LES

COMPOSANTS

PROGRAMMABLES

ET

VHDL

HISTORIQUE. 1-4
INTRODUCTION 2-1
Principe de base et notation. 2-1
Réalisation d'une fonction logique 2-2
LES PALS. 3-1
Principe 3-1
22v10. 3-1
Architecture du circuit 3-1
Cellule de base 3-2
Schéma interne partiel du 22v10. 3-2
29MA16 3-3
Cellule de base 3-3
LES COMPOSANTS ALTERA 4-1
Les différentes familles 4-1
Flexible Logic Element matriX. 4-1
MAX 7000 4-2
Composants de la famille Max 7000. 4-2
Architecture des circuits 4-3
Cellule de base 4-4
Elément d'entrée-sortie 4-4
Prédiction des délais et analyse temporelle. 4-5
COMPOSANTS DE LA FAMILLE FLEX 8000 4-6
FLEX 8000 4-6
Composants de la famille Flex 8000 4-6
Architecture. 4-7
Cellule de base 4-8
Elément d'entrée-sortie 4-8
FLEX10KE 4-9
Les composants de la famille 10ke 4-9
Cellule de base 4-11
LOGIQUE COMBINATOIRE 5-2
Porte ET 5-2
Fonction booléenne 5-2
Fichier source VHDL 5-3
Multiplexeur 2 Bits vers 1 Bit. 5-5
Multiplexeur de 8 bits vers 1 bit avec signal de validation 5-6
LOGIQUE SEQUENCIELLE. 6-1
Implémentation de registres 6-1
Bascule d. 6-2
Compteur. 6-4
Compteur avec instanciation lpm_counter 6-5
Description de machines d'états 6-6
Machine d'état avec variables d'entrées asynchrones. 6-7
Machine de Moore ou de Mealy 6-8

Dans les années 1970, la grande majorité des systèmes électronique étaient construit à partir des circuits standard de la logique TTL. (Transistor Transistor Logic). Les circuits les plus complexes sont alors:

- 7490 Compteur décimal asynchrone
- 7491 Registre à décalage (Entrée Série - Sortie Série)
- 7492 Diviseur par 12
- 7493 Compteur binaire 4 Bits
- 7494 Registre à décalage (ES-EP/SS)
- 7495 Registre à décalage 4 Bits
- 7496 Registre à décalage 5 bits (ESP/SP)

Avec le développement de la technologie MOS (Metal Oxyde Semiconductor), sont apparus les premiers réseaux de logique programmable appelés PLA (Programmables Logic Aray).

Dans les années 1980, les PLD étaient utilisés pour le remplacement de boîtiers logiques TTL. En 1993, les composants Altera comprennent de 300 à 24 000 portes en technologie CMOS 0,8 micron; Trois ans plus tard, la densité des composants varie de 300 à 100 000 portes. En 1999, la famille APEX comprend des composants dont la densité va de 100 000 à 1 million de portes; la technologie est à 0,25 micron. Le boitier des composants comporte jusqu'à 600 pattes. (Ball Grid Array). Le pas des pattes est de 1 mm. Le diamètre de la bille est de 0,45 mm.

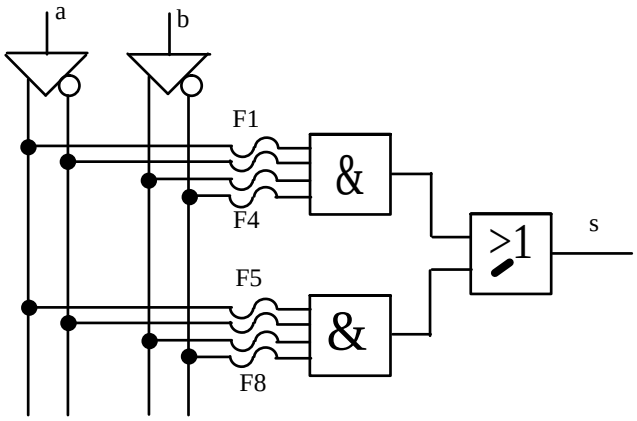
De nos jours, les PLD sont utilisés pour l'intégration de systèmes et sont une alternative au **ASIC** (**A**pplication **S**pecific **I**ntegrated **C**ircuit).

Les composants de logique programmable, plus connu sous le nom de PAL ont été inventés par la société MONOLITHIC MEMORIES INC.

En général, un circuit programmable est un circuit pouvant être programmé par l'utilisateur pour réaliser une fonction logique.

La plupart des PLD (Programmable Logic Device) consiste en un réseau de ET suivi par un réseau de OU; l'un des réseaux ou les deux sont programmables.

Principe de base et notation



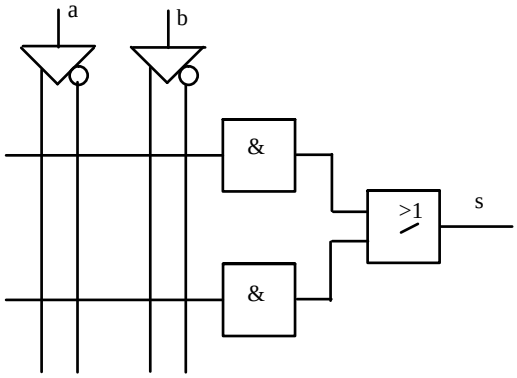
s = (a+/F1)(/a+/F2)(b+/F3)(/b+/F4)

+ (a+/F5)(/a+/F6)(b+/F7)(/b+/F8)

Fusible intact : F = 0
Fusible grillé : F =1

Une PAL non-programmée a tous les fusibles intacts

Pour simplifier la représentation du réseau de ET programmable, on laisse un seul fil connecté à l'entrée de chaque ET. Il faut bien comprendre qu'il y a autant de fils, qu'il y a de variables.



Réseau de ET programmable

Réseau de OU cablé

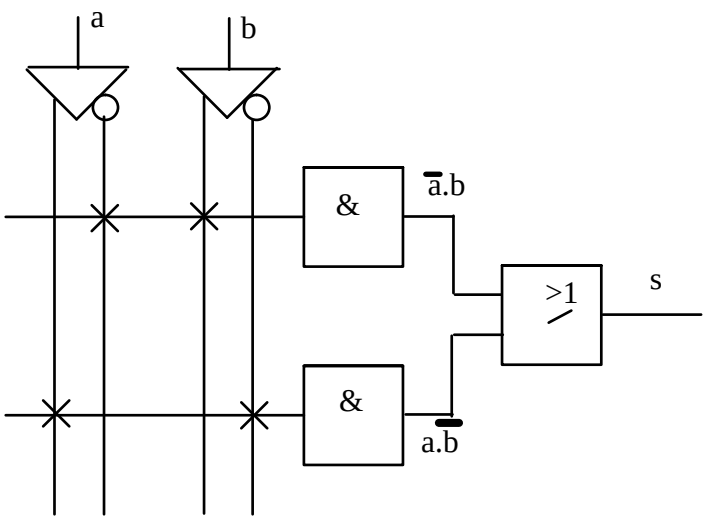
s= /a.b + a./b
/a indique a barre

L'équation devra toujours s'écrire sous la forme de somme de produit.

Les termes produits sont des ET. Exemple : a.b (On parle de somme de "minterms" ou terme-produit).

Ensuite on fait un OU logique des termes-produits.

s est fonction de deux termes-produits



Réseau de logique programmable. (Programmable Array Logic)

Principe

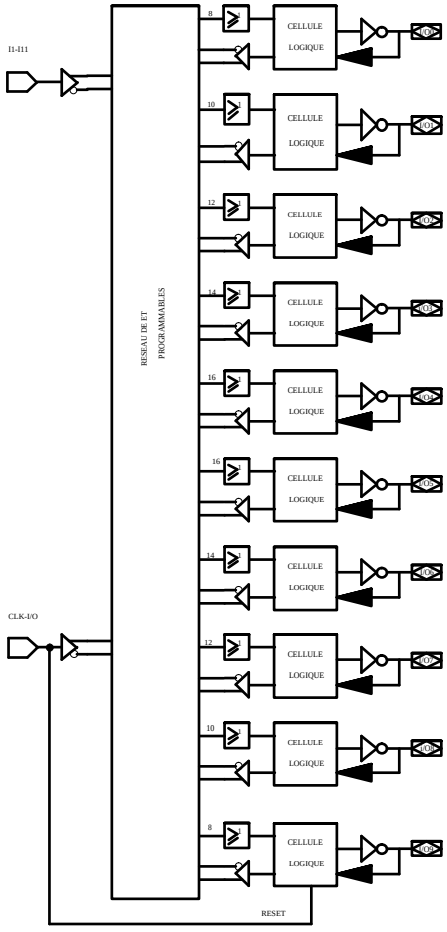
Toutes les pattes d'entrés du circuit sont reliées à une matrice de ET Programmable. C'est le principe de la matrice à diode. A chaque intersection de lignes et de colonnes, on peut établir une connection qui va créer un ET logique.

22v10

C'était l' un des circuits les plus utilisé, à l'instar du 7400 dans la série TTL.

- 11 pattes pour les entrées. (I1 à I11)
- 1 entrée d'horloge globale (CLK/I0) pouvant servir aussi d'entrée
- 10 pattes d'entrée-sortie. (I/O0 à I/O9)
- Chaque patte d'entrée-sortie est connecté à un buffer trois états.

