LE LANGAGE ABEL HDL

I Introduction: STRUCTURE DES FICHIERS' SOURCE' ABEL HDL

Les fichiers "source" construit pour le compilateur ABEL (VIEWPLD sur VIEWLOGIC) ont l'extension .ABL et ont la forme ci-dessous.

MODULE

L'instruction MODULE nomme le module et indiques si des variables intermédiaires sont utilisées

OPTIONS

L'nstruction OPTIONS suivies d'arguments contrôle la façon dont sera compilé le fichier source.

TITLE

L'instruction TITLE est utilisée pour donner un titre ou une description du module

ABEL-HDL Source File

Header

```
Module source2
Options '-trace wave'
Title 'Example of a Source File
       Lisa Matheson, Data I/O'
```

Declarations

```
Declarations
    dev1 DEVICE 'p16r4' ;
    in1, in2, in3, clk PIN ;
    all, none, other PIN ISTYPE 'reg' ;
    out = [all,none,other];
```

```
@MESSAGE 'Declarations completed'
```

Description

```
Equations
  out.clk = clk;
  all     := in1 & in2 & in3 ;
  none    := !in1 & !in2 & !in3 ;
  other   := (in1 & in2 & !in3) # (in1 & !in2 & in3)
           # (in1 & !in2 & !in3) # (!in1 & in2 & in3)
           # (!in1 & in2 & !in3) # (!in1 & !in2 & in3) ;
```

Test Vectors

```
Test_Vectors
    ([in1,in2,in3,clk ] -> [all,none,other])
     [ 1 , 1 , 1 ,.c. ] -> [ 1 ,  0 ,  0  ];
     [ 0 , 1 , 1 ,.c. ] -> [ 0 ,  0 ,  1  ];
     [ 1 , 0 , 0 ,.c. ] -> [ 0 ,  0 ,  1  ];
     [ 1 , 0 , 1 ,.c. ] -> [ 0 ,  0 ,  1  ];
     [ 0 , 0 , 0 ,.c. ] -> [ 0 ,  1 ,  0  ];
```

End

```
End source
```

DECLARATIONS

Déclarations associées aux noms de variables ou constantes, **Devices, Pins, Nodes, Constants, Macros** et **Sets** ainsi que leurs attributs à l'aide de **IsType**

DIRECTIVES

Les directives permettent des manipulations avancées dù fichier source pour la compilation et peuvent être placées à n'importe quel endroit où elle sont nécessaires.

EQUATIONS

Vous pouvez utiliser, équations booléennes, table de vérité ou diagrammes d'états pour décrire votre conception logique.

TEST_VECTORS

Les vecteur s de test sont utilisés pendant la phase de simulation pour s'assurer que votre conception fonctionne comme vous le souhaitez.

END

Terminaison du module et fin de compilation

DOCUMENT PROTEGE PAR UN OUTIL DE COPIE

1) Caractères ascii autorisés.

a -z , (lowercase alphabet)
A - Z (uppercase alphabet)'
0 - 9 (digits)

space <tab> ! @ # $? + & * ( ) -

_ = [{] } ; : ' "

' ~ \ | < > . / ^ %

2)Règles de base

_lignes de 150 caractères max.
_terminaison de ligne CR (Carriage Return)
_mots clefs, identificateurs, et nombres doivent être séparés par au moins un espace.(31 carac. max.)
_les identificateurs sont "case sensitive": 'Output' est différent de 'output'.
_une constante est encadrée par 2 points; exemple .P.

3) Mots clefs réservés

case	goto	property
déclarations	if	state
device	in (obsolete)	state_diagram
else	istype	test_vectors
enable (obsolete)	library	then
end	macro	title
endcase	module	trace
endwith	node	truth - table
équations	options	when
flag (obsolete)	pin	with
fuses		

4) constantes réservées.

Constant	Description
.C.	Clocked input (low-high-low transition)
.D.	Clock down edge (high-low transition)
.F.	Floating input or output signal
.K.	Clocked input (high-low-high transition)
.P.	Register preload
.SVn.	n = 2 through 9. Drive the input to super voltage 2 through 9.
.u.	Clock up edge (low-high transition) Don't care condition
.Z.	Tristate value

5) Commentaires

Il sont encadrés par le symbole " exemple: "ceci est un commentaire"

6) Représentation des nombres

Base Name	Base	Symbol
binary	2	^b
octal	8	^o
decimal	10	^d (default)
hexadécimal	6	^h

DOCUMENT PROTEGE PAR UN DROIT DE COPIE

exemples:

Specification	Decimal Value
75	75
^h75	117
^b101	5
^o17	15
^h0F	15

7) Chaînes de cacractères

Elles sont délimitées par ' exemples: 'ceci est une chaîne de caractères'

8) Expressions

Même règles qu'en pascal ou en C.

opérateurs logiques:

Operator	**Description**
!	NOT:ones complément
&	AND
#	OR
$	XOR.- exclusive OR
!$	XNOR: exclusive NOR

opérateurs arithmétiques

Operator	**Example**	**Description**
-	-A	twos complément (negation)
-	A-B	subtraction
+	A+B	addition
*	A*B	multiplication
/	A/B	unsigned integer division
%	A%B	modulus: remainder from /
<<	A<<B	shift A left by B bits
>>	A>>B	shift A right by B bits

opérateurs relationnels

Operator	**Description**
==	equal
!=	not equal
<	less than
<=	less than or equal
>	greater than
>=	greater than or equal

priorité des opérateurs

Priority	Operator	Description
1	-	negate
1	!	NOT
2	&	AND
2	<<	shift left

DOCUMENT PROTEGE PAR UN DROIT DE COPIE

2	>>	shift right
2	*	Multiply
2	/	unsigned division
2	%	modulus
3	+	add
3	-	subtract
3	#	OR
3	$	XOR. exclusive OR
3	!$	XNOR. exclusive NOR
4	==	equal
4	!=	not equal
4	<	less than
4	< =	less than or equal
4	>	greater than
4	> =	greater than or equal

exemples

Expression	Result	Comments
2 * 3/2	3	operators with same priority
2 *3 / 2	3	spaces are OK
2* (3/2)	2	fraction is truncated
2 + 3 *4	14	
2#4 $2	4	
2#(4 $2)	6	
2 == ^hA	0	false
14 == ^hE	-1	true

<u>assignation</u>

une distinction est faite en ABL pour les assignations où ,à droite de l'opérateur un oui plusieurs signaux sont de type à mémorisation (Registered) ou simplement combinatoire.

Operator	**Description**
=	Combinatorial assignment
:=	Registered assignment

exemple: equations

```
enable   =   a15 #  a14;
out.ar   =   clock & RESET;
valid    :=  out.q;
cumul.q  :=  basc1.d  &  basc2.d  &  !clock
```

<u>9) ensembles et intervalles (SET)</u>

la définition ressemble à celle du PASCAL; chaque constante ou variable est séparées par une virgule, le tout est encadré par des crochets([ ]).

un intervalle est définie par : [valeur init . . valeur fin]

exemples: [B0,B1,B2,B3,B4,B5,B6,B7]

[B0 . . B7]ce qui permet de faire des déclarations du type:

Addr = [A15,A14,A13]; "qui déclare l'ensemble (SET) Addr

et des équations comme:

Chip_Select = Addr == [1,0,1];

ou Chip_Select = Addr == 5;

DOCUMENT PROTEGE PAR UN DROIT DE COPIE

qui est fonctionnellement équivalent à la déclaration:

Chip_Select = A15 & !A14 & A13;

exemples d'utilisation:

Example 1

"declare some constants in declaration section
Addr = [a15 . .a0];
x = .x. "simplify notation for don't care constant
Chip_Sel = Addr == [1,0,1,x,x,x,x,x,x,x,x,x,x,x,x];

example 2

"declare some constants in declaration section
Addr = [a15 . .a0];
Chip-Sel = (Add >= ^hA000) & (Addr <= ^hBFFF);

II STRUCTURE DES FICHIERS SOURCES

) Déclarations des signaux (Signals declaration)

Déclaration de BROCHES (**PIN**)

Elle associe un nom de signal à un numéro (physique) de broche. Ce numéro dépend donc directement du type de PLD choisi .

exemple : **S1,S0 PIN 1,2** "S1 nom de signal associé à la broche 1"
"S0 nom de signal associé à la broche 2"

* Il est possible de déclarer à l'aide d'une extension au nom du signal (DOT extension) le type de signal (la fonction obtenue sur la broche)

exemple : **S2.Q PIN 3 ;**"le signal S2 est associé à la broche 3 du PLD et correspond à une sortie de bascule." (voir paragraphe DOT extension).

* Il est également possible de préciser la nature physique d'un signal (pas seulement logiquement) en précisant le type ce signal (IS Type)

exemple : **Output PIN 15 ISTYPE 'reg,invert';** "le signal Output associé à la broche 15 du PLD est une sortie de registre et possède une inversion entre la sortie de la bascule et la broche du circuit.

* Une dernière possibilité est de déclarer des signaux comme étant des PIN sans leur associer de numéro, laissant le maximum de liberté au complilateur et/ou au partionneur/ optimiseur pour choisir les broches lui-même.Ceci est fait dans le but de réduire la complexité des équations du "design", d'optimiser l'utilisation des fonctionnalités du/des PLD(s) et donc de réduire les temps de propagation des signaux qui sont les 3 soucis permanents du concepteur (et des fabricants) de PLDs.

Déclarations des NOEUDS enterrés (Buried **NODE**)

Idem que précédemment, pour les signaux internes (réels) non accessibles sur une broche que l'on appelle NOEUDS.

exemple.

A,B,C NODE; "A , B et C sont des Noeuds
Abs,CSE NODE Istype 'reg'; " les noeuds Abs et CSE sont des bascules enterrées

Déclarations de signal fictif (**ISTYPE**)

Ces signaux n'existent pas physiquement (seule une équation les définit) et sont introduits pour faciliter l'écriture et la lisibilité (la maintenance) du "design"(Not Buried NODES).

exemple: **Signal1,Signal2 Istype 'buffer';** 'les signaux Signal1 et Signal2 sont du type

DOCUMENT PROTEGE PAR UN DISPOSITIF DE COPIE

"amplifiés, sans inversion".

2)Attributs de signaux

Dot extension	Architec Indep.	Description
'buffer'		No Inverter in Target Device
'com'	X	Combinatorial
'invert'		Inverter in Target Device
'neg'	X	Complement signal
'pos'	X	Positive Polarity
'reg'	X	Clocked Memory Element
'reg_d'		D Flip-flop CIocked Memory
'reg_g'		D Flip-flop Gated Clock Memor Element
'reg_jk'		JK Flip-flop Clocked Memory E
'reg_sr'		SR Flip-flop Clocked Memory E
'reg_t'		T Flip-flop Clocked Memory El
'xor'		XOR Gate in Target Devioe

buffer' L'architecture cible ne possède pas d'inverseur entre la bascule associée (s'il en est) et la broche de sortie.

'invert' L'architecture cible possède un inverseur.
Le contrôle de l'inversion dans les composants est réalisée au travers de l'utilisation des attributs 'invert' et 'buffer'. Ces attributs force l'existance ('invert') ou l'absence ('buffer') d'un inverseur matériel à la broche du composant qui est associé au signal concerné.

Dans les composants "latchés", l'attribut 'invert' assure que l'inversion sera située entre la sortie de la bascule et la broche de sortie; ceci est important car la position de l'inversion affecte les conditions de "reset", "preset" et chargement (preload), ainsi que les conditions de démarrage lors de l'alimentation du circuit

'com' Indique un symbole combinatoire.

'neg' Complemente le signal avant de l'utiliser dans les équations sommes de produits.

