WINCUPL 5.0.

Guide de référence du programmeur CUPL

Logical Devices, Inc. 1221 S Clarkson St. Suite 200 Denver, CO 80210 Technical Support Telephone: (303) 722-686

Notes sur ce document

Ce document est une conversion de la documentation en ligne qui accompagne la distribution Atmel WinCUPL 5.0. Il a été produit par le département de génie électrique de l'Université de Californie du Sud pour aider les étudiants inscrits à la classe EE 459Lx (Embedded Systems Design Laboratory). Le document original est protégé par les droits d'auteur de Logical Devices, Inc. (voir page de titre)

La documentation originale en ligne se trouvait dans « . HLP » et pourrait être parcouru sur un système Windows. Le fichier de documentation, « CUPLREF. HLP », a été converti au format RTF (. RTF) par le programme « helpdeco » de la distribution « helpdc21.zip » de www.simtel.net. Le programme helpdeco a également converti la plupart des chiffres en « . WMF » ou « .BMP » graphiques. Le fichier RTF a été lu dans Adobe Framemaker 6.0 et (très laborieusement) reformaté en quelque chose qui semblait raisonnable. Dans certains cas, la numérotation des sections a été légèrement modifiée car la disposition originale du HLP semblait un peu étrange par endroits. Après qu'Adobe ait rendu évident que les jours de Framemaker étaient comptés, le document a été reformaté à nouveau en LaTeX 2e.

La plupart des figures ont été converties du format WMF et BMP en Postscript encapsulé (. EPS) par Canvas 7SE. Beaucoup d'entre eux n'avaient pas l'air très beaux, ils ont donc été redessinés dans Canvas 7SE, ou plus tard dans Canvas X. Il a été constaté que bon nombre des figures mentionnées dans ce document étaient absentes du fichier HLP et que jusqu'à ce qu'elles puissent être localisées ou reproduites, des espaces vides ont été laissés dans le document. Dans certains cas, de meilleures versions des figures ont été trouvées dans le document Atmel DOC0737.PDF qui est une version courte de certains des mêmes sujets abordés dans ce document. Ces chiffres ont été extraits de ce fichier PDF et convertis par Canvas 7SE au format EPS. La plupart des figurines récupérées ont finalement été redessinées dans Adobe Illustrator après le retrait de Canvas du marché Mac.

Tous les efforts ont été faits pour reproduire avec précision le contenu du fichier HLP. Cependant, ce document ne doit pas être considéré comme une référence définitive sur la distribution de la CUPL par Atmel. Veuillez vous référer à la documentation originale en ligne pour obtenir les informations les plus précises.

Allan G. Weber
University of Southern California
Department of Electrical Engineering - Systems
Los Angeles, CA 90089-2564
weber@sipi.usc.edu
(Notes dated 10/22/12

Contenu

Chapitre 1

Référence du langage CUPL

Ce chapitre explique les éléments du langage CUPL et la syntaxe du langage CUPL.

1 Éléments de langage

Cette section décrit les éléments qui composent le langage de description logique CUPL.

1.1 Variables

Les variables sont des chaînes de caractères alphanumériques qui spécifient des broches de périphérique, des nœuds internes, des constantes, des signaux d'entrée, des signaux de sortie, des signaux intermédiaires ou des ensembles de signaux. Cette section explique les règles de création de variables.

Les variables peuvent commencer par un chiffre numérique, un caractère alphabétique ou un trait de soulignement, mais doivent contenir au moins un caractère alphabétique.

Les variables sont sensibles à la casse; c'est-à-dire qu'ils distinguent les lettres majuscules et minuscules.

N'utilisez pas d'espaces dans un nom de variable. Utilisez le caractère de soulignement pour séparer les mots.

Les variables peuvent contenir jusqu'à 31 caractères. Les variables plus longues sont tronquées à 31 caractères.

Les variables ne peuvent contenir aucun des symboles réservés CUPL (voir le tableau 1.2 à la page 3).

Les variables ne peuvent pas être identiques à un mot-clé réservé CUPL (voir le tableau 1.1 à la page 3).

Voici des exemples de noms de variables valides :

a0
A0
8250 _ENABLE
Real_time_clock_interrupt
_address

Notez comment l'utilisation du trait de soulignement dans les exemples ci-dessus facilite la lecture des noms de variables. Notez également la différence entre les noms de variables majuscules et minuscules. La variable A0 n'est pas la même que a0.

Voici des exemples de noms de variables non valides :

99	ne contient pas de caractère alphabétique
I/O enable	contient un caractère spécial (/)
out 6a	contient un espace; Le système le lit comme deux variables distinctes
tbl-2	contient un tiret; Le système le lit comme deux variables.

1.1.2 Variables indexées

Les noms de variables peuvent être utilisés pour représenter un groupe de lignes d'adresse, de lignes de données ou d'autres éléments numérotés séquentiellement. Par exemple, les noms de variables suivants peuvent être affectés aux huit lignes d'adresse LO-order d'un microprocesseur :

A0 A1 A2 A3 A4 A5 A6 A7

Les noms de variables qui se terminent par un nombre, comme indiqué ci-dessus, sont appelés variables indexées

✎Note: Il est préférable de commencer les variables indexées à partir de zéro (0), par exemple Utiliser X0.. 4 au lieu de X1.. 5.

