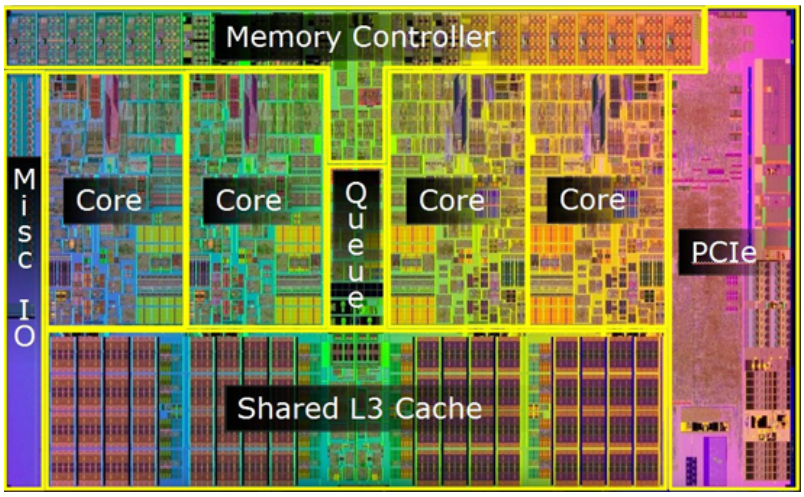
DGEI

Année 2014 – 2015

Architectures matérielles de systèmes informatiques



Intel I5

Prof. Daniela Dragomirescu

Architectures matérielles pour systèmes informatiques

Support de cours

Prof. Daniela Dragomirescu
INSA - DGEI

La participation active des étudiants va être une partie essentielle de ce cours

Bibliographie

- VHDL - Du langage à la modélisation - R.Airiau et al. - Presses Polytechniques et Universitaires Romandes
- Digital Design and Modeling with VHDL and Synthesis - K.C. Chang - IEEE Press
- Principles of CMOS VLSI Design: A system Perspective - N. Weste, K.Eshraghian - Addison Wesley
- The Zynq book – L. H. Crockett, R.A. Elliot, M.A. Enderwitz, R. W. Stewart

Evolution de systèmes informatiques

- Systèmes embarqués
- Réseaux mobiles
- Internet of Things
- Usine du futur
- Smart cities

Systèmes Hardware-software

Table de matières

Table de matières détaillé

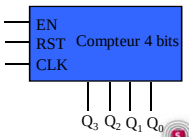
Table de matières

Table de matières

Introduction

- Rappels :
- Circuits logiques combinatoires
  - leur sortie est déterminé par la valeur courante de l'entrée
  - portes logiques, multiplexeurs, demultiplexeurs, décodeurs, circuits arithmétiques
- Circuits logiques séquentielles
  - Leur sortie est partiellement déterminé par l'évolution de l'entrée. La réponse du circuit dépend de l'histoire plus ou moins récente du fonctionnement du circuit. Fonction **MEMOIRE** interne.
  - Bascule D, registres, mémoires RAM, compteurs

- Exercice : réalisez un compteur 4bits en utilisant des bascules D en logique séquentielle synchrone



Introduction
Evolution de la méthodologie de conception

Intel Processeur	Date de production	Fréquence de fonctionnement	Nr. de transistors par puce
8086	**1978**	**8 MHz**	**29 K**
80286	1982	12.5 MHz	134 K
80386 DX	1985	20 MHz	275 K
80486 DX	**1989**	**25 MHz**	**1.2 M**
Pentium	1993	60 MHz	3.1 M
Pentium Pro	1995	200 MHz	5.5 M
Pentium II	1997	266 MHz	7 M
Pentium III	1999	500 MHz	8.2 M
Pentium III Xeon	1999	700 MHz	28 M
Pentium 4	**2001**	**1.7 GHz**	**42 M**

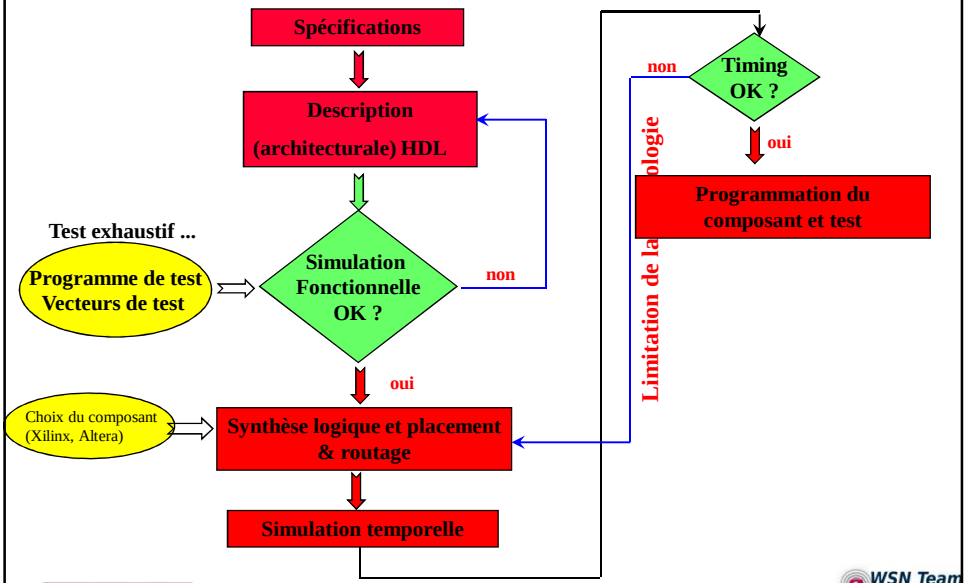
Introduction

- Evolution de la méthodologie de conception
- Apparition des langages HDL - Hardware Description Language
  - Verilog
  - VHDL

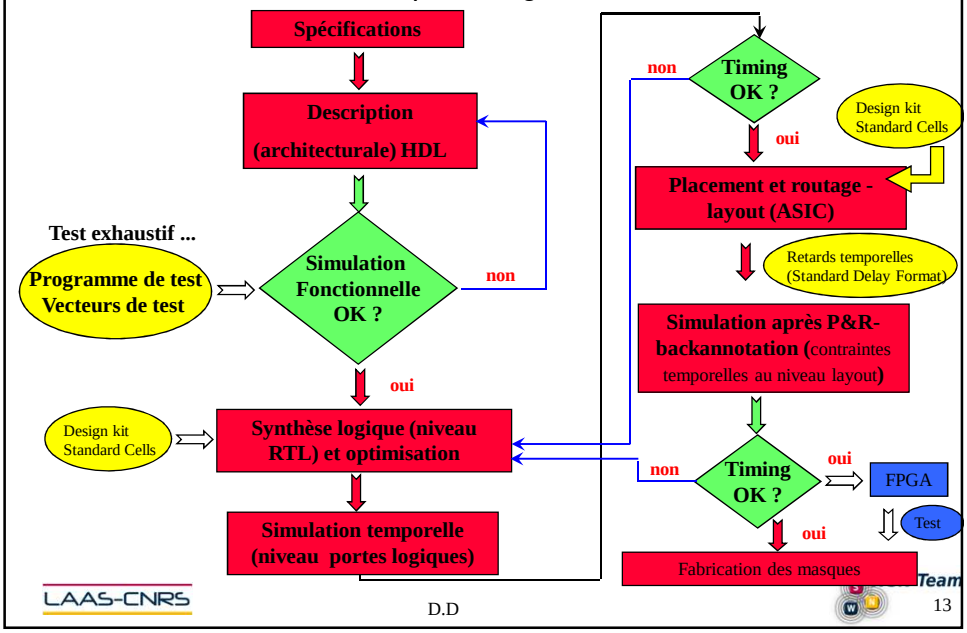
  } **Même principe, syntaxe différente**
- Travail avec des cellules standard - briques de base appartenant à une bibliothèque spécifique à chaque fondeur de circuits intégrés

Standard cells - design kit

Cycle de conception des systèmes numériques intégrés - FPGA



Cycle de conception des systèmes numériques intégrés - ASIC



2. Conception des circuits numériques
Le langage VHDL

2.1.1 Historique

- VHDL - VHSIC Hardware Description Language
  VHSIC - Very High Speed Integrated Circuits
- 1980 - demande du département de la défense des Etats-Unis
- ADA pour le logiciel et VHDL pour le matériel
- VHDL - dédié seulement aux architectures matérielles? ⇐
  NON !
- VHDL sert à décrire des systèmes matériels à un haut niveau d'abstraction
  - circuits intégrés
  - cartes de composants
  - systèmes entiers (logiciel + matériel) - réseaux d 'ordinateurs

2.1.2 Qu'est-ce qu'un langage de description de matériel ?

- Un langage de description matériel ne vise pas une « exécution »
- Un langage de description matériel tel VHDL peut être utilisé pour différents buts :
  - Spécification
  - Simulation
  - Synthèse
  - Preuve formelle

2.1.3 Avantages et inconvénients de VHDL

- Avantages :
  - VHDL est standardisé - standard IEEE
  - pérennité assurée par la norme
  - **conception modulaire et hiérarchique**
  - VHDL est un langage moderne, puissant et général
    - **haute modularité**
    - sécurité d'emploi
    - fiabilité
    - **notion de temps bien définie**
    - unités de compilation séparées
    - typage fort
    - **généricité**
  - en utilisant VHDL on minimise le risque d'erreur sur le circuit intégré en silicium - on diminue le coût de production

- Inconvénients:
  - langage de simulation → tout n'est pas synthétisable !

2.1.4 Standardisation

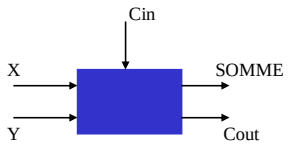
- IEEE Standard 1076 - 1987
- IEEE Standard 1076 - 1993
- IEEE Standard 1076 - 2003

2.2 Bibliothèques

- Langage modulaire - unités petites et hiérarchisées
- Unités de conception - peuvent être compilées séparément
- Description VHDL correcte - bibliothèque de travail - WORK
  - portabilité
- Utilitaires ou modèles généraux - bibliothèques de ressources
  - facilite le travail en équipe
- Bibliothèques : IEEE , STD
  - **package** IEEE.std_logic_1164 et **package** IEEE.std_logic_arith
  - **package** STD.textio et **package** STD.STANDARD

2.3 Unités de conception
2.3.1 Entité

- Entité - model VHDL
  - Vue externe



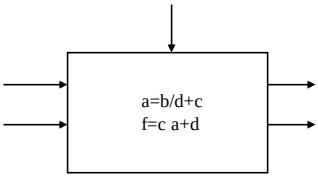

```
entity adder is
        port (
                X, Y, Cin: in bit;
                Somme, Cout:    out bit);
end adder;
```

Les ports

PORT FORMEL / OBJET : CONNECTE	MODE **IN**	MODE **OUT**	MODE **INOUT**	MODE **BUFFER**	MODE **LINKAGE**
Port de mode **in**	oui	non	non	non	oui
Port de mode **out**	non	oui	non	non	oui
Port de mode **inout**	oui	oui	oui	non	oui
Port de mode **buffer**	oui	non	non	oui	oui
Port de mode **linkage**	non	non	non	non	oui
Signal local	oui	oui	oui	oui	oui
Mot clé **open** (non-connecté)	non	oui	oui	oui	oui

2.3 Unités de conception
2.3.2 Architecture

- Vue interne d 'une entité - ARCHITECTURE



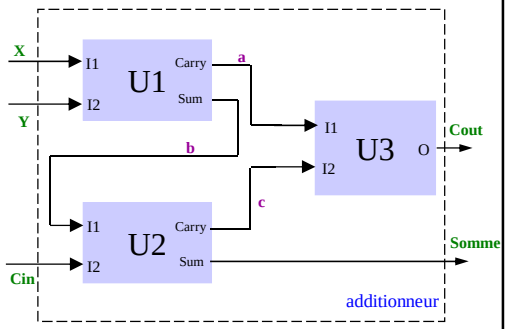
- Style de description de l 'architecture :
  - Description structurelle : interconnexion de composants
  - Description flot de données: comportement sous forme d'équations
  - Description comportementale: comportement sous forme algorithmique
- On peut combiner les différents styles des descriptions dans une même architecture

2.3.2 Architecture

- Architecture structurelle de l 'additionneur

```
architecture  vue_structurelle of adder is
    component demi_additionneur
            port( I1,I2:        in bit;
                  Carry:        out bit;
                  Sum:          out bit);
    end component;
    component porte _OU
            port( I1, I2:       in bit;
                  O:            out bit);
    end component;
    signal a,b,c : bit;

    begin
      U1:   demi_additionneur    port map(X,Y,a,b);
      U3:   porte_OU             port map(a,c,Cout);
      U2:   demi_additionneur    port map(b,Cin,c,Somme);
                end vue_structurelle;
```