'pos' Ne complemente pas le signal avant calcul. 'neg' est utilisé

fichier : c:\pm5\tse95-96\cours\abel_hdl.pm5
fichiers associés: sous designer 4.0: c:\designer\1995-96\cours\abel_hld.ds4 fichier exportés au format .WMF
fichiers scannés: sous scanjet: c:\scanjet\images\abel_hdl*.tif et *.pic

DOCUMENT PROTEGE PAR UN DROIT DE COPIE

typiquement avec l'option -reduce fixed pour contrôler la polarité de l'équation finale.
N'importe qu'elle équation associée à un tel signal sera optimisée avec cette polarité.

'reg' — Le signal spécifié correspond à une sortie "latchée".Les équations, les tables de verité et les diagrammes d'états genereront une logique orientée bascule D (D flip-flop), normalisée pour prendre en compte les inverseurs du composant cible.

'reg_d' — Le signal spécifié correspond à une sortie "latchée".Les équations, les tables de verité et les diagrammes d'états généreront une logique orientée bascule D (D flip-flop),mais il est nécessaire de spécifier si la sortie est inversée dans le composant cible (à l'aide des attributs 'invert' ou 'buffer').

'reg_g' — Le signal spécifié correspond à une sortie "latchée" transparent . Les équations, les tables de verité et les diagrammes d'états généreront une logique orientée bascule D (transparent D flip-flop),mais il est nécessaire de spécifier si la sortie est inversée dans le composant cible (à l'aide des attributs 'invert' ou 'buffer').Dans ce cas, équations et tables de vérité doivent préciser les extensions .D et .CE (Dots extensions)pour le signaux de ces bascules.

'reg_jk' — Le signal spécifié correspond à une sortie type bascule JK. Les diagrammes d'états génèrent la logique pour ce type de bascule, mais il est nécessaire de préciser si un inverseur est placé en sortie ('invert' ou 'buffer'). Les equations et les tables de vérité doivent être écrites avec les extensions .J et .K (Dots extensions) quand cet attribut est utilisé.

'reg_sr' — Le signal spécifié correspond à une sortie type bascule RS. Les diagrammes d'états génèrent la logique pour ce type de bascule, mais il est nécessaire de préciser si un inverseur est placé en sortie ('invert' ou 'buffer'). Les equations et les tables de vérité doivent être écrites avec les extensions .S et .R (Dots extensions) quand cet attribut est utilisé.

'reg_t' — Le signal spécifié correspond à une sortie type bascule T. Les diagrammes d'états génèrent la logique pour ce type de bascule, mais il est nécessaire de préciser si un inverseur est placé en sortie ('invert' ou 'buffer'). Les equations et les tables de vérité doivent être écrites avec l' extension .T(Dot extension) quand cet attribut est utilisé.

'xor' — Le signal spécifié sera réalisé en utilisant la porte OU exclusif à la sortie du composant.

DOCUMENT PROTEGE PAR UN INCOPIEUR

3)description logique du projet ("design").

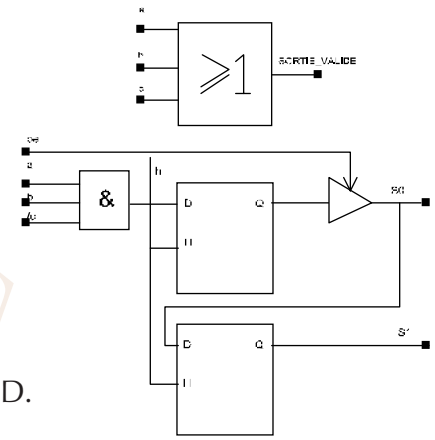
4 types principaux:

.Equations.
.Tables de verité.
.Diagrammes d'états.
.Plans fusibles (peu utilisés).

3.1 DESCRIPTION SOUS FORME D'EQUATIONS

EQUATION

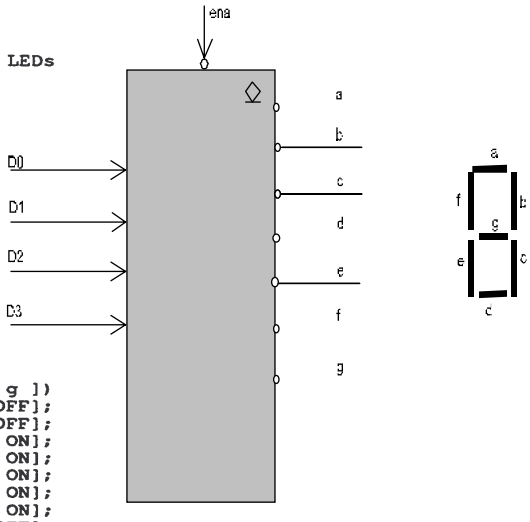
SORTIE_VALIDE = a#b#c;
S0.q = (a&b&!c) & oe;
S1.d = S0.pin;



3.2 DESCRIPTION SOUS FORME DE TABLE DE VERITE

On veut réaliser un décodeur type 7449 pour afficheur 7 segmentsà leds; il doit comporter une entrée de validation ena capable de mettre les sortie du décodeur en haute impédance. les entrées D0-D3 sont codées en BCD.

```
module  BCD7
title 'seven segment display decoder    1 Aug 1990
Walter Bright  Data I/O Corp  Redmond WA'
"          a
"         ---        BCD-to-seven-segment decoder similar to the 7449
"       f| g |b
"         ---            segment identification
"       e| d |c
"         ---
        bcd7 device  'P16P8';

        D3,D2,D1,D0,Ena pin 2,3,4,5,6;
        a,b,c,d,e,f,g   pin 13,14,15,16,17,18,19 istype 'com';

        bcd     = [D3,D2,D1,D0];
        led     = [a,b,c,d,e,f,g];

        ON,OFF  = 0,1;                  " for common anode LEDs
        L,H,X,Z = 0,1,.X.,.Z.;

equations
        led.oe = !Ena;

@dcset
truth_table (bcd -> [ a ,  b ,  c ,  d ,  e ,  f ,  g ])
             0  -> [ ON,  ON,  ON,  ON,  ON,  ON, OFF];
             1  -> [OFF,  ON,  ON, OFF, OFF, OFF, OFF];
             2  -> [ ON,  ON, OFF,  ON,  ON, OFF,  ON];
             3  -> [ ON,  ON,  ON,  ON, OFF, OFF,  ON];
             4  -> [OFF,  ON,  ON, OFF, OFF,  ON,  ON];
             5  -> [ ON, OFF,  ON,  ON, OFF,  ON,  ON];
             6  -> [ ON, OFF,  ON,  ON,  ON,  ON,  ON];
             7  -> [ ON,  ON,  ON, OFF, OFF, OFF, OFF];
             8  -> [ ON,  ON,  ON,  ON,  ON,  ON,  ON];
             9  -> [ ON,  ON,  ON,  ON, OFF,  ON,  ON];

test_vectors ([Ena,bcd] -> [ a ,  b ,  c ,  d ,  e ,  f ,  g ])
             [ 0 , 0 ] -> [ ON,  ON,  ON,  ON,  ON,  ON, OFF];
             [ 0 , 1 ] -> [OFF,  ON,  ON, OFF, OFF, OFF, OFF];
             [ 0 , 2 ] -> [ ON,  ON, OFF,  ON,  ON, OFF,  ON];
             [ 0 , 3 ] -> [ ON,  ON,  ON,  ON, OFF, OFF,  ON];
             [ 0 , 4 ] -> [OFF,  ON,  ON, OFF, OFF,  ON,  ON];
             [ 0 , 5 ] -> [ ON, OFF,  ON,  ON, OFF,  ON,  ON];
             [ 0 , 6 ] -> [ ON, OFF,  ON,  ON,  ON,  ON,  ON];
             [ 0 , 7 ] -> [ ON,  ON,  ON, OFF, OFF, OFF, OFF];
             [ 0 , 8 ] -> [ ON,  ON,  ON,  ON,  ON,  ON,  ON];
             [ 0 , 9 ] -> [ ON,  ON,  ON,  ON, OFF,  ON,  ON];
             [ 0 ,10 ] -> [ X ,  X ,  X ,  X ,  X ,  X ,  X ];
             [ 0 ,11 ] -> [ X ,  X ,  X ,  X ,  X ,  X ,  X ];
             [ 0 ,12 ] -> [ X ,  X ,  X ,  X ,  X ,  X ,  X ];
             [ 0 ,13 ] -> [ X ,  X ,  X ,  X ,  X ,  X ,  X ];
             [ 0 ,14 ] -> [ X ,  X ,  X ,  X ,  X ,  X ,  X ];
             [ 0 ,15 ] -> [ X ,  X ,  X ,  X ,  X ,  X ,  X ];
             [ 1 ,15 ] -> [ Z ,  Z ,  Z ,  Z ,  Z ,  Z ,  Z ];
end
```



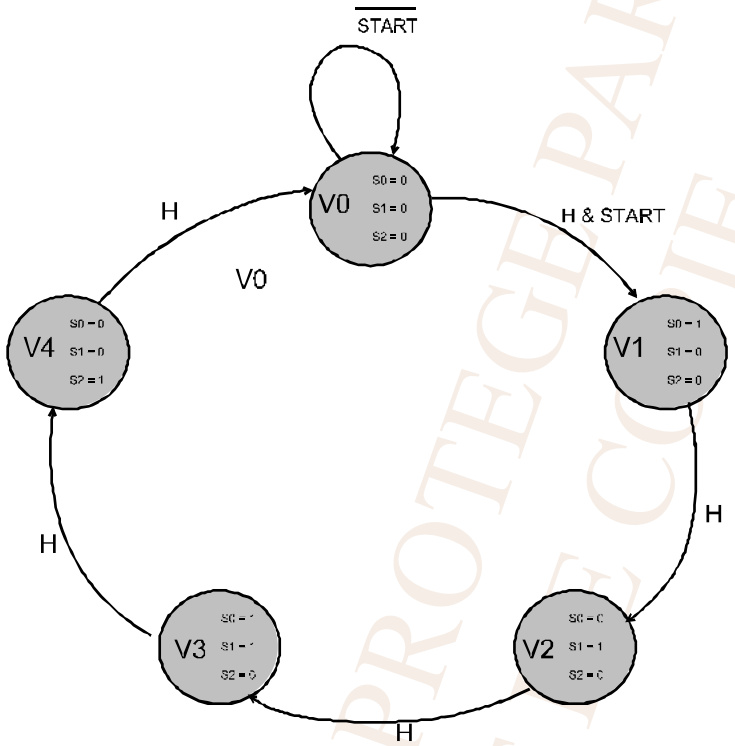
DOCUMENT PROTEGE PAR UN DROIT DE COPIE

fichier : c:\pm5\tse95-96\cours\abel_hdl.pm5
fichiers associés: sous designer 4.0: c:\designer\1995-96\cours\abel_hld.ds4 fichier exportés au format .WMF
fichiers scannés: sous scanjet: c:\scanjet\images\abel_hdl*.tif et *.pic

3.3 DESCRIPTION SOUS FORME DE DIAGRAMME D'ETAT.

On veut réaliser un compteur modulo 5 posssédant les sortie S0, S1 et S2 et une entrée d'horloge active sur front montant. Le compteur de peut démarrer que si l'entrée Départ_compteur (Start)est à 1.

