

# **GUIDE DE REFERENCE DES FONCTIONS DYNAMIC C 5.x**

**Révision 3 - version Française  
(Format *PDF*)**

# TABLE DES MATIERES

<b>A propos de ce manuel</b>	<b>AP</b>
<b>Librairies support générales</b>	<b>1-1</b>
Initialisation globale	1-1
Fonctions BIOS	1-1
MATH.LIB	1-4
STDIO.LIB	1-7
STRING.LIB	1-10
SYS.LIB	1-13
XMEM.LIB	1-15
<b>Librairies multitâches</b>	<b>2-1</b>
RTK.LIB	2-1
SRTK.LIB	2-2
VDRIVER.LIB	2-3
<b>Librairies contrôleurs</b>	<b>3-1</b>
BL1000.LIB	3-1
BL11xx.LIB	3-1
BL14_15.LIB	3-2
BL16xx.LIB	3-7
PK21xx.LIB	3-7
PK22xx.LIB	3-8
CM71_72.LIB	3-9
<b>Librairies AASC</b>	<b>4-1</b>
AASC.LIB	4-1
Fonctions XModem de AASC.LIB	4-4
<b>Autres librairies</b>	<b>5-1</b>
5KEY.LIB	5-1
5KEYEXTD.LIB	5-5
CPLC.LIB	5-6
DRIVERS.LIB	5-7
DMA.LIB	5-12
FK.LIB	5-14
XP88xx.LIB	5-15
IOEXPAND.LIB	5-18
KDM.LIB	5-21
LCD2L.LIB	5-26
PBUS_LG.LIB	5-28
PBUS_TG.LIB	5-30

**Annexe A: Librairies Dynamic C**

**A-1**

**Annexe B: Utilisation des librairies AASC**

**B-1**

Description des librairies AASC	B-2
Fonctionnement des librairies AASC	B-3
Read	B-3
Write	B-3
Peek	B-4
Status et erreurs	B-4
Utilisation des librairies	B-4
Programme d'exemple	B-4
Transfert XModem	B-5
Utilisation des librairies	B-5
Programme d'exemple	B-5

## **A PROPOS DE CE MANUEL**

Les clients Z-World développent des programmes pour leurs contrôleurs en utilisant le système de développement Dynamic C à partir d'un PC. Le contrôleur est connecté sur un port COM du PC, classiquement COM2 qui fonctionne par défaut à 19200 bps.

La version standard de Dynamic C est recommandée pour des programmes jusqu'à 80K de code source, avec un accès limité à la mémoire étendue. La version Deluxe supporte des programmes jusqu'à 512K en ROM (code et constantes) et 512K en RAM (data variables), avec un accès complet à la mémoire étendue.

### **Les trois manuels**

Dynamic C est documenté par trois manuels de référence:

- Dynamic C Technical Reference
- Dynamic C Application Frameworks
- Dynamic C Function Reference (celui-ci).

Le manuel Technical Reference décrit comment utiliser le système de développement Dynamic C pour écrire des programmes pour les contrôleurs Z-World.

Le manuel Application Frameworks traite différents sujets en profondeur. Parmi les sujets traités on trouve le noyau temps réel Z-World, les Costatements, les fonctions de chaînage et la communication série.

Ce manuel contient les descriptions de toutes les bibliothèques Dynamic C ainsi que la description de toutes les fonctions incluses dans ces bibliothèques.

NOTE: Lire également les éventuelles notes de mise à jour pour obtenir certaines informations de dernière minute.

### **Avertissements**

Il est recommandé à l'utilisateur de posséder des connaissances de base dans les domaines suivants:

- Manipulation de logiciels et édition de fichiers sous Windows / PC.
- Notions sur le langage C. Le Dynamic C n'est pas identique aux C standards.
- Connaissances de base du Z80/Z180 et de l'assembleur associé. Un ouvrage en trois volumes "La Bible Zilog" est disponible chez votre revendeur.

### **Sigles utilisés dans ce manuel**

<b>SIGLE</b>	<b>SIGNIFICATION</b>
EPROM	Erasable Programmable Read-Only Memory
EEPROM	Electrically Erasable Programmable Read-Only Memory
NMI	Nonmaskable Interrupt (Interruption non masquable)
PIO	Parallel Input/Output circuit (Entrées/Sorties programmables individuellement)
PRT	Programmable Reload Timer (Timer programmable)
RAM	Random Access Memory
RTC	Real Time Clock (Horloge programmable)
SIB	Serial Interface Board (Module interface série)
SRAM	Static Random Access Memory (Mémoire sauvegardée)
UART	Universal Asynchronous Receiver Transmitter (Circuit de transmission série asynchrone)

## Conventions

Le tableau ci-dessous liste et définit les conventions typographiques qui sont utilisées dans ce manuel.

EXEMPLE	DESCRIPTION
<b>While</b>	Une police Times (gras) indique un programme, un morceau de programme, une phrase ou un mot clé Dynamic C.
// IN-01...	Les commentaires du programme sont imprimés en police Times.
<i>Italics</i>	Indique que quelque chose doit être tapé à la place des mots en italique. Par exemple, taper un nom de fichier à la place de Filename en italique.
<b>Edit</b>	Une police Sans Arial (gras) indique un menu ou un choix dans un menu.
...	Les points de suspension indiquent (1) Texte programme omis par soucis de brièveté (2) Le texte programme qui précède doit être répété indéfiniment.
[ ]	Indique une directive optionnelle dans une définition de fonction C ou un segment de programme.
< >	Ce symbole comporte occasionnellement des classes de termes.
a   b   c	Une barre verticale indique qu'un choix doit être fait parmi les options proposées.

## Abbréviations de programmation

Ce manuel utilise des abbréviations de programmation.

- **uint** signifie **unsigned integer** (entier non signé)
- **ulong** signifie **unsigned long** (long non signé)

Ces abbréviations ne sont pas des mots clés du C standard et ne fonctionneront pas tant qu'elles n'auront pas été au préalable déclarées avec **typedef** ou **#define**, comme dans les exemples ci-dessous:

```
typedef unsigned int uint
ou
#define ulong unsigned long
```

## Autres abbréviations

- (FD) Factory Default - Configuration d'usine.  
(HV) High Voltage - Haute tension.

# 1 - LIBRAIRIES SUPPORT GENERALES

Les librairies décrites au chapitre 1 comportent des fonctions spécifiques aux contrôleurs Z-World ainsi que des fonctions mathématiques et des chaînes C standard.

## Initialisation globale

L'initialisation globale est un sujet important, difficile à classer. Il est décrit ici. Votre programme peut initialiser des variables et procéder à des options d'initialisation (ou actions) de différentes complexité si vous suivez ce qui suit:

1. Incorporez des segments `_GLOBAL_INIT` dans vos fonctions:

```
void init_ios () ;
int my_func( void* thing ) {
    int table[10] , j ;
    float x , y ;
    ...
    segchain _GLOBAL_INIT {
        for ( j=0 ; j<10 ; j++ ) { table [ j ] = 10 - j ; }
        x = y = 0.781 ;
        init-ios ( ) ;
    }
    ...
}
```

2. Faites un appel à la fonction chaîne `_GLOBAL_INIT` au début du programme principal.

Quand votre programme démarre (après un Reset ou un incident), l'appel à `_GLOBAL_INIT` effectue l'initialisation pour tous les `_GLOBAL_INIT` du programme (librairies comprises). Le nom `_GLOBAL_INIT` n'est pas le nom d'une fonction en librairie, il y a toutefois une fonction `GLOBAL-INIT` dans `VDRIVER.LIB`. Si vous appelez `VdInit`, par exemple en invoquant le Virtual Driver, `VdInit` effectue une initialisation globale pour vous. Vous n'avez pas besoin de le faire vous-même. La fonction `uplc_init` lance aussi un `_GLOBAL_INIT`.

## Fonctions BIOS

Ces fonctions résident en BIOS. Le code source est mis à votre disposition. Pour outrepasser les fonctions BIOS, vous pouvez utiliser

```
#kill fonctionname
```

au début de votre programme et redéfinir la fonction.

- `uint inport ( uint port )`

Lit une valeur sur le port E/S spécifié. Cela peut être un registre interne du Z180, ou un accès à des composants externes. Se référer au manuel de référence du contrôleur pour obtenir la liste des ports d'E/S.

La fonction renvoie la valeur lue sur l'octet de poids faible et zéro sur l'octet de poids fort.

- **void outport ( uint port , uint value )**

Ecrit une valeur sur un port d'E/S. Cela peut être un registre interne du Z180, ou un accès à des composants externes. Se référer au manuel de référence du contrôleur pour obtenir la liste des ports d'E/S.

- **int ee\_rd ( int address )**

Lit une valeur à une adresse spécifiée sur l'EEPROM. La fonction renvoie la valeur lue dans l'EEPROM (0-255). Elle renvoie une valeur négative en cas d'échec de lecture sur l'EEPROM.

- **int ee\_wr ( int address )**

Ecrit une valeur à une adresse spécifiée sur l'EEPROM. La fonction renvoie 0 si l'écriture se passe correctement. Elle renvoie une valeur négative en cas d'échec.

- **void di ( )**

Invalide les interruptions. Utiliser **DI** pour une meilleure efficacité.

- **void DI ( )**

Invalide les interruptions. Dynamic C dispose de cet appel en ligne.

- **void ei ( )**

Valide les interruptions. Utiliser **EI** pour une meilleure efficacité.

- **void EI ( )**

Valide les interruptions. Dynamic C dispose de cet appel en ligne.

- **int iff ( )**

Renvoie l'état du masque d'interruption du Z180. Renvoie 0 en cas d'invalidité des interruptions, dans les autres cas les interruptions sont validées.

- **uint bit ( void\* address, uint bit )**

Lit la valeur du **bit** spécifié à l'adresse mémoire. Le **bit** peut être compris entre 0 et 31. Utiliser **BIT** (voir plus haut) pour disposer en ligne de cet appel, ce qui est équivalent à l'expression suivante:

$$(* (\text{long}^*) \text{address} \ll \text{bit}) \& 1$$

La fonction retourne 1 si le bit spécifié est mit; 0 s'il est effacé.

- **uint BIT ( void\* address, uint bit )**

Lit la valeur du **bit** spécifié à l'adresse mémoire. Le **bit** peut être compris entre 0 et 31. Dynamic C pourrait effectuer cet appel en ligne, ce qui est équivalent à l'expression suivante:

$$(* (\text{long}^*) \text{address} \ll \text{bit}) \& 1$$

La fonction retourne 1 si le bit spécifié est mit; 0 s'il est effacé.

- **void set ( void \*address, uint bit )**

Met le **bit** spécifié de l'adresse mémoire à 1. Ce **bit** peut être compris entre 0 et 31. Utiliser **SET** (voir plus haut) pour disposer de l'appel en ligne, ce qui est équivalent à l'expression suivante:

$$* (\text{long}^*) \text{address} \mid = 1 \ll \text{bit}$$

- **void SET ( void \*address, uint bit )**

Met le **bit** spécifié de l'adresse mémoire à 1. Ce **bit** peut être compris entre 0 et 31. Dynamic C pourrait effectuer cet appel en ligne, ce qui est équivalent à l'expression suivante:

$$* (\text{long}^*) \text{address} \mid = 1 \ll \text{bit}$$

- **void res ( void \*address, uint bit )**

Efface le **bit** spécifié de l'adresse mémoire (mise à 0). Ce **bit** peut être compris entre 0 et 31. Utiliser **RES** (voir plus haut) pour disposer de l'appel en ligne, ce qui est équivalent à l'expression suivante:

```
* ( long* ) address &= ~ ( 1L << bit )
```

- **void RES ( void \*address, uint bit )**

Efface le **bit** spécifié de l'adresse mémoire (mise à 0). Ce **bit** peut être compris entre 0 et 31. Dynamic C pourrait effectuer cet appel en ligne, ce qui est équivalent à l'expression suivante:

```
* ( long* ) address &= ~ ( 1L << bit )
```

- **uint IBIT ( uint port, uint bit )**

Lit le port d'E/S et retourne la valeur du **bit** spécifié. Ce **bit** peut être compris entre 0 et 7. Le port peut être un registre interne du Z180, ou un accès à des composants externes. Se référer au manuel de référence du contrôleur pour obtenir la liste des ports d'E/S. La fonction renvoie 1 si le **bit** spécifié est mit , et 0 si le **bit** est effacé.

- **void ISET ( uint port, uint bit )**

Met le **bit** spécifié du port d'E/S à 1. Ce **bit** peut être compris entre 0 et 7. Le port peut être un registre interne du Z180, ou un accès à des composants externes. La fonction génère un code comme le suivant:

```
in a, (c)
set bit, a
out (c), a
```

Se référer au manuel de référence du contrôleur pour obtenir la liste des ports d'E/S.

- **void IRES ( uint port, uint bit )**

Remet le bit spécifié du port d'E/S à 0. Ce **bit** peut être compris entre 0 et 7. Le **port** peut être un registre interne du Z180, ou un accès à des composants externes. La fonction génère un code comme le suivant:

```
in a, (c)
set bit, a
out (c), a
```

Se référer au manuel de référence du contrôleur pour obtenir la liste des ports d'E/S.

- **void hitwd ( )**

Déclenche le chien de garde (Watchdog timer), provoquant un reset hardware après 1,2 à 1,6 secondes (dépend du contrôleur). A moins que le chien de garde n'ai été invalidé, le programme doit appeler cette fonction périodiquement, sinon le contrôleur reset automatiquement. Ceci permet au contrôleur de sortir d'une erreur qui peut entraîner le programme dans une boucle infinie. Si le virtual driver est validé, il appellera **hitwd** en tâche de fond mais produira un chien de garde virtuel à la place. Voir **VdWdogHit** pour en savoir plus. Pour savoir comment positionner les cavaliers du chien de garde (pas disponible sur tous les contrôleurs), se référer au manuel du contrôleur.

- **int wderror ( )**

Détermine si le reset précédent a été provoqué par le chien de garde. Cette fonction n'est pas disponible sur tous les contrôleurs (voir le manuel de référence du contrôleur). La fonction retourne une valeur positive non nulle si le chien de garde a provoqué le dernier reset, et 0 dans le cas contraire. La fonction renvoie une valeur négative si cette propriété n'est pas supportée par le contrôleur.

- **void intrmode\_0 ( )**

Met le Z180 en mode interruption 0. Par défaut, le Dynamic C est en mode 2. Ne pas choisir un autre mode à moins que les interruptions de tous les périphériques utilisant le mode 2 n'aient été invalidés.

- **void intrmode\_1 ( )**

Met le Z180 en mode interruption 1. Par défaut, le Dynamic C est en mode 2. Ne pas choisir un autre mode à moins que les interruptions de tous les périphériques utilisant le mode 2 n'aient été invalidés. Cette fonction ne renvoie rien.

- **void intrmode\_2 ( )**

Met le Z180 en mode interruption 2. C'est le mode par défaut du Dynamic C. Ne pas choisir un autre mode à moins que les interruptions de tous les périphériques utilisant le mode 2 n'aient été invalidés.

- **void runwatch ( )**

Permet au Dynamic C de mettre à jour les expressions Watch. L'appel périodique à **runwatch** valide l'évaluation des expressions Watch pendant que le programme se déroule. Les expressions Watch sont toujours évaluées quand le programme est stoppé.

- **int kbhit ( )**

Détecte les frappes au clavier dans la fenêtre Dynamic C **STDIO**. La fonction renvoie une valeur non nulle si une touche a été frappée, sinon elle renvoie zéro.

- **void exit ( int exitcode )**

Stoppe le programme et retourne l'**exitcode** (code de sortie) au Dynamic C. Le Dynamic C utilise des valeurs de code supérieures à 128 pour les erreurs d'exécution (run-time errors). En mode courant (non Debug) cette fonction provoque un Time-out du chien de garde si celui-ci est validé.

*Cette fonction ne retourne pas dans le programme*

- **uint sysclock ( )**

Renvoie la vitesse d'horloge du système en unités de 1200 Hz. Certaines vitesses d'horloge courantes ainsi que les valeurs correspondantes de **sysclock** sont données ci-dessous:

6.144 MHz	0 x 1400 (5120)	9.126 MHz	0 x 1E00 (7680)
12.288 MHz	0 x 2800 (10240)	18.432 MHz	0 x 3C00 (15360)

Cette fonction retourne la vitesse d'horloge / 1200.

- **int powerlo ( )**

Il arrive que la tension d'alimentation descende assez bas pour générer une interruption de dysfonctionnement d'alimentation (power fail interrupt) sans toutefois descendre assez bas pour provoquer un Reset, puis revienne à un niveau correct. L'appel à cette routine à partir d'une interruption non masquable NMI permet de déterminer si l'alimentation est redevenue correcte. Se référer au manuel du contrôleur pour savoir si cette fonction est supportée. La fonction renvoie 1 si la tension est sous le niveau de déclenchement de la NMI, et 0 dans le cas contraire.

## MATH.LIB

La librairie standard Z-World contient des fonctions d'E/S et des fonctions virgule flottante. Les limitations mathématiques classiques s'appliquent à ces fonctions ainsi qu'à toute fonction générant une valeur en dehors des limites admises en virgule flottante (environ  $10^{38}$  à  $-10^{38}$ ), ce qui provoque une erreur de dépassement (overflow). L'infini est défini INF dans **DC.HH**.

Les fonctions trigonométriques comme tan(x) acceptent généralement des arguments en radians. Certaines fonctions trigo peuvent ne pas fonctionner si leurs arguments sont trop grands. Tout angle doit être normalisé pour tomber dans l'intervalle [ -pi, +pi] sans perte de précision.

- **int abs ( int x )**

Calcule la valeur absolue d'un argument entier.

- **float acos ( float x )**

Calcule l'arccosinus de x. La valeur de x doit être comprise entre -1 et +1. Si x est en dehors de ces limites, la fonction retourne 0 et signale une erreur de domaine.

- **float acot ( float x )**

Calcule l'arccotangente de **x**. La valeur de **x** doit être comprise entre - INF et + INF.

- **float acsc ( float x )**

Calcule l'arccoséquente de **x**. La valeur de **x** doit être comprise entre - INF et + INF.

- **float asec ( float x )**

Calcule l'arcséquente de **x**. La valeur de **x** doit être comprise entre - INF et + INF.

- **float asin ( float x )**

Calcule l'arcsinus de **x**. La valeur de **x** doit être comprise entre -1 et +1. Si **x** est en dehors de ces limites, la fonction retourne 0 et signale une erreur de domaine.

- **float atan ( float x )**

Calcule l'arctangente de **x**. La valeur de **x** doit être comprise entre - INF et + INF.

- **float atan2 ( float y, float x )**

Calcule l'arctangente de  $y/x$ . Si **y** et **x** sont nuls, la fonction renvoie 0 et signale une erreur de domaine. Autrement le résultat est retourné sous la forme suivante:

<i>angle</i>	<b>x</b> différent de 0, <b>y</b> différent de 0
PI/2	<b>x</b> = 0 , <b>y</b> > 0
-PI/2	<b>x</b> = 0 , <b>y</b> < 0
0	<b>x</b> > 0 , <b>y</b> = 0
PI	<b>x</b> < 0 , <b>y</b> = 0

- **float ceil ( float x )**

Renvoie le plus petit entier supérieur ou égal à **x**.

- **float cos ( float x )**

Calcule le cosinus de **x**.

- **float cosh ( float x )**

Calcule le cosinus hyperbolique de **x**. Si  $|x| > 89,8$  (approx.), la fonction renvoie INF et signale une erreur de domaine.

- **float deg ( float x )**

Renvoie en degrés l'angle **x** donné en radians.

- **float rad ( float x )**

Renvoie en radian l'angle **x** donné en degrés.

- **float exp ( float x )**

Renvoie la valeur de  $e^x$ . Si  $x > 89,8$  (approx.) la fonction retourne INF et signale une erreur de domaine. Si  $x < -89,8$  (approx.), la fonction retourne 0 et signale une erreur de domaine.

- **float fabs ( float x )**

Calcule la valeur absolue de **x**. La fonction retourne **x** si  $x \geq 0$ ; retourne **-x** dans le cas contraire.

- **float floor ( float x )**

Calcule l'entier le plus grand, inférieur ou égal au nombre donné.

- **float fmod ( float x , float y )**

Fonction modulo. Renvoie le reste de **x** une fois divisé par tous les multiples possibles de **y**. Par exemple si  $x = 22,7$  et  $y = 10,3$  le résultat entier de la division est 2, et le reste =  $22,7 - 2 \times 10,3 = 2,1$ .

- **float frexp ( float x, int \*n )**

Cette fonction partage **x** entre une fraction et un exposant ( $f \times 2^n$ ). La fonction retourne l'exposant dans l'entier **\*n** et la fraction (entre 0,5 et 0,999...) comme résultat de la fonction.

- **long labs ( long x )**

Calcule la valeur absolue de l'entier long **x**. La fonction renvoie **x** si  $x \geq 0$ ; **-x** dans le cas contraire.

- **float ldexp ( float x, int n )**

Calcule  $x * (\text{radix}^{**n})$ , où **n** est un entier et  $0,5 \leq x < 1,0$ .

- **float log ( float x )**

Calcule le logarithme naturel (base e) de **x**. La fonction renvoie -INF et signale une erreur de domaine quand  $x \leq 0$ .

- **float log10 ( float x )**

Calcule le logarithme décimal de **x**. La fonction renvoie -INF et signale une erreur de domaine quand  $x \leq 0$ .

- **float modf ( float x , int \*n )**

Partage **x** entre une partie entière et une partie décimale,  $f + n$ , où **n** est un entier et  $f$  satisfait à la condition  $|f| < 1,0$ . La fonction retourne la partie entière dans **\*n** et la partie décimale comme résultat de la fonction.

- **float poly ( float x , int n , float c [ ] )**

Calcule une valeur polynomiale par la méthode de Horner. Le terme **x** est la variable du polynome, **n** l'ordre de ce polynome, et **c** est un tableau contenant les coefficients de chaque puissance de **x**. Par exemple pour le polynome du quatrième ordre

$$10 x^4 - 3 x^2 + 4 x + 6$$

**n** sera 4 et les coefficients seront

$$c [4] = 10,0 \quad c [3] = 0,0 \quad c [2] = -3,0 \quad c [1] = 4,0 \quad c [0] = 6,0$$

- **float pow ( float x, float y )**

Renvoie  $x^y$ .

- **float pow10 ( float x )**

Renvoie  $10^x$ .

- **float sin ( float x )**

Calcule le sinus de **x**.

- **float sinh ( float x )**

Calcule le sinus hyperbolique de **x**. Si  $x > 89,8$  (approx.) la fonction retourne INF et signale une erreur de domaine. Si  $x < -89,8$  (approx.) la fonction retourne -INF et signale une erreur de domaine.

- **float sqrt ( float x )**

Calcule la racine carrée de **x**.

- **float tan ( float x )**

Retourne la tangente de **x**, où  $-8 \times \text{PI} \leq x \leq +8 \times \text{PI}$ . Si **x** est en dehors de ces bornes, la fonction retourne 0 et signale une erreur de domaine. Si la valeur de **x** est trop près d'un multiple de  $90^\circ$  ( $\text{PI} / 2$ ), la fonction retourne INF et signale une erreur de domaine.

- **float tanh ( float x )**

Renvoie la tangente hyperbolique de **x**. Si  $x > 49,9$  (approx.), la fonction retourne INF et signale une erreur de domaine. Si  $x < -49,9$  (approx.), la fonction retourne -INF et signale une erreur de domaine.

- **float \_pow10 ( int exp )**

Calcule les puissances intégrales de 10 (  $10^{\text{exp}}$  ).

## STDIO.LIB

Les fonctions suivantes adressent la fenêtre d'E/S standard du Dynamic C, qui est utilisée pour le débogage.

- **char \*gets ( char\* s )**

Cette fonction attend après la frappe d'une chaîne terminée par un <CR> (Carriage Return). Elle ne retourne pas dans le programme tant qu'un <CR> n'est pas tapé dans la fenêtre **STDIO**. Toutefois, la chaîne renvoyée n'est pas affectée par ce caractère (null terminated). La fonction renvoie la chaîne tapée aux coordonnées identifiées par le pointeur **s**. Assurez vous que le stockage est assez large pour la chaîne et qu'un seul traitement à la fois appelle cette fonction.

- **char getchar ( void )**

Cette fonction attend (dans une boucle passive) la frappe d'un caractère à partir de la fenêtre **STDIO** du Dynamic C. Assurez vous qu'un seul traitement à la fois appelle cette fonction.

- **int puts ( char \*s )**

Cette fonction écrit une chaîne, identifiée par le pointeur **s**, dans la fenêtre Dynamic C **STDIO**. La fenêtre **STDIO** interprétera toute séquence dans la chaîne contenant le code Escape. Assurez vous qu'un seul traitement à la fois appelle cette fonction. La fonction retourne 1 en cas de succès.

- **void putchar ( int ch )**

Ecrit un caractère unique dans **STDIO** (les 8 bits de poids faible de **ch**). Assurez vous qu'un seul traitement à la fois appelle cette fonction.

- **void sprintf ( char \*buffer, char \*format, ... )**

Analogue à la fonction standard **printf**, cette fonction prend une chaîne "format" (**\*format**), et un nombre variable de valeurs arguments à formater. Elle formate les arguments et place la chaîne formatée dans **\*buffer**. Assurez vous que:

1. Il y a assez d'arguments après **format** pour remplir les paramètres format de la chaîne format.
2. Les types d'arguments après **format** correspondent aux champs format dans **format**.
3. **buffer** est assez large pour contenir la chaîne formatée la plus longue possible.

Par exemple,

```
sprintf ( buffer, "%s=%x" , "Variable x" , 256 )
```

Devrait mettre la chaîne "Variable x = 100" dans **buffer**. Cette fonction peut être appelée par traitement de différentes priorités.

Les fonctions **printf ( )** et **sprintf ( )** ne sont pas réentrantes. La fonction **doprnt** implémente **printf** et **sprintf**, et utilise la fonction de sortie de caractères **putc** spécifiée par le programmeur. Ces fonctions acceptent des chaînes format et un nombre variable de paramètres dont les valeurs sont à imprimer selon le format, par exemple,

```
printf ( "Récapitulatif pour %s: \n" , personne ) ;  
printf ( " Age: %d, Revenus: Frs%8.2f" , age , revenus ) ;
```

La première ligne imprime une chaîne de caractères. Le **%s** dans le format indique à la fonction où et comment imprimer la chaîne de caractères.

La seconde ligne imprime 2 nombres, un entier **age** et un flottant **revenus**. Le **%d** dans le format indique à la fonction où et comment imprimer l'entier: comme une chaîne décimale, de format libre. Le **%8.2f** dans le format indique à la fonction d'imprimer revenus comme une valeur flottante, avec une largeur de champs de 8 caractères et 2 décimales.

```
Récapitulatif pour Marc Dutour:  
Age: 39 , Revenus: Frs98527.50
```

La syntaxe complète d'un code champs est:

`%[ + | - ] [ largeur [ .precision ] ] lettre` ou

+ justifie à droite la valeur dans le champs, si une largeur de champs est spécifiée.

- justifie à gauche la valeur dans le champs, si une largeur de champs est spécifiée.

*largeur* est la largeur du champs. Si elle n'est pas spécifiée, la largeur varie en fonction de la valeur à imprimer.

*precision* est le nombre de chiffres de la partie décimale pour les valeurs en virgule flottante.

*lettre* fixe l'interprétation de la donnée conformément à la liste suivante.

<b>d</b>	conversion décimale ( du type <b>int</b> )
<b>o</b>	conversion octal
<b>x</b>	conversion hexadécimale
<b>u</b>	conversion décimale non signé ( type <b>uint</b> )
<b>c</b>	représentation d'un <b>char</b> ( type <b>char</b> )
<b>s</b>	chaîne (terminaison null )
<b>e</b>	virgule flottante en représentation mantisse / exposant ( type <b>float</b> )
<b>f</b>	virgule flottante classique ( type <b>float</b> )
<b>g</b>	utilise une conversion e ou f , la plus courte des deux ( type <b>float</b> )
<b>l</b>	conversion décimale ( type <b>long</b> )

- **void printf ( char \*fmt , ... )**

Cette fonction standard accepte un nombre variable de valeurs arguments, compose une chaîne formatée à partir des valeurs, et imprime la chaîne formatée dans la fenêtre **STDIO**. Se référer à la description de **sprintf** pour plus de détails. Seulement un seul traitement à la fois peut utiliser cette fonction.

- **void doprnt ( int ( \*put ) ( ) , char \*fmt , void\* arg1 )**

C'est la routine support derrière toute routine **printf**. Passe une fonction **put** qui sort un octet. Elle sera appelée à chaque fois que **doprnt** sortira un caractère. Le terme **fmt** est la chaîne format qui spécifie la sortie. Le terme **arg1** pointe sur le premier paramètre devant être utilisé par la chaîne formatée. L'interprétation des paramètres dépend des champs format de la chaîne format. Cette routine entraîne la compilation et le chargement de beaucoup de fonctions mathématiques. Cette routine peut être appelée par traitement de différentes priorités.

- **char \*gtoa ( ulong num , char \*ibuf )**

Cette fonction utilise **\_gtoa** pour sortir un entier long non signé, **num**, dans le tableau de caractères **\*ibuf**. La fonction retourne un pointeur dans **ibuf**.

- **char \*ltoa ( long num , char \*ibuf )**

Cette fonction utilise **\_ltoa** pour sortir un entier long signé, **num**, dans le tableau de caractères **\*ibuf**. La fonction retourne un pointeur dans **ibuf**.

- **int gtoan ( ulong num )**

Cette fonction renvoie le nombre de caractères requis pour afficher un entier long non signé, **num**.

- **int ltoan ( long num )**

Cette fonction renvoie le nombre de caractères requis pour afficher un entier long signé, **num**.

- **void pint ( char flag , char code , int width , int (\*put) ( ) , int value )**

Ecrit une valeur entière courte comme une chaîne décimale conformément à la procédure de sortie de caractère **put** spécifiée par l'utilisateur. Le terme **width** spécifie la largeur de champs, si la valeur 0 est donnée, le champs sera aussi large que nécessaire pour représenter **value**. Le **flag**, si ' - ', indique que le champs est justifié à gauche, sinon il est justifié à droite. Si **code** est ' d ', la fonction traite **value** comme un entier signé, sinon elle le traite comme un entier non signé. La fonction imprime une série d'astérisques si la valeur ne rentre pas dans le champs spécifié.

- **void plint ( char left, char code, int n1, int (\*put) ( ) , long num )**

Cette fonction produit le même effet que **pint**, mais accepte et imprime un entier long.

- **int ftoa ( float f , char \*buf )**

Convertit le nombre pointeur flottant **f** en chaîne de caractères **\*buf**. La chaîne ne doit pas dépasser 12 caractères de long. La chaîne de caractères affiche seulement la mantisse jusqu'à 12 chiffres sans partie décimale. La fonction renvoie l'exposant (base 10) qui peut être utilisée pour compenser la partie décimale manquante. Par exemple,

**ftoa ( 1.0 , buf )**

génère la chaîne "1000000000", et renvoie -10. Si **f** est 45.678, **ftoa** génèrera la chaîne de caractères "45678" et retournera l'entier exposant -3, ce qui signifie  $45678 \times 10^{-3}$ .

- **void plhex ( char left , int n1 , int (\*put) ( ) , long num )**

Ecrit un entier long (signé ou non signé) en format hexadécimal. Le terme **left** spécifie le caractère de remplissage qui va du coté gauche du nombre actuel. Si **left** est ' ', un espace est utilisé comme caractère de remplissage. Le terme **n1** est la longueur attendue de la sortie. Des astérisques seront écrits si **num** nécessite une largeur supérieure à **n1**. Sinon le caractère de remplissage **left** sera utilisé pour combler les espaces restant. Passer une fonction (**put**) qui sortira un caractère. La fonction **put** devrait prendre un caractère argument. Le terme **num** est le nombre à convertir et à sortir. Cette fonction peut être appelée par traitement de différentes priorités.

