Développement logiciel en langage

C

sur micro-contrôleur.

Applications sur CC11 et carte CBOYF1

Christian Dupaty
Professeur de génie électrique
dupaty@unimeca.univ-mrs.fr

CC11 Compilateur croisé C pour 68HC11

CC11 est un produit de la société CONTROLORD, une version freeware limitée à 200 octets de code exécutable est téléchargeable sur **www.controlord.fr**

Document réalisé à partir de la documentation CC11 de Controlord,, une aide très complète est disponible en ligne

Le cross-compilateur CC11 crée à partir des fichiers sources C un fichier format S19 de Motorola.

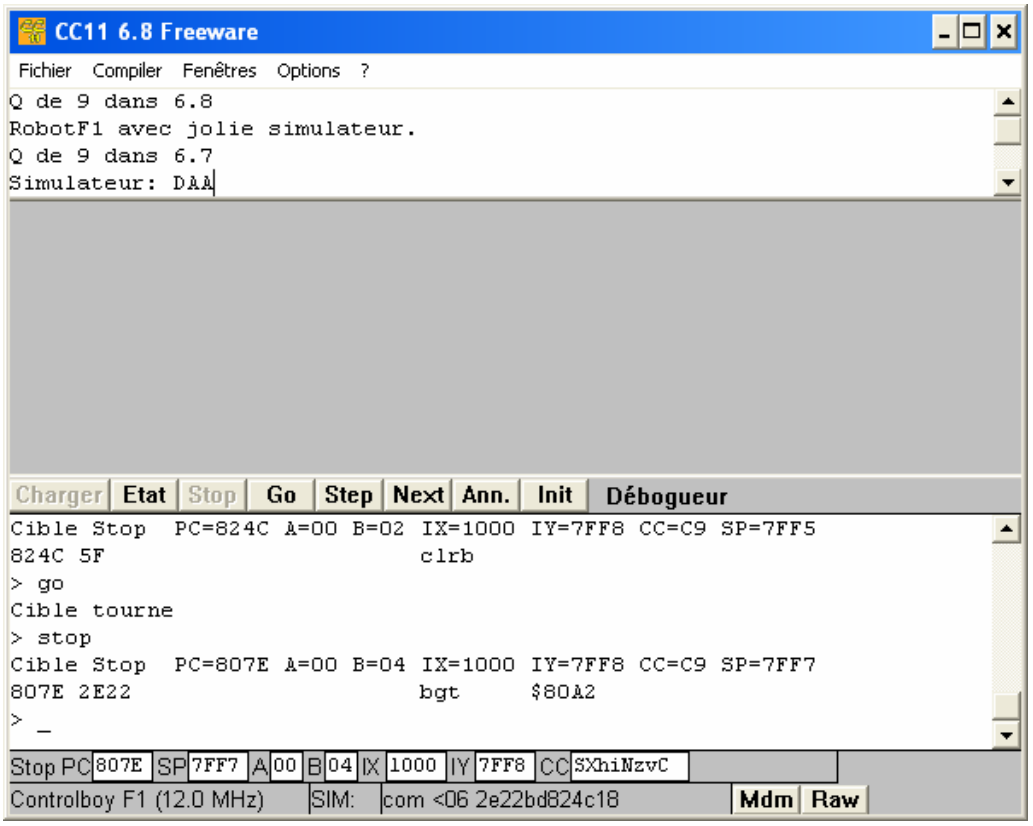
1. Arborescence de CC11 :

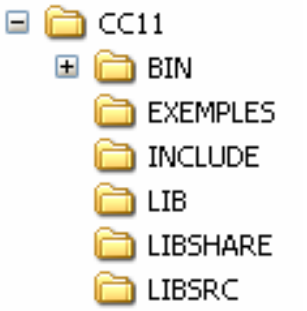
Bin	les différents programmes de CC11
Include	les fichiers en-tête (fichiers #include)
Lib	les fichiers pré-compilés
Libsrc	les fichiers sources de la bibliothèque
Exemples	les exemples de programme
Talkers	les sources de talkers pour le débogueur
Travail	les programmes C utilisateur

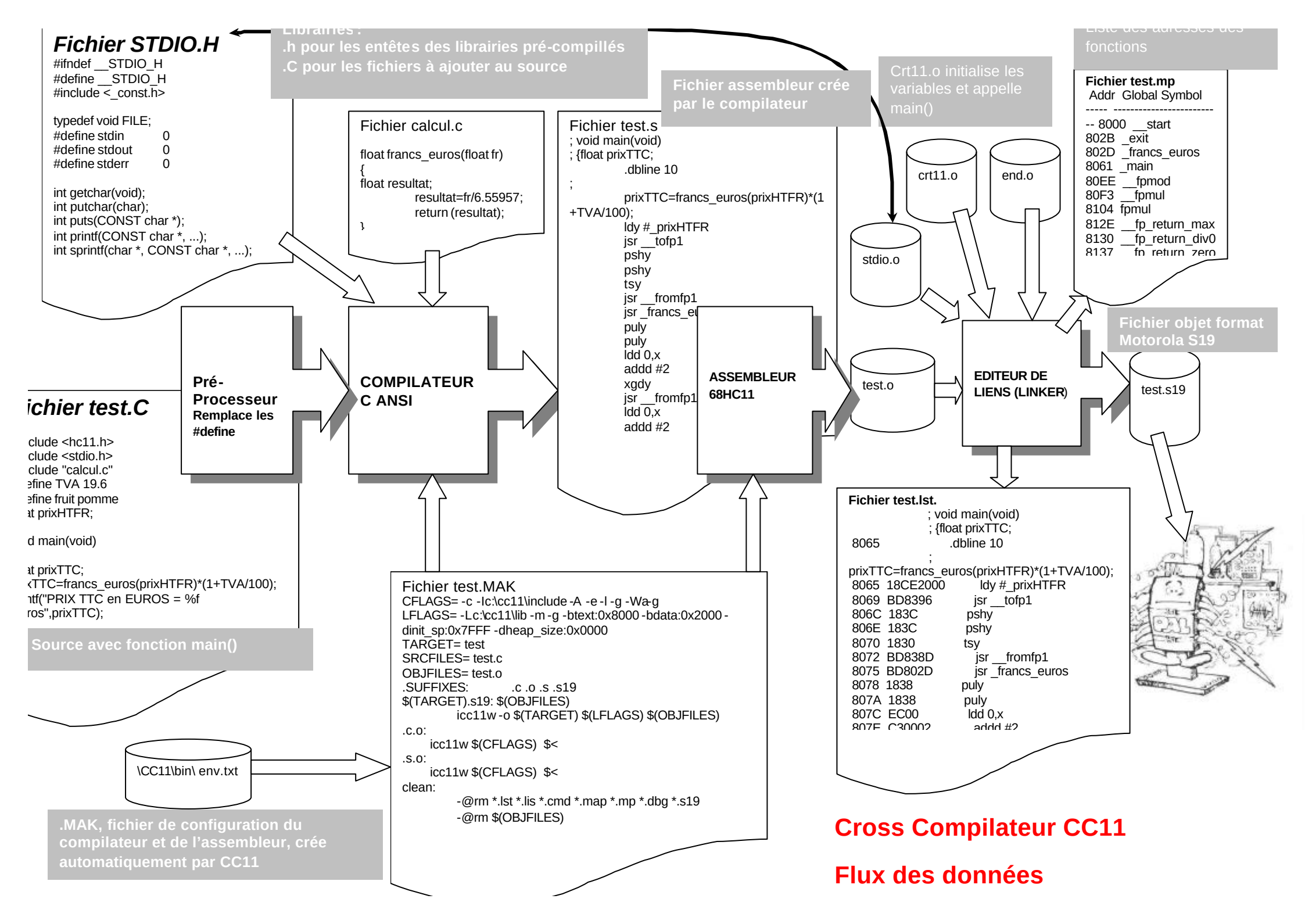
1.1. Répertoire BIN

Compile.exe	Programme DOS	Exécute des programmes DOS à partir de WINDOWS
Env.txt	Fichier de configuration	
las6811w.exe	Programme DOS	Assembleur
lcc11w.exe	Programme DOS	Pilote de compilateur
lccom11w.exe	Programme DOS	Compilateur C
lcppw.exe	Programme DOS	Préprocesseur
llibw.exe	Programme DOS	Gestionnaire de bibliothèques pré-compilées
llinkw.exe	Programme DOS	Editeur de liens
lmake.exe	Programme DOS	Make

2. Aspect CC11






Fichier STDIO.H
#ifndef __STDIO_H
#define __STDIO_H
#include <_const.h>
typedef void FILE;
#define stdin 0
#define stdout 0
#define stderr 0
int getchar(void);
int putchar(char);
int puts(CONST char *);
int printf(CONST char *, ...);
int sprintf(char *, CONST char *, ...);
Librairies :
.h pour les entêtes des librairies pré-compillés
.C pour les fichiers à ajouter au source
Fichier calcul.c
float francs_euros(float fr)
{
float resultat;
resultat=fr/6.55957;
return (resultat);
}
Fichier assembleur crée par le compilateur
Fichier test.s
; void main(void)
; {float prixTTC;
.dbline 10
;
prixTTC=francs_euros(prixHTFR)*(1+TVA/100);
ldy #_prixHTFR
jsr __tofp1
pshy
pshy
tsy
jsr __fromfp1
jsr _francs_eu
puly
puly
ldd 0,x
addd #2
xgdy
jsr __fromfp1
ldd 0,x
addd #2
Crt11.o initialise les variables et appelle main()
Liste des adresses des fonctions
Fichier test.mp
Addr Global Symbol
-- 8000 __start
802B _exit
802D _francs_euros
8061 _main
80EE __fpmod
80F3 __fpmul
8104 fpmul
812E __fp_return_max
8130 __fp_return_div0
8137 __fp_return_zero
crt11.o
end.o
stdio.o
Pré-Processeur
Remplace les #define
COMPILATEUR C ANSI
ASSEMBLEUR 68HC11
test.o
EDITEUR DE LIENS (LINKER)
test.s19
Fichier objet format Motorola S19
ichier test.C
clude <hc11.h>
clude <stdio.h>
clude "calcul.c"
efine TVA 19.6
efine fruit pomme
t prixHTFR;
d main(void)
t prixTTC;
TTC=francs_euros(prixHTFR)*(1+TVA/100);
tf("PRIX TTC en EUROS = %f
ros",prixTTC);
Source avec fonction main()
Fichier test.lst.
; void main(void)
; {float prixTTC;
8065 .dbline 10
;
prixTTC=francs_euros(prixHTFR)*(1+TVA/100);
8065 18CE2000 ldy #_prixHTFR
8069 BD8396 jsr __tofp1
806C 183C pshy
806E 183C pshy
8070 1830 tsy
8072 BD838D jsr __fromfp1
8075 BD802D jsr _francs_euros
8078 1838 puly
807A 1838 puly
807C EC00 ldd 0,x
807E C30002 addd #2
Fichier test.MAK
CFLAGS= -c -Ic:\cc11\include -A -e -l -g -Wa-g
LFLAGS= -Lc:\cc11\lib -m -g -btext:0x8000 -bdata:0x2000 -dinit_sp:0x7FFF -dheap_size:0x0000
TARGET= test
SRCFILES= test.c
OBJFILES= test.o
.SUFFIXES: .c .o .s .s19
$(TARGET).s19: $(OBJFILES)
icc11w -o $(TARGET) $(LFLAGS) $(OBJFILES)
.c.o:
icc11w $(CFLAGS) $<
.s.o:
icc11w $(CFLAGS) $<
clean:
-@rm *.lst *.lis *.cmd *.map *.mp *.dbg *.s19
-@rm $(OBJFILES)
\CC11\bin\ env.txt
.MAK, fichier de configuration du compilateur et de l'assembleur, crée automatiquement par CC11
Cross Compilateur CC11
Flux des données

IMAKE Utilitaire Make

Cet utilitaire est utilisé pour maintenir, mettre à jour et régénérer un groupe de programme.
Si aucun makefile n'est spécifié avec l'option -f, l'utilitaire lit un fichier appelé " makefile " s'il existe.
Si aucune cible n'est spécifiée sur la ligne de commande, imake se sert de la première cible définie dans le makefile. S'il doit construire une cible, et qu'aucune règle ne peut être dérivée pour la construire, imake retourne une erreur et s'arrête.

ICCPW Pré-processeur

Le pré-processeur lit les fichiers d'entrée et exécute les directives dans le fichier. Les fichiers d'entrée ont couramment une extension .c et les fichiers de sortie, une extension .i. Si le fichier de sortie n'est pas spécifié, la sortie est la sortie standard.

ICCOM11W Générateur de code

C'est le cœur du compilateur ; il prend en entrée un fichier C préparé et génère en sortie un fichier assembleur. Le compilateur accepte le standard ANSI C mais pas l'ancien style K&R. Généralement, le fichier d'entrée a une extension .i et le fichier de sortie .s. Si vous ne spécifiez pas de fichier de sortie alors celui-ci est écrit sur la sortie standard.

IAS6811W Assembleur

L'assembleur crée un fichier objet relogeable.

ILINKW Le Linker

ilinkw combine les fichiers objets entre eux pour former un exécutable. Il inclue automatiquement les fichiers crt11.o, end11.o et la librairie libc11.a. Le linker cherche dans les librairies après que le fichier objet soit construit, aussi, vous pouvez placer une librairie n'importe où dans la ligne de commande. Vous pouvez vous servir de l'option -L pour spécifier où les librairies se trouvent

ILIBW Gestionnaire de librairie

Cet utilitaire permet de créer et de manipuler les librairies. Les fichiers librairies (par exemple libc11.a) doivent apparaître avant des fichiers objet après les options.
Les options valides sont:
-a Ajoute le module aux archives, crée l'archive si elle n'existe pas. Si le module existe déjà, il sera remplacé.
-t affiche les noms des modules de l'archive,
-x Extrait les modules de l'archive.
-d Efface les modules de l'archive.

Exemples
Pour remplacer puchar.o dans la librairie par une nouvelle version:
ilibw -a libc11.a putchar.o
Pour connaître le contenu de la librairie:
ilibw -t libc11.a

3. Librairie Hc11.h

Ce fichier contient des déclarations pour les registres du 68HC11. Il faut peut-être changer ce fichier selon les caractéristiques de la carte cible

Le fichier contient des lignes de déclaration.

```
#define _IO_BASE      0x1000
#define PORTA         *(unsigned char volatile *)(_IO_BASE + 0x00)
```

0x1000 est transformé en une adresse par un CAST et PORTA est un pointeur sur l'adresse 0x1000

Ces déclarations vous permettent d'écrire dans un programme les opérations suivantes.

