data
8D E8          bsr delai
1D 00 01       bclr    0h00,x,#0h01    ; lower ck
39             rts

97 FF   i2csnd:staa    0hFF            ; parameter in A (data to be sent)
36             psha
37             pshb                    ; ne pas modifier B
C6 08          ldab    #0h08           ; get ready for 8 bits transfer ...
12 FF 80 05 brs:brset   *0hFF,#0h80,hi  ; transmet la donnee stockee en FE
1D 00 02       bclr    0h00,x,#0h02    ; DATA=lo
20 03          bra     lo_done
1C 00 02    hi:bset    0h00,x,#0h02    ; DATA=hi
1C 00 01 lo_done:bset   0h00,x,#0h01    ; raise CK
8D CD          bsr     delai
1D 00 01       bclr    0h00,x,#1       ; lower CK
8D C8          bsr     delai
96 FF          ldaa    0hFF
48             lsla
97 FF          staa    0hFF
5A             decb
26 E2          bne brs
1D 01 02       bclr    0h01,x,#0h02    ; set DATA for input
1C 00 01       bset    0h00,x,#1       ; raise CK  <- ACK
8D B8          bsr     delai
1D 00 01       bclr    0h00,x,#1       ; lower CK  <- ACK
8D B3          bsr     delai
1C 01 02       bset    0h01,x,#0h02    ; set DATA for output
33             pulb
32             pula
39             rts

37      i2crcv:pshb
36             psha                    ; A=02 if last byte read, 00 otherwise
1D 01 02       bclr    0h01,x,#0h02    ; set DATA for input
C6 08          ldab    #0h08           ; transmet la donnee stockee en FE
15 FF FF       bclr    *0hFF,#0hFF     ; raz de FF (utile : on bset mais pas bclr)
96 FF      brr:ldaa    0hFF
48             lsla
97 FF          staa    0hFF
1C 00 01       bset    0h00,x,#1       ; raise CK
8D 99          bsr     delai
1F 00 02 03    brclr   0h00,x,#0h02,i2clo
14 FF 01       bset    *0hFF,#0h01
1D 00 01 i2clo:bclr    0h00,x,#1       ; lower CK
8D 8D          bsr     delai
5A             decb
26 E7          bne     brr
1C 01 02       bset    0h01,x,#0h02    ; set DATA for output           MUST NOT ACK ON
32             pula                    ;                       IF WE ARE READING LAST BYTE
A7 00          staa    0h00,x          ; A=00 if not last byte, 02 if last byte
1C 00 01       bset    0h00,x,#0h01    ; raise CK  <- ACK
BD 00 74       jsr     delai
1D 00 01       bclr    0h00,x,#0h01    ; lower CK  <- ACK
BD 00 74       jsr     delai
33             pulb
39             rts
```

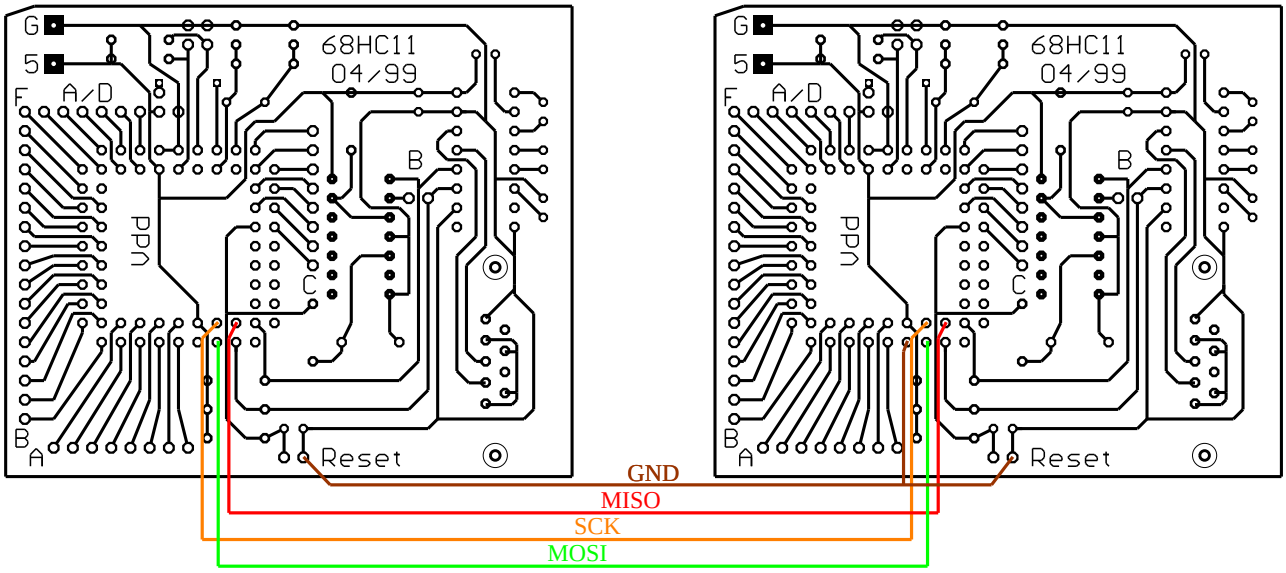
Pour palier à la limitation de ne pouvoir écrire que dans les 16 premiers octets (premier programme d'exemple trop simple), ce second programme écrit les données en EEPROM en précisant à chaque fois à quel emplacement mémoire accéder et quelle donnée y stocker. Ce mode de communication est

beaucoup plus lourd, mais permet avec un programme simple d'accéder aux 256 octets de la mémoire. Un programme mélangeant ces deux exemples est envisageable si l'encombrement du bus I2C peut poser un problème.

```
; RAM : programm uses           : 0->d241=F1
;       temporary datum storage : FF
;       stack                   : ...<-2FF
; I2C : PA0=CK, PA1=DATA  (ctrl packet format : 1010 000 R/W#), ck active hi
; SINGLE DATUM WRITE/READ (no sequential access)

8E 02 FF       lds #0h02FF             ; stack decreases ...
CE 10 00       ldx #0h1000
86 FF          ldaa #0hff
A7 01          staa 0h01,x             ; port A output
1D 00 FF       bclr 0h00,x,#0hff       ; port A=lo all            <- ck=data=lo
1C 28 20       bset 0h28,x,#0h20       ; setup serial port
6F 2C          clr 0h2C,x              ; |
CC 33 0C       ldd #0h330C             ; |
A7 2B          staa 0h2B,x             ; |
E7 2D          stab 0h2D,x             ; |
;-------------------------------------------------------
; start - write A0 - write @ - write datum - stop
C6 2F          ldab    #0h2F           ; on va ecrire 48 donnees (on peut ecrire max
8D 3C    begin:bsr       i2c_start
86 A0          ldaa    #0hA0           ; 1010 000W#=$A0
8D 69          bsr     i2csnd
17             tba                     ; ldaa #0h00 word address
8D 2E          bsr     xm      ; verification de la lecture ...
8D 64          bsr     i2csnd
               ;       bsr rc  ; read datum from serial port
               ; A still holds B (which is @ && datum) <- A=255-B ?
8B 10          adda    #0h10
8D 60          bsr     i2csnd
8D 4B          bsr     i2c_stop        ; SEND STOP
5A             decb
26 EC          bne begin

;-------------------------------------------------------
8D 3D          bsr     delai
               ; bsr   delai
               ; bsr   delai