Ce problème peut être traduit par le diagramme suivant:



traduit en langage ABEL ceci donne:

```
module compt5;
title 'machine detat compteur modulo 5  D Berquet  Decembre 1995';
        compt5 device   'P22V10';
        S0,S1,S2        PIN             23,22,21        Istype  'reg,buffer';
        H               PIN             1;
        START   pin             2;
EQUATIONS
        [S0,S1,S2].clk = H;
DECLARATIONS
        V0      = [0,0,0];
        V1      = 1;  " valeurs exprimées en décimal"
        V2      = 2;
        V3      = 4;
        V4      = 5;

STATE_DIAGRAM           [S2,S1,S0]
        STATE V0:
                If  START then V1;
        STATE V1:
                goto V2;
        STATE V2:
                goto V3;
        STATE V3:
                goto V4;
        STATE V4:
                goto V0;
END
```

DOCUMENT PROTEGE PAR UN DROIT DE COPIE

3.4 Plan fusible (FUSES):

Il est possible (mais peu utilisé) de désigner directement l'état des fusibles par une valeur;
un "0" désigne un fusible intact;
un "1" désigne un fusible "brulé"; un maximum de 32 fusibles peuvent être déclarés sur une même ligne.

exemple :
```
FUSES
    3552 =    1;
    [1..31]   = ^hFF
```

4) Section vecteurs de test (TEST_VECTORS)

Elle permet de définir des vecteurs de test qui appliqué au composant cible compilé, vont permettre de le tester afin de verifier si la synthèse du projet correspond bien au cahier des charges . Les vecteurs de tests sont optionnels mais sont régulièrement employés par les concepteurs de PLDs, c'est le seul moyen ,avant programmation, de s'assurer que tout fonctionne bien.

exemple :voir page précédente

5) Section Fin END

END nom_du_module termine le module et arrête la compilation.

6) Directives de compilation

Ce sont des mots réservés précédé par @, qui vont indiquer au compilateur ce qu'il doit faire et comment il doit le faire:
Available directives are listed below. See @ in the <<languageReference>> for complete information.

@ALTERNATE
@CONST
@DCSET
@EXPR
@EXIT
@IF
@IFB
@IFDEF
@IFIDEN
@IFNB
@IFENDEF
@IFNIDEN
@INCLUDE
@IRP
@IRPC
@MESSAGE
@ONSET
@PAGE
@RADIX
@REPEAT
@STANDARD

ABEL-HDL Language Reference

fichier : c:\pm5\tse95-96\cours\abel_hdl.pm5
fichiers associés: sous designer 4.0: c:\designer\1995-96\cours\abel_hld.ds4 fichier exportés au format .WMF
fichiers scannés: sous scanjet: c:\scanjet\images\abel_hdl*.tif et *.pic

DOCUMENT PROTEGE PAR UN DROIT DE COPIE

III REFERENCE DU LANGAGE(voir doc constructeur)extrait des principaux éléments:

1) Dot extension (.ext) (voir page 3.3 et suivante du document référence ABEL)Les signaux (au sens général: pin , node , signal) utilisés peuvent être accompagnés d'un attribut (dot extension) leur associant une fonction particulière. (voir liste des extensions ci dessous). .

Dot extension	Description
architecture -independant	
.CLK	Clock input to an edge-triggered flip-fl
.OE	Output enable.
.PIN	Pin fedback
.FB	Register feedback
architecture-specific	
.D	Data input t. a D-type flip-flop
.J	J input to a JK-type flip-flop
.K	K input to a JK-type flip-flop
.S	S inpt to - SR-typ. flip-flp
.R	R input to an SRtype flip-flop
.T	T input to a to a T type (toggle) flip-f
.Q	Regiter feedback
.PR	Register preset
.RE	Register reset
.AP	Asynchronous master preset
.AR	Asynchronous master reset
.SP	Synchronous register preset
.SR	Synchronous register reset
.LE	Latch-enable input to a latch
.LH	Latch- enable (high) to a latch
.LD	Register load input
.CE	Clock-enable input to a gated-clock flip
.FC	Flip-flop mode control

DOCUMENT PROTEGE PAR UN DROIT DE COPIE

'able 3-2 shows the dot extensions allowable and required for different egister types.

Register Type	Allowable Extensions	Definition
combinatorial logic (no register)	none	output
	.oe (opt)	output enable
	.pin (opt)	pin
D-type flip-flop	.clk (req)	clock
	.d (req)	data input
	.fc (opt)	flip-flop mode
	.oe (opt)	control
	.q (opt)	output enable
	.sp (opt)	f-f output
	.sr (opt)	sync preset
	.ap (opt)	sync reset
	.ar (opt)	async preset
	.pin (opt)	async reset
		pin
JK-type flip-flop	.clk (req)	clock
	.j (req)	j input
	.k (req)	k input
	.fc (opt)	flip-flop mode
	.oe (opt)	control
	.q (opt)	output enable
	.sp (opt)	f-f output
	.sr (opt)	sync preset
	.ap (opt)	sync reset
	.ar (opt)	async preset
	.pin (opt)	async reset
		pin
SR-type flip-flop	.clk (req)	clock
	.s (req)	set input
	.r (req)	reset input
	.oe (opt)	output enable
	.q (opt)	f-f output
	.sp (opt)	sync preset
	.sr (opt)	sync reset
	.ap (opt)	async preset
	.ar (opt)	async preset
	.pin (opt)	pin
T-type flip-flop	.clk (req)	clock
	.t (req)	toggle input
	.oe (opt)	output enable
	.q (opt)	f-f output
	.sp (opt)	sync preset
	.sr (opt)	sync reset
	.ap (opt)	async preset
	.ar (opt)	async reset
	.pin (opt)	pin
G-type latch	.d (req)	data input
	.le or .lh (req)	latch enable
	.oe (opt)	output enable
	.q (opt)	f-f output
	.pin	pin

DOCUMENT PROTEGE PAR UN DROIT DE COPIE

fichier : c:\pm5\tse95-96\cours\abel_hdl.pm5
fichiers associés: sous designer 4.0: c:\designer\1995-96\cours\abel_hld.ds4 fichier exportés au format .WMF
fichiers scannés: sous scanjet: c:\scanjet\images\abel_hdl*.tif et *.pic

Figures 3-1 through 3-8 show the effect of each dot extension.

Figure 3-1
Pin-to-pin Dot Extensions in an Inverted Output Architecture

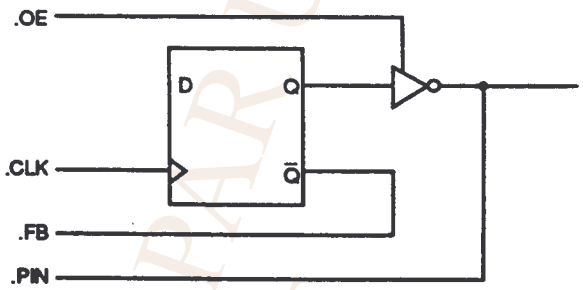


Figure 3-2
Pin-to-pin Dot Extensions in a Non-inverted Output Architecture

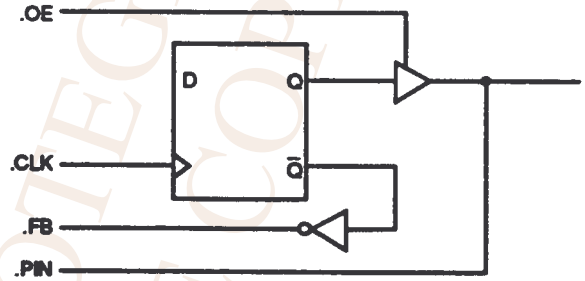


Figure 3-3
Detailed Dot Extensions for a D-type Flip-flop Architecture

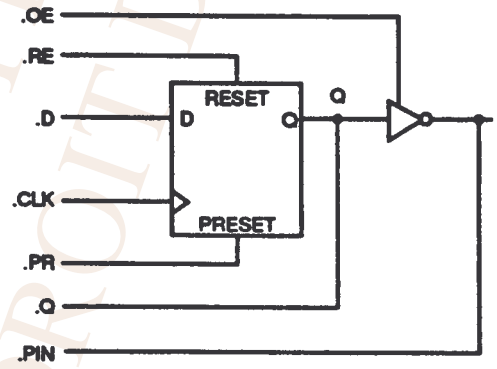
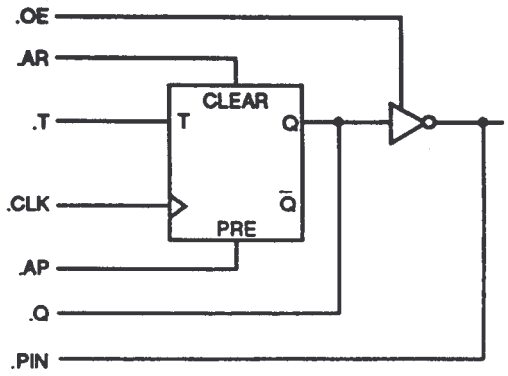


Figure 3-4
Detailed Dot Extensions for a T-type Flip-flop Architecture



DOCUMENT PROTEGE PAR LUN
DROIT DE COPIE

Figure 3-5
Detailed Dot Extensions for an RS-type Flip-flop Architecture

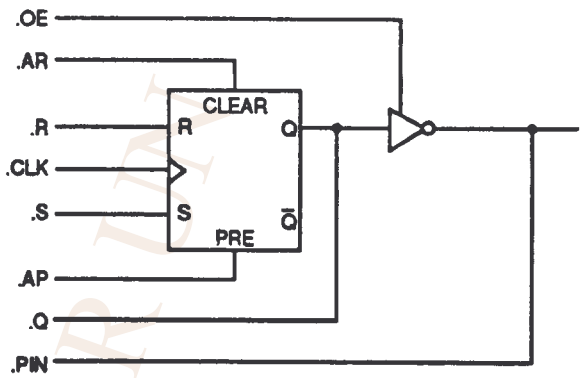


Figure 3-6
Detailed Dot Extensions for a JK-type Flip-flop Architecture

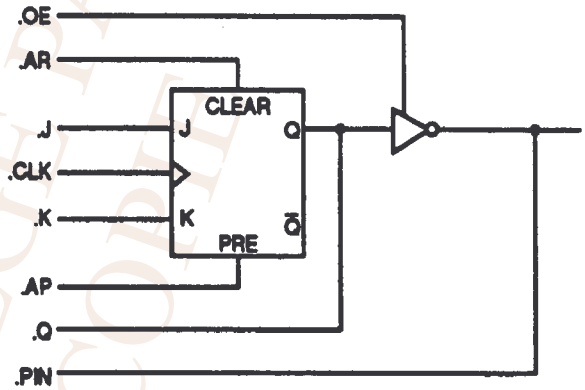
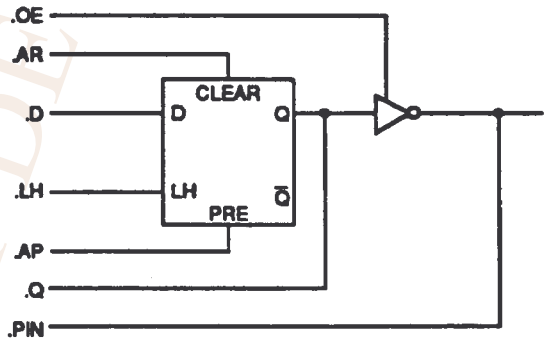
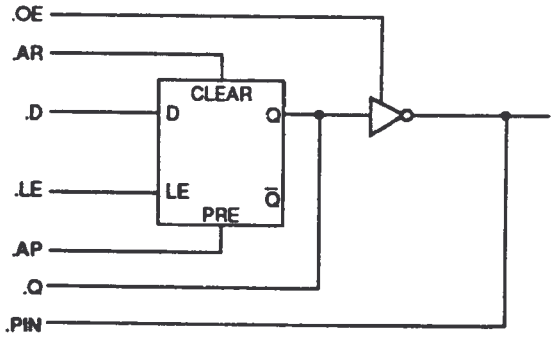


Figure 3-7
Detailed Dot Extensions for a Latch with Active High Latch Enable



LH	D	Q
1	0	0
1	1	1
0	X	Last Q

Figure 3-8
Detailed Dot Extensions for a Latch with Active Low Latch Enable



LE	D	Q
0	0	0
0	1	1
1	X	Last Q

DOCUMENT PROTEGE PAR LE DROIT DE COPIE

fichier : c:\pm5\tse95-96\cours\abel_hdl.pm5
fichiers associés: sous designer 4.0: c:\designer\1995-96\cours\abel_hld.ds4 fichier exportés au format .WMF
fichiers scannés: sous scanjet: c:\scanjet\images\abel_hdl*.tif et *.pic

2) déclarations des constantes.

