Méthodologie de conception et langages de description matérielle

Support de cours

Master CAMSI

Daniela Dragomirescu

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
- Advanced ASIC Chip Synthesis - H. Bhatnagar - Kluwer Academic Publisher
- Principles of CMOS VLSI Design: A system Perspective - N. Weste, K.Eshraghian - Addison Wesley

Table de matières

Table de matières détaillé

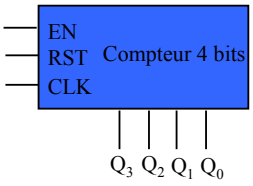
Table de matières

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**

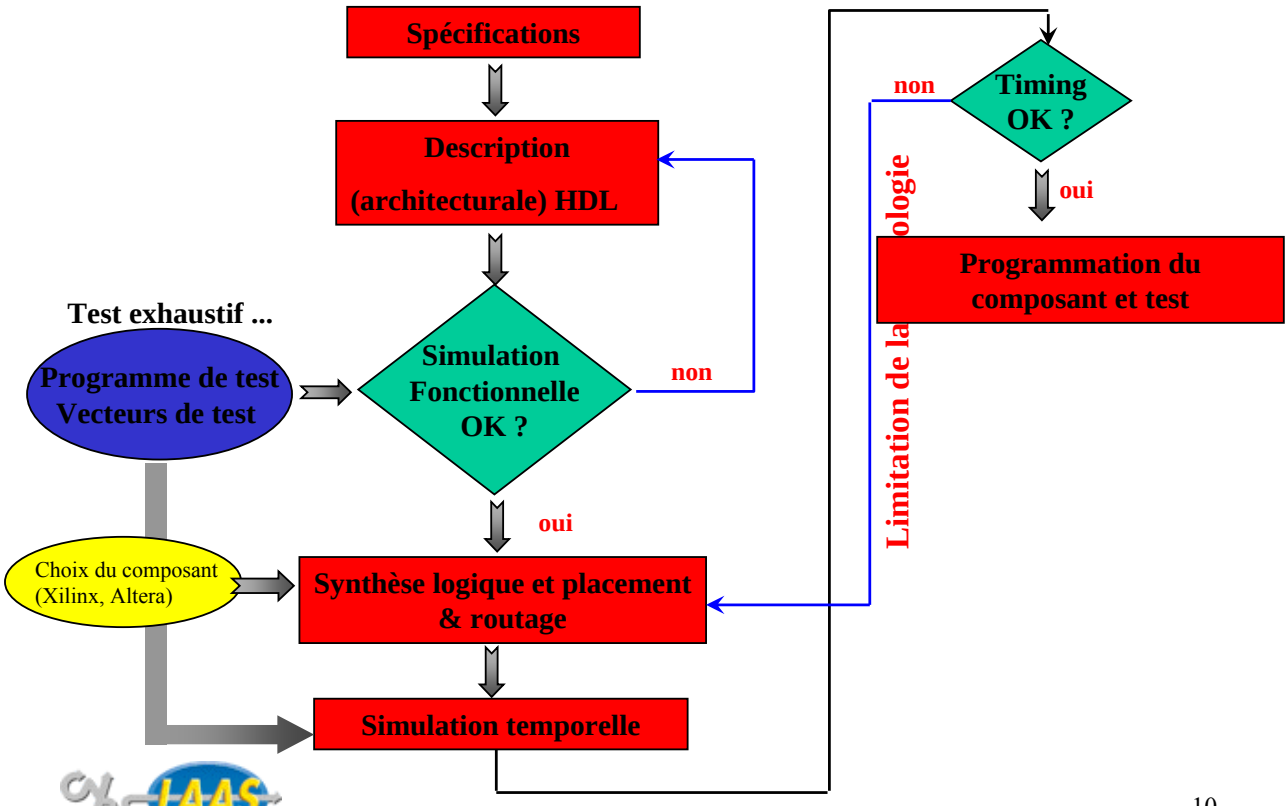
Introduction

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

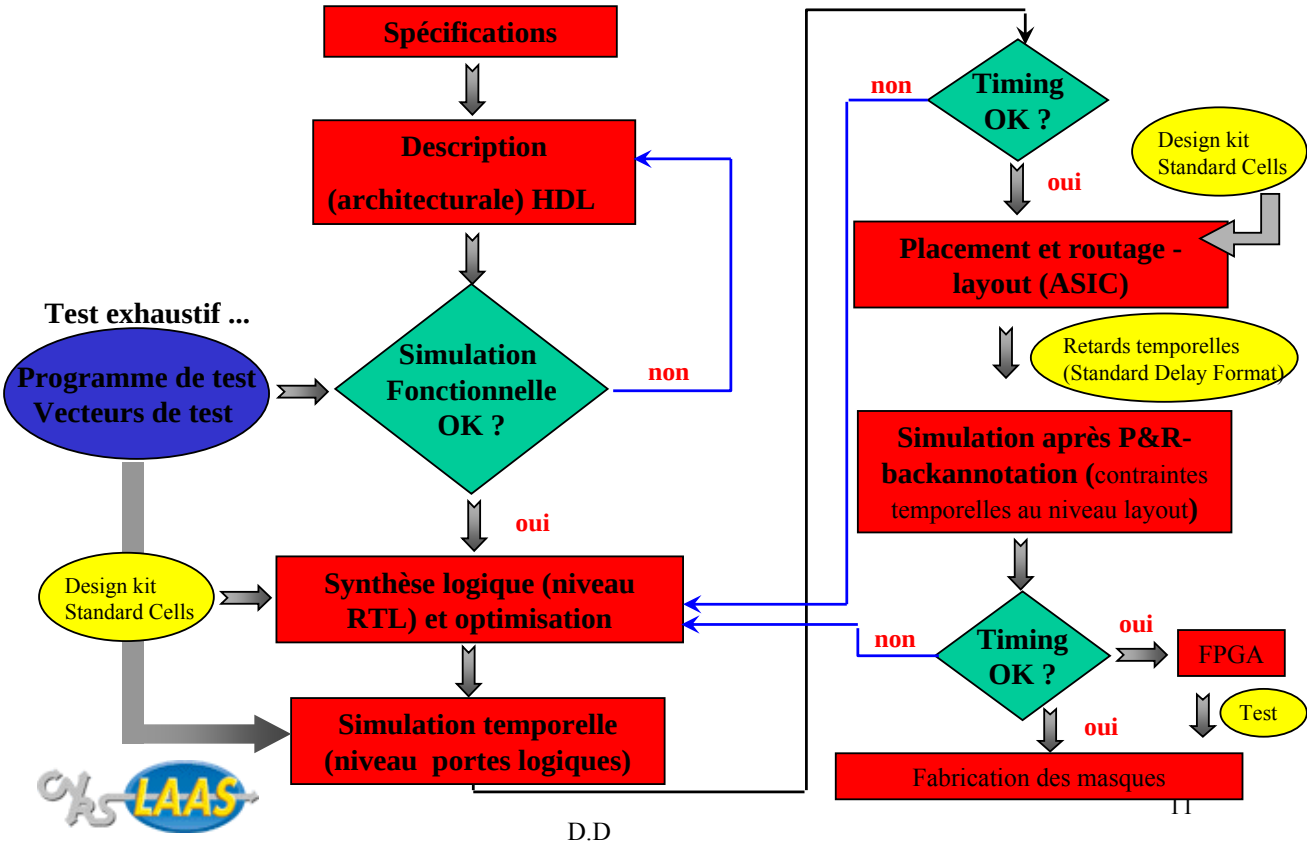
  } **Même principe, syntaxe différente**
- Travail avec des cellules standard - briques de base appartenant à une bibliothèque spécifique à chaque fondeur de circuits intégrés

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 à la microélectronique ? ⇐ 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

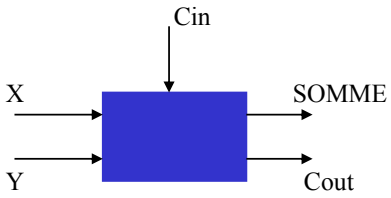
- IEEE Standard 1076 - 1987
- IEEE Standard 1076 - 1993
- IEEE Standard 1076 - 2003

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



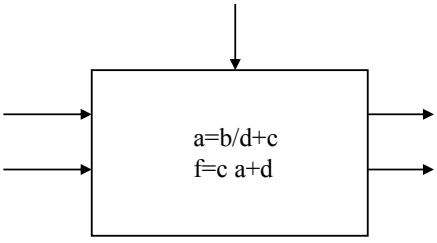

```
entity adder is
        port (
                X, Y, Cin:      in bit;
                Somme, Cout:    out bit);
end adder;
```

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



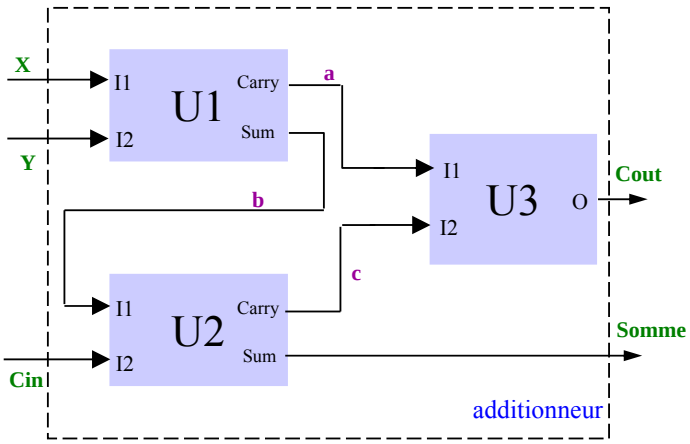
- Style de description de l 'architecture :