C6 35          ldab    #0h35   ; on va lire 53 donnees
8D 24     beg2:bsr     i2c_start
86 A0          ldaa    #0hA0   ; 1010 000R=$A0 : write adress to start reading from
8D 51          bsr     i2csnd
17             tba             ; adress=B but transfered in A ; ldaa   #0h00 word @
8D 4E          bsr     i2csnd

8D 1B          bsr     i2c_start
86 A1          ldaa    #0hA1   ; 1010 000R=$A1 : start reading
8D 48          bsr     i2csnd

86 02          ldaa    #0h02   ;  DO NOT SEND ACK
BD 00 C1       jsr     i2crcv  ; modifies A
96 FF          ldaa    0hFF
8D 07          bsr     xm      ; verification de la lecture ...
8D 2A          bsr     i2c_stop
5A             decb
26 E3          bne     beg2
20 FE      fin:bra fin
;-------------------------------------------------------

1F 2E 80 FC xm:brclr 0h2E,x,#0h80,xm
A7 2F          staa 0h2F,x
39             rts

       ;rc:    brclr 0h2E,x,#0h20,rc
       ;       ldaa 0h2f,x
       ;       rts

     i2c_start:                ; SEND START
1C 00 02       bset    0h00,x,#0h02    ; set DATA hi
8D 10          bsr     delai
1C 00 01       bset    0h00,x,#0h01    ; set ck hi
8D 0B          bsr     delai
1D 00 02       bclr    0h00,x,#0h02    ; lower DATA = START
8D 06          bsr     delai
1D 00 01       bclr    0h00,x,#0h01    ; lower CK
8D 01          bsr     delai
39             rts

18 CE 02 FF delai:ldy #0h02FF
18 09       del:dey
26 FC          bne del
39             rts

      i2c_stop:                       ; SEND STOP
1D 00 02       bclr    0h00,x,#0h02    ; data=lo
8D F2          bsr     delai
1C 00 01       bset    0h00,x,#0h01    ; ck=hi
8D ED          bsr     delai
1C 00 02       bset    0h00,x,#0h02    ; raise data
8D E8          bsr delai
1D 00 01       bclr    0h00,x,#0h01    ; lower ck
39             rts

97 FF   i2csnd:staa     0hFF            ; parameter in A (data to be sent)
36             psha
37             pshb                    ; ne pas modifier B
C6 08          ldab    #0h08           ; get ready for 8 bits transfer ...
12 FF 80 05 brs:brset   *0hFF,#0h80,hi  ; transmet la donnee stockee en FE
1D 00 02       bclr    0h00,x,#0h02    ; DATA=lo
20 03          bra     lo_done
1C 00 02    hi:bset    0h00,x,#0h02    ; DATA=hi
1C 00 01 lo_done:bset    0h00,x,#0h01    ; raise CK
8D CD          bsr     delai
1D 00 01       bclr    0h00,x,#1       ; lower CK
8D C8          bsr     delai
96 FF          ldaa    0hFF
48             lsla
97 FF          staa    0hFF
5A             decb
26 E2          bne brs
1D 01 02       bclr    0h01,x,#0h02    ; set DATA for input
1C 00 01       bset    0h00,x,#1       ; raise CK  <- ACK
8D B8          bsr     delai
1D 00 01       bclr    0h00,x,#1       ; lower CK  <- ACK
8D B3          bsr     delai
1C 01 02       bset    0h01,x,#0h02    ; set DATA for output
33             pulb
32             pula
39             rts

37      i2crcv:pshb
36             psha                    ; A=02 if last byte read, 00 otherwise
1D 01 02       bclr    0h01,x,#0h02    ; set DATA for input
C6 08          ldab    #0h08           ; transmet la donnee stockee en FE
15 FF FF       bclr    *0hFF,#0hFF     ; raz de FF (utile : on bset mais pas bclr)
96 FF      brr:ldaa    0hFF
48             lsla
97 FF          staa    0hFF
1C 00 01       bset    0h00,x,#1       ; raise CK
8D 99          bsr     delai
1F 00 02 03    brclr   0h00,x,#0h02,i2clo
14 FF 01       bset    *0hFF,#0h01
1D 00 01 i2clo:bclr    0h00,x,#1       ; lower CK
8D 8D          bsr     delai
5A             decb
26 E7          bne     brr
1C 01 02       bset    0h01,x,#0h02    ; set DATA for output           MUST NOT ACK ON
32             pula                    ;                    IF WE ARE READING LAST BYTE
A7 00          staa    0h00,x          ; A=00 if not last byte, 02 if last byte
1C 00 01       bset    0h00,x,#0h01    ; raise CK  <- ACK
BD 00 6E       jsr     delai
1D 00 01       bclr    0h00,x,#0h01    ; lower CK  <- ACK
BD 00 6E       jsr     delai
33             pulb
39             rts
```

10 La communication SPI





Connexions à ajouter aux circuits pour faire connecter deux 68HC11 en mode SPI

La sélection de l'esclave à qui le message est envoyé se fait par l'activation de `SS#` qui, au lieu d'être connectée à la masse, est reliée à une broche du port B. Dans notre exemple où nous avons simplement un maître envoyant un message à un seul esclave, nous avons directement relié SS# de l'esclave à la masse. Les autres connexions consistent à relier directement `MOSI`, `MISO` et `SCK` (signal d'horloge) ainsi que les masses des HC11 communiquant entre eux.

Les programmes qui suivent ont pour but de tester la communication SPI : le maître émet dans un premier temps les nombres de 0 à 255 sur le port SPI (et sur son port A pour vérification) et l'esclave affiche sur son port A les valeurs lues. Les deux afficheurs connectés aux ports A de sortie du maître et de l'esclave doivent montrer les mêmes valeurs. Dans un deuxième temps, le maître lit des valeurs sur le port série et les affiche sur son port A tout en les transmettant à l'esclave. Ce second programme permet de vérifier qu'il est possible d'utiliser simultanément la communication RS232 et SPI qui sont toutes deux connectées au port D du maître.

```
0000 8E 00 FF      lds #0h00FF
0003 CE 10 00      ldx #0h1000
0006 86 FF         ldaa #0hFF  ; set port A for output
0008 A7 01         staa 0h01,x
000A 96 2F         ldaa 0h2F   ; SS# hi, SCk lo, mosi hi
000C A7 08         staa 0h08,x ; store port D
000E 86 38         ldaa #0h38  ; - - 1 1 1 0 0 0
0010 A7 09         staa 0h09,x
0012 86 57         ldaa #0h57  ; 7=0111 : polarity, phase, rate1, rate0
                               ; 5=0101 : int, enable, open drain, master
0014 A7 28         staa 0h28,x ; set SPCR

0016 86 00         ldaa #0h00  ; start : counter=0
```

```
0018 A7 2A       snd:staa 0h2A,x  ; send datum
001A A7 00         staa 0h00,x
001C E6 29       wait:ldab 0h29,x
001E 2A FC         bpl wait    ; wait until xfer finished
0020 BD 00 26      jsr delay
0023 4C            inca
0024 20 F2         bra snd

0026 18 CE FF FF delay:ldy #0hFFFF
002A 18 09       wait_d:dey
002C 26 FC         bne wait_d
002E 39            rts
```

Master : émission des nombres de 0 à 255 sur le port SPI (D) et A.

