FPGA1 ou l'aventure de la conception des Systèmes Programmables

Bertrand Granado
Enseignant-Chercheur

LIP6 / UPMC
Mel : Bertrand.Granado@upmc.fr

Hiver 2018

Plan

1. Introduction
2. Outils de conception : VHDL
3. Méthodes : Machines à états
4. Les mémoires
5. FPGA
6. Le Port Jtag
7. Conception d'un système programmable

Introduction

- Equipe enseignante : Julien Denoulet, Amine Rhouni, Bertrand Granado
- Contrôle des connaissances : ER (50 %)+ Projet (50 %)
- Projet : Maitrise des outils (MicroBlaze + IP Matérielles) et Gestion d'un train électrique avec le protocole DCC

VHDL

1 Introduction
2 Outils de conception : VHDL
- Différents types de description
- Les types en VHDL
- Signaux et Variables en VHDL
- Les tableaux
- Générique
- Clause Wait
- Test Bench
- Simulation
- Paquetage, Procédure et Fonction

3 Méthodes : Machines à états
- Les Machines à Etats en VHDL

4 Les mémoires
5 FPGA
6 Le Port Jtag
7 Conception d'un système programmable

Bibliographie

- The designer's guide to VHDL - Peter Ashenden, Morgan Kaufman (http://www.ashenden.com.au/designers-guide/DG.html)
- Alain Vachoux : Digital System Modeling - http://lsmwww.epfl.ch/design_languages/
- Cours Master SdI - Patrick Garda (patrick.garda@upmc.fr)
- Ressources
  - Hamburg VHDL archive :
    - http://tech-www.informatik.uni-hamburg.de/vhdl/vhdl.htm
  - http://www.geocities.com/SiliconValley/Heights/8831/websrc.html

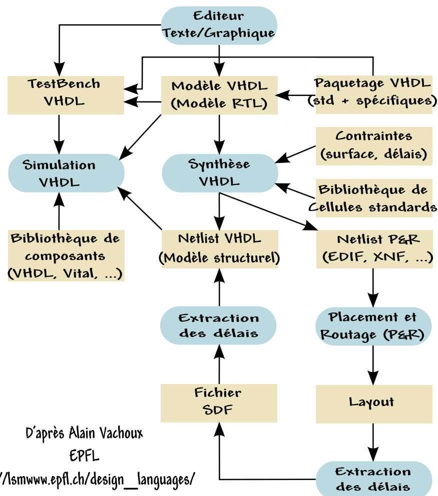
Outils

- Free Model Foundation - http://www.freemodelfoundry.com/
- Opencores - http://www.opencores.org
- XILINX : ISE - http://www.xilinx.com/
- ALTERA : Quartus - http://www.altera.com/
- Mentor Graphics : HDL Designer, ModelSim, Precision - http://www.mentor.com/

Historique

- VHDL : VHSIC Hardware Description Language Historique
- Langage introduit dans le cadre du projet DARPA VHSIC : Very High Speed Integrated Circuits
- IBM, Texas Instruments et Intermetrics ont obtenu le contrat en 1983 et produit VHDL “7.2” en 1985
- Proposé à IEEE pour normalisation en 1986
- Norme en 1987 : IEEE Std 1076-1987, dit VHDL-87
- Norme révisée en 1992 : IEEE Std 1076-1993, dit VHDL-93
- Normalisation simultanée de IEEE 1164-1993, dit std_logic_1164
- Norme révisée en 2002 IEEE Std 1076-2002, VHDL - 2002

Flot de conception VHDL



D'après Alain Vachoux
EPFL
http://lsmwww.epfl.ch/design__languages/

VHDL - RTL

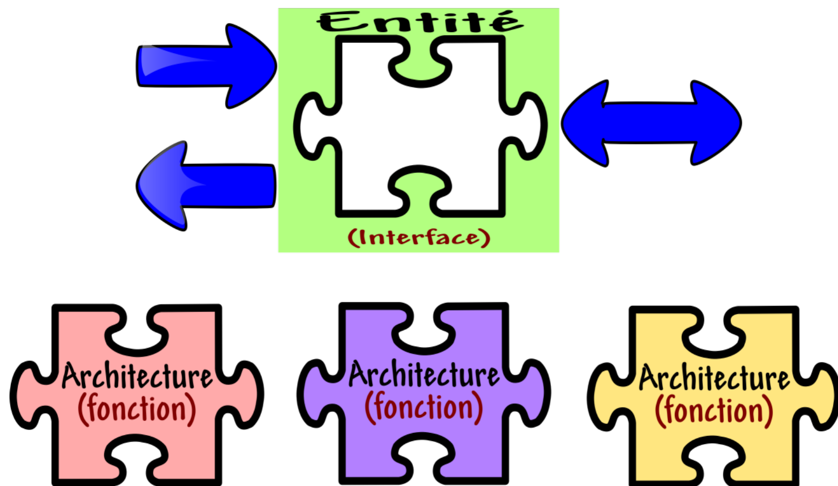
- RTL : Register Transfert Level
- Description Synthétisable
- Utilisable pour configurer un circuit logique programmable
- Sous ensemble de constructions VHDL

VHDL - Base

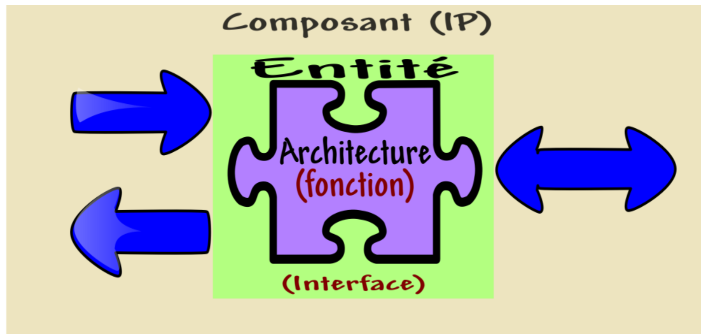
Blocs de base

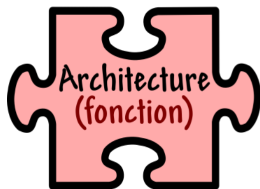
1. Les bibliothèques (ou librairies)
2. L'entité : Décrit l'interfaçage du composant
3. L'architecture : Décrit le fonctionnement du composant

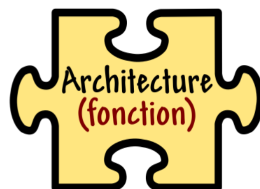
VHDL - Base



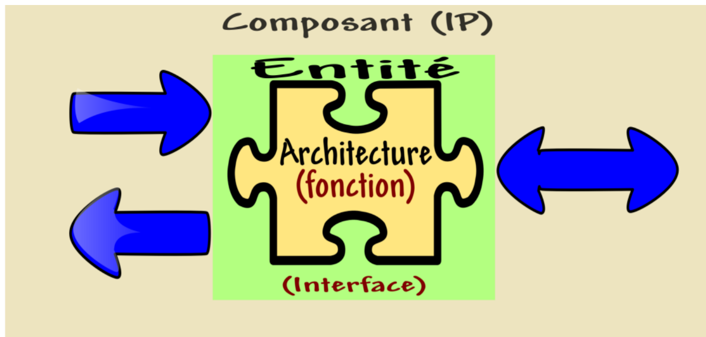
VHDL - Base

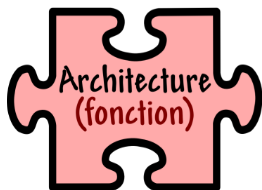


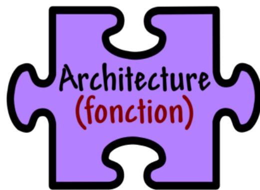




VHDL - Base







VHDL - Bibliothèque

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
```

VHDL - Entité

context-clause
entity entity-name is
[generic (parameter-list)]
[port (port-list) ;]
[local-declarations]

type, subtype, constant, signal, subprogram

[begin
passive-concurrent-statement]

concurrent procedure call, assertion, passive process

end [entity] [entity-name] ;

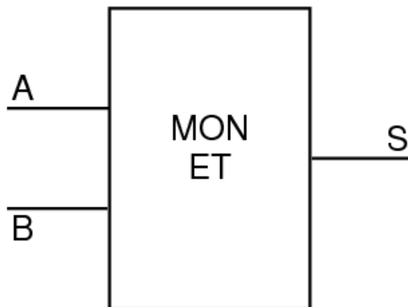
VHDL - Entité

```
generic(
parametre-name ,...: parametre-type [:=default-value];
...
parametre-name ,...: parametre-type [:=default-value]);
```

VHDL - Entité

```
port(
[signal] signal-name ,... : mode signal-type;
...
[signal] signal-name ,... : mode signal-type);
```

VHDL - Entité




```
entity MON-ET is
generic (tp: time := 2ns);
port( A : in std_logic;
      B : in std_logic;
      S : out std_logic);
end entity MON-ET;
```

VHDL - Architecture

concurrent procedure call, assertion, passive process

```
architecture archi-name of entity-name is
[ local-declaration ]
begin
concurrent-statement

end [ architecture ] [ archi-name ] ;
```

VHDL - L'architecture

S = A et B

```
architecture FLOT of MON-ET is
begin
    s <= a and b after tp;
end architecture FLOT;
```

VHDL - L'architecture

- Une architecture décrit le fonctionnement d'une entité de conception
- C'est-à-dire la détermination des sorties en fonction des entrées
- Une architecture peut être décrite de différentes manières :
  - Flot de données
  - Structurelle
  - Comportementale
  - RTL : Register Transfer Level

Plan

1 Introduction

2 Outils de conception : VHDL
- Différents types de description
- Les types en VHDL
- Signaux et Variables en VHDL
- Les tableaux
- Générique
- Clause Wait
- Test Bench
- Simulation
- Paquetage, Procédure et Fonction

3 Méthodes : Machines à états

4 Les mémoires

1. Introduction
2. Outils de conception : VHDL
   - Différents types de description
   - Les types en VHDL
   - Signaux et Variables en VHDL
   - Les tableaux
   - Générique
   - Clause Wait
   - Test Bench
   - Simulation
   - Paquetage, Procédure et Fonction
3. Méthodes : Machines à états
   - Les Machines à Etats en VHDL
4. Les mémoires
5. FPGA
6. Le Port Jtag
7. Conception d'un système programmable

Architecture Flot de données

- Une architecture “flot de données” (dataflow) décrit un circuit au niveau des portes logiques
- Elle représente le flot des informations des entrées (binaires) vers les sorties à l’aide d’opérateurs combinatoires :
  - not, and, nand, or, nor, xor, xnor
- Les signaux se propagent de façon asynchrone
- La détermination des valeurs est faite par l’instruction d’affectation des signaux $<=$
- Elle permet de modéliser le temps de propagation des portes grâce à la clause `after`

Architecture Flot de données

```
architecture flot of addc is
begin
sum <= a xor b xor cin after 2 ns ; -- somme
cout <= (a and b) or (b and cin) or (cin and a) after 1 ns ;
 -- retenue
end architecture flot;
```

Architecture Structurelle

- Utilise des composants déjà définis
- L'architecture structurelle définit alors la structure comme un assemblage d'instances de composants reliées par des signaux
- Elle correspond à la description textuelle d'un schéma utilisant des cellules de bibliothèque

Architecture Structurelle

```
 entity add4 is port (
a, b : in std_logic_vector (3 downto 0) ;
s : out  std_logic_vector (4 downto 0) )
end entity  add4 ;
```

Architecture Structurelle

```
architecture struct of add4 is
signal c : std_logic_vector (3 downto 1) ;
begin
addc_i0 : entity work.addc(flot)
port map ( a(0),  b(0),  '0',  s(0),  c(1)) ;
addc_i1 : entity work.addc(flot)
port map ( a(1),  b(1),  c(1),  s(1),  c(2)) ;
addc_i2 : entity work.addc(flo)t
port map ( a(2),  b(2),  c(2),  s(2),  c(3)) ;
addc_i3 : entity work.addc(flot)
port map ( a(3),  b(3),  c(3),  s(3),  s(4)) ;
end architecture struct ;
```

Architecture Structurelle

```
architecture struct of add4 is
signal c : std_logic_vector (3 downto 1) ;
begin
addc_i0 : entity work.addc(flot)
port map (a => a(0), b => b(0), cin => '0', sum => s(0), cout => c(1)) ;
addc_i1 : entity work.addc(flot)
port map (a => a(1), b => b(1), cin => c(1), sum => s(1), cout => c(2)) ;
addc_i2 : entity work.addc(flo)t
port map (a => a(2), b => b(2), cin => c(2), sum => s(2), cout => c(3)) ;
addc_i3 : entity work.addc(flot)
port map (a => a(3), b => b(3), cin => c(3), sum => s(3), cout => s(4)) ;
end architecture struct ;
```

L'instantiation de composants PORT MAP

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

entity demiadd is
port( a, b : in std_logic;
      c, s : out std_logic);
end entity demiadd;

architecture flot of demiadd is
begin
s <= a xor b;
c <= a and b;
end architecture flot;
```

L'instantiation de composants PORT MAP

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

entity add is
port( a, b, cin : in std_logic;
      cout, s : out std_logic);
end entity add;

architecture struct of add is
signal stmp, ctmp1,ctmp2 : std_logic;
begin
     demiadd1 : entity work.demiadd(flot)
                port map(a,b,stmp,ctmp1);
     demiadd2 : entity work.demiadd(flot)
                port map(cin,stmp,s,ctmp2);
     cout <= ctmp1 or ctmp2;
end architecture struct;
```

Architecture Comportementale

- L'architecture est représentée par un ensemble de processus séquentiels qui s'exécutent simultanément
- Chaque processus est en attente jusqu'à ce que l'une de ses entrées change de valeur
- Lorsque cela se produit le code du processus est exécuté séquentiellement
- Lorsqu'on arrive à la fin d'un processus on recommence son exécution au début
- La réalisation électronique de l'architecture n'est pas décrite

Le Process

- Permet de réaliser des parties séquentielles
- Introduit un niveau d'abstraction proche de l'informatique

Le Process

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

entity reg is
port( d : in std_logic_vector(7 downto 0);
      hor : in std_logic;
      q : out std_logic_vector(7 downto 0);
end entity reg;

architecture comport of reg is
begin
clocked: process(d,hor) is
      begin
          if (hor'event and hor = '1') then
               q<=d;
          end if;
      end process clocked;
end architecture comport;
```

Les variables

- Niveau d'abstraction niveau algorithmique
- Pas assimilable à un fil

L'affectation de variables

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

entity reg is
port( d : in std_logic_vector(7 downto 0);
      hor : in std_logic;
      q : out std_logic_vector(7 downto 0);
end entity reg;

architecture comport of reg is
begin
clocked: process(d,hor) is
      variable q_int : std_logic_vector(7 downto 0);
      begin
          if (hor'event and hor = '1') then
               q_int :=d;
          end if;
          q <= q_int;
      end process clocked;
end architecture comport;
```

Les structures de contrôle

- Les structures Conditionnelles
  - La structure IF
  - La structure CASE
- Les structure de Répétitions LOOP
  - La structure FOR

Conditionnelle IF

```
if condition1 then
instructions séquentielles 1
elsif condition2 then
instructions séquentielles 2
elsif condition3 then
instructions séquentielles 3
else
instructions séquentielles n
end if ;
```

Conditionnelle IF

```
library ieee;
use ieee.std_logic_1164.all;

entity decodeur is
port ( choix : in std_logic_vector(1 downto 0);
       decode : out std_logic_vector(3 downto 0));
end entity decodeur;

architecture comport of  decodeur
decodage : process(choix) is
           begin
             IF (choix= "00") THEN decode <="0001";
             ELSIF (choix="01") THEN decode <="0010";
             ELSIF (choix="10") THEN decode <="0100";
             ELSE  decode <="1000";
             END IF;
           end process decodage;
end architecture comport;
```

Conditionnelle IF

```
architecture behaviour of addc is
begin
 calc_sum : process (a, b, cin) is
 begin
   if ( ((a + b + cin) rem 2) = 0) then sum <= '0' after 2 ns ;
   else sum <= '1 ' after 2 ns ;
   end if ;
 end process calc_sum ;

 calc_cout :  process (a, b, cin) is
 begin
  if (a + b + cin >= 2) then cout <= '1' after 1 ns ;
  else cout <= '0' after 1 ns ;
  end if ;
 end process calc_cout ;
end architecture behaviour ;
```

Conditionnelle CASE

```
case expression is
when VALUE-1 =>
instructions séquentielles 1
when VALUE-2 — VALUE-3 =>
instructions séquentielles 2
when VALUE-M to VALUE-N =>
instructions séquentielles 3
when others =>
instructions séquentielles n
end case ;
```

Conditionnelle CASE

```
library ieee;
use ieee.std_logic_1164.all;

entity decodeur is
port ( choix : in std_logic_vector(1 downto 0);
       decode : out std_logic_vector(3 downto 0));
end entity decodeur;

architecture comport of  decodeur
decodage : process(choix) is
           begin
             CASE choix
                 WHEN "00" => decode <="0001";
                 WHEN "01" => decode <="0010";
                 WHEN "10" => decode <="0100";
                 WHEN "11" => decode <="1000";
                 WHEN OTHERS => NULL;
             END CASE;
           end process decodage;
end architecture comport;
```

Boucle FOR

[loop_label]
for identifier in discrete_range loop
– instructions du corps de boucle
end loop [loop_label] ;

- La variable `identifier` est
  - auto-déclarée
  - elle n'est visible que dans le corps de la boucle

Boucle FOR

```
entity additionneur is
    generic (N : natural := 8);
    port (a, b : in std_logic_vector(N-1 downto 0);
          s : out std_logic_vector(N-1 downto 0));

end additionneur;


architecture flot of additionneur is

begin  -- flot

    add : process(a,b)
        variable sum,btemp : std_logic_vector(N-1 downto 0);
        variable retenue : std_logic_vector(N downto 0);
    begin  -- process add
        btemps := b;
        retenue(0):='0';
        calcul : for i in 0 to N-1 loop
            sum(i):= a(i) xor btemp(i) xor retenue(i);
            retenue(i+1) := (a(i) and btemp(i)) or (a(i) and retenue(i)) or (btemp(i) and retenue(i));
        end loop calcul;
    end process add;
end flot;
```

Plan

1. Introduction

2. Outils de conception : VHDL
   - Différents types de description
   - Les types en VHDL
   - Signaux et Variables en VHDL
   - Les tableaux
   - Générique
   - Clause Wait
   - Test Bench
   - Simulation
   - Paquetage, Procédure et Fonction

3. Méthodes : Machines à états

4. Les mémoires

1. Introduction
2. Outils de conception : VHDL
   - Différents types de description
   - Les types en VHDL
   - Signaux et Variables en VHDL
   - Les tableaux
   - Générique
   - Clause Wait
   - Test Bench
   - Simulation
   - Paquetage, Procédure et Fonction
3. Méthodes : Machines à états
   - Les Machines à Etats en VHDL
4. Les mémoires
5. FPGA
6. Le Port Jtag
7. Conception d'un système programmable

Les différents Type VHDL

- Les types originaux
  - `Bit is ('0', '1');`
  - `Bit_vector is array (Natural range <>) of Bit;`
  - `Boolean is (false, true);`
  - `Character`
  - `Integer`
  - `Natural is Integer range 0 to Integer'high`
  - `Positive is Integer range 1 to Integer'high`
  - `Real`
  - `String is array (Positive range <>) of Character`
  - `Time`

```
units
fs; -- femtoseconde
ps = 1000 fs; -- picoseconde
ns = 1000 ps; -- nanoseconde
us = 1000 ns; -- microseconde
ms = 1000 us; -- milliseconde
sec = 1000 ms; -- seconde
min = 60 sec; -- minute
hr = 60 min; -- heure
end units;
```

Les différents Type VHDL

- Les types ajoutés par ieee (paquetage 1164)
  - ```
    std_ulogic is (
    'U', -- Non Initialisé - Non synthétisé
    'X', -- Forçage Indéterminé - Non synthétisé
    '0', -- Forçage à 0 - Synthèse = 0
    '1', -- Forçage à 1 - Synthèse = 1
    'Z', -- Haute Impedance - Synthèse = Z
    'W', -- Faible Indéterminé - Non synthétisé
    'L', -- Faible à 0 - Synthèse = 0
    'H', -- Faible à 1 - Synthèse = 1
    '-' -- Peu Importe - Non synthétisé
    ```
  - ```
    std_ulogic_vector is array ( natural range <> ) of
    std_ulogic;
    ```

Std_logic type résolu de std_ulogic

	'U'	'X'	'0'	'1'	'Z'	'W'	'L'	'H'	'-'
'U'	'U'	'U'	'U'	'U'	'U'	'U'	'U'	'U'	'U'
'X'	'U'	'X'	'X'	'X'	'X'	'X'	'X'	'X'	'X'
'0'	'U'	'X'	'0'	'X'	'0'	'0'	'0'	'0'	'X'
'1'	'U'	'X'	'X'	'1'	'1'	'1'	'1'	'1'	'X'
'Z'	'U'	'X'	'0'	'1'	'Z'	'W'	'L'	'H'	'X'
'W'	'U'	'X'	'0'	'1'	'W'	'W'	'W'	'W'	'X'
'L'	'U'	'X'	'0'	'1'	'L'	'W'	'L'	'W'	'X'
'H'	'U'	'X'	'0'	'1'	'H'	'W'	'W'	'H'	'X'
'-'	'U'	'X'	'X'	'X'	'X'	'X'	'X'	'X'	'X'

Plan

1 Introduction

2 Outils de conception : VHDL
- Différents types de description
- Les types en VHDL
- Signaux et Variables en VHDL
- Les tableaux
- Générique
- Clause Wait
- Test Bench
- Simulation
- Paquetage, Procédure et Fonction

3 Méthodes : Machines à états

4 Les mémoires

1 Introduction
2 Outils de conception : VHDL
- Différents types de description
- Les types en VHDL
- Signaux et Variables en VHDL
- Les tableaux
- Générique
- Clause Wait
- Test Bench
- Simulation
- Paquetage, Procédure et Fonction
3 Méthodes : Machines à états
- Les Machines à Etats en VHDL
4 Les mémoires
5 FPGA
6 Le Port Jtag
7 Conception d'un système programmable

Les variables

- Les variables sont utilisées dans les instructions séquentielles
- Elles ne correspondent à aucune réalité électronique
- Elles jouent le rôle des variables dans un langage de programmation
- Elles ont une portée limitée au processus dans lequel elles sont déclarées
- Elles sont typées

Les variables

- Affectation
  - Symbole d'affectation de variables $:=$
  - L'affectation d'une variable est instantanée
- Utilisation
  - Affectation d'un signal à une variable
  - Exécution d'un algorithme séquentiel
  - Affectation d'une variable à un signal
  - Autre utilisation : identificateur de boucle

Les signaux

- Les signaux représentent les données physiques échangées entre les modules (anglais data signal)
- Chaque signal sera matérialisé dans le circuit final par une équipotentielle
- Exemples :
  - ports d'entrées et de sorties d'une entité
  - signaux internes à une architecture

Les signaux

- Instruction d'affectation de signaux :
  - s $<=$ d after delay ;
  - s est le signal
  - d est le driver
  - delay est le délai

Les signaux

- Chaque signal a un type, comme dans un langage structuré
- Le type définit l'ensemble des valeurs que peut prendre le signal.
- Les signaux de type synthétisable sont synthétisables.
- VHDL est fortement typé, pas de conversion automatique
  - Nécessité d'avoir recours à des fonctions de conversion

Les signaux

- Types énumérés
  - type boolean is (false, true) ;
  - type character is (liste_des_caractères)
    - Tous les caractères ISO 8 bits
    - Une constante caractère est notée 'A'
  - type bit is ('0', '1') ;
  - type severity_level is (note, warning, error, failure) ;

Les signaux

- Entiers :
  - type integer is range -2147483 to 2147482 ;
  - subtype positive is integer range 1 to integer'high ;
  - subtype natural is integer range 0 to integer'high ;
- Remarques sur entiers :
  - Valeurs négatives codées en complément à 2
  - Entiers synthétisés comme bus 32 bits par défaut
- Intervalles :
  - Subtype byte is integer range -128 to 127 – codé 8 bits C2
- Réels :
  - type real is range $- to $+
  - Réels pas synthétisables

Les signaux

```
library ieee;
use ieee.std_logic_1164.all;

entity aff is
port( a : in std_logic;
      d : out std_logic_vector(3 downto 0));
end entity aff;

architecture flot of aff is
signal aint : std_logic;
begin
  aint <= '1';
  a<=aint; -- Erreur On ne peut pas ecrire sur une entree
  d<="0001";
end architecture flot;
```

Les opérateurs Logiques

- Les opérateurs logiques :

and	et bit à bit
or	ou bit à bit
xor	ou-exclusif bit à bit
not	non bit à bit

- Les opérateurs arithmétiques

+	Addition	
-	Soustraction	
*	Multiplication	
/	Division	Peu Synthétisable
mod	Modulo	Pas Synthétisable
exp	Exponentitation	Pas synthétisable

Plan

1. Introduction

2. Outils de conception : VHDL
   - Différents types de description
   - Les types en VHDL
   - Signaux et Variables en VHDL
   - Les tableaux
   - Générique
   - Clause Wait
   - Test Bench
   - Simulation
   - Paquetage, Procédure et Fonction

3. Méthodes : Machines à états

4. Les mémoires

1. Introduction
2. Outils de conception : VHDL
   - Différents types de description
   - Les types en VHDL
   - Signaux et Variables en VHDL
   - **Les tableaux**
   - Générique
   - Clause Wait
   - Test Bench
   - Simulation
   - Paquetage, Procédure et Fonction
3. Méthodes : Machines à états
   - Les Machines à Etats en VHDL
4. Les mémoires
5. FPGA
6. Le Port Jtag
7. Conception d'un système programmable

Les Tableaux

- Déclaration
  - type nom is array (intervalle) of type_de_base
- Utilisation
  - Modélisation de bus.
  - Modélisation de mémoires.
- Synthèse
  - Tableaux à 1 dimension
  - Indices entiers ou sous-type entiers
  - Eléments du tableau synthétisables

Les Tableaux

- Types tableaux prédéfinis
  - type bit_vector is array (natural range <>) of bit ;
  - type string is array (positive range <>) of character ;
  - type std_ulogic_vector is array (natural range <>) of std_ulogic ;
  - type std_logic_vector is array (natural range <>) of std_logic ;
- Exemples de modélisation de bus
  - Subtype byte is std_logic_vector (7 downto 0) ;
  - Subtype word is std_logic_vector (15 downto 0) ;

Plan

1. Introduction

2. Outils de conception : VHDL
   - Différents types de description
   - Les types en VHDL
   - Signaux et Variables en VHDL
   - Les tableaux
   - Générique
   - Clause Wait
   - Test Bench
   - Simulation
   - Paquetage, Procédure et Fonction

3. Méthodes : Machines à états

4. Les mémoires

1. Introduction
2. Outils de conception : VHDL
   - Différents types de description
   - Les types en VHDL
   - Signaux et Variables en VHDL
   - Les tableaux
   - **Générique**
   - Clause Wait
   - Test Bench
   - Simulation
   - Paquetage, Procédure et Fonction
3. Méthodes : Machines à états
   - Les Machines à Etats en VHDL
4. Les mémoires
5. FPGA
6. Le Port Jtag
7. Conception d'un système programmable

Generic

- Les paramètres génériques sont définis dans l'entité, avant les ports d'entrées-sorties
- Ils peuvent être utilisés :
  - Dans l'entité après leur déclaration
  - Dans le corps de toute architecture associée à l'entité

```
entity MON-ET is
generic (tp: time := 2ns);
port( A : in std_logic;
      B : in std_logic;
      S : out std_logic);
end entity MON-ET;
```

Generic

- La valeur du paramètre générique est précisée lors de l'instanciation de l'entité
- Des instances différentes peuvent utiliser des valeurs différentes

```
entity doubleor is
 port (in1, in2 : in std_logic;
       out2 : out std_logic);
end entity doubleor;

architecture struct of doubleor is
signal out1 : std_logic;
begin
Gate1 : entity work.or2(behaviour)
        Generic map (2 ns)
        Port map (in1, in2, out1) ;
Gate2 : entity work.or2(behaviour)
        Generic map (T_pd => 3 ns)
        Port map (a => out1, b => in2, y => out2) ;
end architecture struct;
```

Generic

- Une valeur par défaut du paramètre générique peut être indiquée lors de sa déclaration
- Lors de l'instanciation, cette valeur peut être :
- Utilisée telle quelle
- Remplacée par une autre valeur

```
entity dff is
Generic (
 T_pd, T_su : time := 2 ns ; T_h : time := 0 ns) ;
Port (clock, d : in std_logic ;
      q : out std_logic)
end entity dff;

architecture struct of reg is
.......
Request_dff : entity work.dff (behaviour)
              Generic map (4 ns, 3 ns, open)
              Port map (system_clock, request, prending_request)
.......
end architecture struct;
```

Generate

- L'instruction concurrente generate permet de dupliquer des instructions concurrentes de manière itérative ou conditionnelle.

Generate

```
entity shiftreg is
  generic (nbits: positive := 8);
  port (clk, rst, d: in std_logic;
        q: out std_logic);
end entity shiftreg;

architecture structure of shiftreg is
 component dff is
      port (clk, rst, d: in std_logic;
            q: out std_logic);
 end component dff;

signal qint: std_logic _vector(1 to nbits-1);

Begin

  cell_array: for i in 1 to nbits generate
    first_cell: if i = 1 generate
     dff1: dff port map (clk, rst, d, qint(1));
    end generate first_cell;

    int_cell: if i > 1 and i < nbits generate
     dffi: dff port map (clk, rst, qint(i-1), qint(i));
    end generate int_cell;

    last_cell: if i = nbits generate
     dffn: dff port map (clk, rst, qint(nbits-1), q);
    end generate last_cell;
  end generate cell_array;
end architecture structure;
```

Plan

1 Introduction

2 Outils de conception : VHDL
- Différents types de description
- Les types en VHDL
- Signaux et Variables en VHDL
- Les tableaux
- Générique
- Clause Wait
- Test Bench
- Simulation
- Paquetage, Procédure et Fonction

3 Méthodes : Machines à états

4 Les mémoires

La clause Wait

- `wait on liste_de_signaux`
- un évènement sur l'un des signaux de la `liste_de_signaux` provoque l'exécution du processus

```
half_adder : process is
begin
 sum <= a xor b  after T_pd ;
 carry <= a and b after T_pd ;
 wait on a, b ;
end process half_adder ;
```

```
half_adder : process(a, b) is
 begin
  sum <= a xor b  after T_pd ;
  carry <= a and b after T_pd ;
 end process half_adder ;
```

La clause Wait

- `wait until condition`
- condition est une condition booléenne
- Ce qui se passe :
  - Le processus est suspendu lorsqu'il arrive à l'instruction wait
  - Le processus est relancé lorsque condition est testée et vraie
- Liste de sensibilité de wait until
  - condition est testée lorsqu'un évènement se produit sur l'un des signaux qui y apparaissent
  - la liste de sensibilité est la liste des signaux qui apparaissent dans condition
- wait until condition
  - équivaut à : `wait on liste_des_signaux_de_condition until condition`

La clause Wait

- `wait for duree`
- Ce qui se passe :
  - Le processus est suspendu lorsqu'il arrive à l'instruction wait
  - Le processus est relancé après une temporisation duree

```
clock_gen : process  is
begin
  clock <= '1' after T_pw, '0' after 2* T_pw ;
  wait for 2* T_pw ;
end process clock_gen ;
```

  - Pas synthétisable

La clause Wait

- `Wait`
- Ce qui se passe :
  - Le processus est suspendu lorsqu'il arrive à l'instruction wait
  - Il reste suspendu jusqu'à la fin de la simulation
- Exemple d'application : test bench
- VHDL permet de décrire dans le même langage :
  - Le circuit à tester
  - La génération des signaux d'entrée.
  - La vérification des signaux de sortie.

Plan

1. Introduction

2. Outils de conception : VHDL
   - Différents types de description
   - Les types en VHDL
   - Signaux et Variables en VHDL
   - Les tableaux
   - Générique
   - Clause Wait
   - Test Bench
   - Simulation
   - Paquetage, Procédure et Fonction

3. Méthodes : Machines à états

4. Les mémoires

Test Bench

- Génération de Stimuli
- Analyse des résultats
- Utilisation de clause `Assert`

Assert

- `ASSERT condition`
  `REPORT string SEVERITY severity_level;`

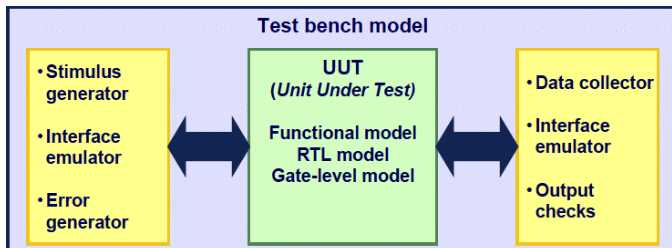
```
check_setup: PROCESS (clk, d)
BEGIN
      IF (clk'EVENT AND clk='1') THEN -- test si front mon
            ASSERT d'STABLE(setup_time) -- regarde si d es
                  REPORT "Setup Violation..." -- affiche u
                  SEVERITY WARNING;
      END IF;
END PROCESS check_setup;
```

Assert

- Niveaux de sévérité :
  - Note : utilisé pour information seulement
    "Note : Chargement de données d'un fichier"
  - Warning : utilisé pour fournir une information sur une erreur en instance
    "Warning : Détection d'un pic"
  - Error : utilisé pour information seulement
    "Error : Violation du temps d'initialisation"
  - Failure : raporte une grosse erreur
    "Failure : Ligne RESET instable"

TestBench

Test bench model




