Annexe

.1 Les directives du préprocesseur

Lors de la compilation d'un programme écrit en C on a coutume de dire que compilateur traduit le langage évolué en langage machine. C'est certes exact dans son ensemble, mais c'est un peu reducteur quant aux différentes étapes qui se produisent réellement. En fait, avant même cette traduction, un premier programme intervient le préprocesseur. Ce dernier interprète un certain nombre de directives qui lui soi propres et fournit en conséquence des informations au compilateur proprement dit compilateur dont le travail ne commence donc qu'après l'intervention de ce préprocesseur. Ce concept n'est pas propre aux seuls compilateurs C pour PIC et tous les compilateurs actuels disposent d'un préprocesseur et des directives associées. Nous n'allons donc traiter dans les lignes qui suivent que celles qui sont propres aux compilateurs C des PIC ; les autres devant vous etre connues même si vous n'avez que quelques notions de langage C. l'environnement de développement du compilateur C de CCS dispose d'un " wizard ", ou magicien en bon français qui est capable, à partir d'informations que vous lui donnez sous forme de cases cocher, d'écrire automatiquement une partie des directives indispensables au préprocesseur.

.1.1 Les directives d'informations de compilation

Un certain nombre de directives du preprocesseur peuvent être qualifiées d'informations de compilation. Elles permettent en effet d'informer le compilateur si l'affectation de noms ou d'étiquettes à des positions mémoire ou des registres particuliers. Voici les plus utiles d'entre elles présentées par ordre alphabétique.

- **#BIT**
  Cette directive s'utilise sous la forme :
  #bit id = x . y Elle a pour effet de créer une variable de taille égale à un bit placée en mémoire ; bit y de l'adresse x. Son but principal est de permettre un accès facile à certains bit d'état du PIC ou de ses ressources internes. Ainsi par exemple :
  #bit TOIF = 0xb.2 définit la variable TOIF comme étant le bit 2 du registre situé à l'adresse OB ; bit q est justement... TOIF !

- **#BYTE**
  Cette directive est très proche de la précédente mais pour un octet. Elle s'utilise sous la forme :

#byte id = x Elle a pour effet de créer une variable de taille égale à un octet placée en mémoire à l'adresse x. Son but principal est de permettre un accès facile à certains registres du PIC ou de ses ressources internes. Ainsi par exemple :
#byte STATUS = 3 Définit la variable STATUS comme étant le registre situé à l'adresse 3 ; registre qui est justement... le registre STATUS

– **#DEFINE**
Cette directive permet de substituer du texte, au sens large du terme, par un identifiant choisi par vos soins. Elle s'utilise sous l'une des deux formes suivantes :
#define id texte
#define id (x, y, ...)
texte Dans le premier cas, elle substituera l'identifiant par le texte qui le suit toutes les fois ou ce dernier apparaîtra dans le programme. Dans le second cas, la substitution sera plus évoluée puisqu'elle pourra concerner les paramètres d'une relation. Voici deux exemples qui devraient vous permettre de bien comprendre cette différence.
#define huit 8
Si le programme comprend la ligne :
a = a +huit
cela aura le meme effet que :
a = a + 8 Si maintenant nous écrivons :
#define poidsfort(x) (x $\ll$ 4)
et si le programme contient :
a = poidsforts(a)
cela aura le meme effet que : a = (a $\ll$ 4)

– **# INCLUDE**
Cette directive n'est pas propre aux seuls compilateurs C pour PIC puisqu'on la retrouve sur tous les compilateurs C. Nous estimons cependant utile de rappeler sa présence ici car on ne peut véritablement pas s'en passer comme vous le constaterez au partie suivante consacré à la présentation de l'environnement de développement. Elle s'utilise de la façon suivante :
#include <nom de fichier>
ou
#include "nom de fichier"
Dans les deux cas, le fichier dont le nom est spécifié est inclus dans le listing source contenant cette directive. La différence entre les deux syntaxes n'a d'influence que sur l'ordre dans lequel le fichier à inclure est recherché dans les répertoires de la machine utilisée pour la compilation. Dans le premier cas, le répertoire contenant le listing source est parcouru en dernier alors que, dans le deuxième cas, il est parcouru en premier.