```
0000 8E 00 FF lds #0h00FF
0003 CE 10 00 ldx #0h1000
0006 96 2F    ldaa 0h2F   ; SS# hi, SCk lo, mosi hi
0008 A7 08    staa 0h08,x ; store port D
000A 86 38    ldaa #0h38  ; - - 1 1 1 0 0 0
000C A7 09    staa 0h09,x ; store DDRD
000E 86 47    ldaa #0h47  ; 7=0111 : polarity, phase, rate1, rate0
                          ; 5=0101 : int, enable, open drain, master
0010 A7 28    staa 0h28,x ; set SPCR
0012 86 FF    ldaa #0hFF
```

```
0014 A7 01    staa 0h01,x ; port A for output
0016 86 88    ldaa #0h88
0018 A7 00    staa 0h00,x ; show 88 on port A while waiting
001A        rcv:
001A E6 29  wait:ldab 0h29,x
001C 2A FC    bpl wait    ; wait until xfer finished
001E A6 2A    ldaa 0h2A,x ; send datum
0020 A7 00    staa 0h00,x
0022 20 F6    bra rcv
```

Slave : réception des signaux sur le port SPI et affichage sur le port A

```
0000 8E 00 FF lds #0h00FF
0003 CE 10 00 ldx #0h1000
0006 86 FF    ldaa #0hFF  ; set port A for output
0008 A7 01    staa 0h01,x
                ; SPI SETUP
000A 96 2F    ldaa 0h2F   ; SS# hi, SCk lo, mosi hi
000C A7 08    staa 0h08,x ; store port D
000E 86 38    ldaa #0h38  ; - - 1 1 1 0 0 0
0010 A7 09    staa 0h09,x
0012 86 57    ldaa #0h57  ; 7=0111 : polarity, phase, rate1, rate0
                ; 5=0101 : int, enable, open drain, master
0014 A7 28    staa 0h28,x ; set SPCR
                ; SERIAL PORT SETUP
```

```
  ; bset 0h28,X,#0h20 ; fontionne pas si D port en open-drain
0016 6F 2C    clr 0h2c,x
0018 CC 33 0C ldd #0h330C ; serial port: 2400 baud, N81
001B A7 2B    staa 0h2B,x
001D E7 2D    stab 0h2D,x

001F 1F 2E 20 FC rs_rc:brclr 0h2E,x,#0h20,rs_rc; receive from RS232
0023 E6 2F    ldab 0h2F,x
0025 E7 2A  snd:stab 0h2A,x ; send datum to SPI
0027 E7 00    stab 0h00,x
0029 E6 29  wait:ldab 0h29,x
002B 2A FC    bpl wait    ; wait until xfer finished
002D 20 F0    bra rs_rc
```

Master : réception des valeurs ASCII sur le port série puis echo sur le port SPI et A.

11 Contrôle d'écran LCD :

Nous nous intéresserons aux écrans basés sur le contrôleur Hitachi HD44780. Dans notre cas il s'agit d'un écran de 4×16 caractères. Ce type d'écran possède un connecteur à 14 broches, qui sont dans l'ordre (en partant du bord de l'écran) : masse (GND), alimentation (+5V), contraste (dans notre cas $\lesssim$-5V), 3 bits de contrôle et 8 bits de données. Il existe un mode de communication sur seulement 4 bits de données, permettant ainsi de totalement contrôler l'afficheur avec un seul bus de 8 bits.

Les 3 bits de contrôle sont, de la broche de polarisation (vers le bord de l'afficheur) vers les broches du bus de données : `RS` qui détermine si la valeur sur le bus de données est une instruction (0) ou une donnée (1), `R/W` qui détermine le sens de la transaction (0 pour une écriture, 1 pour une lecture), et `E` (enable) qui doit être à 1 pour rendre l'afficheur actif. Suivent ensuite les lignes de données en commençant de la ligne 0 (poids le plus faible) jusqu'à la ligne 7 (poids le plus fort, au bord du connecteur). En mode de communication 4 bits, seuls les lignes D4-D7 sont utilisées.

Le mode le plus simple d'utilisation est sur 8 bits de données. Nous connectons le bus de données au port B du 68HC11 et le bus de contrôle au port A. Les tableaux ci-dessous résument le brochage, les commandes à envoyer sur les bus de données et de contrôle, ainsi que les valeurs possibles des variables de configuration de l'écran et leur assignation (entre parenthèses) dans nos exemples.

1	2	3	4	5	6	7	-	14
Vss	Vcc=+5V	Vee<0V	RS	R/W	E	D0	-	D7

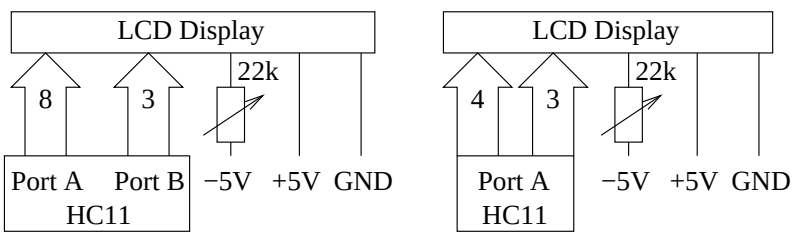
Brochage de l'afficheur LCD (la broche 1 est la plus proche du bord de l'afficheur)

	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Clear Display	0	0	0	0	0	0	0	0	0	1
Entry Mode Set	0	0	0	0	0	0	0	1	`I/D`	`S`
Display ON/OFF	0	0	0	0	0	0	1	`D`	`C`	`B`
Function Set	0	0	0	0	1	`DL`	`N`	`F`	*	*
Set DD RAM address	0	0	1	A	A	A	A	A	A	A
Return Home	0	0	0	0	0	0	0	0	1	*
Cursor and Display Shift	0	0	0	0	0	1	`S/C`	`R/L`	*	*
Set CG RAM address	0	0	0	1	A	A	A	A	A	A
Read busy flag and address	0	1	`BF`	A	A	A	A	A	A	A
Write data to CG or DD RAM	1	0	D	D	D	D	D	D	D	D
Read data from CG or DD RAM	1	1	D	D	D	D	D	D	D	D

D (0) : display on(1)/off(0)
C (0) : curseur on(1)/off(0)
B (0) : curseur clignotant on(1)/off(0)

DL (1) : interface sur 8 bits (4 bits si DL=0)
N (1) : 2 lignes
F (0) : font 5×x7

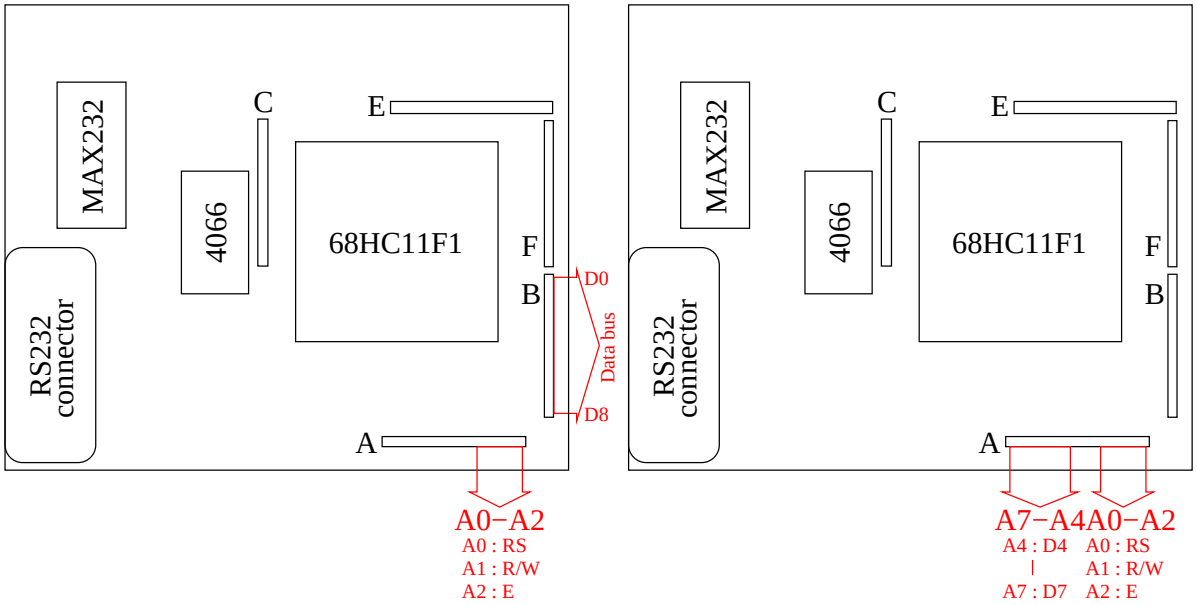
I/D (1) : incrémenter la position du curseur après affichage
S (0) : S=0 lors des utilisations habituelles (?!)



Nous n'utiliserons que les fonctions présentées dans la partie supérieure du tableau. Les fonctions de la partie inférieure servent à modifier les caractères disponibles sur l'afficheur et à lire l'état du contrôleur de l'afficheur.

Il est prudent de toujours attendre au moins 40 μs après chaque instruction.

Connexion du 68HC11 à l'afficheur LCD pour un bus de données sur 8 bits (gauche) et un bus de données sur 4 bits (droite).



Vue de dessus du circuit de base du 68HC11 avec position des bus reliant l'écran LCD aux ports du HC11, dans le cas d'une communication en 8 bits (gauche) et 4 bits (droite).