PORTA = 0x08;	**mettre le port à 08**
i = PORTA;	**lire le port**
PORTA |= 0x08;	**mettre le bit 3 à 1**
PORTA &= ~0x08;	**mettre le bit 3 à 0**
if (PORTA & 0x08)	**examiner le bit 3**

4. Exemple 1: Faire clignoter une LED

Sélectionner le menu Fichier/configuration et vérifier que la configuration de la cible est correcte

Charger le fichier : c:\cc11\exoccbf1\flash.c

Cliquez compiler, la fenêtre suivante s'ouvre

```
imake -f flash.mak
icc11w -c -Ic:\cc11\include -A -e -l -g -Wa-g  flash.c
icc11w -o flash -Lc:\cc11\lib -m -g -btext:0x8000 -bdata:0x2000 -dinit_sp:0x7FFF -dheap_size:0x0000
flash.o
done
```

Le compilateur crée un fichier objet flash.o qui sera lié avec les éventuelles bibliothèques afin de créer le fichier flash.s19.
Dans la fenêtre du débogueur cliquer STOP, puis CHARGER, puis GO, la LED doit clignoter

Visualiser alors les fenêtres flash.c et flash.lis

FLASH.C

```
/*      Clignoter une LED sur Controlboy F1
*/
#include <hc11.h>
void wait(int cnt)
{
        for (;cnt>0; cnt--);
}

void main(void)
{int x;
        DDRG  = 0x01;                   /*
PG0 = sortie */
        for(;;)
        {
                PORTG ^= 0x01;
                /* basculer led */
                if (PORTG&0x2) x= 20000;
                else x=5000;
                wait(x);
        }
}
```

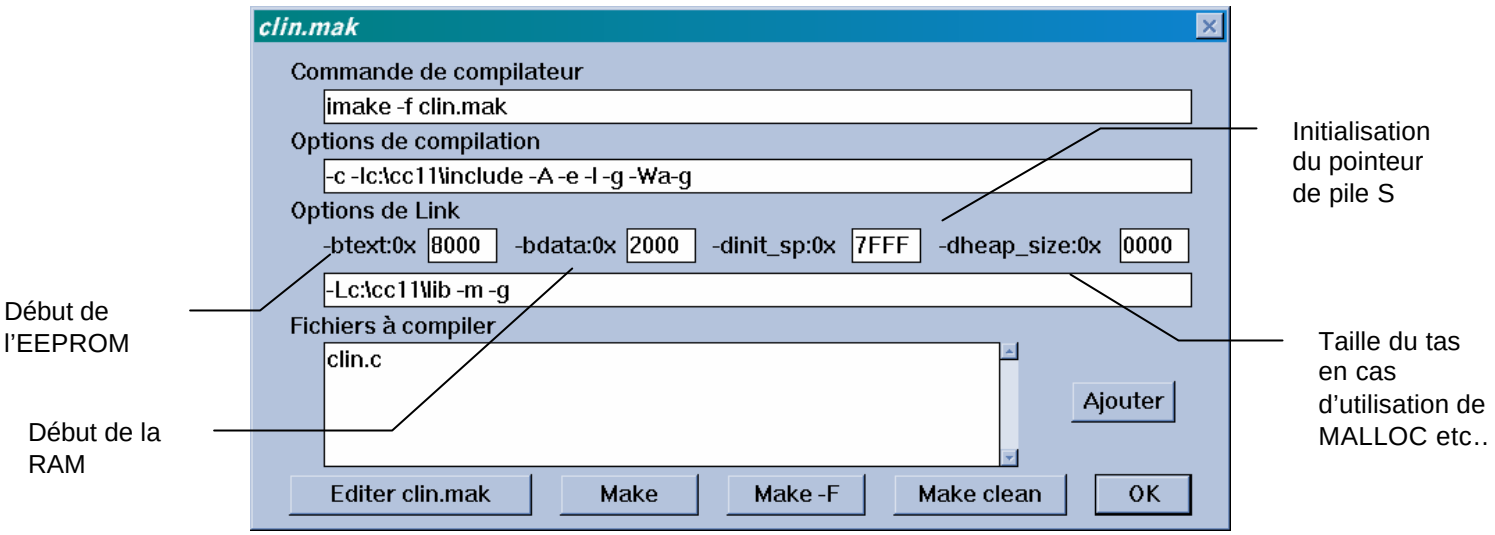
FLASH.LIS (aperçu partiel)

```
                ; void main(void)
                ; {int x;
001F                .dbline 10
                ;     DDRG  = 0x01;          /* PG0 = sortie */
001F C601             ldab #1
0021 F71003           stab 0x1003
0024                .dbline 11
                ;     for(;;)
0024              L8:
0024                .dbline 12{
                ;     {
0024                .dbline 13
                ;         PORTG ^= 0x01;          /* basculer led
*/
                ; vol
0024 F61002           ldab 0x1002
0027 C801             eorb #1
0029 F71002           stab 0x1002
002C                .dbline 14
                ;         if (PORTG&0x2) x= 20000;
002C 18CE1002         ldy #0x1002
0030 181F000207       brclr 0,y,#2,L12
0035                .dbline 14
0035 CC4E20           ldd #20000
0038 ED02             std 2,x
003A 2005             bra L13
003C              L12:
003C                .dbline 15
                ;         else x=5000;
003C CC1388           ldd #5000
003F ED02             std 2,x
0041              L13:
0041                .dbline 16
                ;         wait(x);
0041 EC02             ldd 2,x
0043 BD0000           jsr _wait
0046                .dbline 17}
                ;     }
0046                .dbline 11
0046                .dbline 11
```

Les fenêtres flash.lis permet de visualiser le fichier assembleur généré par le compilateur, les fenêtres flash.lis et flash.c font apparaître sur leur gauche les adresses physiques des instructions et fonctions.
*Tapez à l'invite du débogueur **d x** pour afficher la valeur de la variable x*
Pour placer ou retirer un point d'arrêt, il suffit de cliquer sur l'adresse du point d'arrêt.

Arrêter le programme puis le faire avancer en pas et pas
STEP exécute une seul instruction assembleur et NEXT un sous programme

Cliquez sur OPTIONS, FLASH.MAK et vérifiez les paramètres pour la mémoire de votre cible.



imake qui lancera lui même le compilateur C icc11w. avec de nombreuses options dont :
-btext:0x8000 : indique le début de l'EEPROM
-bdata:0x2000 : indique le début de la RAM
-dinit_sp:0x7FFF: indique l'adresse initiale pointé par S
-dheap_size:0x0000 : indique la taille du TAS (heap) dans le cas d'allocation dynamique de la mémoire, (malloc, calloc, free etc.)

Fichiers CRT11.S et END11.S
CRT11.O et END11.O sont inclus au début et à la fin du fichier objet avant linkage.
CRT11.S initialise la pile S, charge le segment DATA, initialise BSS à 0 et exécute la fonction main
END11.S contient quelques déclarations pour calculer les tailles des segments.

CRT11.S
```
; Do not define any .area above this line. .text must be the first
; line in this file.  You may put .area(s) at the end of the file.
;
; You need to define two symbols on the link command line
;           init_sp             : initial stack pointer
; The entry point of your program is _start, which is defined here.
;
            .text
__start::                       ; entry point
            lds         #init_sp
            cli
; clear BSS
            clra
            ldx         #__bss_start
init_loop:
            cpx         #__bss_end
            beq         init_done
            staa        0,x
            inx
            bra         init_loop

init_done:

; copy initialized idata to data
; idata in ppage and data in dpage
            ldx         #__idata_start
            ldy         #__data_start
copy_loop:
            cpx         #__idata_end
            beq copy_done
            ldab 0,x
            stab 0,y
            inx
            iny
            bra copy_loop
copy_done:

; call user main routine
            jsr _main
_exit::
            bra         _exit
```

END11.S
```
            .area text
__text_end::
            .area idata
__idata_end::
            .area data
__data_end::
            .area bss
__bss_end::
```

5. Exemple 2: printf sur afficheur LCD

Toutes les fonctions de sortie s'appuient sur putchar(char) ; c'est cette fonction qui indique le périphérique de sortie, par défaut putchar envoie un caractère sur l'afficheur à cristaux liquides
Analyser et tester le programme \cc11\exemples\print.c

6. Création d'une bibliothèque

Exercice : créer une bibliothèque en c.
Le programme print.c contient toutes les fonctions permettant de gérer l'afficheur et le clavier. A partir de print.c, on peut créer une bibliothèque, pour cela on retire la fonction main et les prototypes. On enregistre le reste dans un fichier lcdf1.c *par exemple* que l'on placera dans le répertoire libsrc. Il suffira désormais d'insérer la directive **#include " \cc11\libsrc\lcdf1.c"** pour disposer des fonctions de gestion de l'afficheur et du clavier.
Pour tester votre bibliothèque, créer un programme tstlcd.c dans le répertoire travail qui affichera votre prénom sur l'afficheur LCD

Afin d'éviter de recompiler systématiquement les bibliothèques, on peut créer une bibliothèque objet (*.o) qui sera liée au programme grâce à l'option –l de imake
Charger le fichier lcdf1.c et le compiler, un message d'erreur normal (pas de fonction main) apparaît mais un fichier lcdf1.o a été crée dans libscr. Cliquez sur menu démarrer puis exécuter et tapez :
\CC11\BIN\ILIBW –a \cc11\lib\liblcdf1.a \cc11\libsrc\lcdf1.o

Une bibliothèque liblcdf1.a est crée.
Pour l'utiliser vous devez créer un fichier lcdf1.h dans le répertoire « include » contenant les prototypes des fonctions de la bibliothèque. Dans tstlcd.c retirer le #include et ajouter #include <lcdf1.h>
Avant de compiler tstlcd.c taper –llcdf1 dans imake (le premier l pour link une bibliothèque pré compilée)

Le répertoire /cc11/lib contient après l'installation trois librairies pré-compilées :
Libc11.a La librairie de routines standard. Le printf() est limité et n'affiche ni des variables de type long ni ceux de type float. Le link inclut cette librairie automatiquement.
Liblng11.a Contient un printf() qui affiche les variables de type long. Pour inclure cette fonction, ajoutez dans l'option de link -llng11 .
Libfp11.a Contient un printf() qui affiche les variables de type long et ceux de type float. Pour inclure cette fonction, ajoutez dans l'option de link -lfp11 .

7. Création d'une bibliothèque partagée

Le temps de chargement d'un programme gérant l'afficheur et le clavier est prohibitif. CC11 peut intégrer des bibliothèques partagées qui sont implantées en permanence en EEPROM. Le temps de chargement en est considérablement réduit.
Toutes les informations permettant la création d'une bibliothèque partagée se trouvent dans le répertoire libshare. Les programmes utilisant la bibliothèque partagée doivent être compilé avec l'option **–lshare**
Remarque : modifications de la bibliothèque d'origine

cboyf1pr.c, gestion du clavier et de l'afficheur à cristaux liquides sur CBOYF1
ce fichier d'origine cboy a été modifié afin d'inclure les fonctions void gotox(char) et char keyget(int)

gotox : permet de positionner le curseur sur l'écran (x=1 à 16)
keyget retourne le code ASCII de la touche enfoncée 0 sinon
lycee.h doit être recopié dans include, il déclare lcdinit et keyget qui ne sont pas standards

le fichier **libshare.mak** a été également modifié pour inclure ces deux dernières fonctions
Pour créer la bibliothèque partagée, recopier **libshare.mak** et **cboyf1pr.c** dans libshare

Dans un programme pour utiliser le clavier et l'afficheur il faut déclarer
#include <lcdclav.h> déclaration de lcdinit et keyget
#include <stdio.h> pour puts getchar printf etc.
Exercice : Compiler libshare.mak puis l'implanter en EEPROM. Créer ensuite le fichier tstlcd identique à print.c mais utilisant la bibliothèque partagée.

8. **Librairie C partagée lycee.h sur CBOYF1**

(cboyf1pr.c modifiée), version lycée Fourcade 11/2000
Utiliser #include lycee.h

void lcdinit(int lines) // initialise l'afficheur LCD lines vaut 1 ou 2
int putchar(char c) // écrit le caractère c sur l'afficheur à l'emplacement du curseur
void gotox(char c) // place le curseur à l'emplacement c