– **#INLINE**
Cette directive permet d'agir sur l'efficacité du compilateur et de privilégier la vitesse d'exécution par rapport à la compacité du code génére. Elle indique en effet que la fonction qui la suit doit être insérée intégralement dans le programme compilé à chaque

fois qu'il y est fait référence. En l'absence de cette directive, le compilateur n'insère qu'un exemplaire de la fonction qu'il traite alors comme un sous programme, et il génère ensuite des appels à cette dernière lors de toutes ses apparitions dans le programme principal. Cela conduit à un code globalement plus compact mais à un programme plus lent puisqu'il y a une inévitable perte de temps avec les sauvegardes de contexte à chaque appel. Grâce à cette directive ces pertes de temps n'ont plus lieu mais, si la fonction est utilisée dix fois dans le programme, elle y sera reproduite dix fois in extension. C'est donc à vous de décide s'il vaut mieux privilégier la vitesse ou la compacité du code sachant que la directive #SEPARATE, présentée ci-dessous, permet de réaliser le comportement contraire. Sa syntaxe d'utilisation est la suivante : #inline Fonction La fonction qui suit sera alors insérée intégralement dans le programme compilé chaque fois qu'il y sera fait référence.

- **#LOCATE**
  Cette directive fonctionne comme la directive #BYTE vue ci-dessus mais elle interdit en outre au compilateur d'utiliser la mémoire ainsi allouée pour ses propres variable ; La syntaxe a utiliser est de la forme : #locate id = x Elle a pour effet de créer une variable de taille égale à un octet placée en mémoire à l'adresse x.

- **#OPT**
  Cette directive permet de fixer le niveau d'optimisation du compilateur entre 0 et pour les PIC des familles autres que la famille 18xxxx et entre 0 et 11 pour ces derniers. Par défaut, cette valeur est fixée à 5 pour les compilateurs PCB, PCM et PCH et à 9 pour le compilateur PCW. Si vous utilisez des PIC 18xxxx avec PCW, il e possible d'augmenter cette valeur à 10 ou 11 au moyen de cette directive. Réciproquement, si une erreur d'optimisation est suspectée, il est possible de réduire cette valeur afin de simplifier le travail du compilateur. La syntaxe est toute simplement :
  #opt n
  où n est le niveau d'optimisation désiré.

- **#RESERVE**
  Cette directive permet de réserver une zone de mémoire vive qui ne sera alors pas utilisée par le compilateur de manière automatique pour y loger ses propres variables. Cette directive doit apparaître immédiatement après la directive #device (voir ci-dessous) sinon elle est sans effet. Sa syntaxe peut revetir trois aspects différents :
  #reserve adresse
  #reserve adresse1, adresse2, adresse3
  #reserve début : fin
  où adresse, adresse1, adresse2, adresse3 sont des adresses de mots isolés tandis que début et fin sont respectivement les adresses de début et de fin de la zone à réserver.

- **#ROM**
  Cette directive permet de placer en mémoire morte une liste de données. Elle est utilisable par exemple pour programmer la mémoire EEPROM de données d'un 16F84

comme indiqué ci-dessous à titre d'exemple. Elle s'utilise de la façon suivante :
#ROM adresse = (motl, mot2, ...)
où adresse est la première adresse de la mémoire morte concernée et où motl, mot2, etc. sont les mots qui y sont placés, les uns à la suite des autres. Dans le cas de la mémoire EEPROM d'un 16F84, on peut ainsi écrire par exemple :
#rom 0x2100=l, 2, 3, 4
Pour placer 01, 02, 03 et 04 à la suite les uns des autres dans la mémoire EEPROM de données.

– **#SEPARATE**
Cette directive est l'opposé exact de la directive #inline c'est-à-dire qu'elle ordonne au compilateur de traiter la fonction qui la suit comme un sous-programme qui sera donc appelé toutes les fois ou son nom apparaîtra dans le programme principal, même si cela n'a lieu qu'une fois ou, au contraire, même si cela a lieu un trop grand nombre de fois et risque de conduire à un débordement de pile. Contrairement à #inline, cette directive privilégie donc l'occupation mémoire par rapport la vitesse d'exécution du programme. Elle s'utilise tout naturellement de la façon suivante : #separate Fonction La fonction qui suit sera alors traitée comme un sous-programme et sera appelée en tant que telle dans le programme compilé à chaque fois qu'il y sera fait référence.