```
0000 8E 01 FF lds #0h01FF ; setup : stack & port A for output
0003 CE 10 00 ldx #0h1000
0006 86 FF    ldaa #0hFF
0008 A7 01    staa 0h01,x
       ; DISPLAY SETUP : START
000A 86 01  on:ldaa #0h01 ; clear display
000C 8D 21    bsr funct
000E 86 F2    ldaa #0hF2  ; R/W#=1 + display 'F'
0010 8D 4E    bsr delai
0012 86 38  set:ldaa #0h38; set : 0h38 (8 bit, 4 lines)
0014 8D 19    bsr funct
       ;off:ldaa #0h08   ; display off : 0h08
       ;     bsr funct
0016 86 0C  blank:ldaa #0h0C ; displ2 : 0h0C
0018 8D 15    bsr funct
001A 86 06  entry:ldaa #0h06 ; entry mode : 0h06
001C 8D 11    bsr funct
       ; END OF SETUP
001E 8D 50    bsr messg   ; AFFICHE HELLO WORLD
0020 86 15    ldaa #0h15  ; position
0022 C6 33    ldab #0h33  ; char
0024 8D 1C    bsr gotoxy
0026       set_dd:
0026 5C       incb
0027 8D 24    bsr char    ; display
0029 C1 38    cmpb #0h38
002B 26 F9    bne set_dd
002D       start:
002D 20 FE    bra start

  ; data bus (port A) a ete set to action | RS = 0 (set function)
  ; input : function in register A
002F A7 04  funct:staa 0h04,x
0031 86 F0    ldaa #0hF0  ; bit 1 : R/W#=0
0033 8D 2B    bsr delai
0035 86 F4    ldaa #0hF4  ; bit 2 : E=1
0037 8D 27    bsr delai
0039 86 F0    ldaa #0hF0  ; E=0
003B 8D 23    bsr delai
003D 86 F2    ldaa #0hF2  ; R/W#=1
003F 8D 1F    bsr delai
0041 39       rts

; goto a wanted position and display a char (automaticly shifts right
; after) input : char value in register B and position in register A
0042 8A 80  gotoxy:oraa #0h80 ; reg A -> set @ to reg A
0044 A7 04    staa 0h04,x
0046 8D E7    bsr funct
0048 E7 04    stab 0h04,x
004A 8D 01    bsr char
004C 39       rts

; data bus (port A) a ete set to char | RS = 1 (draw char)
; input : char value in register B
004D E7 04  char:stab 0h04,x
004F 86 11    ldaa #0h11  ; RS=1, R/W#=0, E=0
0051 8D 0D    bsr delai
0053 86 15    ldaa #0h15  ; RS=1, R/W#=0, E=1
0055 8D 09    bsr delai
0057 86 11    ldaa #0h11  ; E=0
0059 8D 05    bsr delai
005B 86 12    ldaa #0h12  ; RS=0, R/W#=1, E=0
005D 8D 01    bsr delai
005F 39       rts

0060 A7 00  delai:staa 0h00,x ; met le registre A sur le port A
0062 86 01    ldaa #0h01
0064 18 CE 2F FF dedans:ldy #0h2FFF ; ldab #0h06 et ldy #0hFFFF => 693 ms
0068 18 09  dedans2:dey             ;----1 -- --- ------- => 115 ms
006A 26 FC    bne dedans2           ;----1 -- --- #0h2FFF => 21.5 ms
006C 4A       deca
006D 26 F5    bne dedans
006F 39       rts

       ; DEBUT DE L'AFFICHAGE CST :
0070 86 02  messg:ldaa #0h02 ; position (starts at 0)
0072 C6 48    ldab #0h48  ; H
0074 8D CC    bsr gotoxy
0076 C6 65    ldab #0h65  ; e
0078 8D D3    bsr char    ; display
007A C6 6C    ldab #0h6C  ; l
007C 8D CF    bsr char
007E C6 6C    ldab #0h6C  ; l
0080 8D CB    bsr char
0082 C6 6F    ldab #0h6F  ; o
0084 8D C7    bsr char
0086 C6 20    ldab #0h20  ; ' '
0088 8D C3    bsr char
008A C6 57    ldab #0h57  ; W
008C 8D BF    bsr char
008E C6 6F    ldab #0h6F  ; o
0090 8D BB    bsr char
0092 C6 72    ldab #0h72  ; r
0094 8D B7    bsr char
0096 C6 6C    ldab #0h6C  ; l
0098 8D B3    bsr char
009A C6 64    ldab #0h64  ; d
009C 8D AF    bsr char
009E 39       rts
```

```
0000 8E 01 FF lds #0h01FF    ; setup : stack & port A for output
0003 CE 10 00 ldx #0h1000
0006 86 FF    ldaa #0hFF
0008 A7 01    staa 0h01,x    ; DISPLAY SETUP : START

000A 8D 26    bsr delai

000C 86 28  set: ldaa #0h28
000E 8D 2D    bsr funct
0010 86 01  on:ldaa #0h01    ; clear display
0012 8D 29    bsr funct
0014 86 F2    ldaa #0hF2     ; R/W#=1 + display 'F'
0016 8D 1A    bsr delai
0018 86 0C  blank:ldaa #0h0C ; displ2 : 0h0C
001A 8D 21    bsr funct
001C 86 06  entry:ldaa #0h06 ; entry mode : 0h06
001E 8D 1D    bsr funct
       ; END OF SETUP