Une constante est déclarée par un identificateur suivi d'un signe = et d'une valeur numérique (ou une expression) exemple:***Nom_Const = Val;***

```
ABC        =    3*17;            "ABC prend la valeur 51"
y          =    'Bc'             "Y = + H4263 ;
X          =    .X.;             "X signifie indifférent (voir constantes spéciales)"
ADDR       =    [1,0,15];        "ADDR est un ensemble de  3 éléments"
A, B, C    =    5, [ 1, 0], 6;   "3 déclarations de constantes"
```

avec

```
D PIN 6;
E          =    [5 * 7,D];       "les noms de signaux peuvent être inclus, Eest un
                                  "ensemble de 2 constantes dont la 2eme prend la
                                  "valeur logique de la broche 6.
G          =    [1,2]+[3,4];     "les opérations sur les ensembles sont autorisées
A          =    B & C;           "de même que les opérations sur les identificateurs
A          =    [!B,CI;
```

voir constantes spéciales (page 2).

N'utilisez pas trop de variables intermédiares: exemple

```
 Declarations
TMP1   =  [A3..A0]    ==    [B3..BO];  "    ==    test égalité"
TMP2   = [A7..A4]     ==    [B7..B4];

Equations
F = TMP1 & TMP2;
```

est traduit par le compilateur pendant la phase d'optimisation en

```
F = (A7 !$ B7) & (A6  !$   B6) (A5  !$  B5) & (A4  !$ B4)
(A3 !$ B3) & (A2 !$ B2) (Al !$ Bl) & (A0 !$ B0);
```

ce qui ne mobilise aucune broche du PLD mais nécessite 16 termes "produit"

la déclaration:

```
declarations
TMP1,TMP2 pin 18,19
Equations
TMP1   =  [A3..A0]    ==    [B3..BO];
TMP2   = [A7..A4]     ==    [B7..B4];
F = TMP1 & TMP2;
```

conduit à

```
TMP1 = (A7 !$ B7) & (A6 !$ B6) (A5 !$  B5) & (A4  !$ B4);
TMP2 = (A3 !$ B3) & (A2 !$ B2) (Al  !$ Bl) & (A0 !$ B0);
F =TMP1 &TMP2;
```

ce qui mobilise 2 PIN mais réduit le nombre de produits de moitié.

3) **IF expr. THEN State_exp.**

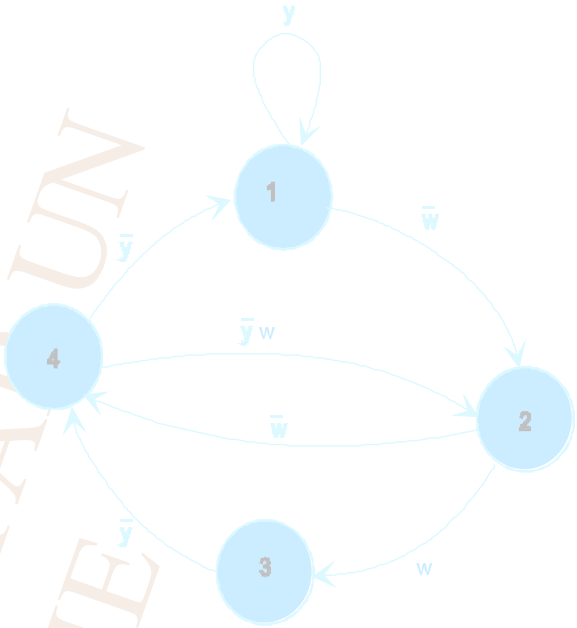
[ELSE state_exp.]

Utilisée uniquement dans les définitions par diagramme d'état pour indiquer la progression dans le diagramme et la (les) conditions(s) de transition. N'importe quel nombre de If peuvent être imbriqué , dans ce cas seul la dernière instruction se termine par un ; indiquant la fin du IF

DOCUMENT PROTEGE PAR UN DROIT DE COPIE

exemple:

```
etat  =     [a,bc];
state_diagram etat
state 1     : a=0;
              b=1;
              c=0
            if  (y==1) & (w==0) then 2;
state 2     :etat =[0,1,0];
            if  (y==1) & (w==1) then 3
            else 4;
state 3     :etat=[0,1,1];
            if  (y==0) & (w==1) then 3;
state 4     :etat=[1,0,0];
            if  (y==0) & (w==0) then 1
            else if  (y==0) & (w==1) then 2;
```



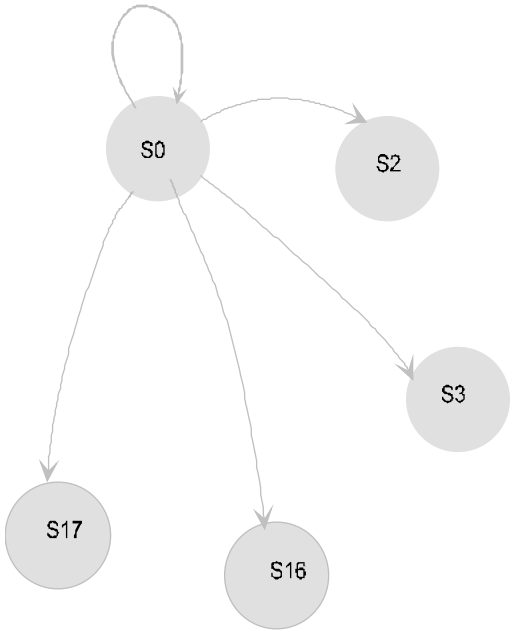
exercice:Donnez le diagramme correspondant

4) CASE

Cette instruction resssemble à l'instruction CASE OF du Pascal. Les instructions IF et CASE peuvent être imbriquées.

exemple:

```
STATE S0:
    CASE (select ==1): IF (address == ^h0100)
                       THEN S16
                       ELSE
                       IF (address > ^hE100)
                       THEN S17
                       ELSE
                       S0;

         (select == 2): S2;

         (select == 3): IF (address <= ^hE100)
                            THEN IF (reset)
                                 THEN S3;
                                 ELSE S0;
                       ELSE S17;

         (select == 0): S0;
    ENDCASE;
```



exercice:traduire cette partie de diagramme d'état sous forme graphique:

5)GOTO

s'utilise uniquement dans les diagrammes d'états

exemple : GOTO S0;

6)MACRO

Permet de définir une "macro" (macro-commande), évite de répeter un segment de

DOCUMENT PROTEGE PAR UN DROIT DE COPIE

code qui revient souvent dans un programme
exemple:

```
module mac
title 'Demonstrates difference between MACRO and declared equations'
            m1 device 'P16H8';
            A,B,C,X1,X2,X3 pin 1,2,3,14,15,16;
            Y1 macro {B # C};
            Y2 = B # C;

equations
         X1 = A & Y1;
         X2 = A & (Y1);
         X3 = A & Y2;

"    Note: Because Y1 is a text replacement macro the equation
"    for Xi vill expand to A & B # C. If the deaired function
"    was A & (B # C) use parentheses around the macro or use
"    a sub expression in the declarations.

"    The macro could also be vritten           Y1 macro {(B # C)};

test_vectors     ([A,B,C]     ->  [Xl,X2,X3])

                 [0,0,0]      ->  [0,0,0],
                 [0,0,1]      ->  [l,0,1];
                 [0,1,0]      ->  [0,0,0];
                 [0,1,1]      ->  [1,0,0];
                 [1,0,0]      ->  [0,0,0];
                 [1,0,1]      ->  [1,1,1];
                 [1,1,0]      ->  [1,1,1];
                 [1,1,1]      ->  [1,1,1];
 end
```

7) LIBRARY

Cette commande permet d'inclure le contenu du fichier spécifié dans le source ABEL en cours: exemple LIBRARY 'name' ; name correspond au nom du fichier à inclure(sans extension; ABEL ajoute l'extension ***.inc*** après name ; l'insertion se fait à partir de l'endroit où library est rencontré.

exemple de developpements:

```
module compt5;
      title 'machine détat compteur modulo 5 D Berquet Décembre 1995';
            compt5     device     'P22V10';
            S0,S1,S2   PIN        23,22,21   Istype  'reg,buffer';
            H          PIN        1;
            START      pin        2;
      EQUATIONS
            [S0,S1,S2].clk = H;
      DECLARATIONS
            V0    = [0,0,0];
            V1    = 1;  " valeurs exprimées en décimal"
            V2    = 2;
            V3    = 4;
            V4    = 5;
```

DOCUMENT PROTEGE PAR UN DROIT DE COPIE

```
STATE_DIAGRAM      [S2,S1,S0]
        STATE V0:
                If  START then V1;
        STATE V1:
                goto V2;
        STATE V2:
                goto V3;
        STATE V3:
                goto V4;
        STATE V4:
                goto V0;
END
```

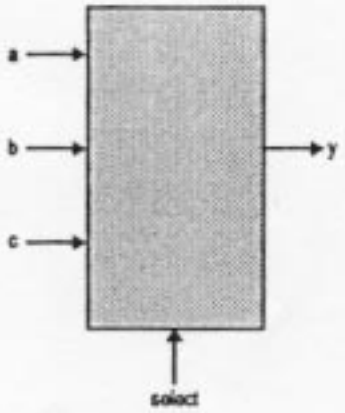
Exercice de synthèse:

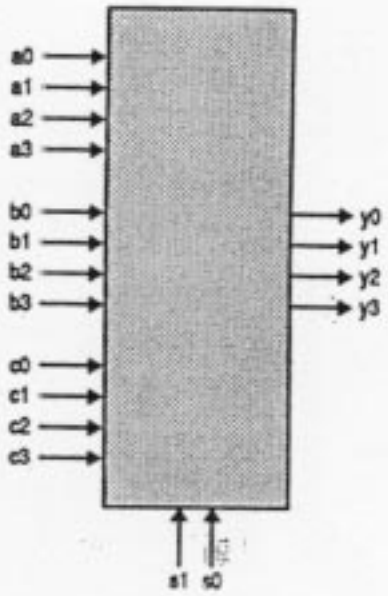
Reprenez les spécifications du séquenceur du thème de BAC 1995-96; décrivez, en source ABEL HDL ,ce séquenceur en n'imposant pas de brochage particulier au compilateur de façon à ce qu'il vous propose plusieurs solutions matérielles de réalisation.

Faites votre choix de composant en fonction :
_du prix
_de l'efficacité
_de la disponibilité
_des moyens de programmation dont vous disposez.