  - Description structurelle : interconnexion de composants
  - Description flot de données: comportement sous forme d'équations
  - Description comportementale: comportement sous forme algorithmique
- On peut combiner les différents styles des descriptions 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) after 20 ns;
        Cout <= carry_vector(N) after 30 ns;
        wait on X, Y, Cin;
    end process;
end vue_comprtementale;
```

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

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  ma_lib;

for  all : demi_additionneur use entity
        ma_lib.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;

    type  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;
                    Sum, Cout : out std_logic);
    END COMPONENT;

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

    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"
  - 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
  - type mon_integer is INTEGER range -65_536 to 65_535 ;
- Les flottants
  - type mon_flottant is REAL range 5.36 downto 2.15;
- Les types énumérés
  - type COULEUR is (BLEU, 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 after 10 ns;
            end if;
   end process;
```

```
process
  begin
    wait until CLK'event and CLK ='1';
    q <=d after 10 ns;

  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

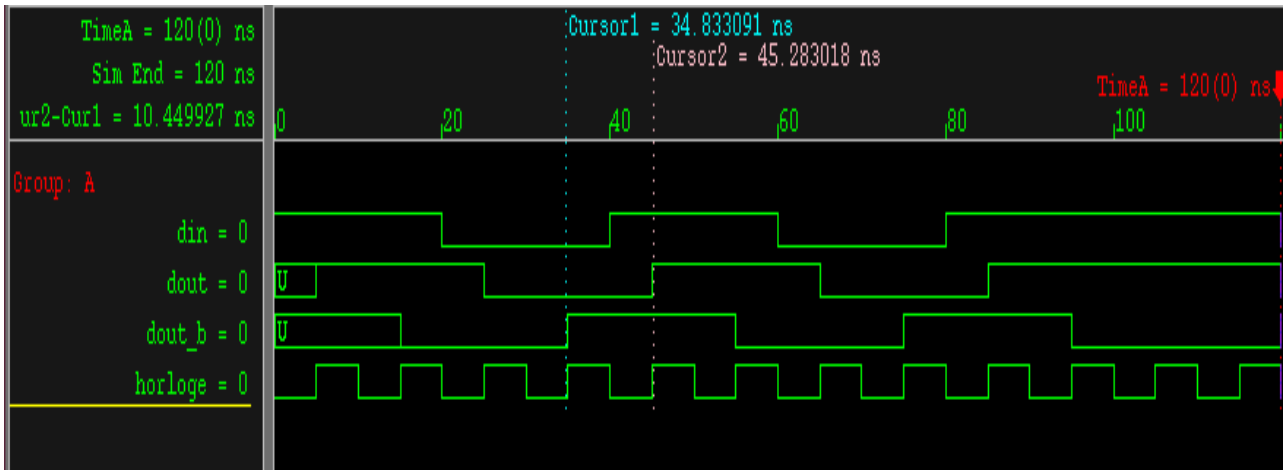
Bascule D