Les indices sont toujours des nombres décimaux compris entre 0 et 31. Lorsqu'elle est utilisée dans des opérations de champ binaire (voir Sec. 1.1.11, Déclarations de champ binaire), la variable avec le numéro d'index 0 est toujours le bit d'ordre le plus bas.

✎Note: Les variables se terminant par un nombre supérieur à 31 ne sont pas des variables indexées.

Voici des exemples de noms de variables indexées valides :

a23
D07
D7
counter_bit_3

Notez la différence entre les variables d'index avec des zéros non significatifs ; la variable D07 n'est pas la même que D7.

Voici des exemples de noms de variables indexées non valides :

D0F	Le numéro d'index n'est pas décimal
a36	indice hors plage

Ce sont des noms de variables valides, mais ils ne sont pas considérés comme indexés.

1.1.3 Mots et symboles réservés

CUPL utilise certaines chaînes de caractères avec des significations prédéfinies appelées mots-clés. Ces mots-clés ne peuvent pas être utilisés comme noms dans CUPL. Le tableau 1.1 énumère ces mots-clés.

La CUPL réserve également certains symboles à son usage qui ne peuvent pas être utilisés dans les noms de variables. Le tableau 1.2 énumère ces symboles réservés.

APPEND	FUNCTION	PARTNO
ASSEMBLY	FUSE	PIN
ASSY	GROUP	PINNNODE
COMPANY	IF	PRESENT
CONDITION	JUMP	REV
DATE	LOC	REVISION
DEFAULT	LOCATION	SEQUENCE
DESIGNER	MACRO	SEQUENCED
DEVICE	MIN	SEQUENCEJK
ELSE	NAME	SEQUENCERS
FIELD	NODE	SEQUENCET
FLD	OUT	TABLE
FORMAT		

Table 1.1: CUPL Reserved Keywords

&	#	(	)	-
	+	[	]	/
:	.	..	/*	*/
;	,	!	'	=
@	$	^		

Table 1.2: CUPL Reserved Symbols

1.1.4 Chiffres

Toutes les opérations impliquant des nombres dans le compilateur CUPL sont effectuées avec une précision de 32 bits. Par conséquent, les nombres peuvent avoir une valeur comprise entre . La lettre de base est placée entre guillemets simples et peut être en majuscules ou en minuscules. Quelques exemples de spécifications numériques valides sont énumérés dans le tableau 1.4.

Les nombres binaires, octales, hexadécimaux et numériques peuvent avoir des valeurs quelconques. Quelques exemples de spécifications numériques valides avec des valeurs quelconques sont répertoriés dans le tableau 1.5.

0 to $2^{32}-1$

Base Name	Base	Prefix
Binary	2	'b'
Octal	8	'o'
Decimal	10	'd'
Hexadecimal	16	'h'

Table 1.3: Number Base Prefixes

Number	Base	Decimal Value
'b'0	Binary	0
'B'1101	Binary	13
'O'663	Octal	435
'D'92	Decimal	92
'h'BA	Hexadecimal	186
'O'[300..477]	Octal (range)	192..314

Table 1.4: Sample Base Conversions

Number	Base
'b'1X11	Binary
'O'0X6	Octal
'H'[3FXX..7FFF]	Hexadecimal (range)

Table 1.5: Sample Don't Care Numbers

1.1.5 Commentaires

Les commentaires constituent une partie importante du fichier de description logique. Ils améliorent la lisibilité du code et documentent les intentions, mais n'affectent pas de manière significative le temps de compilation, car ils sont supprimés par le préprocesseur avant toute vérification de la syntaxe. Utilisez les symboles /* et */ pour joindre les commentaires ; Le programme ignore tout ce qui se trouve entre ces symboles.

Les commentaires peuvent s'étendre sur plusieurs lignes et ne sont pas terminés à la fin d'une ligne. Les commentaires ne peuvent pas être imbriqués. Quelques exemples de commentaires valides sont présentés à la figure 1.1.

1.1.6 Notation de la liste

Les notations abrégées sont une caractéristique importante du langage CUPL.

La notation abrégée la plus fréquemment utilisée est la liste. Il est couramment utilisé dans les déclarations de broches et de nœuds,

```
/*****************************************/
/*  This is one way to create a title or  */
/*         an information block           */
/*****************************************/

/*
This is another way to create an information block
*/

out1=in1 # in2;        /* A Simple OR Function  */
out2=in1 & in2;        /* A Simple AND Function */
out3=in1 $ in2;        /* A Simple XOR Function  */
```

Figure 1.1: Sample Comments

Déclarations de champs binaires, équations logiques et opérations d'ensemble. Le format de liste est le suivant :

[variable , variable , ... variable]

Où

[ ] sont des parenthèses utilisées pour délimiter les éléments de la liste en tant qu'ensemble de variables.

Deux exemples de notation de liste sont les suivants :

[UP , DOWN , LEFT , RIGHT]
[A0 , A1 , A2 , A3 , A4 , A5 , A6 , A7]

Lorsque tous les noms de variables sont numérotés séquentiellement, soit du plus bas au plus élevé, soit inversement, le format suivant peut être utilisé :

[variablem .. n]

où

m est le premier numéro d'index de la liste des variables.

n est le dernier nombre de la liste des variables; n peut être écrit sans le nom de la variable.