0020 BD 00 B0 jsr messg      ; AFFICHE HELLO WORLD
0023 86 15    ldaa #0h15     ; position
0025 C6 35    ldab #0h35     ; char
0027 8D 42    bsr gotoxy
0029        set_dd:
0029 5C       incb
002A 8D 4A    bsr char       ; display
002C C1 3B    cmpb #0h3B
002E 26 F9    bne set_dd
0030        start:
0030 20 FE    bra start      ; attente
0032 A7 00  delai:staa 0h00,x; met le registre A sur le port A
0034 18 CE 3F FF ldy #0h3FFF
0038 18 09  dedans2:dey
003A 26 FC    bne dedans2
003C 39       rts
 ; data bus (port A) a ete set to action | RS = 0 (set function)
 ; input : function in register A
003D 36     funct:psha
003E 37       pshb
003F 36       psha
0040 84 F0    anda #0hF0     ; hi = char ; lo = 0000
0042 A7 00    staa 0h00,x    ; FIRST NIBBLE
0044 8D EC    bsr delai
0046 1C 00 04 bset 0h00,x,#0h04 ;ldaa       #0hF4 : bit 2 : E=1
0049 8D E7    bsr delai
004B 1D 00 04 bclr 0h00,x,#0h04; E=0
004E 8D E2    bsr delai
0050 32       pula
0051 48       lsla
0052 48       lsla
0053 48       lsla
0054 48       lsla
0055 A7 00    staa 0h00,x    ; SECOND NIBBLE
0057 8D D9    bsr delai
0059 1C 00 04 bset 0h00,x,#0h04;ldaa #0hF4 : bit 2 : E=1
005C 8D D4    bsr delai
005E 1D 00 04 bclr 0h00,x,#0h04 ; E=0
0061 8D CF    bsr delai
0063 1C 00 02 bset 0h00,x,#0h02 ; R/W#=1
0066 8D CA    bsr delai
0068 33       pulb
0069 32       pula
006A 39       rts

; goto a wanted position and display a char (automaticly shifts right after)
; input : char value in register B and position in register A
006B 8A 80  gotoxy:oraa #0h80; reg A -> set @ to reg A
006D A7 04    staa 0h04,x
006F 8D CC    bsr funct
0071 E7 04    stab 0h04,x
0073 8D 01    bsr char
0075 39       rts

 ; data bus (port A) a ete set to char | RS = 1 (draw char)
 ; input : char value in register B
0076 36     char:psha
0077 37       pshb
0078 C4 F0    andb #0hF0
007A E7 00    stab 0h00,x    ; FIRST NIBBLE, RS=1, R/W#=0, E=0
007C 1C 00 01 bset 0h00,x,#0h01; RS=1
007F 8D B1    bsr delai
0081 1C 00 04 bset 0h00,x,#0h04; RS=1, R/W#=0, E=1
0084 8D AC    bsr delai
0086 1D 00 04 bclr 0h00,x,#0h04; E=0
0089 8D A7    bsr delai
008B 58       lslb
008C 58       lslb
008D 58       lslb
008E 58       lslb
008F C4 F0    andb #0hF0
0091 E7 00    stab 0h00,x    ; SECOND NIBBLE, RS=1, R/W#=0, E=0
0093 1C 00 01 bset 0h00,x,#0h01; RS=1
0096 8D 9A    bsr delai
0098 1C 00 04 bset 0h00,x,#0h04; RS=1, R/W#=0, E=1
009B BD 00 32 jsr delai
009E 1D 00 04 bclr 0h00,x,#0h04; E=0
00A1 BD 00 32 jsr delai
00A4 1C 00 02 bset 0h00,x,#0h02; RS=0, R/W#=1, E=0
00A7 1D 00 01 bclr 0h00,x,#0h01; RS=0, R/W#=1, E=0
00AA BD 00 32 jsr delai
00AD 33       pulb
00AE 32       pula
00AF 39       rts

  ; DEBUT DE L'AFFICHAGE CST :
00B0 86 02  messg:ldaa #0h02 ; position (starts at 0)
00B2 C6 48    ldab #0h48     ; H
00B4 8D B5    bsr  gotoxy
00B6 C6 65    ldab #0h65     ; e
00B8 8D BC    bsr  char      ; display
00BA C6 6C    ldab #0h6C     ; l
00BC 8D B8    bsr  char
00BE C6 6C    ldab #0h6C     ; l
00C0 8D B4    bsr  char
00C2 C6 6F    ldab #0h6F     ; o
00C4 8D B0    bsr  char
00C6 C6 20    ldab #0h20     ; ' '
00C8 8D AC    bsr  char
00CA C6 57    ldab #0h57     ; W
00CC 8D A8    bsr  char
00CE C6 6F    ldab #0h6F     ; o
00D0 8D A4    bsr  char
00D2 C6 72    ldab #0h72     ; r
00D4 8D A0    bsr  char
00D6 C6 6C    ldab #0h6C     ; l
00D8 8D 9C    bsr  char
00DA C6 64    ldab #0h64     ; d
00DC 8D 98    bsr  char
00DE 39       rts
```

Exemple de programme pour une interface 4 bits. À noter que pour des raisons de problèmes de délais dans les initialisations, il faut relancer plusieurs fois ce programme avant qu'un message ne s'affiche sur l'écran LCD (effectuer un reset du 68HC11 mais ne pas éteindre le générateur de tension entre deux reprogrammations afin de ne pas ré-initialiser à chaque fois l'écran LCD).

12 Les interruptions

12.1 Les interruptions matérielles IRQ#

12.1.1 Aspect matériel

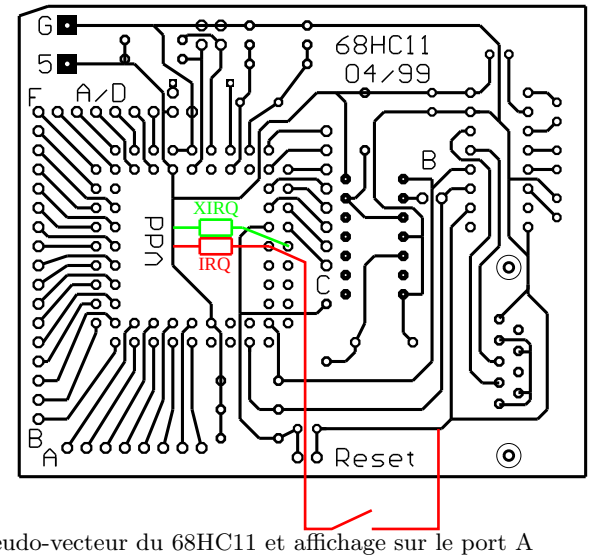
Modifications au circuit initial : mettre le bit `I` du registre de contrôle par `cli` va rendre les interruptions matérielles actives. Par défaut, toutes les sources internes au 68HC11 d'interruptions matérielles (principalement les timers) sont inactives. Il nous faut de plus désactiver les broches d'interruptions matérielles `XIRQ` et `IRQ` (actives par un signal au niveau bas) en ajoutant deux résistances de pull-up entre l'alimentation (Vdd=+5V) et ces broches. L'activation de l'interruption matérielle `IRQ` se fait alors en mettant cette broche au niveau bas (par exemple en fermant un interrupteur dans notre exemple).

12.1.2 Aspect logiciel

Principe des pseudo-vecteurs : les vecteurs d'interruptions sont situés en ROM et ne sont donc pas modifiables pour pointer vers la fonction de l'utilisateur. Ces vecteurs pointent vers des pseudo-vecteurs pré-définis qui sont eux situés en RAM. Il s'agit en fait de trois octets réservés à l'action à effectuer lorsque l'interruption est déclenchée : une instruction `JMP` (0x7E) suivi des 2 octets de l'adresse de la fonction de l'utilisateur. La liste des adresses des pseudo-vecteurs est disponible p. 3-18 du 68HC11 Reference Manual de Motorola.

Le premier exemple que nous proposons ici sert à bien identifier la syntaxe permettant de récupérer l'adresse où se situe une procédure, et de voir comment manipuler cette adresse. Dans le programme ci-dessous, la procédure nommée `irq` commence à l'adresse 0x0026. Nous allons chercher cette adresse, la stocker dans le registre 16 bits X, puis transférer X dans D (qui est en fait la concaténation de A et B) de façon à pouvoir successivement afficher sur le port A les valeurs des registres A et B et ainsi pouvoir visualiser (par exemple sur un afficheur 7 segments connecté au port A) l'adresse de la procédure (cette étape intermédiaire - le passage de X à D - est obligatoire pour pouvoir visualiser la valeur stockée dans X. Il n'est pas possible d'envoyer directement X, qui est sur 16 bits, sur un des ports d'entrée-sortie 8 bits). Il faut prendre soin, lors de la recherche de l'adresse de la procédure `irq`, d'utiliser la syntaxe `ldx #irq` qui renvoie l'adresse où se trouve `irq` (0h0023 dans notre cas), et non pas `ldx irq` qui renvoie le contenu de l'octet se trouvant à l'adresse `irq` (0x86 dans notre cas, qui est l'opcode de l'instruction `ldaa`).