```
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 ;
```

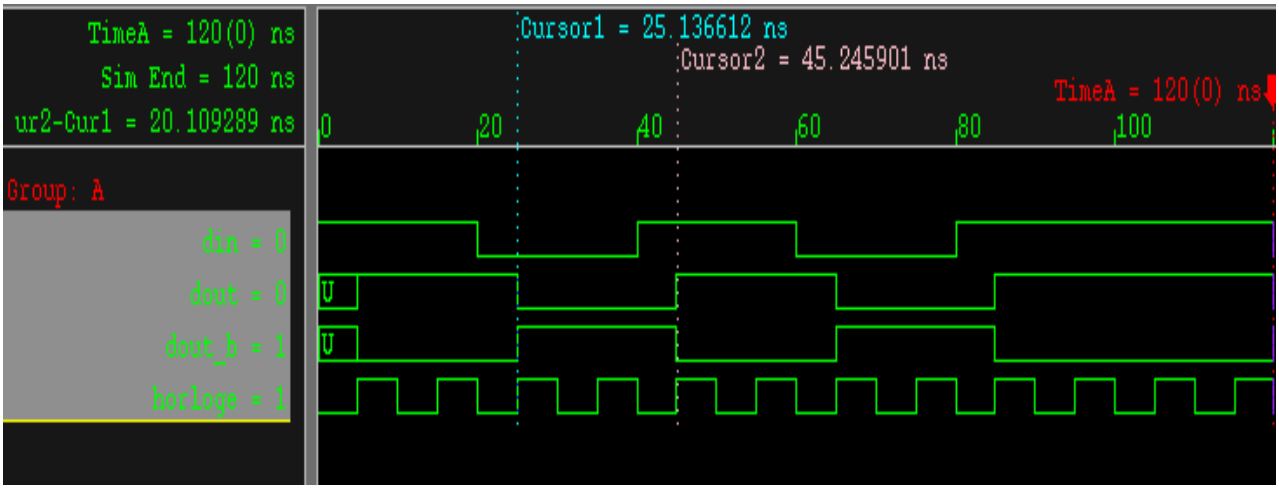
Architecture incorrecte

```
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
  Qb <= not Q;
  process
    begin
    wait until clk'event and clk='1';
    Q <= d;
  end process;
end beh2 ;
```


Bascule D - architecture beh1



Bascule D - architecture beh2

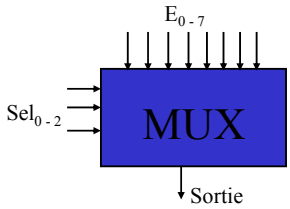


Bascule D

```
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(beh);

    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 to 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
  - Pour SYNOPSYS : seulement les paramètres génériques de type INTEGER ou énumérés sont synthétisable.