2.3.2 Architecture

- Architecture flot de données de l 'additionneur

S = X⊕Y
Somme = S ⊕ Cin
Cout = XY+ SCin

Equations booléennes de l 'additionneur

```
architecture data_flow of adder is
    signal S: bit;
 begin
    Somme <= S xor Cin    ;          --after 10 ns;
    S <= X xor Y          ;          --after 10 ns;
    Cout <= (X and Y) or (S and Cin)  ;   --after 20 ns;
end  data_flow;
```

2.3.2 Architecture

- Architecture comportementale de l 'additionneur
  - représentée par une table de vérité

```
architecture vue_comportementale of adder is
begin
    process    -- instruction concurrente
            variable N:          integer;
            constant sum_vector:      bit_vector(0 to 3):= "0101";
            constant carry_vector:    bit_vector(0 to 3):= "0011";
    begin
            N:=0;
            if X= '1'  then N:=N+1; end if;
            if Y= '1'  then N:=N+1; end if;
            if Cin='1' then N:=N+1; end if;
            Somme <= sum_vector(N);
            Cout <= carry_vector(N);
            wait on X, Y, Cin;
    end process;
end vue_comportementale;
```

X	Y	C_{in}	Sum	C_{out}
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

Architecture comportementale de l 'additionneur représentée par une table de vérité

```
architecture vue_comportementale of adder is

Signal N: integer  range 0 to 3 := 0;
        begin
                process    -- instruction concurrente
                        -- variable N:          integer;
        constant sum_vector:bit_vector(0 to 3):= "0101";
        constant carry_vector:          bit_vector(0 to 3):= "0011";
                begin
                         --N<=0;
                        if X= '1'  then N<=N+1; end if;
                        if Y= '1'  then N<=N+1; end if;
                        if Cin=1 then N<=N+1; end if;
                        Somme <= sum_vector(N);
                        Cout <= carry_vector(N);
                        wait on X, Y, Cin;
                end process;
        end vue_comportementale;
```

2.3.3 Configuration

- Donne la correspondance entre le composant et le modèle dont il est instancié
- Permet faire la liaison entre des composants utilisés dans une architecture et leur réalisation effective

Ex: Soit une entité « trois_registres » avec une architecture « beh » ayant 3 registre R1, R2 et R3

```
1. use work.trois_registres;
   configuration alpha of trois_registres is
    for beh
        for R1,R2 : registre_8bit use entity work.registre_8bit(arch);
        for R3  : registre_8bit use entity work.registre_8bit(decalage);
    end for;
   end alpha;
```

2. **FOR ALL** : registre_8bit **use entity** work.registre_8bit(arch);

Exemple : l'additionneur

```
entity adder is
    port (X, Y, Cin : in bit;
          Sum, Cout : out bit);
end adder;
architecture  vue_structurelle of adder is
        component demi_additionneur
                port( I1,I2:        in bit;
                      Carry:        out bit;
                      Sum:          out bit);
        end component;
        component porte _OU
                port( I1, I2:       in bit;
                      O:            out bit);
        end component;
    for  all : demi_additionneur use entity work.half_adder(beh);
    for all : porte_ou use entity work.porte_ou(struct);

        signal a,b,c : bit;
        begin
           U1:   demi_additionneur   port map(X,Y,a,b);
           U2:   demi_additionneur   port map(b,Cin,c,Sum);
           U3:   porte_OU            port map(a,c,Cout);
        end vue_structurelle;
```

```
library  4info;

for  all : demi_additionneur use entity
        4info.half_adder(beh);
```

2.3.4 Le paquetage

- Ensemble des algorithmes, sous-programmes, nouveau types et sous-types, des objets: signaux et constantes (pas de variables)
- Un ou plusieurs paquetage dans la même bibliothèque
- 2 unités de conception :
  - Spécification d'un paquetage - vue externe
    - présente tous ce qu'exporte le paquetage (algorithmes, objets, types)
  - Corps du paquetage - vue interne - *optionnel*
    - contient la description des algorithmes, déclarations locales des types, objets
- Pour avoir accès à un paquetage il faut le référencé par une clause **use**
  - library IEEE;
    use IEEE.std_logic_1164.all ;
  - library ma_lib;
    use ma_lib.mon_paquetage.all;

2.3.4 Le paquetage

- Spécification du paquetage :

```
package SIMPLE is

    constant TAILLE_MAX: integer :=1024;

    subtype  mon_integer is INTEGER range 0 to TAILLE_MAX;

    signal addition_10bits : mon_integer ;

    function MIN(A,B:INTEGER) return INTEGER;
    function MAX(A,B:INTEGER) return INTEGER;

end SIMPLE;
```

  - Les déclarations faite dans la spécification du paquetage sont connus dans le corps du paquetage

2.3.4 Le paquetage

- Corps du paquetage

```
package body SIMPLE is
    function MIN(A,B:INTEGER) return INTEGER is
    begin
            if A<B then return A;
            else return B;
            end if;
    end MIN;
    function MAX(A,B:INTEGER) return INTEGER is
    begin
            if B<A then return A;
            else return B;
            end if;
    end MAX;
end SIMPLE;
```


2.4 Domaine concurrent et domaine séquentiel

- La description d'un système materiel est naturellement concurrente
- Le fond d 'une description VHDL est concurrent
- Les 2 domaines cohabitent en VHDL
- Domaine concurrent
  - La zone de déclarations des entités
  - La zone de déclarations des architectures
- Domaine séquentiel
  - La zone de déclarations de processus
  - Le corps du paquetage

2.5 TEST

- Pour vérifier (simuler) le comportement du composant décrit en VHDL il faut le tester
- Le programme de test - un programme VHDL, avec la même structure (entity, architecture)
  - Applique les stimuli en entrée et regarde les sorties du composant sous test
- Si possible - test exhaustif
  - Difficile à mettre en œuvre pour des systèmes complexes
  - Preuve formelle

2.5 TEST

- Exemple: test de l'additionneur

```
entity test_adder is
end test_adder;
architecture bench of test_adder is
    COMPONENT adder is
            port (X, Y, Cin : in std_logic;
                    Somme, Cout : out std_logic);
    END COMPONENT;

    For all : adder use entity work.adder(data_flow);

    SIGNAL data1, data2, data3 :std_logic;
    SIGNAL dataout, carry_out : std_logic;

    BEGIN
            additioneur: adder PORT MAP (data1,data2, data3,dataout, carry_out);
            data1 <= '0', '1' after 30 ns;
            data2 <= '1', '0' after 50 ns, '1' after 60 ns;
            data3 <= '0', '1' after 12 ns;
    END bench;
```

Test non-exhaustif

2.6 Opérateurs et littéraux

- Classes d'opérateurs par ordre de priorité croissante
  1. Logiques : and or nand nor xor;
     - défini sur les types booléen et BIT
  2. Relationnels: **=** **/=** **<** **<=** **>** **>=**
     - défini sur tous les types sauf le type file
  3. Arithmétiques et concaténation: **+** - &
  4. Signe: **+** -
     - les opérateurs de signe sont moins prioritaires que les opérateurs de multiplication !!!!!
  5. Multiplication: * / mod rem
  6. Exposant, valeur absolue, complément: ** abs not
     - ** élévation à la puissance : opérande de droite (la puissance) doit être entière

- Il est possible de surcharger les opérateurs ; ceci ne change pas leur priorité

2.6 Opérateurs et littéraux

- Classifications:
  - numérique - notation décimale - entier (1345 ou 1_345) ou réel (13.0)
    - notation basée – base 16 X"A2" ou base 2 "11111011"
  - caractères ou chaîne de caractères ('a' , "Bonjour", "1234")
  - énumérés
  - chaînes de bits ("1010")
  - null

- OBS: 1 - nombre entier '1' - bit

- Expressions - portent un type, et au moment de l 'exécution une valeur

- Identificateurs - commencent avec une lettre; pas de différence entre majuscules et minuscules

2.7 Les objets

- Les objets contiennent des valeurs. Il y a 4 clases d 'objets en VHDL : constantes, variables, fichiers et signaux.
- Les constantes ont une valeur unique fixe
- Les variables ont une valeur unique modifiable
- Les fichiers contiennent des séquences de valeurs qui peuvent être lues ou écrites
- Les signaux conservent l'histoire des valeurs passées, de la valeur présente et des valeurs prévues dans le futur : seules les valeur futures peuvent être modifiées par affectation de signal
- Tous les objets ont un type

2.8 Types

- VHDL est un langage typé
- Un type est un ensemble de valeurs ordonnées
- Le type est statique : il ne peut pas être modifié
- Il est possible de définir de nouveaux types en utilisant les types prédéfinis et des constructeurs de types
- Classification :
  - Types scalaires
  - Types composites
  - Types access (pointeur) - seules les variables peuvent être de type acces
  - Types fichiers - mot clé file

2.8.1 Types scalaires

- Les entiers
  - subtype mon_integer is INTEGER range -65_536 to 65_535 ;
- Les flottants
  - subtype mon_flottant is REAL range 5.36 downto 2.15;
- Les types énumérés
  - type COULEUR is (ORANGE, VERT, ROUGE);
  - type BOOLEAN is (FALSE, TRUE);
  - type LOGIC4 is ('X', '0', '1', 'Z');
- Les types physiques
  - TIME - définit dans le paquetage STANDARD
  - TIME -la notion de temps que connaît le simulateur
  - Sont caractérisés par l 'unité de base, l 'intervalle de ses valeurs autorisées, collection des sous-unités et leur correspondance.

2.8.1 Type physiques

- **TIME**

```
type TIME is range -9_223_372_036_854_775_808 to
                                    9_223_372_036_854_775_807
   -- codage sur 64 bits ;
            units fs ;
                  ps =1000 fs;        ms = 1000 us;
                  ns = 1000 ps;       sec = 1000 ms;
                  us = 1000 ns;       min = 60 sec ;
                                      hr = 60 min;
            end units;
```

- **DISTANCE**

```
      type DISTANCE is range 0 to 1E16
            units A ;
                  nm =10 A;           um = 1000 nm;
                  mm = 1000 um;       cm = 10 mm;
                  m = 1000 mm;        km = 1000 m;
            end units;
```

2.8.1 Type STD_LOGIC

- Se trouve dans le paquetage IEEE.std_logic_1164
- Paquetage IEEE.std_logic_unsigned
- Paquetage IEEE.std_logic_arith

- Remplace le type bit ; c'est le type bit résolu

- Il a 5 valeurs:
  - '1' - 1 logique
  - '0' - 0 logique
  - 'Z' - haute impedance
  - 'U' - non-initialisé
  - 'X' - indéterminé

- STD_LOGIC_VECTOR
  - signal A : std_logic_vector(0 to 7);

2.8.2 Types composites

- Classification :
  - Tableaux : collection d 'objets de même type - **array**
  - Articles : collection d 'objets de types différents - **record**

- Tableaux
  - Ses éléments sont désignés par un indice
  - Contraints
    - type MOT is array (0 to 31) of STD_LOGIC;
  - Non-contraints
    - type STD_LOGIC_VECTOR is array (NATURAL range <>) of STD_LOGIC;
    - Définition d'un intervalle d'indiçage lors de l'exécution
      - signal bus : STD_LOGIC_VECTOR (63 downto 0);
  - Attributs de tableaux :
    LEFT, RIGHT, HIGH, LOW, RANGE, REVERSE_RANGE, LENGTH

2.8.2. Tableaux - Attributs

- Exemple :

```
type INDEX 1 is INTEGER range 1 to 20;
type INDEX2 is INTEGER range 19 downto 2;
type VECTEUR1 is array (INDEX1) of STD_LOGIC;
type VECTEUR2 is array (INDEX2) of STD_LOGIC;
```

  - VECTEUR1'LEFT rendra 1 et VECTEUR2'RIGHT rendra 2
  - VECTEUR1'HIGH rendra 20 et VECTEUR2'HIGH rendra 19
  - VECTEUR1'LOW rendra 1 et VECTEUR2'LOW rendra 2
  - VECTEUR1'RANGE rendra 1 to 20 - utilisation pour des boucles
  - VECTEUR1'REVERSE_RANGE rendra 20 downto 1
  - VECTEUR1'LENGTH rend le nombre d'éléments du tableau - donc 20