```
entity tb_xxx is
end entity tb_xxx;

architecture bench of tb_xxx is
   ...
begin
   UUT: entity work.E(A) port map (...);
   stimulus: process begin
      ...
   end process stimulus;
   verification: process begin
      ...
   end process verification;
end architecture bench;
```

TestBench

Test bench for a 1-bit adder (1/5)

```
entity tb_add1 is
end entity tb_add1;

architecture bench1 of tb_add1 is
    signal op1, op2, ci, sum, co: bit;
begin
    UUT: entity work.add1(dfl)
        generic map (TP => 1.2 ns)
        port map (opa => op1, opb => op2,
                  cin  => ci, sum => sum,
                  cout => co);
    Stimulus_check: process
        procedure check (
            op1, op2, ci, co, sum: in bit;
            exp_co, exp_sum: in bit) is
        begin
            assert co = exp_co and sum = exp_sum
                report "Error for (op1, op2, ci) = (" &
                    bit'image(op1) & "," & bit'image(op2) &
                    "," & bit'image(ci) & ")" & LF &
                    "(co, sum) = (" & bit'image(co) & "," &
                     bit'image(sum) & ") / expected: (" &
                     bit'image(exp_co) & "," &
                     bit'image(exp_sum) &")"
                severity error;
        end procedure check;
        ...
```

```
        ...
    begin
        op1 <= '0'; op2 <= '0'; ci <= '0';
        wait for 5 ns;
        check(op1, op2, ci, co, sum, '0', '0');

        op1 <= '0'; op2 <= '0'; ci <= '1';
        wait for 5 ns;
        check(op1, op2, ci, co, sum, '0', '1');
        ...
        op1 <= '1'; op2 <= '1'; ci <= '1';
        wait for 5 ns;
        check(op1, op2, ci, co, sum, '1', '1');

        wait;  -- wait forever
    end process Stimulus_check;

end architecture bench1;
```

```
# ** Error: Error for (op1, op2, ci) = ('0','0','1')
# (co, sum) = ('1','1') / expected: ('0','1')
#    Time: 10 ns  Iteration: 0  Instance: :tb_add1
```

TestBench

Test bench for a 1-bit adder (2/5)

```
architecture bench2 of tb_add1 is
    signal op1, op2, ci: bit;
    signal sum_dfl, sum_str, co_dfl, co_str: bit;
begin

    UUT: entity work.add1(str)
        generic map (TP => 1.2 ns)
        port map (
            opa => op1, opb => op2, cin  => ci,
            sum => sum_str; cout => co_str);

    UREF: entity work.add1(dfl)
        generic map (TP => 1.2 ns)
        port map (
            opa => op1, opb => op2, cin  => ci,
            sum => sum_dfl; cout => co_dfl);
    ...
```

```
    ...
    Stimulus_check: process
        procedure check (
            op1, op2, ci, co, sum: in bit;
            exp_co, exp_sum: in bit) is
        begin
            ...
        end procedure check;

    begin
        op1 <= '0'; op2 <= '0'; ci <= '0';
        wait for 5 ns;
        check(op1, op2, ci, co_str, sum_str, co_dfl, sum_dfl);

        op1 <= '0'; op2 <= '0'; ci <= '1';
        wait for 5 ns;
        check(op1, op2, ci, co_str, sum_str, co_dfl, sum_dfl);
        ...
        op1 <= '1'; op2 <= '1'; ci <= '1';
        wait for 5 ns;
        check(op1, op2, ci, co_str, sum_str, co_dfl, sum_dfl);

        wait;  -- wait forever
    end process Stimulus_check;

end architecture bench2;
```

TestBench

Test bench for a 1-bit adder (3/5)

```
architecture bench3 of tb_add1 is
    signal op1, op2, ci, sum, co: bit;
begin
    UUT: entity work.add1(dfl)
        generic map (TP => 1.2 ns)
        port map (opa => op1, opb => op2,
                  cin  => ci, sum => sum,
                  cout => co);
    Stimulus_check: process
        type table_elem is record
            x, y, ci, co, s: bit;
        end record;
        type table is array (0 to 7) of table_elem;
        constant TT: table :=
            (-- x -- y -- ci ------ co -- s --
              ('0', '0', '0',       '0',  '0'),
              ('0', '0', '1',       '0',  '1'),
              ('0', '1', '0',       '0',  '1'),
              ('0', '1', '1',       '1',  '0'),
              ('1', '0', '0',       '0',  '1'),
              ('1', '0', '1',       '1',  '0'),
              ('1', '1', '0',       '1',  '0'),
              ('1', '1', '1',       '1',  '1'));
        ...
```

```
        ...
    begin
        for i in TT'range loop
            op1 <= TT(i).x; op2 <= TT(i).y; ci <= TT(i).ci;
            wait for 5 ns;

            assert co = TT(i).co and sum = TT(i).s
                report "Error for (op1, op2, ci) = (" &
                       bit'image(op1) & "," & bit'image(op2) & "," &
                       bit'image(ci) & ")" & LF &
                       "(co, sum) = (" & bit'image(co) & "," &
                       bit'image(sum) & ") / expected: (" &
                       bit'image(TT(i).co) & "," &
                       bit'image(TT(i).s) &")"
                severity error;

        end loop;
        wait;  -- wait forever
    end process Stimulus_check;

end architecture bench3;
```

```
# ** Error: Error for (op1, op2, ci) = ('0','1','1')
# (co, sum) = ('1','1') / expected: ('1','0')
#    Time: 20 ns  Iteration: 0  Instance: :tb_add1
```

TestBench

Test bench for a 1-bit adder (4/5)

```
use STD.textio.all;
architecture bench4 of tb_add1 is
    file add1_tt: text open read_mode is "add1_tt.dat";
    signal op1, op2, ci, sum, co: bit;
begin
    UUT: entity work.add1(dfl)
        generic map (TP => 1.2 ns)
        port map (opa => op1, opb => op2,
                  cin  => ci, sum => sum,
                  cout => co);
    Stimulus_check: process
        procedure check (
            op1, op2, ci, co, sum: in bit;
            exp_co, exp_sum: in bit) is
        begin
            assert co = exp_co and sum = exp_sum
                report "Error for (op1, op2, ci) = (" &
                    bit'image(op1) & "," & bit'image(op2) &
                    "," & bit'image(ci) & ")" & LF &
                    "(co, sum) = (" & bit'image(co) & "," &
                    bit'image(sum) & ") / expected: (" &
                    bit'image(exp_co) & "," &
                    bit'image(exp_sum) &")"
                severity error;
        end procedure check;
        ...
```

```
# op1 op2 ci co sum
00000
00101
01001
01110
10001
10110
11010
11111
```

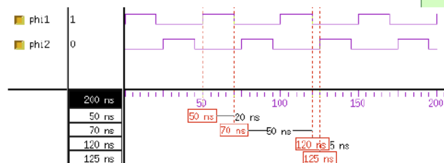
```
        ...
        variable tt: bit_vector(1 to 5);
        variable exp_co, exp_sum: bit;
        variable ll: line;
    begin
        readline(add1_tt, ll);  -- en-tête
        while not endfile(add1_tt) loop
            readline(add1_tt, ll);
            read(ll, tt);
            op1 <= tt(1); op2 <= tt(2); ci <= tt(3);
            exp_co := tt(4); exp_sum := tt(5);
            wait for 5 ns;
            check(op1, op2, ci, co, sum, exp_co, exp_sum);
        end loop;
        wait;  -- wait forever
    end process Stimulus_check;

end architecture bench4;
```

TestBench

Clock generation

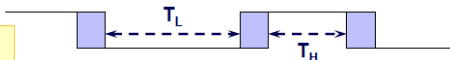
```
library ieee;
    use ieee.std_logic_1164.all;
architecture bench of tb_xxx is
    constant CLK_PER: time := 20 ns;
    signal clk: std_logic := '0';
begin
    UUT: ...
    clk <= not clk after CLK_PER/2;
    Stimulus_check: process
        ...
    end process Stimulus_check
end architecture bench;
```

```
library ieee;
    use ieee.std_logic_1164.all;
architecture bench of tb_xxx is
    signal phi1, phi2: std_logic := '0';
    procedure clkgen (
        signal clk: out bit;
        constant Tperiod, Tpulse, Tphase: in time) is
    begin
        wait for Tphase;
        loop
            clk <= '1', '0' after Tpulse;
            wait for Tperiod;
        end loop;
    end procedure clkgen;
    ...
begin
    UUT: ...
    gen_phi1: clkgen(phi1, Tperiod => 50 ns,
                           Tpulse => 20 ns,
                           Tphase => 0 ns);
    gen_phi2: clkgen(phi2, Tperiod => 50 ns,
                           Tpulse => 20 ns,
                           Tphase => 25 ns);
    ...
end architecture bench;
```



TestBench

Waveform generation (1/2)

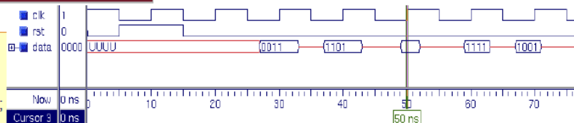



```
library ieee;
   use ieee.math_real.all;

architecture bench of tb_xxx is

   constant PC_MIN: real : = 0.3;    -- % min. value ('0')
   constant PC_MAX: real := 0.3;     -- % max. value ('1')

   constant TL_MIN : time := 5 ns;
   constant TL_MAX: time := 7 ns;

   constant TH_MIN : time := 3 ns;
   constant TH_MAX: time := 5 ns;

   signal S: bit := '0';
begin
   process
      variable seed1: positive := 3812;
      variable seed2: positive := 915;
      ...
```

```
      ...
      impure function random return real is
         variable rnd: real;
      begin
         uniform(seed1, seed2, rnd);
         if rnd < PC_MIN then
            return 0.0;
         elsif rnd < PC_MIN + PC_MAX then
            return 1.0;
         else
            return rnd;
         end if;
      end function random;
   begin
      S <= '0';
      wait for TL_MIN + (TL_MAX - TL_MIN)*random;
      S <= '1';
      wait for TH_MIN + (TH_MAX - TH_MIN)*random;
   end process;
   ...
end architecture bench;
```

TestBench

Waveform generation (2/2)




```
library ieee;
   use ieee.std_logic_1164.all;
architecture bench of tb_proc is
   constant CLK_PER: time := 10 ns;
   constant TSETUP : time := 3 ns;
   constant THOLD  : time := 3 ns;
   signal clk: std_logic := '0';
   signal rst: std_logic;
   signal data: std_logic_vector(3 downto 0);

   procedure do_sync_reset (
      signal clk: in   std_logic;
      signal rst: out std_logic) is
   begin
      rst <= '0'; wait until clk = '0';
      rst <= '1'; wait until clk = '0';
      rst <= '0';
   end procedure do_sync_reset;

   procedure apply_vector (
      constant vector: in   std_logic_vector(3 downto 0);
      signal     data  : out std_logic_vector(3 downto 0);
      constant PER  : in  time := CLK_PER;
      constant TS     : in  time := TSETUP;
      constant TH     : in  time := THOLD) is
   ...
```

```
   ...
   begin
      wait until clk = '0';
      wait for CLK_PER/2 - TS;
      data <= vector;
      wait until clk = '1';
      wait for TH;
      data <= (others => 'X');
   end procedure apply_vector;

begin
   CLKGEN: clk <= not clk after CLK_PER/2;
   DATAGEN: process begin
      do_sync_reset(clk,rst);
      apply_vector("0011", data);
      apply_vector("1101", data);
      apply_vector("0000", data, 1 ns, 2 ns);
      apply_vector("1111", data, 1 ns);
      apply_vector("1001", data, TH => 1 ns)
      wait;
   end process DATAGEN;
end architecture bench;
```

Plan

1 Introduction

2 Outils de conception : VHDL
- Différents types de description
- Les types en VHDL
- Signaux et Variables en VHDL
- Les tableaux
- Générique
- Clause Wait
- Test Bench
- Simulation
- Paquetage, Procédure et Fonction

3 Méthodes : Machines à états

4 Les mémoires

Simulation Evénementielle

Initialisation

1. Assign initial values to signals and variables.
2. $T_c = 0ns$, $\delta = 0$
3. Execute all processes until they suspend.
4. Determine next time T_n according to 4.

Cycle

1. Tc = Tn.
2. Update signals.
3. Execute all processes sensitive to updated signals.
4. Determine next time Tn:
   - if pending transactions at current time: $\delta = \delta + 1$ then 2
   - if no more pending transactions or Tn = time'high then STOP
   - else T_n = time of next earliest pending transaction, $\delta = 0$
5. Execute postponed processes.
6. goto 1.

Plan

1 Introduction

2 Outils de conception : VHDL
- Différents types de description
- Les types en VHDL
- Signaux et Variables en VHDL
- Les tableaux
- Générique
- Clause Wait
- Test Bench
- Simulation
- Paquetage, Procédure et Fonction

3 Méthodes : Machines à états

4 Les mémoires

Paquetage

```
package proc_pkg is
   subtype data is integer range 0 to 3;
   type darray is array (1 to 3) of data;
end package proc_pkg;

use work.proc_pkg.all;
entity procstmt is
 port (
       inar : in darray;
       outar: out darray);
end entity procstmt;
```

Procédure

```
architecture a of procstmt is
begin
  process (inar)
    procedure swap (d: inout darray; l, h: in postitive) is
       variable tmp: data;
       begin
         if d(l) > d(h) then
               tmp := d(l);
               d(l) := d(h);
               d(h) := tmp;
         end if;
       end swap;
    variable tmpar: darray;
    begin
     tmpar := inar;
      swap(tmpar,1,2);
      swap(tmpar,2,3);
      swap(tmpar,1,2);
      outar <= tmpar;
    end process;
 end architecture a;
```

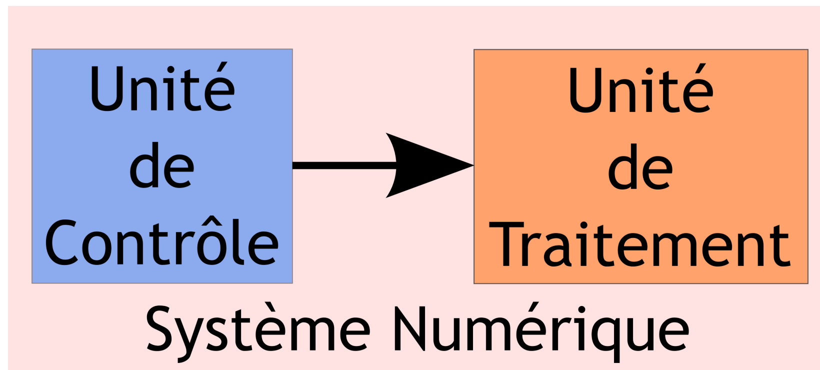
Fonction

```
entity parity_check is
 generic (NBITS: positive := 8);
 port (
       data: in bit_vector(NBITS-1 downto 0);
       prty: out bit);
end entity parity_check;

