

ÉLÉMENTS  
D'ANALYSES ET DE SYNTHÈSES  
EN  
ÉLECTRONIQUE NUMÉRIQUE

Patrick COHEN

# Sommaire

<b>1</b>	<b>FONCTIONS COMBINATOIRES .....</b>	<b>5</b>
1.1	EXEMPLE .....	5
1.2	EQUATIONS.....	6
1.3	LA SYNTHÈSE « CLASSIQUE » .....	6
1.4	DESCRIPTION À L' AIDE DU LANGAGE VHDL .....	6
<b>2</b>	<b>LES FONCTIONS SÉQUENTIELLES.....</b>	<b>7</b>
2.1	LES MACHINES SYNCHRONES À NOMBRE FINIS D'ÉTATS .....	7
2.1.1	<i>Horloge, registre d'état et transitions.....</i>	7
2.1.1.1	Le registre d'état .....	7
2.1.1.2	L'horloge .....	7
2.1.2	<i>Les différentes architectures .....</i>	7
<b>3</b>	<b>OUTILS DE DESCRIPTION .....</b>	<b>8</b>
3.1	LE DIAGRAMME DE TRANSITION .....	8
3.2	DU DIAGRAMME AUX ÉQUATIONS.....	8
3.3	DESCRIPTION VHDL .....	9
<b>4</b>	<b>EXEMPLES .....</b>	<b>9</b>
4.1	SYNTHÈSE D'UN SÉQUENCEUR.....	9
4.1.1	<i>Analyse du cahier des charges.....</i>	9
4.1.2	<i>Tracé des chronogrammes.....</i>	10
4.1.3	<i>Représentation du diagramme de transition.....</i>	10
4.1.4	<i>Description en VHDL.....</i>	11
4.2	ANALYSE D'UN FICHER DE DESCRIPTION VHDL.....	11
<b>5</b>	<b>SYNTAXE DU LANGAGE VHDL .....</b>	<b>12</b>
<b>6</b>	<b>STRUCTURE GÉNÉRALE D'UN FICHER DE DESCRIPTION.....</b>	<b>12</b>
6.1	LES OBJETS ET LEURS TYPES. LES ATTRIBUTS.....	13
6.1.1	<i>Les objets .....</i>	13
6.1.2	<i>Les signaux .....</i>	13
6.1.3	<i>Les variables.....</i>	13
6.1.4	<i>Les constantes .....</i>	13
6.1.5	<i>Les types.....</i>	13
6.1.5.1	Entier .....	13
6.1.5.2	Bits.....	13
6.1.5.3	Les tableaux .....	13
6.1.6	<i>Les attributs .....</i>	13
6.2	LES INSTRUCTIONS CONCURRENTES .....	14
6.2.1.1	Affectations concurrentes de signaux .....	14
6.2.1.1.1	Affectation simple.....	14
6.2.1.1.2	Affectation conditionnelle.....	14
6.2.1.1.3	Affectation sélective.....	14
6.2.1.2	Instanciation de composants .....	14
6.2.1.3	Process .....	14
6.3	LES INSTRUCTIONS SÉQUENTIELLES.....	15
6.3.1.1	Affectation séquentielle .....	15
6.3.1.2	Affectation de variable.....	15
6.3.1.2.1	Le test if ... then ... elsif ... else ... end if ;.....	15
6.3.1.2.2	Le test case ... when ... end case ;.....	15
6.3.1.2.3	La boucle for.....	15
6.3.1.2.4	La boucle while.....	16
<b>7</b>	<b>INTÉGRATION DANS UN CPLD. ....</b>	<b>16</b>

7.1	LES COMPOSANTS ALTERA .....	16
7.1.1	<i>La famille MAX 7000S</i> .....	16
7.1.2	<i>Le composant EPM 7128S</i> .....	17
7.2	PROGRAMMATION MATÉRIELLE.....	18
7.3	ALGORITHME DE PROGRAMMATION .....	18
7.4	LE LOGICIEL MAXPLUS.....	19



On peut considérer qu'il existe en électronique numérique, deux types de fonctions : les fonctions combinatoires et les fonctions séquentielles.

## 1 FONCTIONS COMBINATOIRES

Une fonction combinatoire peut être représentée par un système dont les sorties dépendent uniquement des entrées. Le système ci-contre est parfaitement déterminé par un ensemble de  $p$  équations logiques  $S_0 = FC_0(E_0, \dots, E_n)$  à  $S_p = FC_p(E_0, \dots, E_n)$ . Ces équations font référence aux opérateurs logiques de base (ET, OU, NON).

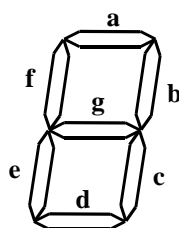


L'outil permettant la description exhaustive d'un tel système est la table de vérité. Des méthodes mathématiques ont été développées afin de réduire ces équations, entre autres les tableaux de Karnaugh. Dès que la complexité du système devient importante (au-delà de quatre variables) il faut faire appel à des outils informatiques.