- **void phex ( char left, int n1, int (\*put) ( ) , int num )**

Similaire à **plhex**. Cette fonction imprime la représentation hexadécimale d'un entier court (signé ou non signé). Se référer à la description de **plhex** pour plus de détails.

- **void pflt ( char flag , char code , int width , int digits , int (\*put) ( ) , float value , int prec )**

Imprime une valeur virgule flottante formatée en utilisant la procédure spécifiée **put** de sortie d'un caractère. Le programmeur a un très bon contrôle du format. Le **flag**, si ' - ', indique que le champs de sortie est justifié à gauche. Si c'est '0', le champs est justifié à droite et rempli de zéros. Sinon le champs est justifié à droite et rempli d'espaces.

Le **code** peut être 'e', 'f', ou 'g'. Ces formats correspondent aux conventions de programmation établies depuis des années. Le format E affiche une mantisse avec "e" et un exposant. Le format F est le format décimal standard. Le format G permet au compilateur de décider d'utiliser le format "e" ou "f".

Le terme **width** est la largeur du champs. Si zéro est spécifié, le champs sera aussi large que nécessaire pour représenter **value**. Les termes **prec** et **digits** régissent le nombre de chiffres significatifs à imprimer. Si **prec** est non nul (true), la fonction imprimera **digits** chiffres significatifs. Sinon la fonction affiche 6 chiffres significatifs.

La fonction n'affiche que des astérisques si la valeur ne rentre pas dans le champs spécifié.

- **char \*itoa ( int value , char \*buf )**

Convertit un entier signé **value** en chaîne de caractères dans **\*buf**, avec quand c'est nécessaire le signe moins en tête. La fonction supprime les premiers zéros superflus mais garde un chiffre zéro pour **value** = 0. La valeur maximum est 32767. La fonction renvoie un pointeur en fin de chaîne dans **\*buf** (terminaison null).

- **char \*utoa ( uint value , char \*buf )**

Convertit un entier non signé **value** en chaîne de caractères dans **\*buf**. La fonction supprime les premiers zéros superflus mais garde un chiffre zéro pour **value** = 0. La valeur maximum est 65535. La fonction renvoie un pointeur en fin de chaîne dans **\*buf** (terminaison null).

- **char \*htoa ( int value , char \*buf )**

Convertit un entier **value** en chaîne de caractères hexadécimale dans **\*buf**. Les zéros de tête ne sont pas supprimés. La fonction renvoie un pointeur en fin de chaîne dans **\*buf** (terminaison null).

- **char \*hltoa ( long value , char \*buf )**

Convertit un entier long **value** en chaîne de caractères hexadécimale dans **\*buf**. Les zéros de tête ne sont pas supprimés. La fonction renvoie un pointeur en fin de chaîne dans **\*buf** (terminaison null).

- **char outchrs ( char c , int n , int ( \*put ) ( ) )**

Utilise la fonction de sortie mono caractère **put** pour générer **n** fois le caractère **c**. La fonction **put** doit prendre un paramètre caractère. La fonction renvoie la valeur du caractère **c**.

- **char \*outstr ( char \*buf , int ( \*put ) ( ) )**

Sort la chaîne **\*buf** par appels à la fonction de sortie mono caractère **put**. La fonction **put** doit prendre un paramètre caractère. La fonction renvoie un pointeur en fin de chaîne dans **\*buf** (terminaison null).

## STRING.LIB

Les fonctions suivantes sont des fonctions de chaîne C standard.

- **float atof ( char \*sptr )**

Convertit une chaîne de caractères en valeur virgule flottante. L'espace initial est ignoré (compatibilité ANSI). La fonction renvoie la valeur convertie.

- **int atoi ( char \*sptr )**

Convertit une chaîne de caractères en valeur entière. L'espace initial est ignoré (compatibilité ANSI). La fonction renvoie la valeur convertie.

- **int atol ( char \*sptr )**

Convertit une chaîne de caractères en valeur entier long. L'espace initial est ignoré (compatibilité ANSI). La fonction renvoie la valeur convertie.

- **void \*memset ( void \*dst, byte ch, uint n )**

Fixe le départ mémoire à **dst** de **n** occurrences de l'octet **ch**. La fonction retourne un pointeur à l'adresse suivant le dernier octet écrit.

- **char \*strcpy ( char \*dst, char \*src )**

Copie la chaîne **\*src** dans la chaîne **\*dst**. La fonction copie au moins un octet (le null) et retourne un pointeur à **\*dst**.

- **char \*strncpy ( char \*dst, char \*src, uint n )**

Copie au plus **n** caractères de **\*src** dans **\*dst**. Peut se terminer avant si la terminaison null est rencontrée dans **\*src** avant les **n** caractères. La terminaison null n'est pas copiée si **n** est rencontré avant la terminaison null (par exemple, le programmeur doit faire attention aux cas de longueur délimité). La fonction renvoie un pointeur à **\*dst**.

- **char \*strcat ( char \*dst, char \*src )**

Concatène la chaîne **\*src** à la fin de **\*dst**. La chaîne destination doit être assez large pour comporter les caractères supplémentaires. La fonction renvoie un pointeur à **\*dst**.

- **char \*strncat ( char \*dst, char \*src, uint n )**

Concatène jusqu'à **n** caractères de **\*src** à la fin de **\*dst**. Une terminaison null est rajoutée à la fin de **\*dst** si **n** caractères sont copiés avant de rencontrer la terminaison null dans **\*src**. La fonction renvoie un pointeur à **\*dst**.

- **int strcmp ( char \*a, char \*b )**

Compare deux chaînes. La fonction retourne la différence relative entre la première paire de caractères différents, ce qui conduit au résultat suivant

= 0     si tous les caractères sont égaux  
< 0     si  $a_i < b_i$   
> 0     si  $a_i > b_i$

Ces fonctions sont très utiles pour le tri.

- **int strncmp ( char \*a, char \*b, uint n )**

Compare deux chaînes jusqu'à **n** caractères. Le retour de cette fonction est similaire à celui de **strcmp**.

- **char\* strchr ( char \*src, char ch )**

Scrute **\*src** jusqu'à la première occurrence de **ch**. La fonction renvoie un pointeur à la première occurrence de **ch** dans **\*src**. Elle renvoie un pointeur null si **ch** n'est pas trouvé.

- **char\* strrchr ( char \*src, int ch )**

Similaire à **strchr**, excepté pour l'algorithme de recherche qui commence par la fin jusqu'au début de **\*src**. La fonction renvoie un pointeur à la dernière occurrence de **ch** dans **\*src**. Elle renvoie un pointeur null si **ch** n'est pas trouvé.

- **uint strspn ( char \*src, char \*set )**

Renvoie la longueur du segment initial maximum de **\*src**, qui est constitué entièrement de caractères dans **\*set**.

- **uint strcspn ( char \*src, char \*set )**

Renvoie la longueur du segment initial maximum de **\*src**, qui est constitué entièrement de caractères non contenus dans **\*set**.

- **char\* strpbrk ( char \*s1, char \*s2 )**

Situe la première occurrence dans **\*src** de tout caractère contenu dans **\*set**. La fonction renvoie un pointeur sur l'occurrence. La fonction renvoie un pointeur null si rien n'est trouvé.

- **void\* memcpy ( void \*dst, void \*src, uint n )**

Copie **n** caractères de la mémoire **\*src** dans la mémoire **\*dst**. Le recouvrement est correctement géré. La fonction renvoie le pointeur **\*dst**.

- **void\* memchr ( void\* src, int ch, uint n )**

Recherche jusqu'à **n** caractères de **ch** dans le buffer **\*src**. La fonction renvoie un pointeur sur la première occurrence de **ch** si elle est trouvée dans une profondeur de recherche de **n** caractères. Sinon elle renvoie un pointeur null.

- **int strlen ( char \*s )**

Calcule la longueur de la chaîne **\*s**, sans inclure la terminaison null. La fonction renvoie le nombre d'octets contenus dans la chaîne.

- **int toupper ( int c )**

Convertit un caractère **c** en son équivalent upper-case.

- **int tolower ( int c )**

Convertit un caractère **c** en son équivalent lower-case.

- **int islower ( int c )**

Vérifie si **c** est un caractère lower-case. La fonction renvoie une valeur non nulle si c'est le cas, zéro autrement.

- **int isupper ( int c )**

Vérifie si **c** est un caractère upper-case. La fonction renvoie une valeur non nulle si c'est le cas, zéro autrement.

- **int isdigit ( int c )**

Vérifie si **c** est un chiffre ASCII (0-9). La fonction renvoie une valeur non nulle si c'est le cas, zéro autrement.

- **int isxdigit ( int c )**

Vérifie si **c** est un chiffre hexadécimal (0-9, a-f, A-F). La fonction renvoie une valeur non nulle si c'est le cas, zéro autrement.

- **int ispunct ( int c )**

Vérifie si **c** est un signe de ponctuation. La fonction renvoie une valeur non nulle si c'est le cas, zéro autrement.

- **int isspace ( int c )**

Vérifie si **c** est un blanc, une tabulation, une nouvelle ligne ou une entrée. La fonction renvoie une valeur non nulle si c'est le cas, zéro autrement.

- **int isprint ( int c )**

Vérifie si **c** est affichable. La fonction renvoie une valeur non nulle si c'est le cas, zéro autrement.

- **int isalpha ( int c )**

Vérifie si **c** est une lettre ASCII. La fonction renvoie une valeur non nulle si c'est le cas, zéro autrement.

- **int isalnum ( int c )**

Vérifie si **c** est alphanumérique (A à Z, a à z et 0 à 9). La fonction renvoie une valeur non nulle si c'est le cas, zéro autrement.

- **int isgraph ( int c )**

Vérifie si **c** est un caractère d'impression visible. La fonction renvoie une valeur non nulle si c'est le cas, zéro autrement.

- **int iscntrl ( int c )**

Vérifie si **c** est un caractère de contrôle (inférieur à H20). La fonction renvoie une valeur non nulle si c'est le cas, zéro autrement.

- **float strtod ( char \*s, char \*\*tailptr )**

Convertit une chaîne en une valeur virgule flottante. Le terme **\*s** est la chaîne à convertir, et **\*\*tailptr** est un pointeur sur un pointeur de caractère. **\*\*tailptr** assigne le point d'arrêt de conversion dans **\*s** ( la reprise de conversion reste possible à **\*\*tailptr**). Si aucune conversion ne s'effectue, **\*\*tailptr** renvoie 0L. L'espace blanc initial est ignoré. Cette fonction est compatible ANSI. La fonction renvoie la valeur convertie.

- **long strtol ( char \*s, char \*\*tail, int base )**

Convertit une chaîne en une valeur entière longue. Le terme **\*s** est la chaîne à convertir, **\*\*tail** assigne la dernière position de conversion, et **base** indique le radix de la conversion, qui peut aller de 2 à 36. Quand **base** est 0, la fonction convertit selon la syntaxe C. Par exemple, si la chaîne commence par "0x," la fonction interprétera la chaîne au format hexadécimal. La fonction saute "l'espace blanc" initial. La fonction positionne le pointeur de queue **\*\*tail** à la position du caractère sur lequel la conversion s'est terminée ou a échoué. La conversion suivante doit reprendre à la position spécifiée par **\*\*tail**. Si aucune conversion ne s'effectue, **\*\*tail** renvoie 0L. L'espace blanc initial est ignoré. Cette fonction est compatible ANSI. La fonction renvoie la valeur convertie.

Soyez prudent avec le double pointeur.

- **char \*strtok ( char \*src, char \*brk )**

Scrute **\*src** pour trouver les tokens séparés par les caractères délimiteurs spécifiés dans **\*brk**. Le premier appel prend un **\*src** non nul. Les appels suivants avec un pointeur nul pour **\*src** continuent de chercher les tokens dans la chaîne. La fonction renvoie un pointeur sur le premier caractère du token. Si elle rencontre un délimiteur de terminaison, elle le modifie en caractère nul pour que le token soit terminé. Cette fonction modifie la chaîne source. La fonction retourne un pointeur nul si elle ne trouve pas de token.

- **char \*strstr ( char \*string, char \*target )**

Retourne un pointeur sur la première occurrence de sous-chaîne **\*target** dans **\*string**. La fonction retourne un pointeur nul si **\*target** n'est pas trouvé dans **\*string**. La fonction retourne la chaîne pointeur si la cible est nulle.

- **int memcmp ( void \*a, void \*b, uint n )**

Compare deux espaces mémoire **a** et **b** et renvoie la différence relative entre la première paire d'octets qui diffèrent, s'il y en a une. Le résultat de la fonction est

```
= 0    si tous les octets sont égaux
< 0    si ai < bi
> 0    si ai > bi
```

La fonction arrête la comparaison après **n** octets.

## SYS.LIB

Cette librairie contient des fonctions support diverses.

- **int setjmp ( jmp-buf env )**

Stocke le compteur programme PC (Program Counter), le pointeur de pile SP (Stack pointer) et autres informations de l'état courant dans **env**. Les informations sauvees peuvent être restaurées en exécutant **longjmp**. Un programme typique est proposé ci-dessous.

```
switch ( setjmp ( e ) ) {
    case 0 :           // première fois
        fx ( ) ;      // fx ( ) doit prendre un longjmp
        break ;      // si on arrive ici, fx ( ) est réalisé avec succès
// si on arrive ici, fx ( ) doit avoir appelé longjmp
    case 1 :
        faire la gestion des exceptions
        break ;
// similaire au case 1, mais code d'exception différent.
    case 2 :
        ...
}
f ( ) {
    g ( )
    ...
}
g ( ) {
    ...           // ici, le code d'exception 2 provoque un
                // saut en arrière sur l'occurrence setjmp,
                // mais force setjmp à retourner 2.
    longjmp ( e,2 ) ; // par conséquent, case 2 s'exécute
}
                // dans la partie des tests.
```

La fonction renvoie zéro quand elle est exécutée. Après exécution de **longjmp**, le compteur programme, le pointeur de pile, etc..., sont restaurés en l'état quand **setjmp** a été exécuté une première fois. De toute façon, **setjmp** renvoie n'importe quelle valeur spécifiée dans l'expression **longjmp**.

- **void longjmp ( jmp-buf env , int value )**

Restaure l'environnement de pile sauvé dans **env**. La valeur entière passée à **longjmp** est retournée comme le résultat de fonction de **setjmp** quand le long jump est pris en compte. Voir la description de **setjmp** pour son utilisation.

- **void \*malloc ( uint size )**

Alloue un bloc dynamique de **size** octets. Appeler **bfree** avant d'utiliser **\*malloc** (le compilateur appelle automatiquement **bfree** avant main si un espace heap est réservé dans les options mémoire logique). **\*malloc** doit être

préempté par un autre **\*malloc**, car **\*malloc** utilise un pointeur de liste libre et global. L'espace heap doit être alloué en utilisant l'option mémoire logique du menu **Options** afin d'utiliser **\*malloc** ( par défaut la taille du heap est 0). La fonction renvoie un pointeur au début du bloc alloué, ou un pointeur nul si l'espace n'est pas disponible.

- **uint bfree ( void \*lo , void \*hi )**

Définit un bloc RAM, de **\*lo** à **\*hi** compris, comme disponible pour une allocation dynamique. La fonction renvoie une valeur non nulle quand elle a été exécutée avec succès, sinon zéro.

- **int free ( void \*f )**

Renvoie un bloc ( **\*f** ) de RAM allouée dynamiquement en liste libre. La fonction renvoie une valeur non nulle quand elle a été exécutée avec succès, sinon zéro.

- **int pack ( void )**

Réduit la fragmentation de mémoire dynamique en regroupant des blocs libres adjacents. La fonction renvoie le nombre total d'octets libres.

- **void \*calloc ( uint count , uint size )**

Alloue de la mémoire "heap" pour un espace de **count** éléments de **size** octets. La fonction trouve un bloc mémoire libre dans la liste, lui donne la bonne taille et retourne un pointeur sur le bloc. La fonction initialise entièrement l'espace avec des zéros. La fonction retourne un pointeur sur le bloc alloué ou retourne un pointeur nul si elle ne trouve pas de bloc.

- **void swap ( byte a [ ] , byte b [ ] , int s )**

Echange un tableau **a** avec un tableau **b**, octet pour octet, pour les premiers **size** octets.

- **int qsort ( void \*base , uint n , uint s , int (\*cmp) ( ) )**

Réalise un tri rapide avec pivot central, contrôle de pile, et méthode de comparaison easy-to-change. Le terme **\*base** pointe à la base d'un tableau à trier (d'une structure taille fixe). La valeur **n** est le nombre d'éléments à trier, et **s** est la taille de chaque élément dans le tableau. Le programmeur doit fournir la fonction de comparaison **cmp** qui indique l'ordre des deux structures. La fonction de comparaison prend les pointeurs des deux structures

**int cmp ( \*p , \*q )**

et retourne -1 si le premier est inférieur au second, 0 si les structures sont égales, et 1 si le premier est supérieur au second. La fonction **qsort** renvoie zéro si elle a été exécutée avec succès, sinon une valeur non nulle.

- **char \*realloc ( void \*ptr , uint size )**

Alloue un nouveau bloc de taille **size**, copie les données de l'ancien bloc ( **\*ptr** ) dans le nouveau bloc, libère l'ancien bloc, et renvoie un pointeur sur le nouveau bloc. Si la fonction ne parvient pas à allouer un nouveau bloc, le résultat de la fonction est un pointeur nul.

- **isr\_ptr getvect ( uint intrno )**

Obtient l'adresse du handler de l'interruption numéro **intrno**. Pour cette fonction le nombre doit être pair et inférieur à 255. La fonction renvoie l'adresse du handler. Le type **isr\_ptr** est un pointeur de fonction qui ne renvoie rien et n'utilise pas d'argument.

- **void setvect ( uint intrno , isr\_ptr isr )**

Fixe un nouveau handler **isr** pour l'interruption numéro **intrno**. Le terme **intrno** doit être pair et inférieur à 255. Le type **isr\_ptr** est un pointeur de fonction qui ne renvoie rien et n'utilise pas d'argument.

- **int iff ( )**

Vérifie si le drapeau d'interruption est ON. La fonction renvoie 1 si le drapeau d'interruption est ON, sinon 0.

- **void setireg ( char value )**

Fixe le registre d'interruption du Z180 avec les 8 bits de poids fort de la valeur 16 bits **value** spécifiée.

- **char readireg ( )**

Renvoie la valeur du registre d'interruptions du Z180 sur les 8 bits de poids fort de la valeur retournée. Les 8 bits de poids faible étant fixés à zéro.

- **void CoBegin ( CoData \*cd )**

**CoBegin** initialise une structure **CoData**. Le drapeau INIT est mis, mais le drapeau STOPPED est effacé.

- **void CoReset ( CoData \*cd )**

**CoReset** reset une structure **CoData**. Les drapeaux INIT *et* STOPPED sont mis.

- **void CoPause ( CoData \*cd )**

**CoPause** pause une structure **CoData**. Le drapeau STOPPED est mis, mais le drapeau INIT est effacé.

- **void CoResume ( CoData \*cd )**

**CoResume** relance une structure **CoData**. Les drapeaux STOPPED *et* INIT sont effacés.

- **int isCoDone ( CoData \*cd )**

La fonction **isCoDone** renvoie 1 (vrai) si les drapeaux INIT *et* STOPPED sont mis. Sinon elle renvoie 0.

- **int isCoRunning ( CoData \*cd )**

La fonction **isCoRunning** renvoie 1 (vrai) si le drapeau STOPPED n'est pas mis. Sinon elle renvoie 0.

- **void \_prot\_init ( )**

Effectue une super initialisation. La fonction initialise les données internes nécessaires à la récupération des variables **protected** après un crash. Pour s'assurer que ce mécanisme de protection fonctionne, lancer cette fonction une fois dans un programme *avant* toute affectation des variables **protected**.

- **void \_prot\_recover ( )**

Effectue la récupération de l'assignement partiellement fait d'une variable **protected**. Appeler cette fonction après une coupure d'alimentation ou après une situation similaire qui ne vide pas la mémoire.

- **void reload\_vec ( int vector , int ( \*function ) ( ) )**

Charge une routine de service interruption sur un vecteur spécifié pendant le fonctionnement.

PARAMETRES: **vector** est le vecteur d'interruption à servir. **\*function** est l'adresse de la routine service interruption.

ATTENTION: **reload\_vec** écrit en mémoire flash quand le programme est exécuté sur un contrôleur possédant une Flash EPROM. Faites attention que l'appel de cette fonction n'engendre pas des écritures répétitives, une Flash EPROM ayant une "durée de vie" d'environ 10000 écritures.

## **XMEM.LIB**

Cette librairie contient des fonctions d'accès à la mémoire étendue.

- **ulong xmadr ( void\* address )**

Convertit une adresse logique **address** en adresse physique, selon les registres de plan mémoire. Utiliser BBR, CBR et CBAR pour déterminer l'adresse physique de toute adresse logique donnée. La fonction retourne l'adresse physique.

- **char xgetchar ( long address )**

Trouve le caractère dont l'adresse est spécifiée par **address** physique (20 bits). La fonction retourne la valeur du caractère.

- **int xgetint ( ulong address )**

Trouve un entier dont l'adresse est spécifiée par **address** physique (20 bits). La fonction retourne la valeur entière.

- **ulong xgetlong ( ulong address )**

Trouve un entier long dont l'adresse est spécifiée par **address** physique (20 bits). La fonction retourne un entier long non signé.

- **float xgetfloat ( ulong address )**

Trouve une valeur virgule flottante dont l'adresse est spécifiée par **address** physique (20 bits). La fonction retourne une valeur virgule flottante.

- **void xputchar ( long address , char value )**

Mémoire une valeur **value** de caractère à une **address** physique (20 bits).

- **void xputint ( long address , int value )**

Mémoire une valeur **value** entière à une **address** physique (20 bits).

- **void xputlong ( long address , long value )**

Mémoire une valeur **value** longue à une **address** physique (20 bits).

- **void xputfloat ( ulong address , float value )**

Mémoire une valeur **value** flottante à une **address** physique (20 bits).

- **void xmem2root ( ulong src , void\* dst , uint n )**

Copie un bloc de **n** octets de la mémoire étendue **src** vers la mémoire standard (root) **\*dst**. L'adresse **src** est une **address** physique (20 bits).

- **void root2xmem ( void \*src , ulong dst , uint n )**

Copie un bloc de **n** octets de la mémoire standard (root) **\*src** en mémoire étendue **dst**. L'adresse **dst** est une **address** physique (20 bits).

- **uint xstrlen ( ulong address )**

Retourne la longueur de la chaîne située à l'**address** en mémoire étendue. **Address** est une adresse physique (20 bits).

- **uint x-makadr ( ulong address )**

Calcule l'adresse logique à partir de l'**address** physique. La fonction fixe aussi CBR à un nouveau numéro de page et retourne l'adresse logique en HL. L'ancien CBR est sauvé dans **af** (paire de registres alternatifs A et F). *Ne jamais* appeler cette fonction à partir des fonctions **xmem**. Z-World recommande également de ne pas appeler cette fonction à partir des fonctions C car il est facile d'oublier qu'une fonction C peut être placée automatiquement en **xmem**.

- **ulong a32-24 ( ulong address )**

Convertit l'**address** physique 20 bits (dans un entier 32 bits) en une adresse segmentée (24 bits). Les adresses segmentées ont la structure suivante:

|----- CBR 8 bits -----|----- adresse Z180 16 bits -----|

- **ulong a24\_32 ( ulong address )**

Convertit l'**address** segmentée 24 bits en une adresse physique 20 bits ( dans un entier 32 bits). Le segment (deuxième octet de l'adresse segmentée) est uniquement valable si **address** est en **xmem**, c'est à dire **address** > ou = 0xE000. Sinon le segment est ignoré. Les registres CBR et BBR du MMU sont tous deux utilisés pour le calcul de la valeur sortante. La fonction retourne un entier long non signé contenant les 20 bits d'adresse physique équivalant à l'adresse logique étendue fournie.

## 2 - LIBRAIRIES MULTITACHES

Les bibliothèques multitâches décrites au chapitre 2 incluent le noyau temps réel (real-time kernel), le noyau temps réel simplifié et le driver virtuel (virtual driver).

### **RTK.LIB**

Cette bibliothèque est le noyau temps réel complet. Le noyau temps réel simplifié (SRTK) est décrit plus loin.

- **int request ( uint tasknum )**

Demande au noyau de traiter immédiatement la tâche spécifiée par **tasknum**. Si une demande pour cette tâche est en attente, cet appel n'aura pas d'effet. La tâche spécifiée sera traitée par un top ultérieur, quand la priorité le permettra.

- **void run\_every ( int tasknum, int period )**

Demande au noyau de traiter la tâche spécifiée par **tasknum** tous les **period** tops. La première demande survient après **period** tops. C'est une exactitude, aucun top n'est rajouté ou perdu sur la période.

- **void run\_after ( int tasknum, long delay )**

Demande au noyau de traiter la tâche spécifiée par **tasknum** après qu'un délai de **delay** tops soit écoulé.

- **void run\_at ( int tasknum, void\* time )**

Demande au noyau de traiter la tâche spécifiée par **tasknum** quand le temps est plus grand ou égal au temps spécifié par le pointeur **time**. Le pointeur **time** pointe sur un nombre de 48 bits (stocké avec l'octet de poids faible en tête) qui représente le nombre de tops depuis l'appel à **init\_kernel**.

- **void run\_cancel ( int tasknum )**

Annule toute demande en attente pour la tâche spécifiée par **tasknum**.

- **void gettimer ( void\* time )**

Retourne le temps (48 bits) dans la zone de 6 octets pointée par **time**.

- **void run\_timer ( )**

Cette fonction doit être appelée par une routine d'interruption entre 10 et 500 fois par seconde pour que le noyau temps réel fonctionne. Chaque appel à cette fonction constitue un "top" noyau, ainsi toutes les valeurs de temps utilisées par les autres fonctions du noyau dépendent de la vitesse à laquelle cette fonction est appelée.

- **int comp48 ( void\* time1, void\* time2 )**

Compare deux valeurs de temps 48 bits. La fonction retourne:

-1 pour **time1** < **time2**,  
0 pour **time1** = **time2**, et  
+1 pour **time1** > **time2**.

- **void rkernel ( )**

C'est le coeur du noyau temps réel, appelé par **run\_timer**. Cette fonction sera retournée aussitôt si aucun changement ne c'est produit sur la tâche courante exécutée. Si elle décide de changer les tâches basées sur les interruptions de service comme **run\_every** ou **run\_after**, elle ne sera pas retournée tant que la nouvelle tâche ne sera pas elle même retournée ou appellera **suspend**.

- **void suspend ( uint ticks )**

Cette routine doit être appelée seulement de l'intérieur d'une tâche donnée. Elle permet à une tâche de se suspendre elle-même pendant un nombre spécifié de tops, après lesquels elle continuera à être appelée automatiquement. Son exécution reprend sur l'instruction suivant l'appel à **suspend**.

Si **ticks** est 0, la tâche est suspendue pendant un temps indéfini, jusqu'à ce que la tâche soit appelée par un agent externe, comme par exemple un appel à **run\_every()**. L'utilisation d'une commande **while** est la méthode classique pour demander à **suspend** l'attente d'un événement externe:

```
while ( !event ( ) ) suspend (20) ;
```

Cet exemple vérifie tous les 20 tops si l'événement est survenu, puis reprendra son exécution à cet endroit. Une suspension peut être paramétrée jusqu'à 65535 tops.

- **int init\_kernel ( )**

Initialise le noyau temps réel. Cette fonction n'utilise pas de paramètre. Toutefois, le programme appelant doit contenir certaines définitions.

Les fonctions devant être lancées comme des tâches doivent être déclarées sans paramètre et retourner un entier. Un tableau global de pointeurs de tâches, **Ftask**, doit être déclaré avec la première tâche ( **Ftask[0]** ) donnant la priorité la plus haute, et la dernière tâche la priorité la plus basse. **#define NTASKS** doit être le nombre de tâches. Ensuite, définissez une interruption périodique avec une routine de service qui lance **run\_timer**. Une option possible consiste à définir **TASKSIZE\_STORE** de la taille de la zone de stockage de la tâche (50 par défaut, si **TASKSIZE\_STORE** n'est pas défini).

Toutes les définitions présentées ci-dessus doivent figurer dans le code source avant toute référence aux fonctions du noyau temps réel.

## **SRTK.LIB**

Cette bibliothèque contient les fonctions du noyau temps réel simplifié.

- **void srtk\_hightask ( )**

C'est la routine appelée toute les 25 mSec par le SRTK qui lance les tâches de haute priorité. Celle donnée dans la bibliothèque est une routine "maquette".

Pour obtenir une tâche haute priorité défini par l'utilisateur dans le SRTK, écrivez simplement une routine portant le même nom. Spécifiez **#nointerleave** pour garantir la compilation de cette tâche haute priorité.

- **void srtk\_lowtask ( )**

C'est la routine appelée toute les 100 mSec par le SRTK qui lance les tâches de basse priorité. Celle donnée dans la bibliothèque est une routine "maquette".

Pour obtenir une tâche basse priorité défini par l'utilisateur dans le SRTK, écrivez simplement une routine portant le même nom. Spécifiez **#nointerleave** pour garantir la compilation de cette tâche basse priorité.

- **void init\_srtkernel ( )**

Initialise le noyau temps réel simplifié. Une fois appelée, des interruptions périodiques invoqueront automatiquement les tâches hautes et basses priorités du SRTK.

Initialiser le driver virtuel et **#define RUNKERNEL 1** avant d'appeler cette fonction.

## VDRIVER.LIB

Cette librairie contient les fonctions du driver virtuel. Le driver virtuel procure un certain nombre de services comme les chiens de garde virtuels (watchdog timers) et une tâche à accès immédiat (fastcall) de très haute priorité.

Le driver virtuel procure aussi des routines d'attente à utiliser par les commandes **waitfor**, **DelayMs**, **DelaySec**, et **DelayTick**.

- **void VdInit ( )**

Initialise le driver virtuel. Le PRT1 du Z180 cadence le driver virtuel toute les 1/1280 secondes. Le driver virtuel cadence le RTK ou le SRTK tout les 32 tops (ou 25 millisecondes) si **#define RUNKERNEL** est défini.

Pour les services d'accès immédiat, le driver virtuel cadence **vd-quickloop** chaque **n** tops (1/1280 secondes) avec **n** compris entre 1 et 255. **vd-quickloop** doit être défini et la définition écrasera ainsi la version maquette de la librairie. (**#define VD-FASTCALL 1** doit être défini de la même façon).

**VdInit** doit être appelé avant que le programme puisse utiliser le SRTK, les chiens de garde virtuels, les routines d'attente **waitfor** ou l'accès immédiat fastcall. **VdInit** réalise un appel à **\_GLOBAL\_INIT**. Donc, un programme préparé par l'utilisateur ne devra pas le faire.

- **int VdGetFreeWd ( byte count )**

Retourne un compteur de chien de garde virtuel libre et commence un décomptage à partir de **count**. Les horloges de chiens de garde virtuels décrémentent toute les 25 mSec (32 tops de driver virtuel). Quand un chien de garde virtuel atteint 0, il réinitialise le processeur. Tant que l'horloge de chien de garde virtuel est active, le logiciel doit réinitialiser l'horloge périodiquement avec un appel à **VdWdogHit**. La fonction retourne l'entier ID d'une horloge de chien de garde virtuel inutilisé.