```
0000 8E 00 FF   lds  #0h00FF
0003 86 FF      ldaa #0hFF ; DDRA : output
0005 B7 10 01   staa 0h1001
0008 86 7E      ldaa #0h7E ; opcode de JMP
000A 97 EE      staa 0h00EE; interrupt vector-1=JMP opcode
000C CE 00 23   ldx #irq   ; ISR de IRQ#
000F DF EF      stx 0h00EF ; interrupt vector
0011 CE 10 00   ldx #0h1000
                ;  cli         ; enable interrupts
0014 18 DE EF ici: ldy 0h00EF
0017 18 8F      xgdy        ; echange X et D
0019 A7 00      staa 0h00,x; afficher alternativement
001B 8D 09      bsr delai  ; ... A et B pour montrer
001D E7 00      stab 0h00,x; ... ce qui etait dans X
001F 8D 05      bsr delai
0021 20 F1      bra ici

0023 86 00    irq: ldaa #0h00 ; reset port A
0025 3B         rti

0026 18 CE FF FF delai:ldy #0hFFFF
002A 18 09    wait: dey
002C 26 FC      bne wait
002E 39         rts
```



Programme de lecture de la valeur stockée dans un pseudo-vecteur du 68HC11 et affichage sur le port A (vérification de la programmation de l'adresse de la fonction de gestion des interruptions)

L'instruction `cli` met le bit `I` du registre de contrôle à 0 et met ainsi en marche les interruptions matérielles.

Application au cas de la ré-initialisation d'une variable par une interruption matérielle. Il est nécessaire de passer par une zone mémoire en RAM car l'appel à une fonction provoque un empilement des registres qui seront ensuite ré-initialisés à leurs valeurs initiales lors de l'instruction `rti`. Ce programme fait défiler les nombres binaires sur le port A, et remet le compteur à 0 chaque fois qu'une interruption matérielle est provoquée par la fermeture de l'interrupteur.

```
0000 8E 00 FF      lds        #0h00FF
0003 86 FF         ldaa        #0hFF       ; DDRA : output
0005 B7 10 01      staa        0h1001
0008 86 7E         ldaa        #0h7E       ; opcode de JMP
000A 97 EE         staa        0h00EE       ; interrupt vector-1=JMP opcode
000C CE 00 24      ldx        #irq       ; ISR de IRQ#
000F DF EF         stx        0h00EF       ; interrupt vector
0011 CE 10 00      ldx  #0h1000
0014 0E            cli  ; enable interrupts
0015 86 00         ldaa #0h00
0017 97 E0         staa 0h00E0
0019 96 E0       ici:ldaa 0h00E0
001B 4C            inca
001C 97 E0         staa 0h00E0
001E A7 00         staa 0h00,x
0020 8D 07         bsr delai
0022 20 F5         bra ici

0024 86 00   irq: ldaa #0h00       ; PAS ldaa car rti pop regs
0026 97 E0         staa 0h00E0
0028 3B            rti

0029 18 CE FF FF delai:ldy #0hFFFF
002D 18 09       wait:dey
002F 26 FC         bne wait
0031 39            rts
```

Exemple de remise à 0 par interruption matérielle d'un compteur : le port A affiche les valeurs croissantes d'un constante d'un compteur qui est remis à 0 par la fermeture de l'interrupteur reliant la broche `IRQ` à la masse.

12.2 Les interruptions matérielles liées aux timers

Il faut penser, pour toute utilisation des interruptions matérielles liées aux timers internes du HC11, à effectuer la modification sur les broches IRQ et XIRQ (en les reliant via une résistance d'environ 5 kΩ à l'alimentation Vdd).

L'utilisation du compteur présentée ici permet de réaliser un scheduler lorsque plusieurs tâches doivent être exécutées séquentiellement avec une périodicité donnée, ou une modulation en largeur de pulse (PWM). Dans le but d'avoir le temps de voir le compteur osciller, nous avons décidé d'abaisser la fréquence du timer au $16^{ème}$ de sa valeur par défaut en mettant PR1 et PR0 du registre TMSK2 (0h1024) à 1. Le compteur va de 0 à 65536 en 524.3/2 ms (*i.e.* 4 μs par tic d'horloge). On le fait compter de 1 à 0hDFFF=53248+3840+240+15=57343 soit 229.4 ms entre deux incréments du compteur.

La valeur du compteur, incrémenté à chaque interruption, est affichée sur les ports A et B. La broche 6 du port A oscille de plus à la fréquence des interruptions (sortie PWM).

```
0000 8E 00 FF lds #0h00FF ; stack
0003 86 FF    ldaa #0hFF  ; set port A for output
0005 B7 10 01 staa 0h1001 ; cpt val sur A/B, A6=PWM
0008 86 7E    ldaa #0h7E  ; opcode de JMP
000A 97 DC    staa 0h00DC ; interrupt vector of OC2
000C CE 00 2E ldx #int    ; ISR de int
000F DF DD    stx 0h00DD  ; interrupt vect+1 de OC2=int @

0011 CE 10 00 ldx  #0h1000
0014 1C 24 03 bset 0h24,x,#0h03 ; prescaler : /16
0017 86 40    ldaa #0h40  ; only work on OC_2_
0019 A7 23    staa 0h23,x ; clear any pending flag
001B A7 22    staa 0h22,x ; enable OC2 interrupt
001D 86 40    ldaa #0h40  ; intOC2:0h00
001F A7 20    staa 0h20,x ; 0h40 -> toggle output of A6
0021 86 00    ldaa #0h00  ; initialisation du compteur

0023 97 E0    staa 0h00E0 ; octet en RAM stockant la valeur a afficher
0025 0E       cli ; enable interrupts : check pull ups on IRQ & XIRQ
0026 96 E0  ici:ldaa 0h00E0
0028 A7 04    staa 0h04,x ; display counter value on port B
002A A7 00    staa 0h00,x ; display counter value on port A
002C 20 F8    bra ici