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

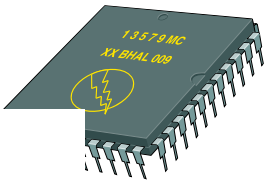
D.D

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
…

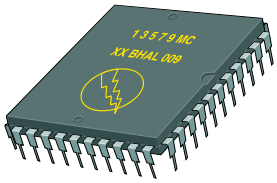
ACTEL
ALTERA
XILINX
…

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**

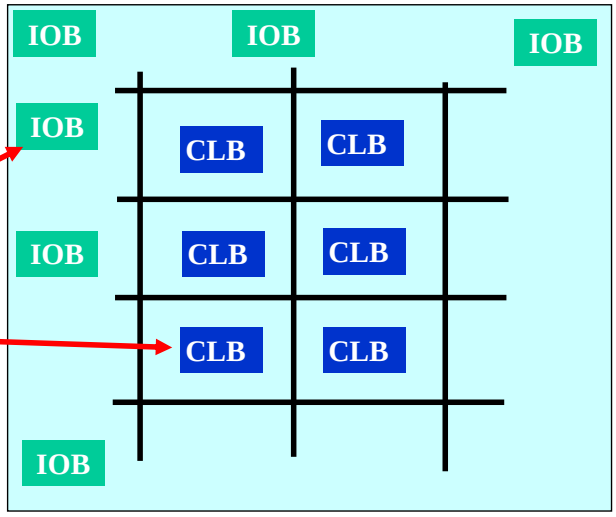
Structure des FPGAs



- Structure

IOB : Input/Output Block

CLB : Configurable Logic Block

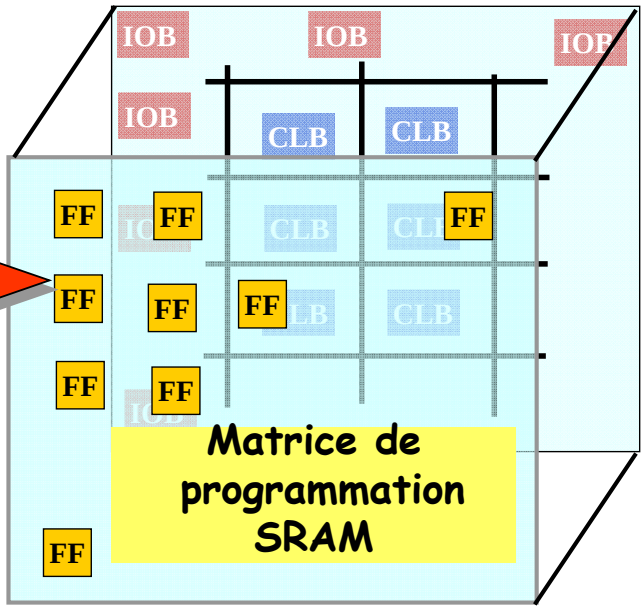


Structure des FPGAs

- Programmation

- **Anti-Fusible**
- **Mémoire SRAM**

(FPGA Spartan)



SPARTAN II

D.D

Structure des FPGAs (SPARTAN)