architecture func of parity_check is
 begin
    process (data)
      function parity (bv: bit_vector) return bit is
        variable result: bit;
        begin
          result := '0';
          for i in bv'range loop
              result := result xor bv(i); -- odd parity
          end loop;
          return result;
        end function parity;
     begin
        prty <= parity(data);
     end process;
  end architecture func;
```

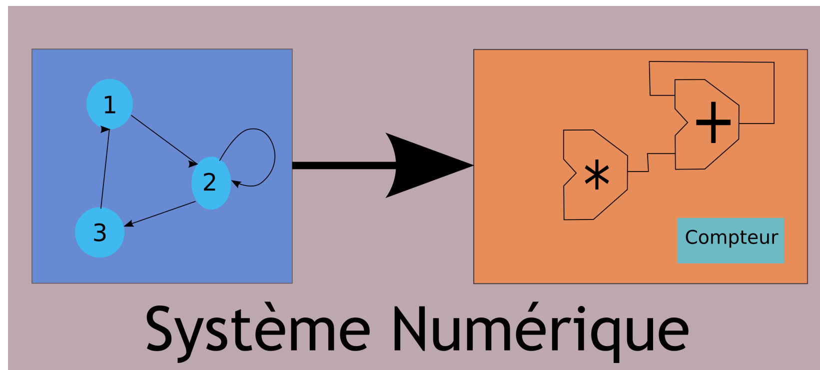
Fonction de conversion

	numeric_std	std_logic_arith
Type Conversion		
std_logic_vector -> unsigned	unsigned(*arg*)	unsigned(*arg*)
std_logic_vector -> signed	signed(*arg*)	signed(*arg*)
unsigned -> std_logic_vector	std_logic_vector(*arg*)	std_logic_vector(*arg*)
signed -> std_logic_vector	std_logic_vector(*arg*)	std_logic_vector(*arg*)
integer -> unsigned	to_unsigned(*arg*,*size*)	conv_unsigned(*arg*,*size*)
integer -> signed	to_signed(*arg*,*size*)	conv_signed(*arg*,*size*)
unsigned -> integer	to_integer(*arg*)	conv_integer(*arg*)
signed -> integer	to_integer(*arg*)	conv_integer(*arg*)
integer -> std_logic_vector	integer -> unsigned/signed ->std_logic_vector	
std_logic_vector -> integer	std_logic_vector -> unsigned/signed ->integer	
unsigned + unsigned -> std_logic_vector	std_logic_vector(*arg1* + *arg2*)	*arg1* + *arg2*
signed + signed -> std_logic_vector	std_logic_vector(*arg1* + *arg2*)	*arg1* + *arg2*
Resizing		
unsigned	resize(*arg*,*size*)	conv_unsigned(*arg*,*size*)
signed	resize(*arg*,*size*)	conv_signed(*arg*,*size*)

Système Numérique



Système Numérique



Définition des systèmes électroniques

- Un système électronique est caractérisé par

Définition des systèmes électroniques

- Un système électronique est caractérisé par
  - Ses entrées : $e_0, \ldots, e_i, \ldots, e_{n-1}$

Définition des systèmes électroniques

- Un système électronique est caractérisé par
  - Ses entrées : $e_0, \ldots, e_i, \ldots, e_{n-1}$
  - Son état électrique E

Définition des systèmes électroniques

- Un système électronique est caractérisé par
  - Ses entrées : $e_0, \ldots, e_i, \ldots, e_{n-1}$
  - Son état électrique E
  - Ses sorties : $s_0, \ldots, s_i, \ldots, s_{n-1}$

Définition des systèmes électroniques

- Un système électronique est caractérisé par
  - Ses entrées : $e_0, \ldots, e_i, \ldots, e_{n-1}$
  - Son état électrique E
  - Ses sorties : $s_0, \ldots, s_i, \ldots, s_{n-1}$
- Il existe deux type de systèmes électroniques
  - Les systèmes combinatoires construits à l'aide de logique combinatoire

Définition des systèmes électroniques

- Un système électronique est caractérisé par
  - Ses entrées : $e_0, \ldots, e_i, \ldots, e_{n-1}$
  - Son état électrique E
  - Ses sorties : $s_0, \ldots, s_i, \ldots, s_{n-1}$
- Il existe deux type de systèmes électroniques
  - Les systèmes combinatoires construits à l'aide de logique combinatoire
  - Les systèmes séquentiels construits à l'aide de logique séquentielle

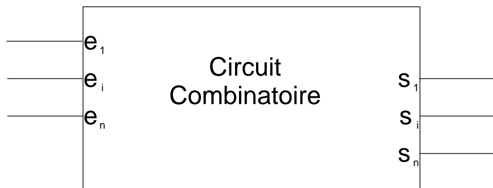
Définition des systèmes électroniques

- Un système électronique est caractérisé par
  - Ses entrées : $e_0, \ldots, e_i, \ldots, e_{n-1}$
  - Son état électrique E
  - Ses sorties : $s_0, \ldots, s_i, \ldots, s_{n-1}$
- Il existe deux type de systèmes électroniques
  - Les systèmes combinatoires construits à l'aide de logique combinatoire
  - Les systèmes séquentiels construits à l'aide de logique séquentielle

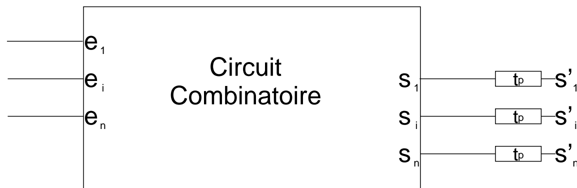
Logique Combinatoire

Définition :
Un circuit électronique est dit combinatoire si ses sorties sont déterminées par la combinaison de ses variables d'entrées et ceci après un temps fini. L'état d'un système est donc défini par la combinaison des variables $e_0, \ldots, e_i, \ldots, e_{n-1}$.

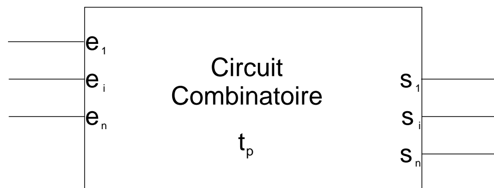
Logique Combinatoire



Logique Combinatoire



Logique Combinatoire



Logique Séquentielle

Définition :
Un circuit électronique est dit séquentiel si ses sorties sont déterminées non seulement par la combinaison de ses variables d'entrées, mais aussi par la séquence des combinaisons précédentes de ses entrées et par l'état initial du système.

Logique Séquentielle

Il apparaît dès lors que :

- Une même combinaison de $e_0, \ldots, e_i, \ldots, e_{n-1}$ peut engendrer différents états du système.

Logique Séquentielle

Il apparaît dès lors que :

- Une même combinaison de $e_0, \ldots, e_i, \ldots, e_{n-1}$ peut engendrer différents états du système.
- Un système séquentiel est un système combinatoire de $e_0, \ldots, e_i, \ldots, e_{n-1}$ et de $y_0, \ldots, y_i, \ldots, y_{n-1}$

Logique Séquentielle

Il apparaît dès lors que :

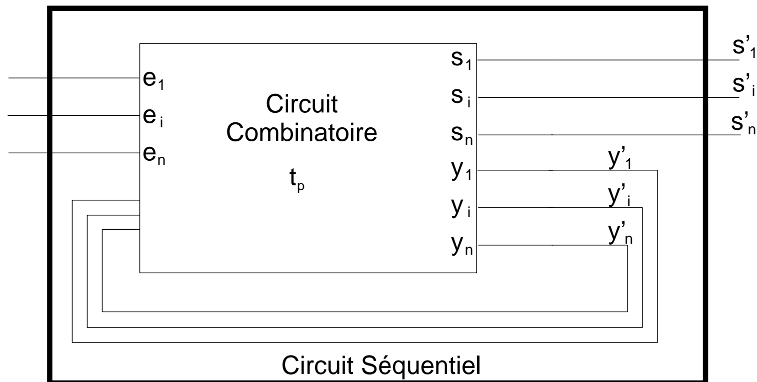
- Une même combinaison de $e_0, \ldots, e_i, \ldots, e_{n-1}$ peut engendrer différents états du système.
- Un système séquentiel est un système combinatoire de $e_0, \ldots, e_i, \ldots, e_{n-1}$ et de $y_0, \ldots, y_i, \ldots, y_{n-1}$
- $y_0, \ldots, y_i, \ldots, y_{n-1}$ sont des variables internes indiquant l'état présent du système

Logique Séquentielle

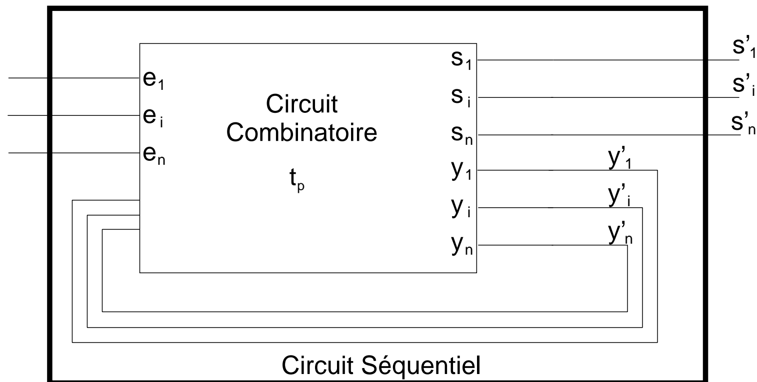
Il apparaît dès lors que :

- Une même combinaison de $e_0, \ldots, e_i, \ldots, e_{n-1}$ peut engendrer différents états du système.
- Un système séquentiel est un système combinatoire de $e_0, \ldots, e_i, \ldots, e_{n-1}$ et de $y_0, \ldots, y_i, \ldots, y_{n-1}$
- $y_0, \ldots, y_i, \ldots, y_{n-1}$ sont des variables internes indiquant l'état présent du système
- Les entrées $e_0, \ldots, e_i, \ldots, e_{n-1}$ et les variables internes $y_0, \ldots, y_i, \ldots, y_{n-1}$ préparent l'état futur du système.

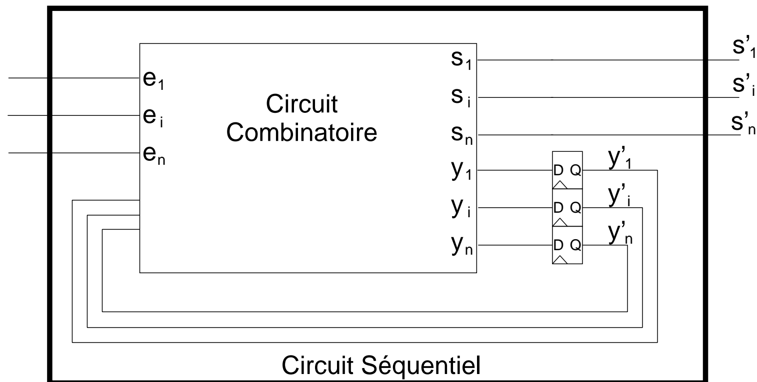
Logique Séquentielle



Logique Séquentielle - Système Séquentiel Asynchrone



Logique Séquentielle - Système Séquentiel Synchrone



Machines à états

- Les combinaisons des entrées conduisent à un nombre fini de combinaisons de sortie.

Machines à états

- Les combinaisons des entrées conduisent à un nombre fini de combinaisons de sortie.
- D'où l'appelation machine à nombre d'états finis ou Machine A Etats.

Machines à états

- Les combinaisons des entrées conduisent à un nombre fini de combinaisons de sortie.
- D'où l'appelation machine à nombre d'états finis ou Machine A Etats.
- Vecteur d'entrée : Combinaison des variables d'entrée

Machines à états

- Les combinaisons des entrées conduisent à un nombre fini de combinaisons de sortie.
- D'où l'appelation machine à nombre d'états finis ou Machine A Etats.
- Vecteur d'entrée : Combinaison des variables d'entrée
- Vecteur de sortie : Combinaison des variables de sortie

Machines à états

- Notations :

Machines à états

- Notations :
  - Entrée : E

Machines à états

- Notations :
  - Entrée : E
  - Etat Présent : EP

Machines à états

- Notations :
  - Entrée : E
  - Etat Présent : EP
  - Etat Futur : EF

Machines à états

- Notations :
  - Entrée : E
  - Etat Présent : EP
  - Etat Futur : EF
  - Sortie : S

Machines à états

- Notations :
  - Entrée : E
  - Etat Présent : EP
  - Etat Futur : EF
  - Sortie : S
- Definitions

Machines à états

- Notations :
  - Entrée : E
  - Etat Présent : EP
  - Etat Futur : EF
  - Sortie : S
- Definitions
  - Etat : Indicateur de position dans le temps

Machines à états

- Notations :
  - Entrée : E
  - Etat Présent : EP
  - Etat Futur : EF
  - Sortie : S
- Definitions
  - Etat : Indicateur de position dans le temps
  - Registre d'Etat : Composé de bascules permettant de mémoriser les valeurs des états

Machines à états

- Notations :
  - Entrée : E
  - Etat Présent : EP
  - Etat Futur : EF
  - Sortie : S
- Definitions
  - Etat : Indicateur de position dans le temps
  - Registre d'Etat : Composé de bascules permettant de mémoriser les valeurs des états
  - Etat Présent : sortie stable du registre d'état à l'instant présent

Machines à états

- Notations :
  - Entrée : E
  - Etat Présent : EP
  - Etat Futur : EF
  - Sortie : S
- Definitions
  - Etat : Indicateur de position dans le temps
  - Registre d'Etat : Composé de bascules permettant de mémoriser les valeurs des états
  - Etat Présent : sortie stable du registre d'état à l'instant présent
  - Etat Futur : état dans lequel se trouvera la machine après une impulsion

Machines à états

Table de transition : deux parties indiquant le présent et le futur.

- E : Entrée à l'instant n

Machines à états

Table de transition : deux parties indiquant le présent et le futur.

- E : Entrée à l'instant n
- EP : Etat Présent à l'instant n

Machines à états

Table de transition : deux parties indiquant le présent et le futur.

- E : Entrée à l'instant n
- EP : Etat Présent à l'instant n
- EF : Etat Suivant à l'instant $n+1$

Machines à états

Table de transition : deux parties indiquant le présent et le futur.

- E : Entrée à l'instant n
- EP : Etat Présent à l'instant n
- EF : Etat Suivant à l'instant $n+1$
- S : Fonction de Sortie à l'instant $n+1$

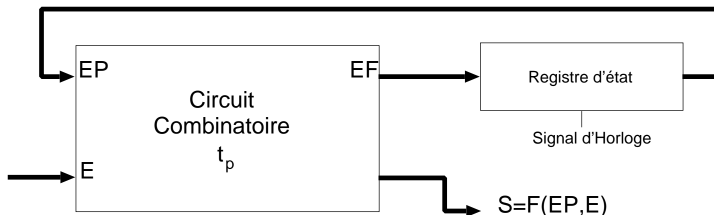
Machines à états

Table de transition : deux parties indiquant le présent et le futur.

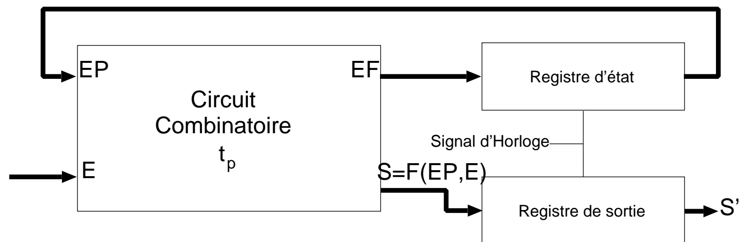
- E : Entrée à l'instant n
- EP : Etat Présent à l'instant n
- EF : Etat Suivant à l'instant $n+1$
- S : Fonction de Sortie à l'instant $n+1$
- Table de transition

E	EP	EF	S
Entrée	Etat Présent	Etat Suivant	Sortie
Avant Impulsion Horloge		Après Impulsion Horloge	

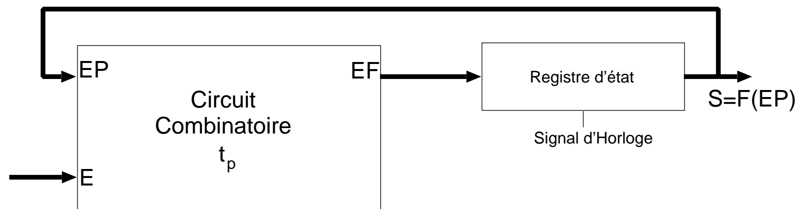
Machines de Mealy asynchrone



Machines de Mealy synchrone



Machines de Moore



Machines à Etats

- Machine de Mealy : $S = f(E, EP)$ et $EF = g(E, EP)$

Machines à Etats

- Machine de Mealy : $S = f(E, EP)$ et $EF = g(E, EP)$
- Dans un état, le vecteur sortie peut varier en fonction des valeurs du vecteur d'entrée

Machines à Etats

- Machine de Mealy : $S = f(E, EP)$ et $EF = g(E, EP)$
- Dans un état, le vecteur sortie peut varier en fonction des valeurs du vecteur d'entrée
- Machine de Moore : $S = f(EP)$ et $EF = g(E, EP)$

Machines à Etats

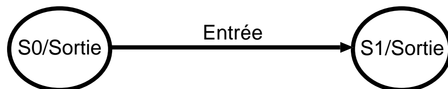
- Machine de Mealy : $S = f(E, EP)$ et $EF = g(E, EP)$
- Dans un état, le vecteur sortie peut varier en fonction des valeurs du vecteur d'entrée
- Machine de Moore : $S = f(EP)$ et $EF = g(E, EP)$
- Dans un état, le vecteur de sortie à une valeur unique

Notations Transitions

Transition

MEALY

MOORE



Réalisation des machines à états

- Deux phases principales :

Réalisation des machines à états

- Deux phases principales :
  - Phase de conception : détermine l'architecture de la machine à états

Réalisation des machines à états

- Deux phases principales :
  - Phase de conception : détermine l'architecture de la machine à états
  - Phase de synthèse : mise en oeuvre de l'architecture de la machine à état

Réalisation des machines à états

- Deux phases principales :
  - Phase de conception : détermine l'architecture de la machine à états
  - Phase de synthèse : mise en oeuvre de l'architecture de la machine à état
- Proposition d'une méthode de conception basée sur 7 étapes

Méthode de réalisation

1. Spécification du cahier des charges (Crucial).

Méthode de réalisation

1. Spécification du cahier des charges (Crucial).
2. Détermination des états.

Méthode de réalisation

1. Spécification du cahier des charges (Crucial).
2. Détermination des états.
3. Identification des entrées et des sorties de la machine à états.

Méthode de réalisation

1. Spécification du cahier des charges (Crucial).
2. Détermination des états.
3. Identification des entrées et des sorties de la machine à états.
4. Etablissement du graphe de transistion.

Méthode de réalisation

1. Spécification du cahier des charges (Crucial).
2. Détermination des états.
3. Identification des entrées et des sorties de la machine à états.
4. Etablissement du graphe de transistion.
5. Etablissement de la table de transistion.

Méthode de réalisation

1. Spécification du cahier des charges (Crucial).
2. Détermination des états.
3. Identification des entrées et des sorties de la machine à états.
4. Etablissement du graphe de transistion.
5. Etablissement de la table de transistion.
6. Détermination du nombre de bascules nécessaires.

Méthode de réalisation

1. Spécification du cahier des charges (Crucial).
2. Détermination des états.
3. Identification des entrées et des sorties de la machine à états.
4. Etablissement du graphe de transistion.
5. Etablissement de la table de transistion.
6. Détermination du nombre de bascules nécessaires.
7. Détermination des équations de l'état futur et des sorties et réalisation.

Les Etapes 1 à 4 : Applications

- Application 1

Les Etapes 1 à 4 : Applications

- Application 1
- Application 2

Les Etapes 1 à 4 : Applications

- Application 1
- Application 2
- Application 3

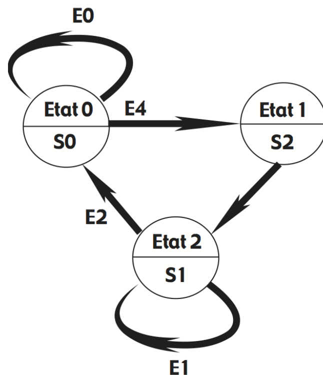
Les Etapes 5 à 7

- Etapes quasi automatiques

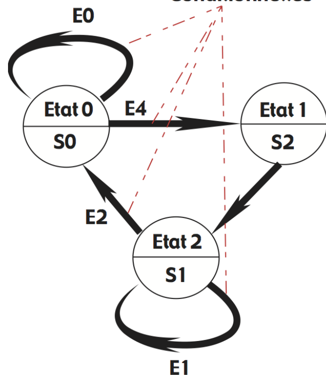
Les Etapes 5 à 7

- Etapes quasi automatiques
- Basée sur des méthodes de réalisation de systèmes électroniques standards

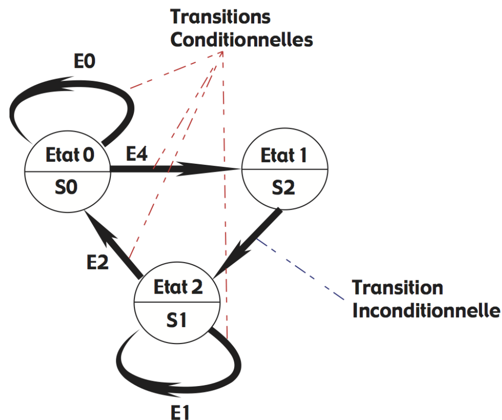
Graphe de Transition : Moore



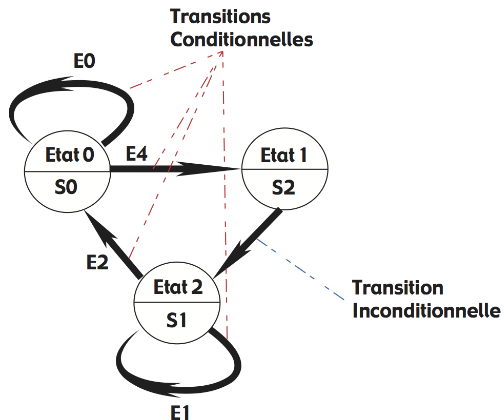
Graphe de Transition : Moore



Graphe de Transition : Moore



Graphe de Transition : Moore



Graphe de Transistion : Moore

- Il existe deux types de transitions :

Graphe de Transistion : Moore

- Il existe deux types de transitions :
  - Les transitions conditionnelles : elles ne s'effectuent qu'en fonction d'une certaine valeur du vecteur d'entrée et sur le front d'horloge. Il y a au moins deux transitions conditionnelle qui parte d'un même état

Graphe de Transistion : Moore

- Il existe deux types de transitions :
  - Les transitions conditionnelles : elles ne s'effectuent qu'en fonction d'une certaine valeur du vecteur d'entrée et sur le front d'horloge. Il y a au moins deux transitions conditionnelle qui parte d'un même état
  - Les transitions inconditionnelles : elles s'effectuent automatiquement sur le front d'horloge. Il ne peut y avoir qu'une seule transition qui parte d'un état lorsque celle-ci est inconditionnelle

Graphe de Transition : Vecteur Entrée et Sortie

- Représentation condensée des entrées et des sorties : les vecteurs d'entrée et de sortie.

Graphe de Transition : Vecteur Entrée et Sortie

- Représentation condensée des entrées et des sorties : les vecteurs d'entrée et de sortie.
  - Ici E dénote le vecteur d'entrée des entrées a,b et c. Lorsque a=0,b=0 et c=0 alors E=0 et on note E0, Lorsque a=0,b=0 et c=1 alors E=1 et on note E1 ainsi de suite jusqu' à a=1,b=1 et c=1 alors E=7 et on note E7

Graphe de Transition : Vecteur Entrée et Sortie

- Représentation condensée des entrées et des sorties : les vecteurs d'entrée et de sortie.
  - Ici E dénote le vecteur d'entrée des entrées a,b et c. Lorsque a=0,b=0 et c=0 alors E=0 et on note E0, Lorsque a=0,b=0 et c=1 alors E=1 et on note E1 ainsi de suite jusqu' à a=1,b=1 et c=1 alors E=7 et on note E7
  - Ici S dénote le vecteur de sortie des sorties x et y. Lorsque x=0 et y=0 alors S=0 et on note S0. Lorsque x=0 et y=1 alors S=1 et on note S1. Lorsque x=1 et y=0 alors S=2 et on note S2 et lorsque x=1 et y=1 S=3 et on note S3.

Graphe de Transition : Vecteur Entrée et Sortie

- Représentation condensée des entrées et des sorties : les vecteurs d'entrée et de sortie.
  - Ici E dénote le vecteur d'entrée des entrées a,b et c. Lorsque a=0,b=0 et c=0 alors E=0 et on note E0, Lorsque a=0,b=0 et c=1 alors E=1 et on note E1 ainsi de suite jusqu' à a=1,b=1 et c=1 alors E=7 et on note E7
  - Ici S dénote le vecteur de sortie des sorties x et y. Lorsque x=0 et y=0 alors S=0 et on note S0. Lorsque x=0 et y=1 alors S=1 et on note S1. Lorsque x=1 et y=0 alors S=2 et on note S2 et lorsque x=1 et y=1 S=3 et on note S3.
- Toutes les combinaisons des vecteurs ne sont pas forcément utilisées

l’horloge : entrée implicite

- l’horloge est une entrée

l'horloge : entrée implicite

- l'horloge est une entrée
- Machine à état synchrone

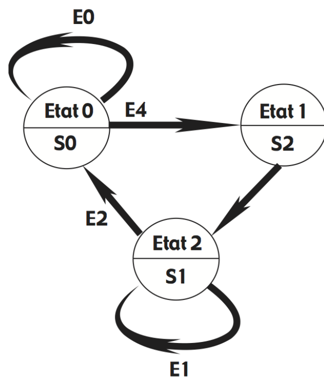
l'horloge : entrée implicite

- l'horloge est une entrée
- Machine à état synchrone
- Horloge = entrée toujours présente

l'horloge : entrée implicite

- l'horloge est une entrée
- Machine à état synchrone
- Horloge = entrée toujours présente
- Le changement d'état s'effectue sur un front d'horloge

Exemple Moore



Exemple Moore : table de transition complète

E	EP	EF
E0	Etat 0	Etat 0
E1	Etat 0	X
E2	Etat 0	X
E3	Etat 0	X
E4	Etat 0	Etat 1
E5	Etat 0	X
E6	Etat 0	X
E7	Etat 0	X
...	...	...

Exemple Moore : table de transition réduite

E	EP	EF
E0	Etat 0	Etat 0
E4	Etat 0	Etat 1
X	Etat 1	Etat 2
E1	Etat 2	Etat 2
E2	Etat 2	Etat 0

Exemple Moore : Codage

- Détermination du nombre de bascules

Exemple Moore : Codage

- Détermination du nombre de bascules
  - $\lceil log_2(nombre d'états) \rceil$

Exemple Moore : Codage

- Détermination du nombre de bascules
  - $\lceil log_2(nombred'états) \rceil$
  - $\lceil log_2(3) \rceil = 2$

Exemple Moore : Codage

- Détermination du nombre de bascules
  - $\lceil log_2(nombred'états) \rceil$
  - $\lceil log_2(3) \rceil = 2$
  - Bascule 0 avec D_0 et Q_0 et Bascule 1 avec D_1 et Q_1

Exemple Moore : Codage

- Détermination du nombre de bascules
  - $\lceil log_2(nombred'états) \rceil$
  - $\lceil log_2(3) \rceil = 2$
  - Bascule 0 avec D_0 et Q_0 et Bascule 1 avec D_1 et Q_1
- Codage

Etat	Q_1	Q_0
Etat 0	0	0
Etat 1	0	1
Etat 2	1	0

Vecteur E	a	b	c
E0	0	0	0
E1	0	0	1
E2	0	1	0
E3	0	1	1
E4	1	0	0
E5	1	0	1
E6	1	1	0
E7	1	1	1

Vecteur S	x	y
S0	0	0
S1	0	1
S2	1	0
S3	1	1

Exemple Moore : table de transition (équations)

E	EP	EF
E0	Etat 0	Etat 0
E4	Etat 0	Etat 1
X	Etat 1	Etat 2
E1	Etat 2	Etat 2
E2	Etat 2	Etat 0

E			EP		EF	
a	b	c	Q_1^n	Q_0^n	Q_1^{n+1}	Q_0^{n+1}
0	0	0	0	0	0	0
1	0	0	0	0	0	1
x	x	x	0	1	1	0
0	0	1	1	0	1	0
0	1	0	1	0	0	0

Exemple Moore : Equations Bascules Registre d'état

Q_0^{n+1} :

		a	0	0	0	0	1	1	1	1
		b	0	0	1	1	1	1	0	0
Q_1^n	Q_0^n	c	0	1	1	0	0	1	1	0
0	0		0	x	x	x	x	x	x	1
0	1		0	0	0	0	0	0	0	0
1	1		x	0	x	0	x	x	x	x
1	0		x	0	x	0	x	x	x	x

Q_1^{n+1} :

		a	0	0	0	0	1	1	1	1
		b	0	0	1	1	1	1	0	0
Q_1^n	Q_0^n	c	0	1	1	0	0	1	1	0
0	0		0	x	x	x	x	x	x	0
0	1		1	1	1	1	1	1	1	1
1	1		x	x	x	x	x	x	x	x
1	0		x	1	x	0	x	x	x	x

Exemple Moore : Equations sorties

x :

Q_1^n \ Q_0^n	0	1
0	0	1
1	0	x

y :

Q_1^n \ Q_0^n	0	1
0	0	0
1	1	x

Exemple Moore : Equations

- $Q_0^{n+1} =$

Exemple Moore : Equations

- $Q_0^{n+1} =$
- $Q_1^{n+1} =$

Exemple Moore : Equations

- $Q_0^{n+1} =$
- $Q_1^{n+1} =$
- $x =$

Exemple Moore : Equations

- $Q_0^{n+1} =$
- $Q_1^{n+1} =$
- $x =$
- $y =$

Exemple Moore : Equations

- $Q_0^{n+1} =$
- $Q_1^{n+1} =$
- $x =$
- $y =$
- Schéma

Exemple Mealy

Exemple Mealy : Codage

- Détermination du nombre de bascules

Exemple Mealy : Codage

- Détermination du nombre de bascules
  - $\lceil log_2(nombred'états) \rceil$

Exemple Mealy : Codage

- Détermination du nombre de bascules
  - $\lceil log_2(nombred'états) \rceil$
  - $\lceil log_2(2) \rceil = 1$

Exemple Mealy : Codage

- Détermination du nombre de bascules
  - $\lceil log_2(nombred'états) \rceil$
  - $\lceil log_2(2) \rceil = 1$
  - Bascule 0 avec D_0 et Q_0

Exemple Mealy : Codage

- Détermination du nombre de bascules
  - $\lceil log_2(nombred'états) \rceil$
  - $\lceil log_2(2) \rceil = 1$
  - Bascule 0 avec D_0 et Q_0
- Codage

Etat	Q_0	Vecteur E	a	Vecteur S	x	y
Etat 0	0	E0	0	S0	0	0
Etat 1	1	E1	1	S1	0	1
				S2	1	0
				S3	1	1

Exemple Moore : table de transition

E	EP	EF	S
E0	Etat 0	Etat 0	S0
E1	Etat 0	Etat 1	S1
E0	Etat 1	Etat 0	S2
E1	Etat 1	Etat 1	S3

E	EP	EF	S	
a	Q_0^n	Q_0^{n+1}	x	y
0	0	0	0	0
1	0	1	0	1
0	1	0	1	0
1	1	1	1	1

Exemple Mealy : Equations Bascules et sorties

Q_0^{n+1} :

Q_0^n \ a	0	1
0	0	1
1	0	1

x :

Q_0^n \ a	0	1
0	0	0
1	1	1

y :

Q_0^n \ a	0	1
0	0	1
1	0	1

Exemple Mealy : Equations

- $Q_0^{n+1} =$

Exemple Mealy : Equations

- $Q_0^{n+1} =$
- $x =$

Exemple Mealy : Equations

- $Q_0^{n+1} =$
- $x =$
- $y =$

Exemple Mealy : Equations

- $Q_0^{n+1} =$
- $x =$
- $y =$
- Schéma

One hot Encoding

- Codage par défaut : codage binaire = coder les états avec le moins de bits possibles

One hot Encoding

- Codage par défaut : codage binaire = coder les états avec le moins de bits possibles
- One hot Encoding : rajouter des bascules afin de simplifier les équations combinatoires

One hot Encoding

- Codage par défaut : codage binaire = coder les états avec le moins de bits possibles
- One hot Encoding : rajouter des bascules afin de simplifier les équations combinatoires
- Principe : Une bascule par état

One hot Encoding

- Codage par défaut : codage binaire = coder les états avec le moins de bits possibles
- One hot Encoding : rajouter des bascules afin de simplifier les équations combinatoires
- Principe : Une bascule par état
- Il n'y a qu'une bascule ayant sa sortie à 1, toutes les autres ont leur sortie à 0

One hot Encoding

- Codage par défaut : codage binaire = coder les états avec le moins de bits possibles
- One hot Encoding : rajouter des bascules afin de simplifier les équations combinatoires
- Principe : Une bascule par état
- Il n'y a qu'une bascule ayant sa sortie à 1, toutes les autres ont leur sortie à 0
- Lorsque la sortie de la bascule *i* est à 1 alors la machine est dans l'état *i*

Plan

1. Introduction
2. Outils de conception : VHDL
3. Méthodes : Machines à états
   - Les Machines à Etats en VHDL
4. Les mémoires
5. FPGA
6. Le Port Jtag
7. Conception d'un système programmable

1. Introduction
2. Outils de conception : VHDL
   - Différents types de description
   - Les types en VHDL
   - Signaux et Variables en VHDL
   - Les tableaux
   - Générique
   - Clause Wait
   - Test Bench
   - Simulation
   - Paquetage, Procédure et Fonction
3. Méthodes : Machines à états
   - Les Machines à Etats en VHDL
4. Les mémoires
5. FPGA
6. Le Port Jtag
7. Conception d'un système programmable

Les machines à états en VHDL

- Moore ou Mealy
- Utilisation d'au moins 2 process
  - Un process de transition de l'état futur à l'état présent
  - Un process de détermination des sorties et de l'état futur
- Utilisation de type énuméré

Machines de Moore - 2 process - Entité

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY mae IS
   PORT(
      a   : IN     std_logic;
      hor : IN     std_logic;
      raz : IN     std_logic;
      b   : OUT    std_logic
   );
END mae ;
```

Machine de Moore - 2 process - Architecture

```
ARCHITECTURE diagram OF mae IS

   TYPE STATE_TYPE IS (Etat0,Etat1,Etat2);

   SIGNAL EtatPresent : STATE_TYPE ;
   SIGNAL EtatFutur : STATE_TYPE ;

BEGIN
   clocked : PROCESS(hor,raz)

   BEGIN
      ...
   END PROCESS clocked;

   nextstate : PROCESS (EtatPresent,a)
   BEGIN
      ...
   END PROCESS nextstate;
END diagram;
```

Machine de Moore - 2 process - Architecture

```
clocked : PROCESS(hor,raz)
BEGIN
   IF (raz = '0') THEN
      EtatPresent <= Etat0;
   ELSIF (hor'EVENT AND hor = '1') THEN
      EtatPresent <= EtatFutur;
   END IF;
END PROCESS clocked;
```

Machine de Moore - 2 process - Architecture

```
nextstate : PROCESS (EtatPresent,a)
BEGIN
   CASE EtatPresent IS
   WHEN Etat0 =>
      b <= '1';
      EtatFutur <= Etat1;
   WHEN Etat1 =>
      b <= '0';
      IF (a = '1') THEN
         EtatFutur <= Etat2;
      ELSIF (a = '0') THEN
         EtatFutur <= Etat1;
      ELSE
         EtatFutur <= Etat1;
      END IF;
   WHEN Etat2 =>
      b <= '0';
      EtatFutur <= Etat0;
   WHEN OTHERS =>
      EtatFutur <= Etat0;
   END CASE;
END PROCESS nextstate;
```

Machine de Mealy - 2 process - Entité

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY mae IS
   PORT(
      a   : IN     std_logic;
      hor : IN     std_logic;
      raz : IN     std_logic;
      b   : OUT    std_logic
   );
END mae ;
```

Machine de Mealy - 2 process - Architecture

```
ARCHITECTURE diagram OF mae IS
   TYPE STATE_TYPE IS (Etat0,Etat1,Etat2);

   SIGNAL EtatPresent : STATE_TYPE ;
   SIGNAL EtatFutur : STATE_TYPE ;

BEGIN

   clocked : PROCESS(hor,raz)
   BEGIN
      ...
   END PROCESS clocked;

   nextstate : PROCESS (EtatPresent,a)
   BEGIN
      ...
   END PROCESS nextstate;
END diagram;
```

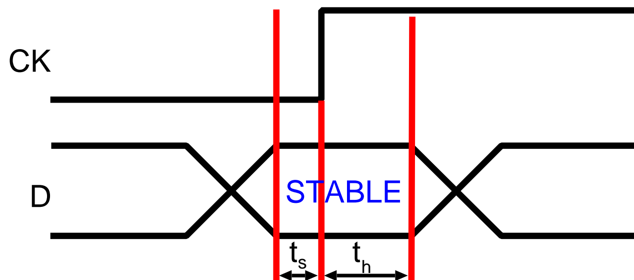
Machine de Mealy - 2 process - Architecture

```
clocked : PROCESS(hor,raz)

 BEGIN
    IF (raz = '0') THEN
       EtatPresent <= Etat0;
       -- Reset Values
    ELSIF (hor'EVENT AND hor = '1') THEN
       EtatPresent <= EtatFutur;
       -- Default Assignment To Internals

    END IF;

 END PROCESS clocked;
```

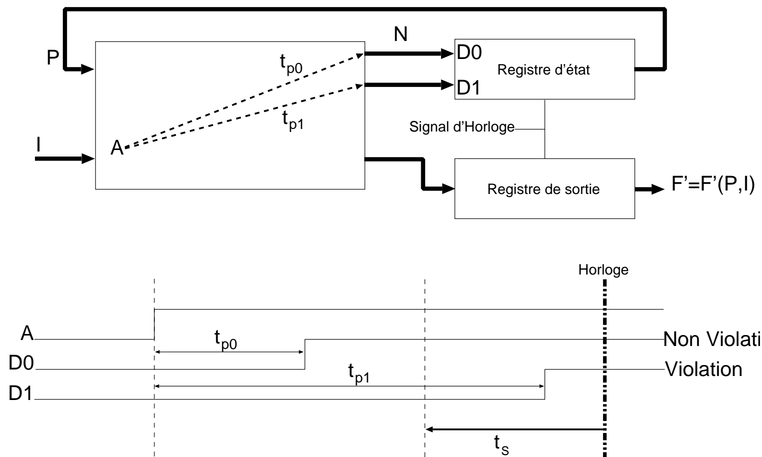
Machine de Mealy - 2 process - Architecture

