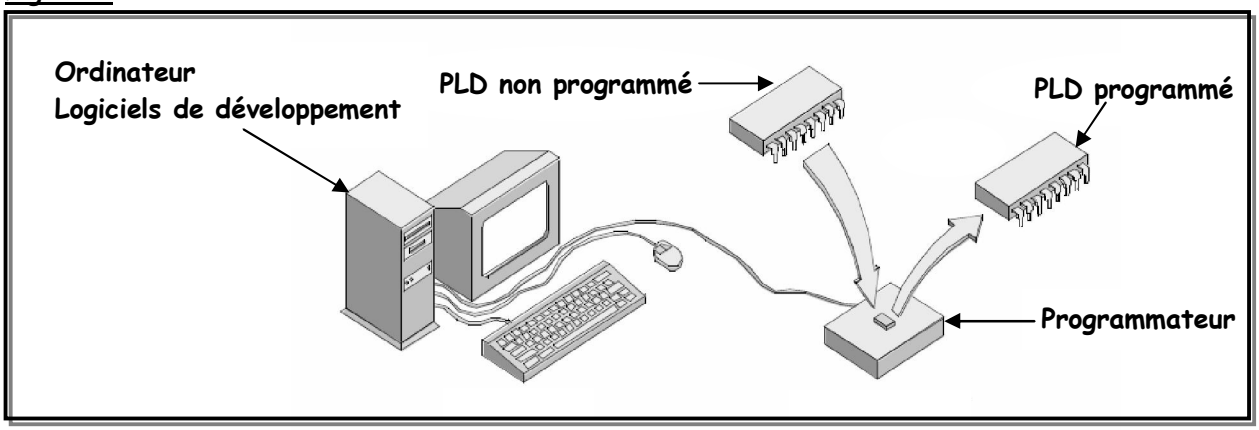
1. Introduction

La programmation des PLDs nécessite :

☑ Un logiciel de développement permettant de générer un fichier **JEDEC** à partir des données rentrées par l'opérateur. Ce fichier étant un ensemble de données binaires indiquant au programmateur les fusibles à "griller".
☑ Un programmateur permettant de "griller" les fusibles du PLD en fonction des données du fichier **JEDEC**. Il est en général associé à un logiciel de pilotage.

Figure 1



2. Programmation des PLDs

Lors de la programmation des PLDs, il est conseillé de suivre les démarches suivantes :

☑ Etude du système et élaboration des relations liant les entrées/sorties.
☑ Choix du composant en fonction des besoins du système : nombre d'entrées/sorties, structure des sorties, …
☑ Ecriture des modules de description : C'est le rôle du logiciel de développement. Il permet de créer le fichier de fusibles à griller(table de fusibles). Ce fichier est normalisé au format **JEDEC**.
Les moyens de description sont multiples, à savoir : Les logigrammes, les équations logiques, les tables de vérités, …
Les langages de description qui sont adaptés à la programmation des PLDs sont **ABEL**(langage de description matériel **HDL**) et **VHDL**(langage de description matériel de haut niveau). L'écriture d'un fichier **ABEL** ou **VHDL** ne nécessite qu'un éditeur de texte. Cependant les fabricants des circuits mettent des outils (logiciels) à la disposition des développeurs(programmeurs). Parmi ces logiciels, on trouve :
Synario de LATTICE, **Express d'ORCAD**, **ISP Design Expert** et **ISP Lever de LATTICE**,…
☑ Compilation : C'est le rôle du compilateur. Le document de description(fichier source) doit être traduit en un document objet compatible avec la programmation des composants. C'est le fichier **JEDEC** dans lequel un **0** ou un **1** signifie qu'un croisement ligne colonne de la matrice de programmation doit participer ou non au terme **ET**.
☑ Simulation : C'est le rôle du logiciel de simulation. Pendant cette phase, le programmeur procède à un test et une mise au point du circuit. Le simulateur logique permet donc de vérifier la programmation des circuits.
☑ Programmation : C'est le rôle du programmateur. Il permet de vérifier la virginité du circuit, lire le fichier au format **JEDEC**, programmer le circuit et vérifier la programmation.

3. Langage de description ABEL

31. Structure d'un fichier ABEL

On trouve cinq éléments :

☑ La section d'entête.
☑ La section de déclarations.
☑ La section de description.
☑ La section de test.
☑ La déclaration de fin de fichier.