.1.2 Les directives de définitions matérielles

Contrairement aux précédentes, ces directives sont directement liées au processeur utilisé. Elles permettent d'en définir le type, les fusibles de configuration utilisés, les interruptions à valider, etc.

– **#DEVICE**
Comme son nom le laisse supposer, cette directive indique au compilateur quel e le type de processeur utilisé ainsi qu'un certain nombre d'informations telles que
- la taille des pointeurs (*=5, *=8 ou *=i6 selon que l'on souhaite 5, 8 ou 16 bits'
- le nombre de bits du convertisseur analogique/digital interne (adc = x) ;
- la validation du mode de mise au point ("debug") en circuit (icd = true).
Cette directive s'utilise de la façon suivante :
#device processeur options
Ainsi par exemple :
#device PIC16F877 *=16 ADC=10
Indique que l'on utilise un 16F877 avec des pointeurs sur 16 bits et que la résolution du convertisseur analogique/digital est fixée à 10 bits.

– **#FILL_ROM**
Elle s'utilise sous la forme :
#fill_rom
valeur ou valeur est une constante sur 16 bits. Elle a pour effet de remplir toutes les zone inutilisées de la mémoire de programme avec la valeur spécifiée.

– **# FUSES**
Ici aussi le nom parle de lui-même puisque cette directive indique comment programmer les bits ou fusibles de configuration du PIC. La syntaxe est la suivante
#fuses liste et état des fusibles
Liste et état des fusibles est une suite d'informations séparées par des virgul précisant les fusibles à programmer ou l'état de la fonction à valider ou non à choi parmi :
- lp, xt, hs, rc pour le type d'horloge ;
- wdt ou nowdt pour valider ou non le timer chien de garde ;
- protect ou nprotect pour protéger ou non la mémoire de programme ;
- put ou noput pour valider ou non le timer à la mise sous tension ;
- brownout ou nobrownout pour valider ou non la détection de baisse anormale de la tension d'alimentation ;
- etc.
Ainsi par exemple :
#fuses XT, NOWDT, PUT, BROWNOUT
Choisit un oscillateur de type horloge à quartz de fréquence inférieure ou égal 4 MHz (horloge de type XT), pas de timer chien garde (nowdt), un timer à la mise sous tension (put) et la validation de la détection anormale de la baisse de la tension d'alimentation (brownout). La consultation du fichier d'en-tête du processeur (xxxx.h où xxxx est la référence du PIC utilisé) vous permettra si nécessaire de découvrir quelles sont toutes les options disponibles pour cette directive en fonction du PIC choisi.

– **#INT_xxxx**
Cette directive indique au compilateur que la fonction qui suit correspond au traitement de l'interruption spécifiée par xxxx. Ce paramètre est évidemment lié au type de PIC utilise puisque tous ne disposent pas des memes ressources internes et donc des mêmes sources potentielles d'interruptions. Le tableau suivant, liste les valeurs autorisées par le compilateur PCWH de CCS. L'utilisation de la directive est alors fort simple :
#int_xxxx
Fonction d'interruption xxxx Si l'application dispose de plusieurs sources d'interruptions potentielles, cette directive doit être répétée avant chaque fonction de traitement de l'interruption correspondante. Ici aussi, la consultation du fichier d'en tête du processeur (xxxx. h où xxxx est la référence du PIC utilisé) vous permettra si nécessaire de découvrir quelles sont toutes les options disponibles pour cette directive en fonction du PIC choisi.