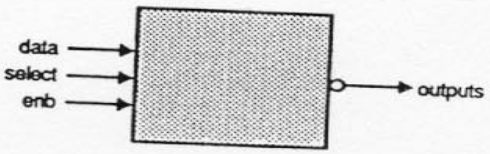
DOCUMENT PROTEGE PAR UN DROIT DE COPIE

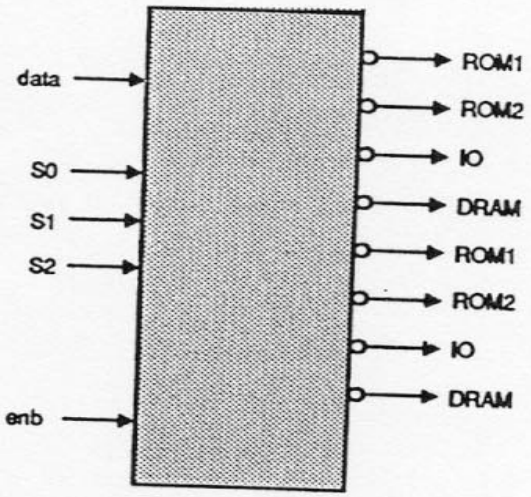
fichier : c:\pm5\tse95-96\cours\abel_hdl.pm5
fichiers associés: sous designer 4.0: c:\designer\1995-96\cours\abel_hld.ds4 fichier exportés au format .WMF
fichiers scannés: sous scanjet: c:\scanjet\images\abel_hdl*.tif et *.pic






```
module Mux12T4
title '12 to 4 multiplexer   5 June 1990
Dave Pellerin   Data I/O Corp.  Redmond WA'

        mux12t4         device  'P16V8S';

        a0,a1,a2,a3     pin     1,2,3,4;
        b0,b1,b2,b3     pin     5,6,7,8;
        c0,c1,c2,c3     pin     9,11,12,13;
        s1,s0           pin     18,19;
        y0,y1,y2,y3     pin     14,15,16,17;

        H       =       [1,1,1,1];
        L       =       [0,0,0,0];
        X       =       .x.;
        select  =       [s1, s0];
        y       =       [y3,y2,y1,y0];
        a       =       [a3,a2,a1,a0];
        b       =       [b3,b2,b1,b0];
        c       =       [c3,c2,c1,c0];

equations
        when (select == 0) then y = a;
        when (select == 1) then y = b;
        when (select == 2) then y = c;
        when (select == 3) then y = c;

test_vectors ([select, a, b, c] -> y)
               [0     , 1, X, X] -> 1;"select = 0, gates lines a to output
               [0     ,10, H, L] -> 10;
               [0     , 5, H, L] -> 5;
               [1     , H, 3, H] -> 3;"select = 1, gates lines b to output
               [1     ,10, 7, H] -> 7;
               [1     , L,15, L] -> 15;
               [2     , L, L, 8] -> 8;"select = 2, gates lines c to output
               [2     , H, H, 9] -> 9;
               [2     , L, L, 1] -> 1;
               [3     , H, H, 0] -> 0;"select = 3, gates lines c to output
               [3     , L, L, 9] -> 9;
               [3     , H, L, 0] -> 0;
end
```

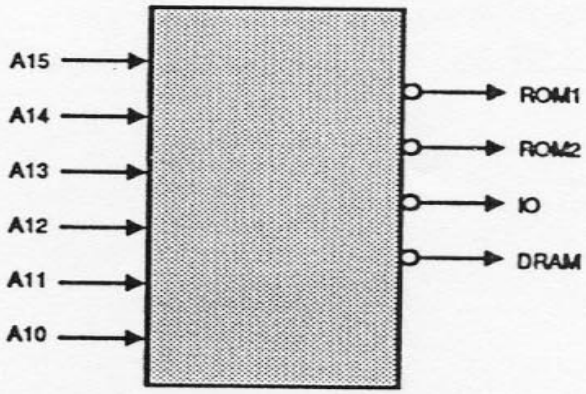


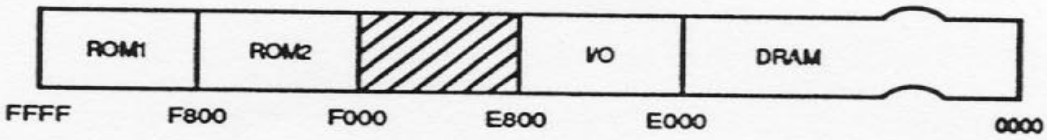



```
module DMUX1T8
title '1 to 8 line demultiplexer
Susan Todd and Mark Kuenster   Data I/O Corp.  5 Aug 1990'

                dmux1t8 device  'P16L8';

        y0,y1,y2,y3,y4,y5,y6,y7 pin     12,13,14,15,16,17,18,19;
        s0,s1,s2,data,enb       pin     1,2,3,4,5;

        H,L,Z    =        1,0,.Z.;
        select   =        [s2, s1, s0];
        outputs  =        [y7, y6, y5, y4, y3, y2, y1, y0];

equations
        !y0      = (select == 0) & data;
        !y1      = (select == 1) & data;
        !y2      = (select == 2) & data;
        !y3      = (select == 3) & data;
        !y4      = (select == 4) & data;
        !y5      = (select == 5) & data;
        !y6      = (select == 6) & data;
        !y7      = (select == 7) & data;

        outputs.oe = enb;

test_vectors  'Test the demuliplexer with a high input'
        ([enb,select,data] -> [y7,y6,y5,y4,y3,y2,y1,y0])
         [ H ,   0  , H  ] -> [H, H, H, H, H, H, H, L];  "Select y0
         [ H ,   1  , H  ] -> [H, H, H, H, H, H, L, H];  "Select y1
         [ H ,   2  , H  ] -> [H, H, H, H, H, L, H, H];  "Select y2
         [ H ,   3  , H  ] -> [H, H, H, H, L, H, H, H];  "Select y3
         [ H ,   4  , H  ] -> [H, H, H, L, H, H, H, H];  "Select y4
         [ H ,   5  , H  ] -> [H, H, L, H, H, H, H, H];  "Select y5
         [ H ,   6  , H  ] -> [H, L, H, H, H, H, H, H];  "Select y6
         [ H ,   7  , H  ] -> [L, H, H, H, H, H, H, H];  "Select y7
         [ L ,   0  , H  ] -> [Z, Z, Z, Z, Z, Z, Z, Z];

test_vectors  'Test the demuliplexer with a low input'
        ([enb,select,data] -> [y7,y6,y5,y4,y3,y2,y1,y0])
         [ H ,   0  , L  ] -> [H, H, H, H, H, H, H, H];  "Select y0
         [ H ,   1  , L  ] -> [H, H, H, H, H, H, H, H];  "Select y1
         [ H ,   2  , L  ] -> [H, H, H, H, H, H, H, H];  "Select y2
         [ H ,   3  , L  ] -> [H, H, H, H, H, H, H, H];  "Select y3
         [ H ,   4  , L  ] -> [H, H, H, H, H, H, H, H];  "Select y4
         [ H ,   5  , L  ] -> [H, H, H, H, H, H, H, H];  "Select y5
         [ H ,   6  , L  ] -> [H, H, H, H, H, H, H, H];  "Select y6
         [ H ,   7  , L  ] -> [H, H, H, H, H, H, H, H];  "Select y7
         [ L ,   0  , L  ] -> [Z, Z, Z, Z, Z, Z, Z, Z];
end
```


fichier : c:\pm5\tse95-96\cours\abel_hdl.pm5
fichiers associés: sous designer 4.0: c:\designer\1995-96\cours\abel_hld.ds4 fichier exportés au format .WMF
fichiers scannés: sous scanjet: c:\scanjet\images\abel_hdl*.tif et *.pic

y





The address ranges associated with each section of memory are shown below. These address ranges can also be seen in the source file in Figure 5-3.

Memory Section	Address Range (hex)
DRAM	0000-DFFF
I/O	E000-E7FF
ROM2	F000-F7FF
ROM1	F800-FFFF

```
module M6809A
title '6809 memory decode  Jean Designer  Data I/O Corp Redmond WA'

                        m6809a  device  'P14L4';
        A15,A14,A13,A12,A11,A10 pin 1,2,3,4,5,6;
        ROM1,IO,ROM2,DRAM       pin 14,15,16,17;
        H,L,X    = 1,0,.X.;
        Address = [A15,A14,A13,A12, A11,A10,X,X, X,X,X,X, X,X,X,X];

equations
        !DRAM    = (Address <= ^hDFFF);
        !IO      = (Address >= ^hE000) & (Address <= ^hE7FF);
        !ROM2    = (Address >= ^hF000) & (Address <= ^hF7FF);
        !ROM1    = (Address >= ^hF800);

test_vectors
              (Address -> [ROM1,ROM2,IO,DRAM])
               ^h0000  -> [  H,  H,  H,   L ];
               ^h4000  -> [  H,  H,  H,   L ];
               ^h8000  -> [  H,  H,  H,   L ];
               ^hC000  -> [  H,  H,  H,   L ];
               ^hE000  -> [  H,  H,  L,   H ];
               ^hE800  -> [  H,  H,  H,   H ];
               ^hF000  -> [  H,  L,  H,   H ];
               ^hF800  -> [  L,  H,  H,   H ];
end M6809A
```

A.1.4. A l'aide du schéma (Figure A3.1), compléter le fichier ABEL permettant la programmation du circuit **IW05** et indiquer sur le schéma partiel figure A2, le câblage de **IW05**.

```
module IW05
title 'Adressage OT1'



Declarations
IW05 device 'P16L8';
RW,E,A15,A14,A13,. . . pin  . . . . . . . . . . ;
EPROM1,RAM,LATCH,VE,MR pin  . . . . . . . . . . istype'com';
EPROM2. . . . . . . . . . . . . . . . . . . . . . . . . . . . .



Equations
!EPROM1 = A15 & A14 & E;
!RAM = !A15 & A14 & E;
LATCH = !A15 & !A14 & A13 & E;
!VE = (E  & !RW);
MR = !Reset ;
. . . . . .



end IW05
```

