I Présentation :

ABEL-HDL est un langage de description de la structure à intégrer dans un PLD (HDL = Hardware Description Language) précurseur des langages VHDL et VERILOG.
Un fichier écrit en ABEL constitue la source d'outils qui génèrent les fichiers de programmation (fichier JEDEC) des PLD.

II Organisation d'un fichier ABEL :

L'exemple ci-dessous montre l'organisation d'un fichier ABEL:

module pap	indique sous quel nom seront créés les fichiers de programmation du PAL
title 'pas a pas'	titre du fichier (facultatif)
h,oe,r,m pin 1,13,2,3;	déclaration du brochage retenu
q1,q2,q3,q4 pin 23,22,21,20;	pour le PAL
sortie=[q1,q2,q3,q4];	définition d'un vecteur (facultatif)
sortie istype 'buffer,reg_d';	définition du type de sortie (ici séquentiel)
equations	définition du fonctionnement des
sortie.clk=h;	registres de sortie
sortie.oe=oe;	
sortie.ar=r;	
q1.d=!q2&!q4&!m#q3#!q4&m;	définitions des équations
q3.d=q&!q4#!q1&q3&!m#q2&m;	
q2.d=q4&!q1#q2&!q3&!m#q4&m;	
q4.d=q1&!q3#!q3&!q2&!m#q1&m;	
end	fin du fichier

III Fichier ABEL pour un PAL combinatoire :

Pour un fonctionnement combinatoire les sorties ne dépendent que des entrées . Les sorties doivent être déclarées de type combinatoire par la directive **istype** : par exemple ***Q0 istype 'com'*** . La directive **istype** affecte uniquement les sorties; les autres broches sont automatiquement considérées comme des entrées.
Certains PAL (ex P16L8) ont des sorties actives à l'état 0, c'est à dire que la sortie se fait par l'intermédiaire d'un inverseur; il faudra alors le préciser dans la directive **istype** de la façon suivante ***istype 'invert,com'***.

Si la sortie est active à l'état 1 l'argument est **buffer** par ex:

Q0 istype 'buffer,com'.
Si la sortie peut être indifféremment de type inversé ou non (cas du 16V8 en mode combinatoire) il est préférable de n'utiliser aucun argument (buffer ou invert); c'est le compilateur ABEL qui choisira le type permettant une optimisation de la structure interne.

III.1 Equations:

Les sorties peuvent être définies par des équations booléennes des entrées en utilisant les opérateurs suivants:

=	affectation (on affecte au terme de gauche l'équation définie à droite
!	complémentation logique
&	ET logique
#	OU logique
$	OU exclusif
!$	N OU exclusif

L'exemple ci-dessous est le fichier ABEL permettant de réaliser un décodage d'adresse pour un boîtier ROM placé de $2000 à $3000 et un boîtier RAM placé de $3000 à $4000 les sélection de boîtier étant actives à l'état 1.

```
module decodadr
title 'décodage d'adresses'
        A12,A13,A14,A15 pin 2,3,4,5;
        csrom,csram pin 13,12 istype 'com';
equations
        csrom=!A15&!A14&A13&!A12;
        csram=!A15&!A14&A13&A12;
end
```

Il est aussi possible de décrire le fonctionnement sous forme comportementale. Ainsi l'équation **Q=X & Y** peut s'écrire :
when **X==1 & Y==1 then Q=1; else Q=0;** les équations comportementales se prêtent mieux à des fonctionnements séquentiels (voir §V).

III.2 Tables de vérité:

Pour un montage comportant un grand nombre de sorties il peut être préférable d'utiliser des tables de vérité plutôt que des équations ce qui permet de s'affranchir du travail fastidieux de recherche des équations avec les outils traditionnels tels que les tableaux de Karnaugh.
L'exemple suivant présente un fichier ABEL permettant de réaliser un décodeur 3 vers 8 à sorties actives à l'état 1:

```
module decod
title 'décodeur 3 vers 8'
        a0,a1,a3 pin 2,3,4;
        q0,q1,q2,q3,q4,q5,q6,q7 pin 19,18,17,16,15,14,13,12 istype 'com';
        entrees=[ a0,a1,a3];
        on,off=1,0;
truth_table (entrees-> [ q0 , q1 , q2 , q3 ,  q4 , q5 ,  q6 ,  q7 ])
                        0   ->[ on , off , off , off , off , off , off , off ];
                        1   ->[ off , on , off , off , off , off , off , off ];
                        2   ->[ off , off , on , off , off , off , off , off ];
                        3   ->[ off , off , off , on , off , off , off , off ];
                        4   ->[ on , off , off , off , on , off , off , off ];
                        5   ->[ off , off , off , off , off , on , off , off ];
                        6   ->[ off , off , off , off , off , off , on , off ];
                        7   ->[ off , off , off , off , off , off , off , on ];
end
```

IV Fichier ABEL pour un PAL séquentiel:

Le fichier ABEL doit contenir les indications concernant la nature de la sortie (**reg_d** pour des sorties sur bascules D ou **reg_jk** pour des sorties sur bascules JK) et les entrées réalisant l'horloge des registres (les PAL sont majoritairement synchrones) la validation des sorties et la mise à 0 asynchrone (si elles existent). Cela est fait à l'aide des extensions qui doivent être définies dans la rubrique **equation** .