Architecture du circuit



Cellule de base

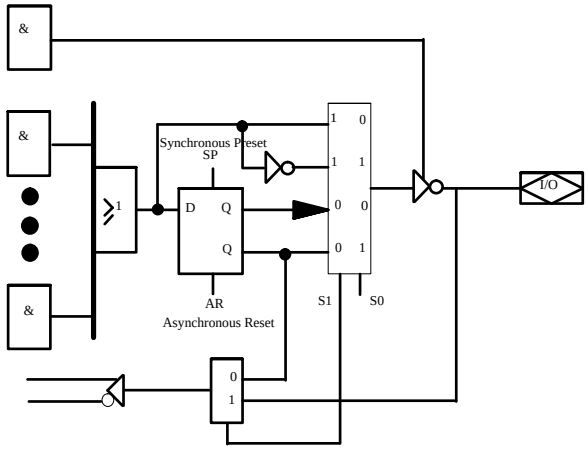
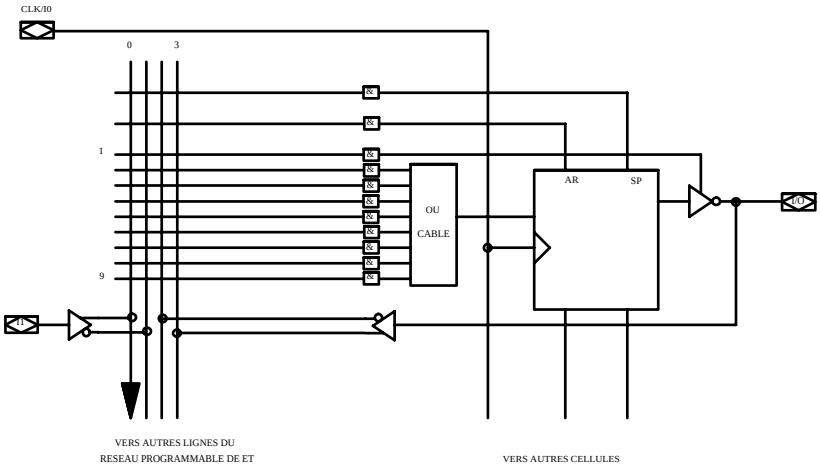


Schéma interne partiel du 22v10



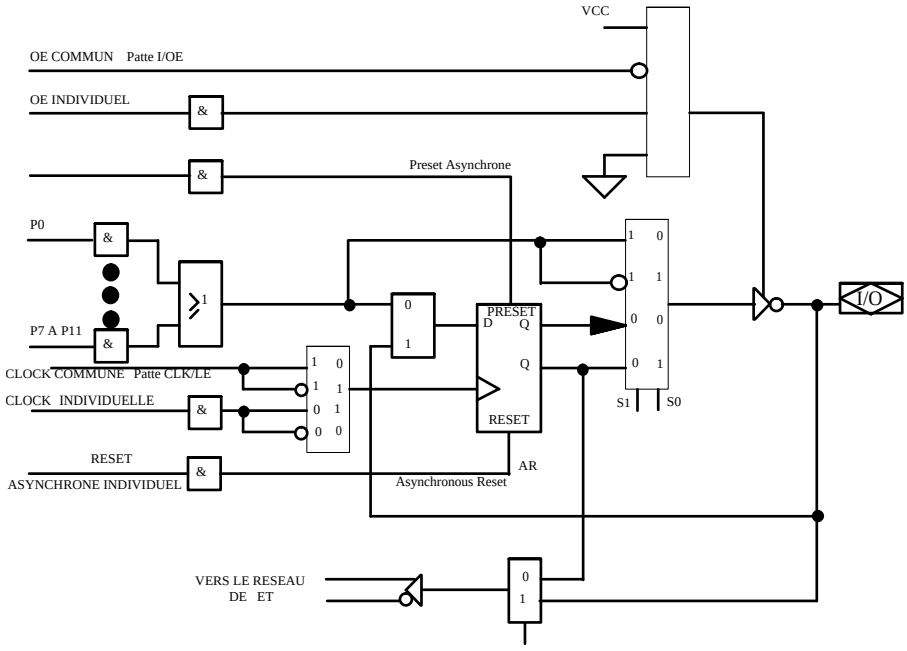
La PAL 29MA16 est optimisée pour les applications asynchrones.
- 4 pattes d'entrées dédicacées
- 16 pattes d'entrée-sortie
- 1 patte d'entrée d'horloge
- 1 patte d'entrée à double usage: patte d'entrée ou patte de contrôle global des amplicateurs de sortie (état trois-états ou non)
- 16 Cellules

La matrice de ET comprend 56 colonnes et 178 lignes.
4 x 2 colonnes provenant des entrées dédicacées
16 x 2 colonnes provenant des macrocellules avec 2 retours
8 x 2 colonnes provenant des macrocellules a 1 retour
1 x 2 colonnes provenant de l'entrée I/OE
Les 178 lignes sont composés de 112 lignes de termes-produits disponibles pour la logique, 66 lignes sont utilisées pour les signaux de contrôle.

Cellule de base

Les signaux **PRESET**, **RESET** et **CLK/LE** sont programmables individuellement pour chaque **MC**.

La sortie de chaque **MC** passe par un buffeur trois-états avant d'être accessible sur la patte du composant. Chaque buffeur peut être contrôlé individuellement ou globalement ou être enable ou disable en permanence.



LES COMPOSANTS ALTERA

Les composants Altera se classent dans la catégorie des CPLD (Complexe PLD). Contrairement au FPGA qui utilisent des lignes de routage segmentées par l'étape de routage, les CPLD utilisent des lignes de connections continues. L'étape de routage interconnecte les structures internes avec ces lignes.

Composants de technologie CMOS.

Les différentes familles

- Classic TP EPROM
- MAX 5000 TP EPROM
- MAX 7000 T-P EEPROM
- MAX 9000 T-P EEPROM
- FLEX 6000 LUT SRAM
- FLEX 8000 LUT SRAM
- FLEX 10K LUT SRAM

Multiple **A**rray matri**X**

Flexible **L**ogic **E**lement matri**X**

EPLD Eraseable Programmable Logic Devices

Composant de logique reprogrammable

C'est grace à la technologie CMOS EPROM que ces composants ont vu le jour
[Altera Application Handbook July 1988]

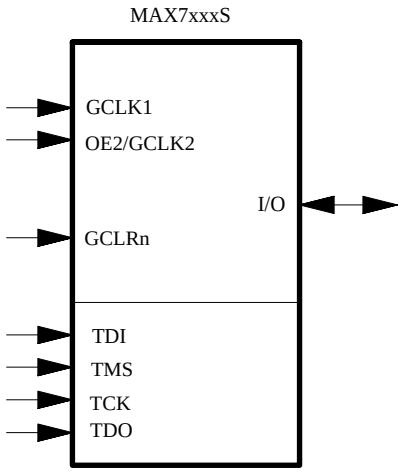
MAX 7000

Programmable sur cartes avec la série S, les composants de la famille MAX7000 comprennent de 32 à 256 Cellules de Base. Le temps de transfert entre le signal d'entrée et le signal de sortie sans registre varie de 5 ns à 7,5 ns suivant les circuits.

Les circuits sont reprogrammables une centaine de fois

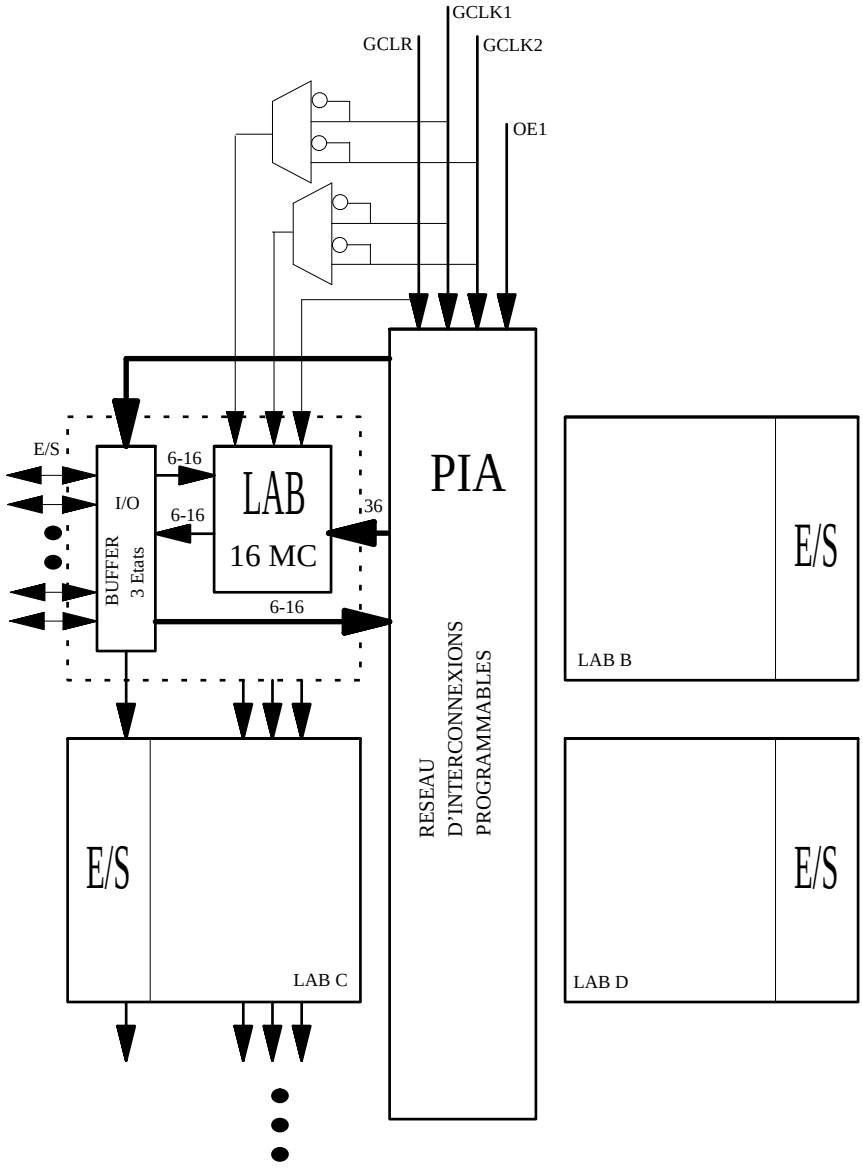
Composants de la famille Max 7000

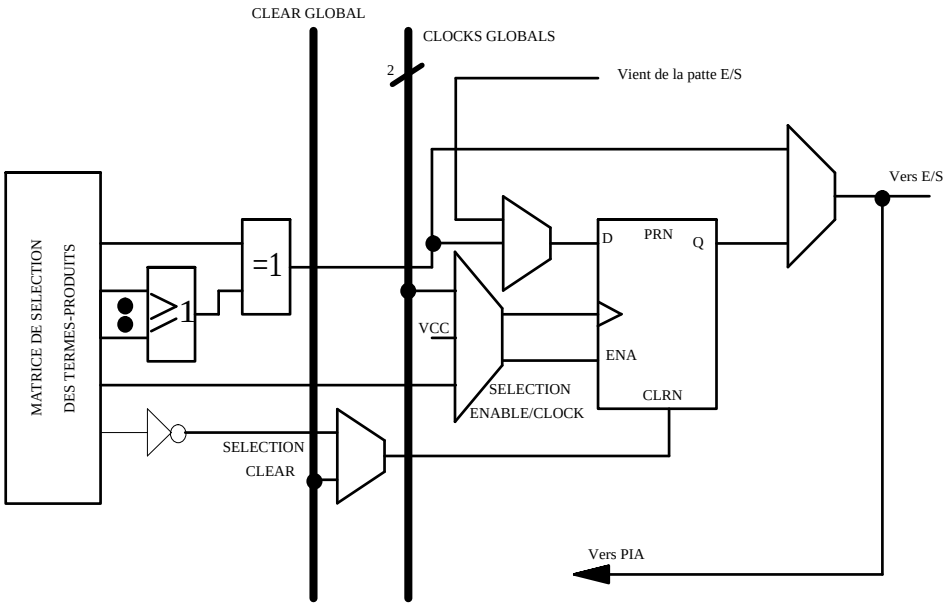
	7032	7064	7096	7128	7160	7192	7256
MC	32	64	96	128	160	192	256
BLOCS	2	4	6	8	10	12	16
E/S	36	68	76	100	104	124	164
tPD	6	5	7.5	6	6	7.5	7.5