– **#INT_DEFAULT**
Cette directive est d'utilisation rarissime. Elle joue le meme rôle que la directive int_xxxx vu ci-dessus mais correspond en fait au cas, très improbable, où le PIC considère qu'une interruption s'est produite alors qu'aucun de ses drapeaux d'interruption internes n'est valide et que, de ce fait, la source d'interruption ne peut pas être déterminée. Sa syntaxe est analogue à celle de la directive précédente à savoir :
#int_default Fonction de traitement

Source de l'interruption	Valeur de xxxx dans la directive #int_xxxx
Fin d'une conversion A/D	AD
Timeout d'une conversion A/D	ADOF
Collision sur le bus I2C	BUSCOL
Capture ou comparaison réussie CCP1	CCP1
Capture ou comparaison réussie CCP2	CCP2
Comparaison analogique	COMP
Fin d'écriture en EEPROM	EEPROM
Interruption d'entrée externe	EXT
Interruption d'entrée externe	EXT
Interruption d'entrée externe 1	EXT1
Interruption d'entrée externe 2	EXT2
Activité I2C (PIC14000 seulement)	I2C
Activité LCD	LCD
Chute de tension	LOWVOLT
Activité port parallèle esclave	PSP
Changement d'état sur B4 à B7	RB
Changement d'état sur C4 à C7	RC
Changement d'état sur C4 à C7	RC
Buffer de réception série plein	RDA
Débordement timer 0(RTCC)	RTCC
Activité I2C ou SPI	SSP
Buffer d'émission série vide	TBE
Débordement timer X (0 à 3)	TIMERX

TAB. 1 – - Paramètres admis par la directive #INT_XXXX

– **#INT_GLOBAL**
Cette directive est egalement d'utilisation rarissime. Elle indique en effet que la fonction qui la suit remplace le répartiteur de traitement d'interruption du compilateur. Elle ne doit donc être utilisée qu'avec de grandes précautions, si le répartiteur de traitement des interruptions du compilateur ne vous convient pas et si vous savez exactement ce que vous faites au niveau du traitement de ces dernières. Sa syntaxe est analogue à celle de la directive précédente à savoir :
#int_global Fonction de traitement

– **#PRIORITY**
Cette directive permet de définir la priorité relative des interruptions dont le traitement a été défini dans le programme. Elle s'utilise sous la forme :
#priority ints
où ints est une suite de noms valides d'interruptions classées la plus prioritaire à la moins prioritaire. Dans ces conditions et si deux interruptions se produisent au même instant l'interruption classée comme la plus prioritaire sera traitée en premier.

– **#USE DELAY**
Comme ci-dessous, le compilateur CCS dispose de deux fonctions appelées delay_ms et delay_us qui sont capables de générer des délais spécifiés respectivement en ms ou en us. Pour que ces fonctions conduisent à des delais corrects, le compilateur doit connaître la fréquence d'horloge exacte du PIC et cette directive permet donc de la lui indiquer. Elle s'utilise sous la forme :
#use delay (clock = vitesse)
où vitesse est la fréquence d'horloge exprimée en hertz. Cette directive supporte également une option particulière à employer si vous utilisez le timer chien de garde. Il faut alors l'écrire sous la forme :
#use delay (clock = vitesse, RESTART_WDT)
ce qui aura pour effet de réinitialiser automatiquement le timer chien de garde pendant les boucles de délai, évitant ainsi son déclenchement inconsidéré.

– **#USE FAST_IO**
Cette directive permet d'accélérer le traitement des entrées/sorties par le compilateur. En effet, en son absence le compilateur positionne les registres de sens de transfert des données (TRIS) lors de chaque appel d'une fonction d'entrée/sortie ce qui génère une perte de temps inévitable. Si cette directive est utilisée, le compilateur ne se préoccupe plus de l'état de ces registres lors des opérations d'entrée/sortie ce qui implique qu'ils aient été correctement programmés par vos soins au préalable. Cette directive reste efficace jusqu'à ce que le compilateur rencontre une directive use xxx_io qui établit alors un des autres modes de fonctionnement décrits ci-après. La syntaxe est la suivante :
#use fast_io(X)
où x est le nom du port parallèle concerné (de A à G selon les PIC).