IW05

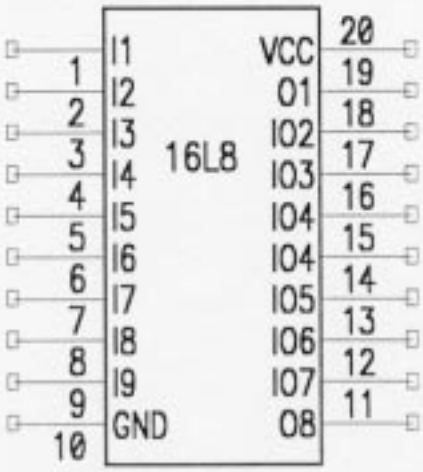
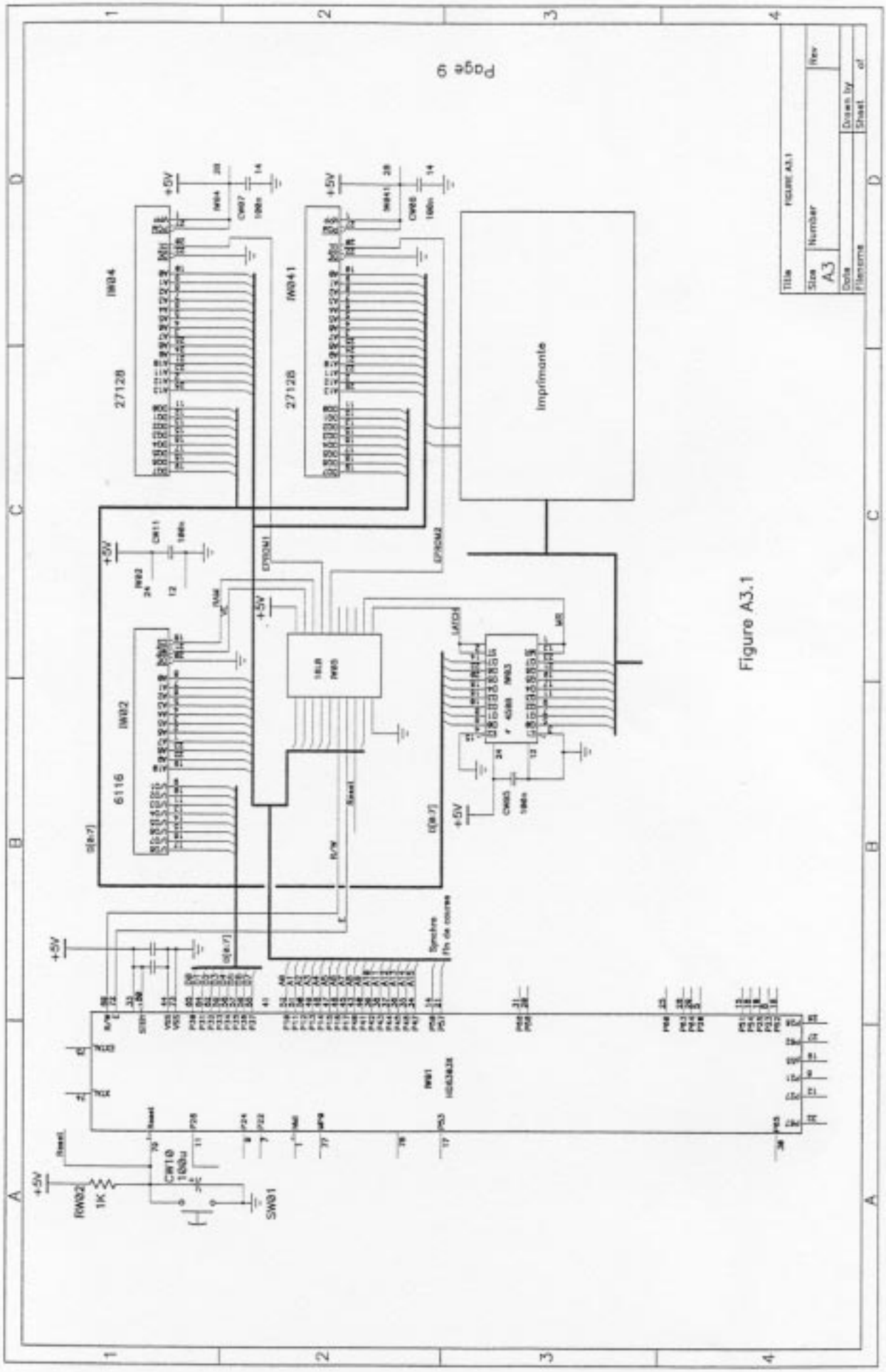


Figure A2

fichier : c:\pm5\tse95-96\cours\abel_hdl.pm5
fichiers associés: sous designer 4.0: c:\designer\1995-96\cours\abel_hld.ds4 fichier exportés au format .WMF
fichiers scannés: sous scanjet: c:\scanjet\images\abel_hdl*.tif et *.pic


Page 9
Imprimante
27128
6116
Figure A3.1

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 pour le PAL
q1,q2,q3,q4 pin 23,22,21,20;	
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 registres de sortie
sortie.clk=h;	
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 .

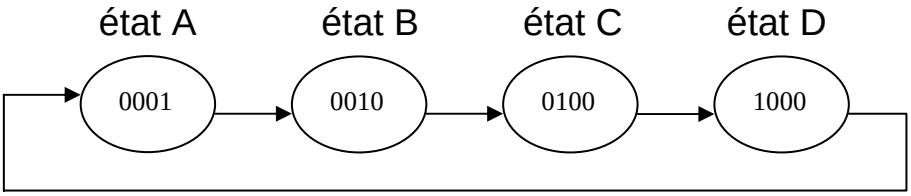
```
        sortie istype 'buffer,reg_d';
equations
        sortie.clk=h;
        sortie.oe=oe;
        sortie.ar=r;
```

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;
```

Le langage ABEL – HDL

(Abel Hardware Description Language)

R. Grisel (IUP GEII Amiens)

Introduction

H.D.L optimisé pour la conception des PLD (CPLD - SPLD)

Introduit par DATA I/O, se compose de parties déclaratives et de parties "exécutables", mais comparé par exemple au C++, le résultat de la compilation d'un "programme" HDL sera une solution "hardware" avec une cible "circuit" et non un programme exécutable.

ABEL permet la spécification d'équations, tables de vérité (truth tables), machines d'états (state diagrams), ou toute combinaison des trois. Il est possible de ne pas scpécifier le circuit ("device") final, voire de ne pas donner d'affectations pour les broches d'entrée/sortie. Le format de sortie des fichiers produits par le compilateur permet l'interfacage avec des produits de CAO électronique (dans notre cas il s'agira des outils Xilinx) ainsi que les standard JEDEC pour les programmateurs de PLD ainsi que les programmateurs de PROM. Dans son expression la plus simple, la description est de type RTL (Register Transfer language) i.e. que l'on définit une sortie par une variation de ses signaux d'entrée. ABEL permet d'écrire des structures indépendanters du choix technologique final, qui sera fait après simulation et optimisation (avec généralement une aide par les outils de CAO).

Structure de base d'un fichier ABEL HDL

Les rectrictions suivantes s'appliquent :

- Une ligne fait max. 150 caractères.
- Une ligne se termine par un caractère de fin ((0AH), (0BH), (0CH), (ODH)). C'est automatique avec un éditeur de texte.
- Mots clés, identificateurs, et nombres doivent être séparés par au moins un espace. Les exceptions concernent les listes ou les identicateurs et nombres seront séparés par dses virgules.
- Les mots clés peuvent être mis en majuscules, minuscules, mixtes.
- Les identificateurs (noms donnés par l'utilisateurs) peuvent être mis en majuscules, minuscules, mixtes mais sont "case-sensitive" (mais certain outils, notamment les outils Xilinx, ne le sont pas, il faut donc éviter de différencier les identificateurs de cette manière).

Identificateurs

Caractères ASCII valides:

```
a - z (alphabet minuscule)

A - Z (alphabet majuscule)

0 - 9 (chiffres)

<espace>

<tabulation>
```

```
! @ # $ ? + & * ( )- _ = +

[ { } ] ; : ` " ` ~ \ | , < > . / ^ %
```

Les règles sont les suivantes :

- Max. 31 caractères de long.
- Doivent commencer par une lettre ou par un "underscore" (_).
- Le caractère 'tilde' (~) est aussi valide dans les noms de signaux.
- A part le premier caractère, les identificateurs peuvent contenir minuscules, majuscules, chiffres et "underscore".
- Pas d'espace, faire les séparations avec l'"underscore" ou les majuscules.
- Le point n'est pas autorisé sauf pour les extensions valides du langage.

Mots clés réservés

```
DECLARATIONS            IF                  STATE_DIAGRAM
DEVICE                  IN (obs)            TEST_VECTORS
ELSE                    ISTYPE              THEN
ENABLE (obs)            LIBRARY             TITLE
END                     MACRO               TRACE
ENDCASE                 MODULE              TRUTH_TABLE
ENDWITH                 NODE                WHEN
FUSES                   OPTIONS             WITH
EQUATIONS               PIN
FLAG (obs)              PROPERTY
```

Constantes

Système de numération : ABEL supporte le binaire (2), l'octal (8), le decimal (10), l'hexadecimal. Le système par défaut est le décimal (10) et il peut être changé par la déclaration @RADIX. Examples :

```
56      decimal 56
```

^h56 hexadecimal 56 (valeur décimale 86)

^b1001 binaire 1001 (valeur décimale 9)

^o56 octal 56 (valeur décimale 46)

^h0E hexadecimal 0E (valeur décimale 14)

Valeurs logiques

Vrai (valeur logique 1) et faux (valeur logique 0) sont représentés comme des nombres entiers 32 bits, vrai = -1 (tous les bits à 1) et faux =0 (tous les bits à 0).

Constantes Spéciales

```
.C.     Entrée "clockée" (transition bas-haut-bas)
```

.K. Entrée "clockée"input (transition haut-bas-haut)

.U. Front montant (transition bas-haut)

.D. Front descendant

.F. Entrée ou sortie "flottante"

.P. Pré-charge de registre

.SVn. n = 2 .. 9. Force l'entrée à un "supervoltage" de 2 à 9V.

.X. Etat indifférent

.Z. Etat "tristate"

Chaînes de caractères

Suites de caractères ASCII , avec espaces, encadrés par apostrophes (` `). Les chaînes de caractères sont utilisées dans TITLE, MODULE et OPTIONS, et dans les instructions, broche, noeuds interne (node) et déclaration d'attributs.

```
`Hello this is a string!'
```

`Punctuation? is even allowed !!'

```
`It\'s easy to use ABEL'
`He\\she can use backslashes in a string'
```

Operateurs

Constantes, noms de signaux sont utilisés dans les expressions. Les expressions combinent, comparent ou effectuent des opérations sur leurs opérandes pour produire un résultat. Les opérations sont indiquées par des opérateurs.

Il y a quatre types de base d'opérateurs pour ABEL-HDL : logique, arithmétique, relationnel, et affectation.

Opérateurs logiques (Logical Operators) :

!	!A	NOT: Complément à 1
&	A & B	AND
#	A # B	OR
$	A $ B	XOR: exclusive OR
!$	A !& B	XNOR exclusive NOR

Opérateurs Arithmétiques (Arithmetic Operators) :

```
-      -A      Complément à 2  (négation)
        -     A - B    Soustraction
```

```
+      A + B    Addition


*      A*B     Multiplication
/      A/B     Division entière non signée
%      A%B     Reste de la division
<<     A<<B    Décalage de A gauche de B bits
>>     A>>B    Décalage de A droite de B bits
```

Opérateurs relationnels (Relational Operators) :

```
==  A == B     Egalité
!=  A != B     Différent
<   A <  B     Inférieur
<=  A <= B     Inférieur ou égal
>   A >  B     Supérieur
>=  A >= B     Supérieur ou égal
```

Opérateurs d'affectation (Assignment Operators) :

Une équation affecte la valeur d'une expression à un signal de sortie. Il y a 4 opérateurs d'affectation, 2 pour le combinatoire et 2 pour les sorties à registre. Les affectations combinatoires arrivent sans délai après l'évaluation de l'expression, les affectations de type "registre" sont efectuées au prochain coup d'horloge actif associé au registre.

```
=   ON(1)  Affectation combinatoire ou equation détaillée
?=   DC(X)  Affectation combinatoire ou equation détaillée
:=   ON(1)  Affectation de sortie de registre
?:=  DC(X)  Affectationd de registre
```

Priorité des opérateurs :

possibilité d'utilisation des parenthèses pour changer l'ordre d'évaluation, pour la même priorité l'évaluation se fait de gauche à droite dans l'ordre d'écriture..

1. - (Négation), ! NOT
2. & (AND), <<, >> (shift left, shift right), * (multiply), / (unsigned division),% (modulus)
3. + (add) - (subtract), # (OR), $ (XOR), !$ (XNOR)
4. ==, != (equal, not equal), < (less than), <= (less than or equal) > (greater than), >= (greater than or equal)

Les Ensembles (Sets)

Un ensemble est une collection de signaux et de constantes, et on peut les référencer avec un seul identificateur. Dans d'autres langages, les "sets" sont appelés des "bus". Les éléments d'un ensemble sont écrits entre crochets ([]) et séparés par des virgules, il speuvent aussi être séparés par l'indicateur d'ordre '..'.

bus_address =[A8, A7, A6, A5, A4, A3, A2, A1, A0]

ou

bus_address =[A8..A0]

Multiplication, division, reste, décalage ne sont pas autorisées sur les ensembles.

!bus_address= [!A8, !A7, ..., !A0]

-bus_adress=!bus_address + 1

bus_address.OE=[A8.OE,.....,A0.OE]

bus_address == BK si égalité de tous les bits

bus_address != BK si au moins un des bits est différent de l'autre

Blocs

Ce sont des sections de texte encadrés par des parenthèses. Ils sont utilisés dans les équations, machines d'états, macros et directives, le texte concerné peut tenir sur une ligne ou sur plusieurs et l'on peut imbriquer un bloc dans un autre bloc.