2.8.2 Types composites - ARTICLES

- Articles - mot clé record - end record
  - Contient des éléments de types différentes
  - Ses éléments sont désignés par un nom
  - Exemple :

```
type BIT_COMPLET is
        record
                valeur : std_logic;
                sortance_min, sortance_typ, sortance _max: integer;
        end record;
 signal A : BIT_COMPLET;
 variable B : BIT_COMPLET;
```

alors A.valeur est de type std_logic, A.sortance_typ est de type entier

- **Affectation**

```
A <= B ;
ou
A.valeur <= B.valeur after 50 ns;
```

2.8.2 Agrégats

- Un agrégat est une façon d'indiquer la valeur d'un type composite
- Un agrégat se note entre parenthèse, les éléments étant séparés par des virgules
- Vérification de type faite par le compilateur
- Exemple : soit
  - type TAB is array (1 to 3) of INTEGER ;
  - type ART is record
    Champ1 : NATURAL;
    Champ2 :STD_LOGIC;
    Champ3 : NATURAL;
    end record;
  - signal A : TAB; signal B : ART;
- 3 notations distinctes de l'agrégat :
  - notation positionnelle
    A <= (12, 13, 14); B <= (5, '0', 15);
  - notation par denomination
    A <= (1=>12, 2=>13, 3=>14); B <=(Champ3 =>15,Champ1 => 5, Champ2 =>'0');
  - notation mixte (positionnelle -denomination)
    A <=(5, others =>0); B <=(Champ2 =>'0', others =>0);

    A <=(others =>0);

2.8.3 Sous-types

- Déclaration de sous-type quand on souhaite restreindre les valeurs d'un certain type
- Un sous-type est toujours compatible avec son type de base
- Exemples:
  - subtype integer_8bits is INTEGER range 0 to 255 ;
  - subtype NATURAL is INTEGER range 0 to INTEGER'HIGH;

- Sous-types dynamiques:
  - Font intervenir des expressions calculables à la simulations

    Subtype MOT is STD_LOGIC_VECTOR (1 to MAX);
    où MAX est un paramètre d'un sous-programme ou paramètre générique d'une entité

2.9 Notions de signal et affectation du signal

- Un signal correspond à la représentation matérielle du support de l 'information en VHDL.
- Un signal a une évolution en temps (une variable - non)
- Toute déclaration de signal se fait dans le domaine concurrent - la zone de déclaration des architectures,des entités et la spécification du paquetage
- Tout signal peut conserver l'historique de ses valeurs passées si celles-ci sont utilisées par ailleurs, la valeur présente et les valeurs prévues dans le futur - le «pilote » du signal
- Affectation du signal :
  - connexion à un port de sortie d 'un composant
  - affectation du signal dans le domaine concurrent
  - affectation du signal dans le domaine séquentiel

2.9.1 Affectation inconditionnelle de signal

S <='0', '1' after 20 ns;

- La valeur doit avoir un type compatible avec celui du signal affecté
- Chaque élément de l'expression a une partie valeur et une partie délai de type physique TIME (délai nul par défaut sans la clause after)

S <= A after 10 ns ;

- Tous les signaux utilisés dans l 'expression sont soit locaux, soit des ports définis en entrée

2.9.2 Exécution d'une affectation de signal

- L'affectation du signal ne change pas la valeur présente du signal, mais modifie les valeurs futures que ce signal sera susceptible de prendre

A<= B ; ou A<=B after 0 ns;

- Le signal A prend la valeur présente du signal B après un delta délai
- Delta délai - est un délai qui est nul pour la simulation et ne représente que la causalité

2.9.2 Exécution d'une affectation de signal

S <= A+B after 20 ns;

- L'expression affectée à un signal est évaluée à chaque changement de valeur sur l'un de ses signaux d'entrée (réponse évènementielle)
- Chaque valeur calculée par l'expression est mémorisée dans le signal destination avec son délai d'apparition
- Si l'affectation est de nouveau évaluée, les anciennes valeurs prévues pour le futur peuvent être remplacées par les nouvelles valeurs - voir exemple affectation conditionnelle

2.9.3 Affectation conditionnelle de signal

- Condition simple:
  - S <= A when condition else
    B;
  - S <= A after 20 ns when condition else
    B after 10 ns;

Instruction conditionnelle concurrente

- Condition multiple avec ordre de précédence
  - S <= 5 when A ='1' and B ='1' else
    4 when A ='1' else
    0;

Instruction conditionnelle concurrente

- Quand n'importe quel signal d'entrée change de valeur, l'ensemble des conditions est évalué dans l 'ordre (de gauche à droite)
- La première valeur produite par la première expression dont la condition est vraie est affectée au signal

2.9.4 Affectation sélective de signal

- Une condition unique détermine quelle expression sera affectée au signal

```
type  op_bit is (et, ou, non) ;
signal opcode: op_bit;
signal result, A, B, : std_logic;
 ……
with opcod select          -- instruction  de choix concurrente
    result <= A and B when et,
              A or B when ou,
              not A when non,
              '0' when others;
```

- L 'affectation sélective modélise les composants de type multiplexeurs et décodeurs

2.9.5 Attributs

- L'attribut est une caractéristique associe à un type ou à un objet
- L'utilisateur peut définir des nouveaux attributs

- Prédéfini :
- S'quiet(T) - TRUE si le signal S est "tranquile" pendant au moins T
- S'stable(T) -TRUE si aucun événement sur le signal S depuis T
- S'delayed(T) -signal S différé de T après NOW

- S'event - TRUE si un événement vient d'arriver sur S

- S'last_value - dernière valeur de S avant le dernier événement
- S'last_event - valeur du temps écoulé depuis le dernier événement de S

2.9.6 Règles de distinction entre variable et signal

- Les variables se déclarent dans le domaine séquentiel des processus ou des sous-programmes
- Les variables s'utilisent et s'affectent dans le domaine séquentiel uniquement a:='1'

- Les signaux se déclarent dans le domaine concurrent des entités, architectures, spécification du paquetage, des blocs
- Les signaux s'utilisent dans les 2 domaines séquentiel et concurrent a <='1'

- Une affectation de variable est immédiate, alors qu 'une affectation de signal est différée jusqu'à la fin de l 'évaluation de toutes les affectations

2.10 Instructions concurrentes

- On ne simule pas en temps « réel » - on simule en temps virtuel (temps de simulation)
- Ces fonctionnements seront décrits par des instructions concurrentes s 'exécutant de manière asynchrone
  - Assignations de signaux (conditionnelle ou sélective ou inconditionnelle)
  - Instanciations de composants
  - Instruction processus
  - Appels concurrents de procédures
  - Instructions d 'assertion
  - Instructions de bloc
  - Instruction generate

2.10 Instructions concurrentes

- Processus
  - est l'objet fondamental manipulé par le simulateur
  - toute instruction concurrente peut toujours être traduite par un processus
  - un processus est sensible aux signaux
  - il contient que des instructions séquentielles
  - la durée de vie d 'un processus est celle de la simulation; un processus est cyclique

2.10 Instructions concurrentes - Processus

{label: } **process** { liste_des_signaux_surveillés}
Déclarations
begin
Instructions séquentielles
end process {label};

liste de sensibilité

- Les déclarations sont élaborés une seule fois, à l'initialisation
- un processus s 'exécute au moins une fois à l 'initialisation jusqu 'à l 'instruction WAIT (si elle existe)

- à chaque eveniment intervenant sur un des signaux surveillés , le processus va s'exécuter
- Restrictions:
  - l'instruction wait bloque le processus - voir détail sur l'instruction wait au chapitre 2.10 Instructions séquentielles
  - l'instruction wait ne peut s'utiliser à l'intérieur d'un processus que si celui-ci ne possède pas de liste de signaux surveillés

2.10 Instructions concurrentes - Processus

Processus avec liste de sensibilité :

{label: } **process**
{liste_des_signaux_surveillés}
Déclarations
begin
Instructions séquentielles
end process {label};

- S'exécute à l'initialisation une fois

Processus sans liste de sensibilité:

{label: } **process**
Déclarations
begin
wait on {signaux_surveillés }
Instructions séquentielles
end process {label};

- Ne s'exécute pas lors de l'initialisation

2.10 Instructions concurrentes

- Block
  - réuni des instructions concurrentes
  - est la base de hiérarchie en VHDL
  - toute hiérarchie VHDL se ramène à un ou plusieurs blocs imbriqués
  - permet de garder des affectations de signaux - circuits synchrones

```
label : block
        { généricité_et_port}
        … Déclarations …
        begin
        ….Instruction concurrentes
        end block;
```

- Appel concurrent de procédure
  - a la même syntaxe que l'appel séquentiel de procédure
  - **domaine concurrent - paramètres de la procédure ne peuvent être que de signaux ou de constantes**
- Instruction concurrente d'assertion

2.11 Instructions séquentielles

- Domaine d 'utilisation des instructions séquentielles : dans le corps d'un processus ou sous-programme
  - Instruction WAIT
  - Instruction ASSERT
  - Instructions d 'affectation de variables et de signaux
  - Appel de procédures
  - Instructions conditionnelles
  - Instructions de contrôle
  - Instruction nulle - null

2.11 Instructions séquentielles

- Instruction WAIT
  - L'exécution de processus ou de programme peut être suspendue, en fonction d 'une condition donnée et pour un temps indiqué par l 'instruction wait
  - wait { on liste _signaux} {until condition_booléenne} { for temps};
  - Tout événement arrivant sur un signal de la liste donnée provoque l 'évaluation de la condition booléenne.
- Exemple:

```
process
  begin
        wait on CLK;
        if clk'event and CLK='1' then
              q <=d ;
        end if;
  end process;
```

```
process
  begin
    wait until CLK'event and CLK ='1';
    q <=d ;

  end process;
```


2.11 Instructions séquentielles

- Instruction ASSERT
  - Surveille une condition et émet un message dans le cas où celle-ci est fausse.
  - L'exécution du programme reprend alors immédiatement derrière l 'instruction d 'assertion.
  - assert condition {report mesg} {severity niveau}
  - assert NOW<1 min report "Fin simu" severity ERROR;
    - Type Severity_Level is (note, warning, error, faillure);

2.11 Instructions séquentielles

- Instructions d'affectation de variables et de signaux
  - Variable1 **:=** 2;
  - Signal1 **<=** 2 after 20 ns;
- Appel de procedure
  - procedure alfa (nr:integer, msg:string) --définition de la procédure
  - Alfa(4, "GO"); --appel positionnel
  - Alfa(nr =>4, msg =>"GO");
- Instruction return
  - Réservée aux sous-programmes
- Instruction nulle
  - null
  - l'exécution passe à la ligne suivante
  - L'instruction nulle n'est pas nécessaire à la compilation de processus ou de corps de procédures vides

2.11 Instructions séquentielles

- Instructions conditionnelles

```
if condition1 then traitement1
elsif condition2 then traitement2
else  traitement3
end if;
```

- Instructions de choix:

```
case expression is
when val1 => instructions1
when val2 => instructions2
when others => instr3
end case;
```

2.11 Instructions séquentielles

- Instructions de boucle
  - loop
    - Instructions séquentielles

    end loop;
  - for indice in intervalle loop
    - Instructions séquentielles

    end loop;
  - while condition loop
    - Instructions séquentielles

    end loop;
  - next label_de_boucle when condition --arête l'itération en cours
  - exit label_de_boucle when condition

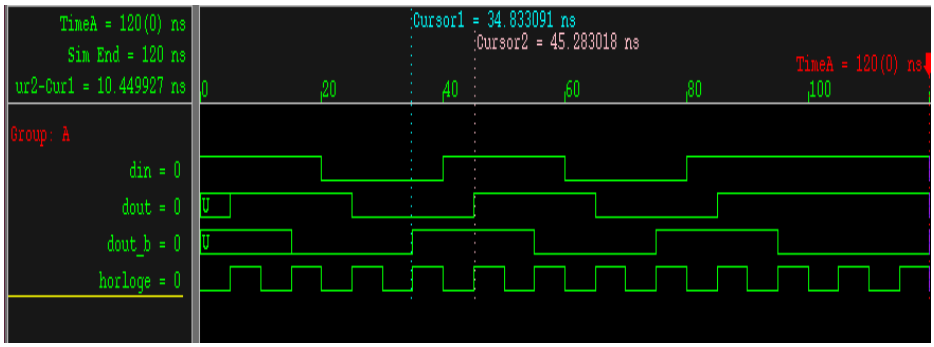
Bascule D

```
entity basculeD is
port(
    D, CLK: in std_logic;
    Q: inout std_logic;
    Qb: out std_logic);
end basculeD;
architecture beh1 of basculeD is
begin
  process
    begin
    wait until clk'event and clk='1';
    Q <= d;
    Qb <= not Q;
  end process;
end beh1 ;
```