### 1.1 EXEMPLE

On veut réaliser un dispositif effectuant la conversion BCD vers 7-segments.

$b_3$	$b_2$	$b_1$	$b_0$	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	0	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	0	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1



## 1.2 EQUATIONS

$$a = \overline{b_3} \cdot \overline{b_2} \cdot \overline{b_1} \cdot \overline{b_0} + \overline{b_3} \cdot \overline{b_2} \cdot b_1 \cdot \overline{b_0} + \overline{b_3} \cdot \overline{b_2} \cdot b_1 \cdot b_0 + \overline{b_3} \cdot b_2 \cdot \overline{b_1} \cdot \overline{b_0} + \text{etc...}$$

...

$$g = \text{etc....}$$

## 1.3 LA SYNTHÈSE « CLASSIQUE »

Les équations précédentes sont élaborées à l'aide de circuits électroniques standard, après avoir simplifié ces équations afin de réduire la complexité de la réalisation.

## 1.4 DESCRIPTION À L'AIDE DU LANGAGE VHDL

```

library ieee ;
use      ieee.std_logic_1164.ALL;
entity BCD_2_7SEG is
  port (
    BIN           : in  STD_LOGIC_VECTOR(3 downto 0) ;
    SORTIE        : out STD_LOGIC_VECTOR(6 downto 0)
  );
end BCD_2_7SEG;
architecture COMPORTEMENT of BCD_2_7SEG is
begin
  process(BIN)
    -- Décodeur Binaire 7 segments. Les sorties sont actives au niveau bas.
  begin
    --
    "abcdefg"
  case BIN is
    when 0 => SORTIE <= "0000001" ;
    when 1 => SORTIE <= "1001111" ;
    when 2 => SORTIE <= "0010010" ;
    when 3 => SORTIE <= "0000110" ;
    when 4 => SORTIE <= "1001100" ;
    when 5 => SORTIE <= "0100100" ;
    when 6 => SORTIE <= "0100000" ;
    when 7 => SORTIE <= "0001111" ;
    when 8 => SORTIE <= "0000000" ;
    when 9 => SORTIE <= "0000100" ;
    when others => SORTIE <= "01100000" ; -- E pour erreur
  end case ;
end process ;
end COMPORTEMENT;
-----

```

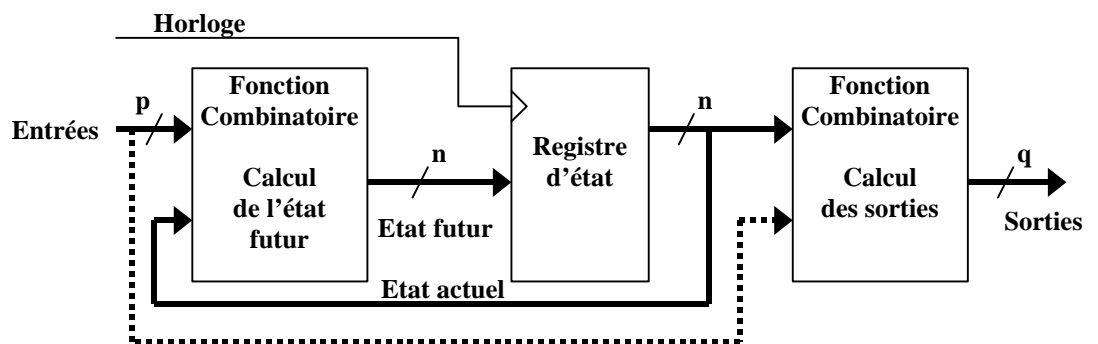
Dans ce cas le logiciel de synthèse se chargera de la réduction des équations en fonction de paramètres liés au type de circuit utilisé.

## 2 LES FONCTIONS SÉQUENTIELLES

La différence essentielle entre une fonction combinatoire et une fonction séquentielle réside dans la capacité de cette dernière de «se souvenir » des événements antérieurs : une même combinaison des entrées, à un certain instant, pourra avoir des effets différents suivant les valeurs des combinaisons précédentes de ces mêmes entrées.

Pour traduire cet effet de mémoire on introduit la notion d'**état interne** de la fonction, l'action des entrées est alors de provoquer d'éventuels changements d'**état**, la situation qui suit le changement de l'une des **entrées** dépend de l'état **précédent**. Si le nouvel état est différent du précédent on dit qu'il y a eu une **transition**.

### 2.1 LES MACHINES SYNCHRONES À NOMBRE FINIS D'ÉTATS



Une machine à états (M.A.E.) en anglais Finite State Machine (F.S.M.) est un système dynamique, qui peut se trouver, à chaque instant, dans une position parmi un nombre fini de positions possibles. Elle parcourt des cycles, en changeant éventuellement d'état lors des transitions actives de l'horloge. L'architecture générale d'une machine à état est présentée ci-dessous.