```
{ this is a block }
```

{this is also a block, and it

spans more than one line.}

{ A = B # C;

D = [0, 1] + [l, 0];

}

Commentaires

Un commentaire commence par " et se termine par ".

```
 "You can use comments to write observation in your program."
```

Paramètres et paramètres formels

Les paramètres formels n'ont de valeur que dans les macros. Dans les modules ou directives, ils pointent le paramètre sur lequel est appliqué l'opération. Dans le corps d'une macro, avant le paramètre formel, on écrit un point d'interrogation, et ils sont séparés par des espaces.

```
EXP MACRO (a, b, c, d)
```

{?a & ?b # ?c & ?d};

`This was the macro declaration'

`The call of the defined macro:'

F = EXP (x, y, z, w);

`The evaluation of the macro will be:'

F= x & y # z & w;

Directives

Les directives donnent des options qui contrôlent le contenu ou le traitement d'un fichier source.

@ALTERNATE (Alternate operator Set)

@ALTERNATE valide les opérateurs altenatifs :

Standard	Alternate	Description
!	/	NOT
&	*	AND
#	+	OR
$	:+:	XOR
!$	:*:	XNOR

@STANDARD (Standard operator Set):

@STANDARD repositionne les opérateurs standard ABEL.

@CONST (Constant Declaration) hors de la section de déclaration normale

Syntaxe:

```
@CONST identifier = expression;
```

Exemple:

```
@CONST OK = 1;
```

@DCSET (Don't Care Set)

@DCSET permet l'optimisation des fonctions partiellement définies (supprime les attributs `dc', `neg' et `pos').

@ONSET neutralise @DCSET.

@IF

@IF inclut ou exclut des sections de code source par rapport à la valeur d'une expression.

Syntaxe:

```
@IF expression {block}
```

Exemple:

```
@IF (A>B) {C = D # E}
```

@INCLUDE

@INCLUDE inclut un fichier ABEL

Syntaxe : @INCLUDE `file_name.abl`

Exemple:

```
@INCLUDE `macros.abl`
```

@INCLUDE `C:\\ABEL\\INCLUDE\\CONVERT.ABL`

@PAGE

@PAGE Saut de page

@RADIX (Default Base Numbering Directive)

@RADIX change la base par défaut.

Syntaxe:

```
@ RADIX expression
```

Example:

```
A = 10;         `A is 10 (decimally)'
```

@RADIX 2;

A = 10; ` A vaut 2 (decimal)

@RADIX 10000; `base passe à 16"

A = 10; ` A vaut 16 (decimal)'

@RADIX 0A; `restaure à 10'

A = 10; `A vaut 10 (decimal) de nouveau

@REPEAT

@REPEAT répète un bloc du nombre de fois donné dans l'expression (peut être un entier).

Syntaxe :

```
@REPEAT expression { block };
```

ABEL a aussi une multitude d'autres directives listées ci-dessous.

```
@EXPR   (Expression Directive)
```

@IFB (If Blank Directive)

@IFDEF (If Defined Directive)

@IFIDEN (If Identical Directive)

@IFNB (If Not Blank Directive)

@IFNDEF (If Not Defined Directive)

@IFNIDEN (If Not Identical Directive)

Description des éléments d'un fichier ABEL-HDL

Structure du fichier source ABEL HDL

La figure suivante donne un exemple de fichier ABEL.

essai.abl - HDL Editor

File Edit Search View Synthesis Project Tools Help

```
1  module essai
2  Title 'essai'
3
4  Declarations
5
6  circuit DEVICE 'P16R4';
7  in1 PIN;
8  in2 PIN;
9  in3 PIN;
10 clk PIN;
11 tout PIN istype 'reg';
12 rien PIN istype 'reg';
13 autre PIN istype 'reg';
14 sortie=[tout,rien,autre];
15
16 "  <<add your declarations here>>
17
18 Equations
19 sortie.clk = clk;
20 tout := in1 & in2 & in3;
21 rien := !in1 & !in2 & !in3 ;
22 autre := ( in1 & in2 & !in3) # ( in1 & !in2 & in3) # ( in1 & !in2 & !in3)
23         # ( !in1 & in2 & in3) # ( !in1 & in2 & !in3) # ( !in1 & !in2 & in3) ;
24
25
26
27 end essai
28
```

La description fonctionnelle est définie dans le "module".

Un module comprend les sections suivantes :

Un en-tête (Header)

Une partie Declarations

Une partie Description logique (Equations)

Des vecteurs de test éventuels (Test_Vectors)

Une indication de fin pour le module (End)

Les règles suivantes s'appliquent à la structure du module :

- Il doit y avoir au moins un en-tête avec l'instruction Module (Title et Options sont optionnels)
- Toutes les autres sections peuvent être répétées dans n'importe quel ordre. Les déclarations doivent suivrent l'en-tête. A
- Aucun symbole (identificateur) ne peut être utilisé sans avoir été déclaré.

En-tête (MODULE)

- **MODULE** ***nom_du_module***
- **OPTIONS** (éventuellement)
- **TITLE** ***`Definition du fonctionnement'*** (Chaîne de caractères qui sera écrite en en-tête du fichier JEDEC)

Déclarations

Généralement cela consiste en :

- **DEVICE** PLD_device_declaration (identificateur du PLD utilisé, est **OPTIONNEL**)
- **DECLARATIONS** (Mot-Clé)
- **PIN** (Déclarations des entrées/sorties)
- **NODE** Déclaration des signaux internes
- **ISTYPE** Attribut optionnel pour les déclarations de PIN/NODE
- **Constant Declarations**
- **MACRO** déclarations des macros
- **LIBRARY** Références

Description de la logique du circuit (EQUATIONS)

La description est fonctionnelle et/ou structurellle. Tous les éléments utilisés doivent avoir été précédemment déclarés.

Les différents éléments suivants permmettent de faire la description :

- Equations
- Truth Tables (Tables de vérité)
- State Diagrams (Diagramme, machine d'états)
- Fuses (Fusibles)
- XOR Factors

Test Vectors (Vecteurs de test)

Ne sont quasiment plus utilisés (On préfère la simulation)

End Statement

Termine le module et va de pair avec l'instruction MODULE.

Header (MODULE)

The MODULE statment defines the beginning of an HDL program and must be paired with END statment that defines the functional description end.

The Header Section can consist of the following elements:

MODULE *name* (The identifier is the beginning of module statement (required))

OPTIONS (Optional element that can influence the run of the program)

TITLE *`string'* (Optional element that it is written in the header of JEDEC file)

The order of the identifiers must be the order presented

Remarques

Chaque module doit avoir au moins une section de déclarations. Le "device" est déclaré de la manière suivante :

```
device_identifier DEVICE real_device;
```

Cette déclaration est optionnelle, elle associe un module effectivement avec une cible matérielle (défini par real_device, du type P22V10, P16L8, etc...), Le "device_identifier" doit correspondre à un nom de fichier valide car ce sera celui du fichier JEDEC auquel on éjoutera l'extension .JED. Le ";" est obligatoire.

- **XABEL :** (Environnement Xilinx ABEL) Ne pas spécifier de "device"' Xilinx™ , Le "device sera spécifié danbs le "Project Manager".
- **XABEL EPLD Design:** L'instruction suivante (ABEL-HDL) doit être spécifiée dans l'en-tête du source ABEL pour le niveau d'une conception hiérarchique ou pour un fichier simple pour signaler que le fichier représente une conception complète, la syntaxe en est la suivante :

```
nom_de_module DEVICE;
```

Exemples:

```
D1 device `P22V10' ;
```

module UART; "Xilinx™ XEPLD design"

UART device;

Déclarations des Signaux

Il y a plusieurs types de déclaration :

```
ATTRIBUTE
CONSTANT
LIBRARY
MACRO
NODE
PIN
```

PIN et NODE permettent de déclarer les signaux du module, et optionnellement d'associer les numéros de broche du circuit. Cette étape peut être faite en dernier lors du "mapping" final sur une architecture. Les attributs sont affectés au signaux (PIN/NODE) par l'intermédiaire de la déclaration ISTYPE. Les extensions (.NOM_EXTENSION) permettent de décrire plus précisément les signaux.

Declarations des "Pin" et "Node"

```
[!] pin_id [,[!] pin_id. . .] PIN (pin# [, pin#]] [ISTYPE `
attributes']
```

[!] node_id [[!] node_id. .] NODE [node# [, node#]] [ISTYPE `attributes']

avec *pin#* et *node#* numéros de broche du circuit, et
attributes une chaîne de caractères qui spécifie des attributs pour les circuits avec possibilité de programmation
(majoritairement pour les sorties).

Attributs pour les signaux :

signal_name, [signal_name] ISTYPE `attr'

Attributs génériques ou "indépendant de l'architecture" (Architecture Independent)

`com' , Signal combinatoire

`reg' , Elément mémoire, (bascule), peut être combiné avec 'invert' ou 'buffer' pour préciser le type de sortie.

`neg' , Le signal (entrée ou sortie) est inversé (logique négative pour optimisation), affectation à 1 pour logique non attribuée.

`pos' , Le signal (entrée ou sortie) n'est pas inversé (logique positive), affectation à 0 pour logique non attribuée.

Attributs "dépendants de l'architecture" (Architecture dependent)

`buffer' , Pas d'inverseur entre la fonction de sortie (éventuellement Flip-Flop) et la broche de sortie.

`invert' , Un inverseur entre la fonction de sortie (éventuellement Flip-Flop) et la broche de sortie.

`reg_D' , Une bascule D (D-type flip-flop). Implique une description détaillée (voir plus loin), si l'on spécifie 'invert' ou 'buffer', le compilateur transforme l'équation := (et .FB) en expressions avec .D et .Q.

`reg_T', Une bascule bi-stable (T-type flip-flop). Implique une description détaillée (voir plus loin).

`reg_SR' , Bascule SR (SR-type flip-flop). Implique une description détaillée (voir plus loin).

`reg_JK' , Bascule JK (JK-type flip-flop). Implique une description détaillée (voir plus loin).

`reg_G' , élément mémoire(D-type flip-flop avec validation de clock). Implique une description détaillée (voir plus loin).

`xor' , L'architecture cible contient une porte XOR.

ISTYPE permet de définir des attributs lorsque l'on ne spécifie pas de "device" ni de numéro de broche, cependant il est préférable de les spécifier même quand les numéros de broche pour un "device" sont donnés de manière à permettre une lecture claire du fichier.

Déclarations des constantes

```
Syntaxe :
```

identifier [, identifier]. .= expression [, expression]. . ;

Une constante correspond à un identificateur qui garde la même valeur dans le module. On fait l'affectation de gauche à droite.

Extensions (suffixe) .EXT :