- Structure IO

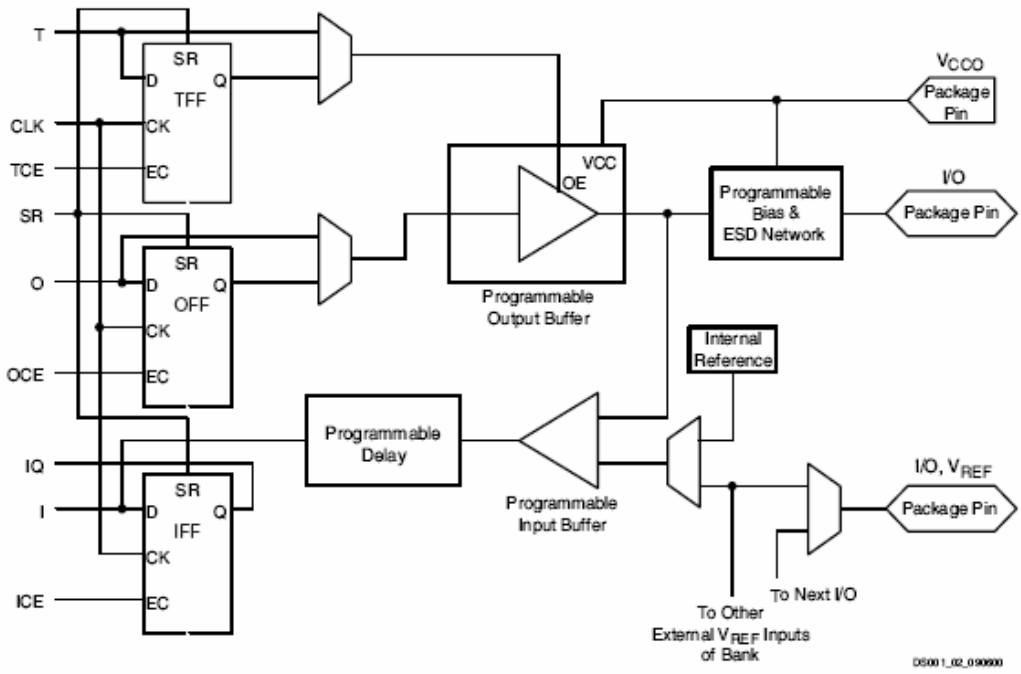


Figure 1: **Spartan-II Input/Output Block (IOB)**

❖ Structure CLB

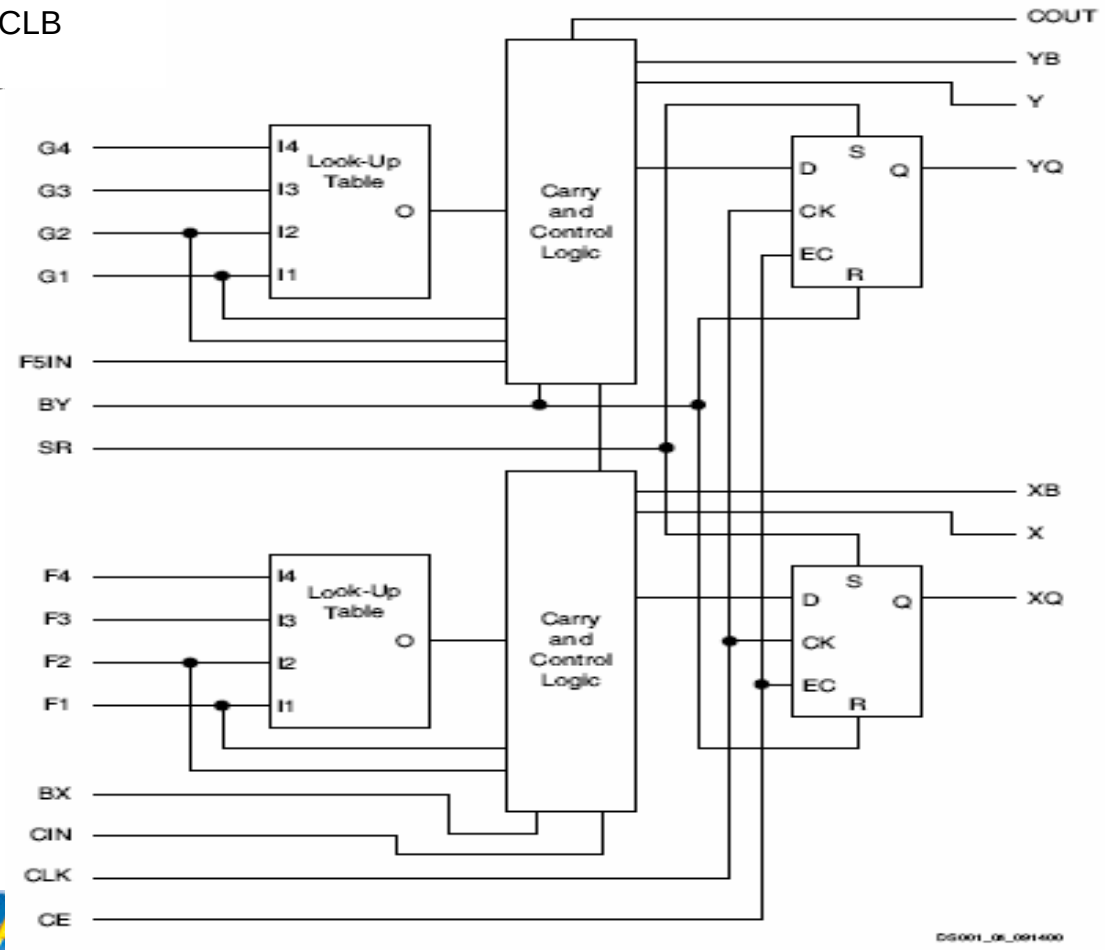
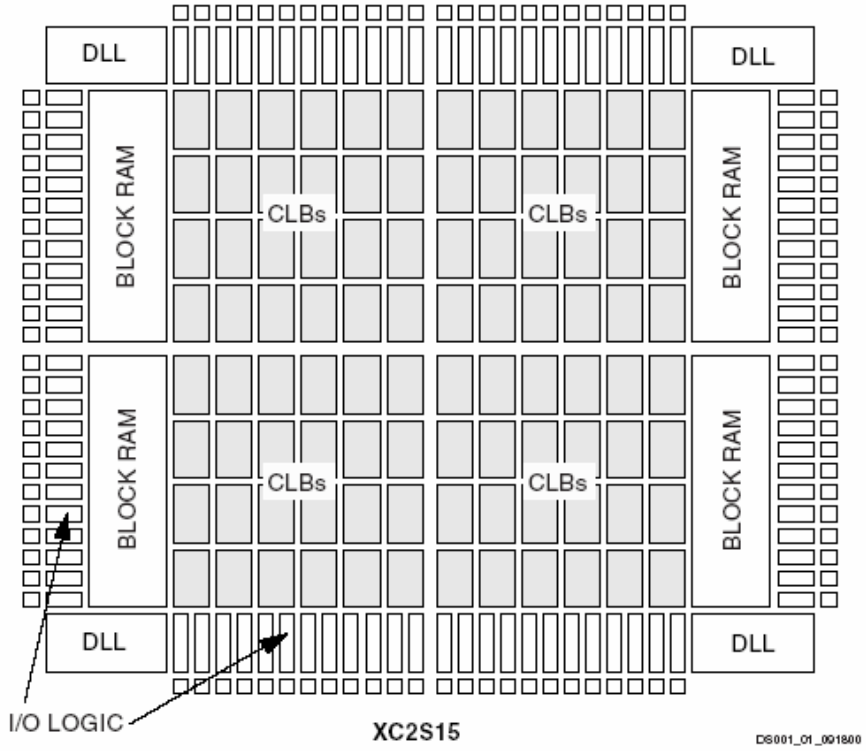


Figure 3: **Spartan-II CLB Slice** (two identical slices in each CLB)

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

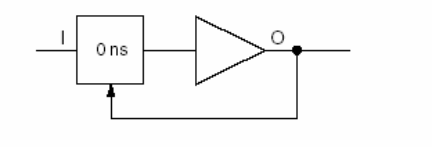
FPGA Spartan II



DLL (Delay Lock Loop)

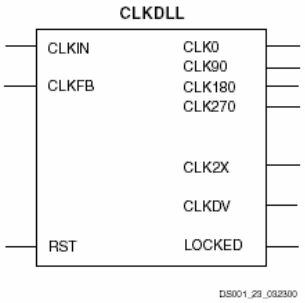
- Delay-Locked Loop (DLL) circuits which provide zero propagation delay and low clock skew between output clock signals distributed throughout the device.
- Each DLL can drive up to two global clock routing networks within the device. The global clock distribution network minimizes clock skews due to 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



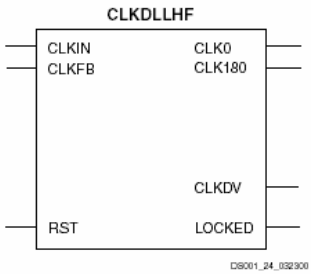
Simplified DLL Macro Symbol BUFGDLL

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

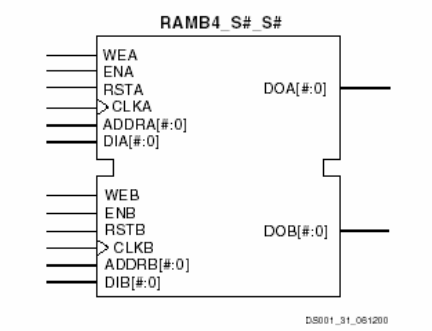
- The Spartan-II FPGA family provides dedicated blocks of on-chip, true dual-read/write port synchronous RAM, with 4096 memory cells.
- Each port of the block RAM memory can be independently configured as a read/write port, a read port, a write port, and can be configured to a specific data width.
-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