– **#USE FIXED_IO**

Cette directive permet de fixer le sens de transfert d'une ou plusieurs lignes des ports parallèles une fois pour toutes et ce jusqu'à ce qu'une nouvelle directive use xxx_io vienne le modifier. Si cette directive est utilisée, le compilateur ne se préoccupe plus de l'état des registres TRIS correspondants lors des opérations d'entrée/sortie sur ces lignes. La syntaxe est de la forme :
#use fixed_io (port_outputs=pin, pin)
où port est le nom du port parallèle concerné (de A à G selon les PIC) et où pin est un nom de patte valide sur ce port conformément a ce qui figure dans le fichier d'en tête xxxx.h du PIC concerné. Ainsi par exemple :
#use fixed_io (A_outputs=PIN_A0, PIN_A3)
place en sortie les lignes A0 et A3 du port A.

– **#USE I2C**
Comme son nom le laisse supposer, cette directive indique au compilateur de fa appel à l'interface I2C et precise ses différents modes d'utilisation au moyen de syntaxe suivante :
#use i2c (options)
où options est constitué par un ou plusieurs des mots clés présentés dans tableau suivant . Notez à propos de ces options que, si force_hw n'est pas utilisée, les sous-programes de gestion I2C sont entièrement réalisés sous forme logicielle et que les lignes S et SCL peuvent donc etre affectées à n'importe quel port parallèle. Dans le cas contraire, le périphérique I2C matériel interne est utilise et les lignes SDA et SCL doivent alors être choisies en conséquence comme précisé dans la fi technique du PIC concerné. Le mode esclave (option slave) ne peut être utilisé qu'en mode forçage matériel.

Options de #USE I2C	Signification
MASTER	PIC en mode maître I2C
SLAVE	PIC en mode esclave I2C
SCL=pin	Définition de la patte SCL
SDA=pin	Définition de la patte SDA
ADDRESS=N	Définition de l'adresse en mode esclave
FAST	Fonctionnement en mode I2C rapide
SLOW	Fonctionnement en mode I2C lent
RESTART.WDT	Réinitialisation du timer chien de garde en mode réception I2C
FORCE_HW	Utilisation de l'interface I2C matérielle intégrée
NOFLOAT_HIGH	Interdiction de mise en haute impédance des signaux
SMBUS	Fonctionnement en mode SMBUS au lieu de l'12C

Tab. 2 – Options supportées par la directive #USE I2C

– **#USE RS232**
Comme son nom l'indique, cette directive indique au compilateur de faire us de l'interface série asynchrone, appelée RS 232 par un raccourci un peu restrictif et précise ses différents modes d'utilisation au moyen de la syntaxe suivante :

Options de #use rs232	Signification
STREAM=valeur	Définit la liaison comme utilisant le flux égal à valeur
BAUD=n	Définit la vitesse de transmission en bits par seconde
XMIT=pin	Définit la patte d'émission de données
RCV=pin	Définit la patte de réception de données
FORCE_SW	Force la génération d'un UART par logiciel
BRGH10K	Autorise la définition d'une « mauvaise » vitesse de transmission
DEBUGGER	Indique que ce flux est utilisé pour communiquer avec un outil de développement en circuit via B3
RESTART_WDT	Permet à getc ( ) de réinitialiser le timer chien de garde pendant l'attente d'un caractère
INVERT	Inverse la polarité des signaux pour le cas où un circuit d'interface RS 232 externe n'est pas utilisé
PARITY=n	Interdiction de mise en haute impédance des signaux
BITS=n	Définit le nombre de bits de données (5 à 9)
FLOAT_HIGH	Mise en haut impédance de la ligne d'émission si non utilisée
LONG_DATA	Permet à getc() et à putc() d'utiliser des entiers sur 16 bits lorsque bits=9 est défini
DISABLE_INTS	Désactive les interruptions lorsqu'un caractère est émis ou reçu

Tab. 3 – Options supportées par la directive #USE I2C

#use rs232 (options)
où options est constitué par un ou plusieurs des mots clés présentés dans le tableau suivant. Afin que cette directive fonctionne correctement, elle doit impérativement être précédee de la directive #use delay afin de connaître la fréquence d'horloge du PIC et de pouvoir ainsi générer des vitesses de transmission correctes. Cette directive doit être utilisée si vous faites appel dans votre programme aux fonctions standard d'entrées/sorties que sont getc, putc et printf. Si les lignes xmit et rcv spécifiées pour cette directive correspondent à celles de USART interne du PIC choisi, ce dernier est automatiquement validé et utilisé. Dans le cas contraire, un UART sous forme logicielle est programmé par le compilateur. Enfin, si compte tenu de la vitesse de transmission spécifiée et de la fréquence d'horloge du PIC utilise, cette vitesse ne peut être générée avec une precision meilleure que 3 %, une erreur de compilation est affichée.