Afficheur 1 ligne

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Afficheur 2 lignes

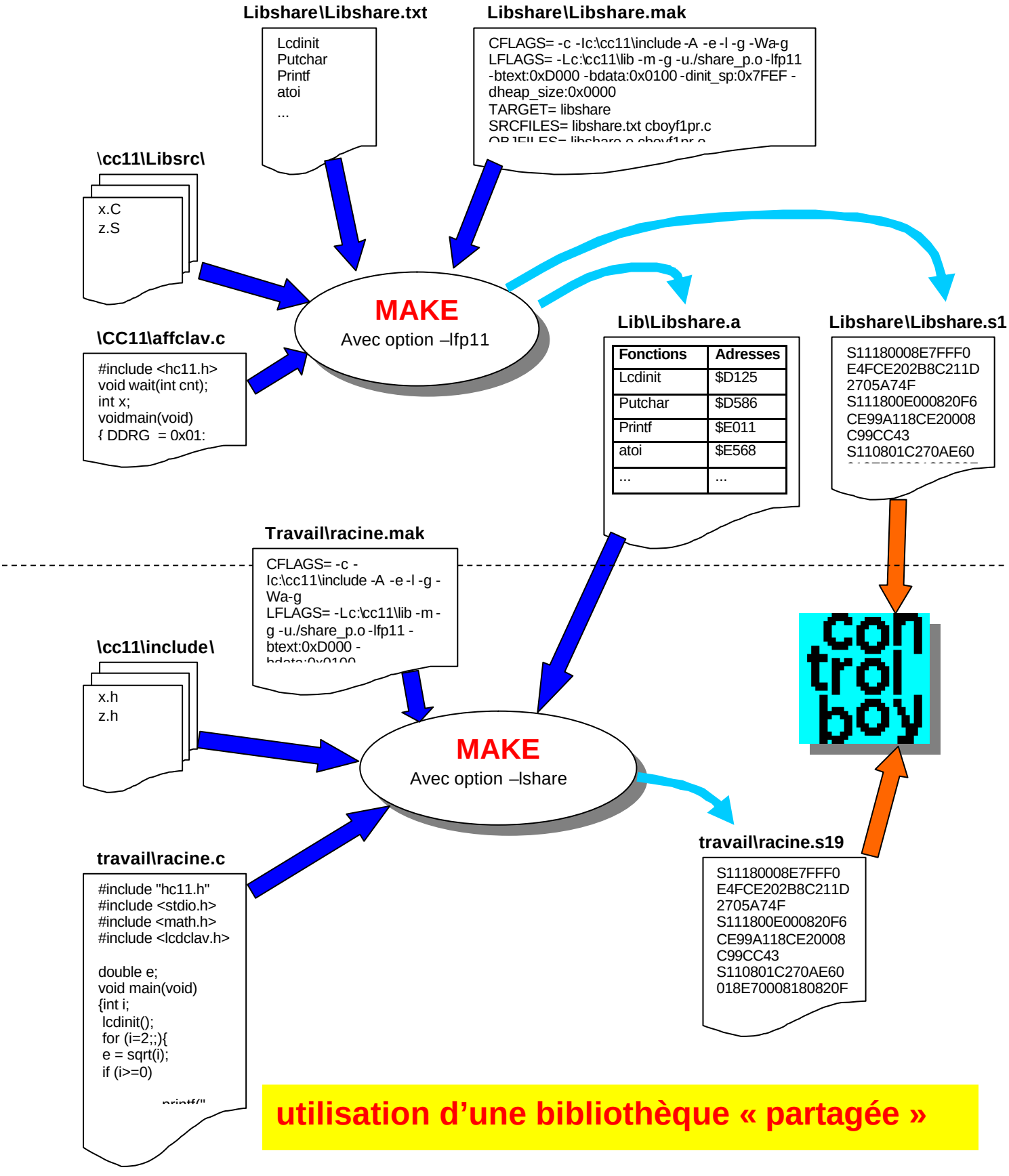
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32

char keyget(void) //retourne le code ASCII de la touche enfoncée et ? si inconnue
// clavier 3x4 par défaut , faire **opt_key4x4=1** pour activer la gestion 4x4
unsigned char analogin(unsigned char ch) // retourne la valeur sur le l'entrée ch du CAN
void analogout(unsigned char ch, unsigned char val)// envoie val sur le canal ch du CNA
void wreeprom(unsigned char *adr, unsigned char val)// écrit val à l'adresse adr de l'EEPROM
void initsci(void) // initialise la gestion des communications en interruption
char getsci(void) // retourne le premier caractère reçu sur SCI
void putsci(char c) // émet c sur SCI
char *getstsci(char *s) // lit une chaîne de caractère sur SCI se terminant par finst
// la variable finst contient le caractère de fin de chaîne (* par défaut)
int putstsci(char *s) // émet la chaîne s (finit par 0)

Liste de fonctions mathématiques, et de gestion de chaînes

__divu	_isxdigit	__lrsh	__fp_to_int	__pi
__divi	_tolower	__lirsh	__fp2int	_tan
__modu	_toupper	__lursh	_fabs	__maxnum
__modi		__luirsh	__fpadd	_degree_to_
__muli	__ly2reg	__land	__fpcmp	radian
__mulu	__ly2reg_x	__lor	__fpdiv	_radian_to_
	__ly2reg2	__lxor	_exp10	degree
_atoi	__lreg2y	__lneg	_exp	__c_fp2long
_atol	__lreg2y_x	__lneg2	_pow	_acos
_itoa	__lreg22y	__lcom	__fp_to_string	_asin
_ltoa	__lregmov	__lcmp	__int2fp	
	__d2lreg	__c_long2fp	__uint2fp	
_printf	__d2lreg2		__long2fp	
__va_start	__ud2lreg	__fp_is_acc1_zero	__ulong2fp	
	__ud2lreg2	__fp_xfer_acc1_to_acc2	__fp2long	
_memcpy	__lreg2d	__push_fpacc2	_log10	
_strchr	__lret	__pull_fpacc2	_log	
_strcpy	__pshlr1	__to_fpacc1	__fpmod	
_strlen	__pshlr2	__to_fpacc2	__fpmul	
	__pullr1	__from_fpacc1	__fp_return_max	
_isalnum	__pullr2	__from_fpacc2	__fp_return_div0	
_isalpha	__ladd	__fpneg	__fp_return_zero	
_iscntrl	__lsub	__pulfp1	__fp_remcomp	
_isdigit	__lmul	__pulfp2	_sqrt	
_isgraph	__ldiv	__pshfp1	_atof	
_islower	__ludiv	__pshfp2	__fpsub	
_isprint	__lmod	__fromfp1	__fp_intfrac	
_ispunct	__lumod	__tofp1	_atan	
_isspace	__llsh	__tofp2	_sin	
_isupper	__lilsh	__fpmov	_cos	

Création d'une bibliothèque « partagée »



utilisation d'une bibliothèque « partagée »

9. Exemple 3: Virgule flottante

Le programme calcule la racine carrée de 2 et l'affiche sur l'afficheur LCD.
Il faut ajouter -lfp11 dans la ligne OPTIONS DE LINK dans la fenêtre FLOAT.MAK.afin de pouvoir utiliser printf avec des réels. *Sauf si cette bibliothèque se trouve en EEPROM (utiliser alors –lshare)*

Fichier tstfloat.c

```
// Programme sur Controlboy F1: printf virgule flottante sur LCD
#include "hc11.h"
#include <stdio.h>
#include <math.h>
#include <lycee.h>
double e;
void main(void)
{     int i;
      lcdinit(1); //1 pour un lcd 1 ligne sinon 2
      gotox(1);
      puts("sqrt(");
      for (i=2;;){
            e = sqrt(i);
            if (i>=0)
                  {
                  gotox(6);
                  printf("%d)=%f", i, e);
                  }
            i=keyget(0)-'0';
      }
}
```

10. Caractéristiques du compilateur

Sections

Le compilateur stocke des données dans trois segments.

- Le segment TEXT contient les instructions du programme. Ce segment sera chargé dans l'EPROM ou dans l'EEPROM de la cible.
  Les constantes sont stockés dans le segment TEXT.
- Le segment DATA contient des données initialisées. Ce segment se trouve dans la RAM de la cible. Les données sont stockées dans un segment IDATA qui se trouve dans l'EEPROM. Avant de lancer le programme C, ces données sont copiées de l'EEPROM dans la mémoire vive.
- Le segment BSS contient les données non initialisées. Ce segment est mis à zéro avant de lancer le programme C.

*Pour utiliser l'allocation dynamique de mémoire de la bibliothèque, il faut prévoir de la mémoire pour le tas. Les fichier *.mp contiennent les informations d'occupation mémoire.*

Exemples

char a[100]; La variable se trouve dans le segment BSS dans la RAM. La variable est mise à zéro avant de lancer le programme.

char b[]="Controlboy"; La variable se trouve dans le segment DATA dans la RAM. Les données sont stockées dans l'EEPROM et sont copiées dans la RAM avant de lancer le programme.

const char c[]="Controlboy"; La variable avec ses données se trouve dans le segment TEXT dans l'EEPROM. Le programme ne peut pas changer la variable.

11. Routine d'interruption

De par la structure du 68HC11, les programmes utilisent de nombreuses interruptions
Exemple : le fichier \cc11\exemples\it.c

```
/* ce programme traite l'IT RTI et incrémente les secondes et les minutes*/
#include <hc11.h>
int second;
int minute;

#pragma interrupt_handler RtiInt
void RtiInt(void)
{     static int tictac = 0;
      if (++tictac >= 1953){        /* 1s avec Q=16MHz   */
            tictac = 0;
            if (++second >= 60) {
                  second = 0;
                  minute++;
            }
      }
      TFLG2 |= 0x40;
}

#pragma abs_address:0xFFF0
void (*rtiint_vector)(void) = RtiInt ;
#pragma end_abs_address

void main(void)
{
      PACTL &= 0xFC;          /* selectioner la vitesse (n/1) */
      TMSK2 |= 0x40;          /* declencher le timer */
      asm(" cli ");           /* autoriser les interruptions */
      while(1) ; /* le compilateur respectant la norme ANSI refuse les
boucles infinies du type for( ;;) */
}
```

Indique au compilateur de terminer cette fonction par RTI et non RTS

Positionnement du vecteur d'interruption, le pragma force le compilateur à placer le code à une adresse imposée

Déclaration d'un pointeur sur une fonction dont l'adresse est ici RTIiInt

Après avoir chargé et lancé le programme dans la cible, tapez dans la fenêtre du débogueur pour vérifier la marche du temps.

- **d minute, second**

Pour vérifier qu'un programme exécute une interruption, il suffit de placer un point d'arrêt dans celle ci

12. Répertoire /cc11/libsrc

Il contient les sources de tous les fichiers de la librairie standard C ANSI.
Un fichier makefile libc.mak pour compiler tous les fichiers de la librairie et installer les librairies dans /cc11/lib
Les sources de crt11.s et end11.s
Un fichier crt11.mak pour compiler ces fichiers et installer leurs fichiers objet.

12.1. Ctype.h: Classification de caractère

int isalnum (int c) retourne non nul si C est un digit ou alphabétique.
int islpha (int c) retourne non nul si C est alphabétique.
int iscntrl (int c) retourne non nul si C est un caractère de contrôle (en général, LF, FF, BELL...etc).
int isdigit (int c) retourne non nul si C est un digit.
int isgraph (int c) retourne non nul si C est un caractère imprimable et non un espace.
int islower(int c) retourne non nul si C est un caractère minuscule.
int isprint(int c) retourne non nul si C est un caractère imprimable.
int ispunct(int c) retourne non nul si C est caractère imprimable sans être le caractère espace ou un digit ou an alphabétique.
int isspace(int c) retourne non nul si C est le caractère espace ou un caractère du type CR, FF, HT, NL, and VT.
int isupper(int c) retourne non nul si C est an caractère majuscule.
int isxdigit(int c) retourne non nul si C est un digit hexadécimal.
int tolower(int c) retourne le caractère C en minuscule.
int toupper(int c) retourne le caractère C en majuscule

12.2. Float.h: Caractéristiques de nombres en virgule flottante

Bibliothèque assembleur fp*.s

12.3. Math.h: Fonctions virgule flottante

double exp(double x) retourne l'exponentielle x.
double fabs(double x) retourne la valeur absolue de x.
double fmod(double x, double y) retourne le reste de la division x / y.
double log(double x) retourne le logarithme népérien de x.
double log10(double x) retourne le logarithme décimal de x.
double pow(double x, double y) retourne x élevée à la puissance y.
double sqrt(double x) retourne la racine carrée de x.
double sin(double x) retourne le sinus de x exprimé en radians.
double cos(double x) retourne le cosinus de x exprimé en radians.
double tan(double x) retourne la tangente de x exprimé en radians.
double asin(double x) retourne l'arc sinus de x exprimé en radians.
double acos(double x) retourne l'arc cosinus de x exprimé en radians.
double atan(double x) retourne l'arc tangente de x exprimé en radians.

12.4. **Setjmp.h** : défini le type jmp_buf.

int setjmp(jmp_buf buffer) retourne 0 lors du premier appel, et retourne une valeur non nulle quand il retourne par un longjmp()
void longjmp(jmp_buf buffer, int retval) retourne au point defiéné par setjmp()

12.5. Stdargs.h : Arguments variables

stdarg.h permet d'accéder aux variables de type varargs comme dans le printf(char *fmt, …)
va_start(va_list foo, <last-arg>) initialise une variable foo.
va_arg(va_list foo, <promoted type>) accède à l'argument suivant.
va_end(va_list foo) finit l'accès aux variables.

Par exemple printf() peut être utilisé en se servant de vfprintf() comme suit:

```
#include <stdarg.h>
int printf(char *fmt, ...)
{
va_list ap;
va_start(ap, fmt);
vfprintf(fmt, ap);
va_end(ap);
}
```

12.6. Stdio.h: Entrée standard, Sortie standard

int getchar() retourne un caractère de la sortie standard par exemple d'un clavier. C'est votre programme principal qui doit inclure cette fonction.
int printf(char *fmt, ..) écrit une chaîne formatée en accord avec les formateurs dans la chaîne fmt. Les formateurs sont issus du standard C:
%d écrit l'argument qui suit comme un entier décimal
%o - écrit l'argument qui suit comme un entier octal non signé
%x - écrit l'argument qui suit comme un entier hexadécimal non signé
%u - écrit l'argument qui suit comme un entier décimal non signé
%s - écrit l'argument qui suit comme une chaîne de caractère C terminé par le caractère nul.
%c - écrit l'argument qui suit comme un caractère ASCII
%f - écrit l'argument qui suit comme un nombre à virgule flottante (Vous devrez inclure la librairie libfp.a pour vous servir de ce formateur)
Si vous ajoutez la lettre 'l' entre % et un des formateurs du type entier, alors l'argument sera interprété comme un long à la place d'un int. (Vous devrez inclure la librairie liblng.a pour vous servir de ce formateur)
int putchar(int c) écrit un seul caractère sur la sortie standard par exemple sur un afficheur LCD ou sur le port SCI. C'est votre programme principal qui doit inclure cette fonction.
int puts(char *s) écrit un type string suivi par NL. Cela utilise la fonction putchar().
int sprintf(char *buf, char *fmt) écrit un texte formaté dans buf en utilisant les formateurs définis dans fmt. Les formateurs sont identiques à ceux employés dans la fonction printf().
De plus, pour faciliter l'exportation du programme vers un autre environnement, stdout et stderr sont définis comme 0, et FILE est typedefed comme void, et fprintf(FILE *, char *fmt, ...) est identique a la fonction printf(char *fmt, ..).