Block RAM Features

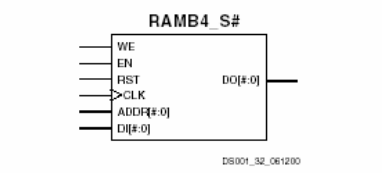
Block RAM Characteristics:

- 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

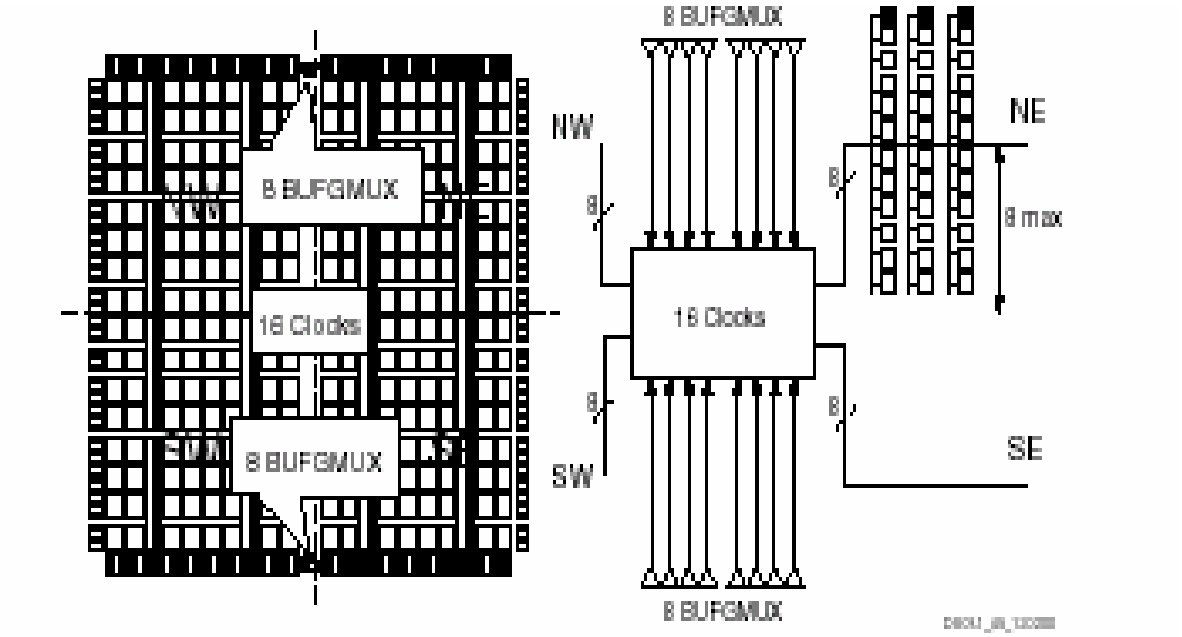


Figure 40: Virtex-II Clock Distribution

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) Array Row x Col.	Slices	Maximum Distributed RAM Kbits	Multiplier Blocks	SelectRAM Blocks 18 Kbit Blocks	Max RAM (Kbits)	DCMs	Max I/O Pads(1)
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