Par exemple, la deuxième ligne de l'exemple ci-dessus pourrait être écrite comme suit :

[A0 ..7]

Les indices sont supposés être décimaux et contigus. Tous les zéros non significatifs de l'index de variable sont supprimés du nom de la variable créée. Par exemple :

[A00 ..07]

est l'abréviation de :

[A0 , A1 , A2 , A3 , A4 , A5 , A6 , A7]

Pas pour :

[A00 , A01 , A02 , A03 , A04 , A05 , A06 , A07]

Les deux formes de la notation de liste peuvent être mélangées dans n'importe quelle combinaison. Par exemple, les deux notations de liste suivantes sont équivalentes :

[A0 ..2 , A3 , A4 , A5 ..7] [A0 , A1 , A2 , A3 , A4 , A5 , A6 , A7]

1.1.7 Fichier de modèle

Lorsqu'un fichier source de description logique est créé à l'aide du langage CUPL, certaines informations doivent être saisies, telles que les informations d'en-tête, les déclarations de broches et les équations logiques. Pour obtenir de l'aide, la CUPL fournit un fichier modèle qui contient la structure appropriée pour le fichier source.

La figure 1.2 montre le contenu du fichier modèle.

```
Name          XXXXX;
Partno        XXXXX;
Date          XX/XX/XX;
Revision      XX;
Designer      XXXXX;
Company       XXXXX;
Assembly      XXXXX;
Location      XXXXX;
/*****************************************************************/
/*  Allowable Target Device Types:                               */
/*****************************************************************/

/**  Inputs  **/
Pin           =            ;      /*                             */
Pin           =            ;      /*                             */
Pin           =            ;      /*                             */
Pin           =            ;      /*                             */
Pin           =            ;      /*                             */
Pin           =            ;      /*                             */
Pin           =            ;      /*                             */
Pin           =            ;      /*                             */
Pin           =            ;      /*                             */
Pin           =            ;      /*                             */
Pin           =            ;      /*                             */
Pin           =            ;      /*                             */
Pin           =            ;      /*                             */
Pin           =            ;      /*                             */

/**  Outputs  **/
Pin           =            ;      /*                             */
Pin           =            ;      /*                             */
Pin           =            ;      /*                             */
Pin           =            ;      /*                             */
Pin           =            ;      /*                             */
Pin           =            ;      /*                             */
Pin           =            ;      /*                             */
Pin           =            ;      /*                             */

/** Declarations and Intermediate Variable Definitions **/

/**  Logic Equations  **/
```

Figure 1.2: Template File

e fichier de modèle fournit les sections suivantes :

Informations d'en-tête - Mots-clés suivis de XXX qui sont remplacés par du texte pour identifier le fichier à des fins d'archivage et de révision.

Bloc de titre - Symboles de commentaires qui entourent l'espace pour décrire la fonction de la conception et des équipements cibles autorisés.

Déclaration de broche - Mots-clés et opérateurs dans le format approprié pour les déclarations de broches d'entrée et de sortie et l'espace de commentaire pour décrire les affectations de broches. Une fois les déclarations d'épingle effectuées, supprimez toutes les lignes. Sinon, une erreur de syntaxe se produira lors de la compilation.

/* Inputs */ and /* Outputs */ - les commentaires qui fournissent des regroupements pour la lisibilité uniquement. Affectez n'importe quel type de code confidentiel dans n'importe quel ordre, quelle que soit la façon dont il est utilisé dans le fichier de description logique.

Déclaration et variable intermédiaire - Espace pour faire des déclarations, telles que des déclarations de champ de bits (voir la section 1.1.11, Déclarations de champ de bits et la section 1.1.10, Déclarations de déclaration de nœud) et pour écrire des équations intermédiaires (voir la section 1.2.9, Équations logiques).

Équation logique - Espace pour écrire des équations logiques décrivant la fonction du dispositif (voir la section 1.2.9, Équations logiques).

1.1.8 Renseignements sur l'en-tête

La section d'informations d'en-tête du fichier source identifie le fichier à des fins de révision et d'archivage. Normalement, placez-le au début du fichier. CUPL fournit 10 mots-clés à utiliser dans les déclarations d'informations d'en-tête. Commencez chaque instruction par un mot-clé qui peut être suivi de n'importe quel caractère ASCII valide, y compris les espaces et les caractères spéciaux. Terminez chaque instruction par un point-virgule. Le tableau 6 répertorie les mots-clés d'en-tête CUPL et les informations à fournir avec chaque mot-clé.

Le fichier de modèle fournit tous les mots-clés d'en-tête à l'exception de DEVICE et FORMAT. Voici un exemple d'informations d'en-tête CUPL appropriées :

```
Name        WAITGEN ;
Partno      9000183 ;
Revision    02 ;
Date        1/11/89 ;
Designer    Osann ;
Company     Logical Devices, Inc. ;
Assembly    PC Memory Board ;
Location    U106 ;
Device      F155;
Format      ij ;
```

Si des informations d'en-tête sont omises, la CUPL émet un message d'avertissement, mais continue la compilation.