12.7. Stdlib.h : Fonctions Standard Library

int abs(int i) retourne la valeur absolue de i.
int atoi(char *s) converti la chaîne de caractère en un entier ou retourne 0 si une erreur est détectée.
double atof(const char *s) converti la chaîne de caractère en un double.
long atol(char *s) converti la chaîne de caractère en un long ou retourne 0 si une erreur survient.
void *calloc(size_t nelem, size_t size) retourne un pointeur sur un block mémoire de largeur contenant nelem objets, chacun de taille “size.” Cette mémoire est initialisé à zéro. Cette mémoire est allouée sur le tas (i.e., vous devez appeler la fonction _NewHeap() avant). Cette fonction retourne 0 si elle ne peut allouer cette mémoire.
void exit(status) met fin au programme. La valeur de sortie est écrite dans le registre D.
void free(void *ptr) libère la mémoire allouée sur le tas.
void *malloc(size_t size) alloue un block mémoire de taille “size” sur le tas. Elle retourne 0 si la mémoire ne peut être alloué.

int rand(void) retourne un nombre pseudo-aléatoire compris entre 0 et RAND_MAX.
void *realloc(void *ptr, size_t size) réalloue un block mémoire précédemment alloué mais de nouvelle taille.
Void srand(unsigned seed) initialise la valeur seed pour rand().
long strtol(char *s, char **endptr, int base) converti le caractère s en un type long int en accord avec la base choisie. Si base vaut 0, alors strtol choisi sa base en fonction du premier caractère de s (en tenant compte d'un éventuel signe moins (-)): 0x ou 0X indique un entier hexadécimal, 0 indique un entier octal, un entier décimal sera choisi sinon. Si endptr n'est pas NULL, alors *endptr sera définie en fin de conversion de s.
unsigned long strtoul(char *s, char **endptr, int base) est identique à la fonction strtol(), sauf que l'entier converti est de type unsigned long et que la valeur retourné est aussi de type unsigned long.

12.8. String.h: Fonctions sur caractères

void *memchr(void *s, int c, size_t n) cherche la première occurrence de c dans le tableau s de taille n.
Elle retourne l'adresse de l'élément recherché ou le pointeur NULL si aucune occurrence n'est trouvée.
int memcmp(void *s1, void *s2, size_t n) compare deux tableau, chacun de taille n. Cette fonction retourne 0 si les tableaux sont égaux et un nombre positif si le premier élément différent de s1 est plus grand que celui de s2; dans le cas contraire elle retourne un entier négatif.
void *memmove(void *s1, void *s2, size_t n) copie s2 dans s1, chacun de taille n. Cette routine marche correctement même si s1 est plus grand que s2. Elle retourne s1.
void *memset(void *s, int c, size_t n) place c dans tous les éléments du tableau s de taille n. Elle retourne s.
char *strcat(char *s1, char *s2) concatène s2 à la suite de s1. Elle retourne s1.
char *strchr(char *s, int c) cherche la première occurrence de c dans s, incluant le caractère nulle de fin de chaîne. Elle retourne l'adresse de l'élément trouvé ou le pointeur NULL si aucune occurrence n'est trouvée.
int strcmp(char *s1, char *s2) compare deux chaînes de caractère. Elle retourne 0 si les deux chaînes sont identiques, et un entier positif si le premier élément différent de s1 est plus grand que celui correspondant de s2. Dans le cas contraire elle retourne un entier négatif.
char *strcpy(char *s1, char *s2) copie s2 dans s1. Elle retourne s1.
size_t strlen(char *s) retourne la longueur de la chaîne s.
char *strncat(char *s1, char *s2, size_t n) concatène jusqu'à n éléments, sans inclure le caractère nulle de fin de chaîne, de s2 vers s1. Elle replace le caractère de fin de chaîne à la suite de la nouvelle chaîne s1. Elle retourne s1.
int strncmp(char *s1, char *s2, size_t n) est identique à la fonction strcmp() sauf qu'elle compare au plus n caractères.
char *strncpy(char *s1, char *s2, size_t n) est identique à la fonction strcpy() sauf qu'elle copie au plus n caractères.
char *strpbrk(char *s1, char *s2) effectue la même recherche que strcspn() sauf qu'elle retourne un pointeur sur l'élément de s1 et NULL s'il n'y a pas d'occurrence.
char *strrchr(char *s, int c) cherche la dernière occurrence de c dans s et retourne un pointeur dessus. Elle retourne un pointeur NULLE si aucune occurrence n'apparaît.
Char *strstr(char *s1, char *s2) cherche la première occurrence de la chaîne s2 dans s. Elle retourne l'adresse de l'élément trouvé ou le pointeur NULL si aucune occurrence n'est trouvée.

13. Annexe : Réalisation d'un Filtre numérique

13.1. Filtre RII

Les plus simples à programmer (moins de coefficient) mais avec une phase non linéaire.
Le logiciel tel que DISPRO propose la synthèse de filtres numériques. Après avoir choisi le modèle analogique ainsi que les caractéristiques du filtre, sont fournis les coefficients de ce filtre sous la forme :

```
BUTTERWORTH LOWPASS FILTER OF ORDER 6  (FILE C:BUTTR06B.DAT)
          Sampling frequency  (Hz) 1000
          End of passband (Hz) 100
          Beginning of stopband (Hz) 200
          Maximum passband attenuation (dB) 3
          Minimum stopband attenuation (dB) 41.91784
           COEFFICIENTS OF SECOND-ORDER SECTIONS
                 Z^2 + D Z + E
              A ------------------
                 Z^2 + B Z + C

     A          D          E          B          C
 0.0829385  2.0000000  1.0000000 -1.4040879  0.7358417
 0.0674973  2.0000000  1.0000000 -1.1426797  0.4126687
 0.0609462  2.0000000  1.0000000 -1.0317751  0.2755600
```

Le filtre ci dessus est du 6ième Ordre

La forme est : $H(z) ? \frac{y(z)}{x(z)} ? A ? \frac{z^2 ? D ? z ? E}{z^2 ? B ? z ? C} ? \ldots\ldots\ldots ? \ldots\ldots\ldots$

Ordre 2 Ordre n

Pour l'ordre 2 :
Les z^2 et z correspondent à des avances. Pour les éliminer on multiplie les deux membres par z^{-n} (n correspond à l'ordre du filtre)

$$y ? z^{?2} ? x ? z^{?2} ? A ? \frac{z^2 ? D ? z ? E}{z^2 ? B ? z ? C}$$

après réduction on obtient : $y ? Ax ? ADxz^{?1} ? AExz^{?2} ? yBz^{?1} ? yCz^{?2}$

(rappel : xz^{-1} correspond à l'échantillon x(n-1))

Dans le cas général on a :

$$y_{(n)} ? A ? x_n ? Dx_{(n?1)} ? Ex_{(n?2)} ? ? ? By_{(n?1)} ? Cy_{(n?2)} ?$$

Les coéficient sont généralement inférieurs à 1 et en virgule flottante. En C cela ne pose pas de problème, en revanche pour réaliser une multiplication de ce type avec un uP en assembleur on essait de rendre le coéficent égal à une somme de type $a ? ? \limits_{n} 2^{?n}$ ce type d'opération est très simple à réaliser à l'aide de décalages à droite (un décalage à droite est une division par 2 donc une multiplication par 2^{-1})

2^{-n}	
-1	0.5
-2	0.25
-3	0.125
-4	0.0625
-5	0.03125
-6	0.015625
-7	0.007812
-8	0.00390625
-9	0.001953125

Exemple : Réaliser $a.e_n$ avec le coefficient **a=0.1**

-1	-2	-3	-4	-5	-65	-7	-8	-9
0	0	0	1	1	0	0	1	1

$$a ? e_n ? ? \limits_{n} 2^{?n} ? e_n ? 2^{?4} ? e_n ? 2^{?5} ? e_n ? 2^{?8} ? e_n ? 2^{?9} ? e_n ? 0.099609375$$

13.2. exemple : rejecteur 50Hz

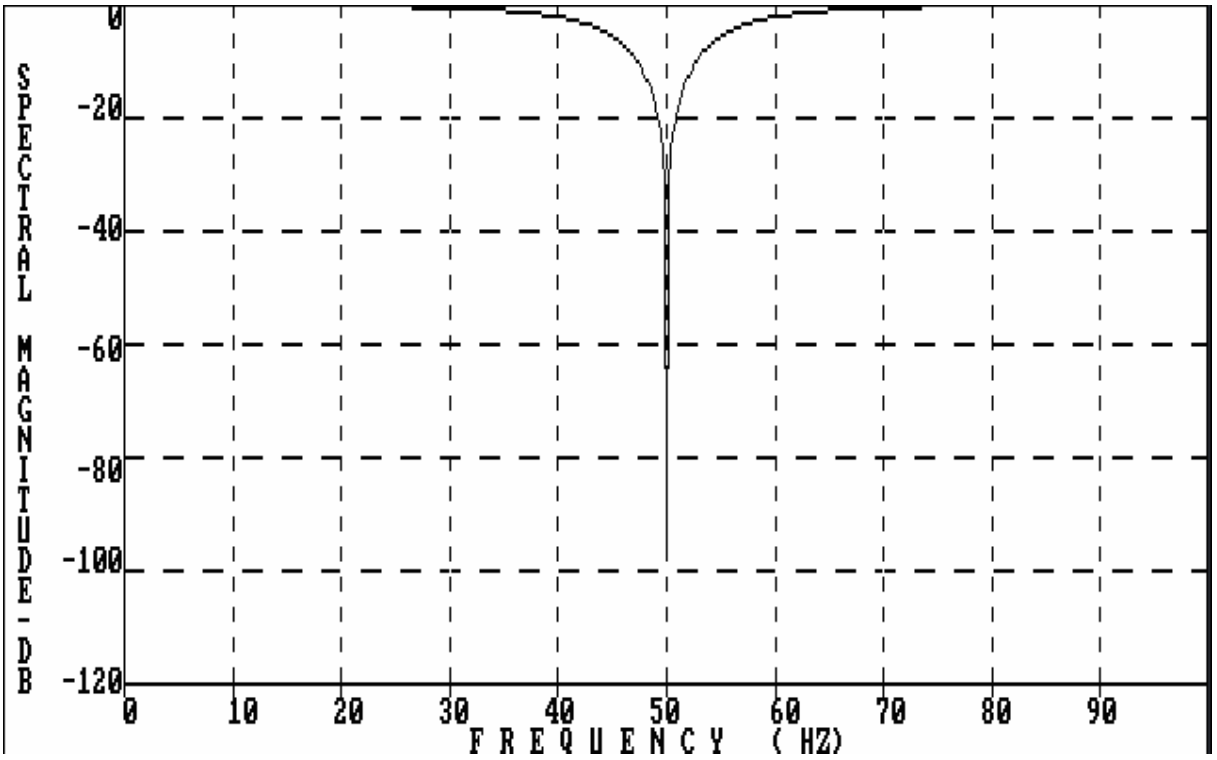
BUTTERWORTH BANDSTOP FILTER OF ORDER 2 (FILE D:BUTTR02B.DAT)

Sampling frequency (Hz) 200
End of lower passband (Hz) 42
Beginning of stopband (Hz) 48
End of stopband (Hz) 52
Beginning of upper passband (Hz) 58
Maximum passband attenuation (dB) 3
Minimum stopband attenuation (dB) 12.44915
Coefficient wordlength: floating point

COEFFICIENTS OF SECOND-ORDER SECTIONS

$$A\ \frac{Z^2 + D\,Z + E}{Z^2 + B\,Z + C}$$

A	D	E	B	C
0.7960849	0.0000000	1.0000000	0.0000000	0.5921699



après multiplication par z^{-2} on obtient

$y_{(n)} = A.x_n + Dx_{(n-1)} + Ex_{(n-2)} - By_{(n-1)} - Cy_{(n-2)}$

$y_{(n)} = Ax_n + ADx_{(n-1)} + AEx_{(n-2)} - By_{(n-1)} - Cy_{(n-2)}$

$$y_{(n)} = 0.7960849\ x_n + x_{(n-2)} - 0.5921699\ y_{(n-2)}$$

13.3. Filtre FIR

Ces filtres possèdent une phase linéaire, (la sortie ne dépend que des échantillons en entrée) mais le calcul de la sortie s'effectue sur un plus grand nombre d'échantillons que les FIR donc le temps de calcul est plus long.

Pratiquement un 68HC11F1 avec un quartz 16MHz effectue une opération du type *y0=ax0+bx1+cx2+dx3+ex4+fx5+gx6+hx7+ix8+jx9* en environ 5 mS, La fréquence d'échantillonnage ne peut donc excéder 200Hz, et les fréquences filtrables 50 à 80Hz

```
/*Filtre Numérique FIR passe bas */
/* CD 12/2000 avec bib  lycée*/
//y0=ax0+bx1+cx2+dx3+ex4+fx5+gx6+hx7+ix8+jx9
#include <hc11.h>       /*déclaration des bibliotheques*/
#include <lycee.h>
#define tech 22222u /* échantillonnage pour 180Hz (5.55mS) CBOYF1 16MHz*/
#define a -0.0385324
#define b 0.0762931
#define c 0.0347435
#define d 0.3035086
#define e 0.4541574
#define f 0.3035086
#define g 0.0347435
#define h -0.0762931
#define i -0.0385324
#define j 0.0

float x0,x1,x2,x2,x3,x4,x5,x6,x7,x8,x9,y0;

#pragma interrupt_handler oc2Int   /*déclaration de l'IT d'oc5*/
void oc2Int(void)                  /*routine d'interruption*/
{
      x9=x8;
      x8=x7;
      x7=x6;
      x6=x5;
      x5=x4;
      x4=x3;
      x3=x2;
      x2=x1;

      x0=0.0196*((float)(analogin(7)));
      y0=a*x0+b*x1+c*x2+d*x3+e*x4+f*x5+g*x6+h*x7+i*x8+j*x9;
// durée de calcul : environ 4.7 mS avec Q=16MHz
      x1=x0;
      analogout(1,(int)(51.0*y0));
      TOC2 +=tech;
      TFLG1 |= 0x40;
}

#pragma abs_address:0xFFE6      /*routine d'IT , vecteur RESET*/
void (*oc2int_vector)(void) = oc2Int ; /*adresse de oc5Int en FFEO*/
#pragma end_abs_address

void main(void)                          /*programme principal*/
{
      OPTION|=0x80;     //active CAN
      TMSK1 |=0x40;   /*IT de oc2 activée */
      TCTL1 &=0x3f;   /*aucune action définie sur le port A*/
      DDRA=1;
      asm(" cli ");   /*autoriser les interruptions*/
      while(1);
```

```
}
```