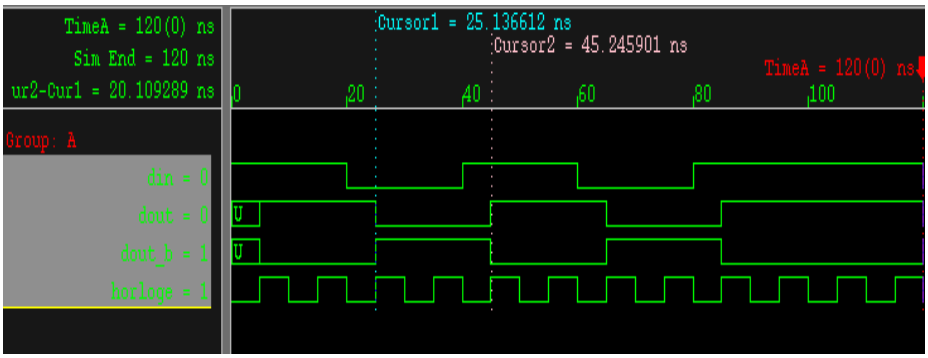
Architecture incorrecte

```
entity basculeD is
port(
    D, CLK: in std_logic;
    Q: inout std_logic;
    Qb: out std_logic);
end basculeD;
architecture beh2 of basculeD is
begin
  process
    begin
    wait until clk'event and clk='1';
    Q <= d;
  end process;
    Qb <= not Q;
end beh2 ;
```

Bascule D - architecture beh1



Bascule D - architecture beh2

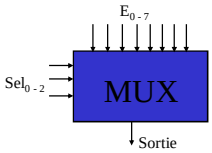


Bascule D

```
entity basculeD is
    port(
        D, CLK: in std_logic;
        Q: out std_logic;
        Qb: out std_logic);
end basculeD;

architecture beh3 of basculeD is
begin
    process
        begin
            wait until clk'event and clk='1';
            Q <= d ;
            Qb <= not D;
        end process;
end beh3 ;
```


Multiplexeur 8 bits (3 entrées de sélection)




```
entity mux is
        port (
                sel : in integer range 0 to 7 ;
                entree : in std_logic_vector(0 to 7);
                Sortie : out std_logic);
        end mux;
Architecture beh of mux is
begin
    with sel select                    -- instruction  de choix concurrente
        sortie <=  entree(0) when 0,
                   entree(1) when 1,
                   entree(2) when 2,
                   entree(3) when 3,
                   entree(4) when 4,
                   entree(5) when 5,
                   entree(6) when 6,
                   entree(7) when 7;
end beh;
```

2.12 Généricité

- une entité est générique, jamais une architecture
- permet de définir une famille de composants par rapport à une caractéristique donnée
- valeur fixée lors de l 'instance de l'entité
  - entity Porte_et is
    generic (nombre_entrees: Natural:=2);
    port (A: in STD_LOGIC_VECTOR (1 to nombre_entrees);
    B: out STD_LOGIC);
    end Porte_et;

  - Instanciation du composant :
  - Porte_et_4entrees : Porte_et generic map (nombre_entrees => 4)
    port map (A=> alfa, B=>beta);

  c'était défini au par avant :
    - Signal alfa : std_logic_vector (1 to 4);
    - Signal beta : std_logic;

2.12 Généricité

Instruction concurrente generate

- Permet l'élaboration itérative ou conditionnelle de lignes de code
- 2 formes : conditionnelle et itérative
- Forme conditionnelle
  - label : if condition_booléenne generate
    suite d'instructions concurrentes
    end generate {label};

- Forme itérative
  - for nom_du_paramètre_de_la_génération in intervalle discret generate
    suite d'instructions concurrentes
    end generate {label};

2.12 Généricité

- Exemple: registre à décalage générique (N)
  - entrées Data et Clock - std_logic;
  - sortie S est un std_logic_vector (N-1 downto 0)
  - architecture structurelle en utilisant un composant de type bascule D

```
entity reg_decalage is
        GENERIC( N: natural :=8);
        port (
                Data, clock : in std_logic;
                S : out std_logic_vector(N-1 downto 0));
end reg_decalage;
```


2.12 Généricité

```
ARCHITECTURE structurelle OF reg_decalage IS

    COMPONENT basculeD PORT (
                            D, clk  : in STD_logic;
                            Q : OUT STD_LOGIC ;
                            Qb : out std_logic );
    END COMPONENT;


    for all : basculeD use entity work.basculeD(beh3);

    begin

    gauche : basculeD port map (data, clock, S(N-1), open);

    boucle : for i IN 1 to N-1 generate
    circ : basculeD PORT MAP (S(N-i), clock, S(N-i-1), open);
    END GENERATE boucle;

END structurelle ;
```

2.12 Généricité

```
ARCHITECTURE structurelle OF reg_decalage IS

    COMPONENT basculeD PORT (
                                D, clk  : in STD_logic;
                                Q : OUT STD_LOGIC ;
                                Qb : out std_logic );
    END COMPONENT;
    signal aux : std_logic_vector (N-1 downto 0);

    for all : basculeD use entity work.basculeD(beh3);

    begin

    gauche : basculeD port map (data, clock, aux(N-1),open);

    boucle : for i IN 1 to N-1 generate
    circ : basculeD PORT MAP (aux(N-i), clock, aux(N-i-1), open);
    END GENERATE boucle;
    S <= aux ;

END structurelle ;
```


2.12 Généricité

```
ARCHITECTURE beh OF reg_decalage IS

begin

     comport : process
     begin
            wait until clock'event and clock ='1';
            s(n-1) <= Data ;
            copie : For i IN n-1 downto 1 LOOP
            s(n-i-1) <= s(n-i);
            end loop copie;
    end process comport ;

END beh;
```

2.12 Généricité

```
ARCHITECTURE beh OF reg_decalage IS

signal aux : std_logic_vector (N-1 downto 0);
 begin

        s <= aux ;
       comport : process
       begin
                wait until clock'event and clock ='1';
                aux(n-1) <= Data ;
                copie : For i IN n-1 to 1 LOOP
                aux(n-i-1) <= aux(n-i);
                end loop copie;
      end process comport ;

 END beh;
```


2.13. Compilation,élaboration, exécution, exploitation

- Utilisation du VHDL :
  - compilation et éditions de liens - aspects statiques de la description;
  - élaboration - aspects paramétrables de la description;
    - effectue l'instanciation de la description VHDL;
    - si la structure est hierarchique il faut commencer par le plus haut niveau
  - exécution - pour un simulateur <=> simulation
    - pour un synthétiseur <=> la synthèse
  - exploitation des résultats plus spécifique de la simulation

2.14. Synthèse

- C 'est un processus de traduction :
  - entrée : description comportementale
  - sortie : description structurelle à partir d 'éléments de base prédéfinis - standard cells
- Respect de la fonctionnalité originale
- Respect des contraintes en temps et en topologie
- Représentation physique :
  - ASIC
  - FPGA

2.14. VHDL et la synthèse

- Points forts :
  - généralité. Le domaine est en évolution rapide. La généralité du langage et sa capacité d 'abstraction le rend apte à supporter cette évolution.
  - lien avec la simulation
- Points faibles :
  - il n 'y a pas pour l 'instant de sous-ensembles pour la synthèse standardisés
  - la sémantique de VHDL pour la simulation est définie et standardisée, mais pas la sémantique VHDL pour la synthèse !

Synthèse à partir de VHDL

- Deux voies :
  - synthèse logique - industrie
    - spécification comportementale de niveau RTL (Register Transfer Level)
    - bibliothèque d'éléments matériels - design kit - standard cells
  - synthèse architecturale - encore du domaine de l 'investigation
    - spécification comportementale abstraite
    - compilation, allocation et partage de ressources, séquencement

2.14. Synthèse logique : principes

- Sous-ensemble de VHDL pour la synthèse:
  - descriptions synchrones (horloges explicites)
  - expressions de délais ignorées (clauses after)
  - restrictions sur l'écriture d'un process
  - seulement certains types sont permis
- Le détail des restrictions varie d'un fournisseur à l'autre
- On va présenter les restrictions minimales
- La configuration n'est pas supporté pour la synthése. Il faut que le nom du "component" = nom du model VHDL

2.14. Types pour la synthèse logique

- énumérés
- type BIT et opérations associées
- types du package STD_LOGIC

- entiers :
  - la plage de variation détermine le nombre de bits nécessaires
  - les bits ne sont pas individuellement accessibles

- Types non synthétisables
  - REAL, ACCES, FILE
  - Types physiques: TIME ou définis par l 'utilisateur

2.14. Synthèse - remarques

- usage des types énumérés fortement recommandé : on reste synthétique et on laisse à l'outil de synthèse le choix de la bonne stratégie de codage

- il n'y a pas de consensus quant au codage des types énumérés

- seuls les tableaux à une dimension sont synthétisables a cause de la difficulté du calcul d'adresse. Les agrégats sont mis à plat.

- les types prédéfinis du package IEEE_1164 sont fortement recommandés

2.14. Objets et synthèse logique

- Constantes
  - Acceptées pour tous les types synthétisables. Leur déclaration ne produit aucun matériel. Leur usage produit du matériel dans les cas suivants :
    - partie droite d 'affectation de signal
    - dans une instruction if ou case
    - dans une instruction concurrente conditionnelle
- Signaux
  - assimilables à des fils ou bus
- Variables (affectation de variable)
  - pas de règle générale pour déduire le matériel produit
  - c'est le contexte d 'utilisation qui est déterminant

2.14. Valeurs initiales

- En VHDL 3 types:
  - Valeurs par défaut héritée de la définition du type ou du sous-type
  - Initialisation explicite à la déclaration de l'objet
  - Valeur affectée par instruction au début d'un process
- Synthèse - les 2 premiers cas sont ignorés par les outils de synthèse

Paramètres génériques

- Permet de définir une famille de composants par rapport à une caractéristique donnée
- Valeur est fixé lors de l'instanciation de l'entité
- Synthèse :
  - Le type de paramètres génériques synthétisable est restreint en fonction de l'outil

2.14. Instructions séquentielles et synthèse logique

- Instruction de synchronisation :
  - sur signal de la liste de sensibilité (process (A,B,C) ou wait A,B,C)
    - matériel combinatoire
  - sur signal reconnu comme horloge (clk '**event** ou clk='1'). Seules les transitions de '0' à '1' ' sont reconnues ou l 'inverse
- Instructions d'itération
  - for : synthétisable dans la mesure où les bornes de variation de l'index sont statiques
  - while : non-synthétisable au niveau RTL
- Appel de sous-programmes
  - certains fonctions sont connues de la synthèse - pas de matériel additionnel
  - fonctions passage de paramètres par valeurs - combinatoire
  - fonctions passage de paramètres par référence - pas du domaine de la synthèse logique

2.14. Processus pour la synthèse logique

- Problème principal : combinatoire ou séquentiel ?

- Processus combinatoire si :
  - liste de sensibilité ou un seul point de synchronisation (wait)
  - pas de déclaration de variable ou bien variables locales systématiquement affectées avant d 'être lues
  - les signaux lus se trouvent tous dans la liste de sensibilité
  - Les signaux écrits sont tous affectés quelques soient les branchements parcourus

- Si création de mémoire - processus séquentiel
  - style synchrone - bascule D

2.14. Instructions concurrentes et synthèse logique

- Affectation de signal simple - matériel combinatoire

- Affectation conditionnelle ou sélectée
  - S <= A when X=' 1 ' else B when Y=' 1 ' else C - combinatoire
  - S <= A when X=' 1 ' else B when Y=' 1 ' else S - séquentiel

- Instanciation (de composants , generate)
  - Les notions d'entité et de composant permettent de hiérarchiser la description d'un système
  - Configuration - ne sont pas supportés
  - durant le processus de synthèse, différentes interprétations possibles
    - instances « aplaties »
    - composant synthétisé une fois. Seule la frontière est resynthétisée pour chaque instance
    - composant synthétisé une fois et dupliqué pour chaque instance

2.14. Synthèse architecturale

- Elle travaille sur des spécifications VHDL abstraites pour aboutir à un niveau où la synthèse logique puisse être appliquée
- Étapes :
  - transformations comportementales sur la spécification: élimination du code superflu ou redondant, mise à plat des boucles, propagation des constantes
  - partitionnement
  - ordonnancement
    - construction de la partie contrôle
  - allocation de ressources
    - construction de la partie chemin de données
  - production de la description RTL obtenue
- La synthèse logique est la continuation de la synthèse architecturale. En distinguant les deux, on distingue ce qu 'on sait bien faire de ce qu 'on ne maîtrise pas encore.