Architecture Virtex II

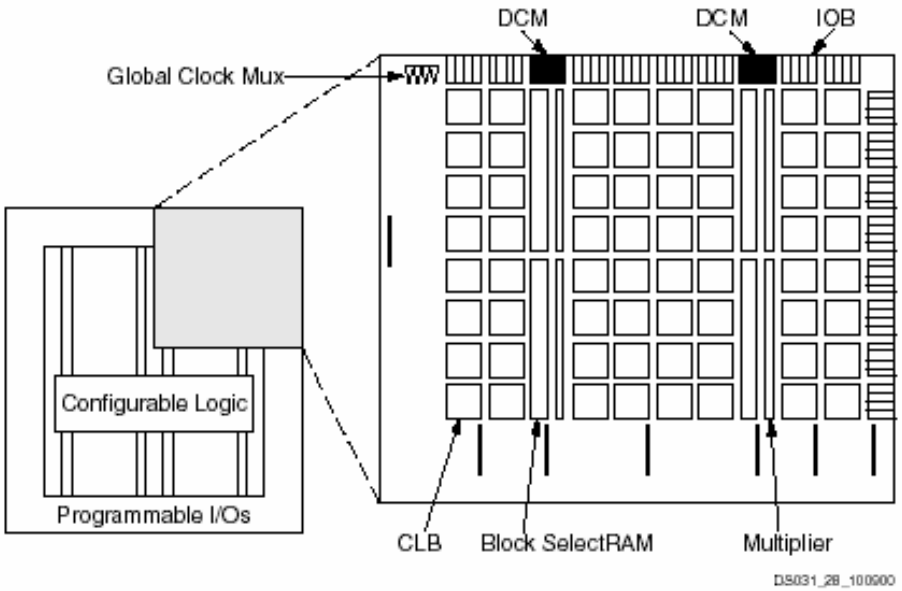


Figure 1: **Virtex-II Architecture Overview**

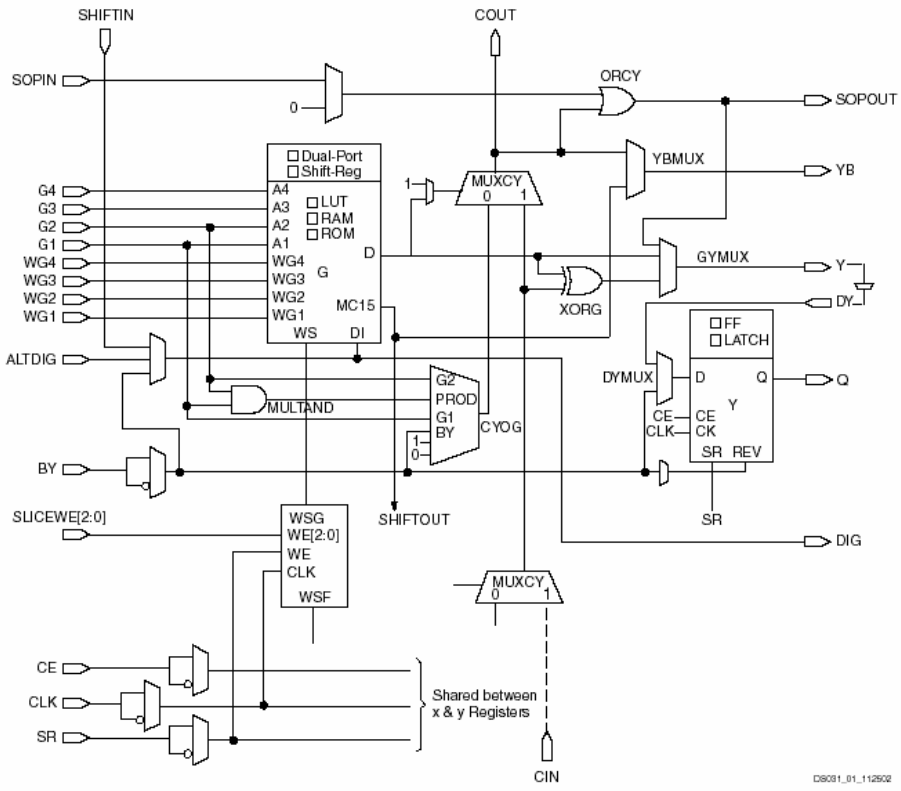


Figure 16: **Virtex-II Slice (Top Half)**

Blocks RAM

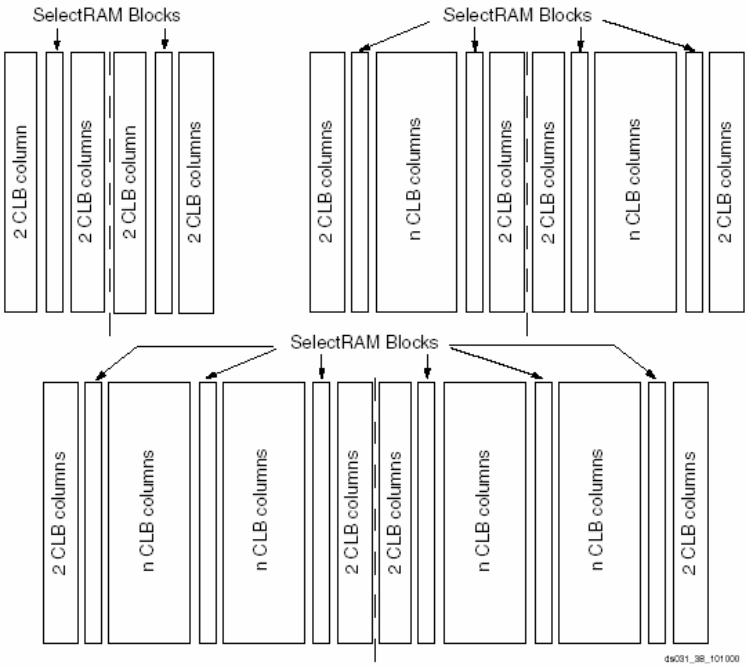
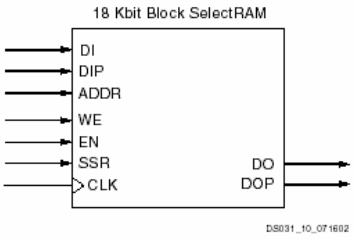
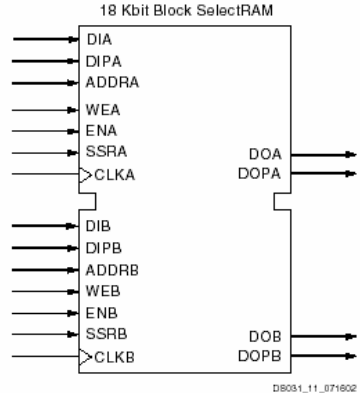


Figure 34: **Block SelectRAM (2-column, 4-column, and 6-column)**

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.