#### 2.1.1 HORLOGE, REGISTRE D'ÉTAT ET TRANSITIONS

Le registre d'état, piloté par son horloge, constitue le cœur d'une machine à états. Les autres blocs fonctionnels sont à son service.

##### 2.1.1.1 LE REGISTRE D'ÉTAT

Il est constitué de  $n$  bascules synchrones. Son contenu représente l'**état actuel** de la machine. Il s'agit d'un nombre codé en binaire sur  $n$  bits. L'entrée du registre d'état constitue l'**état futur**, celui qui sera chargé lors de la prochaine transition active de l'horloge. Le registre d'état est la **mémoire** de la machine.

La taille du registre d'état fixe le nombre d'états accessibles. Si  $n$  est le nombre de bascules, le nombre d'états  $N = 2^n$ .

##### 2.1.1.2 L'HORLOGE

Le rôle de l'horloge est de fixer les instants où les transitions entre états sont prises en compte. Entre deux fronts consécutifs de l'horloge, la machine est figée en position mémoire.

#### 2.1.2 LES DIFFÉRENTES ARCHITECTURES

La figure 2 présente l'architecture générale d'une machine à états. Suivant la façon dont les sorties dépendent des états et des commandes, on distingue deux types de

machines à états : les machines de **Moore** et les machines de **Mealy**. Dans les premières les **sorties** ne dépendent que de l'**état actuel** (la liaison en trait interrompue est absente), pour les secondes les **sorties** dépendent de l'**état actuel et des entrées** (la liaison en trait interrompu est présente).

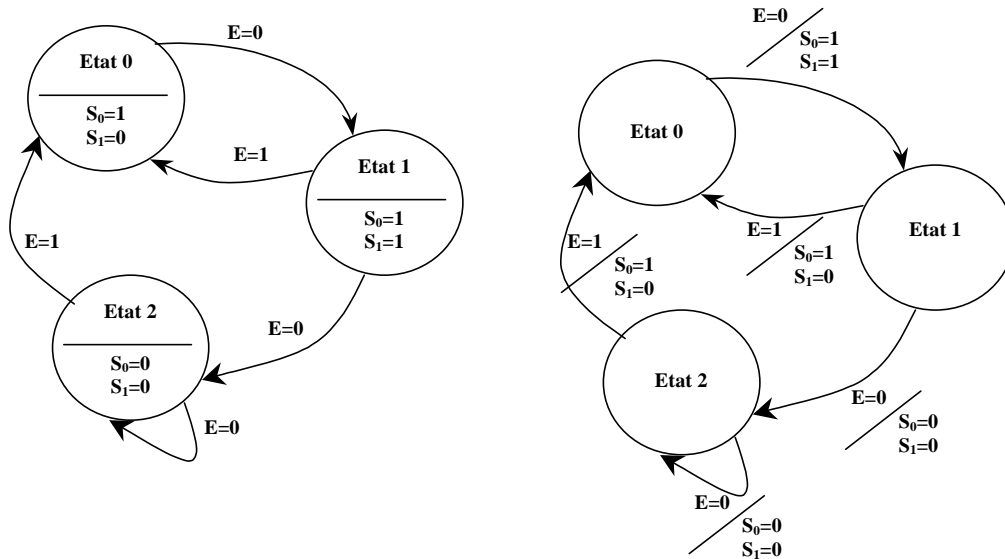
### 3 OUTILS DE DESCRIPTION

Si l'outil d'analyse et de synthèse des fonctions combinatoires est la table de vérité, le diagramme de transition constitue l'outil privilégié pour l'analyse et la synthèse des fonctions séquentielles.

#### 3.1 LE DIAGRAMME DE TRANSITION

- . On associe à chaque valeur possible du registre d'état, une case.
- . L'évolution du système est représentée par des flèches représentant les transitions.
- . Pour qu'une transition soit activée il faut que les trois conditions suivantes soient vérifiées :
  1. Le système se trouve dans l'état «source » considéré
  2. La condition de réalisation sur les entrées est vraie
  3. Un front actif de l'horloge survient

Pour les machines de **Moore** les sorties évoluent **après l'activation de la transition**. Les valeurs des sorties seront représentées dans les cases du diagramme.



Pour les machines de **Mealy** les sorties évoluent **après l'évolution des entrées**. Les valeurs des sorties seront représentées sur les flèches du diagramme.

#### 3.2 DU DIAGRAMME AUX ÉQUATIONS

Le passage du diagramme de transition aux équations est indispensable si on veut synthétiser la machine à états avec des circuits standards. L'outil permettant le passage exhaustif du diagramme aux équations est **la table de transitions et d'états**.

C'est une table de vérité constituée :

**en entrée** de l'état **actuel** du registre d'état

des **entrées** de la machine à états  
**en sortie** de l'état **futur** du registre d'état  
des **sorties** de la machine à états

Les équations des sorties du registre d'état sont ensuite adaptées au type de bascules utilisées.