Si **count < ou = 2**, **VdWdogHit** doit être appelé toutes les 25 mSec. Si **count = 255**, lancer le chien de garde au moins toutes les 6,375 secondes.

- **void VdWdogHit ( int wd )**

Réinitialise l'horloge du chien de garde virtuel à *n* comptages, où *n* est l'argument de l'appel au **VdGetFreeWd** qui obtient le chien de garde virtuel **wd**. La fonction renvoie 0 si **wd** est hors domaine , ou 1 si l'exécution est correcte.

- **int VdReleaseWd ( int wd )**

Désactive le chien de garde virtuel **wd** et le renvoie dans le groupe de chiens de garde. La fonction retourne 0 si **wd** est hors domaine, ou 1 si l'exécution est correcte.

- **int vd\_initquickloop ( int n )**

Initialise la fonction "accès immédiat" du driver virtuel pour être lancée tous tes **n** tops. La valeur de **n** doit être comprise entre 0 et 255. Si **n = 0**, elle désactive **fastcall**. Utiliser **#define VD\_FASTCALL 1**, appeler **VdInit**, puis appeler cette fonction. ( **VdInit** initialise l'accès immédiat sur *off*). La fonction retourne 1 en cas de succès, ou 0 pour une mauvaise valeur de **n**.

- **void VdAdjClk ( )**

Synchronise la seconde horloge logicielle utilisée par **DelaySec** avec l'horloge temps réel. Appeler cette fonction une fois par jour ou alors pour garder les horloges synchronisées.

- **vd\_fastcall ( )**

Cette fonction est appelée par le driver virtuel pour des lancements ultra rapides tous les **n** tops, où **n** est l'argument pour **vd\_initquickloop ( )** et peut être compris entre 0 et 255. Utiliser **#define VD\_FASTCALL 1** pour activer ce lancement. **n = 0** inhibe **fastcall**.

## **3 - LIBRAIRIES CONTROLEURS**

Chaque librairie décrite dans ce chapitre, est la librairie principale pour un type de contrôleur Z-World donné. Quelques librairies du chapitre 5 supportent aussi des contrôleurs particuliers.

### **BL1000.LIB**

Cette fonction supporte le contrôleur BL1000.

- **int ad\_rd8 ( int chan )**

Lit une valeur 8 bits sur le convertisseur A/N du BL1000. **chan** est le numéro de la voie (0 à 3). La valeur renvoyée apparaît comme un nombre 12 bits car elle est décalée sur la gauche par 4 bits.

La fonction renvoie un code entre 0 et 4095, ou sinon -32768 si une erreur est survenue.

### **BL11XX.LIB**

Ces fonctions supportent les contrôleur séries BL 1100.

- **int ad\_rd10 ( int chan )**

Lit une valeur 10 bits sur le convertisseur A/N du BL 1100. Les 3 bits de poids faible de **chan** spécifient le numéro de voie (0 à 7); le quatrième bit doit être 0 pour le mode bipolaire ou 1 pour le mode unipolaire ( ajouter 8 au numéro de voie pour le mode unipolaire). La valeur renvoyée apparaît comme un nombre 12 bits car elle est décalée sur la gauche par 2 bits.

La fonction renvoie un code entre -2048 et 2047 en mode bipolaire; entre 0 et 4095 en mode unipolaire; -32768 si une erreur est survenue.

- **int ad\_rd12 ( int chan )**

Lit une valeur 12 bits sur le convertisseur A/N du BL 1100. Les 3 bits de poids faible de **chan** spécifient le numéro de voie (0 à 7); le quatrième bit doit être 0 pour le mode bipolaire ou 1 pour le mode unipolaire ( ajouter 8 au numéro de voie pour le mode unipolaire).

La fonction renvoie un code entre -2048 et 2047 en mode bipolaire; entre 0 et 4095 en mode unipolaire; -32768 si une erreur est survenue.

- **int ad\_rd10s ( int chan, int count, int \*buf, uint divider )**

Echantillonne à intervalles réguliers des données sur le convertisseur A/N du BL 1100. **chan** est le numéro de voie (0 à 7), plus 8 pour le mode unipolaire (sinon bipolaire), **count** spécifie le nombre d'échantillons à collecter, et **buf** pointe sur le buffer de stockage des échantillons. **divider** détermine la vitesse d'échantillonnage = horloge / (20 \* **divider**). **divider** ne doit pas être plus petit que 36, ce qui produit une vitesse de 12800 éch./sec. avec une horloge 9,216 MHz. Les interruptions seront désactivées sauf si **NODISINT** est défini.

La fonction retourne 1 en cas de succès, 0 si un échantillon est manquant car **divider** est trop petit ou si une interruption est survenue durant l'échantillonnage.

- **int ad\_rd12a ( int chan )**

Lit une valeur 12 bits sur le convertisseur A/N alternatif LTC1290 du BL 1100. Les 3 bits de poids faible de **chan** spécifient le numéro de voie (0 à 7); le quatrième bit doit être 0 pour le mode bipolaire ou 1 pour le mode unipolaire ( ajouter 8 au numéro de voie pour le mode unipolaire). Le temps d'exécution est d'environ 350 microsecondes avec une horloge à 9,216 MHz. Les interruptions sont désactivées pendant environ 300 microsecondes.

- **void wdac ( int value )**

Ecrit **value** sur le convertisseur N/A (DAC) du BL 1100. **value** doit être compris entre 0 et 4095, ce qui produit une sortie de  $2,5 * value / 4096$  Volts.

- **int ad\_rd ( int chan )**

Même fonction que **ad\_rd10**.

- **void setctc ( char nctc, char mode, char timer, char intr )**

Initialise le compteur CTC **nctc** (0 - 3). **mode** spécifie un des sept modes de fonctionnement possible du compteur:

0	Lance le compteur à horloge/16, déclenchement immédiat.
1	Lance le compteur à horloge/256, déclenchement immédiat.
2	Fixe le compteur en mode horloge externe.
4	Lance le compteur à horloge/16, déclenchement sur front montant de CLK/.
5	Lance le compteur à horloge/256, déclenchement sur front montant de CLK/.
6	Lance le compteur à horloge/16, déclenchement sur front descendant de CLK/.
7	Lance le compteur à horloge/256, déclenchement sur front descendant de CLK/.

**timer** spécifie la constante de temps à charger dans le compteur. **intr** indique si oui ou non le timer doit provoquer des interruptions ( une valeur non nulle valide les interruptions, 0 désactive les interruptions).

La fonction ne retourne rien.

- **void setdaisy ( char code )**

Fixe la priorité relative des interruptions entre les trois unités d'E/S du KIO basé sur la valeur de **code** décrite ci-dessous:

0	Désactivé	4	CTC, PIO, SIO
1	SIO, CTC, PIO	5	PIO, SIO, CTC
2	SIO, PIO, CTC	6	PIO, CTC, SIO
3	CTC, SIO, PIO (par défaut)	7	Désactivé

- **void setled1 ( char value )**

Allume la LED #1 si **value** est non nulle, l'éteint si **value** = 0.

## BL14\_15.LIB

Ces fonctions supportent les contrôleurs séries BL1400 et BL1500.

- **int Read555 ( uint \*lapsecount )**

Lit le comptage de timer0 correspondant au laps de temps écoulé pour que le circuit 555 atteigne le temps  $t = 1,1 RC$ . Le comptage du timer est retourné dans **\*lapsecount**. Le circuit 555 doit être au préalable chargé avec **Set555(maxcount)**. La fonction retourne:

0	Si timer0 n'est pas arrêté et que le circuit 555 n'a pas atteint $t = 1,1 RC$ .
1	Si le circuit 555 a atteint $t = 1,1 RC$ et a généré une interruption sur INT1 et DREQ0.
-1	Si timer0 a fini de compter <b>maxcount</b> et que le circuit 555 n'a pas atteint $t = 1,1 RC$ .

- **void set555 ( uint maxcount )**

Charge **maxcount** dans timer0 et le configure pour générer une interruption. Prépare DMA0 à recevoir les données de TMRD0L de timer0. Prépare INT1 et DREQ0 à recevoir un signal “done” du circuit 555. Déclenche le circuit 555.

- **void charger1302 ( int on-off, int diode, int resistor )**

Commute ON ou OFF le chargeur à régime lent du circuit DS1302. **diode** est 1 ou 2 pour le nombre de diodes de VCC2 à VCC1. **resistor** est 2, 4 ou 8 (en K Ohms) pour la résistance en ligne.

- **int ReadTime1302 ( struct tm\* thistime )**

Lit une donnée horloge temps réel (RTC) du DS1302 vers la structure temps pointée par **thistime**. La fonction retourne 0 en cas de succès, -1 si l’horloge temps réel est en mode halte.

- **int WriteTime1302 ( struct tm\* thistime )**

Ecrit la donnée structure temps pointée par **thistime** dans l’horloge temps réel (RTC) du DS1302. La fonction retourne 0 en cas de succès, -1 si l’horloge temps réel est en mode halte.

- **void WriteRam1302 ( int ram\_loc, int data )**

Ecrit une donnée dans **ram\_loc** (0 à 30) du DS1302. La fonction retourne 1 si l’écriture s’effectue avec succès, et -1 si une erreur survient.

- **int ReadRam1302 ( int ram\_loc )**

Lit une donnée dans **ram\_loc** (0 - 30) du DS1302. La fonction retourne une donnée RAM, ou -1 si une erreur survient.

- **void WriteBurst1302 ( char \*pdata, int count )**

Ecrit **count** octets du tableau **pdata** en RAM du DS1302, en commençant à RAM emplacement 0.

- **void ReadBurst1302 ( char \*pdata, int count )**

Retourne un nombre **count** d’octets lus dans DS1302, en commençant à RAM emplacement 0 jusqu’au tableau **pdata**.

- **void Write1302 ( int reg, int data )**

Ecrit **data** dans le registre spécifié du DS1302.

- **int Read1302 ( int reg )**

Lit une donnée dans le registre spécifié du DS1302. La fonction retourne la donnée lue.

- **void setPIOCA ( byte mask )**

Les bits actifs de **mask** (état 1) sont fixés dans **PIOCAShadow**. Ce résultat est ensuite envoyé à **PIOCA**. Les bits actifs deviennent des bits *entrées*.

**PIOCA <----- PIOCAShadow <----- PIOCAShadow OR mask**

- **void resPIOCA ( byte mask )**

Les bits actifs de **mask** (état 1) sont remis à zéro dans **PIOCAShadow**. Le résultat est ensuite envoyé dans **PIOCA**. Les bits actifs deviennent des bits *sorties*.

**PIOCA <----- PIOCAShadow <----- PIOCAShadow AND NOT mask**

- **void setPIODA ( byte mask )**

Les bits actifs de **mask** (état 1) sont fixés comme sortie courante de **PIODA**.

**PIODA <----- PIODA OR mask**

- **void resPIODA ( byte mask )**

Les bits actifs de **mask** (état 1) sont remis à zéro dans la sortie courante de **PIODA**.

**PIODA <----- PIODA AND NOT mask**

- **void setPIOCB ( byte mask )**

Les bits actifs de **mask** (état 1) sont fixés dans **PIOCBshadow**. Ce résultat est ensuite envoyé à **PIOCB**. Les bits actifs deviennent des bits *entrées*.

**PIOCB <----- PIOCBshadow <----- PIOCBshadow OR mask**

- **void resPIOCB( byte mask )**

Les bits actifs de **mask** (état 1) sont remis à zéro dans **PIOCBshadow**. Le résultat est ensuite envoyé dans **PIOCB**. Les bits actifs deviennent des bits *sorties*.

**PIOCB <----- PIOCShadow <----- PIOCShadow AND NOT mask**

- **void setPIODB ( byte mask )**

Les bits actifs de **mask** (état 1) sont fixés comme sortie courante de **PIODB**.

**PIODB <----- PIODB OR mask**

- **void resPIODB ( byte mask )**

Les bits actifs de **mask** (état 1) sont remis à zéro dans la sortie courante de **PIODB**.

**PIODB <----- PIODB AND NOT mask**

- **void mgset12adr ( int addr )**

Fixe l'adresse courante du PLCbus. Toute lecture ou écriture ultérieure du PLCbus accédera aux cartes d'extension avec cette adresse. L'adresse reste valide tant qu'une nouvelle adresse n'est pas fixée. Le terme **addr** est l'adresse physique 12 bits de l'extension PLCbus. Le groupe des 4 bits de poids faible est transmis en dernier (comme **BUSADR2**). Le troisième groupe est transmis en premier (comme **BUSADR0**).

- **void mgwrite12data ( int addr, int data )**

Écrit une donnée sur le périphérique PLCBus à l'adresse **addr**. Seuls, les quatre bits de poids faible de la donnée sont utilisables (pour **BUSWR**).

- **int mgread12data0 ( int addr )**

Lit une donnée (avec **BUSRD0**) sur le périphérique PLCBus à l'adresse **addr**. Le résultat de la fonction contient la donnée.

- **int mgread12data1 ( int addr )**

Lit une donnée (avec **BUSRD1**) sur le périphérique PLCBus à l'adresse **addr**. Le résultat de la fonction contient la donnée.

- **int mgread12data2 ( int addr )**

Lit une donnée (avec **BUSRD1**) sur le périphérique PLCBus à l'adresse **addr**. Le résultat de la fonction contient la donnée.

- **void mgwrite4data ( int value )**

Écrit les quatre bits de poids faible de **value** (avec **BUSWR**) sur un périphérique PLCBus. Cette fonction sous-entend que les adresses des périphériques PLCBus ont été fixés sur le bus (avec **mgset12adr**).

- **void mgsave\_pbus ( )**

Place l'état courant du PLCBus sur la pile. Cette fonction doit seulement être appelée en duo avec **mgrestore\_pbus**. Sinon, la pile ne sera plus équilibrée.

- **void mgrestore\_pbus ( )**

Restaure à partir de la pile l'état courant du PLCBus. Cette fonction doit seulement être appelée en duo avec **mgsave\_pbus**. Sinon la pile ne sera plus équilibrée.

- **void mgplc\_set\_relay ( int number, int relay, int state )**

Commute un relai ON ou OFF sur le PLCBus. La carte d'extension sur le PLCBus doit être un XP8300 ou un XP8400 et son **number** doit être compris entre 0 et 63. Le terme **relay** sélectionne le numéro du relai sur la carte sélectionnée (0-5 pour les cartes XP8300 et 0-7 pour les cartes XP8400). Le terme **state** doit être fixé à 1 pour commuter le relai ON et à 0 pour commuter le relai OFF.

Se référer aux **manuels XP8300, XP8400 et SE1100** pour obtenir les informations concernant ces périphériques et la façon de les numéroter.

- **int mgplcrly\_board ( int number )**

Calcule l'adresse physique d'une carte relais à partir de son numéro de carte **number**. Ce nombre doit être compris entre 0 et 63. (La carte numéro 0 correspond à l'adresse 0x000; La carte numéro 63 correspond à l'adresse 0x11F). La valeur de retour a le troisième et le premier "nibble" interchangé.

Se référer aux **manuels XP8300, XP8400 et SE1100** pour obtenir les informations concernant ces périphériques et la façon de les numéroter.

- **int mgplcuio\_board ( int number )**

Calcule l'adresse physique d'une carte E/S universelle (XP8200) à partir de son numéro de carte **number**. Ce nombre doit être compris entre 0 et 15. (La carte numéro 0 correspond à l'adresse 0x040; La carte numéro 15 correspond à l'adresse 0x04F). La valeur de retour a le troisième et le premier "nibble" interchangé.

Se référer aux **manuels XP8100 et XP8200** pour obtenir les informations concernant ces périphériques et la façon de les numéroter.

- **int mgplc\_dac\_board ( int number )**

Calcule l'adresse physique d'une carte sorties analogiques (XP8600) à partir de son numéro de carte **number**. Ce nombre doit être compris entre 0 et 63. (La carte numéro 0 correspond à l'adresse 0x020; La carte numéro 63 correspond à l'adresse 0x13F). La valeur de retour a le troisième et le premier "nibble" interchangé.

Se référer aux **manuels XP8600 et XP8900** pour obtenir les informations concernant ces périphériques et la façon de les numéroter.

- **void mginit\_dac ( )**

Initialise une carte de sorties analogiques (XP8600) sur le PLCBus. Cette fonction sous-entend que l'adresse de la carte a été fixée sur le bus (avec **mgset12adr**).

- **void mgwrite\_dac1 ( int value )**

Écrit la valeur entière 12-bits **value** dans le registre A de DAC1 d'une carte sorties analogiques (XP8600) sur le PLCBus. Cette fonction sous-entend que l'adresse de la carte a été fixée sur le bus (avec **mgset12adr**). La carte de sorties analogiques n'effectue aucune nouvelle conversion tant qu'un appel à **mglatch\_dac1** n'a pas été lancé.

- **void mglatch\_dac1 ( )**

Pousse les données du registre A dans le registre B de DAC1 d'une carte CNA (XP 8600) sur le PLCBus. La valeur convertie en sortie de DAC 1 est celle du registre B. Cette fonction sous-entend que l'adresse de base de la carte a été fixée sur le bus (avec **mgset12adr**). S'assurer que le registre A contient des données valides. voir **mgwritedac1** plus haut.

- **void mgset\_dac1 ( int value )**

Écrit une valeur 12-bits entière dans le registre A, puis pousse la donnée du registre A vers le registre B de DAC 1 de la carte CNA sélectionnée sur le PLCBus (XP8600). Cette fonction sous-entend que l'adresse de base de la carte a été fixée sur le bus (avec **mgset12adr**). La fonction combine les effets de **mgwrite\_dac1** et **mglatch\_dac1**.

- **void mgwrite\_dac2 ( int value )**

Écrit la valeur entière 12-bits **value** dans le registre A de DAC2 d'une carte sorties analogiques (XP8600) sur le PLCBus. Cette fonction sous-entend que l'adresse de la carte a été fixée sur le bus (avec **mgset12adr**). La carte de sorties analogiques n'effectue aucune nouvelle conversion tant qu'un appel à **mglatch\_dac2** n'a pas été lancé.

- **void mglatch\_dac2 ( )**

Pousse les données du registre A dans le registre B de DAC2 d'une carte CNA (XP 8600) sur le PLCBus. La valeur convertie en sortie de DAC 2 est celle du registre B. Cette fonction sous-entend que l'adresse de base de la carte a été fixée sur le bus (avec **mgset12adr**). S'assurer que le registre A contient des données valides. voir **mgwritedac2** plus haut.

- **void mgset\_dac2 ( int value )**

Ecrit une valeur 12-bits entière dans le registre A, puis pousse la donnée du registre A vers le registre B de DAC 2 de la carte CNA sélectionnée sur le PLCBus (XP8600). Cette fonction sous-entend que l'adresse de base de la carte a été fixée sur le bus (avec **mgset12adr**). La fonction combine les effets de **mgwrite\_dac2** et **mglatch\_dac2**.

- **void lc\_char ( byte data )**

Imprime un caractère sur l'écran LCD.

- **void lc\_ctrl ( byte cmd )**

Ecrit une commande de contrôle sur l'écran LCD.

- **void lc\_init ( )**

Initialise l'écran LCD et les variables accessoires. L'écran LCD utilise le port A des PIO du BL 1400.

- **void lc\_cgram ( void\* ptr )**

Charge jusqu'à 8 caractères spéciaux dans le générateur de caractères de l'écran LCD à partir du tableau d'octets \*p. Le premier octet du tableau est le nombre d'octets à stocker (à raison de 8 octets par caractère), avec une valeur maximale de 64 pour 8 caractères. Les codes pour les caractères spéciaux sont 0, 1, 2, 3, 4, 5, 6 et 7.

- **void lc\_printf ( char\* format, . . . )**

Cette commande fonctionne comme **printf**, mais pour l'écran LCD. Les séquences d'échappement suivantes (Escape) sont également implémentés:

ESC 1	Active le curseur
ESC 0	Désactive le curseur
ESC c	Efface la fin de la ligne à partir du curseur
ESC b	Valide le clignotement du curseur
ESC n	Invalide le clignotement du curseur
ESC e	Efface l'afficheur et retourne le curseur
ESC p n mm	Positionne le curseur en ligne n, colonne mm

Le code caractère d'échappement (Escape) est **0x1B**.

- **lc\_kxinit ( )**

Initialise le pilote du clavier et les variables accessoires. Assurez vous de définir **KEY4x6** quelque part en début de programme si vous utilisez un clavier 4 x 6.

```
# define KEY4x6
```

Sinon, le pilote utilisera par défaut une gestion clavier 2 x 6.

- **int lc\_kxget ( int mode )**

Obtient la valeur de touche à partir du buffer FIFO de clavier. Si **mode = 0**, la valeur de touche est éliminée du buffer. Sinon la valeur de touche est conservée dans le buffer. Dans l'un ou l'autre cas, la fonction retourne la valeur de touche, ou -1 si le buffer clavier est vide.

- **void lc\_keyscan ( )**

Scrute le clavier 2 x 6 ou 4 x 6. Une touche valide devrait être persistante pendant les appels **DebounceCount** à **lc\_keyscan**. La fonction entre les appuis de touche valides dans le buffer FIFO du clavier. Le logiciel accédera à ces appuis de touches par l'utilisation de **lc\_kxget**.

La "stabilisation" est assurée en étant sûr qu'une touche est frappée par des appels consécutifs **DebounceCount** à **lc\_keyscan**. Le nombre de "stabilisation" (debouncing) peut être modifié en redéfinissant **DebounceCount**:

**# define DebounceCount nn**

S'il n'est pas redéfini, **DebounceCount** est de 20 par défaut. Si **lc\_keyscan** est appelé toutes les 25 mSec. et **DebounceCount** = 20, alors une touche doit être pressée pendant 20 x 25 mSec. = 500 mSec. pour être considérée comme valide.

## **BL16XX.LIB**

Ces fonctions supportent les contrôleurs de la série BL 1600.

- **void VIOInit ()**

Fonction "fantôme" utilisée comme hôte pour l'initialisation globale des variables d'E/S virtuelles. Les entrées et sorties virtuelles sont lues ou écrites à chaque appel de la fonction **VIODrvr**. Les entrées numériques vont de **DIGIN1** à **DIGIN12**. Les sorties numériques vont de **OUTB1** à **OUTB8** et de **HC1** à **HC6**. Une entrée numérique doit avoir la même valeur lors de 2 lectures consécutives pour être validée.

- **void VIODrvr ()**

Met à jour les entrées virtuelles **DIGIN1** à **DIGIN12**. Les sorties virtuelles **OUTB1** à **OUTB8** et **HC1** à **HC6** sont envoyées aux ports de sorties correspondants.

- **int up\_digin (int channel)**

Lit la valeur d'une voie entrée numérique, **channel** devant être compris entre 1 et 12. La fonction retourne 1 ou 0, état logique de la voie lue.

- **void up\_setout (int channel, int onoff)**

Fixe une sortie numérique à 1 (active) ou à 0 (inactive), **channel** devant être compris entre 1 et 14. Les voies 1 à 8 correspondent à **OUTB1** - **OUTB8**, les voies 9 à 14 correspondent à **HC1** - **HC6**. Le terme **onoff** est la valeur de sortie à affecter à la voie: 1 --> état haut ou actif, 0 --> état bas ou inactif.

## **PK21XX.LIB**

Ces fonctions supportent les contrôleurs de la série PK 2100.

- **void VIOInit ()**

Fonction "fantôme" utilisée comme hôte pour l'initialisation globale des variables d'E/S virtuelles. Les entrées et sorties virtuelles sont lues ou écrites à chaque appel de la fonction **VIODrvr**. Les entrées numériques vont de **DIGIN1** à **DIGIN7** et **U1IN** à **U7IN**. Les sorties numériques vont de **OUT1** à **OUT10** et de **RELAY1** à **RELAY2**. Une entrée numérique doit avoir la même valeur lors de 2 lectures consécutives pour être validée.

- **void VIODrvr ()**

Met à jour les entrées virtuelles **DIGIN1** à **DIGIN7** et **U1IN** à **U7IN**. Les sorties virtuelles **OUT1** à **OUT10**, **RELAY1** et **RELAY2** sont envoyées aux ports de sorties correspondants.

- **int up\_digin (int channel)**

Lit la valeur d'une voie entrée numérique, **channel** devant être compris entre 1 et 7. La fonction retourne 1 ou 0, état logique de la voie lue.

- **void up\_setout (int nout, int onoff)**

Fixe une sortie numérique à 1 (actif) ou à 0 (inactif). **channel** doit être compris entre 1 et 10, correspondant aux sorties **OUT1** à **OUT10**, 11 pour **RELAY1** et 12 pour **RELAY2**. Le terme **onoff** est la valeur à affecter en sortie: 1 --> état haut ou actif, 0 --> état bas ou inactif.

- **void init\_daccal ( )**

Fonction “fantôme” utilisée comme hôte pour l’initialisation globale des valeurs de calibration DACCAL de la sortie analogique du PK 2100.

- **void up\_daccal ( int val )**

Active la sortie analogique (CNA) avec une valeur de calibration pour le calibre 0 - 10.000 mVolts.

- **void up\_dacout ( int val )**

Envoie une valeur N/A non calibrée sur la voie CNA.

- **void up\_expout ( int val )**

Envoie une valeur N/A non calibrée sur la voie EXP.

- **int up\_adtest ( int chan , int testval )**

Compare une entrée tension d’une entrée universelle **chan** à **testval**. La voie doit être comprise entre 1 et 7. La fonction renvoie 1 si la tension d’entrée est supérieure à **testval**, sinon elle renvoie 0.

- **int up\_uncal ( int val )**

Retourne un entier non calibré (0 - 1023), donné pour une valeur de sortie N/A en mVolts. La fonction renvoie l’équivalent entier de la valeur d’entrée en mVolts.

- **int up\_docal ( int rawval )**

Convertit **rawval** en sa valeur calibrée d’entrée A/N.

- **int up\_adcal ( int chan )**

Lit une entrée universelle spécifiée (1 - 7). La fonction renvoie la valeur A/N calibrée pour 0 - 10000 mVolts.

- **int up\_adrd ( int chan )**

Lit une voie d’entrée universelle (1 - 7). La fonction renvoie la valeur (sortie CAN) du rang de la voie spécifiée.

- **void up\_dac420 ( int current )**

Génère un courant 4-20 mA sur une sortie N/A du CNA. Le matériel doit être configuré pour un fonctionnement en mode boucle de courant. La gamme pour le courant va de 4000 à 20000.

- **int up\_in420 ( )**

Lit l’entrée universelle 6 comme une entrée 4-20 mA. Le matériel doit être configuré pour un fonctionnement en mode boucle de courant. La fonction retourne une valeur dans la gamme 4000-20000.

- **float up\_higain ( int mode )**

Lit une voie haut-gain avec H7 non connecté (pas de cavalier). Les retours de fonction sont comme suit:

Si **mode** = 1, renvoie AN+ (0-1 Volts). Sous-entend AN- à la masse.

Si **mode** = 2, renvoie AN+ - AN- (0-1 Volts).

Si **mode** = 3, renvoie AN- (0-10 Volts).

Si **mode** n’est pas défini, renvoie -100.

(AN+ est le point “chaud” de la mesure, AN- est le point “froid” ou référence)

## PK22XX.LIB

Ces fonctions supportent les contrôleurs des séries PK 2200.

- **void VIOInit ( )**

Fonction “fantôme” utilisée comme hôte pour l’initialisation globale des variables d’E/S virtuelles. Les entrées et

sorties virtuelles sont lues ou écrites à chaque appel de la fonction **VIODrvr**. Les entrées numériques vont de **DIGIN1** à **DIGIN16**. Les sorties numériques vont de **OUT1** à **OUT14**. Une entrée numérique doit avoir la même valeur lors de 2 lectures consécutives pour être validée.

- **void VIODrvr ( )**

Met à jour les entrées virtuelles **DIGIN1** à **DIGIN16**. Les sorties virtuelles **OUT1** à **OUT14** sont envoyées aux ports de sorties correspondants.

- **int up\_digin ( int channel )**

Lit la valeur d'une voie entrée numérique, **channel** devant être compris entre 1 et 16. La fonction retourne 1 ou 0, état logique de la voie lue.

- **void up\_setout ( int nout , int onoff )**

Fixe une sortie numérique à 1 (active) ou à 0 (inactive), **channel** devant être compris entre 1 et 14. Le terme **onoff** est la valeur de sortie à affecter à la voie: 1 --> état haut ou actif , 0 --> état bas ou inactif.

## CM71\_72.LIB

La librairie **CM71\_72** contient des fonctions écrites tout spécialement pour les modules coeur à microprocesseur séries **CM7100** et **CM7200** utilisés conjointement aux cartes d'évaluation et modules optionnels **LCD/Keypad** (écran LCD et clavier). Des fonctions de même nom doivent exister dans d'autres librairies.

- **void lc\_kxinit ( )**

Initialise le pilote clavier (en matrice 2 x 6) et les variables accessoires.

- **int lc\_kxget ( int mode )**

Obtient la valeur de la touche à partir du buffer FIFO clavier. Si **mode** = 0, la valeur de la touche est retirée du buffer. Sinon la valeur de la touche est conservée dans le buffer. Dans les deux cas, la fonction retourne la valeur de touche, ou -1 si le buffer clavier est vide.

- **void lc\_keyscan ( )**

Scrute le clavier 2 x 6 de la carte d'évaluation du **CM7100**. Une frappe clavier, pour être validée, doit être maintenue pendant **DebounceCount** échantillons. (**DebounceCount** est définie globalement avec une valeur par défaut de 10). Appeler **lc\_keyscan** dans une routine périodique. Les touches validées sont placées dans un buffer clavier. On accède au buffer clavier à l'aide de la fonction **lc\_kxget**.

- **void up\_beep ( uint k )**

Active le bip sonore (buzzer) pendant k millisecondes. Le nombre de mSec. pendant lesquelles le buzzer fonctionne dépend de la routine périodique qui appelle **lc\_beepscan**. Par exemple, si **lc\_beepscan** est appelé toute les 50 mSec., alors **BeepScale** = 1/50 = 0,02. **BeepScale** est une valeur globalement définie valant par défaut 0,04.

- **void lc\_beepscan ( )**

Utilisable avec le buzzer de la carte d'évaluation du **CM7100**. La durée du bip est préalablement fixée avec **up\_beep**. Le comptage buzzer est décrémenté à chaque fois que cette fonction est appelée. Le buzzer est désactivé quand le décompte atteint zéro.

Cette fonction doit être appelée par une routine périodique, par exemple une routine lancée toute les 25 mSec.

## 4 - LIBRAIRIES AASC

La librairie AASC (Abstract Application-Level Serial Communication) et ses fonctions supports de bas niveau facilitent la communication série entre contrôleurs, ou entre un contrôleur et un autre périphérique comme par exemple un PC.

### AASC.LIB

Les librairies AASC permettent au programmeur de créer des flux de caractères bufferisés qui effectuent des entrées / sorties sur, ou à partir des ports de communication des périphériques. Une librairie principale, **AASC.LIB**, contient toute les fonctions nécessaires pour cela.

Les routines de haut niveau s'occupent de la gestion des connexions entre le buffer circulaire de bas niveau et les librairies pilote matériel. Cela permet à n'importe quel pilote matériel d'utiliser le même contexte de programmation.

- **CHANNEL aascOpen ( int Type , char CRTS , long Param , void ( \*brqfnc ) ( ) )**  
Ouvre une voie de périphérique **Type**, et initialise ce périphérique avec le paramètre **Param**.

PARAMETRES: **Type** est le type de périphérique de communication à ouvrir.

**DEV\_Z0** pour le port Z0,  
**DEV\_Z1** pour le port Z1,  
**DEV\_SCC** pour le port du contrôleur de communication série,  
**DEV\_ZNET** pour le périphérique réseau, et  
**DEV\_UART** pour le XP 8700.