002E 96 E0  int:ldaa 0h00E0; PAS inca directmt car rti pop regs
0030 4C       inca        ; incremente le compteur
0031 97 E0    staa 0h00E0 ; stocke le result
0033 CC DF FF ldd #0hDFFF ; 1/2 delay PWM <- duree entre 2 transitions
0036 E3 12    addd 18,x   ; add value of counter 2 to delay
0038 ED 12    std  18,x   ; store in counter -> time of next interrupt
003A 86 40    ldaa #0h40
003C A7 23    staa 0h23,x ;  clear flag by setting 0100 0000 (OC2)
003E 3B       rti
```

12.3 Programmation multi-tâ ches au moyen des interruptions - l'instruction `WAI`

Jusqu'ici, nos programmes n'étaient capables que d'effectuer une opération à la fois : soit faire l'acquisition de données provenant des convertisseurs analogiques-numériques, soit transmettre des données sur le port série, soit attendre la réception de données sur le port RS232. L'utilisation des interruptions dans l'exemple qui suit va donner l'impression que le 68HC11 est capable de simultanément incrémenter un compteur de façon périodique tout en étant à l'écoute du port série. De plus, mettre toutes les fonctions à exécuter par le microcontrôleur sous interruption permet une économie d'énergie par l'exécution de l'instruction `WAI` lorsque qu'aucune tâche n'est à exécuter. Cette instruction met le 68HC11 en mode veille qui consomme moins de courant. Le micro-contrôleur est réactivé lorsqu'une interruption qui a été autorisée (dans notre cas le timer, la réception d'un message sur le port série ou l'activation de l'interruption matérielle sur la broche IRQ) est déclenchée.

```
8E 01 20      lds #0h0120   ; set stack
CE 10 00      ldx #0h1000
86 00         ldaa #0h00    ; port A setup for input
A7 01         staa 0h01,x   ; |

86 7E         ldaa #0h7E    ; opcode de JMP
97 C4         staa 0h00C4   ; interrupt vector-1=JMP opcode
CE 00 2F      ldx #irq      ; ISR de IRQ# : ldx irq=00 86 (op de jmp)
DF C5         stx 0h00C5    ; interrupt vector
CE 10 00      ldx #0h1000
0E            cli           ; enable interrupts

1C 28 20      bset  0h28,x,#0h20 ; serial port setup
6F 2C         clr  0h2c,x
CC 33 2C      ldd  #0h332C  ; set baudrate, enable recv & tx
A7 2B         staa  0h2B,x  ; 2400 N81 with 16 MHz xtal
E7 2D         stab  0h2D,x  ; 2 de 2C : enable receive interrupt
```

```
           main: ; ldaa 0h0100  ; read stored value
             ; staa 0h05,x ; and display on port F
3E           wai           ; low power sleep mode
20 FD        bra  main

18 CE FF FF delai:ldy #0hFFFF
18 09       wait:dey
26 FC        bne wait
39           rts

1F 2E 20 FC irq: brclr 0h2E,x,#0h20,irq ; reset serial port
A6 2F        ldaa 0h2F,x   ; load received character
B7 01 00     staa 0h0100   ; store in memory
A7 05        staa 0h05,x   ; display on port F
             rti           ; end of interrupt service routine
```

Exemple d'interruption déclenchée à la réception d'un caractère sur le port série : le compteur est remplacé par la valeur ASCII du caractère reçu.

```
8E 01 20  lds #0h0120  ; set stack
CE 10 00  ldx #0h1000
86 00     ldaa #0h00   ; port A setup for input
A7 01     staa 0h01,x  ; |
      ; setup serial port interrupt routine
86 7E     ldaa #0h7E   ; opcode de JMP
97 C4     staa 0h00C4  ; interrupt vector-1=JMP opcode <- serial port
97 D0     staa 0h00D0  ; interrupt vector-1=JMP opcode <- timer ovflow
      ;   staa 0h00EE  ; interrupt vector-1=JMP opcode <- IRQ
CE 00 4F  ldx #intrs232
DF C5     stx 0h00C5   ; interrupt vector
CE 00 41  ldx #ovltimr
DF D1     stx 0h00D1   ; interrupt vector
      ;   ldx #irq
      ;   stx 0h00EF   ; interrupt vector
CE 10 00  ldx #0h1000
0E        cli          ; enable interrupts
      ; setup serial port
1C 28 20  bset 0h28,x,#0h20 ; serial port setup
6F 2C     clr 0h2c,x
CC 33 2C  ldd #0h332C  ; set baudrate, enable recv, tsmit & int
A7 2B     staa 0h2B,x
E7 2D     stab 0h2D,x
      ; enable timer overflow interrupt
1D 25 7F  bclr 0h25,x,#0h7F ; clear overflow (if any)
```

```
1C 24 83  bset 0h24,x,#0h83 ; enable interrupt and prescaler=16
     ; lsla : logic shift left A
86 00     ldaa #0h00   ; timer related interrupt
B7 01 01  staa 0h0101
       ici:           ; ldaa       0h0100
      ; staa 0h05,x
3E        wai
20 FD     bra ici
18 CE FF FF delai:ldy #0hFFFF
18 09   wait:dey
26 FC     bne       wait
39        rts
86 80   ovltimr:ldaa #0h80
A7 25     staa 0h25,x  ; clear overflow flag
B6 01 01  ldaa 0h0101
4C        inca
A7 05     staa 0h05,x
B7 01 01  staa 0h0101
3B        rti
       intrs232:      ; store char at memory location 0h100
1F 2E 20 FC brclr 0h2E,x,#0h20,intrs232
A6 2F     ldaa 0h2F,x
B7 01 00  staa 0h0100
A7 05     staa 0h05,x
3B        rti
```

Exemple de combinaison des interruptions timer (incrément du compteur) et de réception sur le port série. Noter que la procédure principale est uniquement formée d'une boucle sur la fonction `WAI` dont l'intérêt est l'économie de courant.

Nous atteignons ici les limites du 68HC11F1 en mode mono-composant puisque le gestionnaire de port série et du compteur occupe presque la totalité des 512 octets d'EEPROM disponibles. Il faut ensuite ajouter de l'EEPROM externe ou trouver une version du micro-contrôleur possédant plus de mémoire.

13 Programme d'envoi des données du PC vers le 68HC11 : `hc11.pas` (Turbo Pascal (7), DOS)

Le programme `hc11.pas` fonctionne de la façon suivante (routine `RunTerminal()` : le programme envoie tout d'abord le caractère 0hFF pour prévenir le HC11 que le programme sera transféré par le port série, puis ouvre le fichier qui est passé en premier argument. Ce fichier contient, sous forme de caractères ASCII, une suite de nombres hexadécimaux. Le programme prend chaque paire de caractères (séparés par un espace ou un retour chariot), les interprète comme un nombre en hexadécimal, et envoie le résultat sur le port série. Si deux arguments sont passés en ligne de commande, le programme suppose que le premier programme est `ee.out` (dont le rôle est de récupérer les données sur le port série pour les stocker en EEPROM) et relance donc l'émission des données stockées dans le second fichier après avoir attendu 1 s pour permettre l'exécution de `ee`.

Le rôle de ce programme est donc d'initialiser le port série (choisi dans cet exemple comme COM1 avec une vitesse de transmission de 2400 bauds) et d'envoyer un programme stocké comme une suite de valeurs hexadécimales dans la RAM du 68HC11. Si le programme envoyé en RAM est le logiciel chargé de

stocker des données en EEPROM, le passage en second argument du nom du programme à charger dans l'EEPROM du 68HC11 lancera la routine de transmission de ce second programme après avoir exécuté le premier programme stocké en RAM.

La procédure `RunTerminal` pourra être modifié selon les besoins de communication par le port série, par exemple pour la récupération des données lues en RAM par le 68HC11 et renvoyées au PC par le port série.

Nous n'avons pas inclus, pas souci de place, le code source de `hc11.pas` pour DOS, puisque ce système d'exploitation demande au programmeur d'inclure un grand nombre de fonctions de mise en marche de la communication série. Un exemple plus approprié est disponible ci-dessous, plus concis et contenant les mêmes fonctionnalités.

14 Programme d'envoi des données du PC vers le 68HC11 : `hc11_linux.c` (gcc, Linux)

Le programme suivant utilise la même syntaxe que celle vue dans la partie précédente (13) pour programmer le 68HC11 sous Linux. Le programme est en deux parties : le code d'initialisation du port série d'une part (`rs232.c`) et le code de lecture du fichier contenant le programme à charger en hexadécimal et de communication avec le HC11 (`hc11_linux.c`). La compilation se fait par gcc -c rs232.c;gcc -o hc11_linux hc11_linux.c rs232.o.

Le programme `rs232.c` :