Comme pour les fonctions combinatoires la complexité du problème croit de façon exponentielle avec le nombre d'états et le nombre d'entrées.

### 3.3 DESCRIPTION VHDL

Le langage VHDL offre de multiples possibilités pour traduire le fonctionnement d'une machine à états. Nous ne nous intéresserons qu'à la description *comportementale*. D'une façon générale seules seront envisagées les fonctions séquentielles synchrones.

Le processus qui décrit le fonctionnement d'une machine à états comporte deux structures imbriquées : le traitement des commandes et le traitement de l'état de départ de chaque transitions.

- . Les commandes se prêtent bien à une modélisation par des structures hiérarchiques **if ... elsif ... else ... end if**.
- . Les états se prêtent bien à une modélisation en terme d'aiguillage par les structures **case ... when ... when others ... end case**.

Le registre d'état est matérialisé par deux éléments :

- 1) Un signal interne de type **bit\_vector** ou **integer** déclaré de façon à être codé sur n chiffres binaires.
- 2) Un processus, activé par le *seul signal d'horloge* qui est l'unique endroit où le signal d'état subit une affectation.

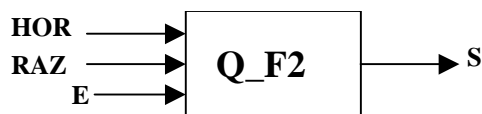
## 4 EXEMPLES

### 4.1 SYNTHÈSE D'UN SÉQUENCEUR

*Plan de l'étude :*

- a) Analyse du cahier des charges
- b) Tracé des chronogrammes.
- c) Représentation du diagramme de transition.
- d) Description en VHDL

#### 4.1.1 ANALYSE DU CAHIER DES CHARGES



On désire réaliser une fonction dont la sortie S recopie l'état logique présent sur son entrée E si celle-ci est restée stable après 2 coups d'horloge successifs.



#### 4.1.4 DESCRIPTION EN VHDL

```

library ieee ;
use ieee.std_logic_1164.ALL;

entity MAE is          --MAE = Machine A Etat
  port ( E,RST,HOR     : in  STD_LOGIC ;
        S              : out STD_LOGIC );
end MAE;
architecture COMPORTEMENT of MAE is
  signal REG_ETAT      : STD_LOGIC_VECTOR(2 downto 0);
  process (HOR,RST)
  begin
    if RST='0' then REG_ETAT <= "000";
      elsif (HOR'event and HOR='1') then
        case REG_ETAT is
          when "000" => S = '0' ;
            if E = '1' then REG_ETAT <= "001";
              else REG_ETAT <= "000";
            end if ;
          when "001" => S = '0' ;
            if E = '1' then REG_ETAT <= "010";
              else REG_ETAT <= "000";
            end if ;
          when "010" => S = '0' ;
            if E = '1' then REG_ETAT <= "011";
              else REG_ETAT <= "000";
            end if ;
          when "011" => S = '1' ;
            if E = '1' then REG_ETAT <= "011";
              else REG_ETAT <= "100";
            end if ;
          when "100" => S = '1' ;
            if E = '1' then REG_ETAT <= "011";
              else REG_ETAT <= "101";
            end if ;
          when "101" => S = '0' ;
            if E = '1' then REG_ETAT <= "011";
              else REG_ETAT <= "000";
            end if ;
          when others => REG_ETAT <= "00";
        end case ;
      end if ;
    end process ;
  end COMPORTEMENT ;

```

#### 4.2 ANALYSE D'UN FICHIER DE DESCRIPTION VHDL

La démarche d'analyse est analogue à la démarche exposée pour la synthèse. Elle consiste à aller du général vers le particulier, de l'**entity** vers l'**architecture**.

- . On identifie la nature des entrées et sorties de la fonction décrite en VHDL.
- . On identifie les blocs fonctionnels et les instructions concurrentes de l'architecture.
- . On analyse chacun des **process** en les identifiant à l'aide d'algorithme, d'organigramme ou de diagramme de transition.

## 5 SYNTAXE DU LANGAGE VHDL

Le langage VHDL (Very High level Description Language) est, comme son nom l'indique, un langage de description matériel. Il ressemble fortement à un langage informatique (PASCAL, C). Il faut toutefois garder à l'esprit que l'objectif de la description est la synthèse de fonctions numérique à l'aide de circuit PLD, FPGA ou ASIC. Pour cela il est important de connaître le circuit « cible » si on veut mener à bien cette synthèse.

La différence essentielle avec un langage informatique est la simultanéité des actions décrites. Pour cela il faudra se familiariser avec la notion d'**instruction concurrente** (simultanée) et d'**instruction séquentielle**. Par défaut toutes les instructions sont concurrentes, pour qu'elles soient séquentielles il faut qu'elles apparaissent dans un **process**.