1.1.9 Déclarations d'épingles

Les instructions de déclaration d'épingle déclarent les numéros de broches et leur attribuent des noms de variables symboliques. Le format d'une déclaration de code PIN est le suivant :

PIN pin_n =[!] var ;

Mot-clé	Information
NAME	Utilisez normalement le nom de fichier de description de la logique source. Utilisez uniquement des chaînes de caractères valides pour le système d'exploitation. Le nom spécifié ici détermine le nom de tous les fichiers de téléchargement JEDEC, ASCII - hexadécimal ou HL. Le champ NAME peut accueillir des noms de fichiers de 32 caractères maximum. Lorsque vous utilisez des systèmes tels que DOS qui autorisent des noms de fichiers de seulement huit caractères, le nom de fichier sera tronqué.
PARTNO	Spécifiez le numéro de pièce exclusif d'une entreprise (généralement délivré par le fabricant) pour une conception PLD particulière. Le numéro de pièce n'est pas le type de PLD cible. Pour les périphériques GAL, les huit premiers caractères sont codés à l'aide d'ASCII sept bits dans les fusibles de signature utilisateur de la carte de fusibles des périphériques.
REVISION	Commencez par 01 lors de la première création d'un fichier et incrémentez chaque fois qu'un fichier est modifié. REV peut être utilisé pour une abréviation.
DATE	Modifiez la date actuelle chaque fois qu'un fichier source est modifié.
DESIGNER	Spécifier le nom du concepteur
COMPANY	Spécifiez le nom de l'entreprise pour une pratique de documentation appropriée et parce que les spécifications peuvent être envoyées aux fabricants de semi-conducteurs pour les commandes PLD à volume élevé.
ASSEMBLY	Indiquez le nom ou le numéro de la carte PC sur laquelle le PLD sera utilisé. L'abréviation ASSY peut être utilisée.
LOCATION	Indiquez la référence de la carte PC ou les coordonnées où se trouve le PLD. L'abréviation LOC peut être utilisée.
DEVICE	Définissez le type de périphérique par défaut pour la compilation. Un type de périphérique spécifié sur la ligne de commande remplace tous les types de périphériques définis dans le fichier source. Pour les fichiers sources multi-périphériques, DEVICE doit être utilisé avec chaque section si les types de périphériques sont différents.
FORMAT	Définissez un remplacement du format de sortie de téléchargement pour la section de description logique actuelle. Les valeurs valides à utiliser pour le format de sortie sont les suivantes : h produce ASCII-hex output i produce Signetics HL output j produce JEDEC output FORMAT remplace tout indicateur d'option sur la ligne de commande. Il est utile dans les fichiers sources multi-périphériques où différentes parties ont des formats de sortie incompatibles. Plusieurs valeurs de format à la fois peuvent être spécifiées pour produire plus d'un type de sortie. La valeur de format doit être une lettre minuscule.

Table 1.6: Informations d'en-tête Mots-clés

où

PIN est un mot-clé permettant de déclarer les numéros PIN et de leur attribuer des noms de variables.

pin n est un numéro de broche décimal ou une liste de numéros de broches regroupés à l'aide de la notation de liste; C'est

[pin_n 1 , pin_n 2 ... pin_nn]

! est un point d'exclamation facultatif pour définir la polarité du signal d'entrée ou de sortie.

= est l'opérateur d'affectation.

var est un nom de variable unique ou une liste de variables regroupées à l'aide de la notation de liste; C'est

[var , var ... var]

; est un point-virgule pour marquer la fin de l'instruction de déclaration.

Le fichier de modèle fournit une section permettant d'entrer les variables d'épingle individuellement ou en groupes à l'aide de la notation de liste.

Le concept de polarité peut souvent être déroutant. Dans toute conception PLD, le concepteur se préoccupe principalement de savoir si un signal est vrai ou faux. Le concepteur ne devrait pas avoir à se soucier de savoir si cela signifie que le signal est élevé ou faible. Pour diverses raisons, une conception de carte peut exiger qu'un signal soit considéré comme vrai lorsqu'il est de niveau logique 0 (faible) et faux lorsqu'il est logique 1 (élevé). Ce signal est considéré comme actif-faible puisqu'il est activé lorsqu'il est faible. Cela pourrait aussi être appelé faible-vrai. Si un signal est changé d'actif-haut à actif bas, la polarité a été modifiée.

Pour cette raison, CUPL vous permet de déclarer la polarité du signal dans la définition de la broche et vous n'avez plus à vous en préoccuper. Lors de l'écriture d'équations en syntaxe CUPL, le concepteur ne doit pas se préoccuper de la polarité du signal. Les déclarations de broches déclarent une translation qui gérera la polarité du signal.