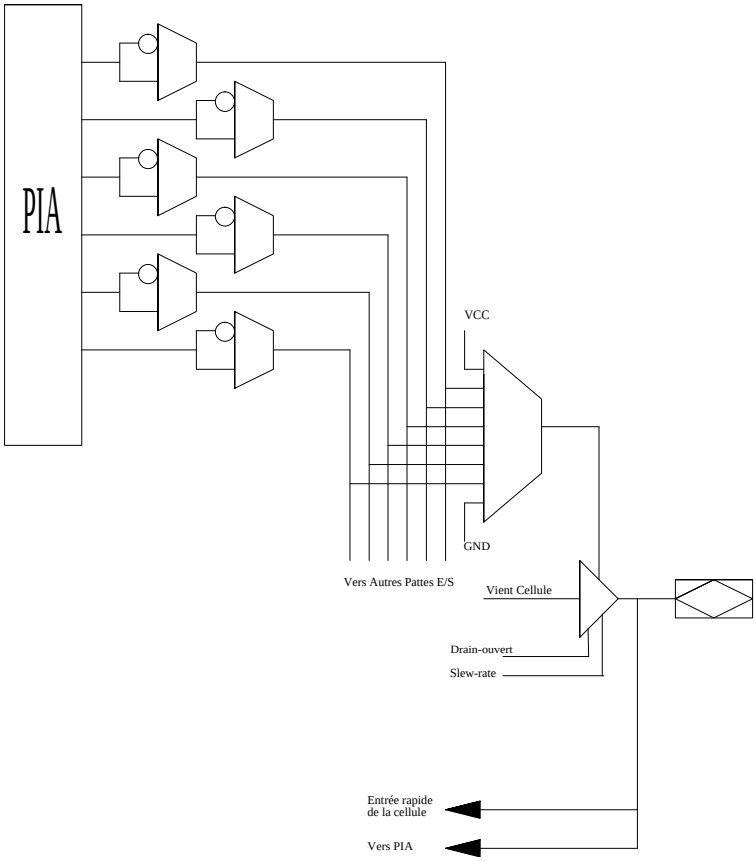
5 Termes-produits par macrocellule (MC)
16 Termes-produits supplémentaires disponibles

1 terme-produit partageable disponible par MC

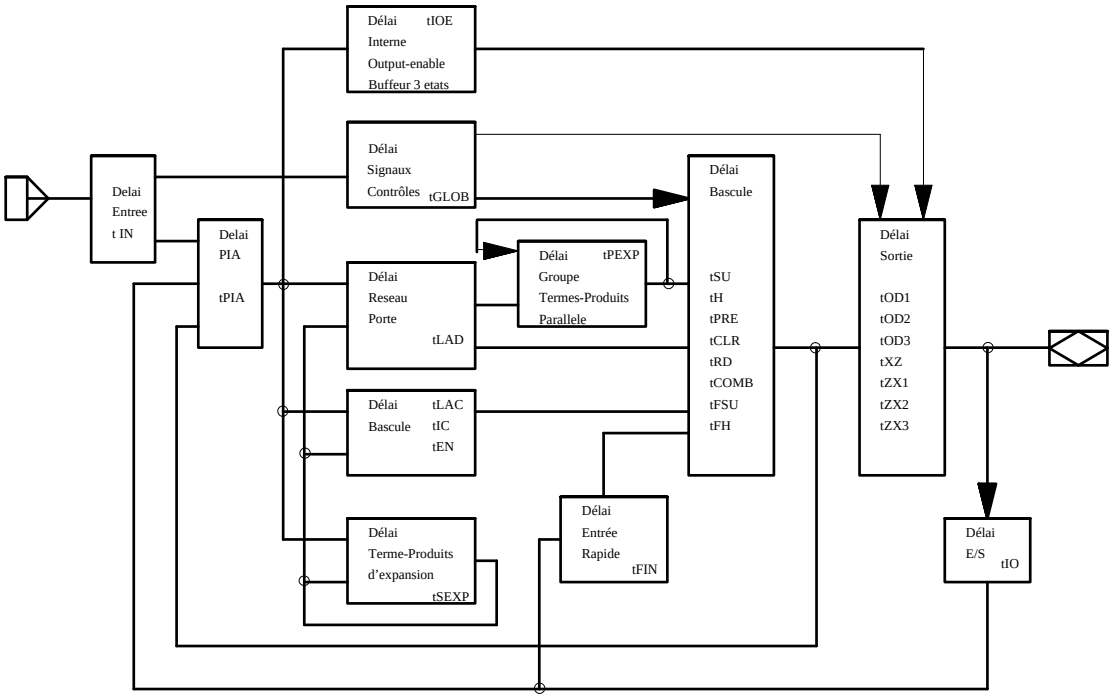

GCLK1
GCLR
GCLK2
OE1
E/S
I/O
BUFFER
3 Etats
6-16
6-16
LAB
16 MC
36
6-16
PIA
RESEAU
D'INTERCONNEXIONS
PROGRAMMABLES
E/S
LAB B
E/S
LAB C
E/S
LAB D



Elément d'entrée-sortie



L’analyse des temps après placement-routage est réalisée soit avec le simulateur maxplus2, soit avec un simulateur du commerce, soit avec le modéle de temps suivant:



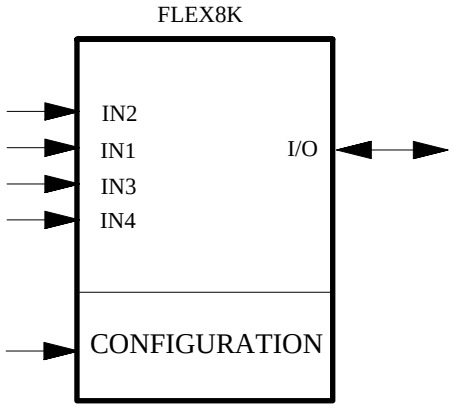
Tous les délais peuvent être calculés avec ce modèle.

FLEX 8000

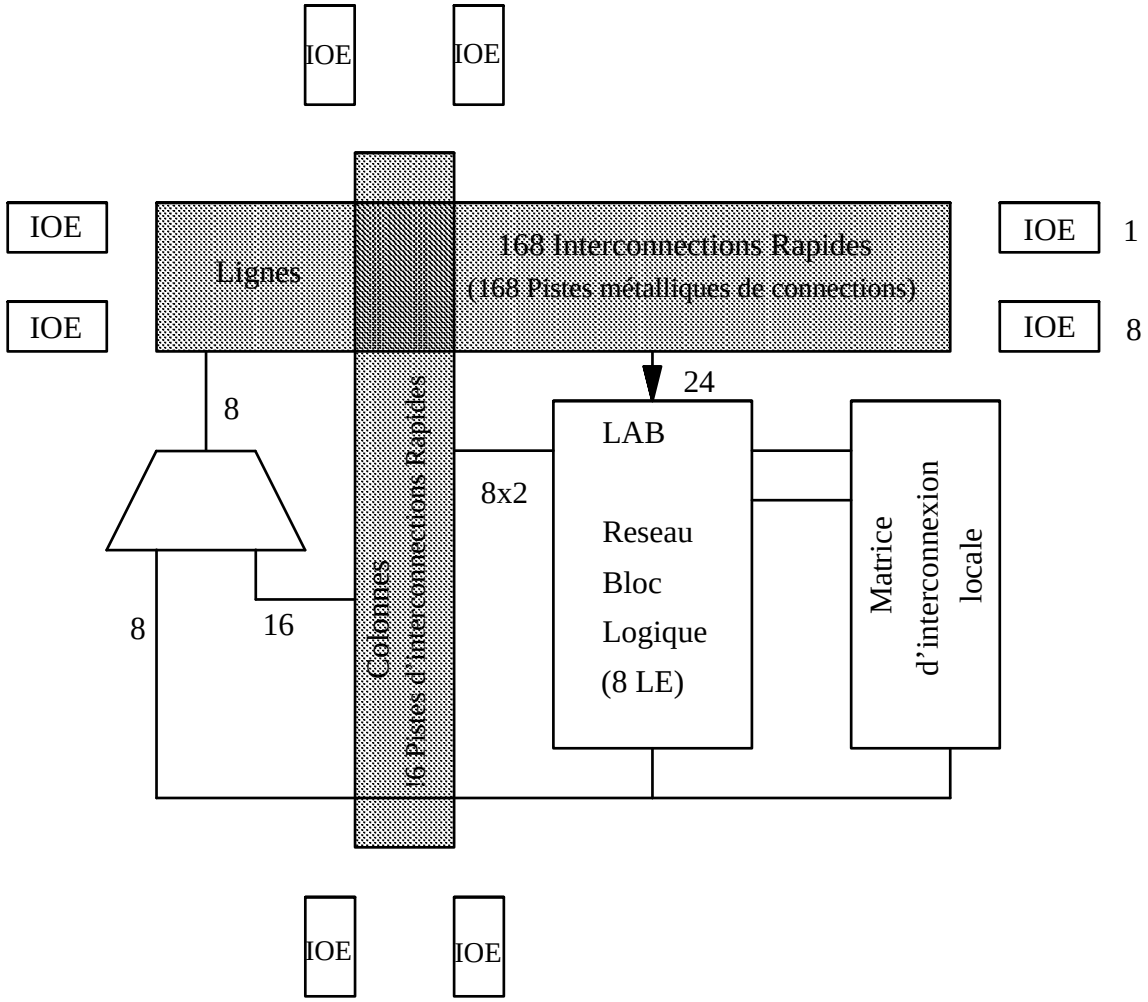
Les composants de la famille Flex8k comprennent de 208 Cellules de base à 1296. Ces composants sont configurables, c'est à dire qu'il est nécessaire de les configurer après la mise sous tension. On configure ces composants avec une EPROM série. On peut le configurer avec un controleur.