**CRTS** spécifie si la poignée de main (handshaking) CTS/RTS doit être utilisée: 1 si elle doit être utilisée, 0 si elle ne doit pas être utilisée.

**Param** spécifie toute les autres options de communication. Z-World a défini les macros suivantes:

Nombre de bits de données	Nombre de bits de stop	Nombre de bits de parité
ASCI_PARAM_7D ASCI_PARAM_8D	ASCI_PARAM_1STOP ASCI_PARAM_2STOP	ASCI_PARAM_NOPARITY ASCI_PARAM_OPARITY ASCI_PARAM_EPARITY
SCC_7DATA SCC_8DATA	SCC_1STOP SCC_2STOP	SCC_NOPARITY SCC_OPARITY SCC_EPARITY

Ces macros s'appliquent pour le port Z0 du Z180 et pour le contrôleur de communication série. Se référer aux descriptions des pilotes Dynamic C ou à l'aide en ligne pour obtenir des macros supplémentaires.

Choisir une macro de chaque colonne et les additionner ensemble pour obtenir une configuration de voie, comme illustré ci-dessous.

**ASCI\_PARAM\_7D | ASCII\_PARAM\_1STOP | ASCII\_NOPARITY**

Deux combinaisons de macros souvent utilisées ont également été définies.

**ASCII\_PARAM\_1200**      Unité de base pour la vitesse de transmission en Bauds. A multiplier par le facteur de vitesse Bauds divisé par 1200 (par exemple 8 pour 9600 bps).

**ASCII\_PARAM\_8N1**      Spécifie 8 bits de données, 1 bit de stop et pas de parité.

Par exemple , le canal Z0 en format 8N1 à 19200 bps doit avoir

**Param = 16 \* ASCII\_PARAM\_1200 | ASCII\_PARAM\_8N1.**

**brqfnc** est un pointeur sur une fonction à appeler par l'interruption du **Z0** quand une demande de break est détectée. Le retour pour **void \*brkfnc** est null.

VALEUR DE RETOUR: En mots de 16 bits de type **CHANNEL** pour tout fonctionnement ultérieur sur les voies. **aascOpen** retourne null si aucune voie ne peut être assignée si le traitement des break n'est pas utilisé.

- **void aascClose ( CHANNEL Channel )**

Ferme la voie numérotée **Channel**. Avant tout, **aascClose** appelle la routine de fermeture spécifique au périphérique. Ensuite le stockage associé à cette voie est réattribué en liste libre.

PARAMETRE: **Channel** est la voie logique.

- **void aascSetReadBuf ( CHANNEL channel , char \*Buffer , uint size )**

Désigne une zone mémoire pointée par **Buffer** de taille **size** comme buffer de réception pour **channel**.

PARAMETRES: **channel** en lecture, doit être ouvert par un appel à **aascOpen**. **Buffer** est l'adresse du buffer de réception. **size** est la taille du buffer de réception.

- **void aascSetWriteBuf ( CHANNEL Channel , char \*Buffer , uint size )**

Désigne une zone mémoire pointée par **Buffer** de taille **size** comme buffer de transmission pour **channel**.

PARAMETRES: **channel** en écriture, doit être ouvert par un appel à **aascOpen**. **Buffer** est l'adresse du buffer de transmission. **size** est la taille du buffer de transmission.

- **void aascRxSwitch ( CHANNEL Channel , char OnOff )**

Active ou désactive la voie de réception.

PARAMETRES: **Channel** est la voie logique. **OnOff** 0 est inactive, sinon la voie de réception est active.

- **void aascTxSwitch ( CHANNEL Channel , char OnOff )**

Active ou désactive la voie de transmission.

PARAMETRES: **Channel** est la voie logique. **OnOff** 0 est inactive, sinon la voie de transmission est active.

- **uint aascReadChar ( CHANNEL Channel , char \*Dest )**

Lit un caractère de la voie **Channel** et l'envoie sur la mémoire pointée par **Dest**. Le récepteur sera validé automatiquement si le flux de contrôle CTS/RTS est validé et que le buffer de réception possède au moins 16 octets de libres (après la lecture).

PARAMETRES: **Channel** est la voie logique. **Dest** est l'adresse (buffer) à laquelle lire les caractères.

VALEUR DE RETOUR: Le nombre actuel d'octets lus sur la voie.

- **uint aascReadBlk ( CHANNEL Channel , void \*Dest , uint Length , char Flags )**

Lit un bloc de **Length** octets de la voie logique **Channel** sur la mémoire pointée par **Dest**. Si **Flags** est non nul, soit **Length** en entier sera lu ou aucun octet ne sera lu. Le récepteur sera validé automatiquement si le contrôle de flux est validé et si le buffer de réception possède au moins 16 octets de libres (après la lecture).

PARAMETRES: **Channel** est la voie logique. **Dest** est l'adresse (buffer) à laquelle lire. **Length** est le nombre d'octets à lire. Si **Flags** est non nul, soit **Length** en entier sera lu ou aucun octet ne sera lu.

VALEUR DE RETOUR: Le nombre actuel d'octets lus sur la voie.

- **uint aascWriteChar ( CHANNEL Channel , char Src )**

Ecrit un caractère **Src** sur la voie logique **Channel**. Le transmetteur est validé automatiquement après le transfert du caractère.

VALEUR DE RETOUR: Le nombre actuel d'octets écrit sur la voie.

- **uint aascWriteBlk ( CHANNEL Channel , void \*Src , uint Length , char Flags )**

Ecrit un bloc de **Length** octets sur la voie logique **Channel** à partir de la mémoire pointée par **Src**. Si **Flags** est non nul, soit **Length** en entier sera écrit ou aucun octet ne sera écrit. Le transmetteur sera validé automatiquement après écriture des octets dans le buffer.

PARAMETRES: **Channel** est la voie logique. **Dest** est l'adresse (buffer) à partir de laquelle il faut écrire. **Length** est le nombre d'octets à écrire. Si **Flags** est non nul, soit **Length** en entier sera écrit ou aucun octet ne sera écrit.

VALEUR DE RETOUR: Le nombre actuel d'octets écrit sur la voie.

- **uint aascPeek ( CHANNEL Channel , void \*pMatchee , uint size )**

Essaye de reconnaître le plus de données possible jusqu'à une taille **size**, pointées par **pMatchee** (sans caractère de terminaison nul).

PARAMETRES: **Channel** est la voie logique. **pMatchee** est l'adresse de la chaîne à reconnaître. **size** est le nombre d'octets à essayer de reconnaître.

VALEUR DE RETOUR: Le nombre d'octet actuellement reconnus.

- **uint aascScanTerm ( CHANNEL Channel , char Term )**

Cherche le caractère de terminaison **Term** dans le buffer de réception de la voie logique **Channel**. A noter que cette fonction ne lit aucun octet dans le buffer de réception. Le récepteur sera validé automatiquement si le contrôle de flux est validé et que le buffer de réception possède au moins 16 octets de libres.

VALEUR DE RETOUR: La taille du paquet de données terminé par **Term**.

- **void aascPipe ( CHANNEL Channel , CHANNEL Out , CHANNEL In )**

Crée un itinéraire (pipe) en détournant la sortie de **Channel** sur l'entrée de **Out**, et en détournant l'entrée de **Channel** à partir de **In**.

PARAMETRES: **Channel**, **Out** et **In** sont des voies logiques.

- **long aascGetError ( CHANNEL Channel )**

Obtient les conditions de l'erreur courante.

PARAMETRE: **Channel** est la voie logique

VALEUR DE RETOUR: Dépend du périphérique. Pour les valeurs de retour spécifiques, voir la description de la fonction du pilote de périphérique `<device_name>GetErr( )` (par exemple, `sio0GetErr( )`).

- **void aascClearError ( CHANNEL Channel )**

Efface la condition d'erreur.

PARAMETRE: **Channel** est la voie logique.

- **uint aascReadBufLeft ( CHANNEL Channel )**

Calcule le nombre d'octets restant à lire dans le buffer de réception de la voie logique **Channel**.

VALEUR DE RETOUR: Le nombre d'octets restant à lire.

- **uint aascWriteBufLeft ( CHANNEL Channel )**

Calcule le nombre d'octets restant à transmettre dans le buffer de transmission de la voie logique **Channel**.

VALEUR DE RETOUR: Le nombre d'octets restant à transmettre.

- **uint aascReadBufFree ( CHANNEL Channel )**

Calcule le nombre d'octets libres dans le buffer de réception de la voie logique **Channel**.

VALEUR DE RETOUR: Le nombre d'octets libres.

- **uint aascWriteBufFree ( CHANNEL Channel )**

Calcule le nombre d'octets libres dans le buffer de transmission de la voie logique **Channel**.

VALEUR DE RETOUR: Le nombre d'octets libres.

- **void aascFlush ( CHANNEL Channel )**

Purge les buffers associés à la voie logique **Channel**, détruisant ainsi les informations qui peuvent encore résider dans les buffers. Si la voie utilise le contrôle de flux CTS/RTS, le programmeur doit déterminer explicitement la revalidation de la voie récepteur en appelant **aascRxSwitch**. **aascRxSwitch** désactivera explicitement RTS pour permettre à l'autre coté de transmettre.

- **void aascFlushRdBuf ( CHANNEL Channel )**

Purge le buffer de lecture associé à la voie logique **Channel**, détruisant ainsi les informations qui peuvent encore résider dans les buffers. Si la voie utilise le contrôle de flux CTS/RTS, le programmeur doit déterminer explicitement la revalidation de la voie récepteur en appelant **aascRxSwitch**. **aascRxSwitch** désactivera explicitement RTS pour permettre à l'autre coté de transmettre.

- **void aascFlushWrBuf ( CHANNEL Channel )**

Purge le buffer d'écriture associé à la voie logique **Channel**. Toute les informations sont effacées dans le buffer.

- **void aascPrintf ( CHANNEL Chan , char \*fmt , ... )**

Imprime une chaîne formatée sur la voie **Chan**.

PARAMETRES: **Chan** est la voie sur laquelle envoyer. **fmt** est le format de la chaîne à imprimer. Les arguments (s'il y en a) doivent suivre **fmt**.

- **void aascVPrintf ( CHANNEL Chan , char \*fmt , void \*firstArg )**

Imprime une chaîne formatée sur la voie **Chan**.

PARAMETRES: **Chan** est la voie sur laquelle envoyer. **fmt** est la chaîne format. **firstArg** est un pointeur sur le premier argument.

## Fonctions XMODEM de AASC.LIB

Le protocole XModem effectue des transferts de fichiers (basé sur le transfert des données par "paquets") avec détection d'erreur CRC.

La structure d'un paquet pour un transfert XModem apparaît comme suit.

<u>Octets</u>	<u>Description</u>
1	Départ de l'entête (header)
1	Numéro de séquence du paquet
1	Complément à 1 du numéro de séquence du paquet
...	DATA ( 128 ou 1024 octets, binaire ou texte )
2	CRC - CCITT ( diviseur 0 x 1021 )

- **void aascXMRdInitPhy ( uint Where , uint Length , ulong XmemSrcAddr )**

Initialise l'emplacement et la taille de la mémoire physique pour le transfert des données PC vers cible de **aascReadXModem ( )**. Spécifie l'emplacement sur la cible et le nombre maximum d'octets à transférer du PC.

PARAMETRES: **Where** est l'emplacement de la mémoire racine (root) sur la cible, où les données en train d'être transférées sont placées. **Length** est le nombre maximum d'octets à transférer. **XmemSrcAddr** est la destination mémoire finale.

- **void aascXMRdInitLog ( uint Where , uint Length )**

Initialise l'emplacement et la taille de la mémoire logique pour le transfert des données PC vers cible de **aascReadXModem ( )**. Spécifie l'emplacement sur la cible et le nombre maximum d'octets à transférer du PC. Cette fonction par défaut demande à la fonction de relecture par défaut **aascRdCBackLocLg** d'avancer le pointeur du buffer d'une taille de paquet après la réception de chaque paquet.

PARAMETRES: **Where** est l'emplacement de la mémoire racine (root) sur la cible, où les données en train d'être transférées sont placées. **Length** est le nombre maximum d'octets à transférer.

- **uint aascReadXModem ( CHANNEL Channel , char \* ( \*read\_callback\_loc ) ( ) , void ( \*read\_callback\_mod ) ( ) , char Initialize )**

Effectue un chargement XModem du PC vers la cible. Appeler cette fonction une fois **Initialize** fixé à 1. Puis fixer **Initialize** à 0 et appeler cette fonction de façon répétitive jusqu'à obtenir une valeur non nulle en retour. Appeler **aascXMRdInitPhy ( )** pour des transferts mémoire physique ou **aascXMRdInitLog ( )** pour des transferts mémoire logique avant d'utiliser **aascReadXModem ( )**.

PARAMETRES: **Channel** est la voie en train d'être lue. **read\_callback\_loc** est un pointeur sur une fonction de rappel qui doit être appelé par cette fonction AVANT que chaque paquet XModem soit reçu. cette fonction détermine où le paquet est placé en mémoire en utilisant la fonction de rappel **aascRdBackLocLg ( )**. **read\_callback\_mod** est un pointeur sur une fonction de rappel qui doit être appelé par cette fonction APRES que chaque paquet XModem soit reçu. cette fonction effectue davantage de traitements sur les données. Une fonction défaut **aascRdCBackLocPh ( )** qui n'effectue pas de traitement est fournie. **Initialize** est le drapeau d'initialisation. Fixer **initialize** à 1 pour initialiser XModem lors du premier appel. Fixer **Initialize** à 0 pour tous les appels ultérieurs.

VALEUR DE RETOUR:

<b>XX_SUCCESS</b>	<b>XX_TIMEOUT</b>
<b>XX_COMMERR</b>	<b>XX_CANCEL</b>
<b>XX_SEQ</b>	<b>XX_CHKSUM</b>
<b>XX_NOSTART</b>	<b>XX_NOBEGPAK</b>
<b>XX_SYNC</b>	

- **uint aascRdCBackLocPh ( uint PackSize , char PackNum )**

Fonction fantôme appelée par **aascReadXModem ( )** après qu'un paquet soit reçu. Peut être remplacée par une fonction définie utilisateur si des modifications sont nécessaires sur un paquet.

PARAMETRES : **PackSize** est la taille du paquet en train d'être utilisé par XModem (128 ou 1024 octets). **PackNum** est le numéro du paquet courant.

VALEUR DE RETOUR: Adresse logique de la mémoire racine (root) où le paquet en provenance du PC sera placé avant le transfert en mémoire physique.

- **uint aascRdBackLocLg ( uint PackSize , char PackNum )**

Fonction de rappel par défaut pour l'adressage de blocs d'un transfert PC vers cible. Elle est appelée par **aascReadXModem ( )** avant de recevoir un paquet du PC. Cette fonction avance le pointeur par **PackSize** sur la mémoire cible après chaque envoi de paquet.

PARAMETRES : **PackSize** est la taille du paquet en train d'être utilisée par XModem: 0 pour utilisation de paquets XModem de 128 octets, 1 pour utilisation de paquets XModem de 1024 octets. **PackNum** est le numéro du paquet courant.

VALEUR DE RETOUR: L'adresse mémoire logique où le paquet en provenance du PC sera placé.

- **void aascXMWrInitPhy ( uint Where , uint Length , ulong XmemSrcAddr )**

Initialise l'emplacement et la taille de la mémoire physique à transférer sur le PC.

PARAMETRES: **Where** est l'adresse sur la cible où les données en train d'être transférées sont placées. **Length** est le nombre maximum d'octets à recevoir. **XmemSrcAddr** est la source mémoire physique des données à transférer.

- **void aascXMWrInitLog ( uint Where , uint Length )**

Initialise l'emplacement et la taille de la mémoire logique à transférer sur le PC.

PARAMETRES : **Where** est l'adresse sur la cible où les données en train d'être transférées sont placées. **Length** est le nombre maximum d'octets à recevoir.

- **uint aascWriteXModem ( CHANNEL Channel , char Pack1K , char Initialize , uint ( \*write\_callback ) ( ) )**

Effectue un chargement XModem de la cible vers le PC. Appeler cette fonction une fois **Initialize** fixé à 1. Puis fixer **Initialize** à 0 et appeler cette fonction de façon répétitive jusqu'à obtenir une valeur non nulle en retour. Appeler **aascXMWrInitPhy ( )** pour des transferts mémoire physique ou **aascXMWrInitLog ( )** pour des transferts mémoire logique avant le premier appel à **aascReadXModem ( )**.

PARAMETRES: **Channel** est la voie logique en train d'être écrite. **Pack1K** est la taille du paquet XModem: 0 pour l'utilisation de paquets XModem de 128 octets, 1 pour l'utilisation de paquets XModem de 1024 octets. **Initialize** est le drapeau d'initialisation. Fixer **Initialize** à 1 pour initialiser XModem lors du premier appel. Fixer **Initialize** à 0 pour les appels suivants. **Write\_callback** est un pointeur sur une fonction de rappel qui sera appelé par cette fonction AVANT chaque envoi de paquet XModem ainsi que des traitements ultérieurs sur les données. Les fonctions par défaut **aascWrCallBackLg ( )** et **aascWrCallBackPh ( )** sont fournies pour les transferts mémoire physique et logique. Pour de plus amples détails, consulter l'aide en ligne du logiciel.

VALEUR DE RETOUR:

**XX\_SUCCESS**  
**XX\_TIMEOUT**  
**XX\_COMMERR**  
**XX\_CANCEL**  
**XX\_NOSTART**  
**XX\_SYNC**

- **uint aascWrCallBackPh ( uint PackSize , char PackNum )**

Fonction de rappel par défaut pour l'adressage de données pour les transferts cible vers PC. **aascWrCallBackPh** est appelée par **aascWriteXModem ( )** avant d'envoyer un paquet au PC. **aascWrCallBackPh** avance le pointeur sur la mémoire cible de **PackSize** après l'envoi de chaque paquet. **aascWrCallBackPh** détermine l'adresse basée sur **PackSize** et **PackNum**.

PARAMETRES : **PackSize** est la taille du paquet en train d'être utilisée par XModem: 0 pour utilisation de paquets XModem de 128 octets, 1 pour utilisation de paquets 1024 octets. **PackNum** est le numéro du paquet courant.

VALEUR DE RETOUR : L'adresse du prochain emplacement où transférer les données; 0 si le numéro de paquet demandé dépasse la taille du fichier.

- **uint aascWrCallbackLg ( uint PackSize , char PackNum )**

Fonction de rappel par défaut pour l'adressage de données pour les transferts cible vers PC. **aascWrCallbackLg** est appelée par **aascWriteXModem ( )** avant d'envoyer un paquet au PC. **aascWrCallbackLg** avance le pointeur sur la mémoire cible de **PackSize** après l'envoi de chaque paquet. **aascWrCallbackLg** détermine l'adresse basée sur **PackSize** et **PackNum**.

PARAMETRES : **PackSize** est la taille du paquet en train d'être utilisée par XModem: 0 pour utilisation de paquets XModem de 128 octets, 1 pour utilisation de paquets 1024 octets. **PackNum** est le numéro du paquet courant.

VALEUR DE RETOUR : L'adresse du prochain emplacement où transférer les données; 0 si le numéro de paquet demandé dépasse la taille du fichier.

## 5 - AUTRES LIBRAIRIES

Les librairies décrites dans ce chapitre sont spécifiques à au moins un type de contrôleur.

### **5KEY.LIB**

Ces fonctions écran LCD et clavier supportent les contrôleurs des séries PK2100 et PK2200. Cette librairie est l'ancien *système five-key*. Elle utilise le noyau temps réel (RTK). L'écran standard LCD est 2 x 20. Pour faire fonctionner le système five-key en LCD 2 x 16, écrire **#define LCD16x2** en début de programme.

- **void \_5keysettime ( char \*time )**

Fixe l'heure de l'horloge temps réel sur la base de la chaîne **\*time**. Le format de la chaîne est "hh:mm:ss".

- **void \_5keysetdate ( char \*date )**

Fixe la date de l'horloge temps réel sur la base de la chaîne **\*date**. Le format de la chaîne est "mm-jj-aa".

- **void \_5keygettime ( char \*time )**

Récupère l'heure de l'horloge temps réel et la stocke dans **\*time**. Le format de la chaîne est "hh:mm:ss".

- **void \_5keygetdate ( char \*date )**

Récupère la date de l'horloge temps réel et la stocke dans **\*date**. Le format de la chaîne est "mm;jj:aa".

- **void lcd\_server ( char mode , long position , char \*lcd\_msg )**

Efface un nombre de lignes, spécifié par **mode**, et affiche le message **\*lcd\_msg** en position **position**. Voir **CPLC.LIB** pour la description des champs de position.

- **int \_5key\_float ( char \*label , float \*value , float max , float min , char \*help [ ] , byte size , byte modify , byte delay )**

C'est le "handler" du système five-key pour un paramètre flottant. Il modifie ou contrôle les paramètres suivants.

<b>label</b>	Chaîne pour le label de rubrique (item)
<b>value</b>	pointeur sur une variable <b>float</b>
<b>max , min</b>	Les limites de donnée
<b>help [ ]</b>	Un tableau de chaînes d'aide
<b>size</b>	Taille du tableau d'aide (le nombre de chaînes d'aide); utilise <b>sizeof (help)</b>
<b>modify</b>	Si 1, <b>value</b> est mis à jour; si 0, <b>value</b> est seulement surveillé
<b>delay</b>	Nombre de tops 25 mSec. du RTK après lesquels le logiciel se dégagera de la tâche five-key courante, libérant ainsi les autres tâches de plus basse priorité.

La fonction retourne un entier représentant une des touches suivantes: MENU, ITEM, ADD ou DELETE. Elle retourne -1 lorsque aucune touche n'a été appuyée.

- **int \_5key\_integer ( char \*label , int \*value , int max , int min , char \*help [ ] , byte size , byte modify , byte delay )**

C'est le "handler" du système five-key pour un paramètre entier. Il modifie ou contrôle les paramètres suivants.

<b>label</b>	Chaîne pour le label de rubrique (item)
<b>value</b>	pointeur sur une variable entière
<b>max , min</b>	Les limites de donnée
<b>help [ ]</b>	Un tableau de chaînes d'aide
<b>size</b>	Taille du tableau d'aide (le nombre de chaînes d'aide); utilise <b>sizeof (help)</b>
<b>modify</b>	Si 1, <b>value</b> est mis à jour; si 0, <b>value</b> est seulement surveillé
<b>delay</b>	Nombre de tops 25 mSec. du RTK après lesquels le logiciel se dégagera de la tâche five-key courante, libérant ainsi les autres tâches de plus basse priorité.

La fonction retourne un entier représentant une des touches suivantes: MENU, ITEM, ADD ou DELETE. Elle retourne -1 lorsque aucune touche n'a été appuyée.

- **int \_5key\_boolean ( char \*label , byte \*value , char \*help [ ] , byte size , byte modify , byte delay )**

C'est le "handler" du système five-key pour un paramètre booléen. Il modifie ou contrôle les paramètres suivants.

<b>label</b>	Chaîne pour le label de rubrique (item)
<b>value</b>	pointeur sur une variable booléenne
<b>help [ ]</b>	Un tableau de chaînes d'aide
<b>size</b>	Taille du tableau d'aide (le nombre de chaînes d'aide); utilise <b>sizeof (help)</b>
<b>modify</b>	Si 1, <b>value</b> est mis à jour; si 0, <b>value</b> est seulement surveillé
<b>delay</b>	Nombre de tops 25 mSec. du RTK après lesquels le logiciel se dégagera de la tâche five-key courante, libérant ainsi les autres tâches de plus basse priorité.

La fonction retourne un entier représentant une des touches suivantes: MENU, ITEM, ADD ou DELETE. Elle retourne -1 lorsque aucune touche n'a été appuyée.

- **int \_5key\_time ( char \*label , char \*string , char \*help [ ] , byte size , byte set\_clock , byte modify , byte delay )**

C'est le "handler" du système five-key pour un paramètre heure. Il modifie ou contrôle les paramètres suivants.

<b>label</b>	Chaîne pour le label de rubrique (item)
<b>string</b>	La chaîne heure
<b>help [ ]</b>	Un tableau de chaînes d'aide
<b>size</b>	Taille du tableau d'aide (le nombre de chaînes d'aide); utilise <b>sizeof (help)</b>
<b>set_clock</b>	Si non nul, fixe l'horloge temps réel
<b>modify</b>	Si 1, <b>value</b> est mis à jour; si 0, <b>value</b> est seulement surveillé
<b>delay</b>	Nombre de tops 25 mSec. du RTK après lesquels le logiciel se dégagera de la tâche five-key courante, libérant ainsi les autres tâches de plus basse priorité.

La fonction retourne un entier représentant une des touches suivantes: MENU, ITEM, ADD ou DELETE. Elle retourne -1 lorsque aucune touche n'a été appuyée.

- **int \_5key\_date ( char \*label , char \*string , char \*help [ ] , byte size , byte set\_clock , byte modify , byte delay )**

C'est le "handler" du système five-key pour un paramètre date. Il modifie ou contrôle les paramètres suivants.

<b>label</b>	Chaîne pour le label de rubrique (item)
<b>string</b>	La chaîne date
<b>help [ ]</b>	Un tableau de chaînes d'aide
<b>size</b>	Taille du tableau d'aide (le nombre de chaînes d'aide); utilise <b>sizeof (help)</b>
<b>set_clock</b>	Si non nul, fixe l'horloge temps réel
<b>modify</b>	Si 1, <b>value</b> est mis à jour; si 0, <b>value</b> est seulement surveillé
<b>delay</b>	Nombre de tops 25 mSec. du RTK après lesquels le logiciel se dégagera de la tâche five-key courante, libérant ainsi les autres tâches de plus basse priorité.

La fonction retourne un entier représentant une des touches suivantes: MENU, ITEM, ADD ou DELETE. Elle retourne -1 lorsque aucune touche n'a été appuyée.

- **void \_5key\_setmenu ( char \*menu , char \*item , byte mode , void \*ptr , float max , float min , char \*help [ ] , byte size , byte modify , byte delay , byte display )**

Ajoute une rubrique (item) à la liste five-key. Les rubriques avec le même label menu sont groupées ensembles. Les paramètres suivants sont utilisés:

<b>menu</b>	Le label menu (chaîne)
<b>item</b>	Le label rubrique (chaîne)
<b>mode</b>	Type de donnée en train d'être créée
<b>ptr</b>	Pointeur sur la donnée
<b>max , min</b>	Limites de la donnée
<b>help [ ]</b>	Tableau de chaînes d'aide
<b>size</b>	Taille du tableau d'aide (le nombre de chaînes d'aide); utilise <b>sizeof (help)</b>
<b>modify</b>	Détermine le traitement de la donnée. Pour modification ou pour surveillance.
<b>delay</b>	Nombre de tops 25 mSec. du RTK après lesquels le logiciel se dégagera de la tâche five-key courante, libérant ainsi les autres tâches de plus basse priorité.
<b>display</b>	1 si la rubrique doit être ajoutée à la liste des rubriques affichées périodiquement; 0 si la rubrique ne doit pas être ajoutée à la liste.

- **int \_5key\_init\_item ( \_5KEYITEM \*thisitem , char \*d\_menu , char \*d\_item , char data\_mode , void \*data\_ptr , float max\_data , float min\_data , char \*my\_help [ ] , char help\_line , char data\_modify , char delay )**

Est appelé par **\_5key\_setmenu** pour créer une rubrique five-key. Les paramètres suivants sont utilisés.

<b>thisitem</b>	Pointe sur une structure de rubrique five-key pour la liste des liens five-key
<b>d_menu</b>	Pointe sur un label menu

<b>d_item</b>	Pointe sur un label rubrique
<b>data_mode</b>	Egal 0 pour les flottants, 1 pour les entiers, 2 pour les Boléens ( <b>chars</b> ), 3 pour les chaînes heures et 4 pour les chaînes dates.

Les macros suivantes peuvent aussi être utilisées.

**\_5key\_Fdata**, **\_5key\_Idata**, **\_5keyBdata**, **\_5keyTdata** et **\_5key\_Ddata.data\_ptr** pointant sur la donnée.

**max\_data** et **min\_data** sont les limites hautes et basses des données

**my-help [ ]** est une liste de chaînes d'aide

**help-line** est deux fois le nombre actuel de chaînes d'aide

**data\_modify** est égal à 1 si la donnée doit être modifiée par le système five-key, sinon 0 si la donnée doit juste être suivie

**delay** est la période de suspend de la tâche five-key

**idisp** est égal à 1 si la donnée doit être affichée périodiquement quand il n'y a aucune activité sur le clavier ou l'écran, sinon 0.

La fonction ne retourne rien.

- **int \_5key\_server ( \_5KEYITEM \*t\_item )**

Sert à l'affichage LCD et pour action d'une rubrique five-key. La fonction retourne l'appui d'une des touches du menu

- **void \_5key\_menu ( )**

sert à la liste des liens créée avec **\_5key\_setmenu ( )**. Cette fonction doit être appelée à l'intérieur d'une tâche RTK.

- **void \_5key\_setalarm ( int ( \*func1 ) ( ) , int ( \*func2 ) ( ) , int ( \*func3 ) ( ) , int ( \*func4 ) ( ) )**

Prépare les fonctions de service pour les alarmes logicielles.

**func1 ( )**, la fonction de service pour **\_ALARM1**

**func2 ( )**, la fonction de service pour **\_ALARM2**

**func3 ( )**, la fonction de service pour **\_ALARM3** et

**func4 ( )**, la fonction de service pour **\_ALARM4**.

Toutes les fonctions prennent par défaut **NO\_FUNCTION**. Les fonctions de service peuvent être changées ou désactivées en fonctionnement sous réserve qu'il n'y ai pas de conflit avec l'exécution d'une fonction de service.

- **void \_5key\_setfunc ( int ( \*func1 ) ( ) , int ( \*func2 ) ( ) , int ( \*func3 ) ( ) , int ( \*func4 ) ( ) )**

Prépare les fonctions de service pour les touches de fonction.

**func1 ( )**, la fonction de service pour F1

**func2 ( )**, la fonction de service pour F2

**func3 ( )**, la fonction de service pour F3 et

**func4 ( )**, la fonction de service pour F4.

Toutes les fonctions prennent par défaut **NO\_FUNCTION**. Les fonctions de service peuvent être changées ou désactivées en fonctionnement sous réserve qu'il n'y ai pas de conflit avec l'exécution d'une fonction de service.

- **void \_5key\_setmsg ( byte message\_no , char \*the-message )**

Fixe une des 10 chaînes de messages pour un affichage périodique.

**message\_no** Le numéro de message, de 0 à 9.

**the\_message** La chaîne message.

Tous les messages prennent une valeur nulle par défaut.

## 5KEYEXTD.LIB

Ces fonctions clavier supportent les contrôleurs des séries PK 2100 et PK 2200. Elles utilisent le noyau temps réel (Real Time Kernel - RTK).

- **int \_5key\_12out ( )**

C'est le serveur five-key pour les dix sorties numériques "virtuelles" et les deux sorties relais "virtuelles". Les états des sorties relais et numériques peuvent être modifiés par le système five-key. Si un état de sortie change, cette fonction rafraîchira l'affichage pour refléter le changement.

La fonction retourne un entier représentant une des touches suivantes: MENU, ITEM, ADD ou DELETE. Elle retourne -1 lorsque aucune touche n'a été enfoncée.

- **int \_5key\_dacout ( )**

C'est le serveur five-key pour la sortie analogique DAC "virtuelle". Si la valeur de sortie change, cette fonction rafraîchira l'écran pour refléter le changement.

La fonction retourne un entier représentant une des touches suivantes: MENU, ITEM, ADD ou DELETE. Elle retourne -1 lorsque aucune touche n'a été enfoncée.