## 6 STRUCTURE GÉNÉRALE D'UN FICHIER DE DESCRIPTION

```
library NOM_DE_LA_LIBRAIRIE ;
use ELEMENT_DE_LA_LIBRAIRIE ;
```

```
entity NOM_DE_L'ENTITY is
  port ( A, B : in «type des objets » ;
        S : out « type de l'objet »
        );
end NOM_DE_L'ENTITY ;
```

```
architecture NATURE_DE_L'ARCHITECTURE of NOM_DE_L'ENTITY is
  «zone de déclarations des objets internes »
begin
  «ensemble d'instructions concurrentes »
```

```
end NATURE_DE_L'ARCHITECTURE ;
```

Les librairies pré définies sont fournies par le fabricant du logiciel ou du circuit cible. Elles peuvent aussi être constituées par l'utilisateur.



L'**entity** représente le bloc fonctionnel avec ces entrées et sorties. Le nom des entrées et sorties est introduit par le mot clé **port**.

L'**architecture** représente la description interne de la fonction. Cette description n'est pas unique. On dira qu'elle est de nature :

- . *comportementale (behavioral)* lorsqu'elle décrit la fonction sans référence à la structure ou aux équations.
- . *structurelle (structural)* lorsqu'elle décrit la fonction à partir de composants pré définis.

. *flot de données* lorsqu'elle décrit la fonction par des équations booléennes.

Comme dans les langages informatiques nous trouverons les concepts de visibilité de variables, de passages de paramètres, de sous programmes ou de fonctions.

## **6.1 LES OBJETS ET LEURS TYPES. LES ATTRIBUTS**

Il existe trois familles d'objets : les *constantes*, les *signaux* et les *variables*. Chaque objet est repéré par un *identificateur*.

### **6.1.1 LES OBJETS**

### **6.1.2 LES SIGNAUX**

Un signal est synthétisé par une équipotentielle et éventuellement une bascule s'il faut conserver sa valeur. Un signal ne peut être déclaré que dans une architecture.

### **6.1.3 LES VARIABLES**

Une variable peut stocker un résultat intermédiaire. Elle ne peut être déclarée que dans un **process**, une **procedure** ou une **function**.

### **6.1.4 LES CONSTANTES**

Une constante a une valeur fixée une fois pour toutes. Elle peut apparaître dans toute unité VHDL. Elle est synthétisée comme des pull up ou pull down.

### **6.1.5 LES TYPES**

#### **6.1.5.1 ENTIER**

**signal** NOM\_DU\_SIGNAL : **integer** ; (ou **variable**)

On déclare un nombre entier qui pourra être compris entre  $-2^{32}$  et  $+2^{32}-1$ , et qui sera synthétisé par 32 équipotentielles ou 32 bascules suivant son utilisation.

**signal** NOM\_DU\_SIGNAL : **integer range 0 to 127**;

On limite ainsi la taille du **signal** synthétisé.

#### **6.1.5.2 BITS**

Un **bit** peut prendre deux valeurs '0' ou '1'.

**variable** NOM\_DE\_LA\_VARIABLEE : **bit** ; (ou **signal**)

Avec la librairie *ieee.std\_logic\_1164* on peut avoir accès entre autres aux valeurs 'X' pour inconnu et 'Z' pour haute impédance, pour le type *std\_logic*.

#### **6.1.5.3 LES TABLEAUX**

**signal** NOM\_DU\_TABLEAU : **bit\_vector(0 to 4)** ; ( ou 4 **down 0**)

On définit ainsi cinq signaux NOM\_DU\_TABLEAU (0) à NOM\_DU\_TABLEAU (4).

### **6.1.6 LES ATTRIBUTS**

Les attributs sont des propriétés attachées à un objet. Nous ne considérerons que l'attribut '**event**' qui rend la valeur vrai si le signal auquel il est attaché a changé de valeur.

NOM\_DU\_SIGNAL'event

Exemple :

**if** (HOR'event and HOR='1') **then** ... détection d'un front montant de HOR.

## 6.2 LES INSTRUCTIONS CONCURRENTES

Les instructions concurrentes se trouvent à l'intérieur d'une *architecture*. En raison de la simultanéité de leur réalisation, elles peuvent être écrites dans n'importe lequel ordre.

### 6.2.1.1 AFFECTATIONS CONCURRENTES DE SIGNAUX

#### 6.2.1.1.1 Affectation simple

L'affectation traduit une simple interconnexion entre deux équipotentielles. L'opérateur d'affectation est le symbole : **<=**

NOM\_DU\_SIGNAL <= EXPRESSION

#### 6.2.1.1.2 Affectation conditionnelle

L'affectation se fera en fonction du résultat de tests logiques.

```
NOM_DU_SIGNAL <= SOURCE_1 when CONDITION_1 else
SOURCE_2 when CONDITION_2 else
....
SOURCE_n ;
```

#### 6.2.1.1.3 Affectation sélective

Le choix de la valeur à affecter à un signal se fait à partir des valeurs possibles d'une expression.

```
with EXPRESSION select
NOM_DU_SIGNAL <= SOURCE_1 when VALEUR_1
SOURCE_2 when VALEUR_2
....
SOURCE_n when others;
```

### 6.2.1.2 INSTANCIATION DE COMPOSANTS

Cette opération consiste à utiliser un sous-ensemble entity-architecture dans une architecture plus vaste. On décrit ainsi une architecture de nature *structurelle*.