13.4. Correcteur PID numérique

A partir de la loi de commande analogique :

$$u(t) = K\left(\varepsilon(t) + \frac{1}{Ti}\int_0^t \varepsilon(x)dx + Td\frac{d\varepsilon(t)}{dt}\right)$$ *ε représente l'erreur*

on obtient l'équation numérique

$$u(k) = u(k-1) + K\left(1 + \frac{te}{Ti} + \frac{Td}{te}\right)\varepsilon(k) - \left(1 + 2\frac{Td}{te}\right)\varepsilon(k-1) + \frac{Td}{te}\varepsilon(k-2)$$

ε(k-n) représente l'échantillon n de l'erreur
u(k-n) représente l'échantillon n du signal de commande

13.5. Amélioration : Correcteur PID numérique par retour d'état : Tableau de Takahashi

Ce type de correcteur ne sature pas le signal de commande lors de brusques variations de consigne (il ne dérive pas l'erreur)
Rq : ε représente le temps d'échantillonnage te, Ki et Kd représente respectivement Ti et Td

	Essai indiciel τ, T	Pompage K_{osc} T_{osc}
PID	$K = \frac{1{,}2T}{(\tau + \Delta)} - 0{,}5K_i$ $K_i = \frac{0{,}6T\Delta}{(\tau + 0{,}5\Delta)^2}$ $K_d = \frac{0{,}6T}{\Delta}$ (*)	$K = 0{,}6K_{osc} - 0{,}5K_i$ $K_i = 1{,}2\,\frac{K_{osc}\Delta}{T_{osc}}$ (*) $K_d = 0{,}3\,\frac{K_{osc}T_{osc}}{4\Delta}$ (*)
PI avec $K_d = 0$	$K = \frac{0{,}9T}{\tau + 0{,}5\Delta} - 0{,}5K_i$ $K_i = \frac{0{,}9T\Delta}{3{,}3(\tau + 0{,}5\Delta)^2}$	$K = 0{,}45K_{osc} - 0{,}5K_i$ $K_i = \frac{0{,}45}{0{,}83}\,\frac{K_{osc}\Delta}{T_{osc}}$ (*)
P avec $K_d = 0$ et $K_i = 0$	$K = \frac{T}{\tau + \Delta}$	$K = 0{,}5K_{osc}$ (*)
(*) Ces valeurs sont identiques à celles de Ziegler-Nichols.		

$$u(k) = u(k-1) + K_i[y_c(k) - y(k)] + K[y(k-1) - y(k)] + K_d[2y(k-1) - y(k) - y(k-2)]$$

13.6. Approximation du correcteur PID par transformation bilinéaire

C'est la méthode la plus simple elle demande cependant pour être fiable une fréquence d'échantillonnage 4 à 5 fois supérieur à la fréquence de travail la plus élevée.

A partir de la loi de commande analogique

$$u(t) = K_p \varepsilon(t) + K_i \int_0^t \varepsilon(x)dx + K_d \frac{d\varepsilon(t)}{dt}$$

dont la fonction de transfert en p est

$U(p) = K_p.E(p) + K_i.\frac{E(p)}{p} + K_d.p.E(p)$ qui donne H(p) = $\frac{U(p)}{E(p)} = K_p + \frac{K_i}{p} + K_d.p$

La transformation bilinéaire consiste à remplacer p par $p = \frac{2}{te}\left(\frac{1 - z^{-1}}{1 + z^{-1}}\right)$

d'ou $H(z) = K_p + \frac{K_i.te(1 + z^{-1})}{2.(1 - z^{-1})} + K_d.\frac{2}{te}\left(\frac{1 - z^{-1}}{1 + z^{-1}}\right)$

$x.z^{-1}$ correspond à l'échantillon x(n-1) etc.

après réduction on obtient $u(n)=a_0e(n)+a_1 e(n-1)+a_2 e(n-2)+b_2u(n-2)$
avec

$a_0 = K_p + K_i\frac{te}{2} + K_d.\frac{2}{te}$

$a_1 = K_i.te - K_d.\frac{4}{te}$

$a_2 = -K_p + K_i\frac{te}{2} + K_d.\frac{2}{te}$

$b_2 = 1$

A partir de flash.c, clin.c, it.c, tstlcd.c

14. Exercices :

1. Faire clignoter la led verte avec une période de 1S (au plus prêt) en utilisant l'IT RTI
2. Produire un signal carré de fréquence 2,15KHz sur PA3 en utilisant l'interruption du TIMER 0C4
3. Produire un signal rectangulaire de fréquence 2,15KHz et rapport cyclique ¼ sur PA4 en utilisant l'interruption OC4
4. Mesurer la période du signal sur l'entrée IC3 (interruption) le résultat sera rangée dans un variable globale diff. (On mesurera les fréquences min et max du signal mesurable)
5. Mesurer la période du signal sur l'entrée IC3 (interruption) le résultat sera affiché en us sur l'afficheur à cristaux liquides
6. Mesurer la fréquence du signal sur l'entrée IC3 (interruption) le résultat sera affiché en Hz sur l'afficheur à cristaux liquides
7. Réaliser un voltmètre affichant en volts la tension présente sur PE0 (sans interruptions)
8. Analyser la bibliothèque iosciit.c, puis créer un programme echo retournant sur la liaison série le caractère reçu. Utiliser pour émettre la fenêtre « raw » (celle ci peut être remplacée par l'accessoire « terminal » de windows
9. En utilisant sprinft réaliser un programme voltmètre visualisant la tension mesurée sur l'afficheur lcd et la fenêtre « raw »
10. en vous aidant de wreeprom, stocker la dernière mesure dans une « variable » en eeprom
11. Réaliser une fonction « suiveur » recopiant une entrée analogique sur une sortie, le tout cadencé par OC3
12. A l'aide de la documentation réaliser un générateur sinusoïdale utilisant le MAX512 dont la fréquence sera réglé par un potentiomètre (sur PEx)

Annexes :

Docs 68HC11 et CBOYF1

15. Annexes 68HC11

15.1. Vecteurs d'interruption

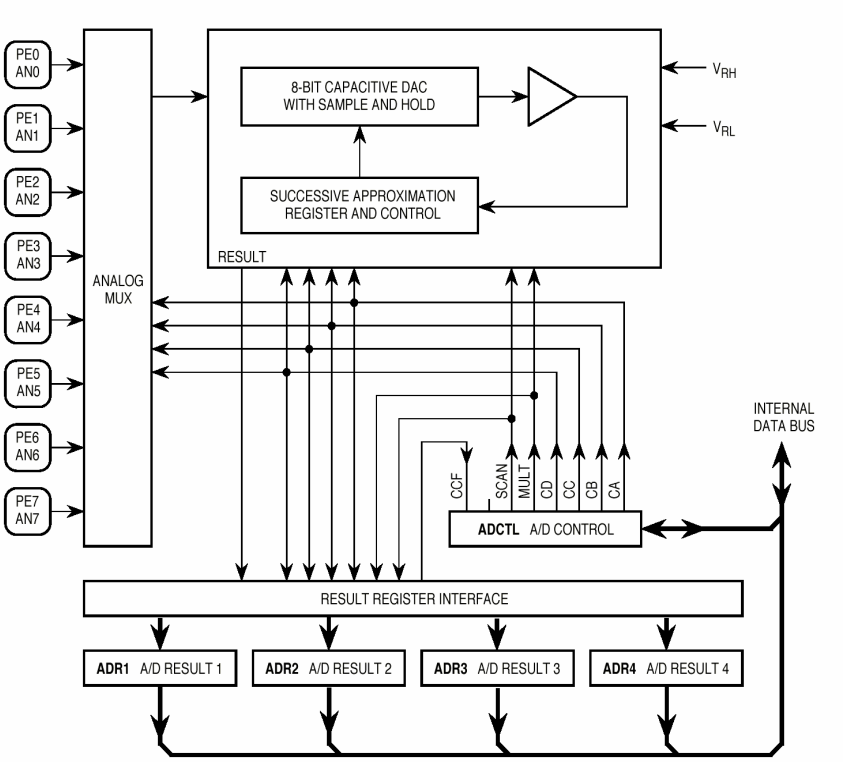
Addresses des vecteurs	Sources des interruptions	Masque du CCR	Masque local
FFC0,C1–FFD4,D5	Reserved	—	—
FFD6,D7	SCI Serial System	I	
•	✍ SCI Receive Data Register Full		RIE
•	✍ SCI Receiver Overrun		RIE
•	✍ SCI Transmit Data Register Empty		TIE
•	✍ SCI Transmit Complete		TCIE
•	✍ SCI Idle LineDetect		ILIE
FFD8, D9	SPI Serial Transfer Complete	I	SPIE
FFDA, DB	Pulse Accumulator Input Edge	I	PAII
FFDC, DD	Pulse Accumulator Overflow	I	PAOVI
FFDE, DF	Timer Overflow	I	TOI
FFE0, E1	Timer Input Capture 4 / Output Compare 5	I	I4/O5I
FFE2, E3	Timer Output Compare 4	I	OC4I
FFE4, E5	Timer Output Compare 3	I	OC3I
FFE6, E7	Timer Output Compare 2	I	OC2I
FFE8, E9	Timer Output Compare 1	I	OC1I
FFEA, EB	Timer Input Capture 3	I	IC3I
FFEC, ED	Timer Input Capture 2	I	IC2I
FFEE, EF	Timer Input Capture 1	I	IC1I
FFF0, F1	Real-Time Interrupt	I	RTII
FFF2, F3	IRQ (External Pin)	I	None
FFF4, F5	XIRQ Pin	X	None
FFF6, F7	Software Interrupt	None	None
FFF8, F9	Illegal Opcode Trap	None	None
FFFA, FB	COP Failure	None	NOCOP
FFFC, FD	Clock Monitor Fail	None	CME
FFFE, FF	RESET	None	None

15.2. Registres internes 68HCF1

Adresses	Bit 7	6	5	4	3	2	1	Bit 0	Registre	
$1000	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0	PORTA	
$1001	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	DDRA	
$1002	PG7	PG6	PG5	PG4	PG3	PG2	PG1	PG0	PORTG	
$1003	DDG7	DDG6	DDG5	DDG4	DDG3	DDG2	DDG1	DDG0	DDRG	
$1004	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0	PORTB	
$1005	PF7	PF6	PF5	PF4	PF3	PF2	PF1	PF0	PORTF	
$1006	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0	PORTC	
$1007	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	DDRC	
$1008	0	0	PD5	PD4	PD3	PD2	PD1	PD0	PORTD	
$1009	0	0	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	DDRD	
$100A	PE7	PE6	PE5	PE4	PE3	PE2	PE1	PE0	PORTE	
$100B	FOC1	FOC2	FOC3	FOC4	FOC5	0	0	0	CFORC	
$100C	OC1M7	OC1M6	OC1M5	OC1M4	OC1M3	0	0	0	OC1M	
$100D	OC1D7	OC1D6	OC1D5	OC1D4	OC1D3	0	0	0	OC1D	
$100E	15	14	13	12	11	10	9	8	TCNT	(High)
$100F	7	6	5	4	3	2	1	0	TCNT	(Low)
$1010	15	14	13	12	11	10	9	8	TIC1	(High)
$1011	7	6	5	4	3	2	1	0	TIC1	(Low)
$1012	15	14	13	12	11	10	9	8	TIC2	(High)
$1013	7	6	5	4	3	2	1	0	TIC2	(Low)
$1014	15	14	13	12	11	10	9	8	TIC3	(High)
$1015	7	6	5	4	3	2	1	0	TIC3	(Low)
$1016	15	14	13	12	11	10	9	8	TOC1	(High)
$1017	7	6	5	4	3	2	1	0	TOC1	(Low)
$1018	15	14	13	12	11	10	9	8	TOC2	(High)
$1019	7	6	5	4	3	2	1	0	TOC2	(Low)
$101A	15	14	13	12	11	10	9	8	TOC3	(High)
$101B	7	6	5	4	3	2	1	0	TOC3	(Low)
$101C	15	14	13	12	11	10	9	8	TOC4	(High)
$101D	7	6	5	4	3	2	1	0	TOC4	(Low)
$101E	15	14	13	12	11	10	9	8	TI4/O5	(High)
$101F	7	6	5	4	3	2	1	0	TI4/O5	(Low)
$1020	OM2	OL2	OM3	OL3	OM4	OL4	OM5	OL5	TCTL1	
$1021	EDG4B	EDG4A	EDG1B	EDG1A	EDG2B	EDG2A	EDG3B	EDG3A	TCTL2	
$1022	OC1I	OC2I	OC3I	OC4I	I4/O5I	IC1I	IC2I	IC3I	TMSK1	
$1023	OC1F	OC2F	OC3F	OC4F	I4/O5F	IC1F	IC2F	IC3F	TFLG1	
$1024	TOI	RTII	PAOVI	PAII	0	0	PR1	PR0	TMSK2	
$1025	TOF	RTIF	PAOVF	PAIF	0	0	0	0	TFLG2	
$1026	0	PAEN	PAMOD	PEDGE	0	I4/O5	RTR1	RTR0	PACTL	
$1027	7	6	5	4	3	2	1	0	PACNT	
$1028	SPIE	SPE	DWOM	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR	
$1029	SPIF	WCOL	0	MODF	0	0	0	0	SPSR	
$102A	7	6	5	4	3	2	1	0	SPDR	
$102B	TCLR	0	SCP1	SCP0	RCKB	SCR2	SCR1	SCR0	BAUD	
$102C	R8	T8	0	M	WAKE	0	0	0	SCCR1	
$102D	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK	SCCR2	
$102E	TDRE	TC	RDRF	IDLE	OR	NF	FE	0	SCSR	
$102F	7	6	5	4	3	2	1	0	SCDR	
$1030	CCF	0	SCAN	MULT	CD	CC	CB	CA	ADCTL	
$1031	7	6	5	4	3	2	1	0	ADR1	
$1032	7	6	5	4	3	2	1	0	ADR2	
$1033	7	6	5	4	3	2	1	0	ADR3	
$1034	7	6	5	4	3	2	1	0	ADR4	
$1035	0	0	0	PTCON	BPRT3	BPRT2	BPRT1	BPRT0	BPROT	
$1036	et	$1037	Reserved							
$1038	GWOM	CWOM	CLK4X	0	0	0	0	0	OPT2	
$1039	ADPU	CSEL	IRQE	DLY	CME	FCME	CR1	CR0	OPTION	
$103A	7	6	5	4	3	2	1	0	COPRST	
$103B	ODD	EVEN	0	BYTE	ROW	ERASE	EELAT	EEPGM	PPROG	
$103C	RBOO	SMOD	MDA	IRV	PSEL3	PSEL2	PSEL1	PSEL0	HPRIO	
$103D	RAM3	RAM2	RAM1	RAM0	REG3	REG2	REG1	REG0	INIT	
$103E	TILOP	0	OCCR	CBYP	DISR	FCM	FCOP	0	TEST1	
$103F	EE3`	EE2	EE1	EE0	1	NOCOP	1	EEON	CONFIG	
$1040	to	$105B	Reserved							
$105C	IO1SA	IO1SB	IO2SA	IO2SB	GSTHA	GSTHB	PSTHA	PSTHB	CSSTRH	
$105D	IO1EN	IO1PL	IO2EN	IO2PL	GCSPR	PCSEN	PSIZA	PSIZB	CSCTL	
$105E	GA15	GA14	GA13	GA12	GA11	GA10	0	0	CSGADR	
$105F	IO1AV	IO2AV	0	GNPOL	GAVLD	GSIZA	GSIZB	GSIZC	CSGSIZ	

15.3. Ports analogiques

Type : 8 entrées analogiques multiplexées et un convertisseur analogique numérique 8 bits à approximations successives, à capacités commutés. Précision ? ½ LSB. Durée d'une conversion 32 cycles soit 128 cycles pour quatre conversions. Le port E est un port analogique si la fonction CAN est activée, dans le cas contraire c'est un port numérique en entrée.