- **int \_5key\_uinput ( )**

C'est le serveur five-key pour les six entrées universelles "virtuelles". Si un état d'entrée change, cette fonction rafraîchira l'affichage pour refléter le changement.

La fonction retourne un entier représentant une des touches suivantes: MENU, ITEM, ADD ou DELETE. Elle retourne -1 lorsque aucune touche n'a été enfoncée.

- **int \_5key\_diginput ( )**

C'est le serveur five-key pour les sept entrées numériques "virtuelles". Si un état d'entrée change, cette fonction rafraîchira l'écran pour refléter le changement.

La fonction retourne un entier représentant une des touches suivantes: MENU, ITEM, ADD ou DELETE. Elle retourne -1 lorsque aucune touche n'a été enfoncée.

- **int \_5key\_bank1dig ( )**

C'est le serveur five-key pour les entrées numériques "virtuelles" 1 à 8. Si un état d'entrée change, cette fonction rafraîchira l'écran pour refléter le changement.

La fonction retourne un entier représentant une des touches suivantes: MENU, ITEM, ADD ou DELETE. Elle retourne -1 lorsque aucune touche n'a été enfoncée.

- **int \_5key\_bank2dig ( )**

C'est le serveur five-key pour les entrées numériques "virtuelles" 9 à 16. Si un état d'entrée change, cette fonction rafraîchira l'écran pour refléter le changement.

La fonction retourne un entier représentant une des touches suivantes: MENU, ITEM, ADD ou DELETE. Elle retourne -1 lorsque aucune touche n'a été enfoncée.

- **int \_5key\_14out ( )**

C'est le serveur five-key pour les 14 sorties numériques "virtuelles". Les états de sortie logiques peuvent être modifiés à l'aide du système five-key. Si un état de sortie change, cette fonction rafraîchira l'affichage pour refléter le changement.

La fonction retourne un entier représentant une des touches suivantes: MENU, ITEM, ADD ou DELETE. Elle retourne -1 lorsque aucune touche n'a été enfoncée.

## CPLC.LIB

Ces fonctions supportent les contrôleurs des séries PK 2100 et PK 2200.

- **void uplc\_init ( )**

Initialise les pilotes (drivers) et les variables pour les services ou périphériques suivants:

- Routine d'interruption pour background timer 1
- LCD, si sélectionné
- Clavier, si sélectionné (le clavier est scruté toutes les 25 millisecondes)
- Pilotes virtuels, timers virtuels et chiens de garde virtuels, si sélectionnés

La routine d'interruption du timer 1 dessert aussi le timer chien de garde.

- **void lc\_kxinit ( )**

Initialise le pilote clavier et variables associées, ainsi que les variables de chien de garde virtuel.

- **void up\_beepvol ( int vol )**

Fixe le niveau sonore du beeper: **vol** = 1 pour niveau faible; 2 pour niveau fort.

- **void lc\_loadtab ( int \*tab , int tab\_size )**

Charge les tables **tab** pour correspondre à l'écran LCD.

- **void lc\_settab ( char flag )**

Fixe la variable tab **lc\_usetab**.

- **int lc\_kxget ( char mode )**

Va chercher une valeur de touche sur le buffer FIFO clavier. Si **mode** = 0, la valeur est retirée du buffer, sinon la valeur reste dans le buffer.

La fonction retourne la valeur de touche, ou -1 si aucune touche n'est frappée.

- **void lc\_setbeep ( int delay )**

Fixe la durée du beeper pour **delay** comptes de cycles de 1280 Hz.

- **void up\_beep ( uint k )**

Active le beeper pendant **k** millisecondes.

- **uint up\_lastkey ( )**

Retourne le temps écoulé depuis la dernière frappe d'une touche, en unités de 1/40 secondes. La fonction retourne le temps écoulé.

- **void lc\_init\_keypad ( )**

Initialise **timer1**, pilote clavier et variables, ainsi que le noyau temps réel.

- **void GLOBAL\_INIT ( )**

Se référer à **VDRIVER.LIB** pour la description de cette fonction.

- **int up\_synctimer ( )**

Synchronise le **SEC\_TIMER** virtuel avec l'horloge temps réel (RTC). La fonction retourne 0 si l'horloge temps réel est lue correctement, sinon elle retourne -1.

## DRIVERS.LIB

Pilotes matériels divers.

- **int pleport ( int bit )**

Vérifie le bit spécifié du port PLCBus. La fonction retourne 1 si le bit spécifié est à l'état 1, sinon 0.

- **void set16adr ( int address )**

Fixe l'adresse courante pour le PLCBus. Toutes les opérations de lecture et d'écriture accéderont à cette adresse tant qu'une nouvelle adresse n'a pas été fixée. **address** est une adresse physique 16-bits (pour bus 4-bits). Le groupe de rang le plus élevé contient la valeur du registre d'extension tandis que les groupes restant forment une adresse 12-bits (le premier et le troisième groupe doivent être échangés).

- **void set12adr ( int address )**

Fixe l'adresse courante pour le PLCBus. Toutes les opérations de lecture et d'écriture accéderont à cette adresse tant qu'une nouvelle adresse n'a pas été fixée. **address** est une adresse physique 12-bits (pour bus 4-bits) avec le premier et le troisième groupe inversés (le groupe le plus significatif se trouve dans les 4 bits bas).

- **void set4adr ( int address )**

Fixe l'adresse courante pour le PLCBus. Toutes les opérations de lecture et d'écriture accéderont à cette adresse tant qu'une nouvelle adresse n'a pas été fixée. **address** contient les derniers 4 bits de l'adresse physique (pour bus 4-bits) dans les bits 8 à 11. Une adresse 12-bits peut être passée à cette fonction, mais les 4 derniers bits seulement seront mis. Cette fonction doit seulement être appelée si les 8 premiers bits de l'adresse sont les mêmes que ceux obtenus lors de l'appel précédent à **set12adr**.

- **char read4data ( int address )**

Fixe les 4 derniers bits de l'adresse PLCBus courante en utilisant **address** (bits 8 à 11). Puis lit 4 bits de données hors du bus à l'aide d'un cycle **BUSRD0**. La fonction retourne la donnée PLCBus dans les 4 bits de poids faible (les bits de poids fort sont indéfinis).

- **char read12data ( int address )**

Fixe l'adresse PLCBus courante en utilisant l'adresse 12-bits **address**. Puis lit 4 bits de données hors du bus à l'aide d'un cycle **BUSRD0**. La fonction retourne la donnée PLCBus dans les 4 bits de poids faible (les bits de poids fort sont indéfinis).

- **void write4data ( int address , char data )**

Fixe les 4 derniers bits de l'adresse PLCBus courante en utilisant **address** (bits 8 à 11). Puis écrit les 4 bits de **data** de poids faible sur le bus.

- **void write12data ( int address , char data )**

Fixe l'adresse PLCBus courante en utilisant l'adresse 12-bits **address**. Puis écrit les 4 bits de **data** de poids faible sur le bus.

- **void hv\_wr ( char v )**

Écrit 8 bits sur le circuit pilote haute-tension. Chaque bit affecte une sortie haute-tension. Un 1 valide la sortie correspondante; 0 désactive la sortie.

- **void hv\_enb ( )**

Valide le circuit pilote haute-tension.

- **void hv\_dis ( )**

Désactive le circuit pilote haute-tension.

- **void lcd\_init ( char mode )**

Initialise le LCD; **mode** doit normalement être fixé à 0x18.

- **void lputc ( char cc )**

Envoi un caractère au LCD et rafraîchit le curseur; **cc** est le caractère à envoyer: si le bit haut est mit, il sera traité comme un caractère de contrôle. Les caractères de contrôle possibles sont les suivants:

<code>\n</code>	Nouvelle ligne (curseur position ligne 1, colonne 0)
<code>\xFF</code>	Efface l'écran
<code>\xF0</code>	Efface ligne 0
<code>\xF1</code>	Efface ligne 1
<code>\xF2</code>	Curseur OFF (curseur invisible, clignotement desactivé)
<code>\xF3</code>	Curseur ON (curseur type bloc plein)
<code>\xF4</code>	Curseur clignotant (clignote en permanence)
<code>\xF5</code>	Décale l'affichage vers la gauche
<code>\xF6</code>	Décale l'affichage vers la droite
<code>\x80-\xA7</code>	Positionne le curseur en ligne 0
<code>\xC0-\xE7</code>	Positionne le curseur en ligne 1

- **void lcd\_clr\_line ( char code )**

Efface une ligne sur le LCD; **code** doit être 0x80 pour effacer la ligne 0 et 0xC0 pour effacer la ligne 1.

- **void lcd\_wait ( )**

Attend jusqu'au moment ou le LCD puisse accepter des données.

- **int lprintf ( char \*fmt, ... )**

Fonctionne comme **printf**, mais est réservé au LCD.

- **char lputs ( char \*p )**

Envoie sur le LCD la chaîne **\*p** terminée par un caractère nul, puis met à jour le curseur (sans recouvrement). Tous les caractères (exceptés les nuls) sont directement envoyés sur le LCD; les caractères de contrôle ne sont pas reconnus. La fonction retourne un pointeur sur la chaîne.

- **void\* intoff ( void\* ptr )**

Sauve l'état d'interruption courante dans **\*ptr** puis désactive les interruptions. La fonction retourne le pointeur **ptr**.

- **void\* inton ( void\* ptr )**

Valide les interruptions si elles étaient précédemment "on", conformément à **\*ptr**. **ptr** doit avoir été précédemment fixé par un appel à **intoff**. La fonction retourne le pointeur **ptr**.

- **void doint ( )**

Valide les interruptions pour un court laps de temps puis les désactive (si elles étaient précédemment "off"). Ceci permet aux interruptions d'être traitées dans le code car elles sont sinon désactivées.

- **int tm\_rd ( struct tm \*t )**

Lit l'heure système courante dans la structure **t**. Cette routine fonctionne avec les horloges Toshiba et Epsom.

La structure suivante est utilisée pour gérer l'heure et la date:

```
struct tm {
    char tm_sec;           // 0 à 59
    char tm_min;           // 0 à 59
    char tm_hour;          // 0 à 23
    char tm_mday;          // 1 à 31
    char tm_mon;           // 1 à 12
    char tm_year;          // 00 à 150 (1900 à 2050)
    char tm_wday;          // 0 à 6, 0 étant le Dimanche
};
```

La fonction retourne 0 quand elle a fonctionné correctement, et -1 si l'horloge ne fonctionne pas ou n'est pas installée.

- **int tm\_wr ( struct tm \*t )**

Fixe l'heure système conformément à la structure **t**. Cette routine fonctionne avec les horloges Toshiba et Epsom. La structure suivante est utilisée pour gérer l'heure et la date:

```

struct tm {
    char tm_sec;           // 0 à 59
    char tm_min;           // 0 à 59
    char tm_hour;         // 0 à 23
    char tm_mday;         // 1 à 31
    char tm_mon;          // 1 à 12
    char tm_year;         // 00 à 150 (1900 à 2050)
    char tm_wday;         // 0 à 6, 0 étant le Dimanche
};

```

La fonction retourne 0 quand elle a fonctionné correctement, et -1 si l'horloge ne fonctionne pas ou n'est pas installée.

- **void mktime ( struct tm \*timeptr, long time )**

Renseigne la structure pointée par **timeptr** conformément à l'heure, spécifiée en secondes depuis le 1er Janvier 1980.

- **long mktime ( struct tm \*timeptr )**

Convertit le contenu de **timeptr** en un entier long. La fonction retourne le temps en secondes depuis le 1er Janvier 1980.

- **long clock ( )**

Lit l'horloge système et convertit l'heure en un entier long. La fonction retourne le temps en secondes depuis le 1er Janvier 1980.

- **long phy\_adr ( char \*adr )**

Convertit une adresse logique (16-bits) en adresse physique (20-bits). **adr** pointe sur l'adresse. La fonction retourne une adresse 20-bits sous forme d'un entier long.

- **void dmacopy ( long dest, long src, uint count )**

Utilise le DMA pour copier **count** octets d'une adresse physique (**src**) à une autre (**dest**).

- **void outportn ( int port, char \*buf, char count )**

Ecrit **count** octets sur le port de sortie spécifié. **buf** pointe sur la séquence d'octets à écrire.

- **void init\_timer0 ( uint count )**

Initialise le timer 0. **count** est la valeur placée dans le registre de rechargement (reload). Quelques valeurs de comptes courantes et les fréquences qu'elles génèrent sont indiquées ci-dessous pour une horloge 9,216 MHz.

9126	50 Hz	7680	60 Hz	7200	64 Hz
4608	100 Hz	2304	200 Hz	1152	400 Hz
900	512 Hz	600	768 Hz	500	928 Hz
450	1024 Hz				

- **void timer0\_isr ( )**

Routine de service d'interruption **timer 0**, cadence le noyau temps réel.

- **void setbeep ( int delay )**

Paramètre un bip temporisé. **delay** spécifie la longueur du bip en nombre de tops **timer1**. L'interruption **timer1** effectue le bip en tâche de fond, cette fonction revient donc immédiatement.

- **void init\_timer1 ( uint count )**

Initialise **timer1**. **count** est la valeur placée dans le registre de rechargement (reload). Quelques valeurs de comptes courantes et les fréquences qu'elles génèrent sont indiquées ci-après pour une horloge 9,216 MHz.

9126	50 Hz	7680	60 Hz	7200	64 Hz
4608	100 Hz	2304	200 Hz	1152	400 Hz
900	512 Hz	600	768 Hz	500	928 Hz
450	1024 Hz				

- **void tdelay ( int msec )**

Attend pendant **msec** millisecondes, **timer1** étant supposé fonctionner à 750 Hz. Le délai courant est décrit à la fréquence de **timer1** par la formule  $\text{delay} = 3 \times (\text{msec} / 4) / \text{freq1}$ .

- **void int\_timer1 ( )**

Routine de service interruption **timer1**. Pilote le bipeur et le clavier. Cadence aussi le noyau temps réel si **RUNKERNEL** est défini.

- **void save\_shadow ( )**

Sauve les registres trace (shadow) du PLCBus sur la pile.

- **void restore\_shadow ( )**

Restitue les registres trace (shadow) du PLCBus sur la pile, et réinitialise l'adresse bus courante.

- **void write24data ( long address, char data )**

Fixe l'adresse PLCBus courante à l'aide de l'adresse 24-bits **address**, puis écrit 8 bits de données **data** sur le bus.

- **void write8data ( long address, char data )**

Fixe les derniers 8 bits de l'adresse PLCBus courante à l'aide de l'adresse **address** (bits 16 à 23), puis écrit 8 bits de données **data** sur le bus.

- **int read24data0 ( long address )**

Fixe l'adresse PLCBus courante avec l'adresse 24-bits **address**, puis lit 8 bits de données hors du bus à l'aide d'un cycle **BUSRD0**. La fonction retourne une donnée PLCBus sur les 8 bits de poids faible (les bits de poids fort étant 0).

- **int read8data0 ( long address )**

Fixe les derniers 8 bits de l'adresse PLCBus courante avec l'adresse **address** (bits 16 à 23), puis lit 8 bits de données du bus à l'aide d'un cycle **BUSRD0**. La fonction retourne une donnée PLCBus sur les 8 bits de poids faible (les bits de poids fort étant 0).

- **int read24data1 ( long address )**

Fixe l'adresse PLCBus courante avec l'adresse 24-bits **address**, puis lit 8 bits de données du bus à l'aide d'un cycle **BUSRD1**. La fonction retourne une donnée PLCBus sur les 8 bits de poids faible (les bits de poids fort étant 0).

- **int read8data1 ( long address )**

Fixe les derniers 8 bits de l'adresse PLCBus courante avec l'adresse **address** (bits 16 à 23), puis lit 8 bits de données du bus à l'aide d'un cycle **BUSRD1**. La fonction retourne une donnée PLCBus sur les 8 bits de poids faible (les bits de poids fort étant 0).

- **void set24adr ( long address )**

Fixe l'adresse courante pour le PLCBus. Tous les processus de lecture et d'écriture se feront à cette adresse tant qu'une nouvelle adresse n'est pas fixée. **address** est une adresse physique 24 bits (pour le bus 8-bits), avec le premier et le troisième octet inversé (l'octet de poids faible étant le plus significatif).

- **void set8adr ( long address )**

Fixe l'adresse courante pour le PLCBus. Tous les processus de lecture et d'écriture se feront à cette adresse tant qu'une nouvelle adresse n'est pas fixée. **address** contient les 8 derniers bits de l'adresse physique (pour le bus 8-bits) dans les bits 16 à 23. Une adresse 24-bits doit être passée à cette fonction, mais les 8 derniers bits seulement seront fixés. Cette fonction doit être appelée seulement si les 16 premiers bits de l'adresse sont identiques dans l'adresse précédemment fixée par **set24adr**.

- **void plcbus\_isr ( )**

Cette fonction est utilisée pour servir toute les interruptions de la ligne /AT du PLCBus. La ligne /AT est connectée sur INT1 du Z180. Chaque routine de service interruption (ISR) est responsable pour assurer l'émission du signal /AT de ses périphériques une fois que l'ISR a été effectuée.

- **void relocate\_int1 ( )**

Reprogramme le vecteur INT1.

- **int DelayTicks ( CoData \*pfb, uint ticks )**

Procure le mécanisme de topage du temps pour les costatements. Les tops arrivent 1280 fois par seconde (la période est de 781,25 µSec). La fonction retourne 1 si le délai du top est écoulé. Sinon elle retourne 0.

- **int DelayMs ( CoData \*pfb, long delayms )**

Procure le mécanisme temps-milliseconde pour les costatements. La fonction retourne 1 si le délai spécifié en millisecondes est écoulé. Sinon elle retourne 0.

- **int DelaySec ( FuncBlk \*pfb, long delaysec )**

Procure le mécanisme temps-seconde pour les costatements. La fonction retourne 1 si le délai spécifié en secondes est écoulé. Sinon elle retourne 0.

- **int eei\_rd ( int address )**

Lit deux zones d'octet consécutives en EEPROM pour une donnée type entier. L'octet de poids faible est lu sur **address** et l'octet de poids fort sur **address + 1**. La fonction retourne l'entier à l'adresse **address** de l'EEPROM.

- **int eei\_wr ( int address , uint value )**

Ecrit une valeur entière à l'adresse **address** de l'EEPROM. L'octet de poids faible est en **address** et l'octet de poids fort en **address + 1**. La fonction retourne 0 si l'écriture s'est déroulée correctement.

- **void DMA0 ( uint cnt )**

Charge **cnt** dans le compteur de **DMA0** pour réaliser un comptage rapide (cablé). Le compte maximum est 64000. **\_DMAFLAG0** est fixé à 0. Si le DMA est arrivé en bout de compte, une routine service d'interruption DMA0 générera une interruption avec laquelle **\_DMAFLAG0** sera fixé à 1. Les signaux sont pris en compte sur front. C1A et C1B doivent tous deux être bas pour que /DREQ0 génère une interruption.

- **void DMA1 ( uint cnt )**

Charge **cnt** dans le compteur de **DMA1** pour réaliser un comptage rapide (cablé). Le compte maximum est 64000. **\_DMAFLAG1** est fixé à 0. Si le DMA est arrivé en bout de compte, une routine service d'interruption DMA1 générera une interruption avec laquelle **\_DMAFLAG1** sera fixé à 1. Les signaux sont pris en compte sur front. C2A et C2B doivent tous deux être bas pour que /DREQ1 génère une interruption. C2B utilise un des récepteurs RS485 pour entrée différentielle. Par exemple, connecter C2B- au 5 Volts; quand le signal sur C2B+ est plus petit que 5 Volts, un front négatif est généré pour le compteur DMA.

- **uint DMASnapShot ( char channel , uint \*count )**

Prend une "photo" d'un canal DMA **channel** (0 ou 1) pour obtenir le nombre d'impulsions comptées. La fonction retourne 0 si le train d'impulsions est trop rapide et ne laisse pas le temps de relever le comptage; ou 1 si le comptage a pu être obtenu et transféré dans **\*count**.

- **void powerdown ( )**

Coupe l'alimentation. L'alimentation ne peut être rétablie qu'en externe. Cette commande ne fonctionne qu'avec les contrôleurs muni d'une alimentation à découpage (excepté PK2200).

- **void powerup ( )**

Inverse les effets de **powerdown**, la validation d'alimentation étant conservée après coupure de l'alimentation externe. Voir **powerdown**.

- **void nmiint ( )**

Fixe par défaut le gestionnaire d'interruptions de coupure alimentation. La fonction ne fait rien et *ne retourne pas*.

- **void setperiodic ( int period )**

Règle un timer pour alimenter périodiquement le BL1100. Après l'appel à cette fonction, la carte peut être mise en mode sommeil et sera automatiquement "réveillée" aux intervalles spécifiés. L'exécution commencera dans la fonction principale quand l'alimentation sera restaurée. **period** peut être égal à 4 (pour lancer une fois par seconde), à 8 (pour lancer une fois par minute), ou à 12 (pour lancer une fois par heure). Fonctionne seulement avec les cartes équipées d'une alimentation à découpage, excepté pour le PK2200.

- **void sleep ( )**

Met le contrôleur en mode sommeil. Fonctionne sur toutes les cartes équipées d'une alimentation à découpage, excepté le PK2200. *La fonction ne retourne pas.*

- **void init\_timer ( )**

Initialise l'horloge système.

## DMA.LIB

Ces fonctions supportent le DMA utilisé sur tous les contrôleurs Z-World.

- **void DMA0Count ( uint count )**

Charge le compte **count** dans le compteur du DMA0 pour du comptage rapide câblé. Le compte maximum est de 64000. La fonction fixe le drapeau **\_DMAFLAG0** à 0. DMA0 provoque une interruption quand **count** fronts négatifs ont été détectés. La routine de service interruption pour DMA0 fixera **\_DMAFLAG0** à 1. Un programme utilisateur peut surveiller **\_DMAFLAG0** pour déterminer si **count** est fini ou pas.

- **void DMA1Count ( uint count )**

Charge le compte **count** dans le compteur du DMA1 pour du comptage rapide câblé. Le compte maximum est de 64000. La fonction fixe le drapeau **\_DMAFLAG1** à 0. DMA1 provoque une interruption quand **count** fronts négatifs ont été détectés. La routine de service interruption pour DMA1 fixera **\_DMAFLAG1** à 1. Un programme utilisateur peut surveiller **\_DMAFLAG1** pour déterminer si **count** est fini ou pas.

- **uint DMASnapShot ( byte channel , uint \*count )**

Lit le nombre d'impulsions qu'un canal DMA ( **channel** = 0 ou 1 ) a compté. Un compteur DMA est initialisé avec soit **DMA0Count** ou **DMA1Count**. La fonction retourne 0 si un canal DMA compte trop rapidement pour permettre une lecture stable de la valeur **count**. Si la fonction lit une valeur **count** stable, elle retourne 1 et fixe le paramètre **\*count**. A noter qu'une interruption DMA surviendra quand le canal DMA aura fini de compter, même si la valeur **count** ne peut pas être lu.

- **void DMA0\_Off ( )**

- **void DMA1\_Off ( )**

Désactive les canaux DMA.

- **uint DMA0\_SerialInit ( byte channel , byte mode , byte baud )**

Initialise le port série **channel** (devant être 0 ou 1) du Z180 pour des transferts série de DMA0. Le terme **mode** est défini comme suit:

bit0 = 0 pour 1 bit de stop	1 pour 2 bits de stop
bit1 = 0 pour pas de parité	1 pour une parité
bit2 = 0 pour 7 bits de données	1 pour 8 bits de données
bit3 = 0 pour une parité paire	1 pour une parité impaire

Le terme **baud** est la vitesse en baud par multiple de 1200 baud ( par exemple 8 pour 9600 baud ).

- **uint DMA0\_Rx ( byte port , ulong address , uint count , int interrupts , int increments )**

Initialise un transfert utilisant DMA0 pour recevoir **count** octets d'un port série (**port** = 0 ou 1) à l'emplacement mémoire absolu commençant à **address**. L'adresse mémoire logique pour des tableaux ordinaires doit être convertie en adresse physique à l'aide de **phy\_adr ( array )**. Passer simplement directement le nom du tableau pour les tableaux **xdata**. DMA0 génèrera une interruption à la fin du transfert si **interrupts** est égal à 1. Le programme utilisateur doit procurer la routine de service interruption. DMA0 ne génère pas d'interruption si **interrupts** est égal à 0. Le terme **increments** doit être égal à 0 pour incrémenter l'adresse mémoire, et 1 pour la décrémenter. La fonction retourne 1 en cas de succès, 0 si DMA0 est occupé, -1 si le port série est occupé, et -2 si **channel** n'est pas 0 ou 1.

- **uint DMA0\_Tx ( byte port , ulong address , uint count , int interrupts , int increments )**

Initialise un transfert utilisant DMA0 pour transmettre **count** octets d'un port série (**port** = 0 ou 1) à l'emplacement mémoire absolu commençant à **address**. L'adresse mémoire logique pour des tableaux ordinaires doit être convertie en adresse physique à l'aide de **phy\_adr ( array )**. Passer simplement directement le nom du tableau pour les tableaux **xdata**. DMA0 génèrera une interruption à la fin du transfert si **interrupts** est égal à 1. Le programme utilisateur doit procurer la routine de service interruption. DMA0 ne génère pas d'interruption si **interrupts** est égal à 0. Le terme **increments** doit être égal à 0 pour incrémenter l'adresse mémoire, et 1 pour la décrémenter. La fonction retourne 1 en cas de succès, 0 si DMA0 est occupé, -1 si le port série est occupé, et -2 si **channel** n'est pas 0 ou 1.

- **uint DMA0\_MM ( ulong dst , ulong src , uint count , int mode , int interrupts )**

Initialise un transfert utilisant DMA0 pour copier **count** octets de l'emplacement mémoire absolu commençant à **dst**. L'adresse mémoire logique pour des tableaux ordinaires doit être convertie en adresse physique à l'aide de **phy\_adr ( array )**. Passer simplement directement le nom du tableau pour les tableaux **xdata**. DMA0 génèrera une interruption à la fin du transfert si **interrupts** est égal à 1. Le programme utilisateur doit procurer la routine de service interruption. DMA0 ne génère pas d'interruption si **interrupts** est égal à 0. Le terme **mode** doit être égal à 0 pour des transferts en vol-de-cycle et 1 pour des transferts en mode burst. La fonction retourne 1 en cas de succès ou 0 si DMA0 est occupé.

- **uint DMA0\_MIO ( uint ioaddr , ulong memaddr , uint count , int interrupts , int increments )**

Initialise un transfert utilisant DMA0 pour écrire **count** octets de l'emplacement mémoire absolu commençant à **memaddr** au port d'E/S désigné par **ioaddr**. Le périphérique externe doit générer des fronts descendants d'impulsions /DREQ0 pour chaque octet transféré. L'adresse mémoire logique pour des tableaux ordinaires doit être convertie en adresse physique à l'aide de **phy\_adr ( array )**. Passer simplement directement le nom du tableau pour les tableaux **xdata**. DMA0 génèrera une interruption à la fin du transfert si **interrupts** est égal à 1. Le programme utilisateur doit procurer la routine de service interruption. DMA0 ne génère pas d'interruption si **interrupts** est égal à 0. Le terme **increments** doit être égal à 0 pour incrémenter l'adresse mémoire, et 1 pour la décrémenter. La fonction retourne 1 en cas de succès, 0 si DMA0 est occupé.

- **uint DMA0\_IOM ( ulong memaddr , uint ioaddr , uint count , int interrupts , int increments )**

Initialise un transfert utilisant DMA0 pour lire **count** octets du port d'E/S désigné par **ioaddr** à l'emplacement mémoire absolu commençant à **memaddr**. Le périphérique externe doit générer des fronts descendants d'impulsions /DREQ0 pour chaque octet transféré. L'adresse mémoire logique pour des tableaux ordinaires doit être convertie en adresse physique à l'aide de **phy\_adr ( array )**. Passer simplement directement le nom du tableau pour les tableaux **xdata**. DMA0 génèrera une interruption à la fin du transfert si **interrupts** est égal à 1. Le programme utilisateur doit procurer la routine de service interruption. DMA0 ne génère pas d'interruption si **interrupts** est égal à 0. Le terme **increments** doit être égal à 0 pour incrémenter l'adresse mémoire, et 1 pour la décrémenter. La fonction retourne 1 en cas de succès, 0 si DMA0 est occupé.

- **uint DMA1\_MIO ( uint ioaddr , ulong memaddr , uint count , int interrupts , int increments )**

Initialise un transfert utilisant DMA1 pour écrire **count** octets de l'emplacement mémoire absolu commençant à **memaddr** au port d'E/S désigné par **ioaddr**. Le périphérique externe doit générer des fronts descendants d'impulsions /DREQ1 pour chaque octet transféré. L'adresse mémoire logique pour des tableaux ordinaires doit être convertie en adresse physique à l'aide de **phy\_adr ( array )**. Passer simplement directement le nom du tableau pour les tableaux **xdata**. DMA1 génèrera une interruption à la fin du transfert si **interrupts** est égal à 1. Le programme utilisateur doit procurer la routine de service interruption. DMA1 ne génère pas d'interruption si **interrupts** est égal à 0. Le terme **increments** doit être égal à 0 pour incrémenter l'adresse mémoire, et 1 pour la décrémenter. La fonction retourne 1 en cas de succès, 0 si DMA1 est occupé.

- **uint DMA1\_IOM ( ulong memaddr , uint ioaddr , uint count , int interrupts , int increments )**

Initialise un transfert utilisant DMA1 pour lire **count** octets du port d'E/S désigné par **ioaddr** à l'emplacement mémoire absolu commençant à **memaddr**. Le périphérique externe doit générer des fronts descendants d'impulsions /DREQ1 pour chaque octet transféré. L'adresse mémoire logique pour des tableaux ordinaires doit être convertie en adresse physique à l'aide de **phy\_addr ( array )**. Passer simplement directement le nom du tableau pour les tableaux **xdata**. DMA1 génèrera une interruption à la fin du transfert si **interrupts** est égal à 1. Le programme utilisateur doit procurer la routine de service interruption. DMA1 ne génère pas d'interruption si **interrupts** est égal à 0. Le terme **increments** doit être égal à 0 pour incrémenter l'adresse mémoire, et 1 pour la décrémenter. La fonction retourne 1 en cas de succès, 0 si DMA1 est occupé.

## FK.LIB

Ce sont les fonctions support pour utiliser le clavier et l'écran LCD sans le noyau temps réel (RTK).

- **int fk\_helpmsg ( char \*\*hptr )**

Affiche une série de messages d'aide quand la touche HELP est enfoncée. L'affichage courant est sauvé et chaque chaîne message est affichée pendant 1,8 secondes, l'affichage initial est ensuite restitué. L'entrée doit être un tableau de chaînes déclaré comme suit:

```
char *hptr [ ] = { "Str 1", "Str 2", ..., "Str n", "" } ;
```

La dernière chaîne doit être *vide*. La fonction retourne une valeur non nulle si l'aide est inactive, et 0 si elle est active.

- **void fk\_monitorkeypad ( )**

Surveille le clavier pour l'appui d'une touche. Cette fonction doit être appelée à partir d'une tâche haute priorité du RTK ou du SRTK. Elle fixe la variable globale **fk\_tkey** sur une valeur de 1 à 12 en fonction de la touche activée. La valeur est 0 si aucune touche n'a été appuyée. La fonction surveille aussi la combinaison des deux touches activant le reset. Si la combinaison de reset est détectée, la fonction ne retournera pas mais forcera le chien de garde à partir en time-out. Il n'y a pas de buffer. Le traitement des touches activées doit se faire dans une fenêtre de 100 mSec. sinon elles sont perdues.