– **#USE STANDARD_IO**
Cette directive permet de rétablir le traitement " normal " des entrées/sorties par le compilateur, c'est-à-dire que le compilateur positionne à nouveau les registres de sens de transfert des données (TRIS) lors de chaque appel d'une fonction d'entrée/sortie. Cette directive est inutile si use fast_io ou use fixed_io n'ont pas été utilise au préalable car elle ne fait que rétablir le fonctionnement par défaut du compilateur La syntaxe est

la suivante :#use standard_io (X) où x est le nom du port parallèle concerné (de A à G selon les PIC).

– **#ZERO_RAM**
Comme son nom l'indique, cette directive remet à zéro toute la mémoire vive susceptible d'être utilisée par le compilateur pour stocker les variables du programme avant que l'exécution de ce dernier ne commence. Sa syntaxe est la suivante :
#zero_ram

.2 Les principales fonctions spécifiques au langage C des PIC

Les directives du preprocesseur ne suffisent pas pour qu'un compilateur C, quel qu'il soit, puisse prendre en compte les particularites d'un microcontroleur. En effet la gestion performante de ses entrées/sorties et de ses ressources internes impose d'ajouter au langage C " de base " un certain nombre de fonctions specifique C'est même de la richesse de ces ajouts que dépend aujourd'hui la facilité d'emploi d'un compilateur car cela décharge le developpeur d'un certain nombre de tâche de programmation fastidieuses. Voici donc, classées par ordre alphabétique, principales fonctions spécifiques du compilateur C de CCS.

– **BIT_CLEAR()**
Cette fonction met à zéro un bit déterminé d'une variable. Elle s'utilise de la façon suivante :
bit_clear (variable, bit)
où variable est un nom de variable valide du programme concerné et bit et le numéro de bit de cette variable, compris entre 0 et 7, 15 ou 31 selon le cas, sachant que le bit 0 est le bit de poids faible.

– **BIT_SET()**
Cette fonction met à 1 un bit déterminé d'une variable. Elle s'utilise de la façon suivante :
Bit_set(variable, bit)
où variable est un nom de variable valide du programme concerné et bit et le numéro de bit de cette variable, compris entre 0 et 7, 15 ou 31 selon le cas, sachant que le bit 0 est le bit de poids faible.

– **BIT_TEST()**
Cette fonction teste un bit déterminé d'une variable. Elle s'utilise de la façon suivante : valeur = bit_test (variable, bit) où variable est un nom de variable valide du programme concerné, bit et le numéro de bit de cette variable, compris entre 0 et 7, 15 ou 31 selon le cas, sachant que le bit 0 est le bit de poids faible et où valeur est égale à 0 ou à 1 selon que le bit est lui-même à 0 ou a 1. De ce fait, cette fonction peut être utilisée dans

toutes les expressions conditionnelles. Pour attendre qu'un bit passe à un, on pourra ainsi écrire :
while (bit_test(variable,bit)) ;

– **CLEAR_INTERRUPT()**
Cette fonction a pour effet de remettre à zéro le drapeau de l'interruption spécifiée. Elle s'utilise de la façon suivante :
clear_interrupt (interruption)
où interruption correspond à l'interruption concernée, codée comme indiqué tableau des interruptions.

– **DELAY_CYCLES()**
Comme son nom l'indique cette fonction réalise une boucle d'attente dont la durée est spécifiée en nombre de cycles machine, compris entre 1 et 255. Rappelons qu'un cycle machine dure quatre périodes d'horloge. Elle s'utilise de la façon suivante :
de1ay_cyc1es (nombre)

où nombre est une constante comprise entre 1 et 255 qui indique le nombre de cycles machine.