Supposons que nous voulions la fonction suivante.

```
Y = A & B ;
```

Ce que cette affirmation signifie, c'est que Y sera vrai lorsque A est vrai et B est vrai. Nous pouvons implémenter cela dans un appareil P22V10 très facilement.

```
Pin 2 = A ;
Pin 3 = B ;
Pin 16 = Y ;
Y = A & B ;
```

Lorsque le dispositif est branché sur un circuit, si une logique 1 est affirmée aux broches 2 et 3, le signal à la broche 16 sera élevé. Supposons que, pour une raison quelconque, nous voulions que les entrées lisent la logique 0 comme vraie. Nous pourrions modifier la conception pour nous comporter de cette façon.

```
Pin 2 = ! A ;
Pin 3 = ! B ;
Pin 16 = Y ;
Y = A & B ;
```

Maintenant, même si le **!** symbole a été placé dans la déclaration de broche pour indiquer la polarité inversée, l'équation se lit toujours « Y est vrai lorsque A est vrai et B est vrai ». Tout ce qui a été changé est la traduction de vrai =0 et faux =1. Donc, au niveau de la conception, rien n'a changé, mais dans les déclarations de broches, nous mappons maintenant 0 à vrai et 1 à faux.

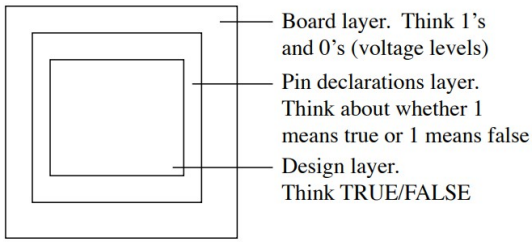

Figure 1.3: Relationship Between Pin Declaration and Signal Polarity.

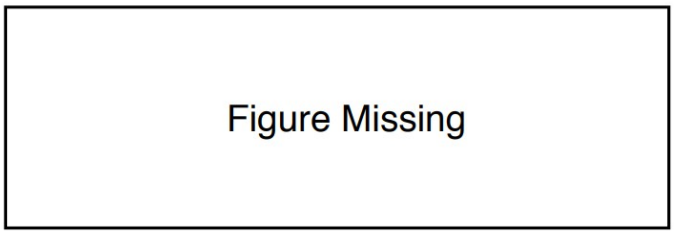

Figure 1.4: Active-HI Pin Declaration for Inverting Buffer

Cela encourage le concepteur à séparer la conception en couches afin de minimiser la confusion liée à la polarité. Il est également important que la CUPL modifie le signal de rétroaction afin que la couche vrai/faux soit maintenue.

Utilisez le point d'exclamation (!) pour définir la polarité d'un signal d'entrée ou de sortie. Si un signal d'entrée est LO de niveau actif (c'est-à-dire que le niveau de tension du signal TTL affirmé est de 0 volts), placez un point d'exclamation avant le nom de la variable dans la déclaration de broches. Le point d'exclamation informe le compilateur de choisir le sens inversé du signal lorsqu'il est répertorié comme actif dans les équations logiques. L'appareil virtuel est toutefois une exception à cette règle. Lors de l'utilisation du périphérique virtuel, CUPL ignore la polarité dans la déclaration de broches. Dans ce cas, l'équation elle-même doit être niée.

De même, si un signal de sortie est LO de niveau actif, définissez la variable avec un point d'exclamation dans la déclaration de broche et écrivez l'équation logique sous une forme logiquement vraie. L'utilisation du point d'exclamation permet de déclarer des broches sans tenir compte des limites du type d'équipement cible. Avec le périphérique virtuel, l'équation elle-même doit être inversée, car le compilateur ignore la polarité dans la déclaration de broches.

Si une déclaration de broche spécifiant une sortie HI de niveau actif est compilée pour un périphérique cible (tel qu'un PAL16L8) qui n'a que des sorties inverses, CUPL exécute automatiquement le théorème s sur l'équation logique pour adapter la fonction au périphérique.

Prenons l'exemple suivant. Le fichier de description logique est écrit pour un périphérique PAL16L8. Toutes les broches de sortie sont déclarées comme active-HI. L'équation suivante a été écrite pour spécifier une fonction OR :

```
c = a # b ;
```

Toutefois, étant donné que le PAL16L8 contient un tampon inverseur fixe sur les broches de sortie, CUPL doit effectuer Par Organisation pour adapter la logique au périphérique. La CUPL génère le terme produit suivant dans le fichier de documentation (voir Formats des fichiers de documentation à l'Annexe C) :

```
c = > ! a & ! b
```

Figure 1.4 montre le processus décrit ci-dessus.

Si une conception contient des termes de produit excessifs, la CUPL affiche un message d'erreur et la compilation s'arrête. Le fichier de documentation (nom de fichier.DOC) répertorie le nombre de termes de produit requis pour implémenter la fonction logique et le nombre de termes de produit que le périphérique possède physiquement pour la broche de sortie particulière.

Voici quelques exemples de déclarations de code PIN valides :

```
pin 1          = clock ;              /* Register Clock */
pin 2          = ! enable ;           /* Enable I / O Port */
pin [3 ,4]     = ![ stop , go ];      /* Control Signals */
pin [5..7]     = [ a0 ..2];           /* Address Bits 0 -2 */
```

Les deux dernières lignes de l'exemple ci-dessus sont des notations abrégées pour les éléments suivants :

```
pin 3          = ! stop ;             /* Control Signal */
pin 4          = ! go ;               /* Control Signal */
pin 5          = a0 ;                 /* Address Bit 0 */
pin 6          = a1 ;                 /* Address Bit 1 */
pin 7          = a2 ;                 /* Address Bit 2 */
```