Composants de la famille Flex 8000

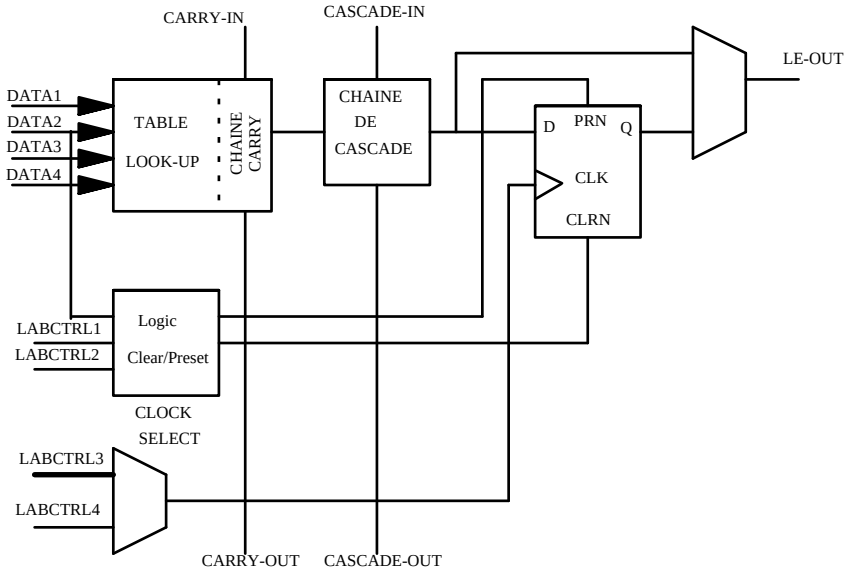
	8282	8452	8636	8820	81188	81500
BASCULES	282	452	636	820	1188	1500
LABS	26	42	63	84	126	162
E/S	78	120	136	152	184	208



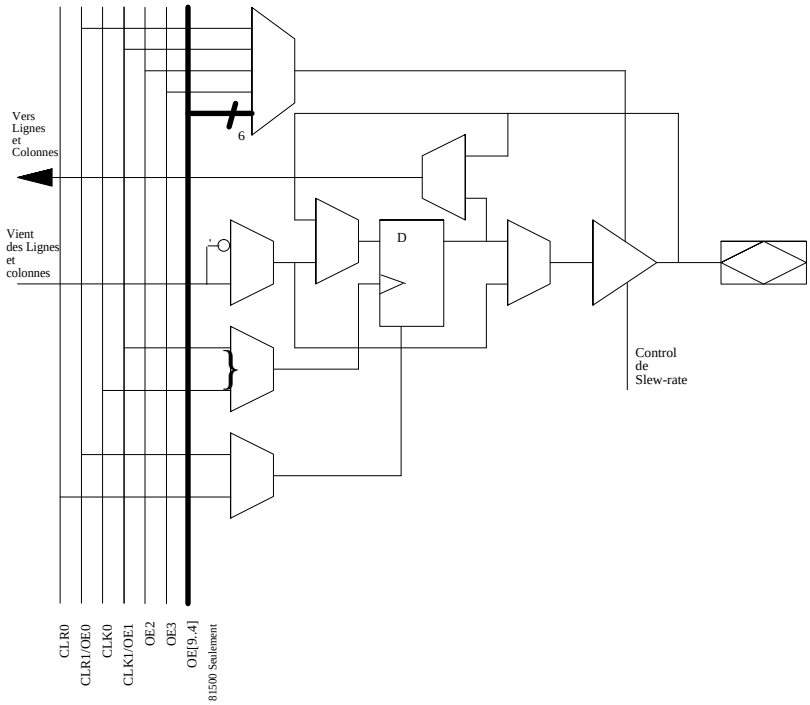
Architecture



Cellule de base



Elément d'entrée-sortie

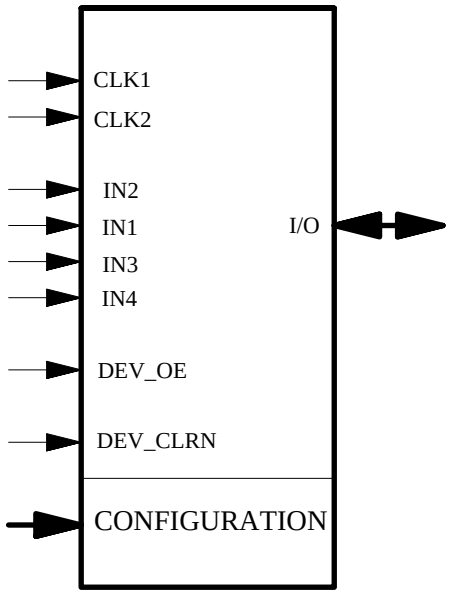


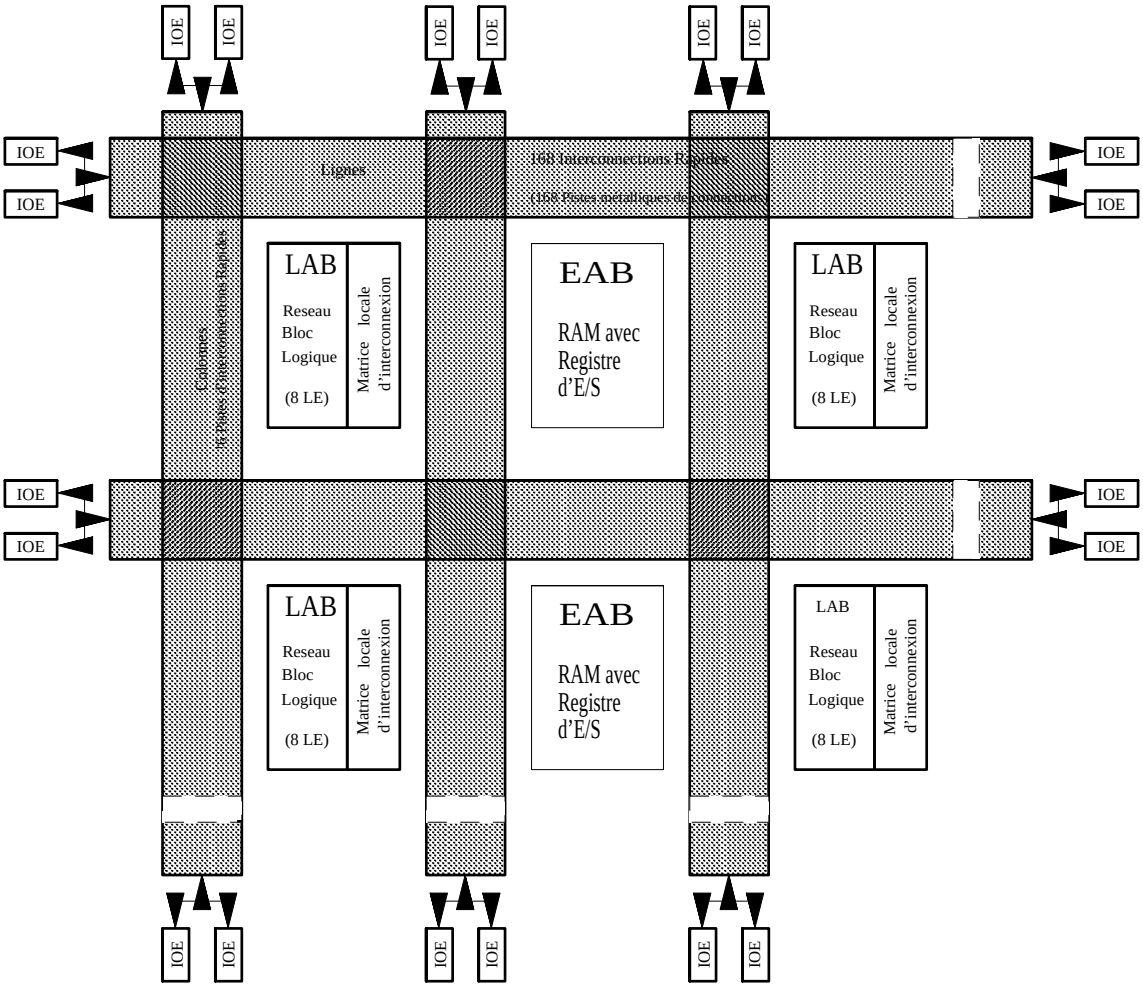
L'élément d'entrée-sortie contient un buffer bidirectionel et un registre qui peut-être utilisé soit comme registre d'entrée ou comme registre de sortie.

La famille 10K a la particularité de contenir de la RAM. Cela permet la mise en oeuvre de ram, de fifo dans votre circuit. La cellule de base et l'élément d'entrée-sortie ont été améliorés. On dispose de deux entrées globales d'horloge et de quatre entrées dédicacées. Ces entrées utilisent des pistes de routage spéciales avec des délais et un "skew" plus faibles que les pistes d'interconnections rapides.