```
/* All examples have been derived from miniterm.c                        */
/* Don't forget to give the appropriate serial ports the right permissions */
/* (e. g.: chmod a+rw /dev/ttyS0)                                        */

#include "rs232.h"

extern struct termios oldtio,newtio;

int init_rs232()
{int fd;
 fd=open(HC11DEVICE, O_RDWR | O_NOCTTY );
 if (fd <0) {perror(HC11DEVICE); exit(-1); }
 tcgetattr(fd,&oldtio); /* save current serial port settings */
 bzero(&newtio, sizeof(newtio)); /* clear struct for new port settings */
// newtio.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD;
 newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;  /* _no_ CRTSCTS */
 newtio.c_iflag = IGNPAR | ICRNL |IXON;
 newtio.c_oflag = ONOCR|ONLRET|OLCUC;
// newtio.c_lflag = ICANON;
// newtio.c_cc[VINTR]    = 0;     /* Ctrl-c */
// newtio.c_cc[VQUIT]    = 0;     /* Ctrl-\ */
// newtio.c_cc[VERASE]   = 0;     /* del */
// newtio.c_cc[VKILL]    = 0;     /* @ */
// newtio.c_cc[VEOF]     = 4;     /* Ctrl-d */
 newtio.c_cc[VTIME]    = 0;     /* inter-character timer unused */
 newtio.c_cc[VMIN]     = 1;     /* blocking read until 1 character arrives */
// newtio.c_cc[VSWTC]    = 0;     /* '\0' */
// newtio.c_cc[VSTART]   = 0;     /* Ctrl-q */
// newtio.c_cc[VSTOP]    = 0;     /* Ctrl-s */
// newtio.c_cc[VSUSP]    = 0;     /* Ctrl-z */
// newtio.c_cc[VEOL]     = 0;     /* '\0' */
// newtio.c_cc[VREPRINT] = 0;     /* Ctrl-r */
// newtio.c_cc[VDISCARD] = 0;     /* Ctrl-u */
// newtio.c_cc[VWERASE]  = 0;     /* Ctrl-w */
// newtio.c_cc[VLNEXT]   = 0;     /* Ctrl-v */
// newtio.c_cc[VEOL2]    = 0;     /* '\0' */
 tcflush(fd, TCIFLUSH);tcsetattr(fd,TCSANOW,&newtio);
// printf("RS232 Initialization done\n");
 return(fd);
}

void sendcmd(int fd,char *buf)
{unsigned int i,j;
 if((write(fd,buf,strlen(buf)))<strlen(buf))
    {printf("\n No connection...\n");exit(-1);}
 for (j=0;j<5;j++) for (i=0;i<3993768;i++) {}
 /* usleep(attente); */
}

void free_rs232(int fd)
{tcsetattr(fd,TCSANOW,&oldtio);close(fd);} /* restore the old port settings */
```

Les headers `rs232.c` :

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#include <sys/types.h>
#include <sys/stat.h>
#include <string.h>                 /* declaration of bzero() */
#include <fcntl.h>
#include <termios.h>
```

```
int init_rs232();
void free_rs232();
void sendcmd(int,char*);
struct termios oldtio,newtio;

#define BAUDRATE B2400
// #define BAUDRATE B19200
#define HC11DEVICE "/dev/ttyS0"
```

Le logiciel de programmation du 68HC11 `hc11_linux.c` :

```
#include "rs232.h"

void vrfy_buf(char* buf,int i)               /* verify buffer's content */
{int j,k;
 for (j=0;j<i;j++) {k=(int)buf[j];if (k<0) k+=256;printf("%d ",k);}
 printf("\n");}

void send_hc11(FILE *f,int fd)
{char buf[255];int i=0,status;unsigned int j;
 buf[0]=255;write(fd,buf,1);                          /* start with 'FF' */
 do {do {status=fscanf(f,"%x",&j);buf[i]=(char)j;i++;} /* read while !EOF */
         while ((status!=EOF) && (i<255));      /* && less than 255 chars */
     if (i<255) i--;
```

```
    vrfy_buf(buf,i);write(fd,buf,i);printf ("%d bytes sent\n",i);i=0;}
 while (status!=EOF);
}

void main(int argc,char **argv)
{int fd;FILE *f;
 if (argc<2) printf("%s filename\n",argv[0]); else {
    fd=init_rs232();
    f=fopen(argv[1],"r");send_hc11(f,fd);
 /* free_rs232(); */
}}
```

Le logiciel `hc11rec.c` permettant de récupérer les données transmises par le 68HC11 en 2400, N81, sous Linux :

```
// sous octvae : load t;q=(t>1000);t=(q*65535*65536-t.*q)+(1-q).*t;plot(t);
//              k=ones(10,1);hold;plot(conv(k,t)/10);

#include "rs232.h"

void read_osc(int fd)
{unsigned char buf;
 while (1) {read(fd,&buf,1);printf ("%u\n",(char)buf);fflush(stdout);}
}

void main(int argc,char **argv)
{int fd;
 fd=init_rs232();
 read_osc(fd);
 /* free_rs232(); */
}
```

15 Programme de réception des données de la RAM : `test_ram.pas` (Turbo Pascal (7), DOS)

Ce programme, très semblable à `hc11.pas`, a pour rôle de déclencher l'émission des données stockées dans la RAM liée au HC11 (par l'envoi du caractère $20) et de récupérer les octets transmis par le micro-contrôleur pour les stocker dans le fichier `ram.dat`. En fait la seule différence par rapport au programme précédent réside dans la procédure `RunTerminal()` : nous envoyons un caractère puis recevons `ram_len` octets (`ram_len` étant passé comme paramètre de `RunTerminal()`). Il suffit ensuite de réaliser un petit traitement de conversion du format binaire (suite d'octets) vers une sortie dans un fichier au format ASCII pour pouvoir afficher la courbe des résultats obtenus (en Turbo C sous DOS, `fin=fopen("ram.dat","rb");for () {fscanf(fin,"%c",&c); fprintf(fout,"%d ", (int)c);}`

Ici encore, par souci de place, nous omettons la version DOS de ce programme, beaucoup plus simple à réaliser sous Linux, et sans nouveauté majeure par rapport aux exemples précédents.