- Le BIT ADPU du registre OPTION doit être mis à 1 pour activer la fonction CAN
- Le bit CSEL doit être mis à 1 si la fréquence de l'horloge interne (E) est supérieure à 750 KHz
- La conversion débute après une écriture dans le registre **ADCTL,** le micro contrôleur effectue alors quatre conversions successives et place les résultats dans les registres **ADR4-ADR1.**

ADCTL (A/D Control)

D7	D6	D5	D4	D3	D2	D1	D0
CCF	x	SCAN	MULT	CD	CC	CB	CA

Une conversion commence après l'écriture dans ADCTL et la mise à 0 de CCF (SOC)
Le bit CCF est mis à 1 après 4 conversions… (EOC)

SCAN = 1 : le CAN effectue une seule fois les quatre conversions
SCAN = 0 : le CAN effectue en permanence fois les quatre conversions

MULT=0 : Le CAN effectue quatre conversions sur le canal sélectionné par CD, CC, CB, CA et range les résultats dans ADR1, ADR2, ADR3, ADR4 *(voir tableau registres)*
MULT=1 : Le CAN effectue quatre conversions sur quatre canaux différents CB et CA sont ignorés

MULT=0

CD : CC : CB : CA	CANAL
0000	AN0 (PE0)
0001	AN1 (PE1)
0010	AN2 (PE2)
0011	AN3 (PE3)
0100	AN4 (PE4)
0101	AN5 (PE5)
0110	AN6 (PE6)
0111	AN7 (PE7)
10xx	Interdit
1100	VRH
1101	VRL
1110	VRH/2
1111	interdit

MULT=1

CD : CC : CB : CA	CANAL	Destination
00xx	AN0 (PE0)	ADR1
00xx	AN1 (PE1)	ADR2
00xx	AN2 (PE2)	ADR3
00xx	AN3 (PE3)	ADR4
01xx	AN4 (PE4)	ADR1
01xx	AN5 (PE5)	ADR2
01xx	AN6 (PE6)	ADR3
01xx	AN7 (PE7)	ADR4
10xx	Interdit	
11xx		ADR1
11xx		ADR2
11xx		ADR3
11xx	interdit	

15.4. Générateur d'interruption périodique

Real Time interrput : RTI

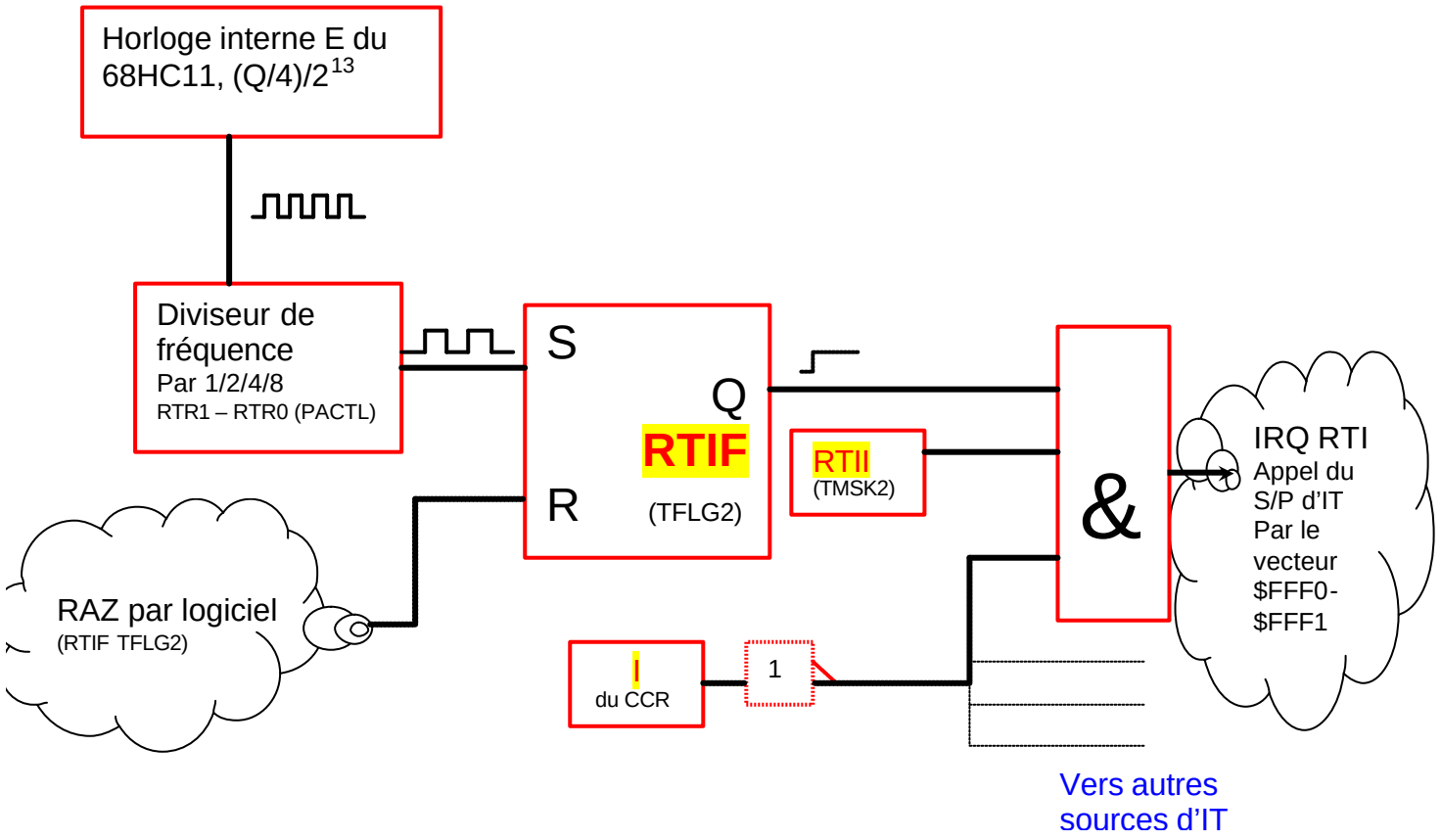
Cette fonction permet de générer une impulsion périodique. (Horloge temps réel, mesures automatiques et périodiques, gestion d'affichage multiplexé etc...)

Le bit 6 (RTII) de TMSK2 permet de valider l'interruption de RTI

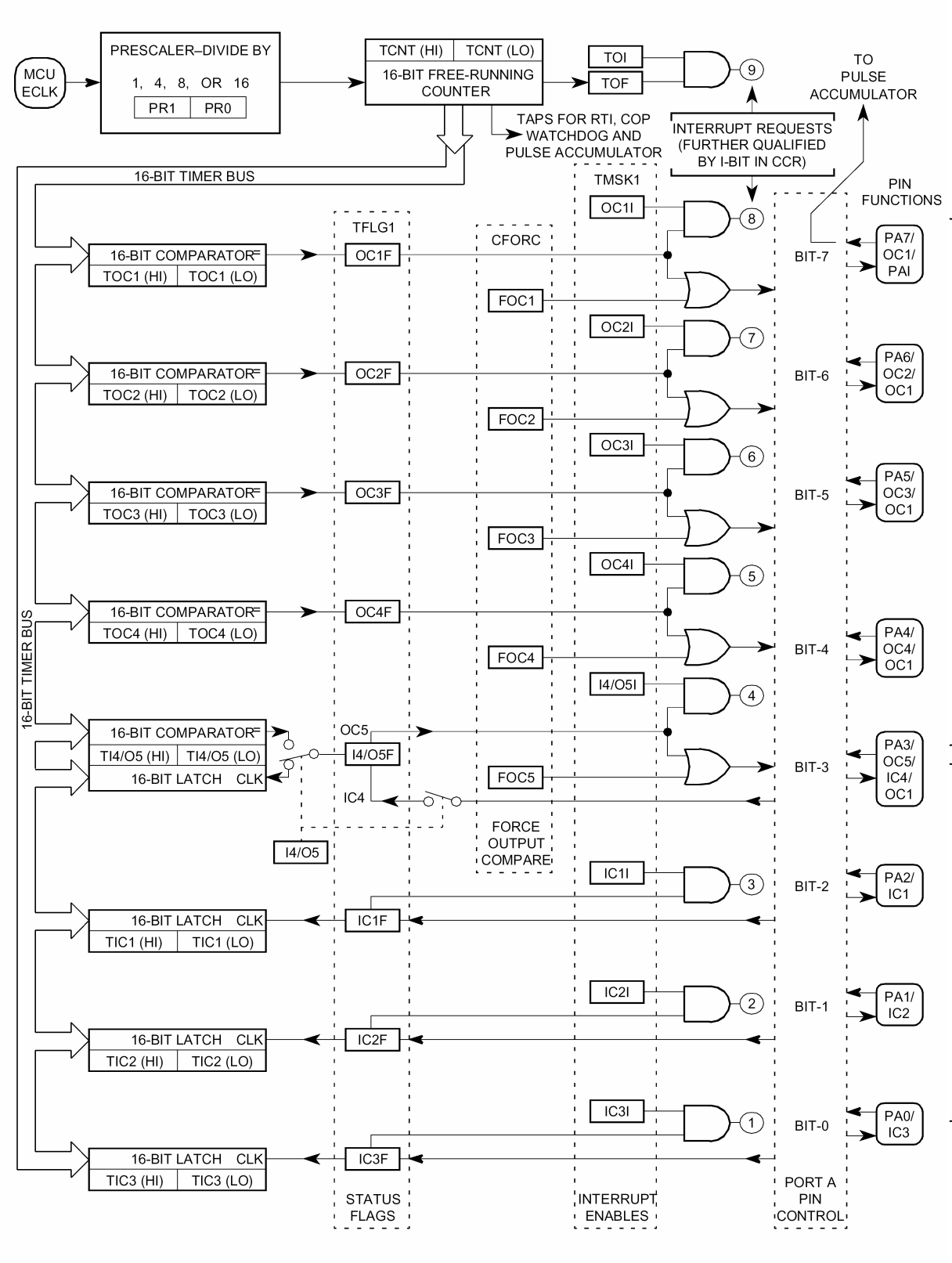
Le bit 6 (RTIF) de TMSG2 est mis à 1 par la fonction RTI à la fin de chaque période (fonctionnement sans interruption)

Les bits 0 et 1 (RTR0 et RTR1) de PACTL permettent de programmer la période des interruptions temps réel en fonction du quartz

RTR1	RTR0	E/2^{13} div	F(RTI) si Q=8MHz	F(RTI) si Q=12MHz	F(RTI) si Q=16MHz
0	0	1	4.096ms	2.7307ms	2.048ms
0	1	2	8.192ms	5.4613ms	4.096ms
1	0	4	16.386ms	10.9227ms	8.192ms
1	1	8	32.768ms	21.8453ms	16.386ms



15.5. TIMERS : Mesures et production de temps



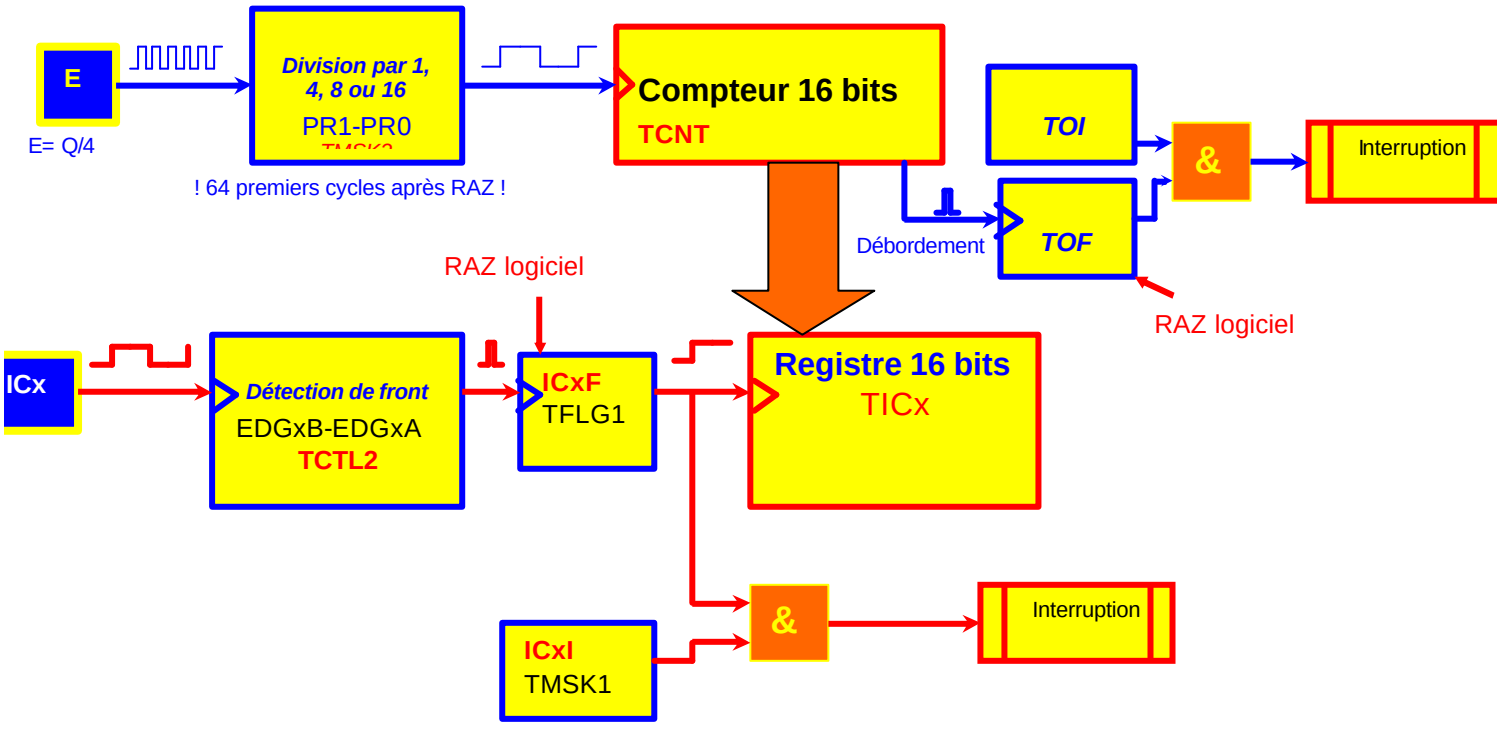
15.6. Comparaison en entrée :