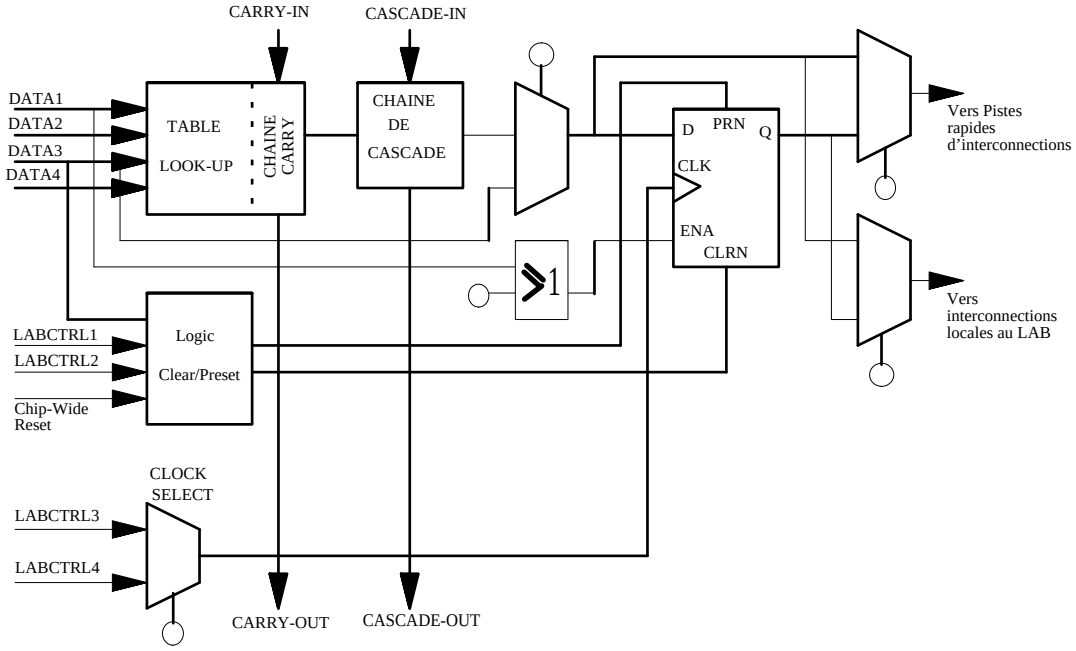
Les composants de la famille 10ke

	10K30	10K50	10K100	10K130	10k200
BASCULES	1728	2880	4992	6656	
LABS	216	360	624	832	
E/S	246	310	406	470	
EAB	6	10	12	16	
BIT RAM	12288	20480	24576	32768	

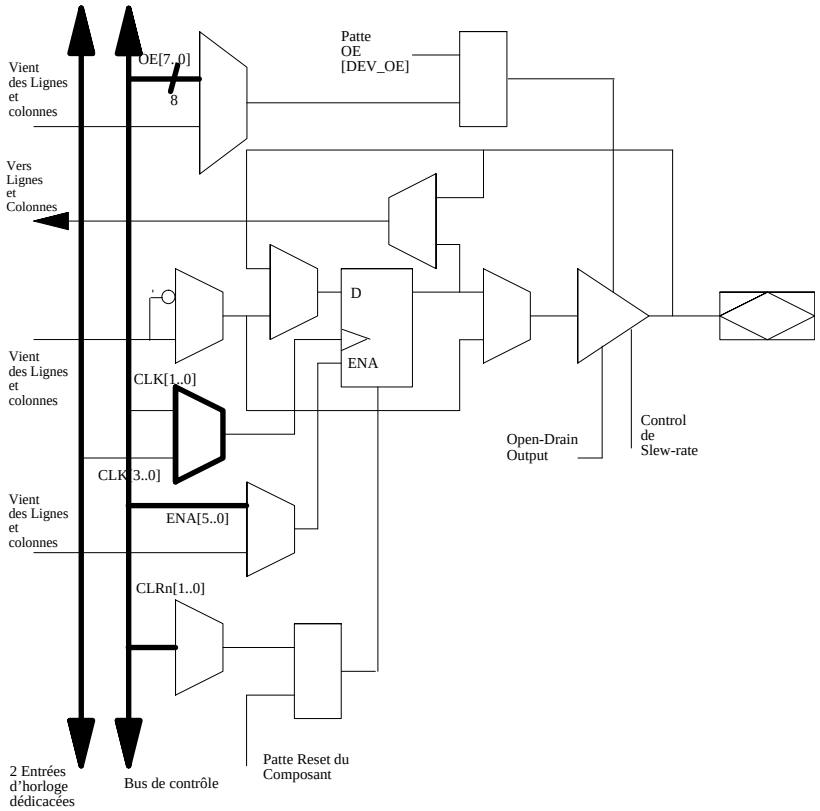



IOE
Lignes
168 Interconnections Rapides
(168 Pistes métalliques de Connexion)
Colonnes
16 Pistes d'interconnexion Rapides
LAB
Reseau
Bloc
Logique
(8 LE)
Matrice locale
d'interconnexion
EAB
RAM avec
Registre
d'E/S

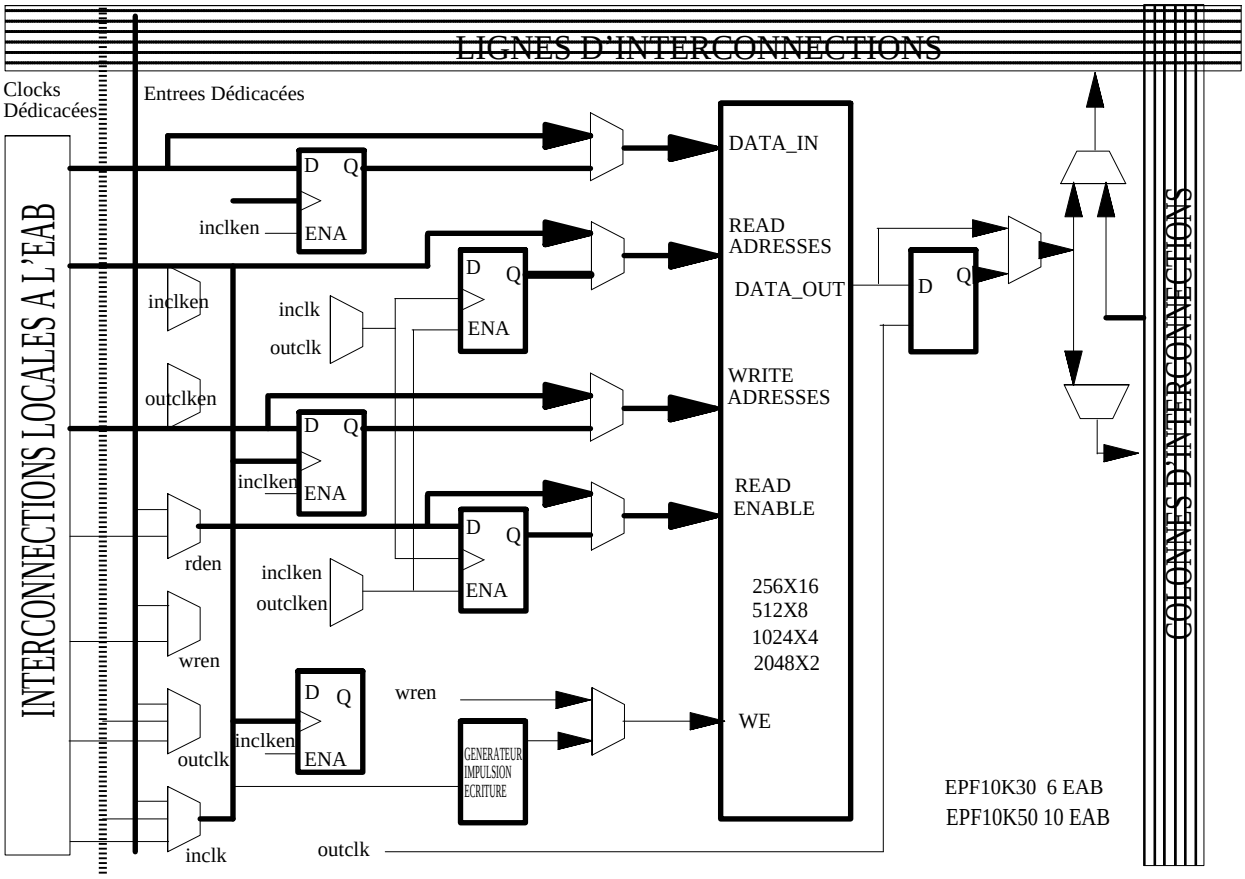
Cellule de base



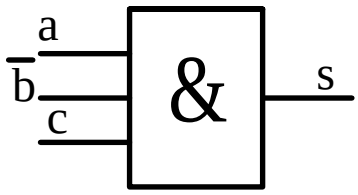
Elément d'entrée-sortie



Mémoire double port



Porte ET



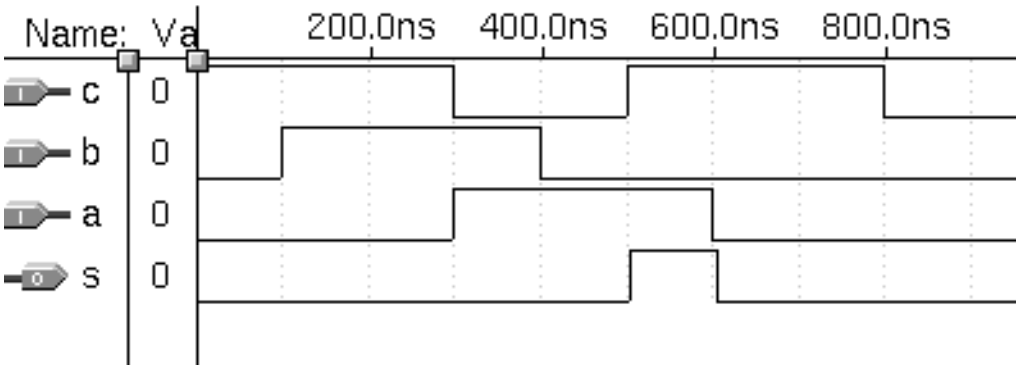
Fonction booléenne

s = a . !b . c
s égal a et b barre et c

Fichier source VHDL

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY combinatoire IS
PORT
(
    a : IN STD_LOGIC ;
    b : IN STD_LOGIC ;
    c : IN STD_LOGIC ;
    s : OUT STD_LOGIC
) ;
END combinatoire ;

ARCHITECTURE simple OF combinatoire IS

BEGIN

    s <= a AND NOT b AND c ;