311. Sections d'entête et de fin

Elles encadrent le document **ABEL**. L'entête commence par le mot **MODULE** suivi du nom qu'on veut lui donner, en général le nom du fichier **ABEL**. La section de fin est annoncée par le mot **END** et doit être suivie du même nom donné au module. La section d'entête peut comporter le mot **TITLE** suivi d'un texte entre apostrophes(facultatif).

312. Section de déclarations

Elle doit être précédée par le mot **DECLARATIONS** :

☑ Variables d'entrées.
Exemple : A, B, C, D **pin** 1, 2, 3, 4 ;

☑ Variables de sorties de type combinatoires.
Exemple : S1, S2 **pin** 10, 11 **istype** **'com'** ;

☑ Variables de sorties de type séquentielles.
Exemple : S1, S2 **pin** 10, 11 **istype** **'reg'** ;

313. Section de test

Elle annoncée par le mot **TEST_VECTORS**. Elle va permettre de tester de manière combinatoire ou séquentielle la fonction désirée par l'opérateur qui sera informé par l'échec ou la réussite du test.
(la section de test est facultative).

314. Section de description logique

3141. Description logique par équations

Elle est annoncée par le mot **EQUATIONS**. Elle permet de décrire sous forme d'équations, les relations liant les sorties aux entrées.
Exemple : S1=A & B ;
Les variables traitées par **ABEL** peuvent être de 1 bit ou un groupe de variables(bus).
Exemple : N=[A3, A2, A1, A0] ; S=[B7..B0] ;
La figure 2 représente quelques opérateurs manipulés par le langage **ABEL**.

3142. Description logique par table de vérité

Elle doit commencer par le mot **TRUTH_TABLE**.
La figure 3 montre un exemple de description logique par la table de vérité.

Figure 2

Opérateurs combinatoires

Opérateur	Noms	Syntaxe	Fonction obtenue
!	NON	C=!A	$\overline{A}$
&	ET	C=A & B	A.B
#	OU	C=A#B	A +B
$	OU exclusif	C=A$B	$A \oplus B$
!$	Non OU exclusif	C= A!$B	$\overline{A \oplus B}$

Opérateurs relationnels

==	Test d'égalité
!=	Test non égal
<	Test inférieur
<=	Test inférieur ou égal
>	Test supérieur
>=	Test supérieur

Opérateurs arithmétiques

-	Soustraction/négation en cpl2
+	Addition
*	Multiplication
/	Division entière non signée
%	Reste de la division entière
<<	Décalage à gauche
>>	Décalage à droite

Exemple

```
C= ! (A#B) ; // Fonction logique Non OU
C= !(A&B) ; // Fonction logique Non ET
S=-A ;
S=A+B ;
S=A<<1 ;
S=A==B ; // S prend la valeur de 1 si A=B
S=A<B ; // S prend la valeur de 1 si A<B
S=A>=B ; // S prend la valeur de 1 si A>=B
Q1 := A ; // Affectation d'une sortie séquentielle
```

Figure 3

```
N= [A3, A2, A1, A0] ;    // Définition de N
TRUTH_TABLE              // Définition par la table de vérité
(N -> [SA, SB, SC, SD, SE, SF, SG]) ;
0  -> [1,1,1,1,1,1,0] ;   // Affichage 0
1  -> [0,1,1,0,0,0,0] ;   // Affichage 1
2  -> [1,1,0,1,1,0,1] ;   // Affichage 2
3  -> [1,1,1,1,0,0,1] ;   // Affichage 3
...
```

Fonction Traiter

32. Exemples de programmation

321. Exemple de logique combinatoire

Figure 4

```
MODULE P1
TITLE 'Fonctions Logiques'
DECLARATIONS
A, B pin 2,3;                    // Variables d'entrées
S0, S1, S2, S3 pin 19, 18, 17, 16 istype 'com'; //Variables de sorties
EQUATIONS
S0=!A;      // Fonction NON
S1=A#B;     // Fonction OU
S2=A&B;     // Fonction ET
S3=A$B;     // Fonction XOR
END P1
```

322. Exemple de logique séquentielle

Figure 5

```
MODULE P2
TITLE 'Compteur 4 bits'
DECLARATIONS
Q3..Q0 pin 16, 17, 18,19 istype 'reg';      // Sorties du compteur
Q=[Q3..Q0];                                 // Q est un vecteur de variables de sorties
EQUATIONS
Q:=Q+1;
END P2
```