```
nextstate : PROCESS (EtatPresent,a)
BEGIN
   CASE EtatPresent IS
   WHEN Etat0 =>
      b <= '1';
      EtatFutur <= Etat1;
   WHEN Etat1 =>
      IF (a = '1') THEN
         b <= '0';
         EtatFutur <= Etat2;
      ELSIF (a = '0') THEN
         b <= '1' ;
         EtatFutur <= Etat1;
      ELSE
         EtatFutur <= Etat1;
      END IF;
   WHEN Etat2 =>
      b <= '0';
      EtatFutur <= Etat0;
   WHEN OTHERS =>
      EtatFutur <= Etat0;
   END CASE;
END PROCESS nextstate;
```

Synchronisation des Entrées



Synchronisation des Entrées : problème



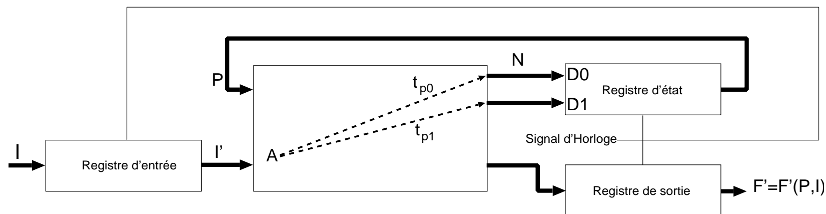
Synchronisation des Entrées : problème

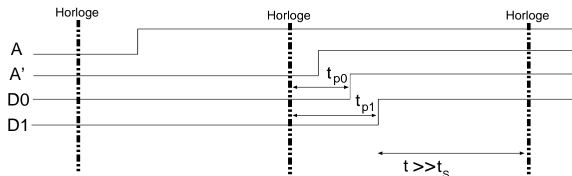
- Le changement d'une seule variable peut entrainer des changements multiples sur les entrées du registre d'état

Synchronisation des Entrées : problème

- Le changement d'une seule variable peut entrainer des changements multiples sur les entrées du registre d'état
- En cas de violation de t_S ou de t_H l'état de la machine n'est pas prédictible

Synchronisation des Entrées : solution ?





Synchronisation des Entrées : solution ?

- En cas de violation de t_S ou de t_H par l'entrée A' :

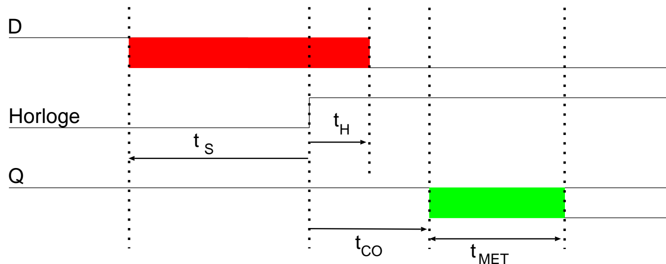
Synchronisation des Entrées : solution ?

- En cas de violation de t_S ou de t_H par l'entrée A' :
  - Si la sortie A de la bascule reste dans son état, le changement de l'information d'entrée sera prise en compte à l'impulsion suivante

Synchronisation des Entrées : solution ?

- En cas de violation de t_S ou de t_H par l'entrée A' :
  - Si la sortie A de la bascule reste dans son état, le changement de l'information d'entrée sera prise en compte à l'impulsion suivante
  - La sortie A de la bascule peut passer dans un état métastable

Métastabilité



Métastabilité

- Respect des contraintes t_S ou de t_H.

Métastabilité

- Respect des contraintes t_S ou de t_H.
- La violation des contraintes peut entrainer un état métastable de la bascule.

Métastabilité

- Respect des contraintes t_S ou de t_H.
- La violation des contraintes peut entrainer un état métastable de la bascule.
- L'état métastable n'apparaît pas forcément.

Métastabilité

- Respect des contraintes t_S ou de t_H.
- La violation des contraintes peut entrainer un état métastable de la bascule.
- L'état métastable n'apparaît pas forcément.
- La probabilité d'un état métastable est lié au processus de fabrication.

Métastabilité

- Respect des contraintes t_S ou de t_H.
- La violation des contraintes peut entrainer un état métastable de la bascule.
- L'état métastable n'apparaît pas forcément.
- La probabilité d'un état métastable est lié au processus de fabrication.
- La métastabilité ne met pas forcément le système dans un état indéterminé.

Métastabilité

- Respect des contraintes t_S ou de t_H.
- La violation des contraintes peut entrainer un état métastable de la bascule.
- L’état métastable n’apparaît pas forcément.
- La probabilité d’un état métastable est lié au processus de fabrication.
- La métastabilité ne met pas forcément le système dans un état indéterminé.
- La métastabilité est caractérisée par le MTBF (Mean Time Between Failure) du circuit

Métastabilité

- Don't look for solution, there is none (don't believe everything you read).
- Can't guarantee correct operation with arbitrary clock and data phase
- Do design such that worst case probability of error is acceptable
- Do understand and be able to identify trouble spots in design

Métastabilité

- Mean Time Between Failure (MTBF) for a synchronization flip-flop can be estimated with the following formula

$$MTBF = \frac{e^{C_2 x t_{MET}}}{C_1 x f_{clock} x f_{data}}$$

- where
  - f_{CLOCK} is the system clock frequency
  - f_{DATA} is the data transfer frequency
  - t_{MET} is the additional time allowed for the flip-flop to settle
  - C_1 and C_2 are device specific parameters found by plotting the natural log of MTBF versus tMET and performing linear regression analysis on the data

Métastabilité

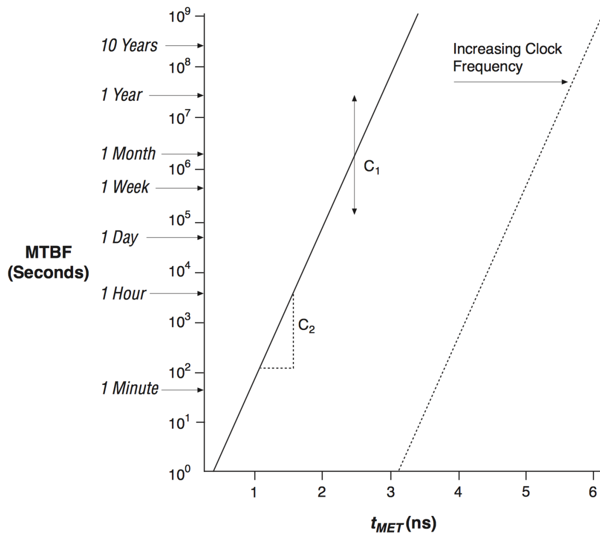


Figure:

Les mémoires - types

- Il existe deux Acronymes pour définir les Mémoires :

Les mémoires - types

- Il existe deux Acronymes pour définir les Mémoires :
  - ROM : Read Only Memory (Mémoire Lecture Seule) - Mémoire Morte

Les mémoires - types

- Il existe deux Acronymes pour définir les Mémoires :
  - ROM : Read Only Memory (Mémoire Lecture Seule) - Mémoire Morte
  - RAM : Random Access Memory (Mémoire à accès Aléatoire) - Mémoire Vive

Les mémoires - types

- Il existe deux Acronymes pour définir les Mémoires :
  - ROM : Read Only Memory (Mémoire Lecture Seule) - Mémoire Morte
  - RAM : Random Access Memory (Mémoire à accès Aléatoire) - Mémoire Vive
- Ces acronymes définissent tous les deux des mémoires à accès Aléatoire

Les mémoires - types

- Il existe deux Acronymes pour définir les Mémoires :
  - ROM : Read Only Memory (Mémoire Lecture Seule) - Mémoire Morte
  - RAM : Random Access Memory (Mémoire à accès Aléatoire) - Mémoire Vive
- Ces acronymes définissent tous les deux des mémoires à accès Aléatoire
- Mais il existe aussi des mémoires à accès séquentiel

Les mémoires - types

- ROM : Read Only Memory = Mémoire Lecture Seule ?

Les mémoires - types

- ROM : Read Only Memory = Mémoire Lecture Seule ?
- Les noms n'ont de sens que pour les auteurs !

Les mémoires - types

- ROM : Read Only Memory = Mémoire Lecture Seule ?
- Les noms n'ont de sens que pour les auteurs !
- Il existes des mémoires avec différentes caractéristiques

Les mémoires - Une classification

- Quels Critères pour classifier ?

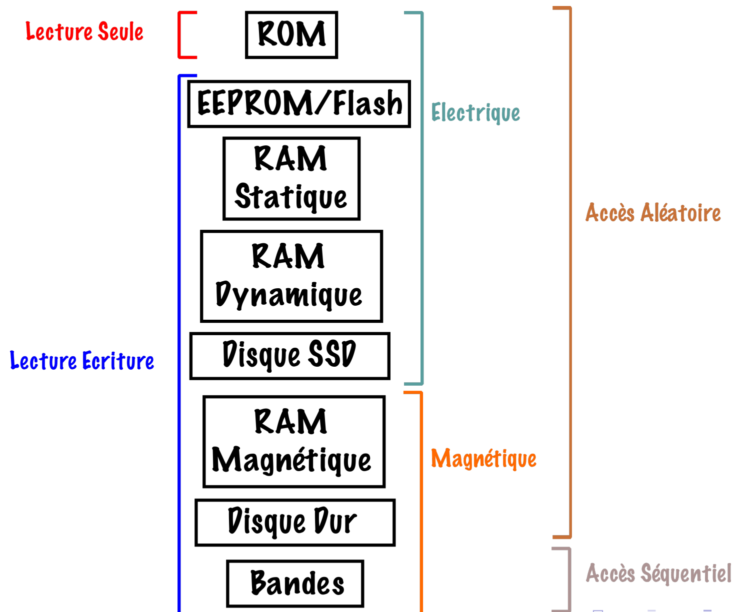
Les mémoires - Une classification

- Quels Critères pour classifier ?
- Critères pertinents ou apparaissant comme tels.

Les mémoires - Une classification

- Quels Critères pour classifier ?
- Critères pertinents ou apparaissant comme tels.
- Critères pas forcément indéfiniment valides.

Les mémoires - Une classification



Les mémoires - Définitions

- Définition sur les données manipulées

Les mémoires - Définitions

- Définition sur les données manipulées
- bit : Binary digIT - Plus petite quantité binaire

Les mémoires - Définitions

- Définition sur les données manipulées
- bit : Binary digIT - Plus petite quantité binaire
- octet (*byte*) : information codée sur 8 bits - Unité de référence

Les mémoires - Définitions

- Définition sur les données manipulées
- bit : Binary digIT - Plus petite quantité binaire
- octet (*byte*) : information codée sur 8 bits - Unité de référence
- mot (*word*) : référence la taille du bus de données utilisé - dépendant du système - Un processeur 16 bits a une taille de mot de 16 bits, un processeur 64 bits a une taille de mot de 64 bits - Taille égale la taille du bus.

Les mémoires - Définitions

- Bus : interfaces entre mémoire et monde extérieur

Les mémoires - Définitions

- Bus : interfaces entre mémoire et monde extérieur
  - Bus de données (M bits) : délivre un mot de M bits de la mémoire

Les mémoires - Définitions

- Bus : interfaces entre mémoire et monde extérieur
  - Bus de données (M bits) : délivre un mot de M bits de la mémoire
  - Bus d'adresses (N bits) : sélectionne 1 mot parmi 2^N

Les mémoires - Définitions

- Bus : interfaces entre mémoire et monde extérieur
  - Bus de données (M bits) : délivre un mot de M bits de la mémoire
  - Bus d'adresses (N bits) : sélectionne 1 mot parmi 2^N
- Signaux de contrôle - Il existe au moins

Les mémoires - Définitions

- Bus : interfaces entre mémoire et monde extérieur
  - Bus de données (M bits) : délivre un mot de M bits de la mémoire
  - Bus d'adresses (N bits) : sélectionne 1 mot parmi 2^N
- Signaux de contrôle - Il existe au moins
  - CS : Chip Select (Sélecteur de boitier) - Commande de l'état haute impédance du bus de données

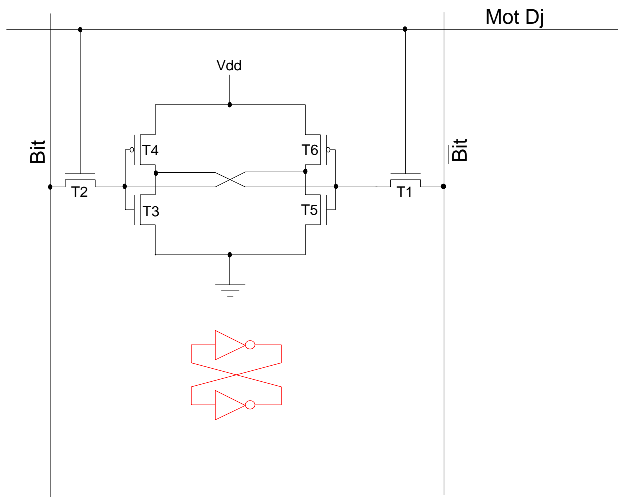
Les mémoires - Définitions

- Bus : interfaces entre mémoire et monde extérieur
  - Bus de données (M bits) : délivre un mot de M bits de la mémoire
  - Bus d'adresses (N bits) : sélectionne 1 mot parmi 2^N
- Signaux de contrôle - Il existe au moins
  - CS : Chip Select (Sélecteur de boitier) - Commande de l'état haute impédance du bus de données
  - R : Read (Lecture) - Indique que l'accès à la mémoire est une lecture

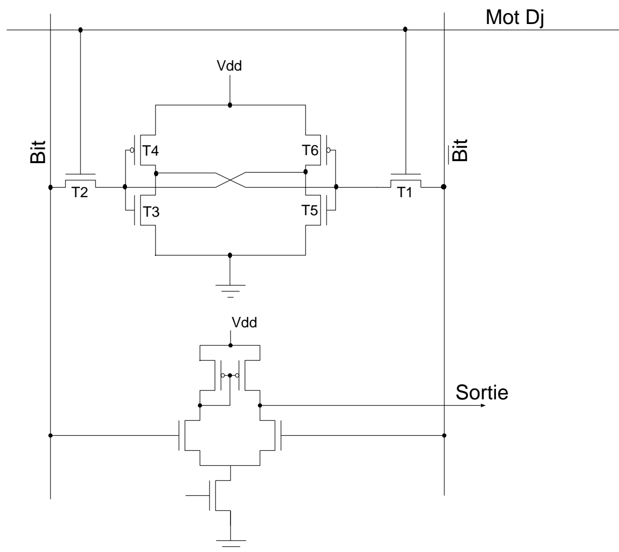
Les mémoires - Définitions

- Bus : interfaces entre mémoire et monde extérieur
  - Bus de données (M bits) : délivre un mot de M bits de la mémoire
  - Bus d'adresses (N bits) : sélectionne 1 mot parmi 2^N
- Signaux de contrôle - Il existe au moins
  - CS : Chip Select (Sélecteur de boitier) - Commande de l'état haute impédance du bus de données
  - R : Read (Lecture) - Indique que l'accès à la mémoire est une lecture
  - W : Write (Ecriture) - Indique que l'accès à la mémoire est une écriture

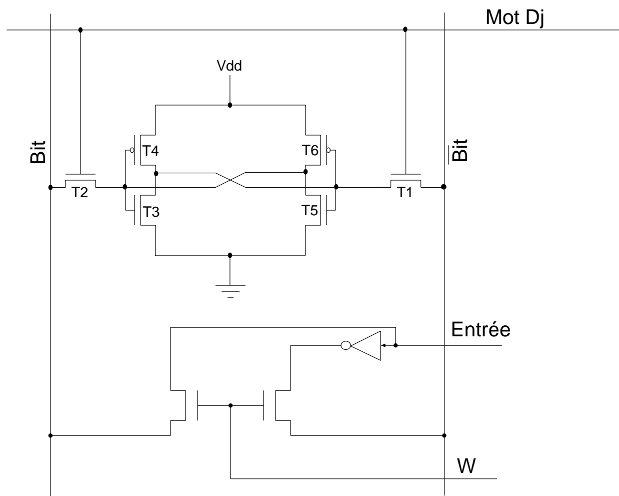
Les mémoires RAM - Technologie SRAM



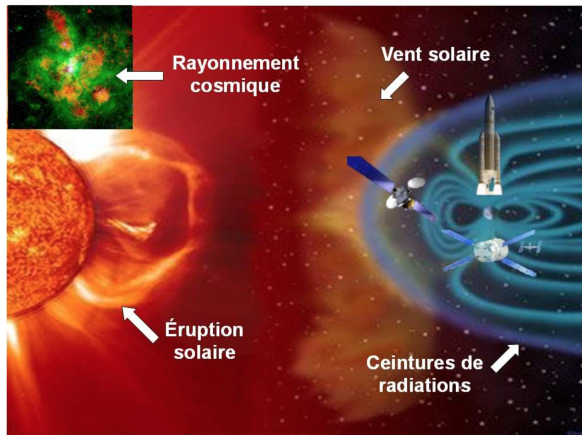
Les mémoires RAM - Technologie SRAM Lecture



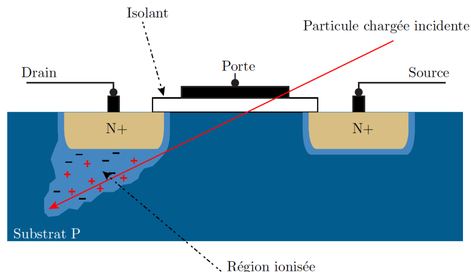
Les mémoires RAM - Technologie SRAM Ecriture



Les mémoires RAM - Techonologie SRAM : les radiations

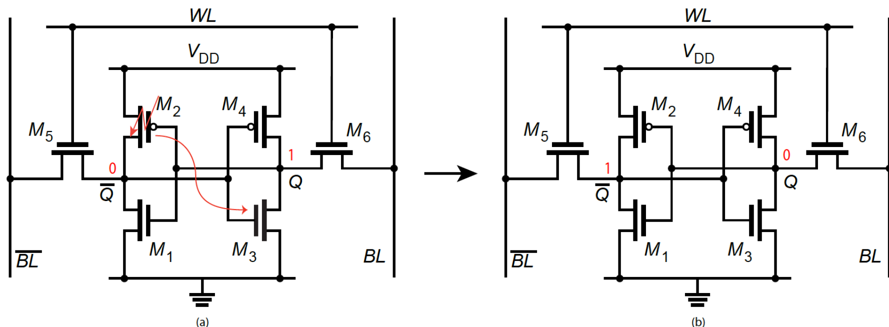


Les mémoires RAM - Techonologie SRAM : les SET



- d'après Fouad Sahraoui - thèse de doctorat - mars 2015

Les mémoires RAM - Techonologie SRAM : les SEU



- d'après Fouad Sahraoui - thèse de doctorat - mars 2015

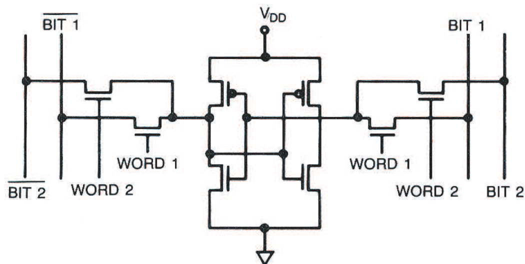
Les mémoires RAM - Autres Cellules SRAM

- Cellule biport : permet d’accèder simultanément à 2 données en mémoires

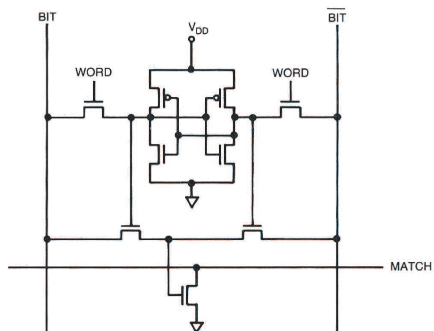
Les mémoires RAM - Autres Cellules SRAM

- Cellule biport : permet d'accèder simultanément à 2 données en mémoires
- Cellule CAM : cellule adressée par contenu, utile pour les mémoires caches associatives

Les mémoires RAM - les Biports



Les mémoires RAM - les CAM



Les mémoires RAM - Technologie SRAM Bilan

- Rapide

Les mémoires RAM - Technologie SRAM Bilan

- Rapide
- Cellule Statique : Rétention infinie tant qu'alimentation

Les mémoires RAM - Technologie SRAM Bilan

- Rapide
- Cellule Statique : Rétention infinie tant qu'alimentation
- Faible Consommation (CMOS)

Les mémoires RAM - Technologie SRAM Bilan

- Rapide
- Cellule Statique : Rétention infinie tant qu'alimentation
- Faible Consommation (CMOS)
- Cellule volumineuse

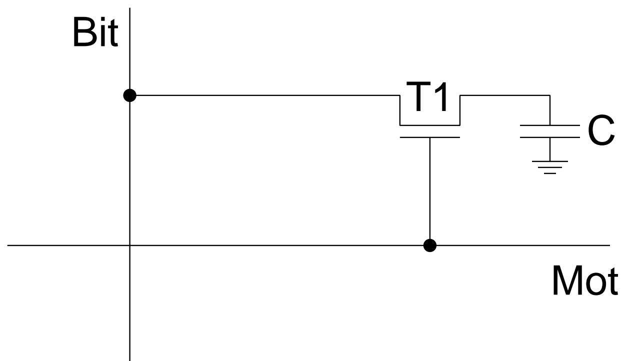
Les mémoires RAM - Technologie SRAM Bilan

- Rapide
- Cellule Statique : Rétention infinie tant qu'alimentation
- Faible Consommation (CMOS)
- Cellule volumineuse
- Chère

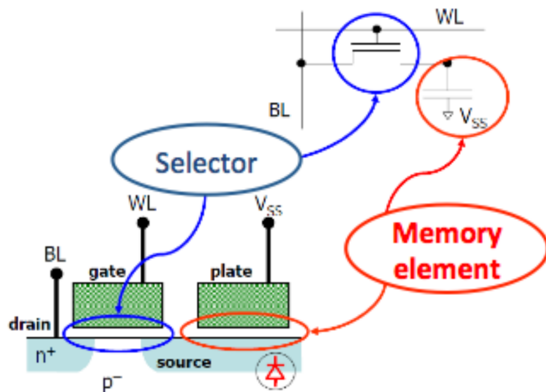
Les mémoires RAM - Technologie SRAM Bilan

- Rapide
- Cellule Statique : Rétention infinie tant qu'alimentation
- Faible Consommation (CMOS)
- Cellule volumineuse
- Chère
- Volatile

Les mémoires RAM - Technologie DRAM

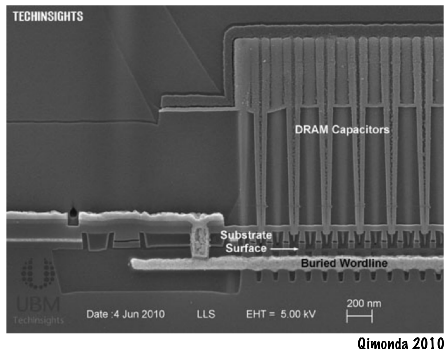


Les mémoires RAM - Technologie DRAM



- d'après Marc Bocquet - AMU

Les mémoires RAM - Technologie DRAM



Qimonda 2010

- d'après Marc Bocquet - AMU

Les mémoires RAM - Technologie DRAM Bilan

- Compact

Les mémoires RAM - Technologie DRAM Bilan

- Compact
- Faible Consommation (CMOS)

Les mémoires RAM - Technologie DRAM Bilan

- Compact
- Faible Consommation (CMOS)
- Lecture destructrice : nécessité de réécriture

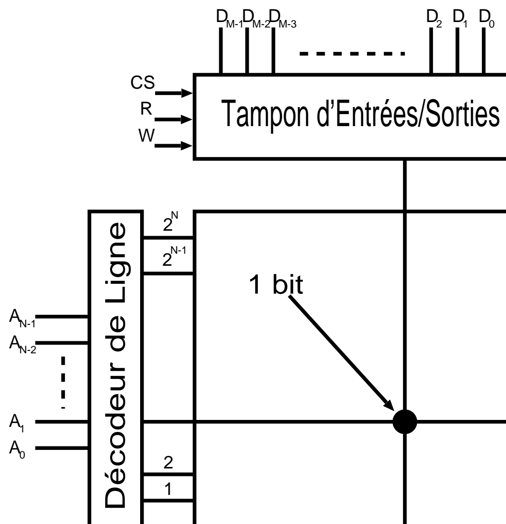
Les mémoires RAM - Technologie DRAM Bilan

- Compact
- Faible Consommation (CMOS)
- Lecture destructrice : nécessité de réécriture
- Rafraîchissement régulier pour combler la décharge du condensateur

Les mémoires RAM - Technologie DRAM Bilan

- Compact
- Faible Consommation (CMOS)
- Lecture destructrice : nécessité de réécriture
- Rafraîchissement régulier pour combler la décharge du condensateur
- Volatile

Les mémoires RAM - Organisation Interne



Les mémoires RAM - Organisation Interne

- Adressage Simple : une adresse $\Rightarrow$ une donnée

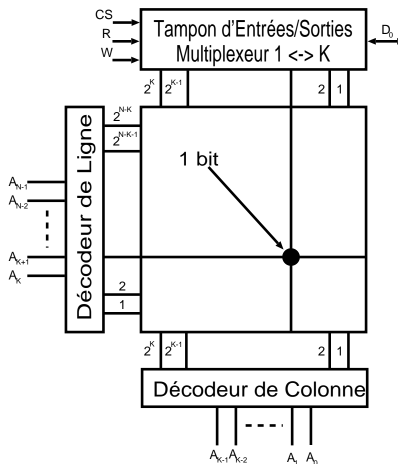
Les mémoires RAM - Organisation Interne

- Adressage Simple : une adresse ⇒ une donnée
- Grande capacité mémoire ⇒ bus de donnée large

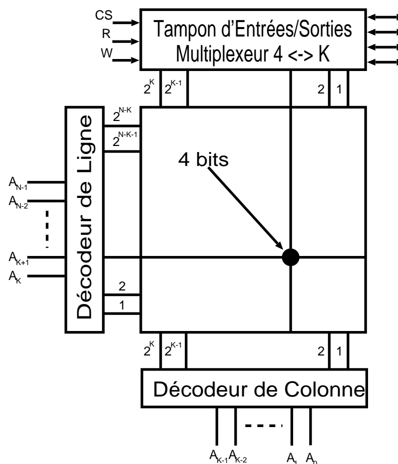
Les mémoires RAM - Organisation Interne

- Adressage Simple : une adresse ⇒ une donnée
- Grande capacité mémoire ⇒ bus de donnée large
- Scinder l'adresse en deux parties : ligne et colonne

Les mémoires RAM - Organisation Interne



Les mémoires RAM - Organisation Interne



Les mémoires RAM - Organisation Externe

- Association de plusieurs blocs mémoire pour

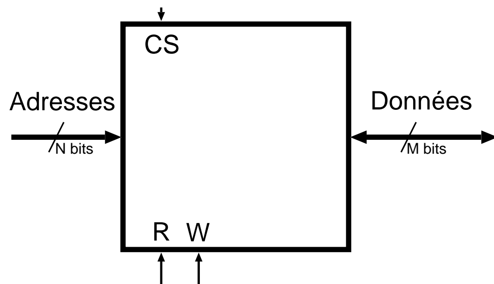
Les mémoires RAM - Organisation Externe

- Association de plusieurs blocs mémoire pour
  - Augmenter la capacité de stockage

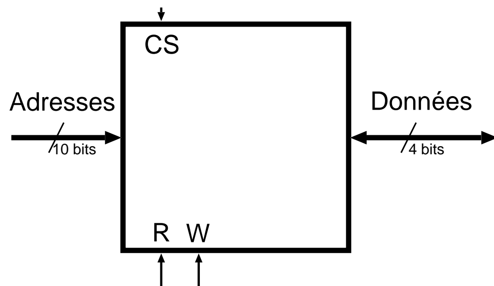
Les mémoires RAM - Organisation Externe

- Association de plusieurs blocs mémoire pour
  - Augmenter la capacité de stockage
  - Augmenter la taille des données

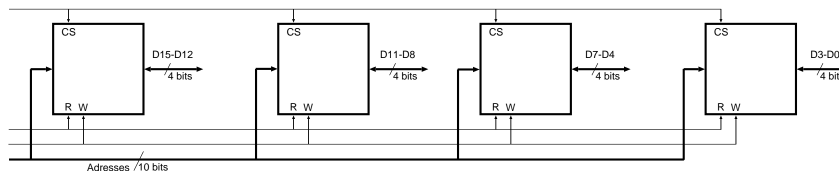
Les mémoires RAM - Organisation Externe



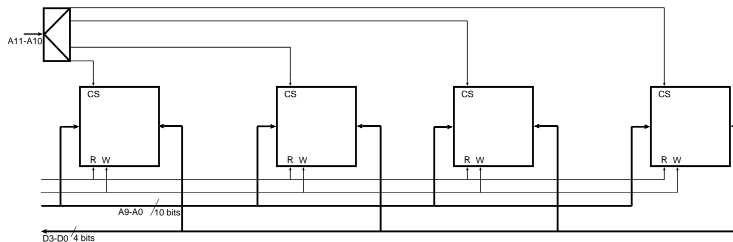
Les mémoires RAM - Organisation Externe



Les mémoires RAM - Organisation Externe



Les mémoires RAM - Organisation Externe



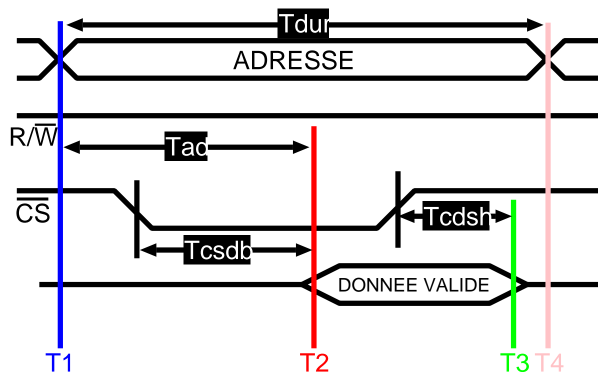
Les mémoires RAM - Cycles

- Pour un bon fonctionnement de la mémoire

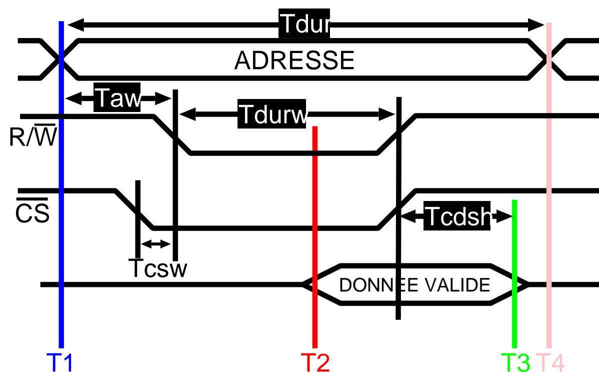
Les mémoires RAM - Cycles

- Pour un bon fonctionnement de la mémoire
- Repect de temps indiqué dans les cycles de lecture et d'écriture

Les mémoires RAM - Cycle de lecture



Les mémoires RAM - Cycle d’écritue



Vhdl : mémoire

```
entity memoire is
port (donnee : in std_logic_vector(7 downto 0);
      adresse: in std_logic_vector(6 downto 0);
      rw : in std_logic;
      cs : in std_logic;
      sortie: out std_logic_vector(7 downto 0));
end entity memoire;
```

Vhdl : mémoire