END simple ;
```

Résultat de la simulation



Et - Exemple 2

Traduction littérale de l'instruction VHDL
s est égal à 1 quand a égal 1 et b égal 0 et c égal 1 autrement s est égal à 0

Fichier source VHDL

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY combinatoire IS
PORT
(
        a : IN STD_LOGIC ;
        b : IN STD_LOGIC ;
        c : IN STD_LOGIC ;
        s : OUT STD_LOGIC
) ;
END combinatoire ;

ARCHITECTURE simple OF combinatoire IS

BEGIN

        s <= '1' WHEN a='1' AND b='0' AND c='1' ELSE '0' ;

END simple ;
```

Fichier rapport

```
s       = LCELL( _EQ001 $  GND);
          _EQ001 =  a & !b &  c;
```

Porte ET- Exemple 3

Traduction littérale de l'instruction VHDL

Chaque fois qu'il y a un changement d'état d'une des variables a, b et c
si a égal 1 et b égal 0 et c égal 1 alors
    s égal (reçoit) 1
autrement s égal 0

Fichier source VHDL

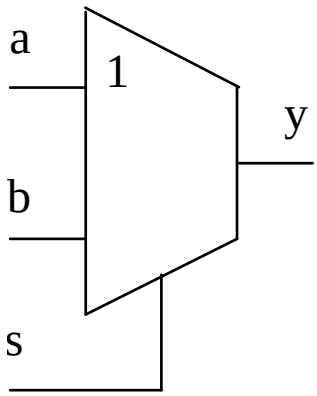
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY combinatoire IS
PORT
(
        a : IN STD_LOGIC ;
        b : IN STD_LOGIC ;
        c : IN STD_LOGIC ;
        s : OUT STD_LOGIC
) ;
END combinatoire ;

ARCHITECTURE simple OF combinatoire IS

BEGIN

        PROCESS (a,b,c)
        BEGIN
                IF a = '1' AND b = '0' AND c = '1' THEN
                        s <= '1' ;
                ELSE
                        s <= '0' ;
                END IF ;
        END PROCESS ;

END simple ;
```

Résultat de la synthèse Altera

```
s       = LCELL( _EQ001 $  GND);
          _EQ001 =  a & !b &  c;
```

Multiplexeur 2 Bits vers 1 Bit

Chaque fois qu'il y a un changement d'état d'une des variables a, b et s
si s égal 1 alors
	y égal a
autrement y égal b




```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY multiplexeur2x1 IS
PORT
(
	a : IN STD_LOGIC ;
	b : IN STD_LOGIC ;
	s : IN STD_LOGIC ;
	y : OUT STD_LOGIC
) ;
END multiplexeur2x1 ;

ARCHITECTURE if_then_else OF multiplexeur2x1 IS

BEGIN

	PROCESS (a,b,s)
	BEGIN
		IF s = '1' THEN
			y <= a ;
		ELSE
			y <= b ;
		END IF ;
	END PROCESS ;

END if_then_else ;
```

Résultat de la synthèse Altera

```
y      = LCELL( _EQ001 $  GND);
  _EQ001 =  a &  s
       #  b & !s;
```

Multiplexeur de 8 bits vers 1 bit avec signal de validation

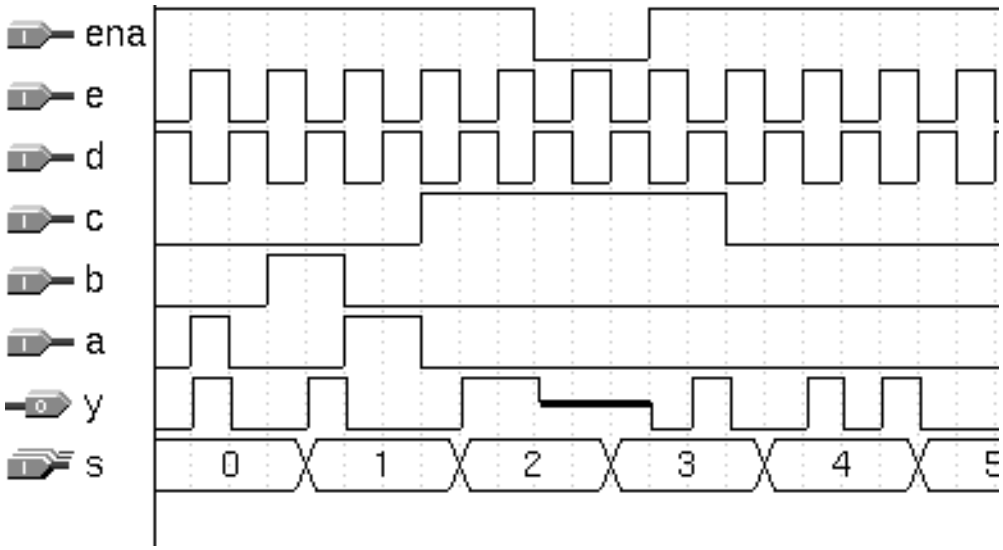
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY multiplexeur8vers1en IS
PORT(
        a : IN STD_LOGIC ;
        b : IN STD_LOGIC ;
        c : IN STD_LOGIC ;
        d : IN STD_LOGIC ;
        e : IN STD_LOGIC ;
        f : IN STD_LOGIC ;
        g : IN STD_LOGIC ;
        h : IN STD_LOGIC ;
        s : IN STD_LOGIC_VECTOR(2 DOWNTO 0) ;
        ena : IN STD_LOGIC ;
        y : OUT STD_LOGIC ) ;
END multiplexeur8vers1en ;

ARCHITECTURE instruction_case OF multiplexeur8vers1en IS

BEGIN

        PROCESS (a,b,c,d,e,f,g,h,s,ena)
        BEGIN
                IF ena = '1' THEN
                CASE s IS
                        WHEN "000" =>
                                y <= a ;
                        WHEN "001" =>
                                y <= b ;
                        WHEN "010" =>
                                y <= c ;
                        WHEN "011" =>
                                y <= d ;
                        WHEN "100" =>
                                y <= e ;
                        WHEN "101" =>
                                y <= f ;
                        WHEN "110" =>
                                y <= g ;
                        WHEN "111" =>
                                y <= h ;
                        WHEN OTHERS =>
                                y <= '0' ;
                END CASE ;
                ELSE
                        y <= 'Z' ;
                END IF ;
        END PROCESS ;

END
```

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY multiplexeur8vers1en IS
PORT
(
        a : IN STD_LOGIC ;
        b : IN STD_LOGIC ;
        c : IN STD_LOGIC ;
        d : IN STD_LOGIC ;
        e : IN STD_LOGIC ;
        f : IN STD_LOGIC ;
        g : IN STD_LOGIC ;
        h : IN STD_LOGIC ;
        s : IN STD_LOGIC_VECTOR(2 DOWNTO 0) ;
        ena : IN STD_LOGIC ;
        y : OUT STD_LOGIC
) ;
END multiplexeur8vers1en ;

ARCHITECTURE instruction_select OF multiplexeur8vers1en IS

        SIGNAL n1 : STD_LOGIC ;

BEGIN

        WITH s SELECT
                n1 <= a WHEN "000",
                        b WHEN "001",
                        c WHEN "010",
                        d WHEN "011",
                        e WHEN "100",
                        f WHEN "101",
                        g WHEN "110",
                        h WHEN "111",
                        '0' WHEN OTHERS ;

        y <= n1 WHEN ena='1' ELSE 'Z' ;
END instruction_select ;
```

Résultat de la synthèse Altera

```
y       = TRI(_LC049,  ena);
_LC049   = LCELL( _EQ001 $  VCC);
  _EQ001 =  _X001 &  _X002 &  _X003 &  _X004 &  _X005 &  _X006 &  _X007 &
            _X008;
  _X001  = EXP( d &  ena &  s0 &  s1 & !s2);
  _X002  = EXP( ena &  h &  s0 &  s1 &  s2);
  _X003  = EXP( ena &  f &  s0 & !s1 &  s2);
  _X004  = EXP( ena &  g & !s0 &  s1 &  s2);
  _X005  = EXP( b &  ena &  s0 & !s1 & !s2);
  _X006  = EXP( c &  ena & !s0 &  s1 & !s2);
  _X007  = EXP( e &  ena & !s0 & !s1 &  s2);
  _X008  = EXP( a &  ena & !s0 & !s1 & !s2);
```

Simulation



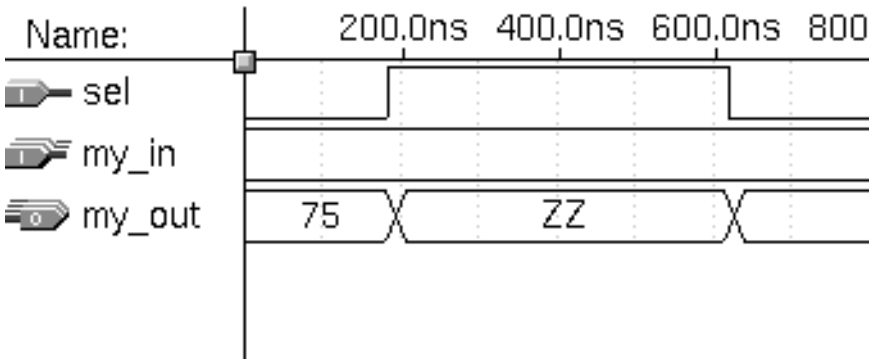
Utilisation de la logique trois états

```
LIBRARY IEEE;
USE ieee.std_logic_1164.ALL;

ENTITY bus3etats IS
  PORT(
    my_in  : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
    sel    : IN STD_LOGIC ;
    my_out : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END bus3etats ;

ARCHITECTURE simple OF bus3etats IS

BEGIN

  my_out <= "ZZZZZZZZ" WHEN (sel = ’1’) ELSE my_in;