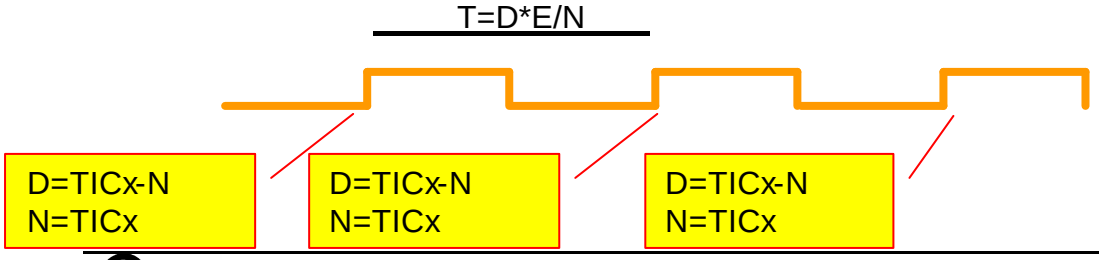
Un front sur une entrée transfert la valeur de TCNT dans le TIC correspondant. La détection peut être programmée (Registre TCL2) pour un front montant, descendant ou l'un ou l'autre. Un drapeau dans TFLG1 est positionné lors de la détection du front. Il provoque une interruption si le bit correspondant de TMSK1 est actif (=1).

- *La différence de comptage de TCNT entre deux fronts identiques permet de mesurer une période.*
- *La différence de comptage de TCNT entre deux fronts différents permet de mesurer une durée.*

EDGxB	EDBxA	Configuration
0	0	Pas de déclenchement
0	1	Déclenchement sur front montant
1	0	Déclenchement sur front descendant
1	1	Déclenchement sur n'importe quel front

TFLG2 : Registre contenant les drapeaux indiquant un évenement en entrée
TMSK2 : Registre contenant les masques d'interruption
L'interruption TO (Timer Overflow permet de compter les débordements de TCNT et donc de mesurer ou de produire des temps très longs (en fait il n'y a pas de limite), en revanche la limite inférieur de mesure ou de production d'un temps dépend de la durée de traitement de l'interruption (au moins 20 uS sur E9)
TOF : détection de passa ge de TCNT de $FFFF à $0000
TOI validation de l'interruption de dépassement de capacité de TCNT





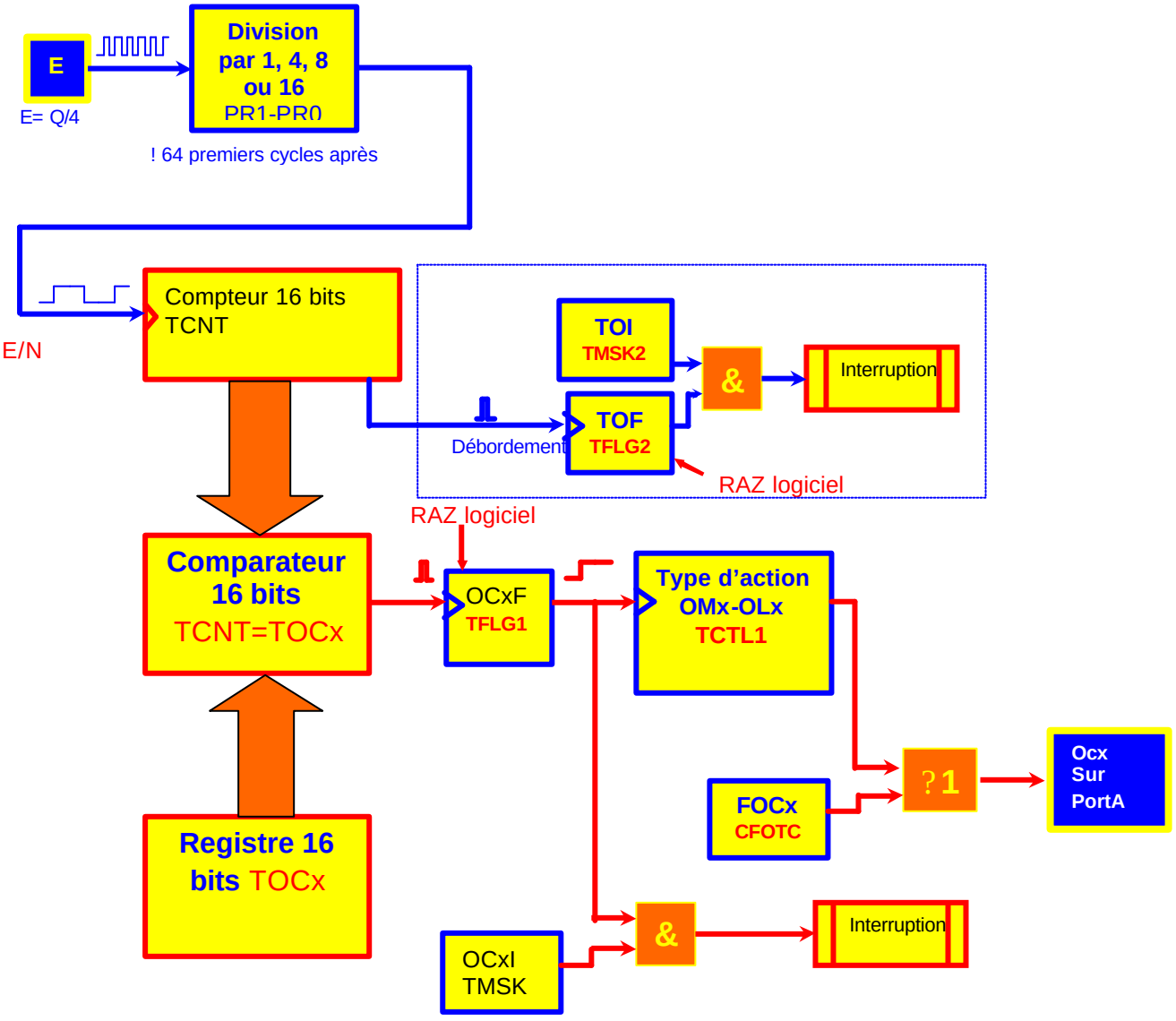
15.7. Comparaison en sortie :

Lorsqu'il y a égalité entre TCNT et TOC le bit OCF de TFLG1 est mis à 0 . Une interruption est générée si le bit correspondant de OC de TMSK1 est à 1. La sortie peut être bloquée par le bit FOC de CFROC correspondant.
OC1M permet de connecter ou de débrancher les broches du port A du TIMER
OC1D contient les valeurs à placer sur les ports PA3 à PA7 lors d'une activation du TIMER OC1

TCTL1 : Permet de définir une action périodique pour les bits 3 à 6 du port A

OMx	OLx	Configuration
0	0	PAx inchangé
0	1	Bascule PAx
1	0	PAx=0
1	1	PAx=1

CFORC : permet d'inhiber l'action sur les sorties



Pour produire un signal carrée, il est nécessaire après chaque interruption de recharger TOCx avec D

D= durée **N=rapport de PR1-¨PR0, E est exprimée en secondes**
ExN

CORRECTION DES EXERCICES

Exercice 1

```
/* FLASHIT, clignotement de la led verte , IT avec fonction RTI*/
/*CD 01/2000*/
#include <hc11.h>

#pragma interrupt_handler RtiInt
void RtiInt(void)
{static int cptit = 0;/*la variable est locale mais concervée entre deux IT*/
       if (++cptit >= 244)  /* pour 0,5S 16MHz Mhz */
              {
              cptit = 0;
              PORTG ^= 0x01;              /* basculer led */
              }
       TFLG2 |= 0x40;        /*efface drapeau IT RTI*/
}

#pragma abs_address:0xFFF0
void (*rtiint_vector)(void) = RtiInt ;
#pragma end_abs_address

void main(void)
{      DDRG  = 0x01;         /*PG0 en sortie*/
       PACTL &= 0xFC;        /* selectioner la vitesse */
       TMSK2 |= 0x40;        /* declencher le timer */
       asm(" cli ");         /* autoriser les interruptions */
       while(1);
}
```

Exercice 2

```
/*\cc11\exoccbf1\carree.c
/*Générer un signal carré de 2.15 KHz sur PA3  */
/* CD 01/2000*/
#include <hc11.h>       /*déclaration des bibliotheques*/
#define t232us 930    /*pour 2.15KHz sur CBOYF1 16MHz*/

#pragma interrupt_handler oc5Int   /*déclaration de l'IT d'oc5*/
void oc5Int(void)                  /*routine d'interruption*/
{
       TOC5 +=t232us;   /*durée de l'interruption = 232us*/
       TFLG1 |= 0x8;    /*on met oc5 au niveau haut (drapeau)*/
}

#pragma abs_address:0xFFE0      /*routine d'IT , vecteur RESET*/
void (*oc5int_vector)(void) = oc5Int ; /*adresse de oc5Int en FFEO*/
#pragma end_abs_address

void main(void)                          /*programme principal*/
{      DDRA  |=0x08;  /*PA3 en sortie*/
       PACTL |=0x04;   /*Timer oc5 activé (bit i4/o5 */
       TMSK1 |=0x08;   /*IT de oc5 activée */
       TCTL1 |=1;      /*action périodique définie sur le port A*/
       TCTL1 &=0xfd;               /*OMx="0" et OLx="1"*/
       asm(" cli ");   /*autoriser les interruptions*/
       while(1);
}
```

Exercice 3

```
/*Générer un signal rectangulaire de 1KHz KHz sur PA3  CBOYF1*/
/*rapport cyclique 1/4*/
/* CD 01/2000*/
#include <hc11.h>             /*déclaration des bibliotheques*/
#define th 1000     /*th=250uS*/
#define tl 3000       /*tl=750uS*/


#pragma interrupt_handler oc5Int   /*déclaration de l'IT d'oc5*/
void oc5Int(void)                        /*routine d'interruption*/
```

```
{
       if (PORTA & 0x08)  TOC5+=th;
                      else TOC5+=tl;
       TFLG1 |= 0x8;    /*on met oc5 au niveau haut (drapeau)*/
}

#pragma abs_address:0xFFE0      /*routine d'IT , vecteur RESET*/
void (*oc5int_vector)(void) = oc5Int ; /*adresse de oc5Int en FFEO*/
#pragma end_abs_address


void main(void)                          /*programme principal*/
{      DDRA  |=0x08; /*PA3 en sortie*/
       TMSK1 |=0x08; /*IT de oc5 activée et PA3 bascule à "0"*/
       TCTL1 |=1;    /*action périodique définie sur le port A*/
       TCTL1 &=0xfd;                     /*OMx="0" et OLx="1"*/
       asm(" cli ");           /*autoriser les interruptions*/
       while(1);
}
```

Exercice 4

```
/* mesure de periode sur IC3 sur CBOYF1 */
/*CD 01/2000*/
/* pour lire le comptage faire d diff dans le debugeur*/

#include <hc11.h>

unsigned int diff;    /* le resultat se trouve dans diff*/
/*********************************************************************/
#pragma interrupt_handler ic3    /* programme d'interruption       */
void ic3(void)
{
unsigned int ancienic3;             /* variable locale*/
       diff=TIC3-ancienic3;    /* donne le tps écoulé entre deux  */
                           /* fronts montants                   */
       ancienic3=TIC3;              /* valeur du 2iéme front montant   */
                           /* devient la valeur du front      */
                             /* à soustraire au suivant         */
       TFLG1|=1;                /* éffface le drapeau ic3f, ce qui autorise une nouvelle it*/
}

#pragma abs_address:0xffea       /* vecteur d'interrup. IC3I */
void(*IC3_vector)(void)=ic3;     /* pointe l'adresse de la fonction ic3 */
#pragma end_abs_address

/*********************************************************************/
void main(void)                          /* programme principal */

{      TCTL2=1;                  /* detecte fronts montants */
       TMSK1|=1;                 /* autorise it IC3 */
       asm ("cli");
       while(1);
}
```

Exercice 5

Reprendre l'exercice 4 avec dans la boucle while :
printf(? T= %f uS ? ,(float)(diff)*0.25); // pour un quartz 16MHz

Exercice 6

```
/*frequencemetre sur CBOYF1 16MHz*/
/*CD 01/2000*/

#include <hc11.h>
#include <lcdclav.h>
#include <stdio.h>

#pragma interrupt_handler ic3
void ic3(void)
{
static unsigned int exic3;
```

```
        gotox(4);
        printf("%fHz    ",4e6/((float)(TIC3-exic3)));
        exic3=TIC3;
        TFLG1|=1;
}

#pragma abs_address:0xffea
void(*IC3_vector)(void)=ic3;
#pragma end_abs_address

void main(void)
{
        TCTL2=1;        /*it ic3 sur front montant*/
        TMSK1|=1;       /*it ic3 autorisee*/
        lcdinit();
        gotox(1);
        printf("F= ");
        asm(" cli ");
        while(1);
}
```