```
architecture comport of memoire is
begin
 process(cs,adresse,donnee,rw) is
   type memory  is array (0 to 79) of std_logic_vector(7 downto 0);
  variable mem : memory;
  begin
   if cs='0' then
    if rw='1' then
     sortie <= mem(conv_integer(adresse));
    elsif rw='0' then
     mem(conv_integer(adresse)) := donnee;
    end if;
   else
    sortie<= "ZZZZZZZZ";
   end if;
  end process;
end architecture comport;
```

Les mémoires RAM - Technologie SDRAM

- Jusque là : technologie asynchrone

Les mémoires RAM - Technologie SDRAM

- Jusque là : technologie asynchrone
- Difficile de maitriser les temps de propagation

Les mémoires RAM - Technologie SDRAM

- Jusque là : technologie asynchrone
- Difficile de maitriser les temps de propagation
- Nécessité de synchroniser les accès mémoire

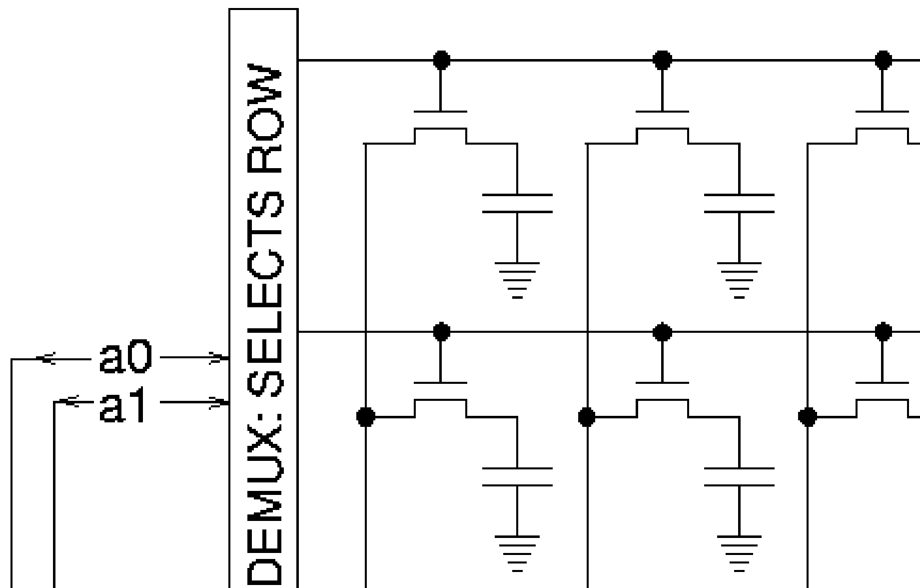
Les mémoires RAM - Technologie SDRAM

- Jusque là : technologie asynchrone
- Difficile de maitriser les temps de propagation
- Nécessité de synchroniser les accès mémoire
- SDRAM: Synchone DRAM

Les mémoires RAM - Technologie SDRAM

- Jusque là : technologie asynchrone
- Difficile de maitriser les temps de propagation
- Nécessité de synchroniser les accès mémoire
- SDRAM: Synchone DRAM
- Introduction du mode Rafale (BURST)

Les mémoires RAM - Technologie DRAM



Les mémoires RAM - Technologie DDRAM

- Fréquence Processeur = 800 MHz, fréquence SDRAM = 100 MHz

Les mémoires RAM - Technologie DDRAM

- Fréquence Processeur = 800 MHz, fréquence SDRAM = 100 MHz
- Nécessité de réduire cet écart

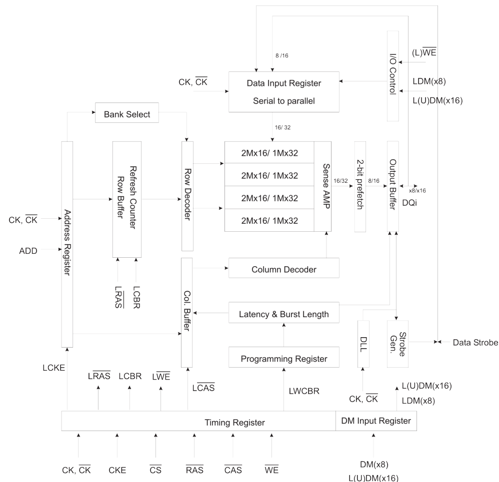
Les mémoires RAM - Technologie DDRAM

- Fréquence Processeur = 800 MHz, fréquence SDRAM = 100 MHz
- Nécessité de réduire cet écart
- Idée : lire sur le front montant et descendant de l'horloge

Les mémoires RAM - Technologie DDRAM

- Fréquence Processeur = 800 MHz, fréquence SDRAM = 100 MHz
- Nécessité de réduire cet écart
- Idée : lire sur le front montant et descendant de l'horloge
- DDRAM : Double rate DRAM (DRAM à double débit)

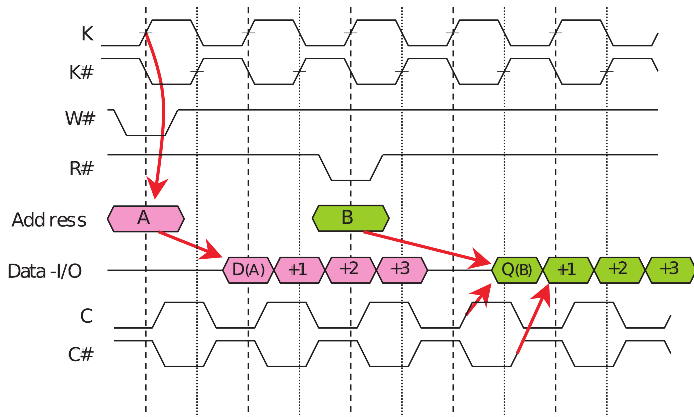
Les mémoires RAM - Technologie DDRAM



Les mémoires RAM - Technologie DDRAM

COMMAND			CKEn-1	CKEn	$\overline{CS}$	$\overline{RAS}$	$\overline{CAS}$	$\overline{WE}$	BA0,1	A10/AP	A0 ~ A9, A11
Register	Extended MRS		H	X	L	L	L	L	OP CODE		
Register	Mode Register Set		H	X	L	L	L	L	OP CODE		
Refresh	Auto Refresh		H	H	L	L	L	H	X		
	Self Refresh	Entry		L							
		Exit	L	H	L	H	H	H	X		
					H	X	X	X			
Bank Active & Row Addr.			H	X	L	L	H	H	V	Row Address	
Read & Column Address	Auto Precharge Disable		H	X	L	H	L	H	V	L	Column Address
	Auto Precharge Enable									H	
Write & Column Address	Auto Precharge Disable		H	X	L	H	L	L	V	L	Column Address
	Auto Precharge Enable									H	
Burst Stop			H	X	L	H	H	L	X		
Precharge	Bank Selection		H	X	L	L	H	L	V	L	X
	All Banks								X	H	
Active Power Down		Entry	H	L	H	X	X	X	X		
					L	V	V	V			
		Exit	L	H	X	X	X	X			
Precharge Power Down Mode		Entry	H	L	H	X	X	X	X		
					L	H	H	H			
		Exit	L	H	H	X	X	X			
					L	V	V	V			
DM(UDM/LDM for x16 only)			H	X					X		
No operation (NOP) : Not defined			H	X	H	X	X	X	X		
					L	H	H	H			

Les mémoires RAM - Technologie DDRAM



Les mémoires RAM - Technologie QDRAM

- Idée1 : lire sur le front montant et descendant de l'horloge comme une DDRAM

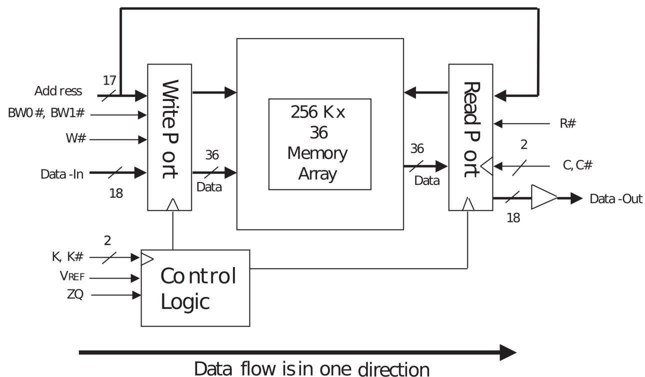
Les mémoires RAM - Technologie QDRAM

- Idée1 : lire sur le front montant et descendant de l'horloge comme une DDRAM
- Idée2 : dissocier entrée et sortie pour permettre une lecture et une écriture simultanée

Les mémoires RAM - Technologie QDRAM

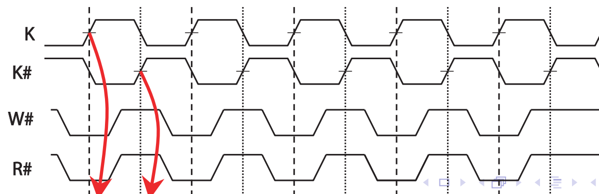
- Idée1 : lire sur le front montant et descendant de l'horloge comme une DDRAM
- Idée2 : dissocier entrée et sortie pour permettre une lecture et une écriture simultanée
- QDRAM : Quad rate DRAM (DRAM à quadruple débit)

Les mémoires RAM - Technologie QDRAM

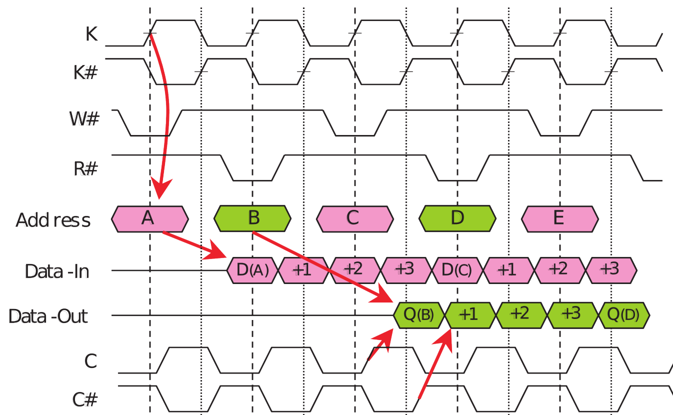


10

Les mémoires RAM - Technologie QDRAM



Les mémoires RAM - Technologie QDRAM



Les mémoires RAM - Technologie QDRAM

	RLDRAM 3	RLDRAM 2	DDR4	DDR3	DDR2+/ QDR2+SRAM
Data Rate	400–2133 Mb/s	350–1066 Mb/s	1600–3200 Mb/s	600–2133 Mb/s	240–1100 MHz
^{t}RC	8ns	15–20ns	44–50ns	46–52ns	1.81ns
Density	576Mb, 1Gb	288Mb, 576Mb	2Gb – 16Gb	1Gb, 2Gb, 4Gb	18Mb, 36Mb, 72Mb
Voltages	1.35V core; 1.2V IO	1.8V core; 1.5–1.8V IO	1.05V or 1.1V (TBD); 1.2V core and IO	1.5V; 1.35V core and IO	1.8V core; 1.5–1.8V IO
Configurations	x18, x36	x9, x18, x36	x4, x8, x16	x4, x8, x16	x18, x36
Burst Length	2, 4, 8	2, 4, 8	8	8	DDR2+: 2 QDR2+: 4
Multibank Write	Enables 1.0ns READ ^{t}RC	Not Applicable	Not Applicable	Not Applicable	Not Applicable
Internal Banks	16	8	16	8	Not Applicable
^{t}FAW Delay	No delay	No delay	>28 clock cycles	>27 clock cycles	Not Applicable
Bus Turnaround Delay (RD-WR-RD)	1–2 clock cycles	1–2 clock cycles	>17 clock cycles	>15 clock cycles	DDR2+: 2–3 QDR2+: 0
RESET Pin	Available	Not Available	Available	Available	Not Available

Les mémoires ROM

- Nécessité de mémoire non-volatile

Les mémoires ROM

- Nécessité de mémoire non-volatile
- Bios de micro-ordinateur

Les mémoires ROM

- Nécessité de mémoire non-volatile
- Bios de micro-ordinateur
- Stockage de programmes dans des systèmes embarqués

Les mémoires ROM

- Nécessité de mémoire non-volatile
- Bios de micro-ordinateur
- Stockage de programmes dans des systèmes embarqués
- Reconfiguration automatique des FPGA

Les mémoires ROM - Technologie

- Mask - ROM : réalisé lors de la fabrication du circuit

Les mémoires ROM - Technologie

- Mask - ROM : réalisé lors de la fabrication du circuit
- PROM : Fusible - One Time Programming

Les mémoires ROM - Technologie

- Mask - ROM : réalisé lors de la fabrication du circuit
- PROM : Fusible - One Time Programming
- EPROM : Transistors à Grille Flottante - Reprogrammabilité

Les mémoires ROM - Technologie

- Mask - ROM : réalisé lors de la fabrication du circuit
- PROM : Fusible - One Time Programming
- EPROM : Transistors à Grille Flottante - Reprogrammabilité
- NOVRAM : RAM + Pile

Les mémoires ROM - Technologie - Mask - ROM

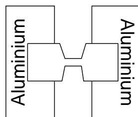
- Aucune Souplesse
- Faible Coût

Les mémoires ROM - Technologie - Mask - ROM

- Aucune Souplesse
- Faible Coût

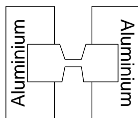
Les mémoires ROM - Technologie - PROM

- Fusible



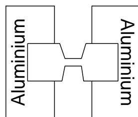
Les mémoires ROM - Technologie - PROM

- Fusible



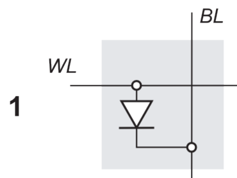
Les mémoires ROM - Technologie - PROM

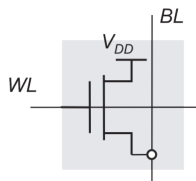
- Fusible

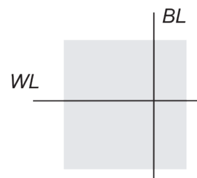


Les mémoires ROM - Cellules

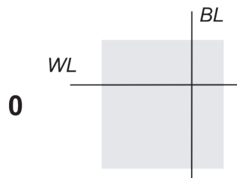
1

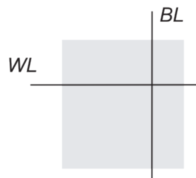


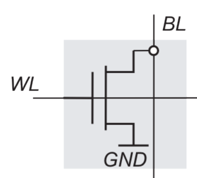




0

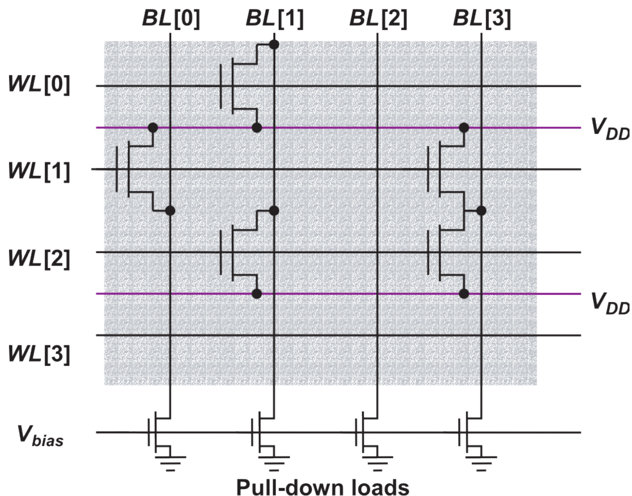




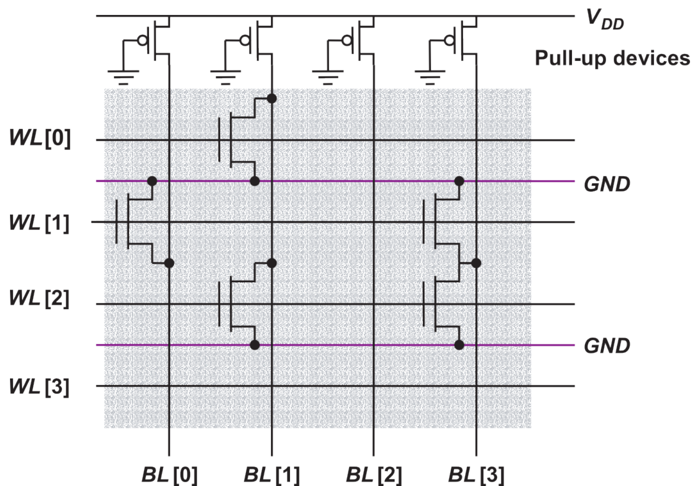


Diode ROM **MOS ROM 1** **MOS ROM 2**

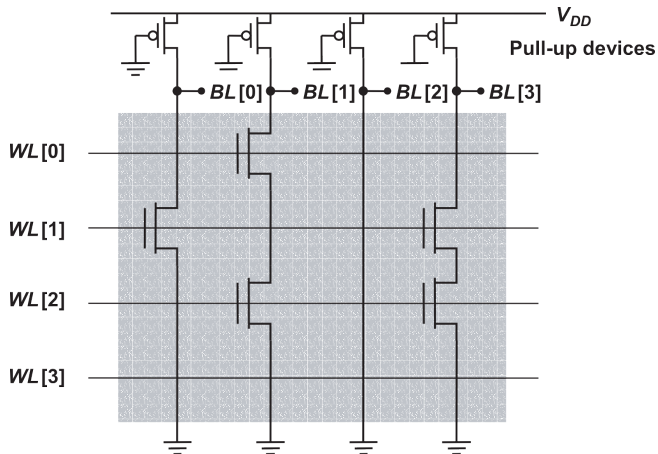
Les mémoires ROM - OR ROM



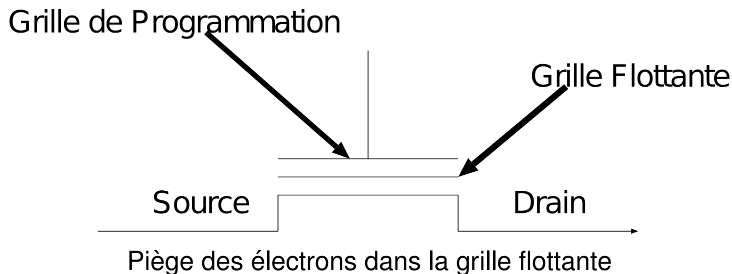
Les mémoires ROM - NOR ROM



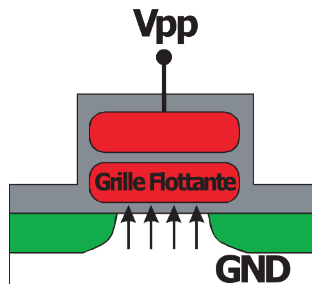
Les mémoires ROM - NAND ROM



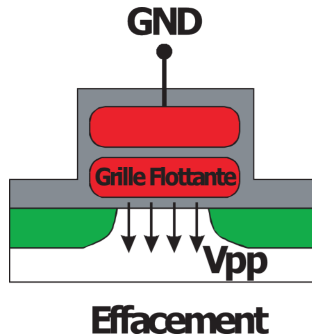
Les mémoires ROM : La Grille Flottante



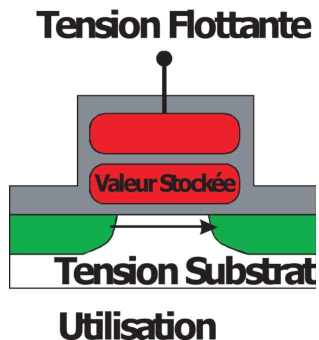
La Grille Flottante - programmation



La Grille Flottante - programmation



La Grille Flottante - programmation



Les mémoires ROM - Technologie - EPROM

- UV-EPROM : Effacement aux ultra-violet

Les mémoires ROM - Technologie - EPROM

- UV-EPROM : Effacement aux ultra-violet
- EEPROM : Effacement électrique par mot mémoire

Les mémoires ROM - Technologie - EPROM

- UV-EPROM : Effacement aux ultra-violet
- EEPROM : Effacement électrique par mot mémoire
- FLASH : Effacement électrique par bloc

Les mémoires ROM - Technologie - UV-EPROM

- Programmation hors système

Les mémoires ROM - Technologie - UV-EPROM

- Programmation hors système
- Effacement hors système

Les mémoires ROM - Technologie - UV-EPROM

- Programmation hors système
- Effacement hors système
- Temps d'effacement long (15 minutes pour les UV-EPROM)

Les mémoires ROM - Technologie - E-EPROM

- Programmation ISP (In Situ Programming ou In Serial Programming)

Les mémoires ROM - Technologie - E-EPROM

- Programmation ISP (In Situ Programming ou In Serial Programming)
- Effacement ISP

Les mémoires ROM - Technologie - E-EPROM

- Programmation ISP (In Situ Programming ou In Serial Programming)
- Effacement ISP
- Programmation par mot ou bloc

Les mémoires ROM - Technologie - E-EPROM

- Programmation ISP (In Situ Programming ou In Serial Programming)
- Effacement ISP
- Programmation par mot ou bloc
- Temps d'effacement rapide

Les mémoires ROM - Technologie - E-EPROM

- Programmation ISP (In Situ Programming ou In Serial Programming)
- Effacement ISP
- Programmation par mot ou bloc
- Temps d'effacement rapide
- Coût élevé car effacement par mot mémoire

Les mémoires ROM - Technologie - FLASH

- Programmation ISP

Les mémoires ROM - Technologie - FLASH

- Programmation ISP
- Effacement ISP

Les mémoires ROM - Technologie - FLASH

- Programmation ISP
- Effacement ISP
- Programmation par mot ou bloc

Les mémoires ROM - Technologie - FLASH

- Programmation ISP
- Effacement ISP
- Programmation par mot ou bloc
- Temps d'effacement très rapide

Les mémoires ROM - Technologie - FLASH

- Programmation ISP
- Effacement ISP
- Programmation par mot ou bloc
- Temps d'effacement très rapide
- Coût limité car effacement par bloc

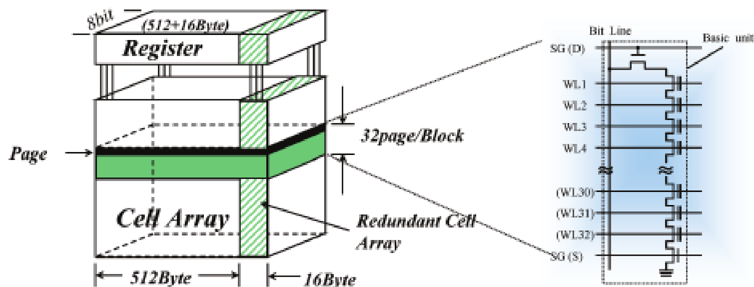
Les mémoires FLASH - NAND Versus NOR

	NAND	NOR
Cell Array	Word line Unit Cell Source line	Bit line Word line Contact Unit Cell Source line
Layout	2F 2F	2F 5F
Cross-section		
Cell size	$4F^2$	$10F^2$

Les mémoires FLASH - NAND Versus NOR

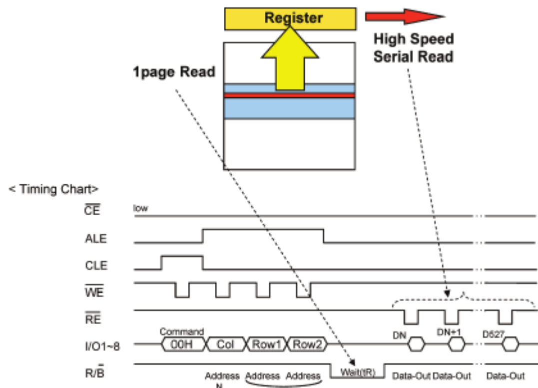
	NAND	NOR
Capacity	~ 1Gbit (2chips/pkg)	~ 128Mbit
Power Supply	2.7-3.6V	2.3-3.6V
I/O	x8	x8/x16
Access Time	50ns(serial access cycle) 25µs(random access)	70ns(30pF, 2.3V) 65ns(30pF, 2.7V)
Program Speed (typ.)	——	8µs/Byte
	200µs/512Byte	4.1ms/512Byte
Erase Speed(typ.)	2ms/Block (16KB)	700ms/Block
Prog+Erase(typ.)	33.6ms / 64KB	1.23s/Block (main:64KB)

Les mémoires FLASH - Organisation NAND

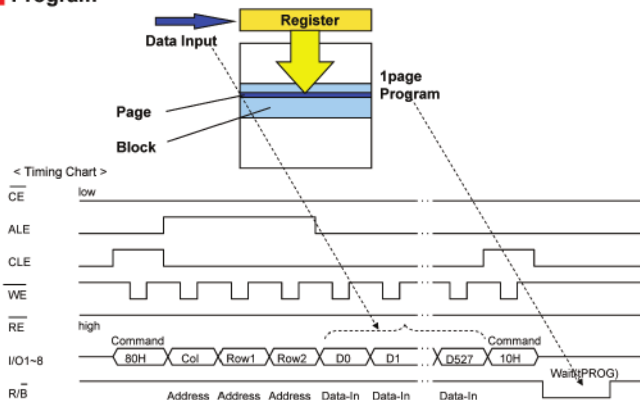


256Mb NAND Flash
Page Size : 512+16 Bytes
Block Size : 16KBytes
# of Blocks : 2048 Blocks

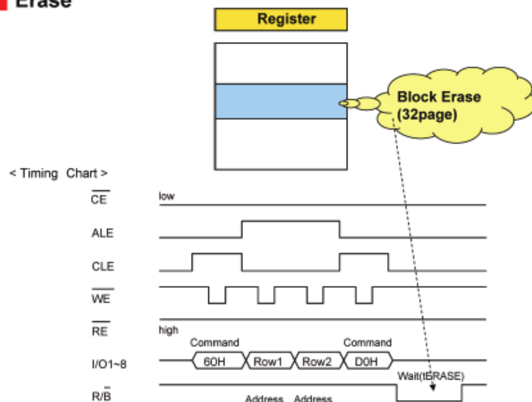
Les mémoires FLASH - Lecture



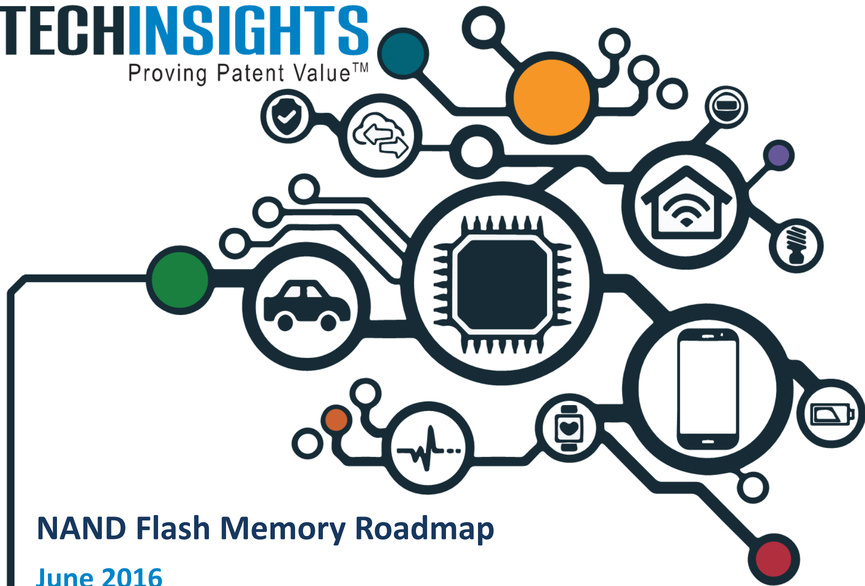
Les mémoires FLASH - Programmation



Les mémoires FLASH - Effacement



Les mémoires FLASH - Roadmap

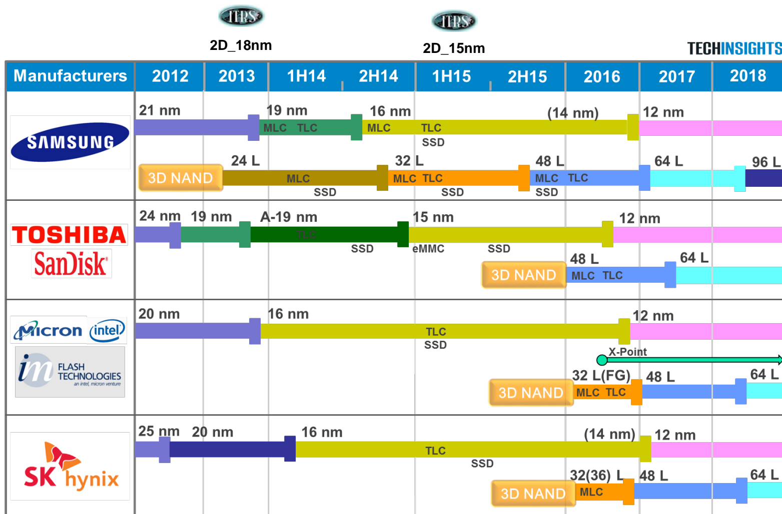


NAND Flash Memory Roadmap

June 2016

Expected 2D & 3D NAND Releases

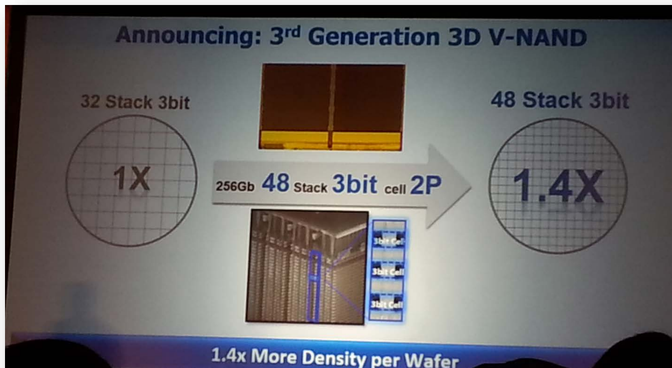
TECHINSIGHTS
Proving Patent Value™



Samsung 3D V-NAND

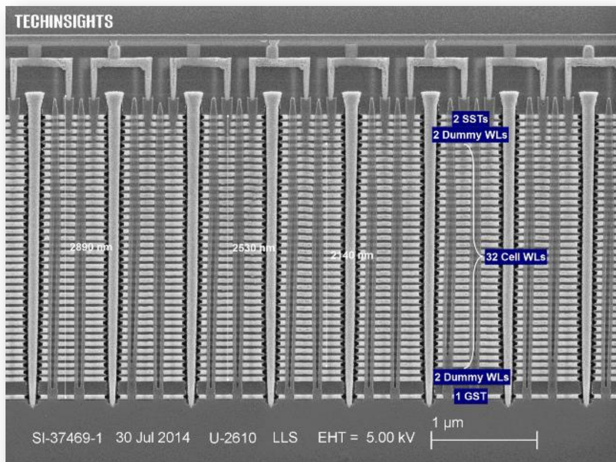
TECHINSIGHTS Proving Patent Value™

- 1st Generation (V1, 2012 ~ 2013): 24 Layers/128Gb
- 2nd Generation (V2, 2014 ~ 2015): 32 Layers/128Gb
- 3rd Generation (V3, 2015 3Q ~ 2016): 48 Layers/256Gb



Samsung 3D V2-NAND with 32 Layers

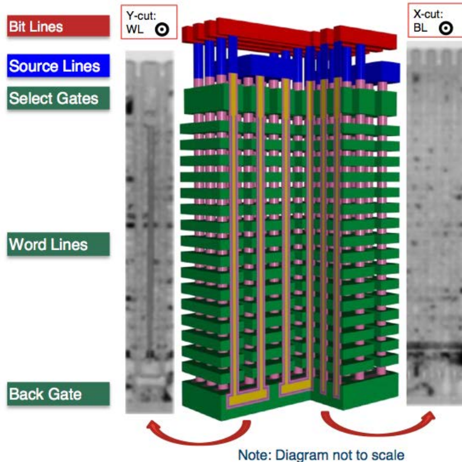
TECHINSIGHTS
Proving Patent Value™



Cross-section

Toshiba/SanDisk 3D-NAND (BiCS)

TECHINSIGHTS Proving Patent Value™

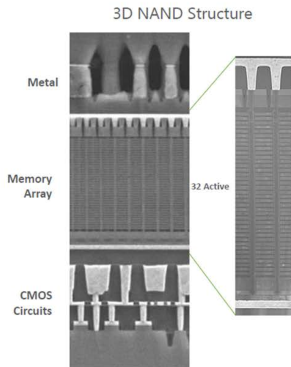


- Toshiba/SanDisk 3D NAND structure is 'BiCS' instead of 'P-BiCS'
- 128Gb 3D 48L BiCS MLC: CS (since March 26, 2015), but not in production yet. Currently sampling as of Aug. 2015
- 256Gb 3D 48L BiCS TLC: CS (since Aug. 4, 2015) and will be revealed on the market 4Q 2015

Micron 3D-NAND

TECHINSIGHTS
Proving Patent Value™

- >20% cost benefit of logic under array
- Continued 3D NAND performance gains through process improvements



Source: Feb. 12, 2016 Micron

Les mémoires - Les MRAM

- SRAM : Rapidité

Les mémoires - Les MRAM

- SRAM : Rapidité
- DRAM : Densité

Les mémoires - Les MRAM

- SRAM : Rapidité
- DRAM : Densité
- Flash : Permanence

Les mémoires - Les MRAM

- SRAM : Rapidité
- DRAM : Densité
- Flash : Permanence
- Allier ces trois caractéristiques : MRAM

TDK

Basic Principles, Challenges and Opportunities of STT-MRAM for Embedded Memory Applications

Luc Thomas

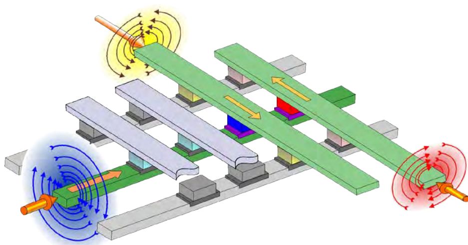
TDK- Headway Technologies,

463 S. Milpitas Boulevard, Milpitas CA 95035, USA

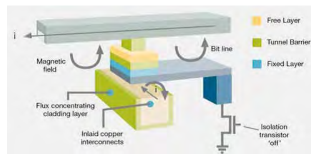
Magnetic Random Access Memories

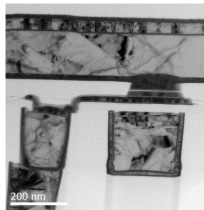
➔ More than 20 years ago: Field-MRAM

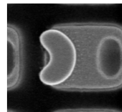
1st research program: IBM / Motorola (1995)
1st product: Freescale / Everspin (2006)



From S. Parkin and K. Roche IBM

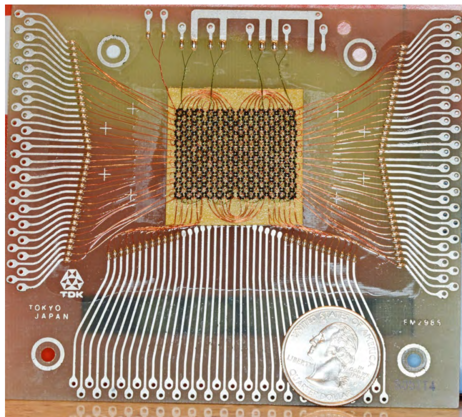




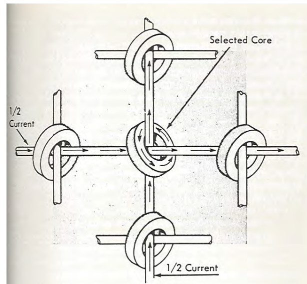


60 years ago: TDK first foray in MRAM technology

➔ TDK's 18x24 bit Magnetic Core Memory



Source: wikipedia.org/wiki/Magnetic-core_memory



Source: columbia.edu/cu/computinghistory/core.html

➔ MRAM was the predominant computer memory from the 50's to the 70's

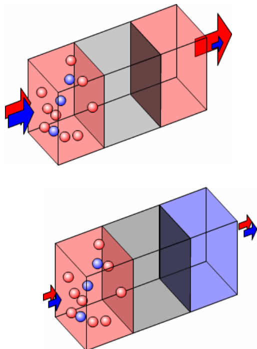
Outline

- ➔ **Basic principles of STT-MRAM**
- ➔ STT-MRAM integration
- ➔ STT-MRAM in emerging memory landscape

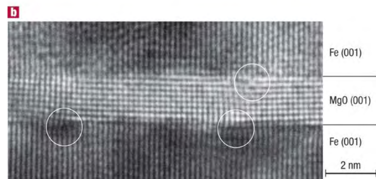
Magnetic Tunnel Junction (MTJ) device