2.14. Remarque finale sur la synthèse

- La sémantique de VHDL pour la synthèse n 'est actuellement définie que pour un sous-ensemble du langage pour les raisons suivantes:
  - Les outils industriels n 'abordent pour l 'instant que la synthèse logique (niveau RTL)
  - Certaines constructions VHDL ne sont pas synthétisable

2.14. Performances

- Timming
  - Estimation des retards induits par les éléments combinatoires, en particuliers détection des "chemins critiques"
  - Estimation des retards induits par les éléments des mémoire
  - Estimation du temps de cycle de l'horloge
- Surface
  - Estimation en fonction des composants utilisées:
    - Standard cells
    - FPGA
- Consommation
- Testabilité, fiabilité, "manufacturabilité"

2.14. Contraintes temporelles

- 4 catégories de contraintes temporelles peuvent être introduites pour optimiser un circuit. --> temps minimums ou maximums entre différents points du circuits:
  - Entrées -registres
  - Registres - sorties
  - Entrées - sorties
  - Registres-registres
- Définition de la période de l'horloge globale $T_{periode}$ + le temps additionnel sur les entrées T_{in} et le temps additionnel sur le sorties T_{out}
- T_{out} - information sur la charge équivalente sur chaque sortie = fan-out
- T_{in} - information sur la puissance de sortie du composant connecté en entrée
- $T_{période}$
  - information de "skew" = déphasage maximal entre les arrivées de l'horloge sur les bascules
  - Information sur la latence = temps maximal entre la commutation de l'horloge et la dernière bascule recevant l'horloge; La latence intervient sur le "skew" entre les composants externes.

2.14. Optimisation temporelle

- Permet d'améliorer les caractéristiques de temps de traversée et de fréquence d'horloge
- Les outils de CAO ne modifient pas le nombre de bascules du circuits lors de l'optimisation
- Les outils CAO optimise la logique combinatoire interne

- Il faut isoler le chemin le plus long de traversée de la logique combinatoire = chemin critique --> OPTIMISATION DU CHEMIN CRITIQUE :
  - Toute logique combinatoire partagée entre le chemin critique et d'autres chemins de logique est dupliquée pour permettre une optimisation poussée sur le chemin critique.
  - Augmentation de la taille des circuits après une optimisation temporelle

Les FPGAs

Xilinx

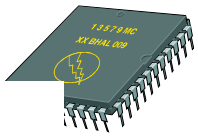
Spartan et Virtex

Structure des FPGAs

- CI Standards (portes, bascules, décodeurs, MPX…)
- CI ASIC 'sur mesure'
- CI Configurable (FPGA …)

Les différents choix :
Coût
Vitesse de conception
Vitesse d'exécution
Consommation/poids
Fiabilité …

Structure des FPGAs



- Circuits Semi-Custom configurables

Circuits programmables une fois ou plusieurs fois
Technologies Fusible ou SRAM

- PLD : PLA / PLS
  Programmable Logic Devices
- EPLD/CPLD Electrically PLD / Complex PLD
- FPGA
  Field Programmable Gate Array

ALTERA
AMD
LATTICE
XILINX
…

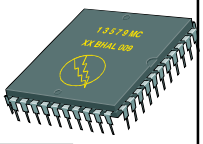
ACTEL
ALTERA
XILINX
…

Structure des FPGAs

- Techniques de programmation

OTP	**One Time Programmable** Fuse **ou** Antifuse
EPROM	**Electriquement programmable, effaçable au rayons UV**
EEPROM	**EPROM effaçable électriquement**
FLASH	**EEPROM à effacement collectif**
SRAM	**Static RAM : mémoire CMOS volatile** **+ ROM ou EEPROM externe**

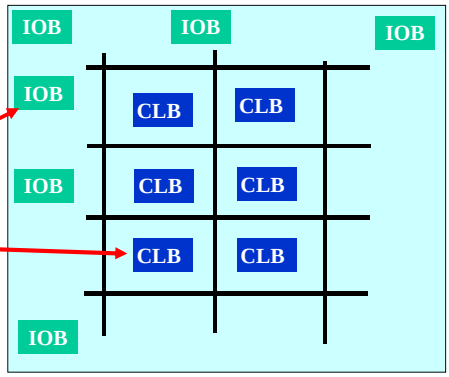
Structure des FPGAs



- Structure

IOB : Input/Output Block

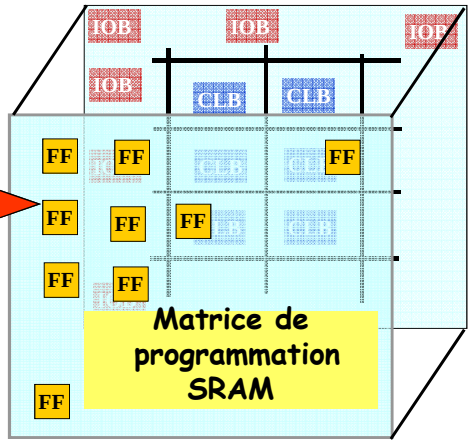
CLB : Configurable Logic Block



Structure des FPGAs

- Programmation

• Anti-Fusible
• Mémoire SRAM
(FPGA Spartan)



SPARTAN II

Structure des FPGAs (SPARTAN)

- Structure IO

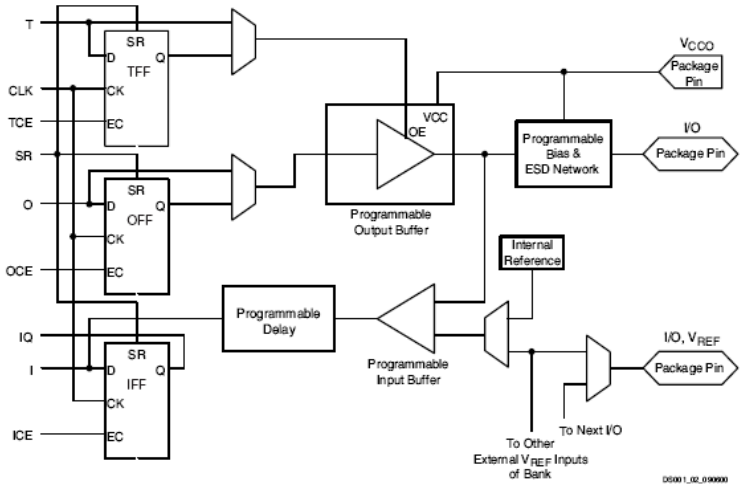


Figure 1: **Spartan-II Input/Output Block (IOB)**

- Structure CLB

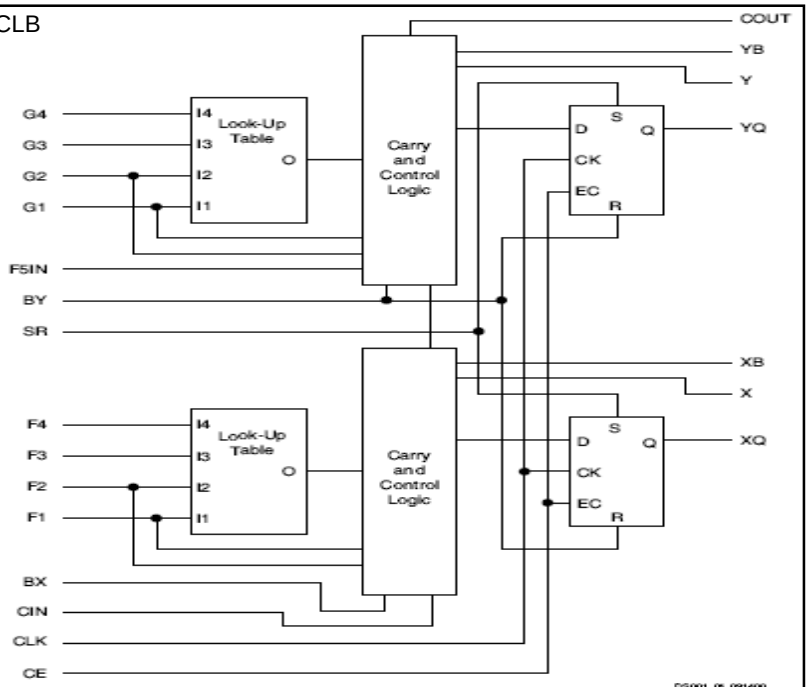
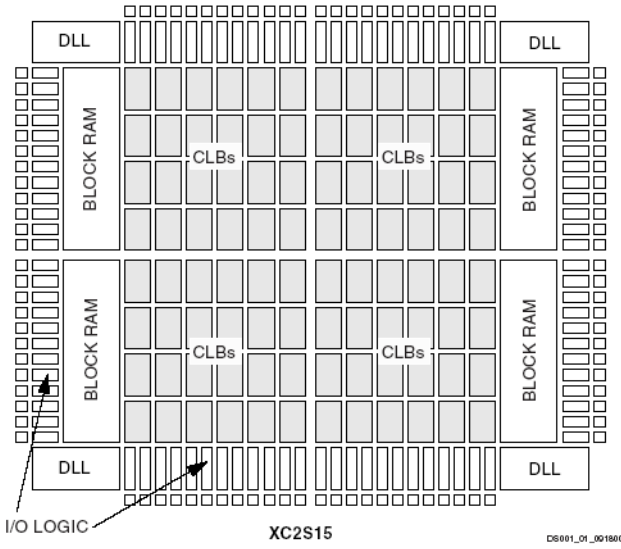


Figure 3: **Spartan-II CLB Slice** (two identical slices in each CLB)

FPGA Spartan II

Device	Logic Cells	System Gates (Logic and RAM)	CLB Array (R x C)	Total CLBs	Maximum Available User I/O (1)	Total Distributed RAM Bits	Total Block RAM Bits
XC2S15	432	15,000	8 x 12	96	86	6,144	16K
XC2S30	972	30,000	12 x 18	216	132	13,824	24K
XC2S50	1,728	50,000	16 x 24	384	176	24,576	32K
XC2S100	2,700	100,000	20 x 30	600	196	38,400	40K
XC2S150	3,888	150,000	24 x 36	864	260	55,296	48K
XC2S200	5,292	200,000	28 x 42	1,176	284	75,264	56K

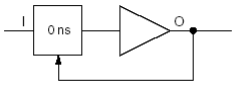
FPGA Spartan II



DLL (Delay Lock Loop)

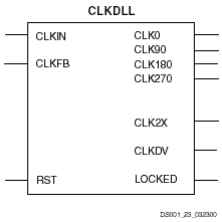
- Delay-Locked Loop (DLL) circuits which provide zero propagation delay and low clock skew between output clock signals distributed throughout the device.
- Each DLL can drive up to two global clock routing networks within the device. The global clock distribution network minimizes clock skews due to loading differences. By monitoring a sample of the DLL output clock, the DLL can compensate for the delay on the routing network, effectively eliminating the delay from the external input port to the individual clock loads within the device.
- The DLL can provide multiple phases of the source clock.
- The DLL can also act as a clock doubler or it can divide the user source clock by up to 16.

Library DLL Symbols



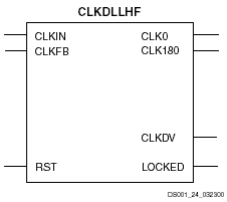
Simplified DLL Macro Symbol BUFGDLL

This macro delivers a quick and efficient way to provide a system clock with zero propagation
delay throughout the device.



Standard DLL Symbol CLKDLL

access to the complete set of DLL features



High-Frequency DLL Symbol CLKDLLHF

access to the complete set of DLL features

Block RAM Features

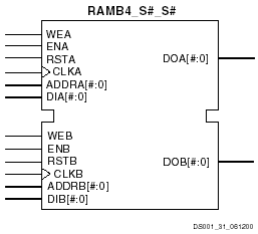
- The Spartan-II FPGA family provides dedicated blocks of on-chip, true dual-read/write port synchronous RAM, with 4096 memory cells.
- Each port of the block RAM memory can be independently configured as a read/write port, a read port, a write port, and can be configured to a specific data width.
- **Operating Modes**
  - Block RAM memory supports two operating modes.
    - Read Through
    - Write Back

Block RAM Features

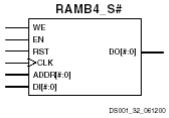
Block RAM Characteristics:

- 1. All inputs are registered with the port clock and have a setup to clock timing specification.
- 2. All outputs have a read through or write back function depending on the state of the port WE pin. The outputs relative to the port clock are available after the clock-to-out timing specification.
- 3. The block RAM are true SRAM memories and do not have a combinatorial path from the address to the output. The LUT cells in the CLBs are still available with this function.
- 4. The ports are completely independent from each other (*i.e.,* clocking, control, address, read/write function, and data width) without arbitration.
- 5. A write operation requires only one clock edge.
- 6. A read operation requires only one clock edge.
- 7. The output ports are latched with a self timed circuit to guarantee a glitch free read. The state of the output port will not change until the port executes another read or write operation.