### 6.2.1.3 PROCESS

Un **process** définit un comportement qui doit se dérouler lorsque ce **process** devient actif. Le comportement est décrit par une suite *d'instructions séquentielles* exécutées dans le process.

```
[ETIQUETTE :] process ( liste de « sensibilité »
« déclaration des objets utilisés dans le process »
begin
« suite d'instructions séquentielles »
end process [ETIQUETTE ] ;
```

La *liste de sensibilité* est l'ensemble des signaux qui déclencheront l'exécution du process.

Remarque : les expressions entre crochets sont facultatives.

## **6.3 LES INSTRUCTIONS SÉQUENTIELLES**

Les instructions séquentielles sont internes aux process, elles constituent les outils de base des descriptions comportementales.

### **6.3.1.1 AFFECTATION SÉQUENTIELLE**

Elle utilise le même symbole `<=` que l'affectation concurrente, elle a la même syntaxe. Seul la place qu'elle occupe (à l'intérieur d'un **process**) la différencie de l'affectation concurrente.

### **6.3.1.2 AFFECTATION DE VARIABLE**

Elle utilise l'opérateur `:=`, c'est toujours une instruction séquentielle.

#### **6.3.1.2.1 Le test if ... then ... elsif ... else ... end if ;**

Son interprétation est la même que dans les langages de programmation informatique comme PASCAL ou C ;

```
if CONDITION_1 then
  INSTRUCTIONS_SEQUENTIELLES ;
[elsif CONDITION_2 then]
  INSTRUCTIONS_SEQUENTIELLES ;
[else]
  INSTRUCTIONS_SEQUENTIELLES ;
end if ;
```

#### **6.3.1.2.2 Le test case ... when ... end case ;**

Il permet de sélectionner une des instructions à exécuter en fonction des valeurs prises par une expression. Toutes les valeurs doivent être traitées. Les choix doivent être mutuellement exclusifs.

```
Case EXPRESSION is
  when CHOIX_1 => INSTRUCTIONS_SEQUENTIELLE ;
  when CHOIX_2 => INSTRUCTIONS_SEQUENTIELLE ;
  ....
  when others => INSTRUCTION_SEQUENTIELLE ;
end case ;
```

Les boucles permettent de répéter des séquences d'instructions, sans oublier que ces instructions génèrent des structures matérielles.

#### **6.3.1.2.3 La boucle for**

```
[ETIQUETTE :] for PARAMETRE in DEBUT to FIN (ou FIN downto DEBUT) loop
  INSTRUCTIONS_SEQUENTIELLE ;
end loop [ETIQUETTE] ;
```

Exemple :

```
process (A)
  begin Z <= "0000";
  for I in 0 to 3 loop
    if (A = I) then Z(I) <= '1'; end if;
  end loop;
end process;
```

### 6.3.1.2.4 La boucle while

```
[ETIQUETTE :] while CONDITION loop
    INSTRUCTIONS_SEQUENTIELLE ;
end loop [ETIQUETTE];
```

## 7 INTÉGRATION DANS UN CPLD.

Nous allons nous intéresser à la structure des composants matériellement « programmables » du constructeur ALTERA, nous envisagerons ensuite la « programmation » d'un de ces composant avec le logiciel MAXPLUS fourni par ALTERA. La démarche proposée pourra être transférée à d'autres types de composants, provenant d'autres constructeurs.

### 7.1 LES COMPOSANTS ALTERA

Les composants programmables proposés par ALTERA sont de type EEPROM et SRAM. Nous allons travailler sur un composant appartenant à la famille MAX 7000S qui est de type EEPROM. La configuration matérielle pourra donc être modifiée à volonté par l'utilisateur directement sur le circuit (I.S.P. : In Situ Programmable) sans avoir à utiliser de programmeur.