➔ Two ferromagnetic electrodes separated by a thin MgO tunnel barrier

➔ Tunnel Magnetoresistance (TMR): device resistance depends on the relative orientation of the magnetization of the two magnetic electrodes



From S. Parkin and K. Roche IBM

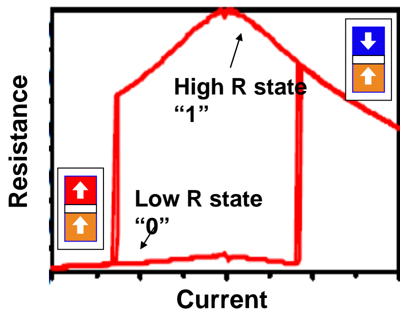


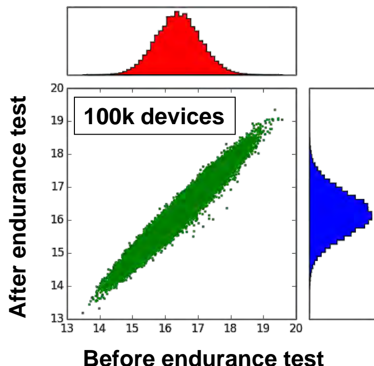
Yuasa et al. (AIST) Nature Materials2004

Reading with Tunnel Magnetoresistance

➔ Read operation by probing the resistance of the device at low voltage bias

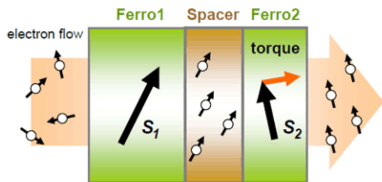
➔ True Binary device: no resistance drift of the 2 resistance state even after repeated cycling at maximum drive current





Writing with Spin-Transfer Torque

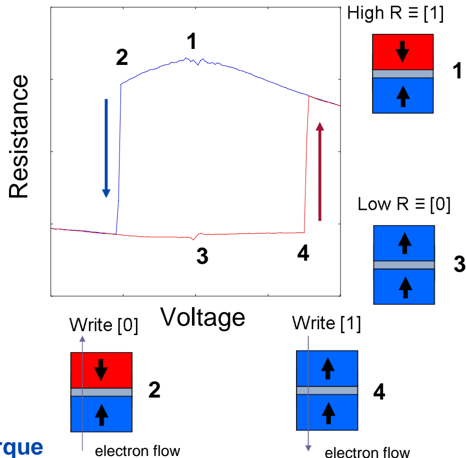
➔ Transfer of spin-angular momentum from polarized conduction electrons to electrodes magnetization



Reproduced from Quantumwise.com

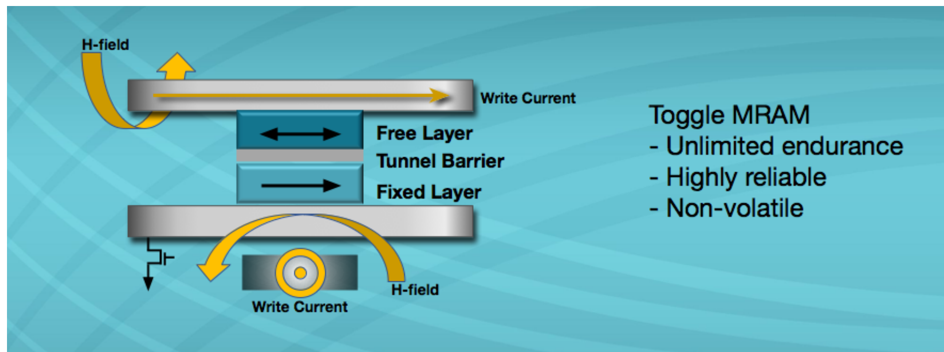
Phenomenon discovered in 1996 by two theoreticians:
John Slonczewski (IBM)
Luc Berger (Carnegie Mellon)

Read:
Tunnel Magnetoresistance



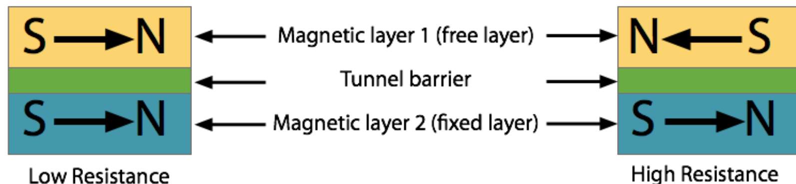
Write:
Spin Transfer Torque

Les mémoires - MRAM - Toggle RAM - MTJ (Magnetic Tunnel Junction)



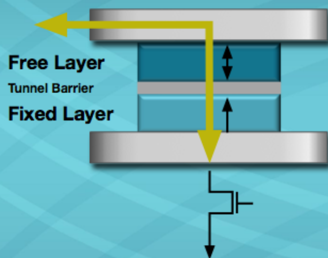
Everspin Technologies

Les mémoires - MRAM - Toogle RAM - MTJ (Magnetic Tunnel Junction)



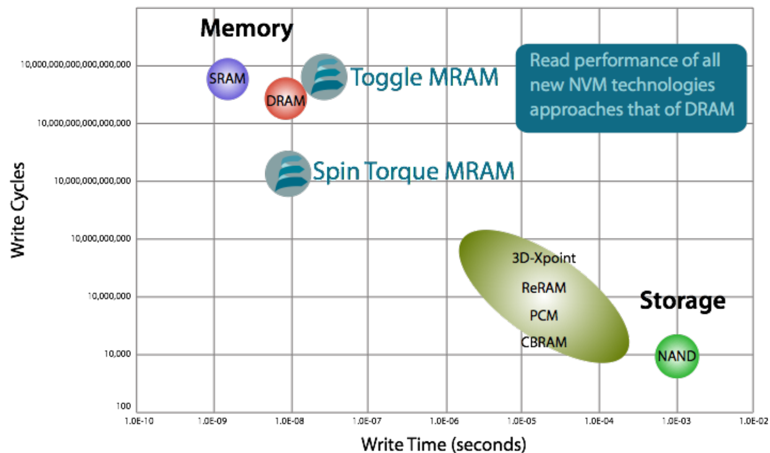
from Everspin Technologies

Les mémoires - MRAM - Spin Transfer Torque



Everspin Technologies

Les mémoires - MRAM- Spin Transfer Torque



Workshop on Memristive systems for Space applications
30 April 2015
ESTEC, Noordwijk, NL

Fundamentals of Memristors

Dirk J. Wouters and Eike Linn

RWTH Aachen, Institut für Werkstoffe der Elektrotechnik, Aachen, Germany

30 April 2015

IWE2 Institut für Werkstoffe der Elektrotechnik 2 | RWTH AACHEN UNIVERSITY

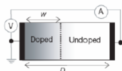
The day MEMRISTOR became famous...

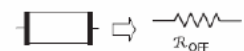
Vol 453 | 1 May 2008 | doi:10.1038/nature06932

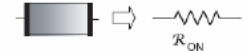
The missing memristor found

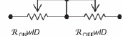
Dmitri B. Strukov[1], Gregory S. Snider[1], Duncan R. Stewart[1] & R. Stanley Williams[1]

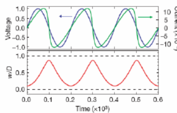
$$M(q) = \mathcal{R}_{OFF}\left(1 - \frac{\mu_V \mathcal{R}_{ON}}{D^2} q(t)\right)$$

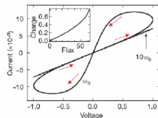












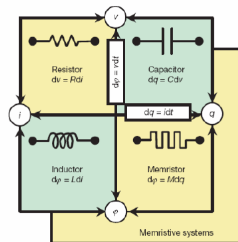
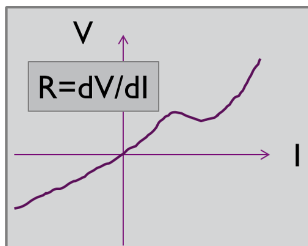
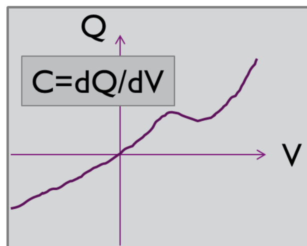


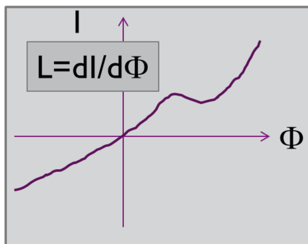
Figure 1 | The four fundamental two-terminal circuit elements: resistor, capacitor, inductor and memristor. Resistors and memristors are subsets of a more general class of dynamical devices, memristive systems. Note that R, C, L and M can be functions of the independent variable in their defining equations, yielding nonlinear elements. For example, a charge-controlled memristor is defined by a single-valued function $M(q)$.

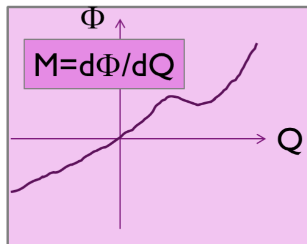
- RRAM seen as practical realisation of a theoretically predicted element
- Triggered a lot of interest especially in the EE (circuit design) world

Memristor = the missing 4th element









Memristor = the missing 4th element

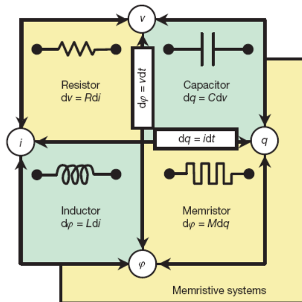
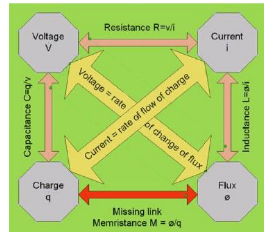
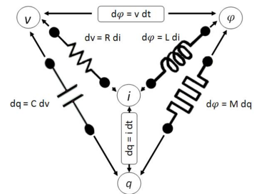


Figure 1 | The four fundamental two-terminal circuit elements: resistor, capacitor, inductor and memristor. Resistors and memristors are subsets of a more general class of dynamical devices, memristive systems. Note that R, C, L and M can be functions of the independent variable in their defining equations, yielding nonlinear elements. For example, a charge-controlled memristor is defined by a single-valued function $M(q)$.





Mathematical Description

- $M(q) = d\varphi/dq$

$$d\varphi/dt = M(q).dq/dt$$

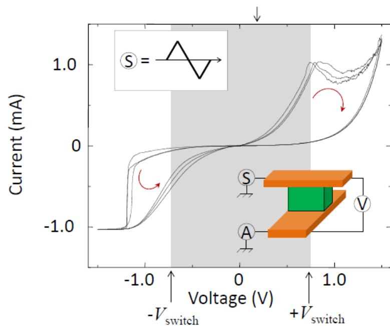
$$V = M(q).I$$

→M has dimension of resistance [Ohm]

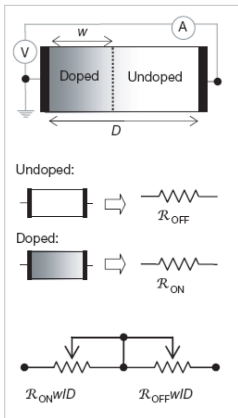
→M depending on integral of passes current : memory

→ name of MEM-RISTOR

Prototypical device = (TiO2 based) RRAM



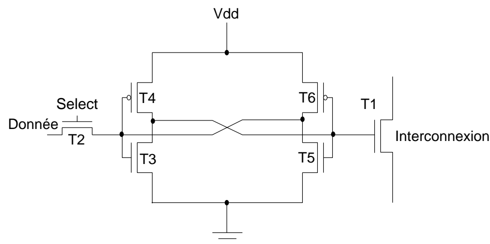
Device Model & Formula‘s



- Total R = series connection
  - $M(t) = R_{ON}.W(t)/D + R_{OFF}.(1\text{-}W(t)/D))$
  - $W(t)$ = state variable
    - $dW(t)/dt = \mu.E$ (drift of ions)
    - $E = R_{ON}.i(t)/D$ (relation E-i)
    - W = function of the amount of charges passed through the device
  - **$M(q) = R_0 - m.R_{ON}.\Delta R.q(t)/D^2$**
    - $R_0 = R_{ON}.W_0/D + R_{OFF}.(1\text{-}W_0/D)$

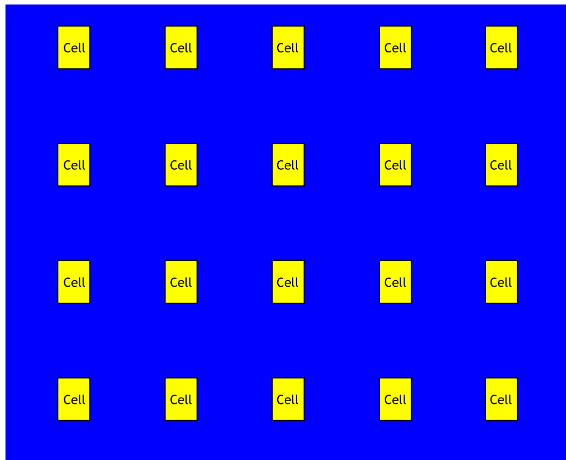
F.P.G.A et SRAM

- Utilisation de cellule de mémoire statique

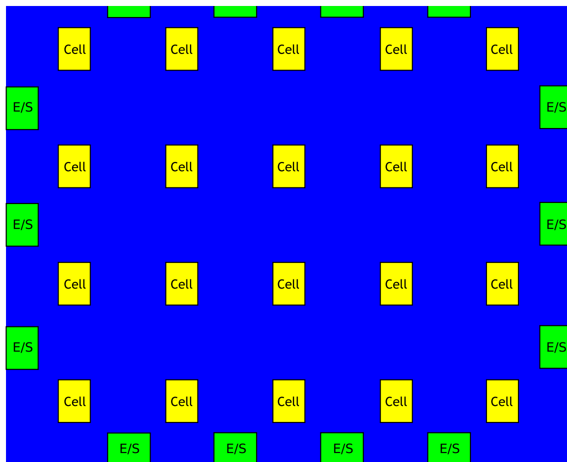


Structure d'un F.P.G.A

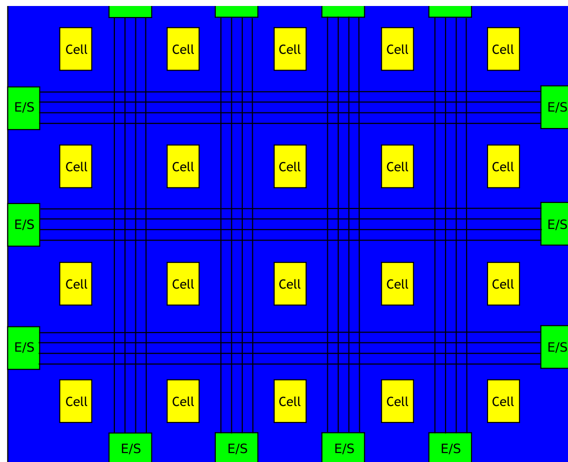
Structure d'un F.P.G.A



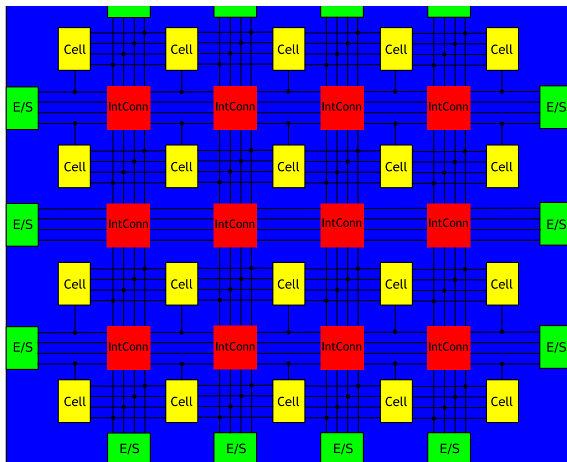
Structure d'un F.P.G.A



Structure d'un F.P.G.A



Structure d'un F.P.G.A



Qu'y a t'il dans un F.P.G.A ?

- Des Cellules de Base

Qu'y a t'il dans un F.P.G.A ?

- Des Cellules de Base
  - LUT (LookUp Table)- *Codage des fonctions* - **Combinatoire**

Qu'y a t'il dans un F.P.G.A ?

- Des Cellules de Base
  - LUT (LookUp Table)- *Codage des fonctions* - **Combinatoire**
  - Bascules - *Synchronisation* - **Séquentiel**

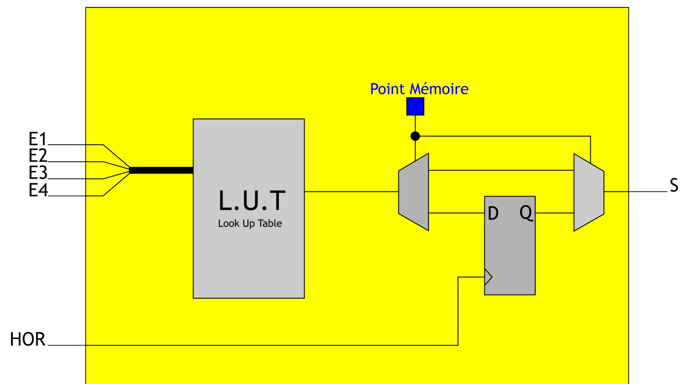
Qu’y a t’il dans un F.P.G.A ?

- Des Cellules de Base
  - LUT (LookUp Table)- *Codage des fonctions* - **Combinatoire**
  - Bascules - *Synchronisation* - **Séquentiel**
- Des Entrées-Sorties

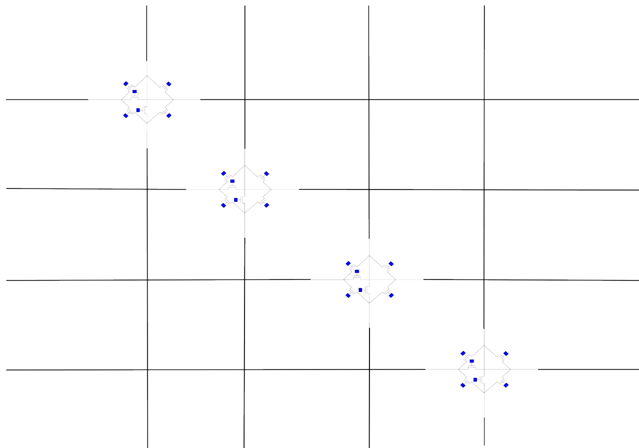
Qu’y a t’il dans un F.P.G.A ?

- Des Cellules de Base
  - LUT (LookUp Table)- *Codage des fonctions* - **Combinatoire**
  - Bascules - *Synchronisation* - **Séquentiel**
- Des Entrées-Sorties
- De la logique de routage

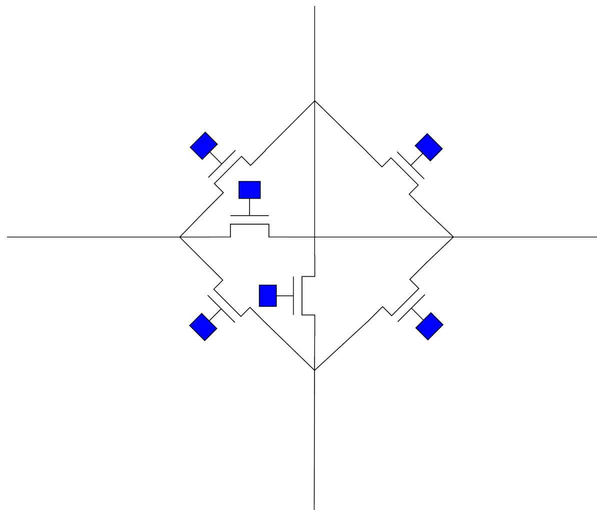
Structure d'une Cellule de Base



Structure de l’interconnexion



Structure de l'interconnexion



FPGA

- Présentation des FPGA commerciaux
- Deux principaux fabricants
  - Xilinx
  - Altera

Altera FPGA Devices

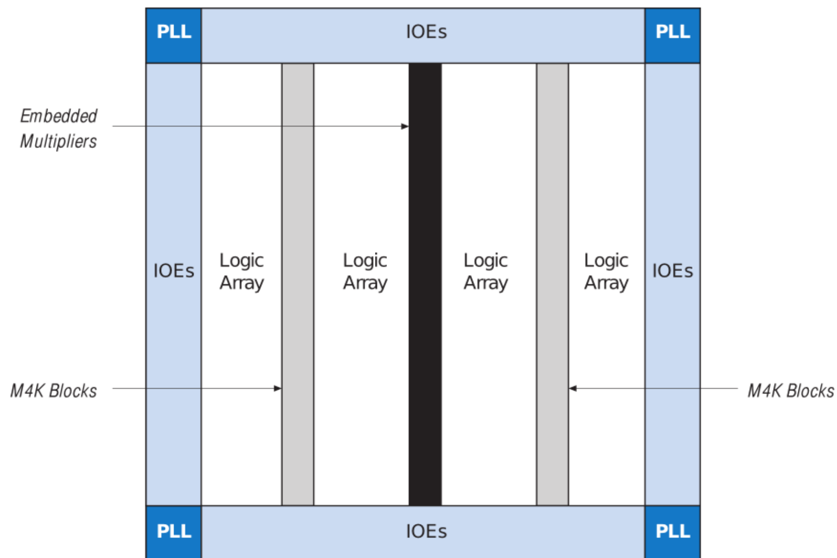
Technology	Low-cost	Mid-range	High-performance
130 nm	Cyclone		Stratix
90 nm	**Cyclone II**		Stratix II
65 nm	Cyclone III	Arria I	**Stratix III**
40 nm	**Cyclone IV**	Arria II	**Stratix IV**

Les FPGA du fabricant Altera

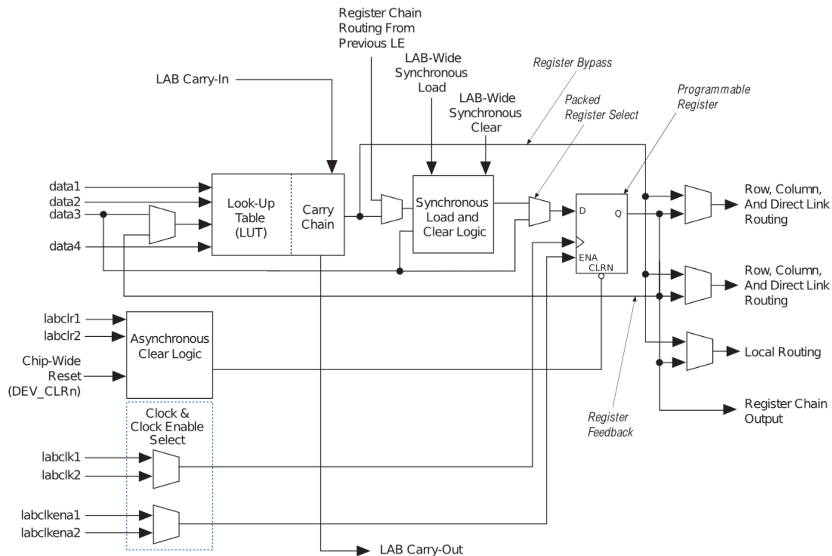
Technologie	Bas coût	Intermédiaire	hautes performances
130 nm	Cyclone		Stratix
90 nm	Cyclone II		Stratix II
65 nm	Cyclone III	Arria I	Stratix III
40 nm	Cyclone IV	Arria II	Stratix IV
28 nm	Cyclone V	Arria V	Stratix V
20 nm		Arria X	
14 nm			Stratix X

Low-cost Altera FPGAs

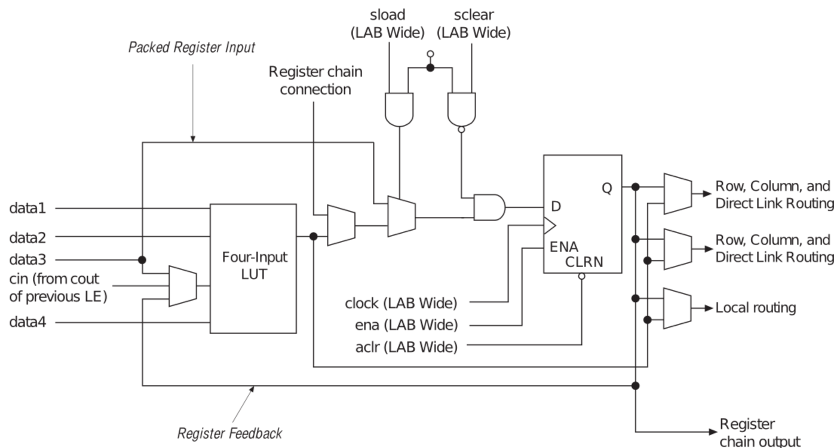
Cyclone II



Cyclone II - Logic Element (Cellule de base)



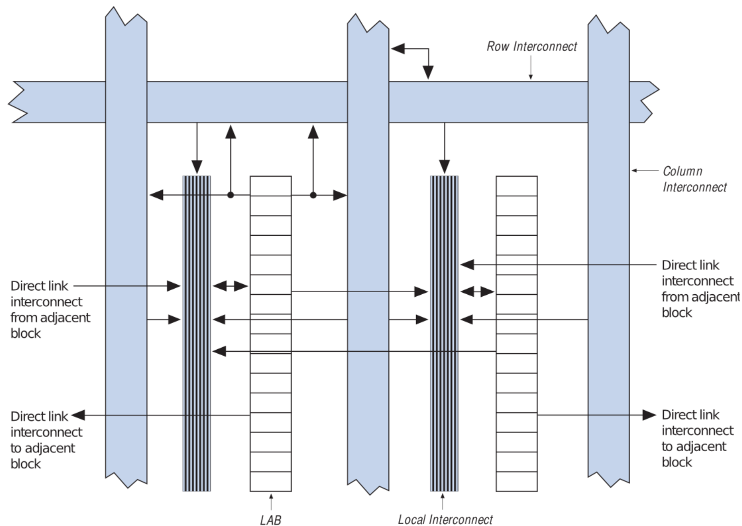
Cyclone II - Logic Element - Mode Normal



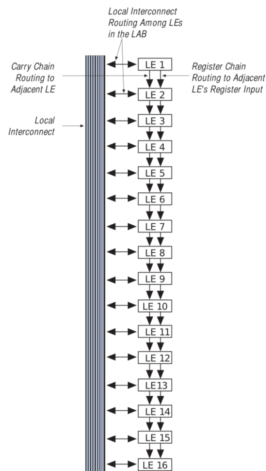
Cyclone II - Logic Element - Mode Arithmétique

sload (LAB Wide)
sclear (LAB Wide)
Register chain connection
data1
data2
Three-Input LUT
Three-Input LUT
cin (from cout of previous LE)
cout
D
Q
ENA
CLRN
clock (LAB Wide)
ena (LAB Wide)
aclr (LAB Wide)
Row, column, and direct link routing
Row, column, and direct link routing
Local routing
Register chain output
Register Feedback

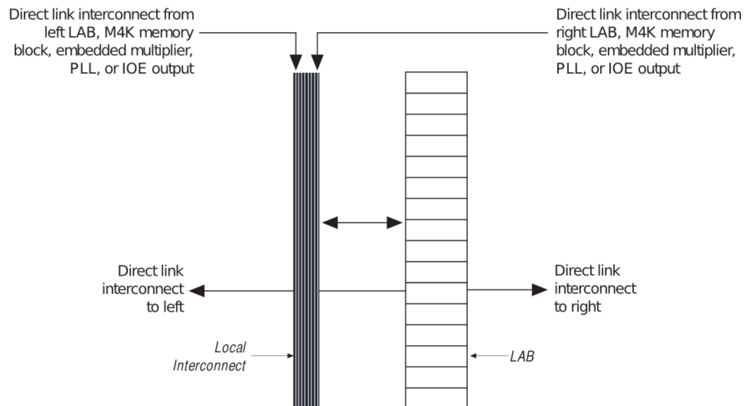
Cyclone II - LAB



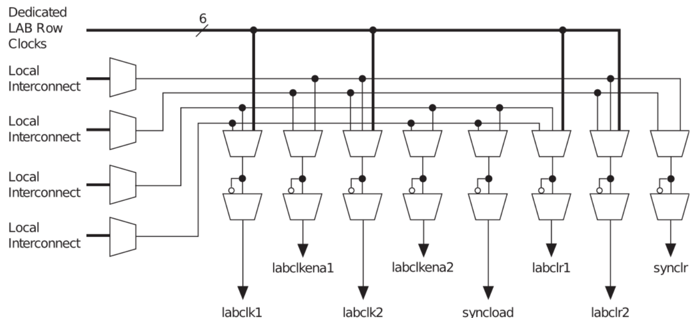
Cyclone II - LAB - Chainage Arithmétique



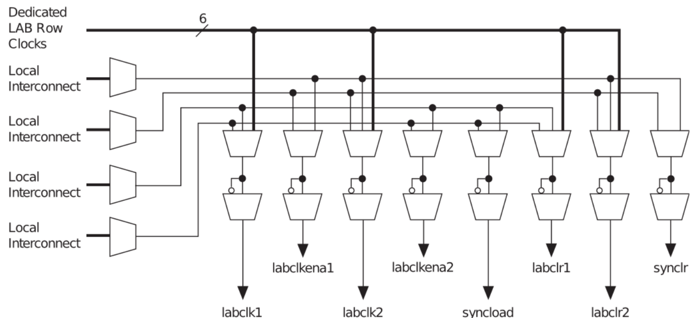
Cyclone II - LAB - Directlink



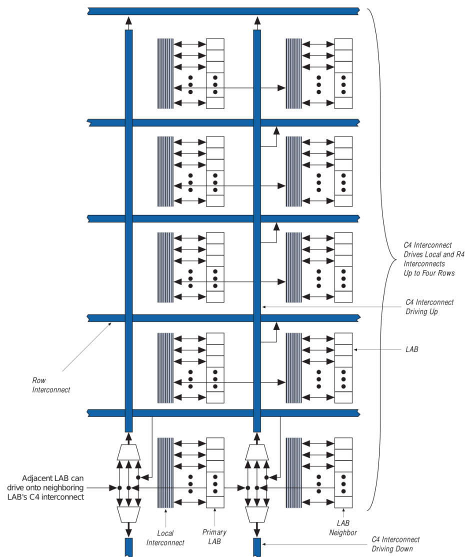
Cyclone II - LAB - Wide



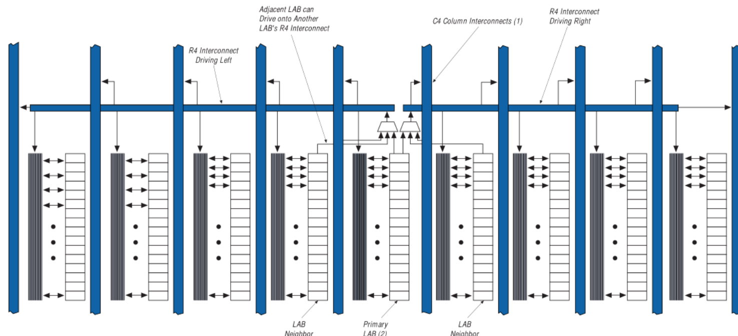
Cyclone II - LAB - Wide



Cyclone II - LAB - Interconnexion Colonne



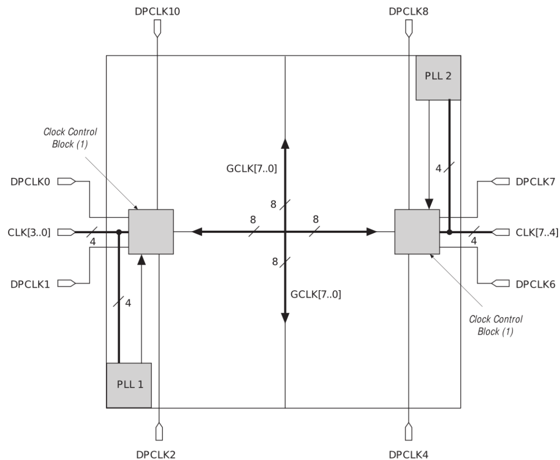
Cyclone II - LAB - Interconnexion Ligne



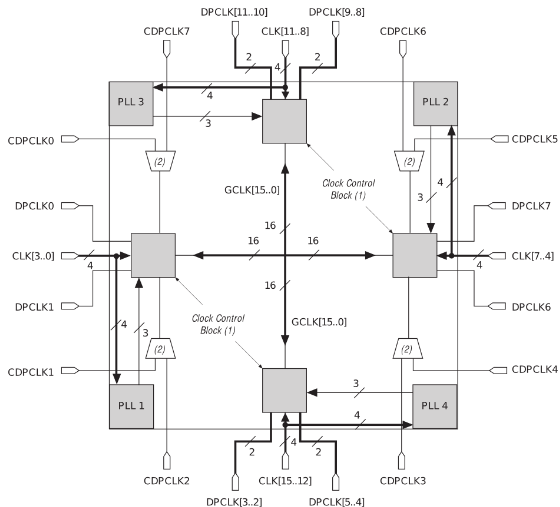
Cyclone II - LAB - Matrice Interconnexion

Source	Destination												
	Register Chain	Local Interconnect	Direct Link Interconnect	R4 Interconnect	R24 Interconnect	C4 Interconnect	C16 Interconnect	LE	M4K RAM Block	Embedded Multiplier	PLL	Column IOE	Row IOE
Register Chain								v					
Local Interconnect								v	v	v	v	v	v
Direct Link Interconnect		v											
R4 Interconnect		v		v	v	v	v						
R24 Interconnect				v	v	v	v						
C4 Interconnect		v		v	v	v	v						
C16 Interconnect				v	v	v	v						
LE	v	v	v	v		v							
M4K memory Block		v	v	v		v							
Embedded Multipliers		v	v	v		v							
PLL			v	v		v							
Column IOE						v	v						
Row IOE			v	v	v	v							

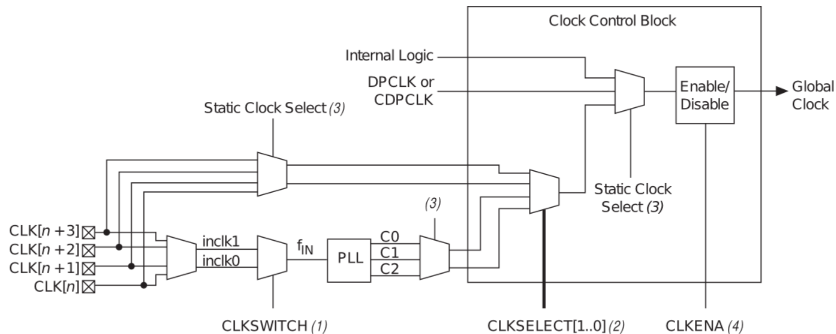
Cyclone II - LAB - Horloge



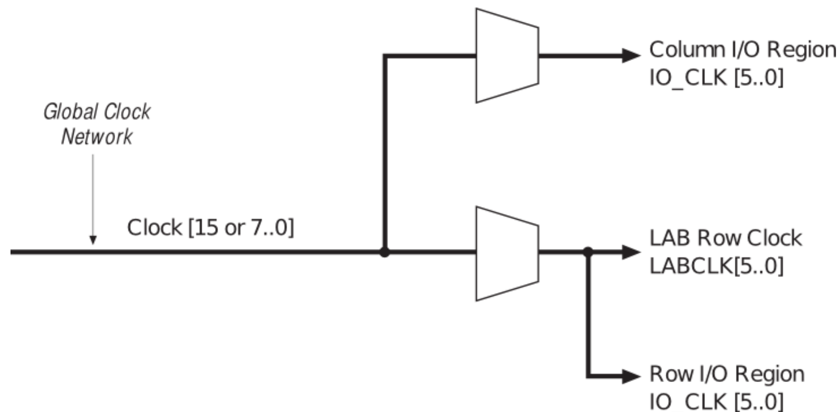
Cyclone II - LAB - Horloge



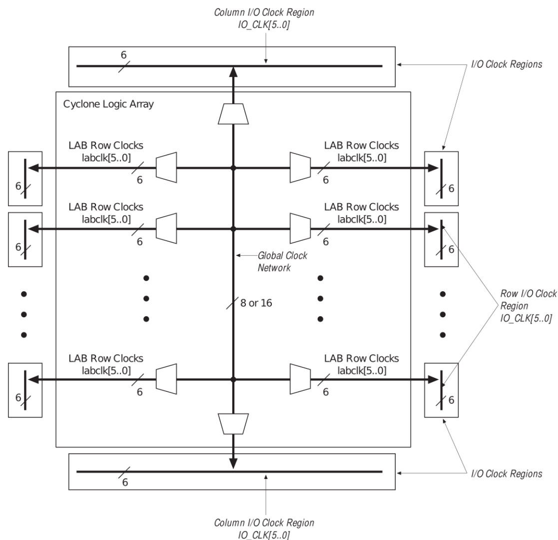
Cyclone II - LAB - Horloge



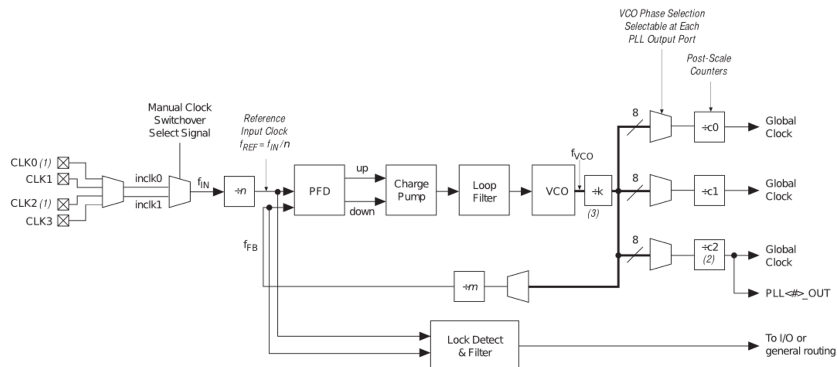
Cyclone II - LAB - Horloge



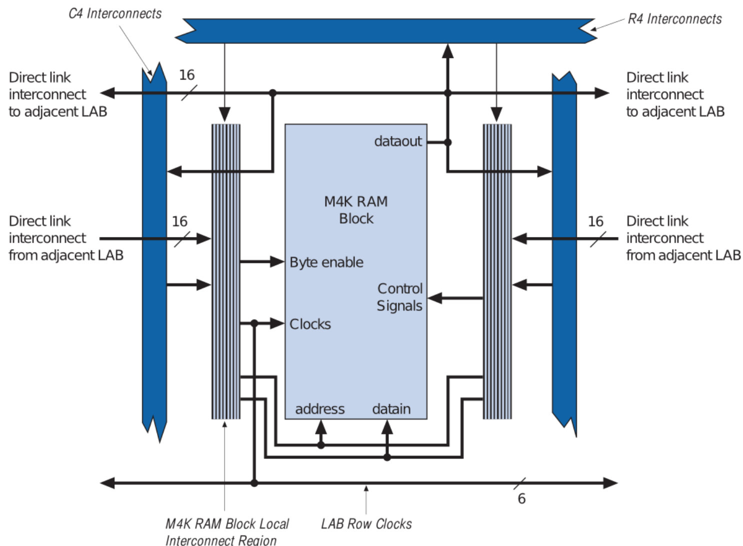
Cyclone II - LAB - Horloge



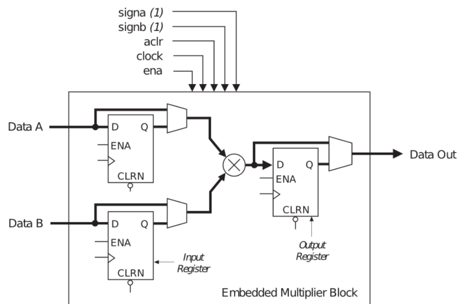
Cyclone II - LAB - PLL



Cyclone II - LAB - Mémoire M4K

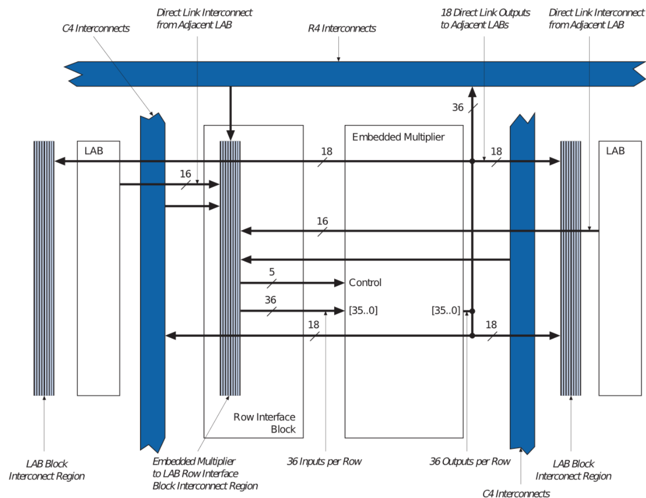


Cyclone II - LAB - Multiplier

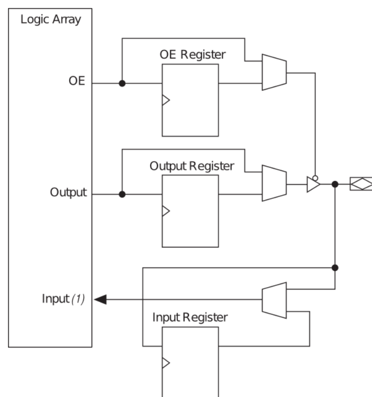


Data A (signa Value)	Data B (signb Value)	Result
Unsigned	Unsigned	Unsigned
Unsigned	Signed	Signed
Signed	Unsigned	Signed
Signed	Signed	Signed

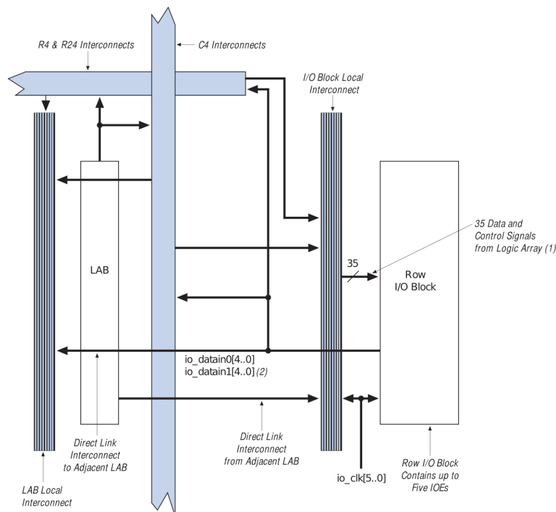
Cyclone II - LAB - Multiplier interconnexion LAB



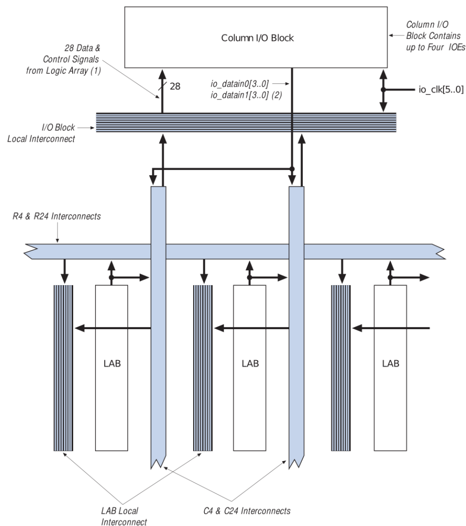
Cyclone II - Entrées/Sorties - Structure



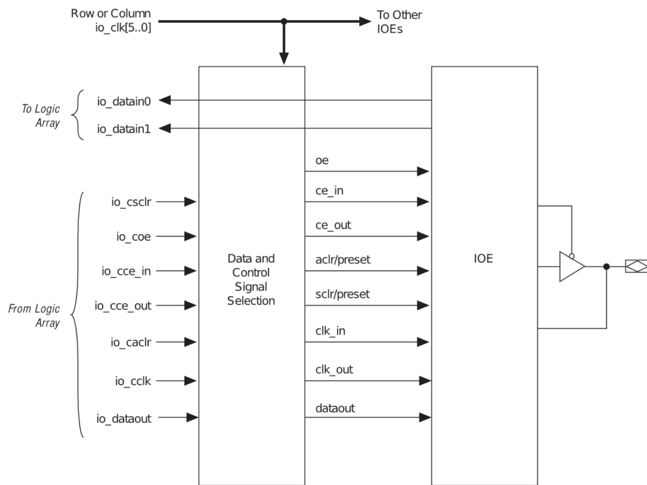
Cyclone II - Entrées/Sorties - interconnexion ligne



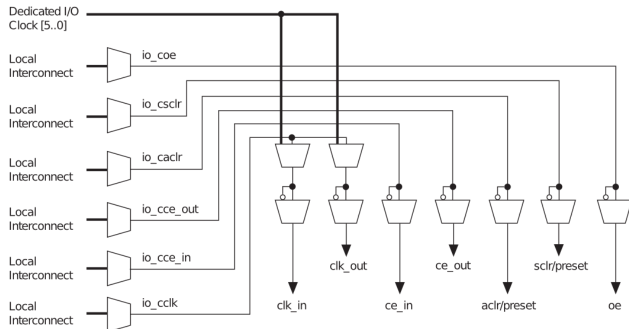
Cyclone II - Entrées/Sorties - interconnexion colonne



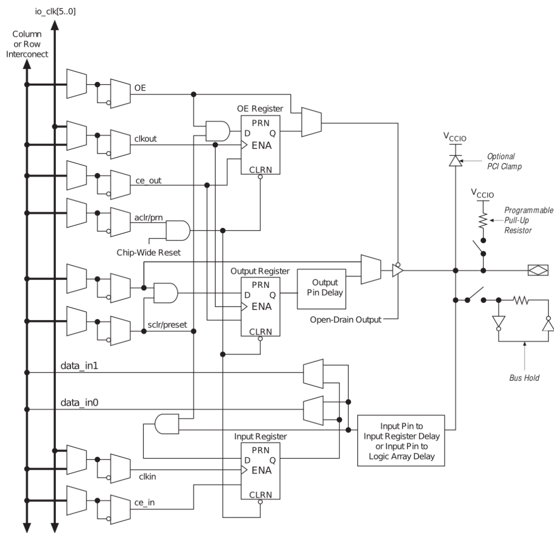
Cyclone II - Entrées/Sorties - distribution signaux commandes



Cyclone II - Entrées/Sorties - signaux contrôle



Cyclone II - Entrées/Sorties - détail



Cyclone II - Entrées/Sorties - types

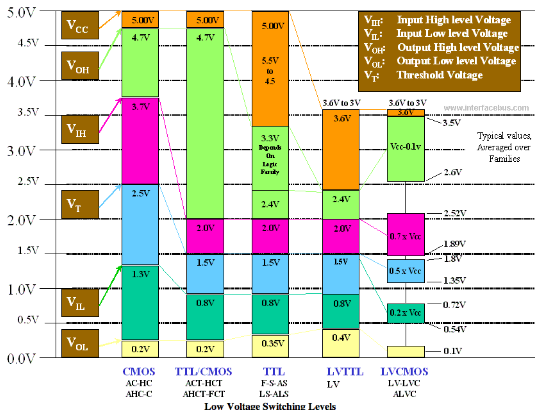
I/O Standard	Type	V_{CCIO} Level		Top and Bottom I/O Pins		Side I/O Pins		
		Input	Output	CLK, DQS	User I/O Pins	CLK, DQS	PLL_OUT	User I/O Pins
3,3V LVTTL/LVCMOS	Single Ended	3,3V/2,5V	3,3 V	x	x	x	x	x
2,5V LVTTL/LVCMOS	Single Ended	3,3V/2,5V	2,5 V	x	x	x	x	x
1,8V LVTTL/LVCMOS	Single Ended	1,8V/1,5V	1,8 V	x	x	x	x	x
1,5V LVCMOS	Single Ended	1,8V/1,5V	1,5 V	x	x	x	x	x
SSTL-2 Class I	Voltage referenced	2,5V	2,5 V	x	x	x	x	x
SSTL-2 Class II	Voltage referenced	2,5V	2,5 V	x	x	x	x	x
SSTL-18 Class I	Voltage referenced	1,8 V	1,8 V	x	x	x	x	x
SSTL-18 Class II	Voltage referenced	1,8 V	1,8 V	x	x			
HSTL-18 Class I	Voltage referenced	1,8 V	1,8 V	x	x	x	x	x
HSTL-18 Class II	Voltage referenced	1,8 V	1,8 V	x	x			
HSTL-15 Class I	Voltage referenced	1,5 V	1,5 V	x	x	x	x	x
HSTL-15 Class II	Voltage referenced	1,5 V	1,5 V	x	x			
PCI and PCI-X	Single Ended	3,3 V	3,3 V	-	-	x	x	x
Differential SSTL-2 class I or II	Pseudo differential		2,5V				x	
Differential SSTL-18 class I or II	Pseudo differential		2,5V				x	
Differential HSTL-15 class I or II	Pseudo differential		2,5V				x	
Differential HSTL-18 class I or II	Pseudo differential		2,5V				x	
LVDS	Differential	2,5V	2,5V	x	x	x	x	x
RSDS and mini-LVDS	Differential		2,5V	-	x	-	x	x
LVPECL	Differential	3,3V/2,5V 1,8V/1,5V	x		x			

LVTTL

Low Voltage Transistor Transistor Level

LVCMOS

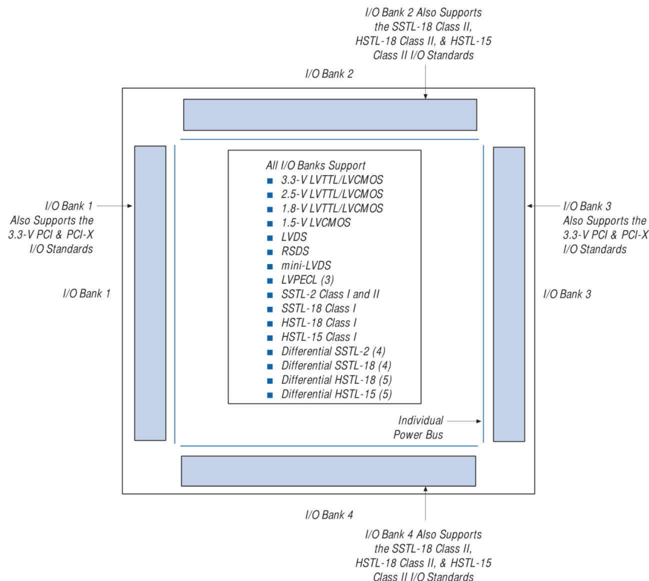
Low Voltage Complementary Metal Oxyde Silicium



SSTL

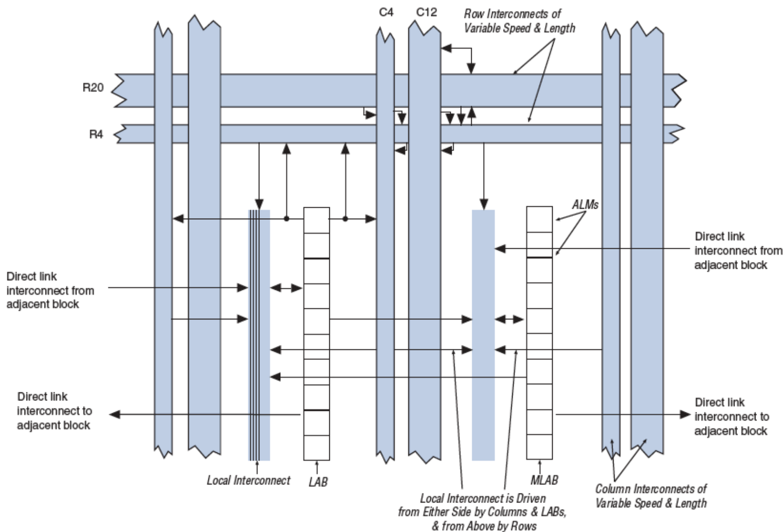
Stub Series Terminated Logic est une interface couramment utilisée pour connecter des mémoires de type DDR. Il existe plusieurs interfaces SSTL : SSTL-3, SSTL-2 et SSTL-18, elles sont normalisée par un document JEDEC.

Cyclone II - Entrées/Sorties - bancs EP2C5 et EP2C8

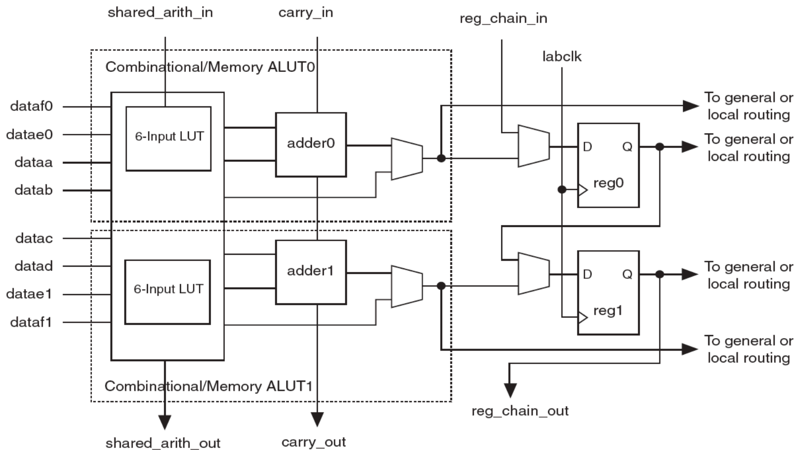


High-Performance Altera FPGAs

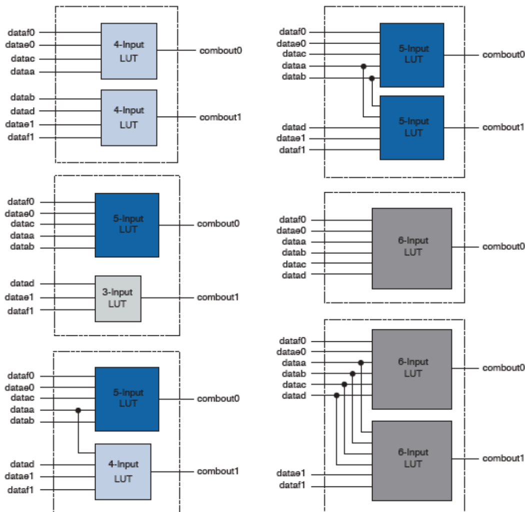
Stratix III Logic Array Blocks (LABs)



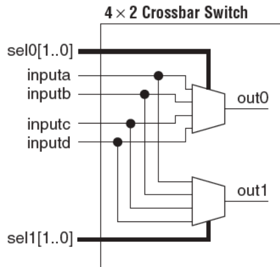
High-Level Block Diagram of the Stratix III ALM

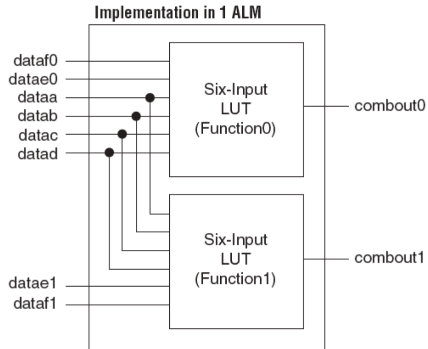


Altera Stratix III
Adaptive Logic Modules (ALM) – Normal Mode

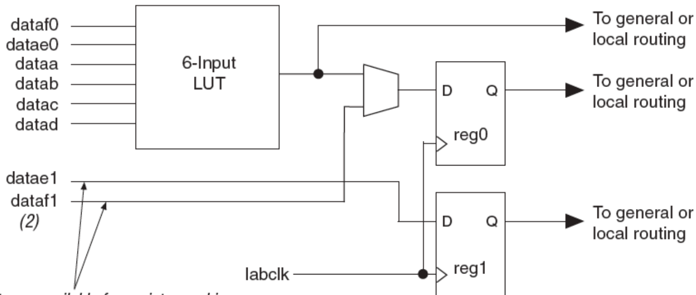


4 × 2 Crossbar Switch Example

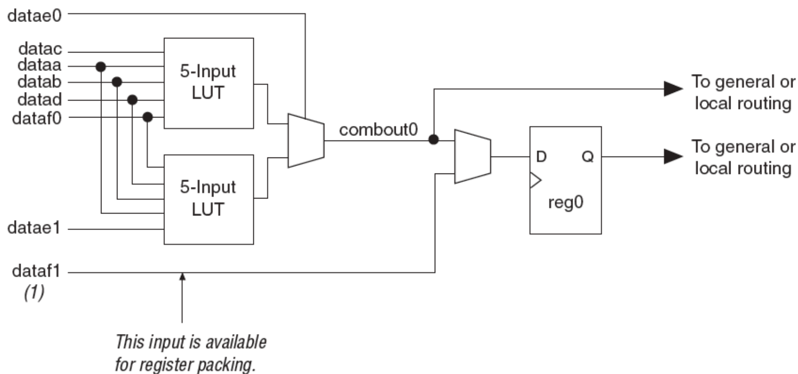




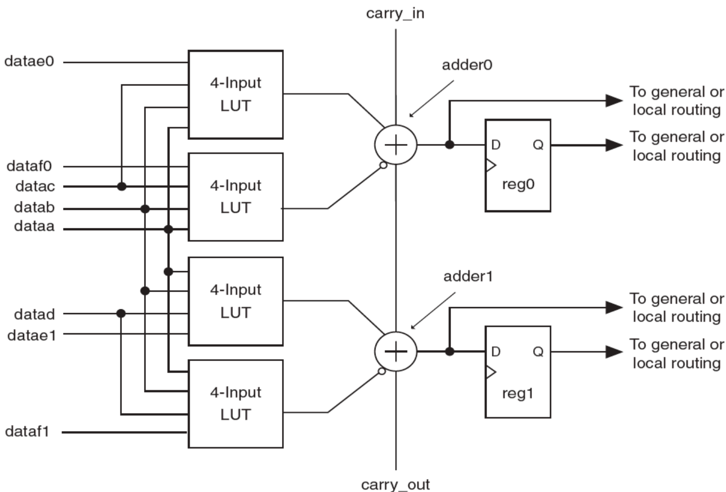
Register Packing



Template for Seven-Input Functions Supported in Extended LUT Mode

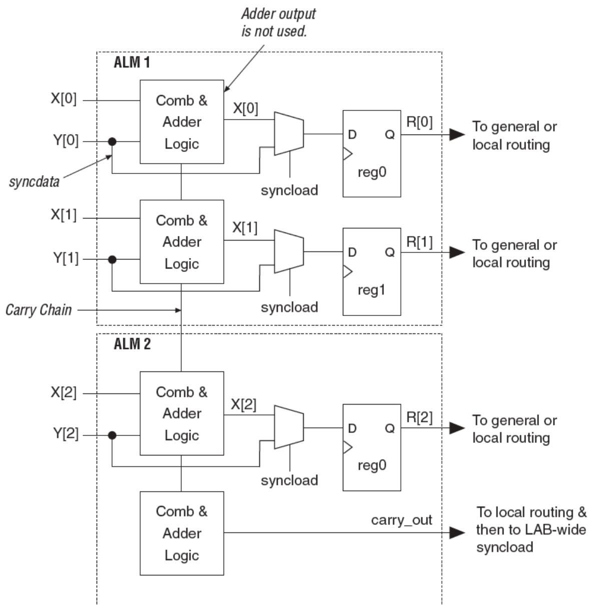


Altera Stratix III, Stratix IV
Adaptive Logic Modules (ALM) – Arithmetic Mode



Performing Operation

R = (X < Y) ? Y : X



Three Operand Addition
Utilizing Shared Arithmetic Mode

3-Bit Add Example

1st stage add is implemented in LUTs.

X3 X2 X1 X0
Y3 Y2 Y1 Y0
\+ Z3 Z2 Z1 Z0

2nd stage add is implemented in s.

S3 S2 S1 S0
\+ C3 C2 C1 C0

R4 R3 R2 R1 R0

Binary Add	Decimal Equivalents
1 1 1 0	14
0 1 0 0	4
+ 1 1 0 1	+ 13
0 1 1 1	7
+ 1 1 0 0	+ 2 x 12
1 1 1 1 1	31

shared_arith_in = '0'
carry_in = '0'

ALM Implementation

ALM 1

3-Input LUT S0
3-Input LUT C0
X0
Y0
Z0
R0
X1
Y1
Z1
3-Input LUT S1
3-Input LUT C1
R1

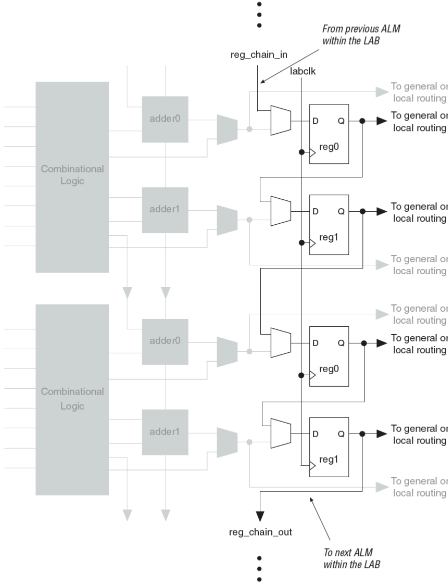
ALM 2

3-Input LUT S2
3-Input LUT C2
X2
Y2
Z2
R2
X3
Y3
Z3
3-Input LUT S3
3-Input LUT C3
R3
R4

LUT-Register Mode

clk
aclr
datain(datac)
sclr
4-input LUT
5-input LUT
sumout
combout
sumout
combout
LUT regout

Register Chain



Example of Resource Utilization Report (1)