RAM Library Primitives



Dual-Port Block RAM Memory



Single-Port Block RAM Memory

RAM Library Primitives

Table 10: **Available Library Primitives**

Primitive	Port A Width	Port B Width
RAMB4_S1	1	N/A
RAMB4_S1_S1		1
RAMB4_S1_S2		2
RAMB4_S1_S4		4
RAMB4_S1_S8		8
RAMB4_S1_S16		16
RAMB4_S2	2	N/A
RAMB4_S2_S2		2
RAMB4_S2_S4		4
RAMB4_S2_S8		8
RAMB4_S2_S16		16
RAMB4_S4	4	N/A
RAMB4_S4_S4		4
RAMB4_S4_S8		8
RAMB4_S4_S16		16
RAMB4_S8	8	N/A
RAMB4_S8_S8		8
RAMB4_S8_S16		16
RAMB4_S16	16	N/A
RAMB4_S16_S16		16

Table 11: **Block RAM Port Aspect Ratios**

Width	Depth	ADDR Bus	Data Bus
1	4096	ADDR<11:0>	DATA<0>
2	2048	ADDR<10:0>	DATA<1:0>
4	1024	ADDR<9:0>	DATA<3:0>
8	512	ADDR<8:0>	DATA<7:0>
16	256	ADDR<7:0>	DATA<15:0>

RAM Library Primitives

- Block RAM instances can have LOC properties attached to them to constrain the placement. The block RAM placement locations are separate from the CLB location naming convention,allowing the LOC properties to transfer easily from array to array.
- The LOC properties use the following form:

  LOC = RAMB4_R#C#

RAMB4_R0C0 is the upper left RAMB4 location on the device.

VIRTEX II

Virtex II

- Virtex2 successeur du Virtex-E
- Produit en Décembre 2000
- Gravé en 0.12µm
- Architecture totalement repensée
  - Grande rapidité entre les modules
- Multiplieurs câblées (18 bits x 18 bits)
  - Plus rapide que les portes logiques
  - Associé à un bloc mémoire
- Matrices de switches bufférisées
  - Réduit effet RC
- DCI –digitally controlled impedances

Virtex II

- Interconnections directes entre CLB
- Bloc mémoire double port
  - Entrées, sorties et Horloges séparées
  - Configurable de 16K * 1bit à 512 * 36 bits
  - 3 Modes d'écriture
    - Write First
    - Read First
    - No Change
  - Éviter l'écriture simultanée sur la même case mémoire

Virtex II

- Horloge d'origine partagée en 4 directions
  - FPGA mieux irrigué
- DCM
  - Avance de phase
  - Multiplication par 2, d'un quotient
- Horloge interne jusqu'à 420 MHz
- Horloge non utilisé, non alimenté
  - Gain en énergie

Distribution de l'horloge

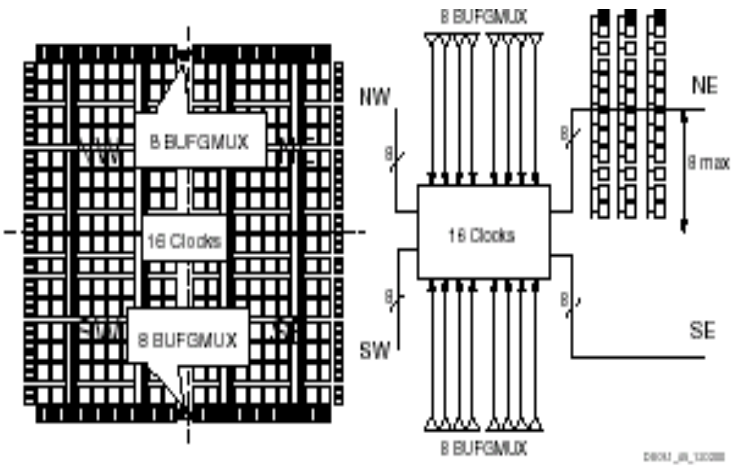


Figure 40: Virtex-II Clock Distribution

La famille Virtex II

Table 1: **Virtex-II Field-Programmable Gate Array Family Members**

Device	System Gates	CLB (1 CLB = 4 slices = Max 128 bits)			Multiplier Blocks	SelectRAM Blocks		DCMs	Max I/O Pads(1)
		Array Row x Col.	Slices	Maximum Distributed RAM Kbits		18 Kbit Blocks	Max RAM (Kbits)		
XC2V40	40K	8 x 8	256	8	4	4	72	4	88
XC2V80	80K	16 x 8	512	16	8	8	144	4	120
XC2V250	250K	24 x 16	1,536	48	24	24	432	8	200
XC2V500	500K	32 x 24	3,072	96	32	32	576	8	264
XC2V1000	1M	40 x 32	5,120	160	40	40	720	8	432
XC2V1500	1.5M	48 x 40	7,680	240	48	48	864	8	528
XC2V2000	2M	56 x 48	10,752	336	56	56	1,008	8	624
XC2V3000	3M	64 x 56	14,336	448	96	96	1,728	12	720
XC2V4000	4M	80 x 72	23,040	720	120	120	2,160	12	912
XC2V6000	6M	96 x 88	33,792	1,056	144	144	2,592	12	1,104
XC2V8000	8M	112 x 104	46,592	1,456	168	168	3,024	12	1,108

LAAS-CNRS D.D 119 WSN Team

Architecture Virtex II

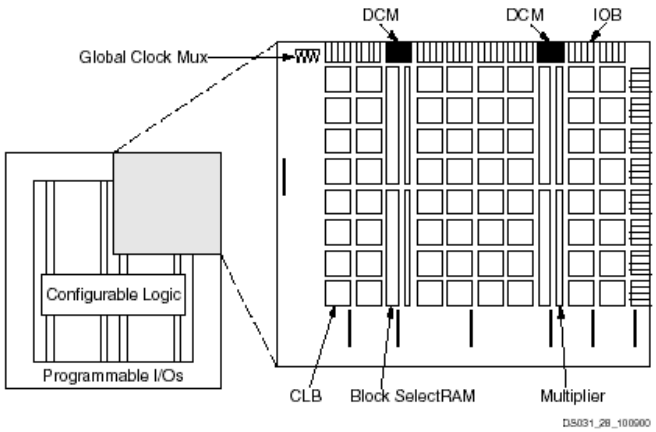


Figure 1: **Virtex-II Architecture Overview**

LAAS-CNRS D.D 120 WSN Team

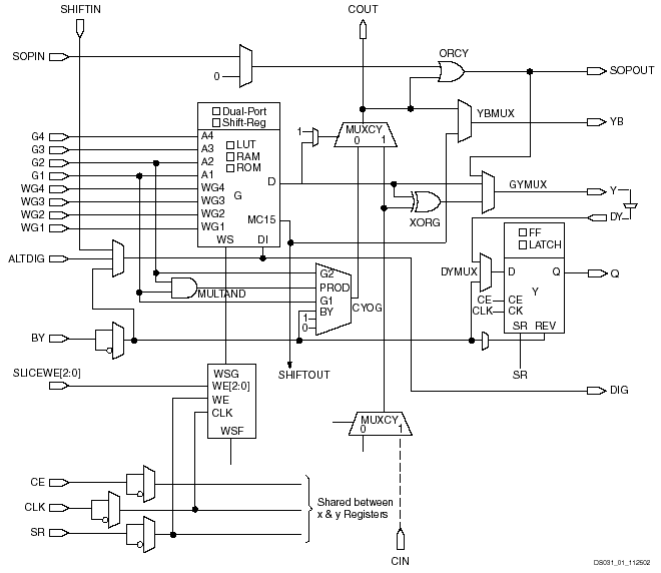


Figure 16: **Virtex-II Slice (Top Half)**

Blocks RAM

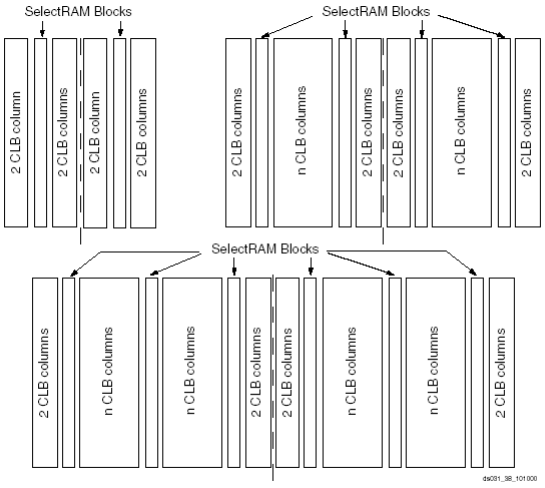
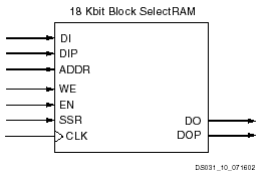
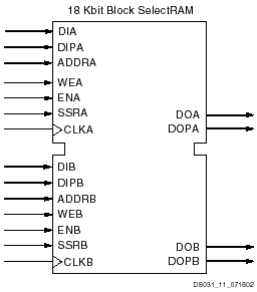


Figure 34: **Block SelectRAM (2-column, 4-column, and 6-column)**

Blocks RAM



18 Kbit Block SelectRAM Memory in Single-Port Mode



18 Kbit Block SelectRAM in Dual-Port Mode

18-Bit x 18-Bit Multipliers

Virtex-II devices incorporate many embedded multiplier blocks. These multipliers can be associated with an 18 Kbit block SelectRAM resource or can be used independently. They are optimized for high-speed operations and have a lower power consumption compared to an 18-bit x 18-bit multiplier in slices.

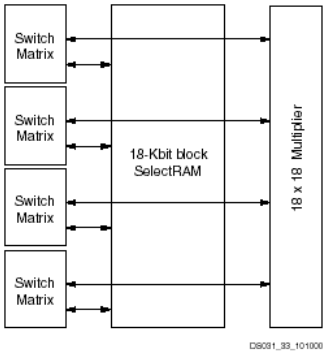


Figure 35: **SelectRAM and Multiplier Blocks**

18-Bit x 18-Bit Multipliers

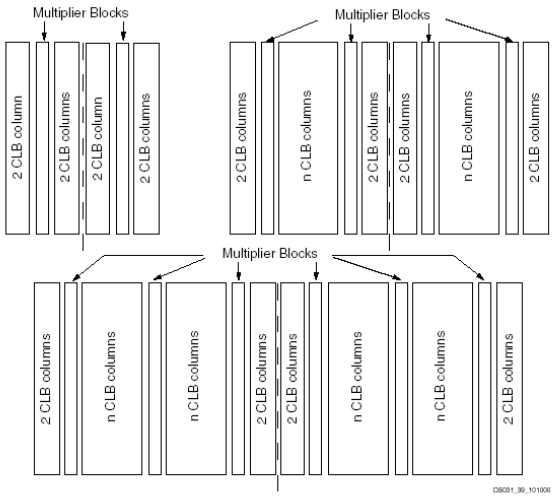
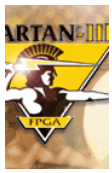


Figure 37: **Multipliers (2-column, 4-column, and 6-column)**

SPARTAN III



Spartan III

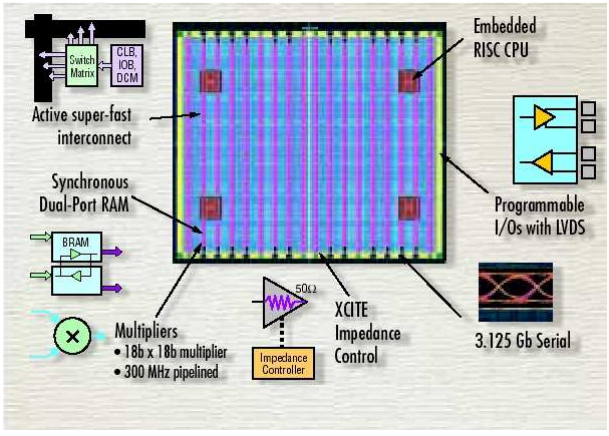
- Avant-Dernier FPGA de Xilinx
- Produit depuis 2003
- Bonnes idées du Spartan2 et Virtex2
- Architecture du Virtex2
- Gravé en 90 nm
- DCM, Multiplieurs câblées
- Forte densité des CLBs
- Fréquence max. de 326Mhz
- Prix de vente faible

Spartan III

Famille	Virtex2	Spartan3	Virtex2	Spartan3
Nom	XC2V250	XC3S200	XC2V1500	XC3S1500
Logic Cells	3456	4320	17280	29952
Multiplieurs	24	12	48	32
Max ram	432 Kbits	216 Kbits	864 Kbits	576 Kbits
DCM	8	4	8	4
CLB	384	480	1920	3328
Prix	158 dollars	?	600 dollars	?