- **int fk\_item\_alpha ( char \*s1, char \*var, int wdsz )**

Modifie une chaîne en utilisant le système five-key. Le terme **\*s1** est une chaîne contenant une invite (prompt). Le terme **\*var** est la chaîne à afficher et/ou modifier. La fonction retourne 0 si l'affichage/modification n'est pas effectué, 1 si l'affichage/modification est fait, et retourne 1 ou 0 dans la variable globale **fk\_newmenu**.

- **int fk\_item\_int ( char \*string, int \*num, int lower, int upper )**

Affiche/modifie un nombre entier en utilisant le système five-key. Le terme **\*string** est un format **printf** ayant la forme **%nu** où **n** représente 1 digit, par exemple, **%5d**. Le terme **\*num** est l'entier à afficher et/ou modifier. Les arguments **upper** et **lower** sont les limites supérieures et inférieures pour ce nombre. La fonction retourne 0 si l'affichage/modification n'est pas effectué, 1 si l'affichage/modification est fait, et retourne 1 ou 0 dans la variable globale **fk\_newmenu**.

- **int fk\_item\_uint ( char \*string, uint \*num, uint lower, uint upper )**

Cette fonction est identique à **fk\_item\_int**, mais s'applique aux entiers non signés. (garder à l'esprit que **uint** est une convention pour ce manuel seulement et n'est pas un mot clé du C).

- **int fk\_item\_float ( char\*s1, float \*num, float lower, float upper )**

Affiche/modifie un nombre virgule flottante en utilisant le système five-key. Le terme **\*s1** est un format **printf** pour l'affichage du nombre. Le code du format doit être sous la forme **%n.mf**. La ligne affichée apparaît comme suit:

```
vvvvvv wwww.YYYY
```

où **vvvvv** est une chaîne d'invite (prompt string), **www** sont  $n$  caractères longs, et **YYYY** sont  $m$  caractères longs. La valeur  $n$  doit être au moins de 1. La somme  $n + m$  ne peut pas dépasser 9. Par défaut,  $n = 5$  et  $m = 2$ . Le terme **\*num** est le nombre virgule flottante à afficher et/ou modifier. Les arguments **upper** et **lower** sont les limites supérieures et les limites inférieures de ce nombre. Cette fonction est seulement valable pour les nombres compris dans les domaines [ 1E6, -1E-4 ], [1E-4, 1E6] avec une spécification du format correcte. La fonction retourne 0 si l'affichage ou la modification n'est pas effectué, 1 si l'affichage ou la modification est fait, et retourne 1 ou 0 dans la variable globale **fk\_newmenu**.

- **int fk\_item\_enum ( char \*prompt, int \*choice, char \*s1, . . . \*sn, “ “ )**

Permet à l'utilisateur de choisir à partir d'une liste de chaînes terminées nulles (maximum 20). La chaîne **\*prompt** doit contenir une chaîne de code champs ( **%s** ou **%ns** ) utilisée pour imprimer les chaînes. La dernière des chaînes (après **\*s1, . . . \*sn** ) doit être **null**. Le terme **\*choice** retourne le choix de l'utilisateur, de 0 à  $n-1$ . La fonction retourne 0 si elle n'est pas faite, 1 si elle est faite, et retourne 1 ou 0 dans la variable globale **fk\_newmenu**.

- **int fk\_item\_setdate ( struct tm \*time )**

C'est une fonction five-key pour modifier les champs de jour, mois et année d'une structure **tm**. Le terme **\*time** est la structure à modifier. La fonction retourne 0 si elle n'est pas faite, 1 si elle est faite, et retourne 1 ou 0 dans la variable globale **fk\_newmenu**.

- **int fk\_item\_settime ( struct tm \*time )**

C'est une fonction five-key pour modifier les champs d'heure, minute et seconde d'une structure **tm**. Le terme **\*time** est la structure à modifier. La fonction retourne 0 si elle n'est pas faite, 1 si elle est faite, et retourne 1 ou 0 dans la variable globale **fk\_newmenu**.

## XP88XX.LIB

Ce sont les fonctions support pour le pilotage des moteurs pas à pas.

- **uint sm\_bdaddr ( int jumpers )**

Retourne l'adresse PLCBus pour une carte de pilotage moteur pas à pas , dans une configuration donnée, comme défini à H4. La fonction retourne l'adresse PLCBus pour la carte.

- **int sm\_poll ( uint bdaddr )**

Recherche une carte moteur à une adresse PLCBus donnée. La fonction retourne 0 si la carte est trouvée, 1 si elle n'est pas trouvée.

- **void sm\_hitwd ( int index )**

Active le chien de garde du MAX705 en exécutant un cycle de lecture compteur.

**index**: est l'index de la carte d'extension.

- **int sm\_find\_boards ( )**

Scrute les 16 adresses possibles pour une carte moteur et charge la ou les adresses des cartes trouvées dans le tableau **sm\_addr**. Les adresses trouvées sont stockées dans le tableau dans l'ordre croissant (0-15). Les tableaux correspondant d'octets d'états et drapeaux de services sont initialisés. La fonction retourne le nombre de cartes trouvées. L'élément suivant la dernière adresse trouvée est fixé à **0xFFFF**. Le registre de contrôle est initialisé à une valeur **0xA7** pour toutes les cartes trouvées. La fonction retourne un entier représentant le nombre de cartes de pilotage moteur répondant à la scrutation.

- **uint smq\_read16 ( int index )**

Retourne en entier le nombre 16 bits du compteur quadrature. **index** est un nombre entre 0 et 15 représentant la séquence des cartes moteur trouvées par **sm\_find\_boards**. La carte avec la plus petite configuration cavalier sera en position 0, la suivante en 1, ...etc. La fonction retourne la valeur 16 bits du compteur quadrature.

- **char smq\_read8 ( int index )**

Retourne l'octet de poids faible du compteur quadrature. **index** est un nombre compris entre 0 et 15 représentant la séquence des cartes moteur trouvées par **sm\_find\_boards**. La carte avec la plus petite configuration cavalier sera en position 0, la suivante en 1, ...etc. La fonction retourne l'octet de poids faible du compteur.

- **void sm\_board\_reset ( int index )**

Effectue un reset hardware sur le contrôleur et l'encodeur. Désactive le pilote (driver) et le fixe en mode dual-phase. Fixe le registre de sélection de ligne à 00. **index** est un nombre compris entre 0 et 15 représentant la séquence des cartes moteur trouvées par **sm\_find\_boards**. La carte avec la plus petite configuration cavalier sera en position 0, la suivante en 1, ...etc.

- **void smq\_hardreset ( int index )**

Envoie une commande de reset hardware au compteur quadrature. Remet le compteur à zéro. **index** est un nombre compris entre 0 et 15 représentant la séquence des cartes moteur trouvées par **sm\_find\_boards**. La carte avec la plus petite configuration cavalier sera en position 0, la suivante en 1, ...etc.

- **void smc\_hardreset ( int index )**

Envoie une commande de reset hardware au PLC-AK. Ceci stoppe la sortie et vide les registres internes. **index** est un nombre compris entre 0 et 15 représentant la séquence des cartes moteur trouvées par **sm\_find\_boards**. La carte avec la plus petite configuration cavalier sera en position 0, la suivante en 1, ...etc.

- **void smc\_softreset ( int index )**

Envoie une commande de reset logicielle au PLC-AK. Ceci stoppe la sortie sans vider les registres internes. **index** est un nombre compris entre 0 et 15 représentant la séquence des cartes moteur trouvées par **sm\_find\_boards**. La carte avec la plus petite configuration cavalier sera en position 0, la suivante en 1, ...etc.

- **void smc\_cmd ( int index, int cmd )**

Écrit sur le registre de commande du contrôleur PLC-AK. **index** est un nombre compris entre 0 et 15 représentant la séquence des cartes moteur trouvées par **sm\_find\_boards**. La carte avec la plus petite configuration cavalier sera en position 0, la suivante en 1, ...etc.

- **void smc\_setspeed ( int index, int s1, int s2 )**

Fixe avec les nombres donnés les registres de vitesses haute et basse. Le registre multiplicateur est fixé à 732 pour exprimer les valeurs de vitesses en impulsions par seconde. La sortie impulsionnelle est fixé à ON, pas d'IRQ de rampe descendante, et polarité normale. **index** est un nombre compris entre 0 et 15 représentant la séquence des cartes moteur trouvées par **sm\_find\_boards**. La carte avec la plus petite configuration cavalier sera en position 0, la suivante en 1, ...etc. **s1** est la vitesse rapide en impulsions par seconde et **s2** la vitesse lente.

- **void smc\_manual\_move ( int index, int dir, int speed )**

Commence un fonctionnement en mouvement manuel. Le moteur tournera jusqu'à ce qu'une commande stop, décélération, un reset logiciel (**smc\_softreset**), ou une impulsion EL ou ORG soit détectée (si validée). **index** est un nombre compris entre 0 et 15 représentant la séquence des cartes moteur trouvées par **sm\_find\_boards**. La carte avec la plus petite configuration cavalier sera en position 0, la suivante en 1, ...etc. **dir** est le sens du mouvement: 1 = en avant, 0 = en arrière; **speed**: 1 = mode rapide (valeur R2), 0 = mode lent (valeur R1).

- **void smc\_seek\_origin ( int index, int dir, int speed )**

Tourne le moteur jusqu'à ce qu'une impulsion d'origine soit détectée. **index** est un nombre compris entre 0 et 15 représentant la séquence des cartes moteur trouvées par **sm\_find\_boards**. La carte avec la plus petite configuration cavalier sera en position 0, la suivante en 1, ...etc. **dir** est le sens du mouvement: 1 = en avant, 0 = en arrière; **speed**: 1 = mode rapide (valeur R2), 0 = mode lent (valeur R1).

- **void smc\_setmove ( int index, long R0, int R1, int R2, int R4, int R6, int R7 )**

Prépare les registres du PLC-AK pour une opération de mouvement. **index** est un nombre compris entre 0 et 15 représentant la séquence des cartes moteur trouvées par **sm\_find\_boards**. La carte avec la plus petite configuration cavalier sera en position 0, la suivante en 1, ...etc. **R0**: nombre d'impulsions de mouvement; **R1**: taux de mouvement en vitesse lente; **R2**: taux de mouvement en vitesse rapide; **R4**: taux d'accélération; **R6**: point de rampe descendante; **R7**: registre multiplicateur (fixé à 732 pour **R1** et **R2** exprimés en impulsions par seconde).

- **uint smcq\_moveto ( int index, uint dest, int dir, uint acc )**

Tourne manuellement le moteur jusqu'à ce que l'encodeur rencontre une valeur donnée. Le mouvement a lieu à la vitesse spécifiée dans **R1** (vitesse lente) du contrôleur. Par exemple,

**smcq\_moveto ( myaddr, 5000, 1, 25 ) ;**

tourne le moteur en avant jusqu'à ce que l'encodeur lise une valeur dans l'intervalle 4075 - 5025.

ATTENTION: la vitesse du mouvement, la résolution de l'encodeur, et la phase/degrés du moteur sont directement liés à la bonne marche du fonctionnement. Il est possible de manquer un point d'arrêt si une précision trop grande est demandée. L'encodeur doit être lu après le fonctionnement (laissant le temps au moteur d'arriver sur un stop) pour s'assurer que sa position soit correcte.

**index** est un nombre compris entre 0 et 15 représentant la séquence des cartes moteur trouvées par **sm\_find\_boards**. La carte avec la plus petite configuration cavalier sera en position 0, la suivante en 1, ...etc. **dest** est la valeur de l'encodeur sur laquelle s'arrêter; **dir** est la direction du mouvement: 1 = en avant, 0 = en arrière; **acc** est la précision de la valeur stop. La fonction retourne la dernière valeur de l'encodeur lue (quand la décision d'arrêt a été prise). L'inertie et l'emplacement des pas feront que la valeur de position finale sera différente de ce nombre.

- **char smc\_stat0 ( int index )**

Lit le registre d'état à l'adresse 0 (A1 = A0 = 0) sur le contrôleur PLC-AK. **index** est un nombre compris entre 0 et 15 représentant la séquence des cartes moteur trouvées par **sm\_find\_boards**. La carte avec la plus petite configuration cavalier sera en position 0, la suivante en 1, ...etc. La fonction retourne la valeur du registre STAT0 sur le PLC-AK.

- **char smc\_stat3 ( int index )**

Lit le registre d'état à l'adresse 1 (A1 = A0 = 1) sur le contrôleur PLC-AK. **index** est un nombre compris entre 0 et 15 représentant la séquence des cartes moteur trouvées par **sm\_find\_boards**. La carte avec la plus petite configuration cavalier sera en position 0, la suivante en 1, ...etc. La fonction retourne la valeur du registre STAT3 sur le PLC-AK.

- **void sm\_ctlreg ( int index, int dat )**

Ecrit une valeur **dat** sur le registre de contrôle (écriture seulement) de la carte d'extension moteur pas à pas. Remet à jour sur le registre la variable "shadow". **index** est un nombre compris entre 0 et 15 représentant la séquence des cartes moteur trouvées par **sm\_find\_boards**. La carte avec la plus petite configuration cavalier sera en position 0, la suivante en 1, ...etc.

- **void sm\_drvoe ( int index, int onoff )**

Commute la sortie du pilote moteur ON ou OFF. **index** est un nombre compris entre 0 et 15 représentant la séquence des cartes moteur trouvées par **sm\_find\_boards**. La carte avec la plus petite configuration cavalier sera en position 0, la suivante en 1, ...etc. **onoff**: 0: OFF, 1: ON. La fonction ne retourne rien.

- **void sm\_led ( int index, int onoff )**

Allume ou éteint la LED utilisateur. **index** est un nombre compris entre 0 et 15 représentant la séquence des cartes moteur trouvées par **sm\_find\_boards**. La carte avec la plus petite configuration cavalier sera en position 0, la suivante en 1, ...etc. **onoff**: 0: éteinte, 1: allumée.

- **sm\_se100 ( int index )**

Fixe les bits sélectionnés dans le registre (en écriture seulement) de la carte d'extension contrôleur moteur pas à pas à 00. Remet à jour le registre "shadow" pour cette séquence. **index** est un nombre compris entre 0 et 15 représentant la séquence des cartes moteur trouvées par **sm\_find\_boards**. La carte avec la plus petite configuration cavalier sera en position 0, la suivante en 1, ...etc.

- **sm\_se101 ( int index )**

Fixe les bits sélectionnés dans le registre (en écriture seulement) de la carte d'extension contrôleur moteur pas à pas à 01. Remet à jour le registre "shadow" pour cette séquence. **index** est un nombre compris entre 0 et 15 représentant la séquence des cartes moteur trouvées par **sm\_find\_boards**. La carte avec la plus petite configuration cavalier sera en position 0, la suivante en 1, ...etc.

- **sm\_se110 ( int index )**

Fixe les bits sélectionnés dans le registre (en écriture seulement) de la carte d'extension contrôleur moteur pas à pas à 10. Remet à jour le registre "shadow" pour cette séquence. **index** est un nombre compris entre 0 et 15 représentant la séquence des cartes moteur trouvées par **sm\_find\_boards**. La carte avec la plus petite configuration cavalier sera en position 0, la suivante en 1, ...etc.

- **sm\_se111 ( int index )**

Fixe les bits sélectionnés dans le registre (en écriture seulement) de la carte d'extension contrôleur moteur pas à pas à 11. Remet à jour le registre "shadow" pour cette séquence. **index** est un nombre compris entre 0 et 15 représentant la séquence des cartes moteur trouvées par **sm\_find\_boards**. La carte avec la plus petite configuration cavalier sera en position 0, la suivante en 1, ...etc.

- **void sm\_int ( )**

Routine de service interruption (ISR) générale pour le contrôleur moteur pas à pas. Vérifie l'état de toutes les cartes listées dans le tableau **sm\_addr** pour une demande d'interruption (remet à jour **sm\_stat**). Quand une demande d'interruption est détectée, le drapeau de tableau du service (**sm\_flags**) est mit, et la carte moteur pas à pas reçoit un reset logiciel. Ce reset désactive les demandes d'interruption émanant du contrôleur. Les drapeaux service sont surveillés par le programme maître pour déterminer quand le fonctionnement est terminé.

Effectuer les étapes suivantes pour s'entraîner avec cette fonction:

1. Appeler **sm\_find\_boards** en début de programme.
2. Ajouter la définition suivante pour linker cette fonction à l'ISR du PLCBus.

```
#define USE_MOTORCARD // active l'ISR motor_int
```

3. Valider l'interruption PLCBus (ligne /AT) avec l'instruction suivante en début de programme.

```
relocate-int1 ( ) ; // réaffecte le vecteur d'interruption.  
outport ( ITC , ( inport ( ITC ) | 0x02 ) ) ; // valide l'IRQ n°1
```

Remplacez le code entre les labels **mirq** et **fin** avec votre propre code pour effectuer tous les traitements moteur en tâche de fond.

- **void set82adr ( int address )**

Envoie sur le PLCBus une adresse sur deux octets (pour le mode 8 x 2).

- **void set81adr ( int addr )**

Envoie sur le PLCBus le dernier octet d'une adresse sur deux octets (pour le mode 8 x 2).

## IOEXPAND.LIB

Ce sont les fonctions support pour les cartes d'extension du BL1100. Elles sont divisées en deux classes.

1. Fonctions codées par le matériel pour une adresse de base par défaut de 0xFxxx.
2. Fonctions qui permettent à l'utilisateur de spécifier une carte par son numéro nodal.

La première classe est plus rapide, mais limitée aux systèmes ne comportant qu'une seule carte d'extension. La deuxième n'est donc à utiliser que pour les systèmes comportant plusieurs cartes d'extension.

Il existe une structure d'adresses par défaut pour améliorer les vitesses de fonctionnement des instructions de la deuxième classe. Il y a une structure qui prend en compte les adresses par défaut. Au lieu de spécifier un indice nodal entre 0 et 3, spécifier -1. Cela chargera les bonnes adresses par défaut. Le second ensemble de fonctions permet d'empiler jusqu'à quatre cartes d'extension sur un BL1100.

Les adresses de carte sont fixés à l'aide du cavalier J10.

SE REFERER A LA DOCUMENTATION XP8100/XP8200 POUR OBTENIR LES BONNES ADRESSES DE BASE

- **int exp\_init ( int ppia, int ppib, int ppicu, int ppicl )**

Initialise les ports d'E/S d'une carte d'extension du BL1100 avec l'adresse par défaut 0xFxxx. Le PPI (U5) utilise le mode 0 ou le mode d'E/S de base. **ppia**, **ppib**, **ppicu**, et **ppicl** sont les valeurs de sortie pour le registre de sortie PPI. Configure le port A en entrée si **ppia** = -1, et le port B en entrée si **ppib** = -1. Configure le groupe de poids fort du port C en entrée si **ppicu** = -1; et le groupe de poids faible du port C en entrée si **ppicl** = -1. Tous les ports de sortie PPI sont réinitialisés à l'état bas quand le mode est changé. Il est important d'envoyer une valeur correcte sur le port de sortie juste après avoir changé de mode.

- **int mux\_ch ( int chan )**

Fixe le multiplexeur DG509A (U17) de la carte d'extension du BL1100 avec l'adresse par défaut 0xFxxx. **chan** est 0 ou 3 pour (AN0- , AN0+) à (AN3- , AN3+), respectivement, pour multiplexer sur (MUX-DA, MUX-DB).

- **int ad20\_mux ( int chan )**

Fixe le multiplexeur pour le AD7703 20-bits de la carte d'extension du BL1100 avec l'adresse par défaut 0xFxxx. Les voies 0 à 3 réservées pour un fonctionnement unipolaire (0 à 2,5 volts) sont référencées (AN0- , AN0+) à (AN3- , AN3+), alors que respectivement les voies 4 à 7 réservées pour un fonctionnement bipolaire (-2,5 à +2,5 volts) sont référencées (AN0- , AN0+) à (AN3- , AN3+).

- **int ad20\_rdy ( )**

teste l'état DRDY du AD7703 à partir de RDTTL bit 1. La fonction retourne 0 si AD20 est prêt, ou 1 si AD20 est occupé.

- **int ad20\_cal ( int mode )**

Calibre l'AD7703 sur la carte d'extension BL1100 avec l'adresse par défaut 0xFxxx. La calibration en mode 0 n'utilise pas le multiplexeur. La calibration en mode 1 utilise le multiplexeur pour obtenir le zéro et la pleine échelle sur Ain. Mux ch0 est le signal analogique à mesurer. Mux ch1 est Ain dans la première étape du mode 1 pour calibrer le décalage constant (offset) du système. Mux ch2 est Ain dans la seconde étape du mode 1 pour calibrer le gain du système. La calibration en mode 2 utilise la voie courante pour obtenir Ain comme zéro pour calibrer le décalage constant du système.

La table suivante montre l'état de SC1 et SC2 pendant la calibration:

Mode	SC1	SC2	Type de cal.	Zéro	Etapas pleine échelle
0	0	0	Auto-calibration	AGND	REF+1
1	1	1	Décalage système	Ain	1 de 2
1	0	1	Gain système	Ain	2 de 2
2	1	0	Décalage système	Ain	REF+1

La fonction retourne zéro quand la calibration est terminée ou -1 si une erreur est survenue.

- **long ad20\_rd ( )**

Lit une donnée 20-bits à partir du port de données série de l'AD7703. Le délai de réponse de 125 mSec. de l'AD7703 oblige à ce qu'un laps de temps soit garanti après commutation du multiplexeur. La donnée analogique sera validée quand DRDY est au niveau bas pour la sortie données jusqu'à une vitesse de 4 KHz. La polarité ainsi que la voie à lire doivent être fixées avant à l'aide de **ad20\_mux**. Les gammes Ain vont de 0 à +2,5 volts en mode unipolaire (PA0=0).  
 LSB = 2,5 volts / 1048576 = 2,384 µVolts. Les gammes Ain vont de -2,5 à +2,5 volts en mode bipolaire (PA0=1).  
 LSB = 5 volts / 1048576 = 4,768 µVolts.

La fonction retourne une donnée analogique sur 20 bits. En mode unipolaire, 0x00000 = AGND, 0x7FFFF = 1,25 volts et 0xFFFFF = 2,5 volts. En mode bipolaire, 0x00000 = -2,5 volts, 0x7FFFF = AGND et 0xFFFFF = 2,5 volts.

- **int exp\_init\_n ( int node, int ppia, int ppib, int ppicu, int ppicl, int def )**

Initialise le port PIO d'une carte d'extension BL1100 correspondant au noeud spécifié. Le noeud va de 0 à 3 pour une adresse nodale respectivement de 0xCxxx à 0xFxxx. Si le noeud = -1, la fonction utilise l'adresse par défaut sauvee dans **def\_na**. Si **def** = 1, le noeud est sauvee comme noeud par défaut dans **def\_na**. Si **def** = 0, le noeud n'est pas sauvee. La fonction retourne 0 si l'initialisation se déroule bien, ou -1 si un noeud inconnu est demandé.

SE REFERER A LA DOCUMENTATION XP8100/XP8200 POUR LA CONFIGURATION DES ADRESSES

- **int get\_na ( int node, struct node\_addr \*na )**

Obtient l'adresse à partir du noeud spécifié (0 - 3). La fonction retourne 0 si **node** est correct; ou -1 si **node** est hors des limites. La donnée adresse du noeud est retournée dans **struct node\_addr \*na**.

- **int set\_def\_na ( int node )**

Fixe l'adresse du noeud en adresse de noeud par défaut. La fonction retourne la donnée contenue dans **get\_na**.

- **int get\_def\_na ( struct node\_addr \*na )**

Obtient l'adresse de noeud par défaut. La fonction retourne le numéro de noeud.

- **int mux\_ch\_n ( int node, int chan, int def )**

Fixe les multiplexeurs DF509A sur les extensions nodales spécifiées du BL1100. **node** = 0 à 3 pour les adresses respectives 0xCxxx à 0xFxxx. **chan** = 0 à 3 pour respectivement, (AN0- , AN0+) à (AN3- , AN3+). Si **node** = -1, la fonction utilise l'adresse par défaut sauvee dans **def\_na**. Si **def** = 1, le noeud est sauvee comme noeud par défaut dans **def\_na**. Si **def** = 0, le noeud n'est pas sauvee. La fonction retourne 0 si la configuration **mux** s'est bien déroulée, ou -1 si **node** est hors limites.

- **int ad20\_mux\_n ( int node, int chan, int def )**

Fixe le multiplexeur DG509A pour l'AD7703 20-bits de la carte d'extension du BL1100. **node** 0 à 3 spécifie l'adresse nodale respectivement de 0xCxxx à 0xFxxx. **chan** 0 à 3 selectionne le fonctionnement en mode unipolaire (0 à 2,5 volts) pour respectivement (AN0- , AN0+) à (AN3- , AN3+). **chan** 4 à 7 selectionne le fonctionnement en mode bipolaire (-2,5 à +2,5 volts) pour respectivement (AN0- , AN0+) à (AN3- , AN3+). Si **node** = -1, la fonction utilise l'adresse par défaut sauvee dans **def\_na**. Si **def** = 1, le noeud est sauvee comme noeud par défaut dans **def\_na**. Si **def** = 0, le noeud n'est pas sauvee. La fonction retourne 0 une fois effectuée avec succès, ou -1 si **node** est invalide.

- **int ad20\_rdy\_n ( int node )**

Teste l'état DRDY de l'AD7703 à partir du bit 1 RDTTL d'un noeud **node** d'une carte d'extension BL1100 spécifiée. **node** 0 à 3 spécifie l'adresse nodale respectivement de 0xCxxx à 0xFxxx. Si **node** = -1, la fonction utilise le noeud sauvee par défaut dans **def\_na**. La fonction retourne 0 si l'AD20 est prêt, ou -1 si l'AD20 est occupé ou si **node** est hors limites.

- **int ad20\_cal\_n ( int mode, int node, int def )**

Calibre l'AD7703 sur la carte d'extension spécifiée du BL1100. **node** 0 à 3 spécifie l'adresse nodale respectivement de 0xCxxx à 0xFxxx. Si **node** = -1, la fonction utilise le noeud sauvee par défaut dans **def\_na**. Si **def** = 1, le noeud est sauvee comme noeud par défaut dans **def\_na**. Si **def** = 0, le noeud n'est pas sauvee.

La calibration en mode 0 n'utilise pas le multiplexeur. La calibration en mode 1 utilise le multiplexeur pour obtenir le zéro et la pleine échelle sur Ain. Mux ch0 est le signal analogique à mesurer. Mux ch1 est Ain dans la première étape du mode 1 pour calibrer le décalage constant (offset) du système. Mux ch2 est Ain dans la seconde étape du mode 1 pour calibrer le gain du système. La calibration en mode 2 utilise la voie courante pour obtenir Ain comme zéro pour calibrer le décalage constant du système.

La table suivante montre l'état de SC1 et SC2 pendant la calibration:

Mode	SC1	SC2	Type de cal.	Zéro	Etapas pleine échelle
0	0	0	Auto-calibration	AGND	REF+1
1	1	1	Décalage système	Ain	1 de 2
1	0	1	Gain système	Ain	2 de 2
2	1	0	Décalage système	Ain	REF+1

La fonction retourne zéro quand la calibration est terminée ou -1 si une erreur est survenue.

- **long ad20\_rd\_n ( int node, int def )**

Lit une donnée 20-bits à partir du port de données série de l'AD7703. Le délai de réponse de 125 mSec. de l'AD7703 oblige à ce qu'un laps de temps soit garanti après commutation du multiplexeur. La donnée analogique sera validée quand DRDY est au niveau bas pour la sortie données jusqu'à une vitesse de 4 KHz. La polarité ainsi que la voie à lire doivent être fixées avant à l'aide de **ad20\_mux**. Les gammes Ain vont de 0 à +2,5 volts en mode unipolaire (PA0=0). LSB = 2,5 volts / 1048576 = 2,384 µVolts. Les gammes Ain vont de -2,5 à +2,5 volts en mode bipolaire (PA0=1). LSB = 5 volts / 1048576 = 4,768 µVolts. **node** 0 à 3 spécifie l'adresse nodale respectivement de 0xCxxx à 0xFxxx. Si **node** = -1, la fonction utilise le noeud sauvé par défaut dans **def\_na**. Si **def** = 1, le noeud est sauvé comme noeud par défaut dans **def\_na**. Si **def** = 0, le noeud n'est pas sauvé. La fonction retourne une donnée analogique sur 20 bits. En mode unipolaire, 0x00000 = AGND, 0x7FFFF = 1,25 volts et 0xFFFFF = 2,5 volts. En mode bipolaire, 0x00000 = -2,5 volts, 0x7FFFF = AGND et 0xFFFFF = 2,5 volts. Retourne -1 pour un noeud non valide.

## KDM.LIB

Ces fonctions KDM (Keyboard/Display Module) procurent des pilotes logiciels pour les claviers KDM, les écrans LCD texte et graphique, le bipeur et le timer qui pilote le clavier. Le bipeur pilote aussi le noyau temps réel (RTK) quand **RUNKERNEL** est défini.

- **int lk\_kxinit ( )**

Initialise les variables, les buffers et les pilotes hardware associés aux services du clavier KDM.

- **int lk\_loadtab ( int \*tab, int tab\_size )**

Charge les valeurs de la table numérique du clavier. Cette fonction est utilisée pour réarranger les touches du clavier. **tab** pointe sur un tableau d'entiers contenant le nouveau clavier arrangé. **tab\_size** est la taille de la table à changer. Par exemple, **new-table [ ] = { 4, 3, 2, 1, . . . }** va réarranger l'ordre des quatre premières touches.

- **int lk\_settab ( char flag )**

Fixe la table d'interprétation du clavier pour des tailles clavier supérieures à 24.

- **int lk\_keyw ( char flag )**

Ecrit aux bits spécifiés dans le registre touches.

- **int lk\_kxget ( char mode )**

Reçoit des caractères du clavier KDM. Si **mode** = 0, retire le caractère du buffer clavier et le retourne. Si **mode** != 0, retourne le caractère (si disponible), mais ne le supprime pas du buffer clavier. La fonction retourne le caractère clavier pressé ou -1 si le buffer clavier est vide.

- **int lk\_setbeep ( int count )**

Configure la variable utilisée pour le bipeur KDM.

- **int lk\_led ( int mode )**

Commute les LEDs du KDM on/off sans entrer en conflit avec le pilote clavier. **mode** = 0 éteint les LEDs. **mode** = 1 allume la LED jaune. **mode** = 2 allume la LED verte. **mode** = 3 allume les deux LEDs. La fonction retourne le mode qui a été passé.

- **int lk\_tdelay ( int delay )**

Mécanisme de délai pratique lié aux interruptions périodiques de **timer1**.

- **int lk\_int\_timer1 ( )**

Routine de service pour l'interruption de **timer1**. Pilote le bipeur et le clavier. Pilote aussi le noyau temps réel si **RUNKERNEL** est défini.

- **int lg\_init\_keypad ( )**

Initialise **timer1**, le pilote clavier KDM et le LCD graphique.

- **int lk\_init\_keypad ( )**

Initialise **timer1**, le pilote clavier et le LCD.

- **void lk\_wr ( int x )**

Ecrit l'octet de poids faible de **x** au registre LCD dans l'octet de poids fort de **x**.

- **int lk\_rd ( int addr )**

Lit une donnée du registre lecture **addr** du LCD. La fonction retourne la donnée du registre lecture **addr** du LCD.

- **int lk\_init ( )**

Initialise le LCD sur le KDM. Initialise les variables logicielles associées à l'utilisation du LCD.

- **int lk\_cmd ( int cmd )**

Envoie la commande de l'octet de poids faible de **cmd** au registre LCD spécifié par l'octet de poids fort de **cmd**.

- **int lk\_wait ( )**

attend l'unité LCD appropriée pour effacer ses drapeaux occupés. La fonction retourne 0 ou 1, cela dépend du contrôleur LCD.