```
+-----------------------------------------------------------------------------+
; Fitter Resource Usage Summary                                               ;
+-------------------------------------------------+---------------------------+
; Resource                                        ; Usage                     ;
+-------------------------------------------------+---------------------------+
; ALUTs Used                                      ; 415 / 38,000 ( 1 % )      ;
;     -- Combinational ALUTs                      ; 415 / 38,000 ( 1 % )      ;
;     -- Memory ALUTs                             ; 0 / 19,000 ( 0 % )        ;
;     -- LUT_REGs                                 ; 0 / 38,000 ( 0 % )        ;
; Dedicated logic registers                       ; 136 / 38,000 ( < 1 % )    ;
;                                                 ;                           ;
; Combinational ALUT usage by number of inputs    ;                           ;
;     -- 7 input functions                        ; 0                         ;
;     -- 6 input functions                        ; 287                       ;
;     -- 5 input functions                        ; 0                         ;
;     -- 4 input functions                        ; 24                        ;
;     -- <=3 input functions                      ; 104                       ;
;                                                 ;                           ;
; Combinational ALUTs by mode                     ;                           ;
;     -- normal mode                              ; 335                       ;
;     -- extended LUT mode                        ; 0                         ;
;     -- arithmetic mode                          ; 80                        ;
;     -- shared arithmetic mode                   ; 0                         ;
```

Example of Resource Utilization Report (2)