Virtex II Pro

Xilinx Virtex II Pro



Xilinx Virtex II Pro

- Matrice de multiplieurs câblés hardware pour le support du traitement de signal parallèle
- Liaison série multi-giga bit pour communications inter-chip ou inter-système
- Microprocesseurs RISC pour réaliser des taches des décision et pour tourner des systèmes d'exploitation temps réel.
- Impédances configurables dynamique pour les entrées/sorties →simplification du système et du circuit imprimé contenant le FPGA

Platforme pour Radio Logiciel

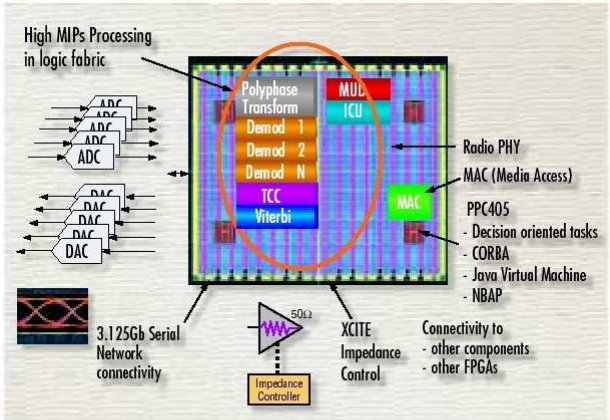
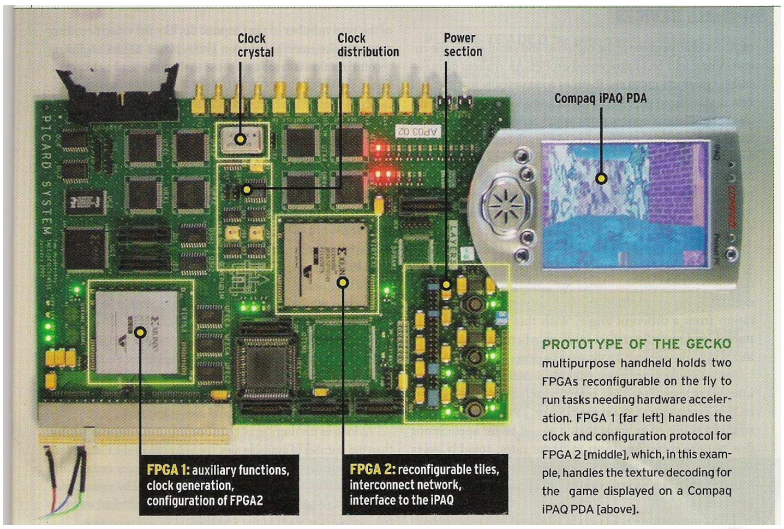
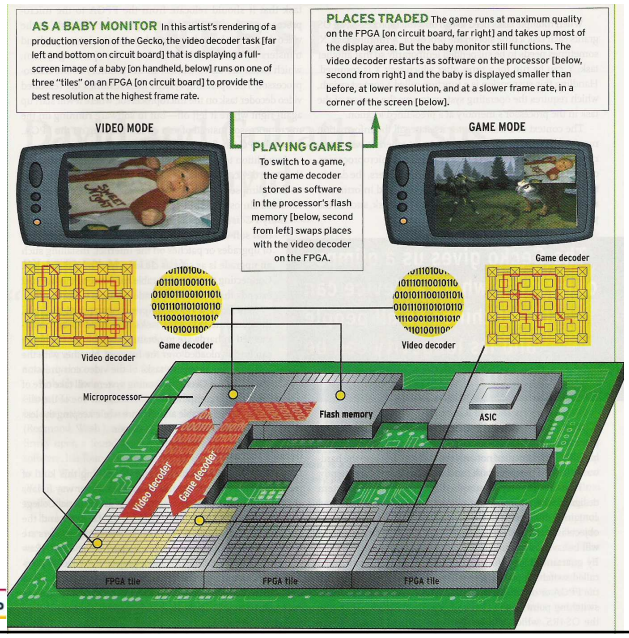


Figure 3. Platform FPGA approach to software-defined radio realization. The high MIPs processing is implemented in the logic fabric, while decision-oriented and non real-time tasks are provided as embedded software running on the Power PC. The multi-Gigabit transceivers could be used for providing connectivity to the broader network.

DO-IT-All Devices



DO-IT-All Devices

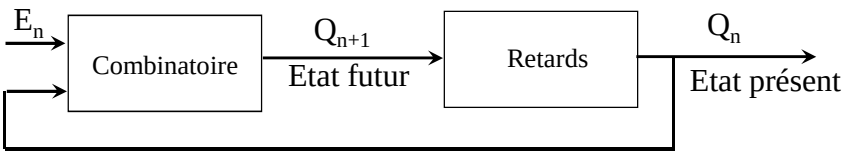


Compléments de cours

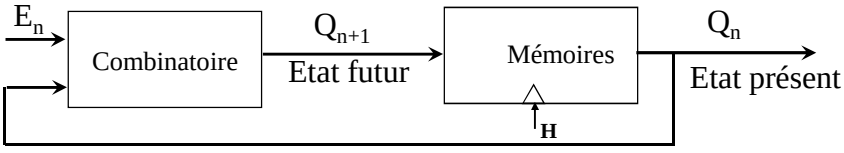
Exercices corrigés

Systèmes séquentiels
Synchrone et Asynchrone

- Système asynchrone ☞ Plus rapide mais de conception complexe



- Système synchrone ☞ Conception moins complexe

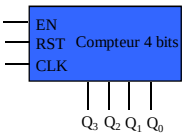


Exercice : Synthèse d'un compteur 4 bits modulo 10

- Logique séquentielle **synchrone**
- Utilisez de bascules D

EN = '1' - comptage autorisé
EN = '0' - comptage bloqué

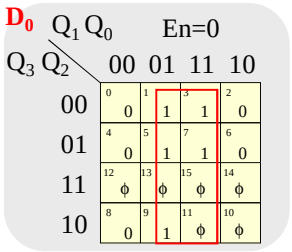
RST = '1' -RAZ
RST = '0' - comptage

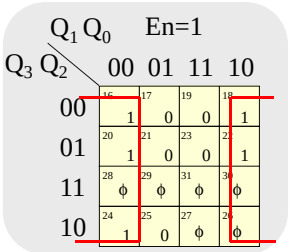


Q_3	Q_2	Q_1	Q_0	En	D_3	D_2	D_1	D_0	En	D_3	D_2	D_1	D_0
0	0	0	0	1	0	0	0	1	0	0	0	0	0
0	0	0	1	1	0	0	1	0	0	0	0	0	1
0	0	1	0	1	0	0	1	1	0	0	0	1	0
0	0	1	1	1	0	1	0	0	0	0	0	1	1
0	1	0	0	1	0	1	0	1	0	0	1	0	0
0	1	0	1	1	0	1	1	0	0	0	1	0	1
0	1	1	0	1	0	1	1	1	0	0	1	1	0
0	1	1	1	1	1	0	0	0	0	0	1	1	1
1	0	0	0	1	1	0	0	1	0	1	0	0	0
1	0	0	1	1	0	0	0	0	0	1	0	0	1

Sorties des bascules D | **état futur des entrées des bascules D pour EN='1'** | **état futur des entrées des bascules D pour EN='0'**

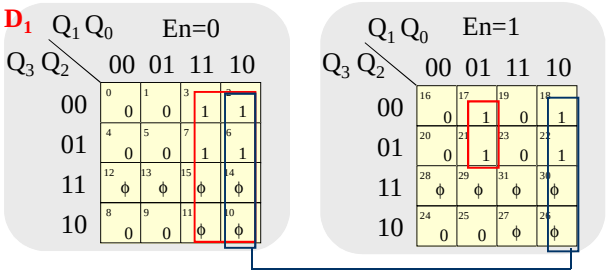
Exercice : Synthèse d'un compteur 4 bits modulo 10





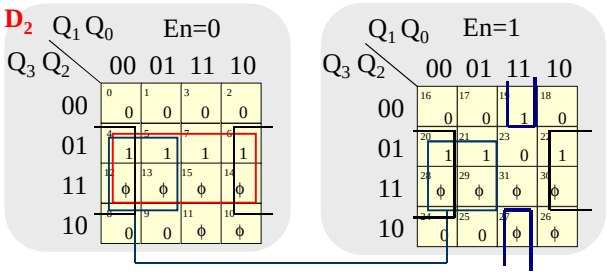
$$D_0 = Q_0 \oplus E_n$$

Exercice : Synthèse d'un compteur 4 bits modulo 10



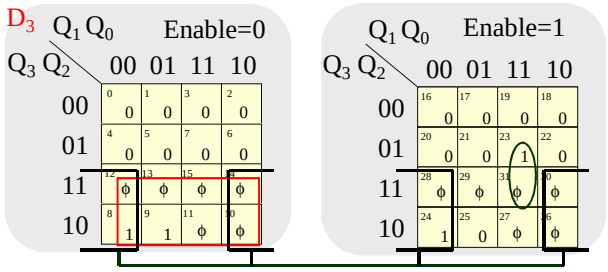
$$D_1 = \overline{E}_n Q_1 + Q_1\overline{Q}_0 + E_n\overline{Q}_3\overline{Q}_1Q_0$$

Exercice : Synthèse d'un compteur 4 bits modulo 10



$$D_2 = \overline{E}_n Q_2 + Q_2\overline{Q}_1 + Q_2\overline{Q}_0 + E_n\overline{Q}_2Q_1Q_0$$

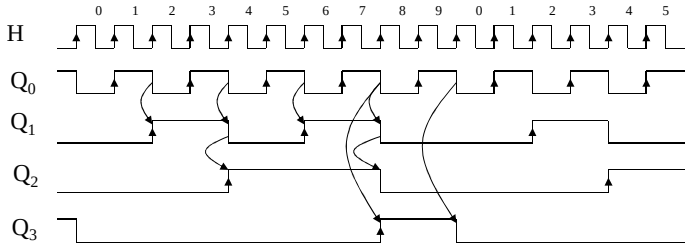
Exercice : Synthèse d'un compteur 4 bits modulo 10



$$D_3 = \overline{E_n}\, Q_3 + Q_3\overline{Q_0} + E_n Q_2 Q_1 Q_0$$

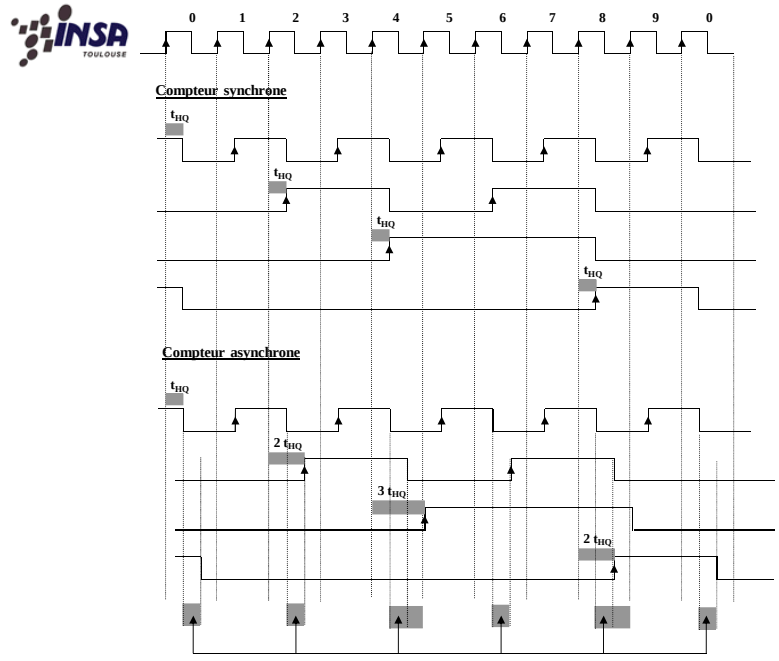
Exercice : Synthèse d'un compteur 4 bits modulo 10 asynchrone