```
Les extensions permettent de donner des préisions par rapport au comportement du
circuit.
```

Syntaxe:

signal_name.EXT

Les extensions peuvent être indépendantes de l'architecture ou spécifiques à une architecture. On définit l'utilisation des extensions spécifiques comme correspondant à une syntaxe **détaillée (detailed)** et les extensions indépendantes comme correspondant à une syntaxe **broche-à-broche (pin to pin)**.

Extensions indépendantes de l'Architecture (Architecture independent):

.CLK Entrée d'Horloge pour bascule

.OE "Output enable"

.PIN "Pin feedback", Retour vers le réseau de la broche (PIN)

.FB "Register feedback", Retour vers le réseau de la sortie du registre

Architecture specific dot extensions:

.D Dans la partie droite d'une équation, .D est le "feedback" combinatoire de l'entrée D de la bascule. Dans la partie gauche d'une équation, c'est l'entrée D de la bascule.

.J Entrée J d'une bascule JK.

.K Entrée K d'une bascule JK.

.R Entrée Rd'une bascule RS.

.S Entrée S d'une bascule RS.

.T Entrée T d'une bascule T.

.Q Sortie de la bascule

.PR Preset du Registre (synchrone ou asynchrone)

.RE Reset du registre (synchrone ou asynchrone)

.ACLR Reset Asynchrone du registre

.ASET Preset Asynchrone du registre

.CLR Reset synchrone du registre

.SET Preset synchrone du registre

.AR Reset Asynchrone du registre

.AP Preset Asynchrone du registre

.SR Reset Synchrone du registre

.SP Preset synchrone du registre

.LE Active bas, validation de latch

.LH Active haut, validation de Latch

.CE Entrée de validation d'horloge

.OE Validation de la sortie (Output Enable)

.COM "Feedback" combinatoire de l'entrée de la bascule, utilisé pour faire la distinction éventuelle entre la broche (.PIN) et le réseau logique interne (.COM).

Description en logique Combinatoire

La logique combinatoire peut être décrite avec des équations booléennes,des tables de vérité.
Les instructions suivantes peuvent être utilisées :

[WHEN condition THEN] equation;

[ELSE equation];

Exemple:

EQUATIONS

@ALTERNATE

A = B + C + /D;

ADDR = AB + 5;

WHEN X THEN A =B; ELSE A = C;

Syntaxe des tables de vérité :

TRUTH_TABLE ([input_signals] -> [output_signals])

[input_values] -> [output_values];

[input_values] -> [output_values];

...

Exemple: Décodeur BCD -> 7 segments

```
"HEX-to-seven-segment decoder
MODULE bcd7seg

Declarations
D3,D2,D1,D0,ENA pin;
A,B,C,D,E,F,G pin istype 'com';
BCD=[D3,D2,D1,D0];
LED=[A,B,C,D,E,F,G];
ON,OFF=0,1;
L,H,X,Z=0,1,.X.,.Z.;


EQUATIONS

LED.OE= !ENA;
truth_table ( BCD -> [G,F,E,D,C,B,A])
0 -> [OFF,ON,ON,ON,ON,ON,ON] ;
1 -> [OFF,OFF,OFF,OFF,ON,ON,OFF];
2 -> [ON,OFF,ON,ON,OFF,ON,ON];
3 -> [ON,OFF,OFF,ON,ON,ON,ON];
4 -> [ON,ON,OFF,OFF,ON,ON,OFF];
5 -> [ON,ON,OFF,ON,ON,OFF,ON];
6 -> [ON,ON,ON,ON,ON,OFF,ON];
7 -> [OFF,OFF,OFF,OFF,ON,ON,ON];
8 -> [ON,ON,ON,ON,ON,ON,ON];
9 -> [ON,ON,OFF,ON,ON,ON,ON];

end;
```

Description de la Logique Séquentielle

Cette description peut se faire par équations logiques booléennes, diagrammes d'états et tables.

Equations Logiques Booléennes

La syntaxe est la même que pour la logique combinatoire, mais on utilise l'opérateur ":=" à la place de l'opérateur "=". L'expression est affectée après le coup d'horloge actif.

EQUATIONS

Q := (A + B) * /RST;

COUNT := COUNT + 1;

Tables de vérité

La syntaxe est la même, mais on utilise l'opérateur ":>" dans la table.

Syntaxe:

TRUTH_TABLE ([IN_SIGNALS] :> [REG_SIGNALS] -> [OUTPUTS]

[IN_VALUES] :> [REG_VALUES] -> [OUT_VALUES;]

[IN_VALUES] :> [REG_VALUES] -> [OUT_VALUES;]

[IN_VALUES] :> [REG_VALUES] -> [OUT_VALUES;]

[...];

Considérations de Conception

Comme nous l'avons dit précédemment, il y a deux types de description pour les éléments à mémoire : "pin to pin" ou "detailed".

Considérons l'équation suivante :

sortie := !sortie # Preset;

Le terme sortie à gauche avec ":=" implique un registre, le terme !sortie à droite correspond à quoi?
- la sortie du registre ?
- la sortie du combinatoire alimentant le registre ?
- la sortie de la broche de sortie (I/O) ?

Il n'y a d'autre part pas d'indication du type de registre. L'écriture suivante :

sortie.CLK = clock;
sortie := !sortie.FB # Preset;

est plus précise. C'est une description dite "PIN TO PIN".

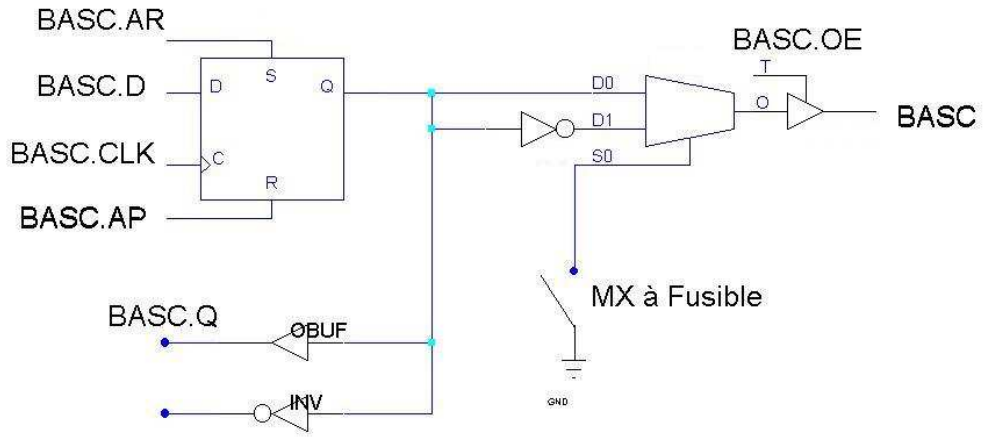
L'écriture en version "detailed" revient à écrire :

sortie.CLK = clock;
sortie.D = !sortie.Q # Preset; mais il n'y a pas d'information sur la sortie (I/O). On peut donc avoir un focntionnement différent selon que l'on ait un circuit avec sortie inversé (P16R4 par exemple) ou un circuit avec sortie normale. Il est donc préférable de mettre un attribut supplémentaire, l'attribut

"buffer" (pas d'inversion entre la sortie de la bascule et la sortie associée) ou son complément "invert" est **nécessaire** , soit une déclaration :

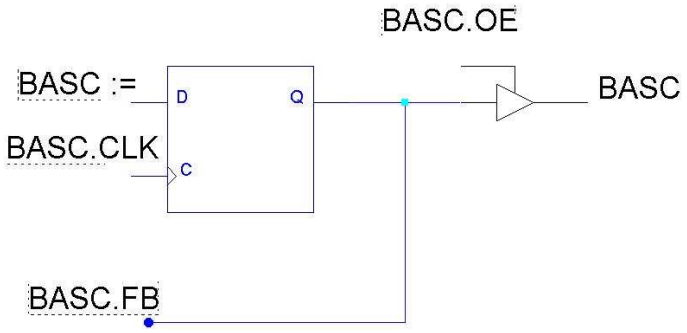
sortie PIN ISTYPE 'buffer';

Il est intéressant de "visualiser" la "macrocellule" équivalente pour le mode "detrailed" :



Il faut noter sur la figure ci-dessus que l'identificateur BASC.Q n'est pas forcément le même que BASC (notamment si le fusible commande l'inversion en sortie).

Le "PIN to PIN" estv plus comportemental et correspond à la macro-cellule suivante (pas de Reset ou de Preset dans ce cas) :



Récapitulons donc l'écriture de notre module :

En "Pin to Pin" (Module 1):

module pin_to_pin
sortie PIN ISTYPE 'reg';
Clock, Preset PIN;
Equations;
Sortie.clk = Clock;
sortie := !sortie.FB # Preset;

En "detailed" (Module 2) :

```
module detail
sortie PIN ISTYPE 'reg_D, buffer';
Clock,Preset PIN
Equations;
Sortie.clk = Clock;
Sortie.D = !Sortie.Q # Preset
```

Le module 1 peut normalement s'implanter dans n'importe quel type de PLD. Le module D ne peut avoir pour cible que les circuits avec bascule D et sortie non-inverseuse. Pour l'implémenter dans un circuit avec sortie inversé, il faut écrire :

(Module 3)

```
module detail_inv
sortie PIN ISTYPE 'reg_D, invert';
Clock,Preset PIN
Equations;
Sortie.clk = Clock;
!Sortie.D = Sortie.Q # Preset; "(Changement de l'équation à cause du type de sortie)"
```

Ceci démontre que le mieux est de faire une description "Pin to Pin" pour ne pas avoir à modifier la decsription en fonction du type de cible, par contre on n'a pas de Preset et Reset (asynchrone), et si l'on en met un, on aura une restriction à certains types de circuits. Par exemple :

(Module 4)

```
module avec_preset
sortie PIN ISTYPE 'reg, buffer';
Clock,Preset PIN
Equations;
Sortie.clk = Clock;
Sortie.AP = Preset;
sortie := !Sortie.FB;
```

.AP pour le Preset est associé à la bascule interne, donc il ne faut pas que le circuit soit à sortie inverrsée entre la bascule et la broche, d'où la nécessité d'indisuer le "buffer" pour être sur que cela fonctionnera correctement.

Si l'on met un attribut "invert", il faudra que le Preset soit connecté sur le Resset Asynchrone pour avoir effectivement un Preset (voir Module 5) :

(Module 5)

```
module avec_preset_inv
sortie PIN ISTYPE 'reg, invert';
Clock,Preset PIN
Equations;
Sortie.clk = Clock;
Sortie.AR = Preset;
sortie := !Sortie.FB;
```

Terminons par l'exemple d'un compteur synchrone par 5 (Q2 Q1 Q0) = (000 001 010 011 100), pour lequel les équations sont :

D0 = /QO AND /Q2; D1 = (QO AND /Q1) OR (/Q0 AND Q1) = Q0 XOR Q1; D2 = Q0 AND Q1;

(Module 7 : Compteur synchrone en description PIN TO PIN)

```
module csync_pin_to_pin
csync device 'P16R4'
Clock, reset PIN 1,2 ;
Q0, Q1, Q2 PIN 15,16,17 ISTYPE 'reg';
QREG = [Q2,Q1,Q0];
Equations;
QREG.CLK = Clock;
Q0 := (!Q0.FB & !Q2.FB) & !reset;
Q1 := (Q0.FB $ Q1.FB) & !reset;
Q2 := (Q0.FB & Q1.FB) & !reset;
END;
```

(Module 8 : Compteur synchrone en description DETAILED)

```
module csync_detailed
syncd device 'P16R4'
Clock, reset PIN 1,2 ;
Q0, Q1, Q2 PIN 15,16,17 ISTYPE 'reg_D,invert';
QREG = [Q2,Q1,Q0];
Equations;
QREG.CLK = Clock;
!Q0.D = (Q0.Q & Q2.Q) & !reset;
!Q1.D = (!Q0.Q $ !Q1.Q) & !reset;
!Q2.D = (!Q0.Q & !Q1.Q) & !reset;
END;
```