#### 7.1.1 LA FAMILLE MAX 7000S

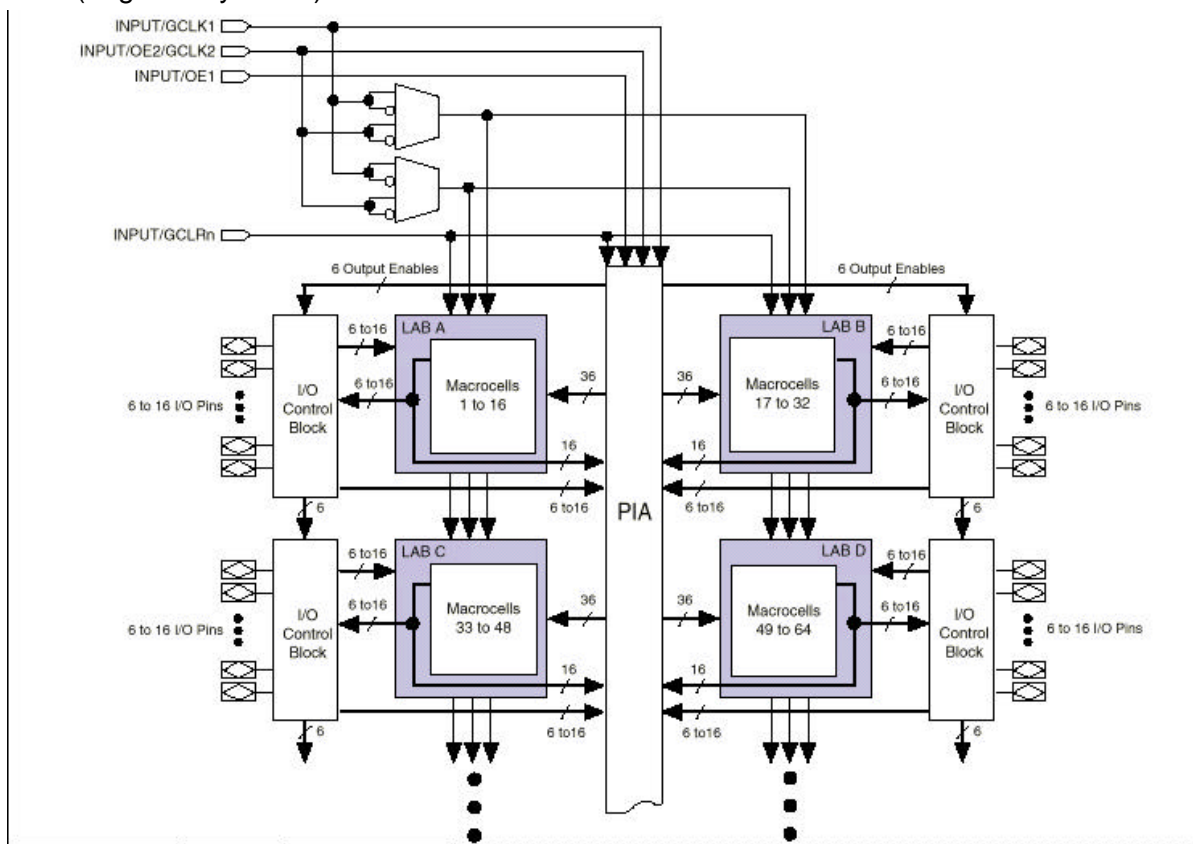
Cette famille comprend six types de composants différant par le nombre de **macrocellules**. Chacun de ces types de composants est fabriqué avec des temps de propagations différents. Le tableau ci-dessous présente ces composants.

Le composant EPM 7128S comporte 128 macrocellules qui sont réalisées par 2500 portes élémentaires (NAND ou NOR). Nous utiliserons ce composant pour intégrer notre carillon.

Feature	EPM7032S	EPM7064S	EPM7128S	EPM7160S	EPM7192S	EPM7256S
Usable gates	600	1,250	2,500	3,200	3,750	5,000
Macrocells	32	64	128	160	192	256
Logic array blocks	2	4	8	10	12	16
Maximum user I/O pins	36	68	100	104	124	164
t <sub>PD</sub> (ns)	5	5	6	6	7.5	7.5
t <sub>SU</sub> (ns)	2.9	2.9	3.4	3.4	4.1	3.9
t <sub>FSU</sub> (ns)	2.5	2.5	2.5	2.5	3	3
t <sub>CO1</sub> (ns)	3.2	3.2	4	3.9	4.7	4.7
f <sub>CNT</sub> (MHz)	175.4	175.4	147.1	149.3	125.0	128.2