- Les bascules ne sont plus reliées à la même horloge
- Choisir le signal de commande le plus adapté



H commande Q_0.
$\overline{Q_0}$ commande les bascule 1 et 3.
$\overline{Q_1}$ commande la bascule 2

$D_0 = \overline{Q_0}$
$D_1 = \overline{Q_3}\overline{Q_1}$
$D_2 = \overline{Q_2}$
$D_3 = Q_2 Q_1$



Régistre à decalage

```
Library IEEE;
Use IEEE.std_logic_1164.all;

entity reg_decalage is
        GENERIC( n: natural :=2);
        port (
                Data, clock : in std_logic;
                S : inout std_logic_vector(n-1 downto 0));
end reg_decalage;

Architecture structurelle OF reg_decalage IS

COMPONENT mem PORT ( D, clk  : in STD_logic;
                     Q : INOUT STD_LOGIC ;
                     Qb : out std_logic );
end COMPONENT;

begin

gauche : mem Port map (data, clock, S(n-1));
boucle : For i IN 1 to n-1 generate
                circ: mem PORT MAP (S(n-i), clock, S(n-i-1));
        END GENERATE boucle;
END structurelle ;

configuration reg_config of reg_decalage is
for structurelle
        for all: mem use entity work.basculeD(beh);
        end for;
end for;
end reg_config;
```

Registre à decalage

```
Architecture beh OF reg_decalage IS
 begin

 comport : process
                        begin
                         wait until clock ='1';
                         s(n-1) <= Data after 5 ns;
                         copie : For i IN n-1 to 1 LOOP
                         s(n-i-1) <= s(n-i) after 5 ns;
                         end loop copie;
end process comport ;

end beh;
```

Conception d'un compteur synchrone 8 bits

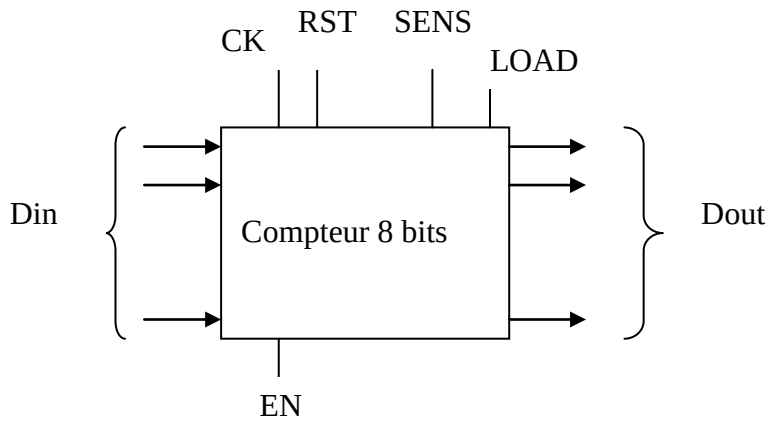
Objectifs du TD :

- Apprendre à écrire son premier code VHDL
- Apprendre et maitriser le flot de conception FPGA (synthèse, placement routage et implémentation sur FPGA)
- Découvrir le logiciel XILINX ISE

Spécifications :

Il faut écrire le code VHDL décrivant le comportement d'un compteur 8 bits **synchrone** sur le front montant. Ce compteur a les signaux suivants :

CK	- horloge.
RST	- reset – signal de remise à zéro du compteur **synchrone** avec l'horloge (actif bas).
LOAD	- signal de commande **synchrone** de chargement du compteur (actif haut).
SENS	- à l'état bas, le circuit décrément à chaque transition montante de l'horloge, - à l'état haut, le circuit incrément.
EN	- enable – permet au compteur de compter s'il est à la masse (actif bas)
Din	- donnée à charger dans le compteur (sur 8 bits) quand la commande LOAD est active
Dout	- sortie sur 8 bits



Les signaux CK, SENS, RST et LOAD sont du type **std_logic**.
Les signaux Din et Dout sont des **std_logic_vector**.

A chaque front montant de l'horloge, si le signal de RESET est activé (égal à 0), la remise à zéro du compteur sera faite. Le signal RESET a la plus grande priorité. Si le signal de LOAD du compteur est activé (égal à 1) sur le front montant d'horloge, la donnée présente sur les entrées Din sera chargé dans le compteur et affiché sur la sortie Dout. Tant que le signal de

LOAD est actif, la sortie du compteur est maintenue à la valeur de Din. Si le signal LOAD n'est pas actif et le signal ENABLE est actif (égal à 0), le compteur va compter ou décompter en fonction de la valeur du signal SENS.

Travail à faire :

1. Ecrire le code VHDL correspondant.
2. Ecrire le programme de test de ce compteur.
3. Simuler.
4. Faire la synthèse de ce code VHDL.
    - Regarder le circuit logique obtenu
    - Regarder le rapport de synthèse et vérifier le nombre des bascules obtenus et la fréquence maximale de fonctionnement
5. Effectuez le placement routage.
6. Effectuer la simulation du compteur 8 bits placé et routé. Comparez avec la simulation fonctionnelle du compteur (code VHDL).
    - Trouver la fréquence maximale de fonctionnement de votre compteur (une indication est donnée dans le rapport après synthèse, mais n'oubliez pas que lors de la synthèse les temps de propagation sur les interconnexions ne sont pas pris en compte)
    - Faites fonctionner le compteur à une fréquence très grande, pour bien le voir décrochez (il ne compte plus correctement)
    - Il y a-t-il des états aléatoires dans votre circuit ? Sont-ils gênants ?

7. Implémenter le compteur 8 bits sur le FPGA. Placez :
    - Din sur les switch
    - RST, LOAD,SENS, EN sur les bottons poussoirs
    - Dout sur le LEDs
    - CLK à ralentir à 1-3Hz (clk de la maquette à 50MHz). Concevez votre circuit diviseur d'horloge en VHDL.

Réalisation d'un contrôleur DMA

Concevez l'architecture et écrivez le code VHDL correspondant du contrôleur DMA décrit par la suite.

Le transfert des données entre les périphériques (disque dur par exemple) et la mémoire RAM est géré par le contrôleur DMA. Pendant le temps de transfert des données entre le disque dur et la mémoire, le processeur est déconnecté du bus des données et le contrôleur DMA devient le maître du système des bus.

Le contrôleur DMA active les bus des données et d'adresses pour gérer le transfert entre le périphérique et la mémoire. La méthode la plus utilisée consiste à employer un signal de contrôle spécial qui s'appelle « bus request » BR à '1', signal envoyé par le DMA vers le microprocesseur pour lui demander l'utilisation du bus. Le µP fini l'exécution de l'instruction en cours, relâche les bus en mettant les lignes en haute impédance et il envoie le signal « bus grant » BG = à '1' vers le contrôleur DMA. A la fin de transfert DMA, le contrôleur met le signal BR à '0'et le µP reprend le contrôle.

Une fois que BG ='1', le contrôleur DMA prend le contrôle des bus pour communiquer directement avec la mémoire.

Le bus de données et la taille des mots dans la mémoire est de 32 bits.

Pendant un transfert DMA un seul mot de 32 bits peut être transféré ou un bloc entier contenant plusieurs mots. Le contrôleur DMA à réaliser est présenté dans la Fig.1

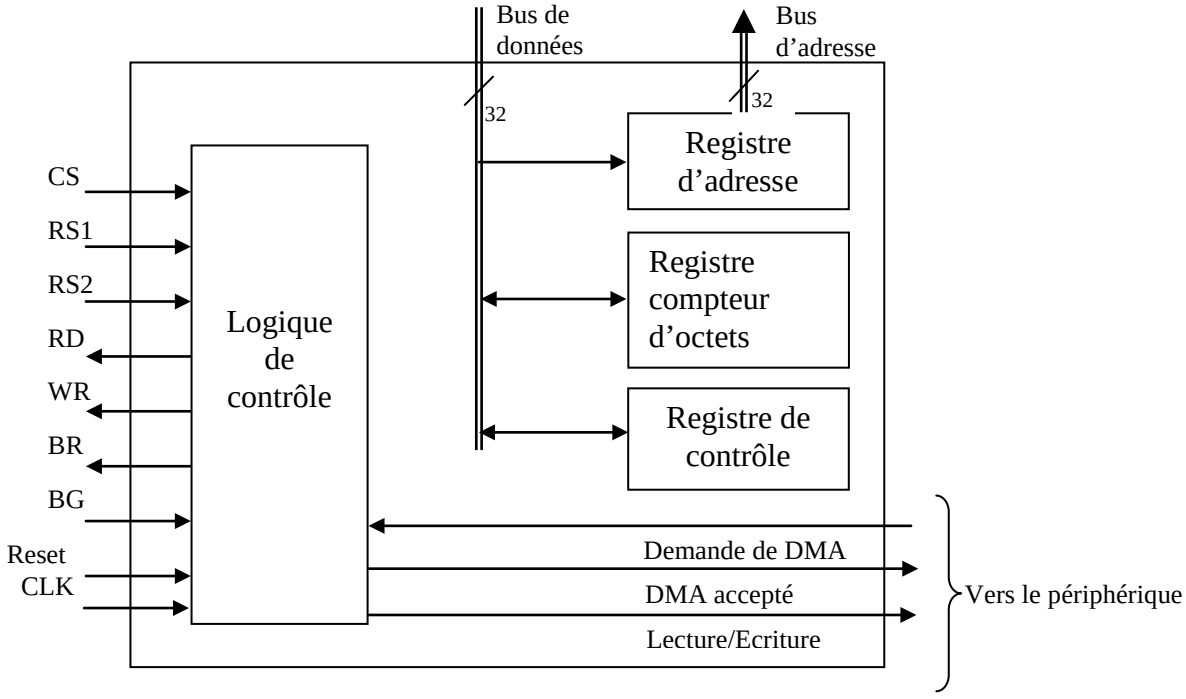


Figure 1. Contrôleur DMA

Le contrôleur dispose de trois registres :

Registre d'adresse : C'est un registre de 16 bits qui spécifie l'adresse du mot de la mémoire principale (RAM) que l'on veut accéder. La valeur du registre doit sortir sur le bus d'adresse qui a une taille de 32 bits. Le registre d'adresse est incrémenté à chaque transfert DMA d'un octet. Comme le bus a 4 octets, le registre s'incrémente de quatre en quatre.

Registre compteur d'octets : C'est un registre de 16 bits qui est initialisé avec le nombre d'octets à transférer lors d'un accès DMA. Il est décrémenté après chaque transfert de quatre en quatre.

Registre de contrôle : C'est un registre de 3 bits. Il sert à indiquer le type de transfert désiré, soit en lecture (de la mémoire vers le périphérique= bit 1 du registre à '1') soit en écriture (du périphérique vers la mémoire = bit 2 du registre à '1') et pour programmer le transfert DMA (le bit 0 du registre à '1').

La programmation de ces trois registres est faite par le µP, en utilisant le bus de données et les signaux de contrôle CS, RS1, RS2. Les registres du DMA sont sélectionnés par le µP en activant registre d'adresse ; RS1=1 et RS2=0 au registre compteur et RS1=0 et RS2=1 au registre de contrôle.

Quand le signal BG='1' le DMA peut communiquer directement avec la mémoire en spécifiant une adresse sur le bus d'adresse et en activant un de signaux de contrôle RD ou WR.
Le contrôleur DMA communique avec le périphérique à travers les signaux de « demande DMA », de « DMA accepté » et « lecture/écriture » (voir fig.1). « Lecture » – transfert de la mémoire vers le périphérique et « écriture »- transfert du périphérique vers la mémoire. Le contrôleur DMA met le signal « DMA accepté » à '1' quand le µP a donnée le contrôle au contrôleur DMA. Quand le périphérique fait une demande de DMA (signal « demande DMA » ='1'), le contrôleur demande le bus au µP(a vous de voir les signaux qu'il faut affecter). Le périphérique est connecté directement au bus de données.

Le transfert DMA fini quand le registre compteur arrive à zéro ou quand le périphérique met le signal « demande DMA » à zéro.

Tout le fonctionnement du DMA est synchrone sur le front montant de l'horloge (signal CLK).
Le reset est lui aussi synchrone avec l'horloge. Le signal de reset est active bas '0' et provoque la mise à zéro des 3 registres.