Exercice 7

```
// \cc11\exoccbf1\lcdvolt.c
/*voltmetre sur CBOYF1 16MHz*/
/*CD 01/2000*/

#include <hc11.h>
#include <lycee.h>
#include <stdio.h>

unsigned char mesvolt(void)
{
        ADCTL=0x00;     /* le 68HC11 effectue 4 conversions successives*/
                        /*sur le canal AN0 (PE0)*/
        while ((ADCTL & 0x80)==0); /*attend fin de la conversion*/
        return (ADR1);
}

void main(void)         /*programme principal*/
{
        OPTION |=0x80; /*active CAN interne du HC11*/
        lcdinit(2);
        gotox(1);
        asm(" cli "); /* pour valider le debugeur*/
        printf("U= ");
        for(;;)
        {
                gotox(4);
                printf("%fv     ",(float)(mesvolt())*1.96078e-2);
        }
}
```

Exercice 8

(sans printf sur afficheur)

```
/*      autorise les IT */
#include "\cc11\exoccbf1\iosciit.c"

#define autoriseIT asm(" cli\n")
char message[20];       /* pour la démo le message max vaut 20 caractères*/

/*      Programme principal
qui ne fait pas grand chose*/

void main(void)
        {
        initsci();
                                        /* le pseudo vecteur sur CBOY*/
        autoriseIT;                     /* autorise les IT */
        for (;;) {
/*              putsci(getsci()+1);     /*retourne le caractère suivant pour le test*/
                putstsci(getstsci(message));  /* ou retourne la chaine envoyée (qui finit par
(*)*/
                                        /* c'est au choix */
```

```
            }
        }
```

Exercice 11

```
// \cc11\exoccbf1\lcdvolt.c
/*voltmetre sur CBOYF1 16MHz*/
/*CD 01/2000*/

#include <hc11.h>
#include <lycee.h>
#include <stdio.h>

char message[20];

void main(void)       /*programme principal*/
{
        OPTION |=0x80; /*active CAN interne du HC11*/
        lcdinit(2);
        initsci();
        gotox(1);
        asm(" cli "); /* pour valider le debugeur*/
        printf("U= ");
        for(;;)
        {
                sprintf(message,"%fv     ",(float)(analogin(0))*1.96078e-2);
                gotox(4);
                putstsci(message);
                printf(message);
        }
}
```

Exercice 12

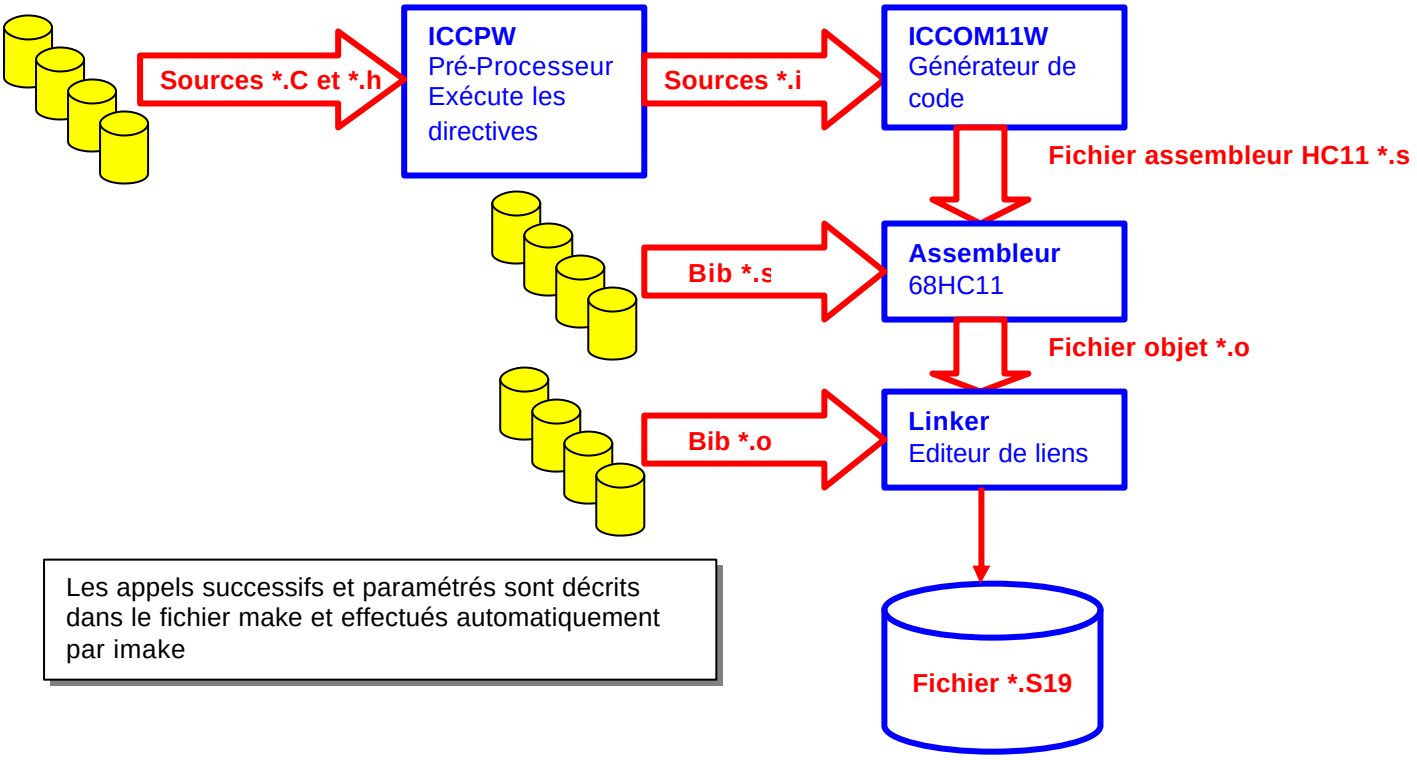
```
// génération d'une sinus par calcul mathématique (sin(x)
// !!!!!!!!!!!!!!! SGDG
// Testé sur simulateur CBOYF1

#include <hc11.h>
#include <math.h>

#define PORTM  *(unsigned char *)(_IO_BASE + 0x62)
#define PORTN  *(unsigned char *)(_IO_BASE + 0x63)

/************************** tempo ***************************************/
void wait(int cnt)
{
        for (;cnt>0; cnt--);

}
/************************** analogin ************************************/

unsigned char analogin(unsigned char ch)
{       ADCTL = ch;    // start conversion
        while ((ADCTL & 0x80)==0) ;

        return ADR1;
}

/************************** analogout ***********************************/
// Max512
// M0 SCLK
// M1 SDIN
// M4 CS

void max512(unsigned char v)
{       unsigned char cnt;
        for (cnt=0; cnt<8; cnt++){
                if (v&0x80) {
                        PORTM = 0x22;  // SDIN = 1
                        PORTM = 0x23;  // SCLK = 1
                } else {
                        PORTM = 0x20;  // SDIN = 0
                        PORTM = 0x21;  // SCLK = 1
                }
```

```
                PORTM = 0x20;          // SCLK = 0
                v <<= 1;
        }
}

void analogout(unsigned char ch, unsigned char val)
{       PORTM = 0x20;                  // CS = 0
        max512(ch);
        max512(val);
        PORTM = 0x30;                  // CS = 1
}

/************************* programme principal ***********************************/

void main(void)
{
unsigned char tempo,sortie;
float d=0;
        OPTION |= 0x80;                       // active le CAN
        asm (" cli");
        for(;;)
                {
                tempo=analogin(0);    //lecture de PE0
                wait(tempo);
                sortie=(unsigned char)(255*sin(d));
                analogout(0,sortie);  // calculer en radians
                d=d+24e-3;            // 256 valeurs par tour
                if (d>6.2831) d=0;
                }
}
```

Exercice 13

```
/*Filtre Numérique réjecteur 50Hz, modèle Butterworth
/* OC3 le cadence  */
/* CD 12/2000 avec bib lcdclav lycée*/
#include <hc11.h>       /*déclaration des bibliotheques*/
#include <lcdclav.h>
#define tech 4000     /* échantillonage pour 200Hz (5mS) sur CBOYF1 16MHz*/

const float a=0.796,c=0.5922;
float x0,x1,x2,y0,y1,y2;

#pragma interrupt_handler oc2Int   /*déclaration de l'IT d'oc5*/
void oc2Int(void)                  /*routine d'interruption*/
{       PORTA=1;
//      y0=(float)(analogin(7));
//      analogout(1,(int)(y0));

        analogout(1,analogin(7));

        TOC2 +=tech;
        TFLG1 |= 0x40;
        PORTA=0;
}

#pragma abs_address:0xFFE6      /*routine d'IT , vecteur RESET*/
void (*oc2int_vector)(void) = oc2Int ; /*adresse de oc5Int en FFEO*/
#pragma end_abs_address

void main(void)                         /*programme principal*/
{
        OPTION|=0x80;  //active CAN
        TMSK1 |=0x40;   /*IT de oc2 activée */
        TCTL1 &=0x3f;   /*aucune action définie sur le port A*/
        DDRA=1;
        asm(" cli ");   /*autoriser les interruptions*/
        while(1);
}
```

TRANSPARENTS



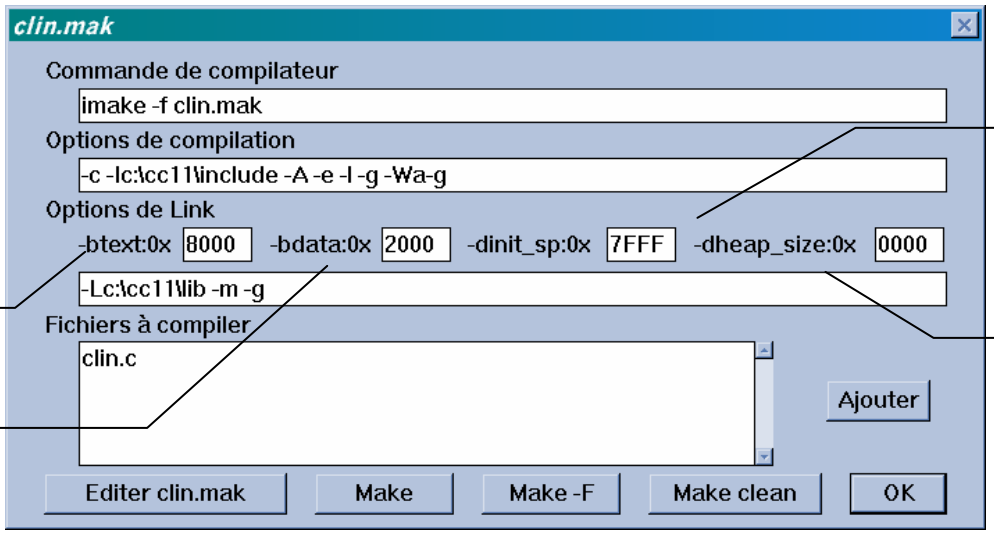
Les appels successifs et paramétrés sont décrits dans le fichier make et effectués automatiquement par imake

FLASH.C

```
/*      Clignoter une LED sur
Controlboy F1 */
#include <hc11.h>
void wait(int cnt)
{
        for (;cnt>0; cnt--);
}

void main(void)
{int x;
        DDRG  = 0x01;
        /* PG0 = sortie */
        for(;;)
        {
                PORTG ^= 0x01;
        /* basculer led */
                if (PORTG&0x2) x= 20000;
                else x=5000;
                wait(x);
        }
}
```

FLASH.LIS (aperçu partiel)

```
                ; void main(void)
                ; {int x;
001F                .dbline 10
                ;     DDRG  = 0x01;         /* PG0 = sortie */
001F C601             ldab #1
0021 F71003           stab 0x1003
0024                .dbline 11
                ;     for(;;)
0024              L8:
0024                .dbline 12{
                ;     {
0024                .dbline 13
                ;         PORTG ^= 0x01;        /* basculer led */
                  ; vol
0024 F61002           ldab 0x1002
0027 C801             eorb #1
0029 F71002           stab 0x1002
002C                .dbline 14
                ;         if (PORTG&0x2) x= 20000;
002C 18CE1002         ldy #0x1002
0030 181F000207       brclr 0,y,#2,L12
0035                .dbline 14
0035 CC4E20           ldd #20000
0038 ED02             std 2,x
003A 2005             bra L13
003C              L12:
003C                .dbline 15
                ;         else x=5000;
003C CC1388           ldd #5000
003F ED02             std 2,x
0041              L13:
0041                .dbline 16
                ;         wait(x);
0041 EC02             ldd 2,x
0043 BD0000           jsr _wait
0046                .dbline 17}
                ;     }
0046                .dbline 11
0046                .dbline 11
0046 20DC             bra L8
```



Initialisation du pointeur de pile S

Début de l'EEPROM

Taille du tas en cas d'utilisation de MALLOC etc..

Début de la RAM

Fichier tstfloat.c

```
// Programme sur Controlboy F1: printf virgule flottante sur LCD
#include "hc11.h"
#include <stdio.h>
#include <math.h>
#include <lycee.h>
double e;
void main(void)
{       int i;
        lcdinit(2);
        gotox(1);
        puts("sqrt(");
        for (i=2;;)     {       e = sqrt(i);
                                if (i>=0)
                                {
                                        gotox(6);
                                        printf("%d)=%f", i, e);
                                }
                                i=keyget(0)-'0';
                        }
}
```

```
/* ce programme traite l'IT RTI et incrémente les secondes et les minutes*/
#include <hc11.h>
int second;
int minute;

#pragma interrupt_handler RtiInt
void RtiInt(void)
{       static int tictac = 0;
        if (++tictac >= 1953){                  /* 1s avec Q=16MHz   */
                                tictac = 0;
                                if (++second >= 60) {       second = 0;
                                                            minute++;
                                        }
                                }
        TFLG2 |= 0x40;
}
#pragma abs_address:0xFFF0
void (*rtiint_vector)(void) = RtiInt ;
#pragma end_abs_address
void main(void)
{       PACTL &= 0xFC;              /* selectioner la
        TMSK2 |= 0x40;              /* declencher le timer */
        asm(" cli ");           /* autoriser les interruptions */
        while(1) ; /* le compilateur respectant la norme ANSI refuse les boucles
infinies du type for( ;;) */
}
```

Indique au compilateur de terminer cette fonction par RTI et non RTS

Positionnement du vecteur d'interruption, le pragma force le compilateur à placer le code à une adresse imposée

Déclaration d'un pointeur sur une fonction dont l'adresse est ici RTIiInt