- **int lk\_char ( char x )**

Envoie un caractère au registre de données du LCD approprié.

- **int lk\_ctrl ( char x )**

Envoie un caractère au registre de contrôle du LCD approprié.

- **int lk\_putc ( char x )**

Pilote bas-niveau pour le LCD ( similaire à **printf**). Envoie un caractère au LCD et met à jour les variables logicielles pour stocker l'état de l'écran LCD.

- **int lk\_nl ( )**

Génère une nouvelle ligne sur l'écran LCD.

- **int lk\_pos ( int line, int col )**

Positionne le curseur LCD à l'emplacement défini par **line** et **col**.

- **int lk\_printf ( char \*fmt, . . . )**

C'est la commande similaire à **printf** pour le LCD. Les séquence "escape" suivantes sont disponibles.

esc p <i>n m</i>	Positionne le curseur en ligne <i>n</i> et colonne <i>mm</i> . Exemple: <b>lk_printf ( "\x1bp234" )</b> ; signifie ligne 2, colonne 34. Les lignes sont numérotés 0, 1, 2, 3 et les colonnes 0, 1, 2, ...39.
esc 1	Active le curseur.
esc 0	Désactive le curseur.
esc c	Efface à partir de la position du curseur jusqu'à la fin de la ligne.
esc b	Active le mode curseur clignotant.
esc n	Désactive le mode curseur clignotant.
esc e	Efface l'afficheur et réinitialise la position du curseur (home).

- **void lk\_cgram ( char \*p )**

Générateur de caractères spéciaux pour le LCD. \*p (premier octet) est le nombre d'octets à stocker (jusqu'à 64 pour 8 caractères). Les cinq bits de poids les plus faibles de chaque octet forment une rangée de caractères de la gauche vers la droite et du haut vers le bas. La huitième rangée de chacun correspond à la position du curseur.

- **int lk\_stdcg ( )**

Charge une table de caractères spéciaux **lk\_stdchars** sur le LCD.

- **int lk\_run\_menu ( char \*call\_menu, struct lk\_menu \*menu, int index )**

Schéma de menu pour le module KDM. Les codes mtype suivants sont disponibles dans les structures menu:

0	Fin de menu.
1	Voir les flottants.
2	Voir les flottants et les ajuster entre bornes.
3	Voir les flottants et entrer une nouvelle valeur sur "enter".
4	Comme 2 mais appelle une fonction spécifiée passant un pointeur après chaque étape.
5	Comme 3 mais appelle une fonction spécifiée passant un pointeur à la nouvelle valeur.
8	Voir la logique.
9	Voir la logique et ajuster vrai/faux.
10	Comme 9 mais appelle une fonction spécifiée passant un pointeur à une variable.
16	Voir date et heure.
17	Voir / modifier date et heure.
18	Voir / modifier date et heure et appeler une routine.
20	Voir l'heure (16 bits).
21	Voir / modifier l'heure (16 bits).
22	Voir / modifier l'heure (16 bits) puis appeler une routine.
32	Appeler un nouveau menu ( <b>msg</b> est le nom de la ligne du haut pour le nouveau menu, <b>valptr</b> est le pointeur sur la structure du nouveau menu, l' <b>index</b> est toujours passé comme 0).
40	Appelle une fonction ( <b>msg</b> est affiché, <b>ptr</b> et <b>limit</b> sont ignorés). La chaîne <b>call_menu</b> est initialement imprimée quand le menu est entré. Le pointeur <b>menu</b> pointe sur la structure <b>lk_menu</b> . L' <b>index</b> est le point de départ dans le menu, souvent zéro. La fonction <b>run_menu</b> retourne la dernière valeur de l' <b>index</b> .

- **void lk\_setdate ( char \*msg, struct tm \*dat )**

Fixe la donnée date et l'imprime sur le LCD. Imprime aussi **msg** sur le LCD. Fonction utilisée par **lk-run-menu**.

- **int lk\_chkdat ( struct tm \*dat )**

Vérifie la validité de la donnée date. Peut changer le jour du mois. La fonction retourne 0 si la donnée date est correcte, ou 1 pour une donnée date non valide.

- **void lk\_showdate ( char \*msg, struct tm \*tmm )**

Affiche la donnée date et **msg** sur le LCD.

- **uint lk\_settime ( char \*msg, uint time )**

Fixe l'heure et l'affiche sur le LCD. Affiche aussi **msg** sur le LCD.

- **int lk\_showtime ( char \*msg, uint time )**

Affiche **msg** et la donnée heure sur le LCD.

- **int st\_hour ( uint j )**

Décimateur d'heures utilisé par **lk\_run\_menu**. La fonction retourne  $j / 1800$ .

- **int st\_min ( uint j )**

Décimateur de minutes utilisé par **lk\_run\_menu**. La fonction retourne  $(j \text{ modulo } 1800) / 30$ .

- **int st\_sec ( uint j )**

Décimateur de secondes utilisé par **lk\_run\_menu**. La fonction retourne  $2 \times (j \text{ modulo } 30)$ .

- **uint mk\_st ( int hour, int min, int sec )**

Générateur de données heures utilisé par **lk\_run\_menu**. La fonction retourne **hour** x 1800 + **min** x 30 + **sec** x 2.

- **uint ad\_st ( uint t1, uint t2 )**

Additionneur de données heures utilisé par **lk\_run\_menu**. La fonction retourne la donnée heure ajustée, somme des deux données heure.

- **int lk\_secho ( )**

Retire un caractère du buffer touche et génère un bip court.

- **int lk\_lecho ( )**

Retire un caractère du buffer clavier et génère un bip long.

- **void lk\_view1 ( char \*fmt, char var )**

Visualise une variable logique.

- **float lk\_getknum ( )**

Obtient un nombre virgule flottante à partir du clavier. La fonction retourne le nombre virgule flottante entré à l'aide du clavier.

- **void lg\_init ( )**

Initialise le LCD graphique et ses variables logicielles associées.

- **void lg\_char ( char x )**

Ecrit un caractère sur le LCD graphique.

- **void lg\_putc ( char x )**

Pilote bas-niveau pour le LCD graphique (similaire à **printf**). Envoie **char** sur le LCD graphique et remet à jour les variables logicielles qui stockent les états de l'écran LCD graphique.

- **void lg\_nl ( )**

Génère une nouvelle ligne sur l'écran du LCD graphique.

- **void lg\_pos ( int line, int col )**

Positionne le curseur sur l'écran du LCD graphique.

- **void lg\_printf ( char \*fmt, . . . )**

C'est la commande similaire à **printf** pour le LCD graphique. Les séquences "escape" suivantes sont disponibles.

esc p <i>n m</i>	Positionne le curseur en ligne <i>n</i> et colonne <i>mm</i> . Exemple: <b>lg_printf ( "\x1bp234" )</b> ; signifie ligne 2, colonne 34. Les lignes sont numérotés 0, 1, 2, 3 et les colonnes 0, 1, 2, ...39.
esc l	Active le curseur.
esc 0	Désactive le curseur.
esc c	Efface à partir de la position du curseur jusqu'à la fin de la ligne.
esc b	Active le mode curseur clignotant.
esc n	Désactive le mode curseur clignotant.
esc e	Efface l'afficheur et réinitialise la position du curseur (home).

- **void Set\_Display\_Mode ( int mode )**

Fixe le mode d'affichage du LCD graphique. **mode** est **DISPLAY\_TEXT** (4) ou **DISPLAY\_GRAPHICS** (8).

- **void Clear\_GrTxt\_Screen ( )**

Efface l'écran texte du LCD graphique.

- **void Stall ( int tix )**

Boucle d'attente logicielle. Décompte **tix** x 10.

- **void sta01 ( )**

Ecrit 4 sur le registre écriture du LCD et attend après un 3 sur le registre lecture du LCD.

- **void sta03 ( )**

Ecrit 4 sur le registre écriture du LCD et attend après un 0x08 sur le registre lecture du LCD.

- **void lg\_wr ( int x )**

Ecrit une donnée sur le registre du LCD graphique. La valeur du registre se trouve dans l'octet de poids fort et la valeur de la donnée se trouve dans l'octet de poids faible de **x**. Utiliser **sta01** pour attendre l'effacement pour l'écriture.

- **void lg\_wr03 ( int x )**

Ecrit une donnée sur le registre du LCD graphique. La valeur du registre se trouve dans l'octet de poids fort et la valeur de la donnée se trouve dans l'octet de poids faible de **x**. Utiliser **sta03** pour attendre l'effacement pour l'écriture.

- **void lg\_rd ( )**

Attend pour effacer et lire le registre lecture du LCD graphique.

- **void grp\_home\_area ( char gal, char gah, char ghl, char ghh )**

Fixe la zone graphique en définissant l'origine (**ghl, ghh**) et la surface (**gal, gah**).

- **void text\_home\_area ( char tal, char tah, char thl, char thh )**

Fixe la zone texte en définissant l'origine (**thl, thh**) et la surface (**tal, tah**).

- **void Graph\_Init ( )**

Initialise les zones graphique et texte du LCD graphique.

- **void Set\_Pointer ( int address, int ptr )**

Fixe le pointeur approprié en utilisant la commande "pointer set". **address** est l'adresse sur laquelle fixer le pointeur. **ptr** est le pointeur à fixer: 1 = curseur, 2 = offset, 4 = adresse.

VOIR PAGE 25 DU MANUEL TOSHIBA ST-LCD

- **int Text\_Addr ( int col, int row )**

Calcule la position du texte à partir des données **row** et **col**.

La fonction retourne **GRTXT\_BASE\_ADDRESS + row x LK\_COLS + col**.

- **void Set\_Auto\_Mode ( int mode )**

Fixe le LCD graphique en mode auto.

- **void Set\_Overlap\_Mode ( int mode )**

Fixe le LCD graphique en mode recouvrement (overlap).

- **void Define\_Cursor ( int lines )**

Définit le curseur pour le LCD graphique.

- **void Set\_Pixel ( int col, int row, int wr\_mode )**

Allume un pixel LCD aux coordonnées (**col, row**). **wr\_mode** = 0 pour effacer, **wr\_mode** = 1 pour allumer, et **wr\_mode** = 2 pour effectuer un XOR. (0,0) est le coin bas-gauche. **col** est compris entre 0 et 239; **row** est compris entre 0 et 63.

- **void Clear\_Gr\_Screen ( )**

Efface la palette graphique en remplissant de zéros toutes les adresses RAM du LCD graphique.

- **void Map\_Bit\_Pattern ( int \*config, char \*bitarray, int wr\_mode )**

Remplit une zone du LCD graphique avec une trame (motif). **config** pointe sur un tableau de 4 données entières définissant le coin haut-gauche (x,y) pour démarrer la trame et la hauteur et largeur du motif en points. **bitarray** pointe sur un tableau de données caractères représentant la matrice du motif à l'aide de "1" ou "\*" pour chaque petit point du motif. Les données apparaissent en ordre séquentiel en commençant par le coin haut-gauche et en progressant de la gauche vers la droite et du haut vers le bas. **wr\_mode** = 0 pour effacer, **wr\_mode** = 1 pour imprimer, et **wr\_mode** = 2 pour effectuer un XOR.

- **void Draw\_Line ( int stx, int sty, int enx, int eny, int wr\_mode )**

Trace une ligne du point de départ (**stx, sty**) au point d'arrivée (**enx, eny**). **wr\_mode** = 0 pour effacer, **wr\_mode** = 1 pour tracer, et **wr\_mode** = 2 pour effectuer un XOR.

- **void Draw\_Poly ( int numpoints, int \*point, int wr\_mode )**

Trace un polygone en connectant des points succésifs. **numpoints** est le nombre de paires de coordonnées (x, y). **point** pointe sur un tableau d'entiers de paires de coordonnées (x, y).

- **void Draw\_Axis ( int ox, int oy, int ex, int ey, int ticks\_x, int ticks\_y, int wr\_mode )**

Trace un axe avec (**ox, oy**) comme origine. (**ex, ey**) sont les coordonnées les plus grandes de l'axe. **ticks\_x** est le nombre de graduations sur l'axe x et **ticks\_y** est le nombre de graduations sur l'axe y.

- **void Sin\_Wave ( int ox, int oy, int ex, int ey, int cycles, int wr\_mode )**

Trace une onde sinusoïdale avec (**ox, oy**) comme origine. (**ex, ey**) sont les coordonnées les plus hautes de la sinusoïde. **cycles** est le nombre de cycles à afficher.

## LCD2L.LIB

Ce sont les fonctions support au LCD. Elles supportent les LCD 2 x 20 et tous les produits Z-World qui possèdent un port LCD.

- **void lc\_wr ( char data )**

Routine bas-niveau pour écrire **char data** dans un registre de contrôle du LCD. Le registre de contrôle accédé est embarqué dans **char data**.

- **int lc\_rd ( )**

Routine bas-niveau pour lire le registre **LCDWR** du LCD. La fonction retourne le drapeau occupé dans le bit 7 et le compteur de l'adresse du LCD dans les 7 bits de poids faible.

- **void lc\_init ( )**

Initialise le LCD du PK2100 ou du PK2200 en exécutant le protocole de mise en route recommandé du LCD. Fixe le LCD en auto incrément; affichage et curseur "on"; et efface la mémoire affichage.

- **int lc\_cmd ( int cmd )**

Attend que le drapeau occupé du LCD s'efface puis envoie **cmd** au registre de commande du LCD. La fonction retourne 0 si l'écriture du LCD s'est bien déroulé, ou -1 en cas de "time-out" si l'écran est occupé.

- **int lc\_wait ( )**

Attend que le drapeau occupé du LCD s'efface. La fonction retourne 0 quand le drapeau s'efface, ou -1 en cas de "time-out" après dix essais.

- **void lc\_char ( char x )**

Ecrit **char x** sur le registre de données du LCD.

- **void lc\_ctrl ( char x )**

Ecrit **char x** sur le registre de contrôle du LCD. A la différence de **lc\_wr**, cette fonction attend le drapeau occupé du LCD pour effacer avant l'écriture d'une donnée sur un registre de contrôle du LCD.

- **int lc\_putc ( char x )**

Décode **char x** pour des séquences de commandes spéciales pour l'écriture sur les registres de commande ou de données du LCD. Cette fonction sert de pilote à **lc\_printf**.

- **void lc\_nl ( )**

Déplace le curseur LCD sur la première colonne de la ligne suivante. Si la ligne courante est la dernière ligne du LCD, alors le curseur est seulement ramené en colonne 0 de la ligne courante.

- **void lc\_pos ( int line, int col )**

Positionne le curseur du LCD PK2100 sur la ligne **line** (0 à 3) spécifiée et la colonne **col** (0 à 19).

- **void lc\_printf ( char \*fmt, ... )**

C'est la commande similaire à **printf** pour le LCD du PK2100.

- **void lc\_cgram ( char \*p )**

Matrice de caractères de 5 rangées sur 8 colonnes. **p** pointe sur un tableau de données au format suivant: le premier caractère est le nombre d'octets à stocker (8 octets par caractère) avec un maximum de 64, les cinq bits de poids faible de chaque octet forment une rangée du caractère de la gauche vers la droite, et la huitième rangée par caractère spécial est la position du curseur.

- **void lc\_stdcg ( )**

charge 8 caractères spéciaux de flèches et de lignes dans l'emplacement pour caractères spéciaux du LCD.

- **void lcd\_init\_printf ( )**

Initialise le LCD avec **lcd\_init**. Initialise également les variables relatives pour permettre de sauver une duplication de l'image de l'écran LCD.

- **void lcd\_putc ( char x )**

Décode **char x** pour une séquence de commandes spéciales pour l'écriture sur les registres de données ou de commandes du LCD. Sert de pilote à **lcd\_printf**. Comme **lc\_putc** sauf que les variables "shadow" pour le LCD sont aussi remise à jour.

- **void lcd\_erase ( )**

Efface complètement le LCD et réinitialise la position du curseur (home). Les variables "shadows" du LCD sont remise à jour.

- **void lcd\_erase\_line ( int line )**

Efface une ligne spécifiée sur le LCD et remet à jour les variables "shadow".

- **void lcd\_printf ( long cursor, char \*fmt, ... )**

C'est la commande similaire à **printf** pour l'écran LCD. Affiche une chaîne à une position de départ spécifiée et laisse le curseur sur une position de fin spécifiée. Les octets du curseur **cursor** sont Y1, X1, Y2, X2, où l'octet le plus significatif, Y1, est le numéro de ligne de départ (0, 1, 2, ou 3); X1 est le numéro de la colonne de départ (0, 1, 2, ...), et Y2 et X2 sont les coordonnées des lignes et colonnes finales. Les quatre bits de poids fort de Y2 sont utilisés pour spécifier l'état final du curseur (1 = on, 0 = off). Seul le positionnement du curseur prend effet si **\*fmt** est une chaîne nulle.

Quand **lcd\_printf** fonctionne, un sémaphore est invoqué pour s'assurer qu'une seule execution soit lancée dans la séquence, malgré que cette fonction puisse être appelée dans plusieurs tâches sans interférence. L'exécution est suspendue pendant dix tops quand le sémaphore est occupé.

Une copie du contenu de l'affichage et l'emplacement du curseur sont remis à jour en mémoire quand **lcd\_printf** imprime sur l'afficheur LCD. **lcd\_savscrn** copie cette image dans une zone spécifiée par l'utilisateur. **lcd\_resscrn** restitue la zone sauvee par l'utilisateur sur l'écran. En utilisant ces routines, une tâche peut interrompre le fil du déroulement et sauver l'affichage courant, utiliser l'affichage dans une nouvelle séquence puis restituer l'affichage original.

- **void lcd\_savscrn ( void\* s )**

Sauve l'image de l'écran LCD dans un vecteur identifié par s.

- **void lcd\_resscrn ( void\* s )**

Restitue sur le LCD l'image stockée dans le vecteur identifié par s.

## **PBUS\_LG.LIB**

Cette librairie contient les fonctions support au PLCBus pour le contrôleur BL1100 et la librairie interface pour les contrôleurs BL1100 et BL1300. La librairie contient les fonctions nécessaires pour accéder aux périphériques PLCBus via le PIO port A du BL1100. La librairie procure également des fonctions PLCBus bas-niveau et haut-niveau pour les cartes d'extension relais et CNA (DAC).

Le bus doit s'interfacer au port PIO comme suit:

PIO pin 0: STB	PIO pin 4: D2
PIO pin 1: A3	PIO pin 5: D3
PIO pin 2: A2	PIO pin 6: D0
PIO pin 3: A1	PIO pin 7: D1

- **void PBus12\_Addr ( int addr )**

Fixe l'adresse courante pour le PLCBus. Toutes les opérations de lecture et d'écriture accéderont à cette adresse tant qu'une nouvelle adresse n'est pas définie. **addr** est l'adresse physique 12-bits dont le premier et le troisième groupe de 4-bits sont inversés (le groupe le plus significatif étant les 4 bits de poids faible).

- **void PBus4\_write ( char data )**

Ecrit une donnée 4 bits sur le PLCBus. L'adresse doit être fixée à l'aide d'un call à **PBus12\_Addr** avant d'appeler cette fonction. **data** doit contenir la valeur à écrire sur les 4 bits de poids faible.

- **int PBus4\_Read0 ( )**

Lit 4 bits de données sur le PLCBus en utilisant un cycle **BUSRD0**. L'adresse doit être fixée à l'aide d'un call à **PBus12\_Addr** avant d'appeler cette fonction. La fonction retourne la donnée PLCBus dans les 4 bits de poids faible (les bits de poids fort étant indéfinis).

- **int PBus4\_Read1 ( )**

Lit 4 bits de données sur le PLCBus en utilisant un cycle **BUSRD1**. L'adresse doit être fixée à l'aide d'un call à **PBus12\_Addr** avant d'appeler cette fonction. La fonction retourne la donnée PLCBus dans les 4 bits de poids faible (les bits de poids fort étant indéfinis).

- **int PBus4\_ReadSp ( )**

Lit 4 bits de données sur le PLCBus en utilisant un cycle **BUSSPARE**. L'adresse doit être fixée à l'aide d'un call à **PBus12\_Addr** avant d'appeler cette fonction. La fonction retourne la donnée PLCBus dans les 4 bits de poids faible (les bits de poids fort étant indéfinis).

- **int Relay\_Board\_Addr ( int board )**

Convertit une adresse logique de carte relais en adresse physique PLCBus. **board** doit être un nombre entre 0 et 63, et représenter la carte relais sur laquelle accéder. Ce nombre à la forme binaire *pppzyx* où **ppp** est déterminé par le numéro de PAL de la carte et **x**, **y** et **z** sont déterminés par le cavalier J1 sur la carte. Les valeurs de **ppp** comme 000, 001, 010, ...etc, correspondent aux numéros de PAL FPO4500, FPO4510, FPO4520, ...etc; **x**, **y** et **z** correspondent respectivement au cavalier J1 pins 1-2, 3-4, et 5-6 (0 = fermé, 1 = ouvert). L'adresse résultante est de la forme **pppx000y000z**. La fonction retourne l'adresse PLCBus de la carte spécifiée, avec le premier et le troisième groupe de bits intervertis; Cette adresse peut être passée directement à **PBus12\_Addr**.

- **void set\_PBus\_Relax ( int board, int relay, int state )**

Fixe un relai sur une carte d'extension relais. **board** doit être un nombre entre 0 et 63, et représenter la carte relais sur laquelle accéder. Ce nombre à la forme binaire *pppzyx* où **ppp** est déterminé par le numéro de PAL de la carte et **x**, **y** et **z** sont déterminés par le cavalier J1 sur la carte. Les valeurs de **ppp** comme 000, 001, 010, ...etc, correspondent aux numéros de PAL FPO4500, FPO4510, FPO4520, ...etc; **x**, **y** et **z** correspondent respectivement au cavalier J1 pins 1-2, 3-4, et 5-6 (0 = fermé, 1 = ouvert). L'adresse résultante est de la forme **pppx000y000z**. **relay** est le numéro de relai de la carte (0 à 5 pour XP8300, 0 à 7 pour XP8400). **state** doit être égal à 1 pour commuter le relai ON et égal à 0 pour commuter le relai OFF.

- **int DAC\_Board\_Addr ( int bd )**

Convertit une adresse logique de carte CNA (DAC) en une adresse physique PLCBus. **bd** doit être un nombre compris entre 0 et 63 et représenter la carte CNA sur laquelle accéder. Ce nombre a la forme binaire *pppzyx* où **ppp** est déterminé par le numéro de PAL de la carte et **x**, **y**, et **z** sont déterminés par le cavalier J3 sur la carte. Les valeurs *ppp* comme 000, 001, 010, ...etc, correspondent aux numéros de PAL FPO4800, FPO4810, FPO4820, ...etc ; **x**, **y** et **z** correspondent respectivement au cavalier J3 pins 1-2, 3-4, et 5-6 (0 = fermé, 1 = ouvert). L'adresse résultante est de la forme **pppx001y000z**. La fonction retourne l'adresse PLCBus de la carte spécifiée, avec le premier et le troisième groupe de bits intervertis; Cette adresse peut être passée directement à **PBus12\_Addr**.

- **void Write\_DAC1 ( int val )**

Charge le registre A de DAC#1 avec la valeur 12 bits donnée. L'adresse de la carte doit avoir été fixé au préalable à l'aide d'un appel à **PBus12\_Addr**. La valeur contenue dans **val** ne sera pas effective en sortie tant que **Latch\_DAC1** n'est pas lancé.

- **void Write\_DAC2 ( int val )**

Charge le registre A de DAC#2 avec la valeur 12 bits donnée. L'adresse de la carte doit avoir été fixé au préalable à l'aide d'un appel à **PBus12\_Addr**. La valeur contenue dans **val** ne sera pas effective en sortie tant que **Latch\_DAC2** n'est pas lancé.

- **void Latch\_DAC1 ( )**

Décale la valeur du registre A de DAC1 dans le registre B. La valeur dans le registre B représente la sortie actuelle du DAC. L'adresse de la carte doit avoir été fixé au préalable à l'aide d'un appel à **PBus12\_Addr**, et la valeur doit avoir été chargée dans le registre A avec un appel à **Write\_DAC1**.

- **void Latch\_DAC2 ( )**

Décale la valeur du registre A de DAC2 dans le registre B. La valeur dans le registre B représente la sortie actuelle du DAC. L'adresse de la carte doit avoir été fixé au préalable à l'aide d'un appel à **PBus12\_Addr**, et la valeur doit avoir été chargée dans le registre A avec un appel à **Write\_DAC2**.

- **void Init\_DAC ( )**

Initialise la carte DAC et fixe toutes les valeurs de sortie à 0. Appeler cette fonction avant d'écrire une donnée sur le DAC. L'adresse de la carte doit avoir été fixé au préalable à l'aide d'un appel à **PBus12\_Addr**.

- **void Set\_DAC1 ( int val )**

Fixe le DAC1 avec la valeur spécifiée dans les 12 bits de poids faible de **val**. En mode sortie-tension (pins 2-3 de J1 reliées),  $V_{out} = ( val / 4096 ) \times 10,22$  volts en configuration Z-World par défaut. En mode sortie-courant ( pins 1-2 de J1 reliées),  $I_{out} = ( val / 4096 ) \times 22$  milliampères en configuration Z-World par défaut. L'adresse de la carte doit avoir été fixé au préalable à l'aide d'un appel à **PBus12\_Addr**.

- **void Set\_DAC2 ( int val )**

Fixe le DAC2 avec la valeur spécifiée dans les 12 bits de poids faible de **val**. En mode sortie-tension (pins 2-3 de J1 reliées),  $V_{out} = ( val / 4096 ) \times 10,22$  volts en configuration Z-World par défaut. En mode sortie-courant ( pins 1-2 de J1 reliées),  $I_{out} = ( val / 4096 ) \times 22$  milliampères en configuration Z-World par défaut. L'adresse de la carte doit avoir été fixé au préalable à l'aide d'un appel à **PBus12\_Addr**.

- **void DAC\_On ( )**

Contrôle la ligne d'activation du commutateur haut. Utilisé seulement avec l'option commutateur U10 - LT1188.

- **void DAC\_Off ( )**

Contrôle la ligne d'activation du commutateur haut. Utilisé seulement avec l'option commutateur U10 - LT1188.

- **void Reset\_PBus ( )**

Réinitialise le PLCBus.

- **int Poll\_PBus\_Node ( int addr )**

Scrute un périphérique PLCBus en effectuant un cycle **BUSRD0** et en vérifiant le bit de poids faible de la valeur renvoyée. **addr** est l'adresse physique 12 bits du périphérique, dont le premier et le troisième groupe de 4-bits sont inversés. La fonction retourne 1 si **node** répond à la scrutation, sinon la fonction retourne 0.

- **void Reset\_PBus\_Wait ( )**

Produit le délai minimum de temps nécessaire après une réinitialisation du bus des cartes d'extension PLCBus (sous-entend l'utilisation d'un CPU 9 MHz). Ce délai peut s'avérer insuffisant pour des CPU plus rapides et doit donc être rallongé.

## **PBUS\_TG.LIB**

Ces fonctions supportent les contrôleurs de la famille BL1000. La librairie interface PLCBus est procurée pour le BL1000. Cette librairie contient les fonctions nécessaires pour accéder aux périphériques PLCBus via le port B du PIO du BL1000. La librairie procure des fonctions PLCBus bas-niveau ainsi que des fonctions haut-niveau pour les relais et les cartes d'extension DAC.

Le bus doit s'interfacer au port PIO comme suit.

PIO pin 0: D1	PIO pin 4: A1
PIO pin 1: D0	PIO pin 5: A2
PIO pin 2: D3	PIO pin 6: A3
PIO pin 3: D2	PIO pin 7: STB

- **void PBus12\_Addr ( int addr )**

Fixe l'adresse courante pour le PLCBus. Toutes les opérations de lecture et d'écriture, accéderont à cette adresse jusqu'à ce qu'une nouvelle adresse soit fixée. **addr** est l'adresse physique 12 bits avec le premier et le troisième groupe de 4-bits inversés (le groupe le plus significatif se trouve dans les quatre bits du bas). La fonction ne retourne rien.

- **void PBus4\_Write ( char data )**

Ecrit une donnée 4-bits sur le PLCBus. L'adresse doit être fixée par un appel à **PBus12\_Addr** avant d'appeler cette fonction. **data** doit contenir la valeur à écrire dans les 4 bits de poids faible.

- **int PBus4\_Read0 ( )**

Lit 4 bits de donnée sur le PLCBus en utilisant un cycle **BUSRD0**. L'adresse doit être fixée par un appel à **PBus12\_Addr** avant d'appeler cette fonction. La fonction retourne la donnée PLCBus dans les 4 bits de poids faible (les bits de poids fort sont indéfinis).

- **int PBus4\_Read1 ( )**

Lit 4 bits de donnée sur le PLCBus en utilisant un cycle **BUSRDI**. L'adresse doit être fixée par un appel à **PBus12\_Addr** avant d'appeler cette fonction. La fonction retourne la donnée PLCBus dans les 4 bits de poids faible (les bits de poids fort sont indéfinis).

- **int PBus4\_ReadSp ( )**

Lit 4 bits de donnée sur le PLCBus en utilisant un cycle **BUSSPARE**. L'adresse doit être fixée par un appel à **PBus12\_Addr** avant d'appeler cette fonction. La fonction retourne la donnée PLCBus dans les 4 bits de poids faible (les bits de poids fort sont indéfinis).

- **int Relay\_Board\_Addr ( int board )**

Convertit une adresse logique de carte relai en adresse physique PLCBus. **board** doit être un nombre compris entre 0 et 63, et représente la carte relai sur laquelle accéder. Ce nombre a la forme binaire **pppzyx** où **ppp** est déterminé par le numéro de PAL de la carte et **x**, **y** et **z** sont déterminés par le cavalier J1 de la carte. Les valeurs **ppp** 000, 001, 010, ...etc, correspondent aux numéros de PAL de FPO4500, FPO4510, FPO4520, ...etc; **x**, **y** et **z** correspondent respectivement au cavalier J1 pins 1-2, 3-4, et 5-6 (0 = fermé, 1 = ouvert). L'adresse résultante est de la forme **pppx000y000z**. La fonction retourne l'adresse PLCBus de la carte spécifiée, avec le premier et le troisième groupe de 4-bits inversés; cette adresse peut être passée directement à **PBus12\_Addr**.

- **void Set\_PBus\_Relay ( int board, int relay, int state )**

Fixe un relai sur une carte d'extension relai PLCBus. **board** doit être un nombre compris entre 0 et 63, et représente la carte relai sur laquelle accéder. Ce nombre a la forme binaire **pppzyx** où **ppp** est déterminé par le numéro de PAL de la carte et **x**, **y** et **z** sont déterminés par le cavalier J1 de la carte. Les valeurs **ppp** 000, 001, 010, ...etc, correspondent aux numéros de PAL de FPO4500, FPO4510, FPO4520, ...etc; **x**, **y** et **z** correspondent respectivement au cavalier J1 pins 1-2, 3-4, et 5-6 (0 = fermé, 1 = ouvert). **relay** est le numéro de relai de la carte (0-5 pour la carte XP8300; 0-7 pour la carte XP8400). **state** doit être égal à 1 pour commuter le relai ON et égal à 0 pour commuter le relai OFF.

- **int DAC\_Board\_Addr ( int bd )**

Convertit une adresse logique de carte CNA (DAC) en adresse physique PLCBus. **bd** doit être un nombre compris entre 0 et 63, et représente la carte CNA sur laquelle accéder. Ce nombre a la forme binaire **pppzyx** où **ppp** est déterminé par le numéro de PAL de la carte et **x**, **y** et **z** sont déterminés par le cavalier J3 de la carte. Les valeurs **ppp** 000, 001, 010, ...etc, correspondent aux numéros de PAL de FPO4800, FPO4810, FPO4820, ...etc; **x**, **y** et **z** correspondent respectivement au cavalier J3 pins 1-2, 3-4, et 5-6 (0 = fermé, 1 = ouvert). L'adresse résultante est de la forme **pppx001y000z**. La fonction retourne l'adresse PLCBus de la carte spécifiée, avec le premier et le troisième groupe de 4-bits inversés; cette adresse peut être passée directement à **PBus12\_Addr**.