Pour le périphérique virtuel, les codes PIN peuvent être omis. Cela fournit un moyen de faire une conception sans tenir compte des restrictions liées à l'appareil. Le concepteur peut ensuite examiner les résultats et ainsi déterminer les exigences de mise en œuvre. L'appareil cible peut alors être choisi. Voici des déclarations de code confidentiel valides lors de l'utilisation du périphérique virtuel.

```
pin            = ! stop ;             /* Control Signal */
pin            = ! go ;               /* Control Signal */
pin            = a0 ;                 /* Address Bit 0 */
pin            = a1 ;                 /* Address Bit 1 */
pin            = a2 ;                 /* Address Bit 2 */
```

L'entrée, la sortie ou la nature bidirectionnelle d'une broche de périphérique n'est pas spécifiée dans la déclaration de broche. Le compilateur déduit la nature d'une broche de la façon dont le nom de la variable de broche est utilisé dans la spécification logique. Si la spécification logique et les caractéristiques physiques de l'appareil cible sont incompatibles, la fonction CUPL affiche un message d'erreur indiquant l'utilisation incorrecte de la broche.

1.1.10 Déclarations de déclaration de nœud

Certains périphériques contiennent des fonctions qui ne sont pas disponibles sur des broches externes, mais des équations logiques doivent être écrites pour ces fonctionnalités. Par exemple, le 82S105 contient à la fois des registres d'états enterrés (bascules) et un mécanisme permettant d'inverser tout terme de transition via un tableau de compléments. Avant d'écrire des équations pour ces bascules (ou tableaux de compléments), il faut leur attribuer des noms de variables. Comme aucune broche n'est associée à ces fonctions, le mot clé PIN ne peut pas être utilisé. Utilisez le mot clé **NODE** pour déclarer des noms de variables pour les fonctions enterrées.

Le format des déclarations de nœud est le suivant :

```
NODE [!] var ;
```

Où

NODE est un mot-clé permettant de déclarer un nom de variable pour une fonction enterrée.

! est un point d'exclamation facultatif pour définir la polarité du signal interne.

var est un nom de variable unique ou une liste de variables regroupées à l'aide de la notation de liste.

; est un point-virgule pour marquer la fin de l'instruction.

Placez les déclarations de nœuds dans la section « Déclarations et définitions de variables intermédiaires » du fichier source fourni par le fichier modèle.

La plupart des nœuds internes sont de niveau HI actif ; par conséquent, le point d'exclamation ne doit pas être utilisé pour définir la polarité d'un signal interne comme LO de niveau actif. L'utilisation du point d'exclamation entraîne presque toujours la génération par le compilateur d'un nombre significativement plus important de termes de produit. Une exception est le nœud de tableau de complément, qui, par définition, est un signal LO de niveau actif.

Bien qu'aucun numéro de broche ne soit indiqué dans l'instruction de déclaration, la **CUPL** attribue le nom de la variable à un pseudo-code **PIN** interne. Ces nombres commencent par le nombre le plus bas possible et sont définis séquentiellement même si un nœud a été affecté avec l'instruction **PINNODE**. L'affectation est automatique et déterminée par l'utilisation (bascule, tableau de compléments, etc.), de sorte que l'ordre variable n'est pas un problème. Toutefois, une fois qu'une variable de nœud est déclarée, une équation logique doit être créée pour la variable, sinon une erreur de compilation en résulte.

CUPL Utilise la déclaration de nœud pour distinguer une équation logique pour une fonction enterrée d'une expression intermédiaire.
Voici des exemples d'utilisation du mot-clé **NODE** :

```
NODE [ State0 ..5];          /* Internal State Bit */
NODE ! Invert ;              /* For Complement Array */
```

Une alternative pour assigner des fonctions enterrées au lieu de permettre à **CUPL** de les attribuer automatiquement via le mot-clé **NODE**, est d'utiliser le mot-clé **PINNODE**. Le mot-clé **PINNODE** est utilisé pour définir explicitement les nœuds enterrés en attribuant un numéro de nœud à un nom de variable symbolique. Ceci est similaire au fonctionnement des instructions de déclaration de code confidentiel. Le format d'une déclaration de nœud d'épingle est le suivant :

```
PINNODE node_n  =  [!] var ;
```

Où

PINNODE est un mot-clé permettant de déclarer les numéros de nœud et de leur attribuer des noms de variables.

node n est un nombre de nœuds décimaux ou une liste de numéros de nœuds regroupés à l'aide de la notation de liste; C'est

```
[ node \ _n1 , node \ _n2 ... node_nn ]
```

! est un point d'exclamation facultatif pour définir la polarité du signal interne.

= est l'opérateur d'affectation.

var est un nom de variable unique ou une liste de variables regroupées à l'aide de la notation de liste ; C'est

```
[ var , var ... var ]
```

; est un point-virgule utilisé pour marquer la fin de l'instruction.

Placez les déclarations de nœuds d'épingle dans la section « Déclarations et définitions de variables intermédiaires » du fichier source fourni par le fichier modèle.

Comme pour les déclarations de nœuds, la plupart des nœuds internes sont de niveau HI actif ; par conséquent, le point d'exclamation ne doit pas être utilisé pour définir la polarité d'un signal interne comme étant de niveau actif LO. L'utilisation du point d'exclamation entraîne presque toujours la génération par le compilateur d'un nombre significativement plus important de termes de produit. Une exception est le nœud de tableau du complément, qui est par définition un signal LO de niveau actif.