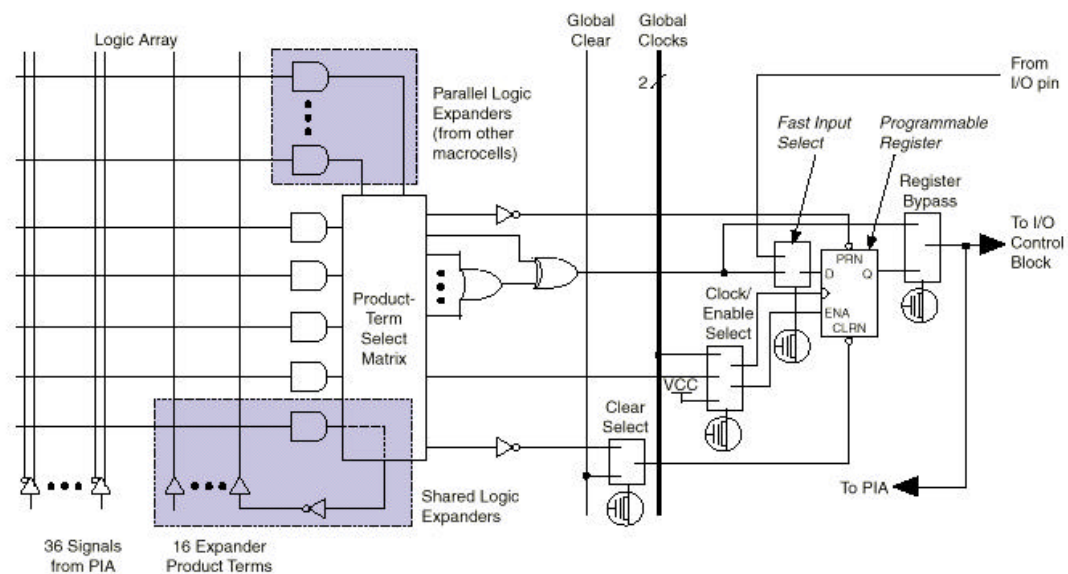
## 7.1.2 LE COMPOSANT EPM 7128S

La **macrocellule** est l'élément de base de ce type de composant, c'est un ensemble de portes associées à une bascule. Ces macrocellules sont réunies par 16 dans un L.A.B. (Logic Array Block).



Le composant EPM 7128S comporte donc 8 L.A.B. interconnectés par un **P.I.A.** (Programmable Interconnect Array). Ces blocks sont « entourés » par de **I/O Control Block** assurant le contrôle des entrées et sorties (entrée ou sortie, sorties trois états, temps de monté, etc ...).

Une macrocellule est constituée d'un réseau de porte de type somme de produit (voir les circuits P.A.L.) associé à des sélecteurs et à une bascule.



La connaissance de cette organisation est indispensable pour permettre le choix du nombre de cellules (le prix croît avec le nombre de cellule).

## 7.2 PROGRAMMATION MATÉRIELLE

La programmation consiste, à l'aide d'un logiciel et du matériel de transfert, à positionner les sélecteurs et à organiser les connexions du P.I.A. conformément au fichier de description écrit en VHDL.

Le logiciel MAXPLUS fourni par ALTERA permet de effectuer l'ensemble des opérations de programmation. Ce logiciel comporte de plus un simulateur permettant de valider le fonctionnement du circuit.

## 7.3 ALGORITHME DE PROGRAMMATION

- 1) Ecriture du fichier VHDL
- 2) Compilation (analyse syntaxique)
  - . Si il y a une erreur elle est indiquée et parfois expliquée.
  - . Correction et compilation jusqu'à élimination de toutes les erreurs
- 3) Simulation fonctionnelle
  - . Si les signaux obtenus ne correspondent pas à ceux espérés
  - . Revoir l'analyse du problème
- 4) Choix du composant et affectation des broches
- 5) Compilation et « fitter » (le logiciel intègre le fichier compilé dans le composant)
  - . Si le « fittage » échoue (composant « trop petit », affectation des broches trop contraignante)
  - . Changer de composant. Revoir le fichier VHDL (analyse)
- 6) Simulation tenant compte des temps de propagation
  - . Si les signaux obtenus ne correspondent pas à ceux espérés
  - . Changer de composant. Revoir le fichier VHDL (analyse)
- 7) Programmation du composant (transfère du fichier de configuration par un câble entre le PC et la carte)

## 8) Test structurel

**7.4 LE LOGICIEL MAXPLUS**

The screenshot displays the MAX+plus II software interface. The top window, titled "car.vhd - Text Editor", contains the following VHDL code:

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;

entity CAR is
port ( HOR : in std_logic;
      BP : in std_logic;
      RAZ : in std_logic;
      UMIC : out std_logic;
      UCV : out std_logic;
      ADR : out std_logic_vector(14 downto 0);
      LED : out std_logic );
end CAR;

architecture COMPORTEMENT of CAR is
signal H,A15:std_logic;
type TYPE_ETAT is (ETAT_0,ETAT_1,ETAT_2);
signal ETAT: TYPE_ETAT;

begin
SEQUENCEUR: process(HOR,RAZ)
begin
if RAZ = '0' then ETAT <= ETAT_0; --init;
elseif (HOR'event and HOR = '1')then
case ETAT is

```

The bottom window, titled "Floorplan Editor", shows a schematic diagram of the circuit. It features several components labeled A0d, E, F, G, and H. A central component is labeled "Selected Node(s) & Pin(s): ADR11 @ 57(I/O)". The diagram includes various input and output pins, such as ADR14, ADR13, ADR12, ADR11, and ADR10, along with other pins labeled (NO), (NO, TMS), and (NO, TCK). The floorplan editor also shows a toolbar with various icons for editing and simulation.