END simple ;
```

Extrait du fichier rapport de maxplus2 .rpt

```
~
my_out0  = TRI(_LC126, !sel);
_LC126   = LCELL( my_in0 $  GND);
```

Simulation



Tous les circuits de logiques séquencielles utilisent un ou plusieurs registres.

Les machines d'états sont un moyen très puissant de décrire de la logique séquencielle.

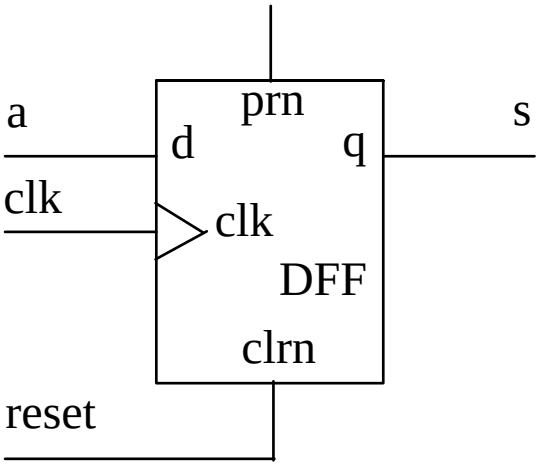
Les compteurs, les registres à décalage font aussi partie de la famille "logique séquencielle"

Implémentation de registres

Les instructions VHDL suivantes inférent des registres, lorsqu'elles sont dans le corps d'un processus

Instruction IF signal_d'horloge'EVENT AND signal_d'horloge='1' THEN
Instruction WAIT UNTIL signal_d'horloge'EVENT AND signal_d'horloge='1'

Bascule d



Fichier source

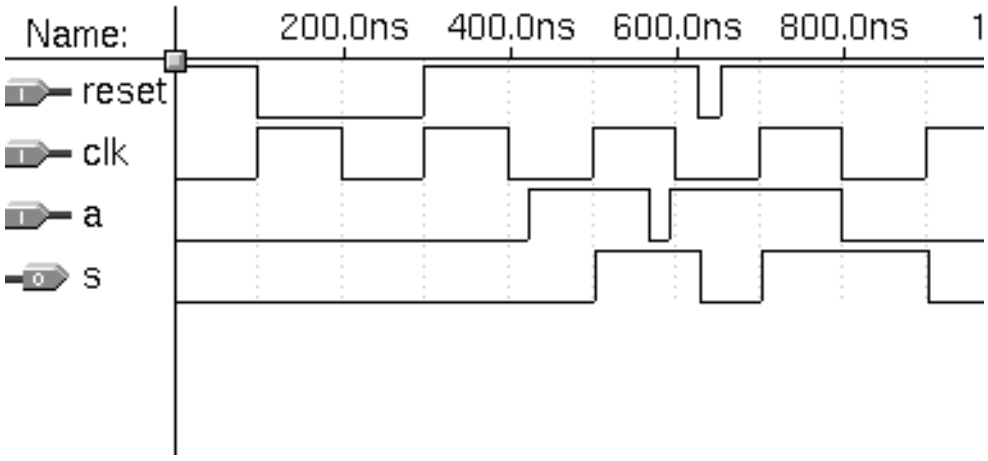
```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY registre IS
PORT
(
   a     : IN STD_LOGIC ;
   clk   : IN STD_LOGIC ;
   reset : IN STD_LOGIC ;
   s     : OUT STD_LOGIC
) ;
END registre ;

ARCHITECTURE processus OF registre IS

BEGIN

   PROCESS (reset,clk)
   BEGIN
            IF reset = '0' THEN
                     s <= '0' ;
            ELSIF clk'EVENT AND clk = '1' THEN
                     s <= a ;
            END IF ;
   END PROCESS ;

END processus ;
```

La liste de sensibilité du processus comprend les signaux **reset** et **clk**. A chaque changement d'état sur un de ces deux signaux, le processus va être activé. L'instruction IF [ou ELSIF] clk'EVENT AND clk = '1' est utilisée pour décrire de la logique synchrone.

Résultat de la synthèse

s = DFFE(a $ GND, GLOBAL(clk), GLOBAL(reset), VCC, VCC);

Simulation



un composant de la librairie en précisant pour chaque port du composant les signaux connectés.

Instanciation d'une bascule dffe

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

LIBRARY altera ;
USE altera.maxplus2.all ;

ENTITY registre IS
PORT
(
  a     : IN STD_LOGIC ;
  clk   : IN STD_LOGIC ;
  ena   : IN STD_LOGIC ;
  preset : IN STD_LOGIC ;
  reset : IN STD_LOGIC ;
  s     : OUT STD_LOGIC
) ;
END registre ;

ARCHITECTURE instanciation OF registre IS

BEGIN

  instance1 : dffe
          PORT MAP (d => a ,
                    clk => clk ,
                    ena => ena ,
                    prn => preset ,
                    clrn => reset ,
                    q => s ) ;

END instanciation ;
```

Résultat de la synthèse

```
s      = DFFE( a $  GND, GLOBAL( clk), GLOBAL( reset),  preset,  ena);
```

Compteur

```
LIBRARY ieee;
USE ieee.std_logic_signed.all;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ENTITY decompteur IS
PORT
(
      clear : IN STD_LOGIC ;
      load  : IN STD_LOGIC ;
      clk   : IN STD_LOGIC ;
      din   : IN STD_LOGIC_VECTOR (7 DOWNTO 0) ;
      zero  : OUT STD_LOGIC
);
END decompteur ;

ARCHITECTURE fonctionnelle OF decompteur IS

      SIGNAL cpt : STD_LOGIC_VECTOR (7 DOWNTO 0) ;

BEGIN

      -- Process
      PROCESS (clk,clear)
      BEGIN
                  IF clear = '1' THEN
                              cpt <= (OTHERS => '0') ;
                  ELSIF clk'event and clk = '1' THEN
                              IF (load = '1') THEN
                                          cpt <= din ;
                              ELSE
                                          cpt <= cpt - 1 ;
                              END IF ;
                  END IF ;
      END PROCESS ;

      zero <= '1' WHEN cpt = 0 ELSE '0' ;

END fonctionnelle ;
```

Résultat de la synthèse

Version 9.01 07/30/98

EPM7128ELC84-7 11 1 0 16 0 12 %

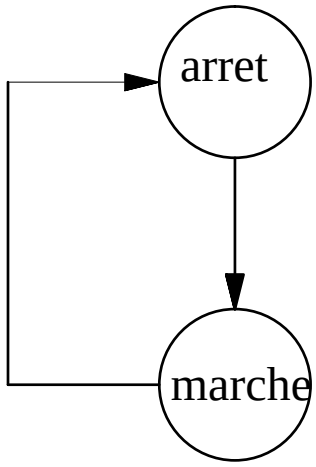
INFO: Signal 'clk' chosen for auto global Clock

** FILE HIERARCHY **

|LPM_ADD_SUB:99|
LPM_ADD_SUB:99	addcore:adder
LPM_ADD_SUB:99	altshift:result_ext_latency_ffs
LPM_ADD_SUB:99	altshift:carry_ext_latency_ffs
LPM_ADD_SUB:99	altshift:oflow_ext_latency_ffs

Compteur avec instanciation lpm_counter

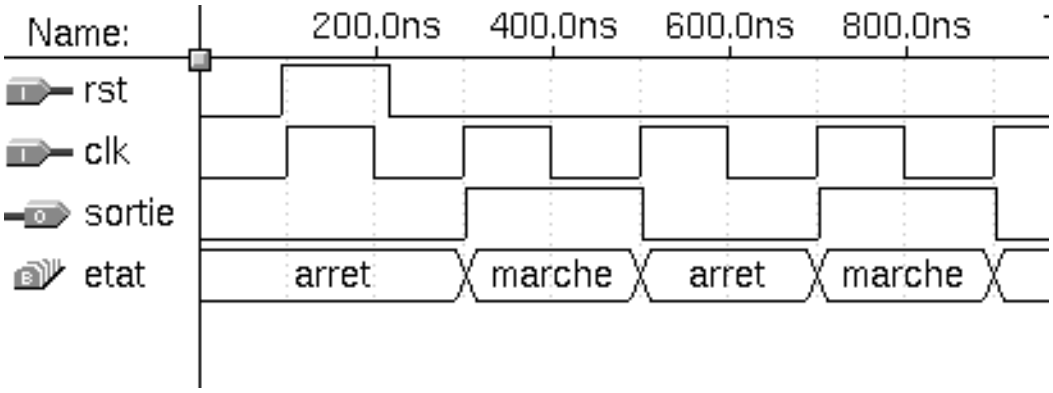
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY lpm ;
USE lpm.lpm_components.all ;

ENTITY decompteur IS
PORT
(
    clear : IN STD_LOGIC ;
    load  : IN STD_LOGIC ;
    clk   : IN STD_LOGIC ;
    din   : IN STD_LOGIC_VECTOR (3 DOWNTO 0) ;
    zero  : OUT STD_LOGIC
);
END decompteur ;

ARCHITECTURE fonctionnelle OF decompteur IS

    SIGNAL egal : STD_LOGIC_VECTOR (15 DOWNTO 0) ;

BEGIN


    instance1 : lpm_counter
            GENERIC MAP (
                    LPM_WIDTH => 8,
                    LPM_DIRECTION => "DOWN"
            )
            PORT MAP (
                    clock => clk,
                    aclr => clear,
                    sload => load,
                    data => din,
                    eq => egal
            ) ;

    zero <= egal(0) ;

END fonctionnelle ;
```



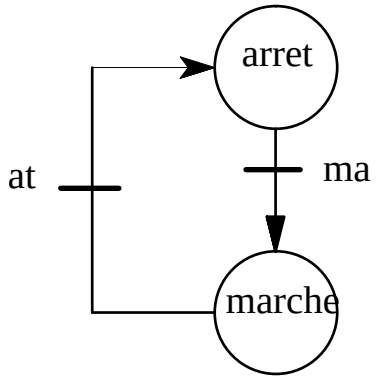

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY machine IS
        PORT (clk : IN BIT ;
                rst : IN BIT ;
                sortie : OUT BIT) ;
END machine ;

ARCHITECTURE fonctionnelle OF machine IS

        TYPE type_etat IS (arret,marche) ;
        SIGNAL etat : type_etat ;

BEGIN

        PROCESS (clk,rst)
        begin
                IF rst = '1' THEN
                        etat <= arret ;
                ELSIF (clk'event and clk = '1') THEN
                        CASE etat IS
                                WHEN arret =>
                                        etat <= marche ;
                                WHEN marche =>
                                        etat <= arret ;
                                WHEN others =>
                                        etat <= arret ;
                        END CASE ;
                END IF ;
        END PROCESS ;

        sortie <= '1' WHEN etat=marche ELSE '0' ;


END fonctionnelle ;
```

Résultat de la synthèse

```
** STATE MACHINE ASSIGNMENTS **

etat: MACHINE
      OF BITS (
        etat~1
      )
      WITH STATES (
                    arret = B"0",
                   marche = B"1"
);

sortie  = TFFE( VCC, GLOBAL( clk), !rst,  VCC,  VCC);
```