Une liste des numéros de nœuds pour tous les dispositifs contenant des nœuds internes figure à l'annexe D. Veuillez-vous référer à ces numéros de nœud pour les déclarations de nœuds d'épingle.

Voici des exemples d'utilisation du mot-clé PINNODE :

```
PINNODE [29..34]  =     [ State0 ..5];     /* Internal State Bits */
PINNODE 35        =     ! Invert ;         /* Complement Array */
PINNODE 25        =     Buried ;           /* Buried register part */
                                           /* of an I / O macrocell */
                                           /* with multiple */
                                           /* feedback paths */
```

1.1.11 Instructions de déclaration de champ binaire

Une déclaration de champ de bits attribue un nom de variable unique à un groupe de bits. Le format est le suivant :

```
FIELD var = [ var , var , ... var ] ;
```

Où

FIELD est un mot-clé.
var est un nom de variable valide.
[var, var, ... var] est une liste de noms de variables en notation de liste.
= est l'opérateur d'affectation.
; est un point-virgule utilisé pour marquer la fin de l'instruction.

✎Note: Les crochets n'indiquent pas les éléments facultatifs. Ils sont utilisés pour délimiter des éléments de la liste.

Placez les déclarations de champs binaires dans la section « Déclarations et définitions de variables intermédiaires » du fichier source fourni par le fichier modèle.

Après avoir attribué un nom de variable à un groupe de bits, le nom peut être utilisé dans une expression ; L'opération spécifiée dans l'expression est appliquée à chaque bit du groupe. Reportez-vous à la Section 1.2.10, Opérations définies, pour une description des opérations autorisées pour les instructions **FIELD**. L'exemple ci-dessous montre deux façons de référencer les huit bits d'entrée d'adresse (A0 à A7) d'un décodeur d'E/S en tant que variable unique nommée **ADDRESS**.

```
FIELD ADDRESS = [ A7 , A6 , A5 , A4 , A3 , A2 , A1 , A0 ] ;
```

ou

```
FIELD ADDRESS = [ A7 ..0] ;
```

Lorsqu'une instruction **FIELD** est utilisée, le compilateur génère un seul champ de 32 bits en interne. Ceci est utilisé pour représenter les variables dans le champ de bits. Chaque bit représente un membre du champ binaire. Le nombre de bits qui représente un membre d'un champ binaire est identique au numéro d'index si des variables indexées sont utilisées. Cela signifie que A0 occupera toujours le bit 0 dans le champ binaire. Cela signifie également que l'ordre d'apparition des variables indexées dans un champ de bits n'a aucune signification. Champ de bits déclaré comme [A0.. 7] est exactement identique à un champ de bits déclaré comme [A7.. 0]. En raison de ce mécanisme, différentes variables indexées ne doivent pas être incluses dans le même champ binaire. Un champ binaire contenant A2 et B2 affectera ces deux variables à la même position de bit. Cela entraînera la génération d'équations erronées.

En outre, les champs binaires ne doivent jamais contenir à la fois des variables indexées et non indexées. Cela entraînera presque certainement une génération erronée d'équations.

✎Note : Ne mélangez pas les variables indexées et non indexées dans un énoncé de champ. Le compilateur peut produire des résultats inattendus.

1.1.12 Déclarations MIN

L'instruction de déclaration MIN remplace, pour les variables spécifiées, le niveau de minimisation spécifié sur la ligne de commande lors de l'exécution de CUPL. Le format est le suivant :

MIN var [. ext] = level ;

Où

MIN est un mot-clé permettant de remplacer le niveau de minimisation de la ligne de commande.
var est une variable unique déclarée dans le fichier ou une liste de variables regroupées à l'aide de la notation de liste; C'est
[var , var , ... var]
.ext est une extension facultative qui identifie la fonction de la variable.
level est un entier compris entre 0 et 4.
; est un point-virgule pour marquer la fin de l'instruction.

Les niveaux 0 à 4 correspondent aux indicateurs d'option sur la ligne de commande, -m0 à -m4.

La déclaration MIN permet de spécifier différents niveaux pour différentes sorties dans la même conception, telles que aucune réduction pour les sorties nécessitant des termes de produit redondants ou contenus (pour éviter les risques asynchrones conditions) et réduction maximale pour une application de machine d'état.

Voici des exemples de déclarations MIN valides.

```
MIN async_out          = 0;      /* no reduction */
MIN [ outa , outb ]    = 2;      /* level 2 reduction */
MIN count . d          = 4;     /* level 4 reduction */
```

Notez que la dernière déclaration de l'exemple ci-dessus utilise l'extension .d pour spécifier que la variable de sortie enregistrée est celle à réduire.

1.1.13 Déclaration FUSE

L'instruction FUSE prévoit des cas particuliers où il est nécessaire de faire sauter des bits **TURBO** ou **MISER**. Cette déclaration doit être utilisée avec le plus grand soin, car elle peut entraîner des résultats imprévisibles si elle est mal utilisée.