- **void Write\_DAC1 ( int val )**

Charge le registre A de DAC#1 avec la valeur 12 bits donnée. L'adresse de la carte doit avoir été fixé au préalable à l'aide d'un appel à **PBus12\_Addr**. La valeur contenue dans **val** ne sera pas effective en sortie tant que **Latch\_DAC1** n'est pas lancé.

- **void Write\_DAC2 ( int val )**

Charge le registre A de DAC#2 avec la valeur 12 bits donnée. L'adresse de la carte doit avoir été fixé au préalable à l'aide d'un appel à **PBus12\_Addr**. La valeur contenue dans **val** ne sera pas effective en sortie tant que **Latch\_DAC2** n'est pas lancé.

- **void Latch\_DAC1 ( )**

Décalle la valeur du registre A de DAC1 dans le registre B. La valeur dans le registre B représente la sortie actuelle du DAC. L'adresse de la carte doit avoir été fixé au préalable à l'aide d'un appel à **PBus12\_Addr**, et la valeur doit avoir été chargée dans le registre A avec un appel à **Write\_DAC1**.

- **void Latch\_DAC2 ( )**

Décalle la valeur du registre A de DAC2 dans le registre B. La valeur dans le registre B représente la sortie actuelle du DAC. L'adresse de la carte doit avoir été fixé au préalable à l'aide d'un appel à **PBus12\_Addr**, et la valeur doit avoir été chargée dans le registre A avec un appel à **Write\_DAC2**.

- **void Init\_DAC ( )**

Initialise la carte DAC et fixe toutes les valeurs de sortie à 0. Appeler cette fonction avant d'écrire une donnée sur le DAC. L'adresse de la carte doit avoir été fixé au préalable à l'aide d'un appel à **PBus12\_Addr**.

- **void Set\_DAC1 ( int val )**

Fixe le DAC1 avec la valeur spécifiée dans les 12 bits de poids faible de **val**. En mode sortie-tension (pins 2-3 de J1 reliées),  $V_{out} = ( val / 4096 ) \times 10,22$  volts en configuration Z-World par défaut. En mode sortie-courant ( pins 1-2 de J1 reliées),  $I_{out} = ( val / 4096 ) \times 22$  milliampères en configuration Z-World par défaut. L'adresse de la carte doit avoir été fixé au préalable à l'aide d'un appel à **PBus12\_Addr**.

- **void Set\_DAC2 ( int val )**

Fixe le DAC2 avec la valeur spécifiée dans les 12 bits de poids faible de **val**. En mode sortie-tension (pins 2-3 de J1 reliées),  $V_{out} = ( val / 4096 ) \times 10,22$  volts en configuration Z-World par défaut. En mode sortie-courant ( pins 1-2 de J1 reliées),  $I_{out} = ( val / 4096 ) \times 22$  milliampères en configuration Z-World par défaut. L'adresse de la carte doit avoir été fixé au préalable à l'aide d'un appel à **PBus12\_Addr**.

- **void DAC\_On ( )**

Contrôle la ligne d'activation du commutateur haut. Utilisé seulement avec l'option commutateur U10 - LT1188.

- **void DAC\_Off ( )**

Contrôle la ligne d'activation du commutateur haut. Utilisé seulement avec l'option commutateur U10 - LT1188.

- **void Reset\_PBus ( )**

Réinitialise le PLCBus.

- **int Poll\_PBus\_Node ( int addr )**

Scrute un périphérique PLCBus en effectuant un cycle **BUSRDO** et en vérifiant le bit de poids faible de la valeur renvoyée. **addr** est l'adresse physique 12 bits du périphérique, dont le premier et le troisième groupe de 4-bits sont inversés. La fonction retourne 1 si **node** répond à la scrutation, sinon la fonction retourne 0.

- **void Reset\_PBus\_Wait ( )**

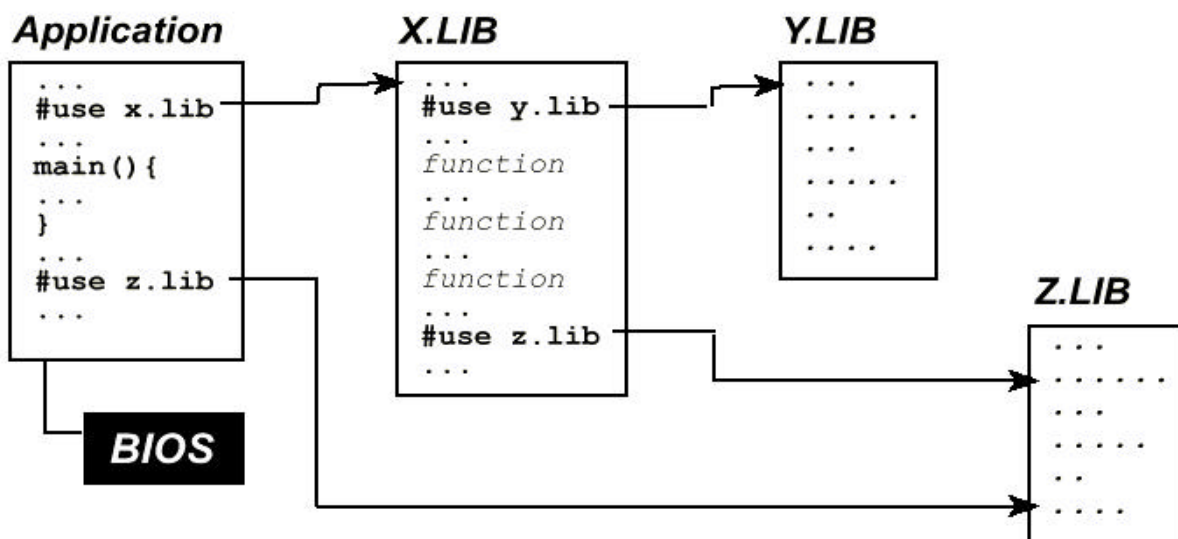
Produit le délai minimum de temps nécessaire après une réinitialisation du bus des cartes d'extension PLCBus (sous-entend l'utilisation d'un CPU 9 MHz). Ce délai peut s'avérer insuffisant pour des CPU plus rapides et doit donc être rallongé.

## ANNEXE A - LIBRAIRIES DYNAMIC C

Les bibliothèques décrites au chapitre 1 incluent des fonctions mathématiques et des chaînes Dynamic C standards en supplément des fonctions spécifiques de support général aux contrôleurs Z-World.

Les bibliothèques de fonctions Dynamic C procurent une solution pour n'embarquer seulement que celles utilisées dans l'application de l'utilisateur. Le fichier **LIB.DIR** contient une liste de toutes les bibliothèques Dynamic C connues. Cette liste peut être modifiée par l'utilisateur. Et en particulier, toute bibliothèque créée par un utilisateur peut être ajoutée à cette liste.

Les bibliothèques sont "liées" avec une application utilisateur grâce à la directive **#use**. Les fichiers identifiés par les directives **#use** peuvent s'imbriquer comme le montre la figure suivante.



Le fichier **DEFAULT.H** contient plusieurs listes de bibliothèques à utiliser (**#use**), une liste pour chaque produit que Z-World livre. Le Dynamic C normalement, connaît le contrôleur en train d'être utilisé, il sélectionne donc les bibliothèques appropriées à ce contrôleur. Ces listes sont les listes *par défaut*. Un programmeur peut trouver pratique ou nécessaire d'ajouter ou de supprimer des bibliothèques dans certaines listes.

Les bibliothèques par défaut pour un contrôleur Z-World contiennent des tas de noms de fonctions, de noms de variables globales et en particulier, des tas de noms de macro. C'est comme si un programmeur essayait d'utiliser un des noms Z-World dans un nouveau programme. Des problèmes imprévisibles peuvent survenir. Z-World recommande d'éditer le fichier **DEFAULT.H** et de mettre en commentaire les bibliothèques inutiles.

La table suivante donne la liste des bibliothèques incluent avec le Dynamic C. D'autres bibliothèques existent mais seulement pour assurer une compatibilité descendante: **LSTAR.LIB**, **MICROG.LIB**, **LGIANT.LIB**, **RG.LIB**, **SCOREZ1.LIB**, **LPLC.LIB** et **PS.LIB**.

NOM	DESCRIPTION
5KEY.LIB	Système de base "five-key system" pour les contrôleurs des séries PK2100 et PK2200.
5KEYEXTD.LIB	Extensions du "five-key system".
96IO.LIB	Fonctions pilote pour la carte fille DGL96 du BL1100.
AASC.LIB	Fonctions abstraites de communication série asynchrone.
AASCDIO.LIB	Routines spécifiques STDIO supportant la librairie AASC.
AASCSCC.LIB	Routines spécifiques SCC supportant la librairie AASC. Le SCC est le contrôleur de communication série Zilog 85C30.
AASCUART.LIB	Fonctions support à la série XP8700 pour la librairie AASC. Le XP8700 est la carte d'extension RS232 pour PLC bus.
AASCZ0.LIB	Routines spécifiques Z0 pour supporter la librairie AASC. Z0 est le port série 0 de l'ASCII du Z180.
AASCZ1.LIB	Routines spécifiques Z1 pour supporter la librairie AASC. Z1 est le port série 1 de l'ASCII du Z180.
AASCZN.LIB	Routines spécifiques ZNet pour supporter la librairie AASC.
BIOS.LIB	Contient les prototypes de fonctions et déclarations de variables définies et utilisées par le BIOS.
BL1000.LIB	Fonctions pour le contrôleur BL1000.
BL11xx.LIB	Fonctions pour le contrôleur BL1100.
BL12xx.LIB	Librairie vide.
BL13xx.LIB	Fonctions pour le contrôleur BL1300.
BL14_15.LIB	Fonctions pour les contrôleurs séries BL1400 et BL1500.
BL16xx.LIB	Fonctions pour le contrôleur BL1600.
CIRCBUF.LIB	Fonctions de types de données abstraites pour les buffers circulaires (utilisées par le pilote AASC).
CM71_72.LIB	Fonctions pour les modules coeur séries CM7100 et CM7200. Anciennement Smartcores Z1 et Z2.
CPLC.LIB	Fonctions pour les contrôleurs PK2100, PK2200 et BL1600.
DC.HH	Ce fichier contient les définitions basiques et nécessaires au Dynamic C. Fichier nécessaire.
DEFAULT.H	Contient la liste des directives #use pour divers contrôleurs Z-World. Le Dynamic C sélectionne automatiquement la liste appropriée en fonction du contrôleur utilisé.
DMA.LIB	Fonctions support pour les canaux DMA du Z180.
DRIVERS.LIB	Fonctions pilotes pour quelques périphériques matériels.
EZIO.LIB	Fonctions pilotes pour un espace d'E/S unifié, indépendant des cartes.
EZIOCMMN.LIB	Définitions communes à toutes les librairies EZIO.
EZIOPBDV.LIB	Pilotes de périphériques PLCBus supportant la librairie EZIO.
EZIOPK23.LIB	Support aux fonctions PK2300 pour la librairie EZIO.
EZIOPLC.LIB	Fonctions PLCBus pour cartes avec port PLCBus natif (séries BL1200, BL1600, PK2100 et PK2200).

NOM (SUITE)	DESCRIPTION (SUITE)
FK.LIB	Nouveau "five-key system" supporté par les contrôleurs séries PK2100 et PK2200. A utiliser en multitâche coopératif (par exemple les costatements).
IOEXPAND.LIB	Fonctions pilote pour les cartes filles de la série BL1100.
KDM.LIB	Fonctions pilote pour les KDM Z-World (Keypad/Display Modules).
LCD2L.LIB	Support aux afficheurs LCD 2 lignes pour les séries PK2100 et PK2200.
MATH.LIB	Fonctions mathématiques et trigonométriques utiles.
MISC.LIB	Fonctions diverses pour le support KDM.
MODEM232.LIB	Fonctions MODEM pour les séries PK2100 et PK2200. Utilisé avec Z0232.LIB, S0232.LIB, XP87XX.LIB, NETWORK.LIB et SCC232.LIB.
NETWORK.LIB	Protocole binaire 9-bits OPTO22 pour support de réseaux maître-esclave. Utilise ASCI port 1 du Z180.
PBUS_LG.LIB	Fonctions de gestion du BL1100 avec le PLCBus.
PBUS_TG.LIB	Fonctions de gestion du BL1100 avec le PLCBus.
PK21XX.LIB	Fonctions pour les contrôleurs séries PK2100.
PK22XX.LIB	Fonctions pour les contrôleurs séries PK2200.
PLC_EXP.LIB	Fonctions PLCBus pour les cartes avec PLCBus natif (séries BL1200, BL1600, PK2100 et PK2200).
PRPORT.LIB	Fonctions implémentant un protocole de communication port parallèle entre un contrôleur et un PC.
PWM.LIB	Fonctions de PWM (Pulse Width modulation).
RTK.LIB	Noyau temps réel (RTK).
S0232.LIB	Pilote de communication série pour SIO port 0 sur les contrôleurs BL1100.
S1232.LIB	Pilote de communication série pour SIO port 1 sur les contrôleurs BL1100.
SCC232.LIB	Pilote de communication série pour les ports du composant SCC (contrôleur de communication série Zilog 85C30).
SRTK.LIB	Noyau temps réel simplifié pour tous les contrôleurs.
STDIO.LIB	Fonctions relatives à la fenêtre STDIO du Dynamic C.
STRING.LIB	Ce fichier contient les fonctions de manipulation de chaînes.
SYS.LIB	Fonctions systèmes générales.
VDRIVER.LIB	Fonctions du pilote virtuel (pour tout contrôleur).
XMEM.LIB	Fonctions de gestion de la mémoire étendue (calcul d'adresse, mouvement d'information, ...etc).
XP82XX.LIB	Fonctions pilote pour les cartes XP8200 au PLCBus.
XP87XX.LIB	Fonctions de communication série pour une carte XP8700 au PLCBus.
XP87XX2.LIB	Fonctions de communication série pour une seconde carte XP8700 (voir ci-dessus).
XP88XX.LIB	Fonctions pour les cartes XP8800 au PLCBus.
Z0232.LIB	Pilote de communication série pour Z0. Z0 est le port série ASCI n°0 du Z180.
Z1232.LIB	Pilote de communication série pour Z1. Z1 est le port série ASCI n°1 du Z180.
ZNPAKFNT.LIB	Fonctions bas-niveau supportant le ZNet.

## **ANNEXE B - UTILISATION DES LIBRAIRIES AASC**

La librairie AASC (Abstract Application-level Serial Communication) et ses fonctions support bas-niveau facilitent la communication série entre les contrôleurs et entre un contrôleur et un autre périphérique comme un PC.

### **Description de la librairie AASC**

Les librairies AASC permettent au programmeur de créer des flux de caractères bufferisés qui effectuent des entrées/sorties sur les ports des périphériques de communication. Une librairie principale **AASC.LIB**, contient toutes les fonctions nécessaires à ces tâches. La table ci-dessous liste les librairies support utilisées avec **AASC.LIB**.

<b>Librairie Pilote</b>	<b>Description</b>
AASCDIO.LIB	Contient les routines d'entrées/sorties STDIO pour supporter les librairies AASC.
AASCSCC.LIB	Fait fonctionner les canaux du contrôleur de communication série du Zilog 85C30 utilisés sur les contrôleurs BL1100 et BL1700.
AASCUART.LIB	Fait fonctionner le port RS232 de la carte d'extension PLCBus XP8700 supportée par la plupart des contrôleurs Z-World.
AASCUART2.LIB	Fait fonctionner le port RS232 de la carte d'extension PLCBus XP8700 supportée par la plupart des contrôleurs Z-World en adressage 16 bits du PLCBus.
AASCZ0.LIB	Se charge de la communication sur le port Z0 du Z180 utilisé par les contrôleurs Z-World. Ce port est d'habitude connecté à un pilote RS232.
AASCZ1.LIB	Se charge de la communication sur le port Z0 du Z180 utilisé par les contrôleurs Z-World. Ce port est d'habitude connecté à un pilote RS485.
AASCZN.LIB	Fait fonctionner les routines spécifiques ZNet sur le réseau RS485. Tous les contrôleurs participants doivent avoir le même pilote. Un contrôleur est désigné comme contrôleur maître en définissant la macro ZNMASTER comme non nulle avant d'invoquer #use AASCZN.LIB. Cette librairie utilise le port Z1 du Zilog Z180.

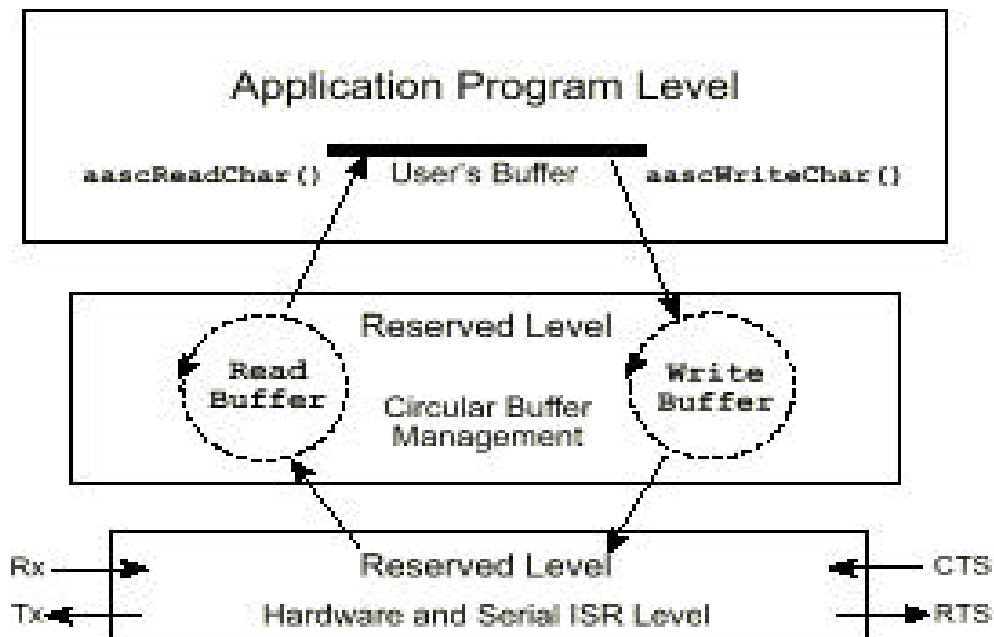
Les librairies AASC sont autant que faire se peut indépendantes des périphériques. Les programmes incluent seulement le code **AASC.LIB** et le code requis par les périphériques de communication dans l'application (par exemple **AASCSCC.LIB**). L'application prend en main les différents périphériques de communication en créant simplement des voies séparées.

Deux buffers circulaires cachés pour chaque voie AASC stockent les informations entrantes et sortantes. Ceci permet à l'application de traiter les informations entrantes et sortantes par paquets pas plus grands que les buffers circulaires. La taille du buffer est spécifiée dans l'application.

Les librairies support AASC implémentent les pilotes de périphériques personnalisés et les routines de service interruption (ISR) pour chaque périphérique de communication. L'application nécessite seulement d'initialiser une voie et un buffer local, puis d'effectuer des appels aux fonctions pour vérifier l'état des buffers et lire/écrire les buffers.

## Fonctionnement de la librairie AASC

Les librairies AASC lisent (reçoivent), écrivent (transmettent), piquent (cherchent), procurent des états et gèrent des erreurs. La figure ci-dessous montre la hiérarchie de ces fonctions AASC. A noter que la gestion des buffers circulaires et des niveaux d'interruptions matériel/série est masquée pour le programmeur. Ces deux niveaux réservés sont contenus dans les librairies support listées au tableau précédent.



### Lecture

L'information est reçue soit par bloc, soit par octet. Une seule méthode est nécessaire, mais l'autre peut toujours être implémentée. Il est toujours plus efficace d'avoir les deux méthodes de disponible. La fonction de lecture bloc supporte les lectures de taille fixe ou variable. L'application peut lire exactement  $n$  octets, elle peut lire rien du tout, ou elle peut lire jusqu'à  $n$  octets. Dans tous les cas, la fonction retourne le nombre d'octets actuels lus.

**IMPORTANT:** Les opérations de lecture peuvent préempter les opérations d'écriture et vice-versa, mais une opération de lecture ou d'écriture ne peut pas préempter une autre opération de lecture ou d'écriture.

### Ecriture

Les routines de transmission (écriture) sont les images miroir des fonction de lecture. Il y à une fonction pour l'écriture d'octet et une autre fonction pour l'écriture de bloc. La fonction d'écriture de bloc peut écrire une partie de bloc, rien du tout ou un bloc entier. Ceci est important pour les programmes multi traitements car écrire tout ou rien du tout empêche les chassés croisés de messages originaires de différents traitements coopératifs.

### Piquage

Une fonction spéciale supportée par les librairies AASC permet à l'application de "piquer" dans le buffer sans extraire un octet. La fonction de piquage `aascPeek` recherche par exemple une sous-chaîne pour identifier le type de paquet entrant, sans changer le contenu du buffer. Un autre type de fonction "piquer", `aascScanTerm`, peut aussi rechercher un caractère particulier comme le caractère de terminaison d'un paquet.

## Etats et erreurs

Les bibliothèques AASC procurent un rapport complet des états de l'application. Les bibliothèques peuvent rapporter le nombre d'octets utilisés et le nombre d'octets encore libres dans les buffers d'écriture ou de lecture. De telles informations sont utiles à l'application pour planifier la vérification des messages ou la transmission dynamique. Les bibliothèques AASC peuvent aussi rapporter à la fois les erreurs matériels (par exemple les erreurs de trame ou de parité) et les erreurs logicielles (par exemple dépassement de buffer). Les conditions d'erreurs ne sont pas effacées automatiquement.

## Utilisation de la bibliothèque

Lors de l'utilisation des bibliothèques AASC, suivre ces six étapes:

1. Identifier le périphérique de communication (par exemple Z0, SCC canal A, UART).
2. Allouer et initialiser le canal avec **aascOpen ( )**.
3. Paramétrer les buffers circulaires de lecture (réception) et d'écriture (émission). Par exemple en utilisant **aascSetReadBuf ( )**.
4. Retirer les lectures et les écritures (utiliser par exemple **aascWriteChar ( )**).
5. Vérifier l'état et relever les erreurs (utiliser par exemple **aascGetError ( )**).
6. Une fois terminé, fermer le canal avec **aascClose ( )**.

## Exemple de programme

Le programme d'exemple suivant propose une façon d'utiliser la structure AASC dans une communication série asynchrone avec un terminal. Le programme démontre comment utiliser le port SCC canal A comme un périphérique AASC. D'autres programmes figurent dans le sous répertoire Dynamic C **SAMPLES / AASC**. Ce programme simplement duplique le texte tapé sur un terminal ascii. Connecter le contrôleur avec un circuit contrôleur de communication série par le SCC canal A sur un PC ou autre petit terminal. A l'aide d'un PC, l'utilitaire Windows **terminal.exe** doit être utilisé avec **ANSI terminal emulation** en mode **Local Echo** désactivé et **Flow Control** fixé à **none**. Si l'acquiescement RTS/CTS est validé en fixant la macro **SHAKE** non nulle, valider **Flow Control** dans **terminal.exe** sur **Hardware**. Ce programme exemple nécessite de fixer par défaut: pas de parité, un bit de stop et huit bits de données. Paramétrez votre PC en conséquence.

Les étapes suivantes décrivent comment ce traitement "écho" fonctionne.

1. Le programme accède à **AASC.LIB** et à la bibliothèque AASC appropriée **AASCSCC.LIB** avec **#use**.
2. Les définitions sont créées pour les buffers circulaires de lecture et d'écriture, ainsi que pour le buffer utilisateur **workBuffer**. Un pointeur de buffer utilisateur **pworkBuffer**, est également créé dans cet exemple.
3. **\_GLOBAL\_INIT ( )** est appelé pour initialiser la structure AASC.
4. La fonction **aascOpen ( )** est utilisée pour créer un canal sur le périphérique **DEV\_SCC** en 8N1.
5. Pour être sûr, le programme vérifie si c'est bien un contrôleur avec circuit SCC qui est en train d'être utilisé.
6. Le transmetteur et le récepteur du canal **chan** sont enclenchés par **aascTxSwitch ( )** et par **aascRxSwitch ( )**.
7. Le programme prépare les buffers circulaires avec **aascReadBuf** et **aascSetWriteBuf**.
8. Si un caractère est lu, le programme rentre dans une autre boucle qui envoie les caractères de **workBuffer** sur le terminal. La fonction ne retourne pas tant que tous les caractères lus dans **workBuffer** ne sont pas envoyés sur le terminal.

## SCCECHO.C

```
#use aasc.lib
#use aascsc.lib

#define BUFSIZE 684           // Taille du buffer circulaire
#define BAUDMULT 8           // Multiples de 1200 bps. (8 x 1200 bps = 9600 bps)
#define SHAKE 0              // Fixer à 1 pour valider l'acquitement RTS/CTS

char readBuffer [ BUFSIZE ] , writeBuffer [ BUFSIZE ] ;
char workBuffer [ BUFSIZE ] , *pworkBuffer ;
struct _Channel *aascChannel ;

main () {

    _GLOBAL_INIT () ;    // Cela doit être la première chose à faire dans main ()

    // Ouvre le canal A du SCC à 8N1

    aascChannel = aascOpen ( DEV_SCC , SHAKE ,
        SCC_A | SCC_1STOP | SCC_NOPARITY | SCC_8DATA |
        SCC_1200*BAUDMULT , NULL ) ;

    if ( aascChannel==NULL ) {
        printf ( "SCC canal A non disponible." ) ;
        return ;
    }

    // Paramétrage des buffers circulaires

    aascSetReadBuf ( aascChannel , readBuffer ,
        sizeof ( readBuffer ) ) ;
    aascSetWriteBuf ( aascChannel , writeBuffer ,
        sizeof ( writeBuffer ) ) ;

    // Traite le transfert des données

    while ( 1 ) {
        hitwd () ;

        // Effectue le transfert des données

        if ( aascReadChar ( aascChannel , workBuffer ) ) {
            while ( !aascWriteChar ( aascChannel ,
                workBuffer [ 0 ] ) ) {
                hitwd () ;
            }
        }
    }
}
```

## Transfert XModem

Les bibliothèques AASC ont un support étendu à l'utilisation du protocole de transfert **XModem-CRC**. Les bibliothèques AASC permettent à l'application de définir des fonctions de rappel pour lire ou écrire chaque bloc d'un paquet XModem. Cela veut dire que l'on a pas besoin d'avoir le bloc entier prêt avant la transmission, ou d'alouer un espace pour le bloc entrant entier. Des fonctions de rappel par défaut sont fournies pour les opérations classiques de lecture mémoire ou d'écriture à partir de la mémoire.

### Utilisation de la bibliothèque

1. Initialiser le driver virtuel.
2. Initialiser la structure AASC avec un périphérique approprié comme le canal A du SCC.
3. Initialiser un buffer de données XModem et le nombre d'octets à transférer avec **aascXMWrInitPhy ( )** ou **aascXMRdInitPhy ( )** pour la mémoire physique, ou **aascXMWrInitLog ( )** ou **aascXMRdInitLog ( )** pour la mémoire logique.
4. Initialiser le transfert XModem avec **aascWriteXModem ( )** ou **aascReadXModem ( )**.
5. Effectuer le transfert XModem avec **aascWriteXModem ( )** ou **aascReadXModem ( )**.

### Programme d'exemple

Le programme suivant donne un exemple de structure AASC dans un transfert de données XModem. Le programme envoie un bloc de 128 caractères sur un périphérique distant en utilisant **XModem - CRC**. Configurer le périphérique distant pour 9600 Bauds en 8N1 sans contrôle de flux RTS/CTS. Le pilote virtuel doit être utilisé car Xmodem incorpore des *costatements* pour permettre le multitâche. Il est à noter que n'importe quel canal peut être utilisé en changeant SCC canal A par le port désiré. Par exemple, pour utiliser le port Z1 du Z180, mettre la bibliothèque **AASCZ1.LIB** à la place de **AASCSCC.LIB**, et changer les paramètres dans **aascOpen ( )** pour adopter ceux de Z1.

Les étapes suivantes décrivent un exemple de transmission XModem:

1. Le programme accède aux bibliothèques nécessaires par **#use**.
2. Les définitions sont créées pour les buffers circulaires d'écriture et de lecture ainsi que pour le buffer XModem.
3. **aascInit ( )** est appelé pour initialiser la structure AASC.
4. Une chaîne de données est créée pour le transfert.
5. **VdInit ( )** est appelée pour initialiser le driver virtuel.
6. **aascOpen ( )** est utilisé pour créer un canal sur le périphérique **SCC\_A** à 8N1 et 9600 bauds.
7. Le programme vérifie la présence du composant SCC sur le contrôleur.
8. Les buffers circulaires sont ensuite initialisés par **aascSetReadBuf ( )** et par **aascSetWriteBuf ( )** et deviennent accessibles pour la structure AASC.
9. La transmission XModem est ensuite effectuée par appels répétitifs à **aascWriteXModem ( )** avec le paramètre d'initialisation fixé à 0.
10. La transmission XModem se termine quand **aascWriteXModem ( )** retourne un 1.

## XM\_SEND.C

```
#use vdriver.lib
#use aasc.lib
#use aascscclib

#define BUFSIZE 1024 // Taille du buffer circulaire
#define BAUDMULT 8 // Multiples de 1200 bps. (8 x 1200 bps = 9600 bps)

struct _Channel *aascChannel ;
char circBufIn [BUFSIZE] , circBufOut [BUFSIZE] ;
char aascBuffer [BUFSIZE] ;

int aascInit (void) ;

void main (void) { // Initialise la structure AASC

    if ( !aascInit ( ) ) exit (-1) ; // Créé quelques données à transférer
    strepy ( aascBuffer , "Ceci est un transfert de données XModem...." ) ; // Effectue le transfert

    while (1) {
        hitwd ( ) ;
        printf ("Appuyez une touche pour commencer le transfert XModem vers le périphérique. \r") ;
        hitwd ( ) ;
        if ( kbhit ( ) ) {
            getchar ( ) ;
            printf ("\n\nTransfert XModem en cours....\n") ;
            hitwd ( ) ;

            // Prépare le transfert XModem en mémoire logique

            aascXMWrInitLog ( (unsigned) aascBuffer , 128 ) ;
            aascWriteXModem ( aascChannel , 0 , 0 , aascWrCallBackLg ) ;
            while ( !aascWriteXModem ( aascChannel , 0 , 0 , aascWrCallBackLg ) ) hitwd ( ) ;
            printf ("\n\nTransfert XModem terminé.....\n\n") ;
            hitwd ( ) ;
        }
    }
}

int aascInit (void) { // Initialise le driver virtuel

    VdInit ( ) ;

    // Ouvre le canal A du SCC en 8N1

    aascChannel = aascOpen ( DEV_SCC , 0 , SCC_A | SCC_1STOP | SCC_NOPARITY |
        SCC_8DATA | SCC_1200*BAUDMULT , NULL ) ;
    if ( aascChannel == NULL ) {
        printf ("SCC canal A non disponible." ) ;
        return ;
    }

    // Prépare les buffers circulaires

    aascSetReadBuf ( aascChannel , circBufIn , sizeof (circBufIn) ) ;
    aascSetWriteBuf ( aascChannel , circBufOut , sizeof (circBufOut) ) ;
}
```