Machine d'état avec variables d'entrées asynchrones

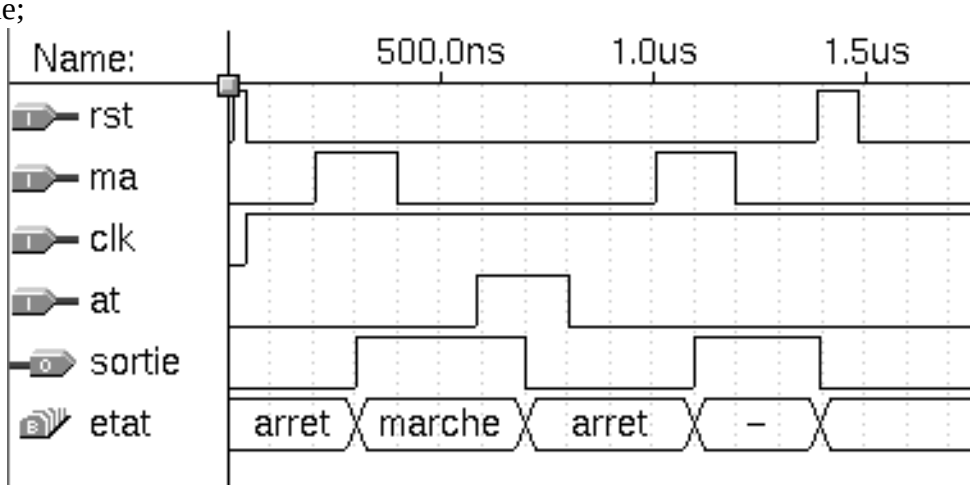



```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY machine IS

PORT (clk : IN BIT ;
      rst : IN BIT ;
      ma  : IN BIT ;
      at  : IN BIT ;
      sortie : OUT BIT) ;
END machine ;

ARCHITECTURE fonctionnelle OF machine IS

      TYPE type_etat IS (arret,marche) ;
      SIGNAL etat : type_etat ;

BEGIN

      PROCESS (clk,rst)
      begin
            IF rst = '1' THEN
                  etat <= arret ;
            ELSIF (clk'event and clk = '1') THEN
                  CASE etat IS
                        WHEN arret =>
                              IF ma='1' THEN
                                    etat <= marche ;
                              END IF ;
                        WHEN marche =>
                              IF at='1' THEN
                                    etat <= arret ;
                              END IF ;
                        WHEN others =>
                              etat <= arret ;
                  END CASE ;
            END IF ;
      END PROCESS ;

      sortie <= '1' WHEN etat=marche ELSE '0' ;

end fonctionnelle ;
```

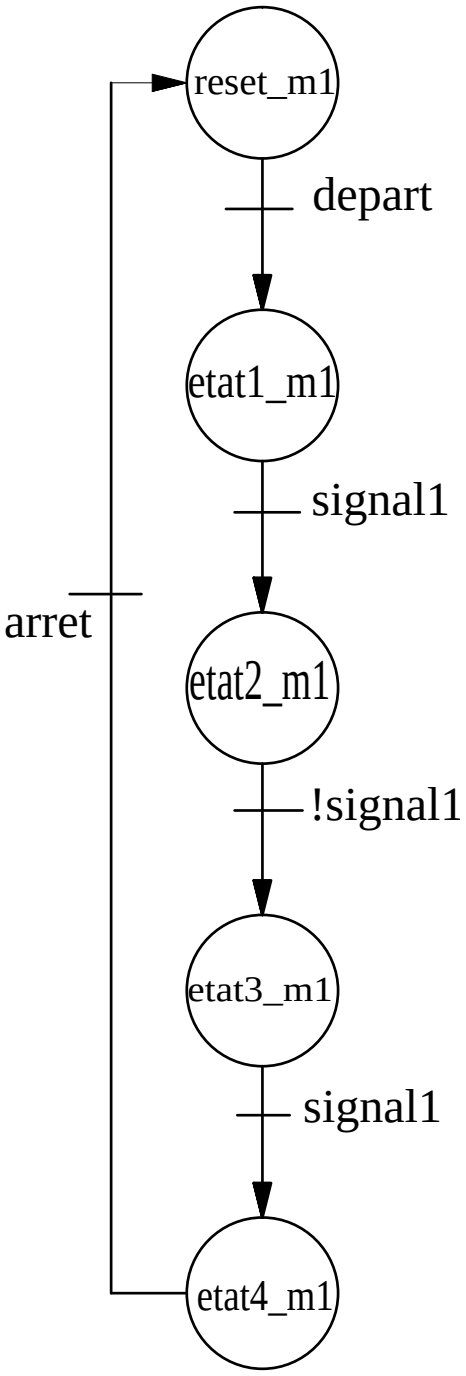
Résultat de la synthèse

```
sortie  = DFFE( _EQ001 $  VCC, GLOBAL( clk), !rst,  VCC,  VCC);
 _EQ001 =  at &  sortie
       # !ma & !sortie;
```

Simulation



Machine de Moore ou de Mealy



Fichier source vhdl

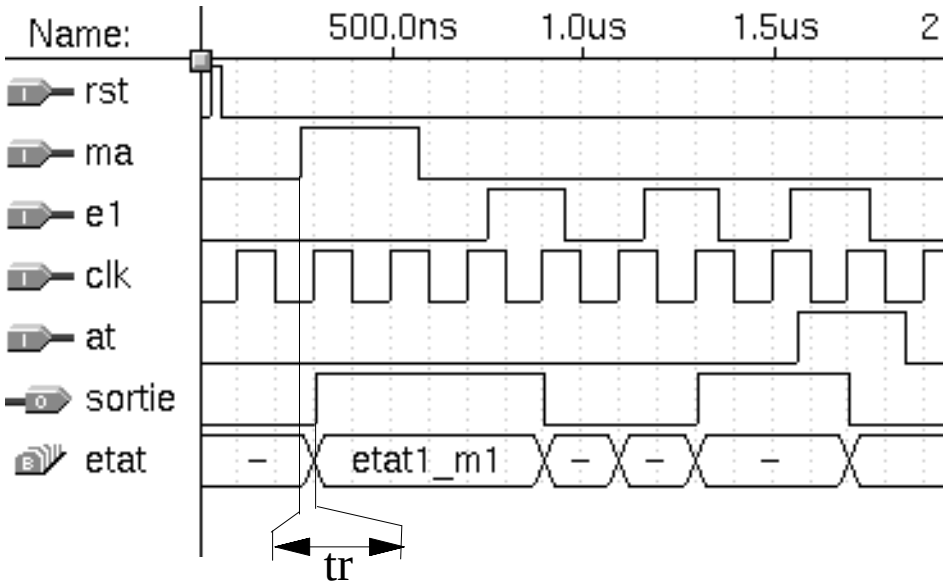
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY machine IS

PORT (clk : IN BIT ;
        rst : IN BIT ;
        ma  : IN BIT ;
        at  : IN BIT ;
        e1  : IN BIT ;
        sortie : OUT BIT) ;
END machine ;

ARCHITECTURE fonctionnelle OF machine IS

        TYPE type_etat IS (reset_m1,etat1_m1,etat2_m1,etat3_m1,etat4_m1) ;
        SIGNAL etat : type_etat ;

BEGIN

        PROCESS (clk,rst)
        begin
                IF rst = '1' THEN
                        etat <= reset_m1 ;
                ELSIF (clk'event and clk = '1') THEN
                        CASE etat IS
                                WHEN reset_m1 =>
                                        IF ma='1' THEN
                                                etat <= etat1_m1 ;
                                        END IF ;
                                WHEN etat1_m1 =>
                                        IF e1='1' THEN
                                                etat <= etat2_m1 ;
                                        END IF ;
                                WHEN etat2_m1 =>
                                        IF e1='0' THEN
                                                etat <= etat3_m1 ;
                                        END IF ;
                                WHEN etat3_m1 =>
                                        IF e1='1' THEN
                                                etat <= etat4_m1 ;
                                        END IF ;
                                WHEN etat4_m1 =>
                                        IF at='1' THEN
                                                etat <= reset_m1 ;
                                        END IF ;
                                WHEN others =>
                                        etat <= reset_m1 ;
                        END CASE ;
                END IF ;
        END PROCESS ;

        sortie <= '1' WHEN etat=etat1_m1
                OR etat=etat4_m1 ELSE '0' ;

end fonctionnelle ;
```

Résultat de la synthèse

```
etat: MACHINE
      OF BITS (
        etat~3,
        etat~2,
        etat~1
      )
      WITH STATES (
                  reset_m1 = B"000",
                  etat1_m1 = B"110",
                  etat2_m1 = B"010",
                  etat3_m1 = B"011",
                  etat4_m1 = B"001"
);

 sortie  = LCELL( _EQ004 $  GND);
  _EQ004 = !etat~1 &  etat~2 &  etat~3
       #  etat~1 & !etat~2 & !etat~3;
```

On peut remarquer que l'équation booléenne de commande1 est seulement fonction des états de la machine machine1.

Simulation



l'état de la machine et d'une variable d'entrée asynchrone.
Le signal commande1 étant synchrone de l'horloge, il pourra avoir un retard d'une période d'horloge dans le pire cas. Si on veut éliminer ce délai, on peut tenir compte à la fois de l'état de la machine et de l'état du signal d'entrée (respectivement depart et signal1 dans cet exemple) pour générer le signal commande1.

Les sorties de la machine sont alors asynchrones. On dit que l'on a une machine de **mealy**.

Souvent on a une machine de mealy sans même le savoir.

Modification de l'équation booléenne de commande1 dans le fichier VHDL.

```
sortie <= '1' WHEN (etat=reset_m1 AND ma='1')
   OR etat=etat1_m1
   OR (etat=etat3_m1 AND e1='1')
   OR etat=etat4_m1 ELSE '0' ;
```

L'équation de sortie est maintenant fonction des états de la machine d'état et de l'état des variables d'entrées

```
 sortie  = LCELL( _EQ004 $ !etat~3);
 _EQ004 =  etat~1 &  etat~2 & !etat~3 & !e1
       # !etat~1 & !etat~3 & !ma
       # !etat~1 &  etat~2;
```

Résultat de la simulation