IV.1 Les extensions:

Les extensions caractérisent le fonctionnement des étages de sorties.

.AR Reset asynchrone
.CLK Horloge des registres de sortie
.D entrée D d'une bascule D
.J entrée J d'une bascule JK
.K entrée K d'une bascule JK
.OE Sortie trois états

L'exemple ci-dessous montre un extrait de fichier ABEL pour lequel :

- les sorties sont regroupées dans un vecteur appelé sortie .
- les sorties se font sur les sorties directes (ou non inversées) des bascules D .
- l'horloge des bascules est l'entrée appelée h .
- la validation des sorties est l'entrée appelée oe .
- la mise à 0 asynchrone est l'entrée appelé r .

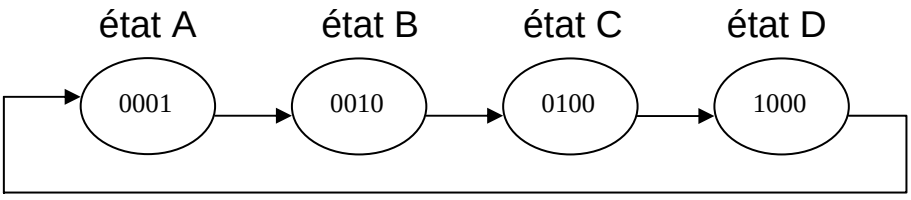
```
            sortie istype 'buffer,reg_d';
equations
            sortie.clk=h;
            sortie.oe=oe;
            sortie.ar=r;
```

IV.2 Description par des équations :

On écrit les équations des entrées des bascules (D J K AR OE) en utilisant les extensions et les opérateurs déjà définis .
L'exemple du § II montre l'écriture des équations d'un PAL séquentiel.
Cette solution convient parfaitement lorsque la mise en équation est simple et rapide ; pour des structures complexes il est préférable de travailler avec des machines d'états.

IV.3 Machines d'état:

Une machine d'état est une représentation symbolique de l'enchaînement séquentiel des états de sortie d'un système synchrone; ainsi la machine d'état correspondant à un compteur Johnson 4 bits serait la suivante :



Le fichier ABEL serait le suivant :

```
module johnson
title 'compteur Johnson 4 bits synchrone'
        h,en pin 2,3;
        q3,q2,q1,q0 pin 19,18,17,16 istype 'reg_d,invert';
        out=[q3,q2,q1,q0];
        A=1; B=2; C=4; D=8;
equations
        out.clk=h;
        out.oe=en;
state_diagram out
        state A:
        goto b;
        state B:
        goto C;
        state C:
        goto D;
        state D:
        goto A;
end
```

Dans l'exemple précédent le passage d'un état à un autre est inconditionnel. Un passage conditionnel utilise l'expression :

if *condition* then *nom_état*
else *nom_état*

exemple : **state X**:

```
            if en then Y;
            else A;
```

lorsque la machine d'état est en X si l'entrée **en** est à 1 alors on passe à l'état Y sinon on passe en A.

V Exemples :

Les exemples ci-dessous ne font apparaître que la partie "utile" du fichier ABEL; ne figurent pas l'en-tête et les déclarations du brochage et du type de sortie.

V.1 Comparateur 4 bits :

2 mots d'entrée X et Y de 4 bits chacun et 3 sorties Sup (X supérieur à Y), Inf (X inférieur à Y) et Egal (X=Y).

```
X=[X3..X0];
Y=[Y3..Y0];
equations
Sup=(X>Y);
Inf=(X<Y);
Egal=(X==Y);
```

V.2 Compteur 4 bits avec mise à 0 asynchrone :

4 sorties regroupées dans un vecteur appelé **out**; **r** est l'entrée de mise à 0 asynchrone active à 1 et H l'entrée d'horloge.

```
out=[q3..q0];
equations
out.clk = H;
when (r==1) then out : = 0; else out := out+1;
```

V.3 Verrou type D (D latch) :

D est l'entrée de donnée et **LE** l'entrée de verrouillage active à 1.

```
equations
When (LE==0) then Q :=D; else Q :=Q;
```

V.4 Bascule D avec mise à 0 asynchrone :

D est l'entrée de donnée, **H** l'entrée d'horloge et **R** l'entrée de mise à 0 active à 1.

```
equations
Q.clk = H;
Q.ar = R;
Q := D;
```