– **DELAY_MS()**
Comme son nom l'indique cette fonction réalise une boucle d'attente dont la durée est spécifiée en millisecondes entre 1 et 65 535. Elle s'utilise de la façon suivante :
delay_ms (nombre)
où nombre est une constante comprise entre 1 et 255 ou entre 1 et 65 535 qui indique la durée d'attente en millisecondes. Cette fonction ne fait appel à aucun timer du PIC. Ceux-ci restent donc totalement libres pour l'application. Par contre, comme le délai généré par cène fonction ne repose que sur des durées d'exécution d'une suite d'instructions bien précises, la survenue d'une interruption pendant l'exécution de cette fonction allonge le délai spécifié de manière imprévisible

– **DELAY_US()**
Comme la précédente, cette fonction réalise une boucle d'attente, mais dont la dut est spécifiée cette fois-ci en microsecondes entre 1 et 65 535. Elle s'utilise de la façon suivante :
delay_us(nombre)
où nombre est une constante comprise entre 1 et 255 ou entre 1 et 65 535 qu'indique la durée d'attente en microsecondes. Cette fonction ne fait appel à aucun timer du PIC. Ceux-ci restent donc totalement libres pour l'application. Par contre, comme le délai généré par cette fonction repose que sur des durées d'exécution d'une suite d'instructions bien précises, survenue d'une interruption pendant l'exécution de cette fonction allonge le dé spécifié de manière imprévisible.

– **DISABLE_INTERRUPTS()**

Cette fonction a pour effet d'interdire la prise en compte de l'interruption spécifié Elle s'utilise de la façon suivante :
disable_interrupts (interruption)
où interruption correspond à l'interruption concernée, codée comme indique le tableau d'interruption. L'utilisation de cette fonction avec " l'interruption " global ne désactive aucune interruption particulière mais interdit la prise en compte de toute interruption agissant sur le bit GIE du PIC. L'annulation de cette fonction au moyen d' enable_interrupts(global) valide donc à nouveau toutes les interruptions qui avaient préalablement été autorisées.

– **ENABLE_INTERRUPTS()**
Cette fonction a pour effet d'autoriser la prise en compte de l'interruption spécifié. Elle s'utilise de la façon suivante :
enable_interrupts (interruption)
où interruption correspond à l'interruption concernée, codée comme indiqué dansle tableau d'interruption. L'utilisation de cette fonction avec " l'interruption " global n'autorise aucune interruption particulière mais permet la prise en compte de toute interruption préalablement autorisée en agissant sur le bit GIE du PIC.

– **ERASE_PROGRAM_EEPROM()**
Cette fonction efface un mot en mémoire EEPROM de programme sous réserve que le PIC utilisé supporte cette possibilite. Aucun controle n'est effectué par le compilateur sur le mot ainsi efface et cette fonction doit donc être utilisée avec prudence. Elle s'utilise de la façon suivante :

erase_program_eeprom (adresse)
où adresse indique l'adresse du mot a effacer en mémoire EEPROM de programme.

– **EXT_INT_EDGE()**
Cette fonction détermine si l'entrée d'interruption externe spécifiée est active sur un front montant ou descendant. Elle s'utilise de la façon suivante :
ext_int_edge (source, edge)
où source est égal à 0, 1 ou 2 pour la famille 18xxxx, afin de spécifier l'entrée d'interruption externe utilisée, et est optionnel et vaut 0 par défaut pour toutes les autres familles de PIC, et où edge est égal à h_to_l pour une transition descendante et à l_to_h pour une transition montante.

– **GET_TIMERx()**
Cette fonction fournit la valeur contenue au moment de son exécution dans le registre de comptage du timer choisi. La valeur fournie est un entier codé sur 8 ou 16 bits selon la largeur du registre du timer concerné. Elle s'utilise de la façon suivante :
valeur = get_timerx()
où x est le numéro du timer choisi conformément à l'appellation utilisée par Microchip dans ses fiches techniques et où valeur est la variable récupérant le résultat de cette