```
; Logic utilization                                      ; 701 / 38,000 ( 2 % )  ;
;     -- Difficulty Clustering Design                    ; Low                   ;
;     -- Combinational ALUT/register pairs used
         in final Placement                              ; 476                   ;
;          -- Combinational with no register             ; 340                   ;
;          -- Register only                              ; 61                    ;
;          -- Combinational with a register              ; 75                    ;
;     -- Estimated pairs recoverable by pairing ALUTs and registers
         as design grows                                 ; -54                   ;
;     -- Estimated Combinational ALUT/register pairs
         unavailable                                     ; 279                   ;
;          -- Unavailable due to Memory LAB use          ; 0                     ;
;          -- Unavailable due to unpartnered 7 LUTs      ; 0                     ;
;          -- Unavailable due to unpartnered 6 LUTs      ; 279                   ;
;          -- Unavailable due to unpartnered 5 LUTs      ; 0                     ;
;          -- Unavailable due to LAB-wide signal
             conflicts                                   ; 0                     ;
;          -- Unavailable due to LAB input limits        ; 0                     ;
```

Example of Resource Utilization Report (3)

```
; Total registers*                                      ; 136                      ;
;     -- Dedicated logic registers                      ; 136 / 38,000 ( < 1 % )   ;
;     -- I/O registers                                  ; 0 / 2,752 ( 0 % )        ;
;     -- LUT_REGs                                       ; 0                        ;
; ALMs:  partially or completely used                   ; 360 / 19,000 ( 2 % )     ;
; Total LABs:  partially or completely used             ; 42 / 1,900 ( 2 % )       ;
;     -- Logic LABs                                     ; 42 / 42 ( 100 % )        ;
;     -- Memory LABs                                    ; 0 / 42 ( 0 % )          ;
;                                                       ;                          ;
; User inserted logic elements                          ; 0                        ;
; Virtual pins                                          ; 0                        ;
; I/O pins                                              ; 20 / 488 ( 4 % )         ;
;     -- Clock pins                                     ; 5 / 16 ( 31 % )          ;
;     -- Dedicated input pins                           ; 0 / 12 ( 0 % )           ;
; Global signals                                        ; 2                        ;
; M9K blocks                                            ; 0 / 108 ( 0 % )          ;
; M144K blocks                                          ; 0 / 6 ( 0 % )            ;
; Total MLAB memory bits                                ; 0                        ;
; Total block memory bits                               ; 0 / 1,880,064 ( 0 % )    ;
; Total block memory implementation bits                ; 0 / 1,880,064 ( 0 % )    ;
; DSP block 18-bit elements                             ; 0 / 216 ( 0 % )          ;
; PLLs                                                  ; 0 / 4 ( 0 % )            ;
; Global clocks                                         ; 2 / 16 ( 13 % )          ;
```

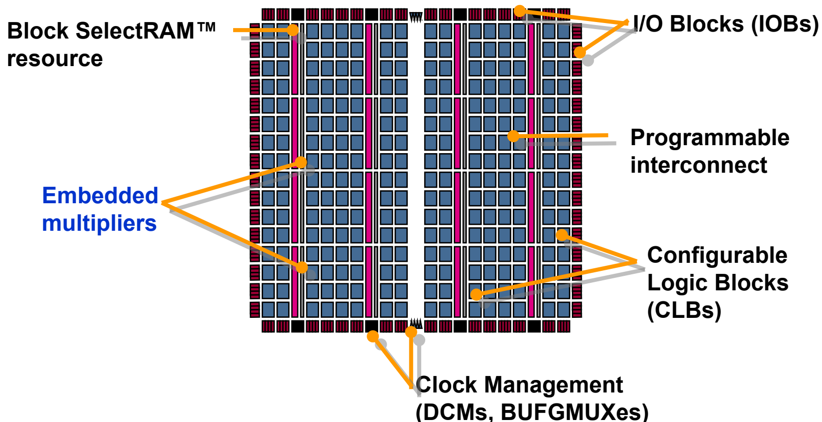
Overview

- All Xilinx FPGAs contain the same basic resources
  - Slices grouped into Configurable Logic Blocks (CLBs)
    - Contain combinatorial logic and register resources
  - IOBs
    - Interface between the FPGA and the outside world
  - Programmable interconnect
  - Other resources
    - Memory
    - Multipliers
    - Global clock buffers
    - Boundary scan logic

© 2009 Xilinx, Inc. All Rights Reserved
XILINX®
For Academic Use Only

Virtex-II Architecture

First family with Embedded Multipliers to enable high-performance DSP



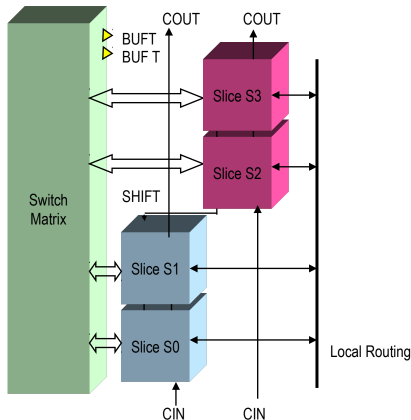
Refer to device data sheet at xilinx.com for detailed technical information

© 2009 Xilinx, Inc. All Rights Reserved
XILINX®
For Academic Use Only

CLBs and Slices

Combinatorial and sequential logic implemented here

- Each Virtex™-II CLB contains four slices
  - Local routing provides feedback between slices in the same CLB, and it provides routing to neighboring CLBs
  - A switch matrix provides access
    to general routing resources

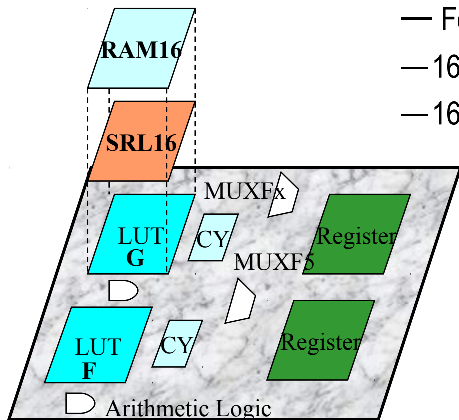


© 2009 Xilinx, Inc. All Rights Reserved

For Academic Use Only

XILINX®

Slice Resources



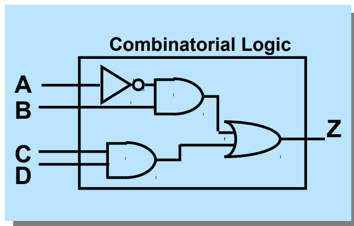
- **Each slice contains two:**
  - Four inputs lookup tables
  - 16-bit distributed SelectRAM
  - 16-bit shift register
- **Each register:**
  - D flip-flop
  - Latch
- **Dedicated logic:**
  - Muxes
  - Arithmetic logic
    - MULT_AND
    - Carry Chain

© 2009 Xilinx, Inc. All Rights Reserved

Look-Up Tables

- Combinatorial logic is stored in Look-Up Tables (LUTs)
  - Also called Function Generators (FGs)
  - Capacity is limited by the number of inputs, not by the complexity
- Delay through the LUT is constant

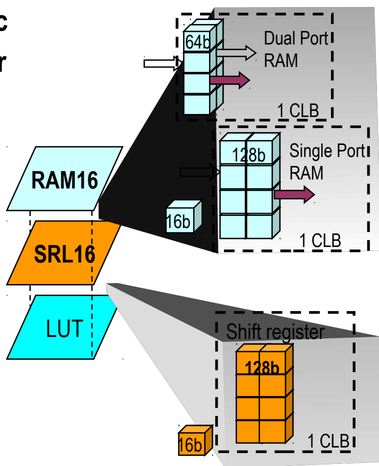
A	B	C	D	Z
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
	.	.	.	
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1



© 2009 Xilinx, Inc. All Rights Reserved

Distributed RAM

- **LUTs used as memory inside the fabric**
- **Flexible, can be used as RAM, ROM, or shift register**
- **Distributed memory with fast access time**
- **Cascadable with built-in CLB routing**
- **Applications**
  - Linear feedback shift register
  - Distributed arithmetic
  - Time-shared registers
  - Small FIFO
  - Digital delay lines (Z^{-1})



XILINX

© 2009 Xilinx, Inc. All Rights Reserved

For Academic Use Only

The SRL16E

- **The 16 SRAM cells have been organized into a shift register**
  - The 'CE' is used, in conjunction with the clock, to write data into the first flip-flop and for all other data to move right by one position
  - Because this is a predictable operation, no address is required for writing
- **The SRL16E is excellent in implementing efficient DSP Functions**
  - A very efficient way to delay data samples
  - Shifting samples and scanning at faster rate

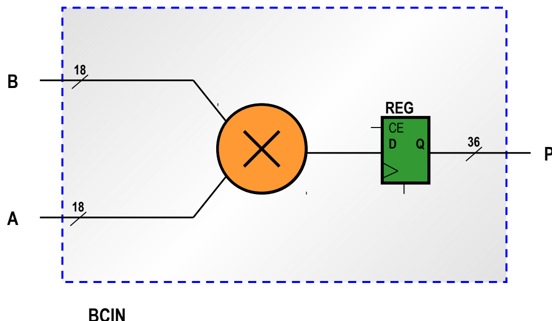
© 2009 Xilinx, Inc. All Rights Reserved

For Academic Use Only

XILINX®

Enabling high-performance DSP

Virtex-II introduced the embedded 18x18 multiplier



- Situated between the Block RAMs and CLB array to enable high-performance multiply-accumulate operations
- This dramatically increased multiplier speed and density compared to LUT based multipliers and enabled FPGA based DSP

XILINX®
© 2009 Xilinx, Inc. All Rights Reserved
For Academic Use Only

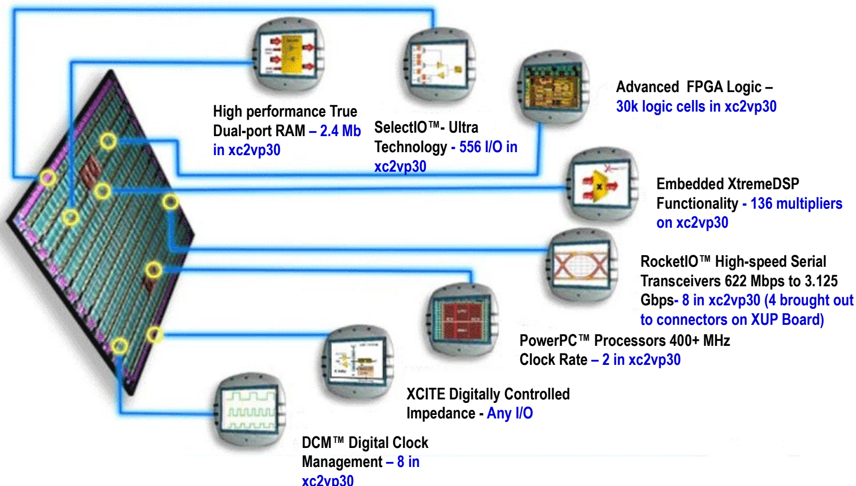
Outline

- Power of Parallelism
- Basic FPGA Architecture
- **Virtex™ -II Pro**
- Virtex-4
- Virtex-5
- Spartan™-3 Family
- Latest Families
  - Virtex-6 Family
  - Spartan-6 Family
- Why should I use FPGAs for DSP?
- The DSP48 Slice Advantage

© 2009 Xilinx, Inc. All Rights Reserved

Virtex-II Pro FPGAs

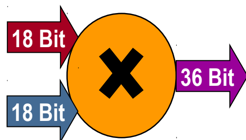
Refer to device data sheet on web for detailed technical information



© 2009 Xilinx, Inc. All Rights Reserved
For Academic Use Only
XILINX®

Multiplier Unit

- Embedded 18-bit x 18-bit multiplier
- The XUP Virtex-II Pro includes a Virtex-II Pro xc2vp30 device with 136 Multipliers
- 2s complement signed operation
- 4- to 18-bit operands
- Combinational & pipelined options
- Operates with block RAM and fabric to implement MAC function



300 MHz Performance in Virtex-II Pro

Pipelined multiplier with registered inputs and outputs

© 2009 Xilinx, Inc. All Rights Reserved
For Academic Use Only
XILINX®

Virtex-4 Family

Advanced Silicon Modular BLock (ASMBL) Architecture
Optimized for logic, Embedded, and Signal Processing

Resource	LX	FX	SX
Logic	14K–200K LCs	12K–140K LCs	23K–55K LCs
Memory	0.9–6 Mb	0.6–10 Mb	2.3–5.7 Mb
DCMs	4–12	4–20	4–8
DSP Slices	32–96	32–192	128–512
SelectIO	240–960	240–896	320–640
RocketIO	N/A	0–24 Channels	N/A
PowerPC	N/A	1 or 2 Cores	N/A
Ethernet MAC	N/A	2 or 4 Cores	N/A

XILINX®

© 2009 Xilinx, Inc. All Rights Reserved
For Academic Use Only

Virtex-4 Features

Refer to device data sheet on web for detailed technical information

200,000
Logic Cells

500 MHz Xesium™
Differential Clocking

BRAM with FIFO
& ECC

AES Secure Chip
Design Security

0.6-6.5 Gbps
RocketIO™
Transceivers

1 Gbps SelectIO™
with ChipSync™

Virtex-4

PowerPC®
Processor with APU

XtremeDSP™
Slice

Virtex-4

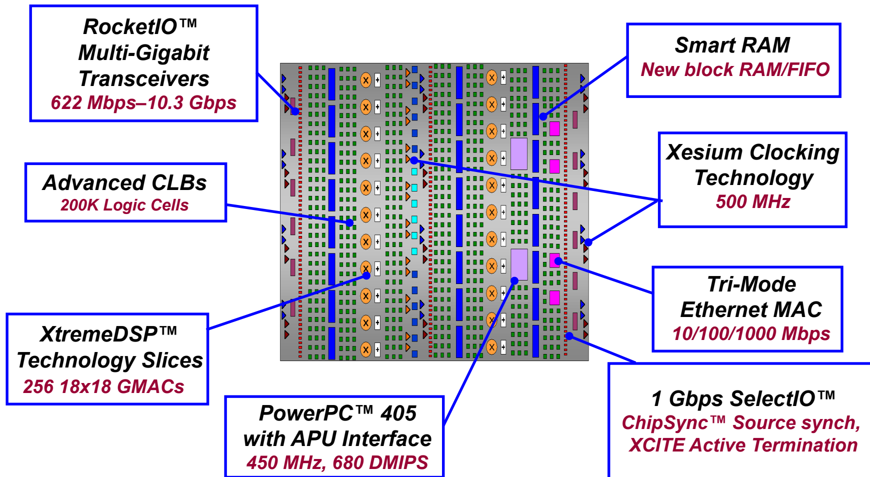
10/100/1000
Ethernet MAC

XILINX®

© 2009 Xilinx, Inc. All Rights Reserved

For Academic Use Only

Virtex-4 Architecture



XILINX®

© 2009 Xilinx, Inc. All Rights Reserved

For Academic Use Only

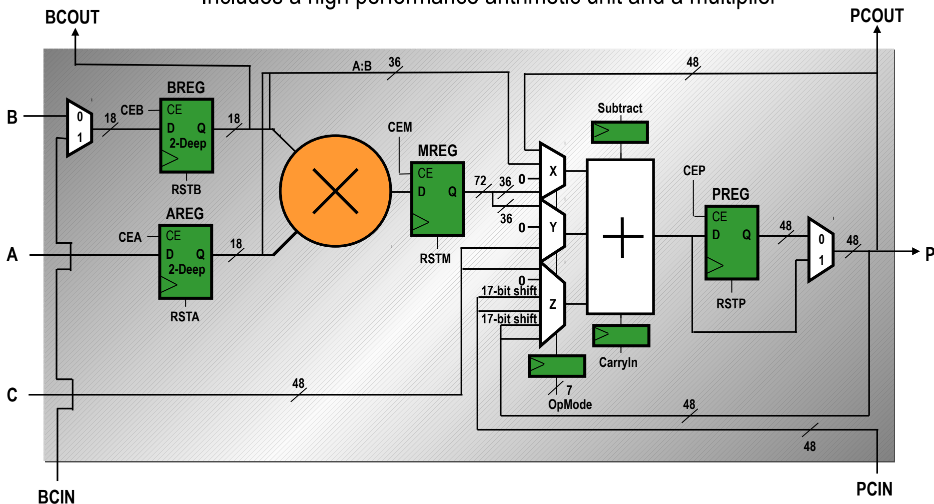
The Virtex-4 SX platform

- Virtex-4 introduced a new DSP block that had both multiply and accumulate functionality
- For the first time a true “MAC” unit was offered in a Xilinx FPGA. This block was called the DSP48 due to it’s 48-bit output precision
- Additional modes of the adder allowed subtract and shift functions to support scaling of results
- Integral registers guarantee high-speed pipelined data-paths for maximum clock frequency

© 2009 Xilinx, Inc. All Rights Reserved
XILINX
For Academic Use Only

DSP48 Block

Includes a high performance arithmetic unit and a multiplier



XILINX®

© 2009 Xilinx, Inc. All Rights Reserved

For Academic Use Only

DSP48 Block

Dynamically Programmable DSP Op Modes

OpMode	Z			Y		X		Output
	6	5	4	3	2	1	0	
Zero	0	0	0	0	0	0	0	+/- Cin
Hold P	0	0	0	0	0	1	0	+/- (P + Cin)
A:B Select	0	0	0	0	0	1	1	+/- (A:B + Cin)
Multiply	0	0	0	0	1	0	1	+/- (A * B + Cin)
C Select	0	0	0	1	1	0	0	+/- (C + Cin)
Feedback Add	0	0	0	1	1	1	0	+/- (C + P + Cin)
36-Bit Adder	0	0	0	1	1	1	1	+/- (A:B + C + Cin)
P Cascade Select	0	0	1	0	0	0	0	PCIN +/- Cin
P Cascade Feedback Add	0	0	1	0	0	1	0	PCIN +/- (P + Cin)
P Cascade Add	0	0	1	0	0	1	1	PCIN +/- (A:B + Cin)
P Cascade Multiply Add	0	0	1	0	1	0	1	PCIN +/- (A * B + Cin)
P Cascade Add	0	0	1	1	1	0	0	PCIN +/- (C + Cin)
P Cascade Feedback Add Ad	0	0	1	1	1	1	0	PCIN +/- (C + P + Cin)
P Cascade Add Add	0	0	1	1	1	1	1	PCIN +/- (A:B + C + Cin)
Hold P	0	1	0	0	0	0	0	P +/- Cin
Double Feedback Add	0	1	0	0	0	1	0	P +/- (P + Cin)
Feedback Add	0	1	0	0	0	1	1	P +/- (A:B + Cin)
Multiply-Accumulate	0	1	0	0	1	0	1	P +/- (A * B + Cin)
Feedback Add	0	1	0	1	1	0	0	P +/- (C + Cin)
Double Feedback Add	0	1	0	1	1	1	0	P +/- (C + P + Cin)
Feedback Add Add	0	1	0	1	1	1	1	P +/- (A:B + C + Cin)
C Select	0	1	1	0	0	0	0	C +/- Cin
Feedback Add	0	1	1	0	0	1	0	C +/- (P + Cin)
36-Bit Adder	0	1	1	0	0	1	1	C +/- (A:B + Cin)
Multiply-Add	0	1	1	0	1	0	1	C +/- (A * B + Cin)
Double	0	1	1	1	1	0	0	C +/- (C + Cin)
Double Add Feedback Add	0	1	1	1	1	1	0	C +/- (C + P + Cin)
Double Add	0	1	1	1	1	1	1	C +/- (A:B + C + Cin)

- Enables time-division multiplexing for DSP
- Over 40 different modes
- Each XtremeDSP Slice individually controllable
- Change operation in a single clock cycle
- Control functionality from logic, memory or processor

© 2009 Xilinx, Inc. All Rights Reserved

For Academic Use Only

XILINX®

DSP48 Block

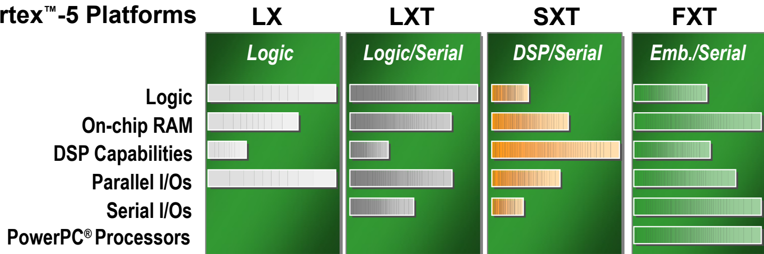
Useful For More Than DSP

- 6:1 high-speed, 36-bit Multiplexer
  - Use four XtremeDSP Slice and op-modes
  - 500 MHz performance using no programmable logic
    - Save 1584 LCs to build equivalent function in logic
- Dynamic 18-bit Barrel Shifter
  - Use two XtremeDSP slices
  - Use dedicated cascade routing and integrated 17-bit shift
    - Save 1449 LCs to build equivalent function in logic
- 36-bit Loadable Counter
  - Use a single XtremeDSP slice, achieve 500 MHz performance
    - Save 540 LCs to build equivalent function in logic

XILINX®
© 2009 Xilinx, Inc. All Rights Reserved
For Academic Use Only

Virtex-5 Family

Optimized for logic, Embedded, Signal Processing, and High-Speed Connectivity

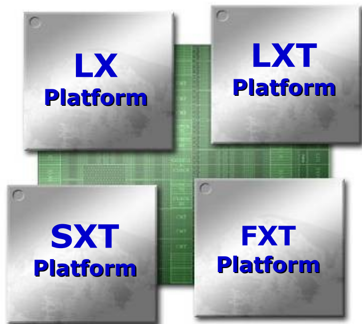


© 2009 Xilinx, Inc. All Rights Reserved

For Academic Use Only

Multiple Platforms

- Easy to create sub-families
  - LX : High-performance logic and parallel IO
  - LXT: High-performance logic with serial connectivity
  - SXT: Extensive signal processing with serial connectivity
  - FXT: Extensive processor oriented
    - Embedded-oriented with Highest Performance Serial Capabilities



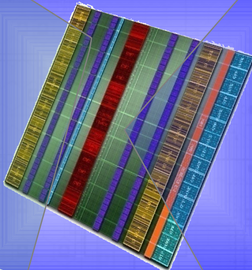
- Users can choose the best mix of resources to optimize cost and performance

© 2009 Xilinx, Inc. All Rights Reserved
For Academic Use Only
XILINX®

Virtex-5 Architecture

Enhanced

- 36Kbit Dual-Port Block RAM / FIFO with Integrated ECC
- 550 MHz Clock Management Tile with DCM and PLL
- SelectIO with ChipSync Technology and XCITE DCI
- Advanced Configuration Options
- 25x18 DSP Slice with Integrated ALU
- Tri-Mode 10/100/1000 Mbps Ethernet MACs



New

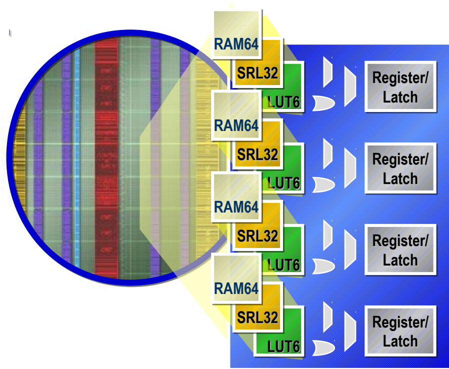
- Most Advanced High-Performance Real 6LUT Logic Fabric
- PCI Express® Endpoint Block
- System Monitor Function with Built-in ADC
- Next Generation PowerPC® Embedded Processor
- RocketIO™ Transceiver Options
  Low-Power GTP: Up to 3.75 Gbps
  High-Performance GTX: Up to 6.5 Gbps

© 2009 Xilinx, Inc. All Rights Reserved

XILINX®
For Academic Use Only

Advanced Logic Structure

- True 6-input LUTs
- Exclusive 64-bit distributed RAM option per LUT
- Exclusive 32-bit or 16-bit x 2 shift register



© 2009 Xilinx, Inc. All Rights Reserved

For Academic Use Only

XILINX®

DSP48E

- Virtex-5SX introduced a few new improvements in the DSP48E “enhanced” DSP block
- The adder block was modified to become a mulifunctional ALU. A pattern compare was added to support the detection of saturation, overflow and underflow conditions
- A 48-bit carry chain supports the propagation of partial sum and product carry’s so multiple DSP48E blocks can be chained to give higher bit precision
- ALU opcodes are dynamically controlled allowing functional changes on a clock cycle basis

© 2009 Xilinx, Inc. All Rights Reserved
For Academic Use Only

DSP48E Block

Includes a high performance ALU, pattern compare, and a multiplier

BCOUT ACOUT PCOUT

B 18 0 1 B REG CE D Q 2-Deep 48 A:B 18 48 ALUMode 4

A 25 0 1 A REG CE D Q 2-Deep 25 M REG CE D Q 72 36 0 X 36 0 1 Y P REG CE D Q 48 P

0 17-bit shift Z 17-bit shift PATTERN DETECT

C 48 C REG CE D Q 7 OpMode CarryIn C or MC 48 48

BCIN ACIN PCIN

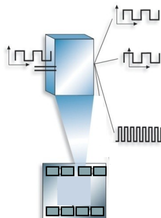
450 MHz operation in the slowest speed grade

XILINX
© 2009 Xilinx, Inc. All Rights Reserved
For Academic Use Only

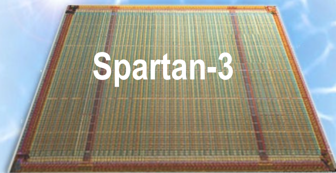
Spartan-3

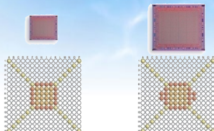
Designed for low-cost, high-volume applications

18x18 bit Embedded Pipelined Multipliers for efficient DSP

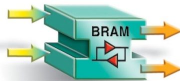


Up to eight on-chip Digital Clock Managers to support multiple system clocks





Guaranteed Density Migration
Numerous parts in the same package



4 I/O Banks, Support for all I/O Standards including PCI™, DDR333, RSDS, mini-LVDS

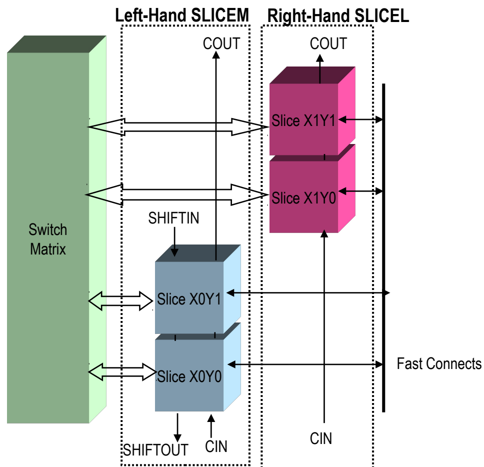
PCI, PCIe, PCI-X and PCI EXPRESS are registered trademarks and/or service marks of PCI-SIG.

XILINX®
© 2009 Xilinx, Inc. All Rights Reserved
For Academic Use Only

Modified Slices

SLICEM and SLICEL

- Each Spartan™-3 CLB contains four slices
  - Similar to Virtex™-II device
- Slices are grouped in pairs
  - Left-hand SLICEM (Memory)
    - LUTs can be configured as memory or SRL16
  - Right-hand SLICEL (Logic)
    - LUT can be used as logic only



© 2009 Xilinx, Inc. All Rights Reserved

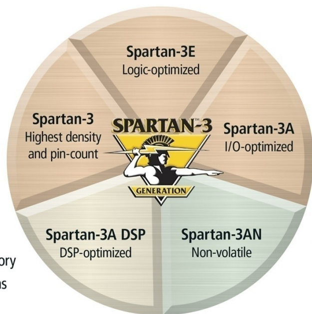
XILINX

For Academic Use Only

Multiple Domain-optimized Platforms

Mainstream

- Broad range of densities, general functionality and targeted specific application solutions
- Lower total system cost while increasing functionality



DSP

- Integrated DSP MACs and expanded memory
- Optimized for signal processing applications

Non-Volatile

- Combines leading-edge technology FPGAs & Flash technologies
- New evolution in security, protection and functionality

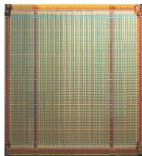
XILINX

© 2009 Xilinx, Inc. All Rights Reserved

For Academic Use Only

Spartan-3A

Spartan-3A DSP is a superset of Spartan-3A



True 3.3v
PPDS_25
PPDS_33
TMDS_33
DDR 2
Hot Swapping
Minimized Power Rails

- Power Management
  - Hibernate and Suspend modes
- Minimized power rails
- New I/O Standards
- BRAM with Byte write enable
- SPI/BPI Flash Interface
- Hot swapping

18K Block RAMs
Byte Write Enable
Improved Bus Access

Hibernate Mode
Suspend Mode

XILINX®
© 2009 Xilinx, Inc. All Rights Reserved
For Academic Use Only

Spartan-3A DSP

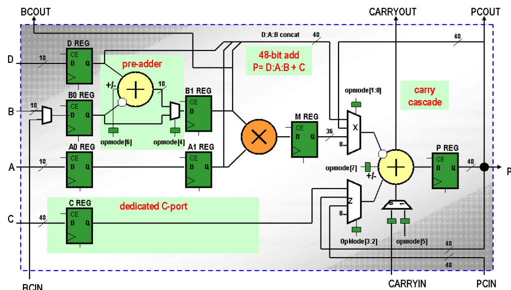
- Incorporates the primary features from earlier Virtex family DSP48 blocks
- The DSP48A block supports full MAC support with a pre-adder stage, multiplier, and add/accumulate state
- Dedicated DSP block offer the lowest cost/MAC in a FPGA

© 2009 Xilinx, Inc. All Rights Reserved
XILINX®
For Academic Use Only

DSP48A Block

Incorporates primary features from V4 DSP48 and includes a pre-adder stage

- Integrated XtremeDSP Slice
  - Application optimized capacity
    - 3400A – 126 DSP48As
    - 1800A – 84 DSP48As
  - Integrated pre-adder optimized for filters
  - 250 MHz operation, standard speed grade
  - Compatible with Virtex-DSP

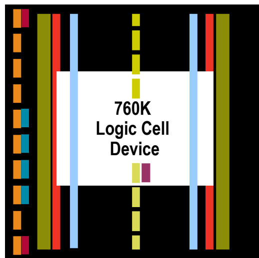


- Increased memory capacity and performance
  - Also important for embedded processing, complex IP, etc

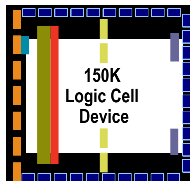
© 2009 Xilinx, Inc. All Rights Reserved
For Academic Use Only
XILINX

Architecture Alignment

Virtex-6 FPGAs



Spartan-6 FPGAs



Common Resources

- LUT-6 CLB
- BlockRAM
- DSP Slices
- High-performance Clocking
- Parallel I/O
- HSS Transceivers*
- PCIe® Interface

- FIFO Logic
- Tri-mode EMAC
- System Monitor

- Hardened Memory Controllers
- 3.3 Volt compatible I/O

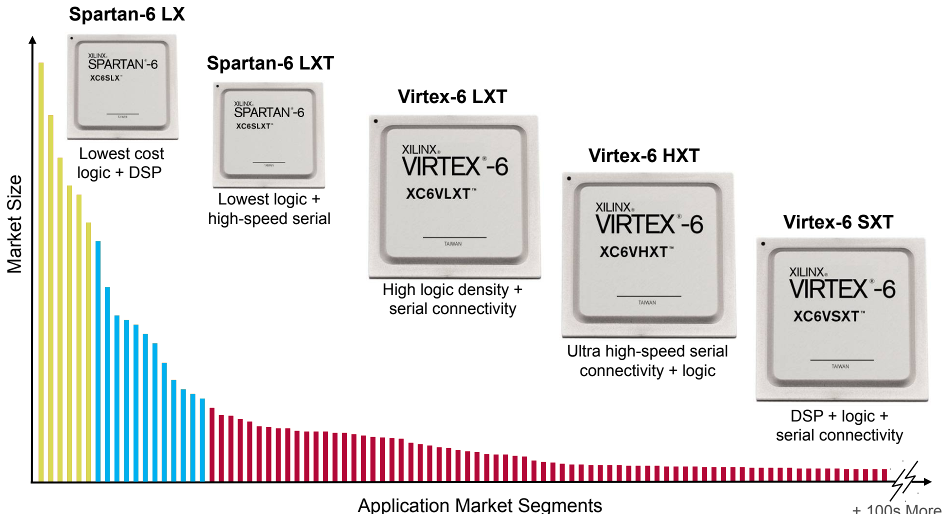
*Optimized for target application in each family

Enables IP Portability, Protects Design Investments

© 2009 Xilinx, Inc. All Rights Reserved

For Academic Use Only

Addressing the Broad Range of Technical Requirements



XILINX

© 2009 Xilinx, Inc. All Rights Reserved

For Academic Use Only

Higher DSP Performance

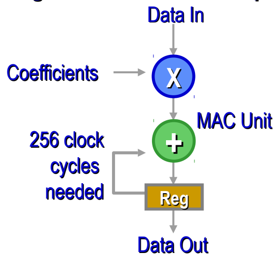
- Most advanced DSP architecture
  - New optional pre-adder for symmetric filters
  - 25x18 multiplier
    - High resolution filters
    - Efficient floating point support
  - ALU-like second stage enables mapping of advanced operations
    - Programmable op-code
    - SIMD support
    - Addition / Subtraction / Logic functions
  - Pattern detector
- Lowest power consumption
- Highest DSP slice capacity
  - Up to 2K DSP Slices

XILINX®
© 2009 Xilinx, Inc. All Rights Reserved
For Academic Use Only

Reason 1: FPGAs handle high computational workloads

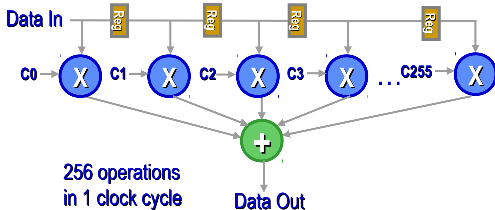
Speed up FIR Filters by implementing with parallel architecture

Programmable DSP - Sequential



$$\frac{1\ GHz}{256\ clock\ cycles} = 4\ MSPS$$

FPGA - Fully Parallel Implementation



$$\frac{500\ MHz}{1\ clock\ cycle} = 500\ MSPS$$

Example 256 TAP Filter Implementation

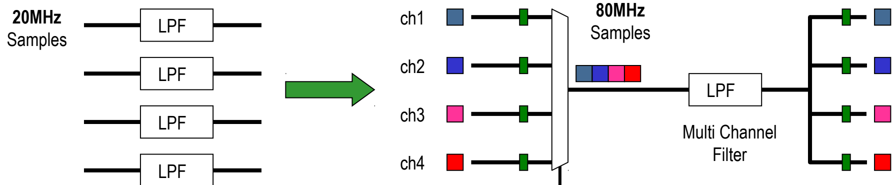
© 2009 Xilinx, Inc. All Rights Reserved

XILINX®

For Academic Use Only

Reason 2: FPGAs are ideal for multi-channel DSP Designs

Can implement multiple channels running in parallel or time multiplex channels into one filter

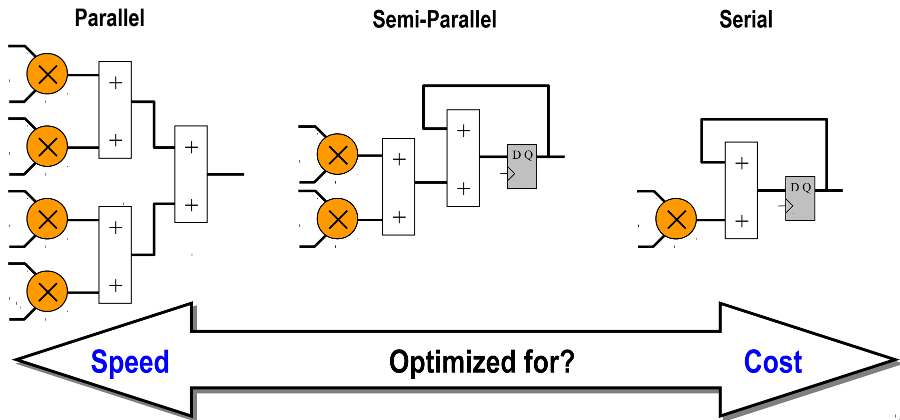


- Many low sample rate channels can be multiplexed (e.g. TDM) and processed in the FPGA, at a high rate
- Interpolation (using zeros) can also drive sample rates higher

© 2009 Xilinx, Inc. All Rights Reserved
XILINX®
For Academic Use Only

Reason 3: Customize Architectures to Suit your Goals

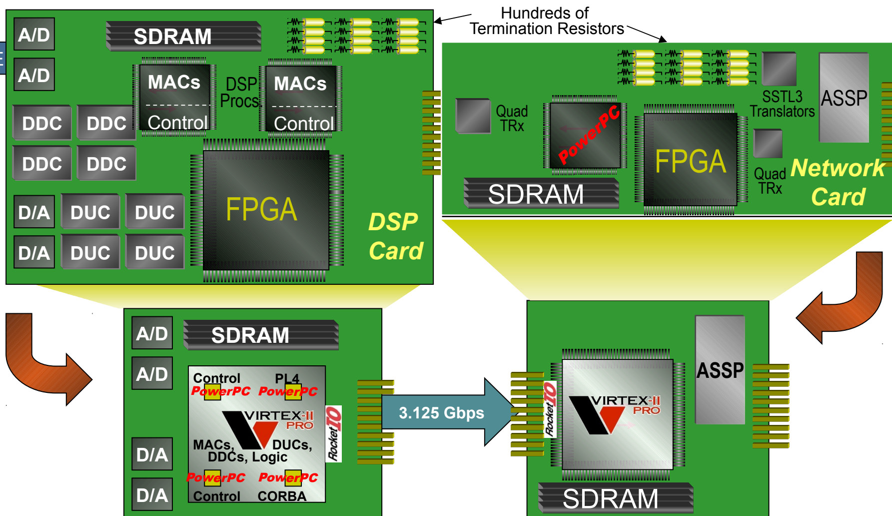
FPGAs allow Cost/Performance tradeoffs



© 2009 Xilinx, Inc. All Rights Reserved
For Academic Use Only
XILINX®

Reason 4: Lower System Cost through Integration

Implement Interface Logic within FPGA to connect DSP functions to I/O and Memory Devices



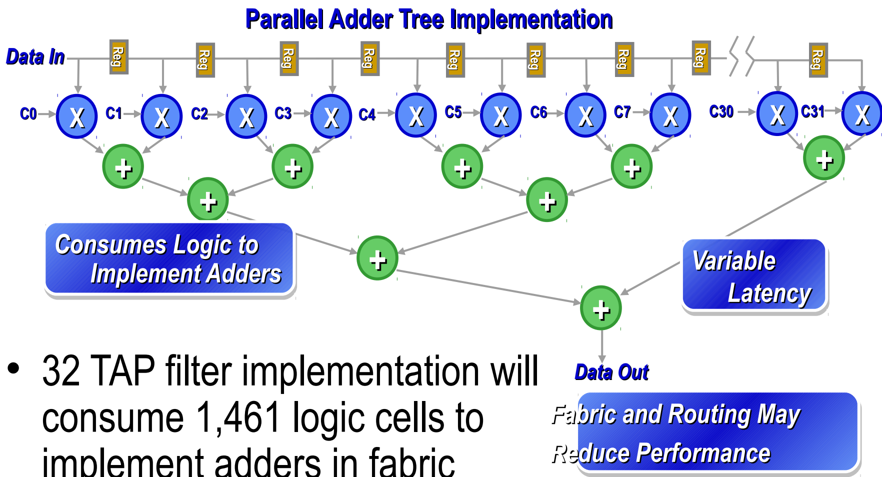
© 2009 Xilinx, Inc. All Rights Reserved

XILINX

For Academic Use Only

The XtremeDSP Slice Advantage

Without XtremeDSP Slice, Parallel Adder Tree Consumes Logic Resources



- 32 TAP filter implementation will consume 1,461 logic cells to implement adders in fabric

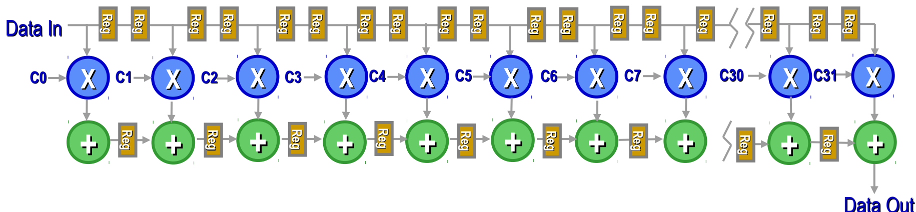
© 2009 Xilinx, Inc. All Rights Reserved

For Academic Use Only

XILINX

The XtremeDSP Slice Advantage

With XtremeDSP Slice, Parallel Adder Tree Consumes Zero Logic Resources



32 TAP filter implementation implemented entirely with XtremeDSP Slices

© 2009 Xilinx, Inc. All Rights Reserved
XILINX®
For Academic Use Only

IEEE 1149.1 JTAG Boundary Scan

- Motivations
- Testeur Bed-of-nails
- Vue matérielle du boundary scan
- Cellule scan de base
- Contrôleur Test Access Port (TAP)
- Instructions Boundary scan
- Conclusion

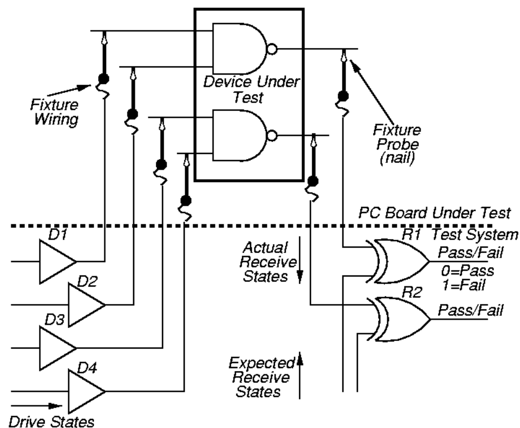
Bed of Nails

Bed of Nails

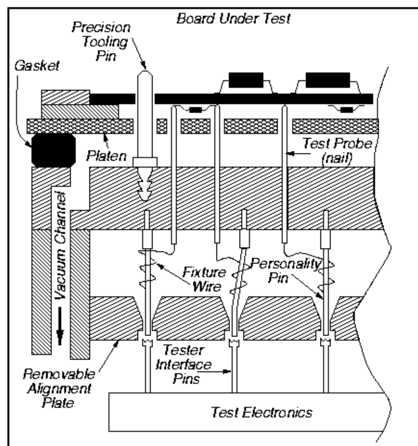
Motivations pour un Standard

- Bed-of-nails printed circuit board tester gone
- We put components on both sides of PCB an replaced DIPs with flat packs to reduce inductance
- Nails would hit components
- Reduced spacing between PCB wires
- Nails would short the wires
- PCB Tester must be replaced with built-in test delivery system – JTAG does that
- Need standard System Test Port and Bus
- Integrate components from different vendors
- Test bus identical for various components
- One chip has test hardware for other chips

Concept des testeurs Bed of Nails



Testeurs Bed of Nails



Optional / Required Instructions

Instruction	Status
BYPASS	Mandatory
CLAMP	Optional
EXTEST	Mandatory
HIGHZ	Optional
IDCODE	Optional
INTEST	Optional
RUNBIST	Optional
SAMPLE / PRELOAD	Mandatory
USERCODE	Optional

Plan

1. Introduction
2. Outils de conception : VHDL
3. Méthodes : Machines à états
4. Les mémoires
5. FPGA
6. Le Port Jtag
7. Conception d'un système programmable