FUSE (fusenumber , x)

où **fusenumber** est le numéro de fusible correspondant au **MISER** Bit ou **TURBO** Bit qui doit être soufflé, et **x** est égal à 0 ou à 1. Spécifiez 0 si le bit ne doit pas être soufflé. Spécifiez 1 pour faire sauter le bit. **Utilisez cette déclaration avec une extrême prudence.**

$DEFINE	$IFDEF	$UNDEF
$ELSE	$IFNDEF	$REPEAT
$ENDIF	$INCLUDE	$REPEND
$MACRO	$MEND	

Table 1.7: Preprocessor Commands

Dans cet exemple, le fusible 101 est un MISER Bit ou TURBO Bit. Cela souffle le fusible numéro 101.

FUSE (101 ,1)

N'ESSAYEZ PAS D'UTILISER CETTE DÉCLARATION POUR FAIRE SAUTER DES FUSIBLES ARBITRAIRES !

L'instruction fusible a été conçue pour souffler **MISER** bits et **TURBO** Bits seulement. Le numéro de fusible exact pour le **TURBO** ou **MISER** Bit doit être spécifié. Chaque fois que cette instruction est utilisée, **CUPL** générera un avertissement. Ceci est un rappel pour vérifier que le numéro de fusible spécifié est correct. Si un mauvais numéro de fusible est spécifié, des résultats désastreux peuvent se produire. Soyez très prudent en utilisant cette déclaration. Si le **FUSE statement** est utilisé dans une conception et des résultats étranges se produisent, Cochez l'icône numéro de fusible spécifié et assurez-vous qu'il est a **MISER** or **TURBO** Bit.

1.1.14 Commandes de préprocesseur

La partie préprocesseur de **CUPL** Fonctionne sur le fichier source avant qu'il ne soit transmis à l'analyseur et aux autres sections du compilateur. Les commandes du préprocesseur ajoutent l'inclusion du fichier, compilation conditionnelle, et des capacités de substitution de chaînes aux fonctionnalités de traitement source de **CUPL**. Table 1.7 Répertorie les commandes de préprocesseur disponibles. Chaque commande est décrite en détail dans cette section.

Le signe dollar **($)** est le premier caractère de toutes les commandes de préprocesseur et doit être utilisé dans la première colonne de la ligne. Toute combinaison de lettres majuscules ou minuscules peut être utilisée pour taper ces commandes.

$DEFINE

Cette commande remplace une chaîne de caractères par un autre opérateur, nombre ou symbole spécifié. Le format est le suivant :

$DEFINE argument1 argument2

où

argument1	is a variable name or special ASCII character.
argument2	is a valid operator, a number, or a variable name.

“Argument1” est remplacé par « argument2 » à tous les emplacements de la spécification source après le **$DEFINE** commande est donnée (ou jusqu'à ce que le préprocesseur rencontre un **$UNDEF** commande). Le remplacement est une substitution de chaîne littérale effectuée sur le fichier d'entrée avant d'être traitée par le compilateur **CUPL**. Notez qu'aucun point-virgule ou signe égal n'est utilisé pour cette commande.

Le **$DEFINE** permet de remplacer des nombres ou des constantes par des noms symboliques, par exemple :

```
$DEFINE         ON          b'1
$DEFINE         OFF         b'0
$DEFINE         PORTC       h'3 F0
```

Le **$DEFINE** permet également la création d'un ensemble personnel d'opérateurs logiques. Par exemple, les éléments suivants définissent un autre ensemble d'opérateurs pour la spécification logique :

```
$DEFINE  {      /*      Alternate Start Comment
$DEFINE  }      */      Alternate End Comment
$DEFINE  /      !       Alternate Negation
$DEFINE  *      &       Alternate AND
$DEFINE  +      #       Alternate OR
$DEFINE  :+:    $       Alternate XOR
```

✎Note: Les définitions ci-dessus figurent dans le **PALASM.OPR** fichier inclus avec le **CUPL** progiciel. Ce fichier peut être inclus dans le fichier source (voir **$INCLUDE** command) pour autoriser les équations logiques à l'aide de l'icône **PALASM** ensemble de symboles d'opérateurs logiques, ainsi que la norme **CUPL** symboles de l'opérateur.

$UNDEF

Cette commande inverse une commande **$DEFINE**. Le format est le suivant :

```
$UNDEF argument
```

Où

argument est un argument précédemment utilisé dans une commande **$DEFINE**.

Avant de redéfinir une chaîne de caractères ou un symbole défini avec la commande **$DEFINE**, utilisez la commande **$UNDEF** pour annuler la définition précédente.

$INCLUDE

Cette commande inclut un fichier spécifié dans la source à traiter par la CUPL. Le format est le suivant :

```
$INCLUDE filename
```

Où

filename est le nom d'un fichier dans le répertoire actif.

L'inclusion de fichiers permet de normaliser une partie d'une spécification couramment utilisée. Il est également utile pour conserver un fichier de paramètres distinct qui définit les constantes couramment utilisées dans de nombreuses spécifications source. Les fichiers inclus peuvent également contenir des commandes **$INCLUDE**, ce qui permet d'insérer des fichiers d'inclusion. Le fichier nommé est inclus à l'emplacement de la commande **$INCLUDE**.

Par exemple, la commande suivante inclut le **PALASM. OPR** dans un fichier source.

```
$INCLUDE PALASM . OPR
```