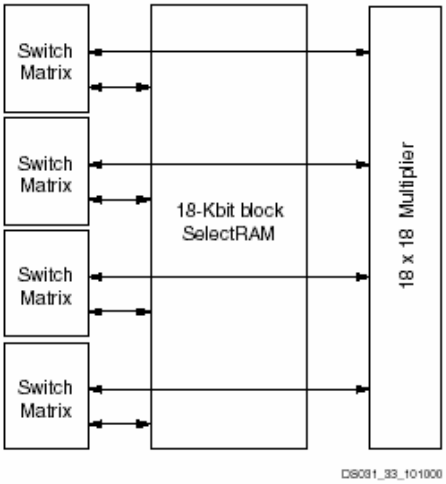


Figure 35: **SelectRAM and Multiplier Blocks**

18-Bit x 18-Bit Multipliers

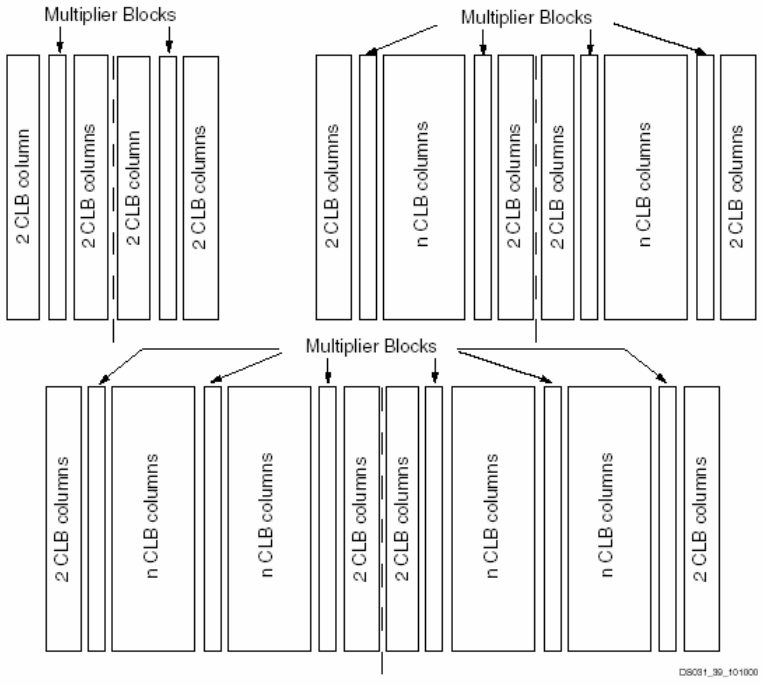


Figure 37: **Multipliers (2-column, 4-column, and 6-column)**

SPARTAN III

D.D

Spartan III

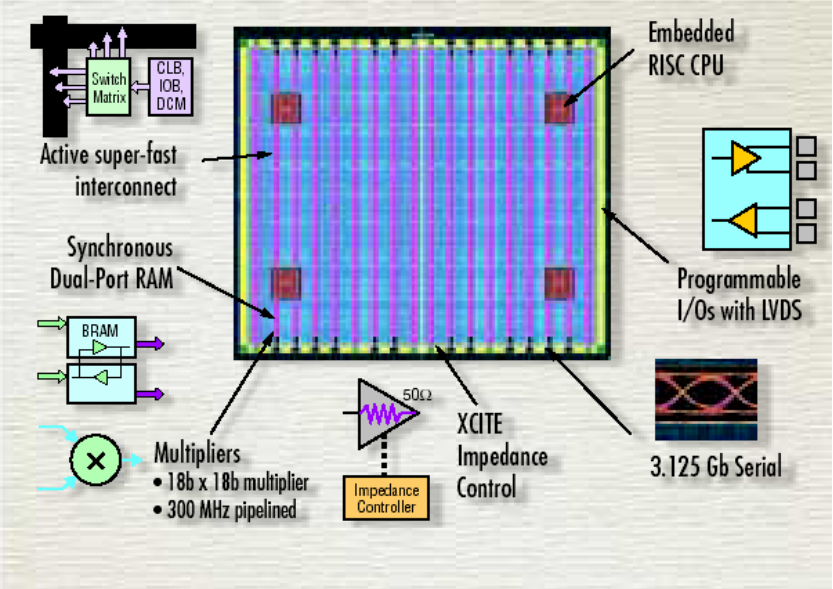
- Avant-Dernier FPGA de Xilinx
- Produit depuis 2003
- Bonnes idées du Spartan2 et Virtex2
- Architecture du Virtex2
- Gravé en 90 nm
- DCM, Multiplieurs câblées
- Forte densité des CLBs
- Fréquence max. de 326Mhz
- Prix de vente faible

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

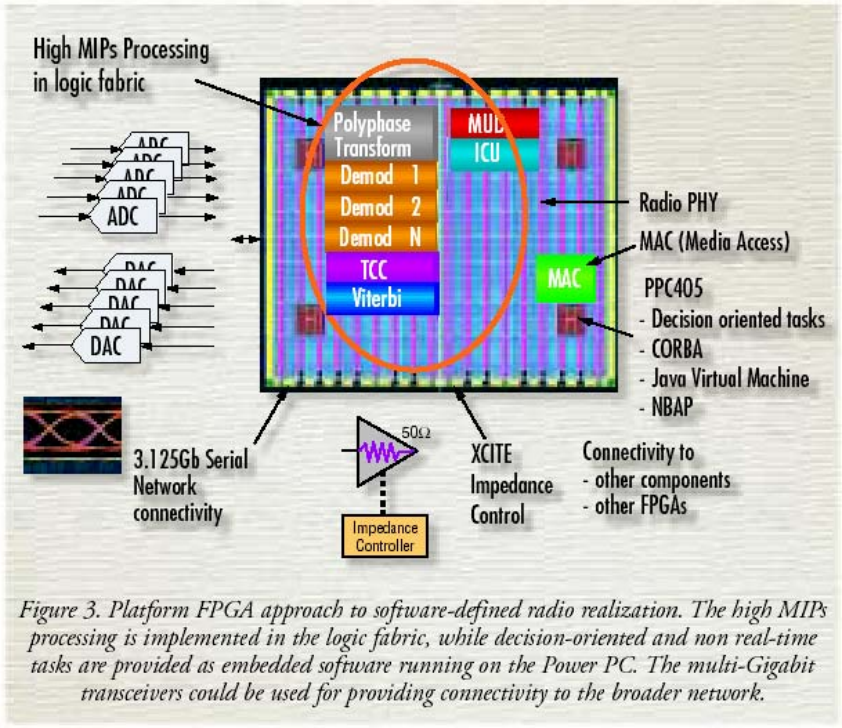
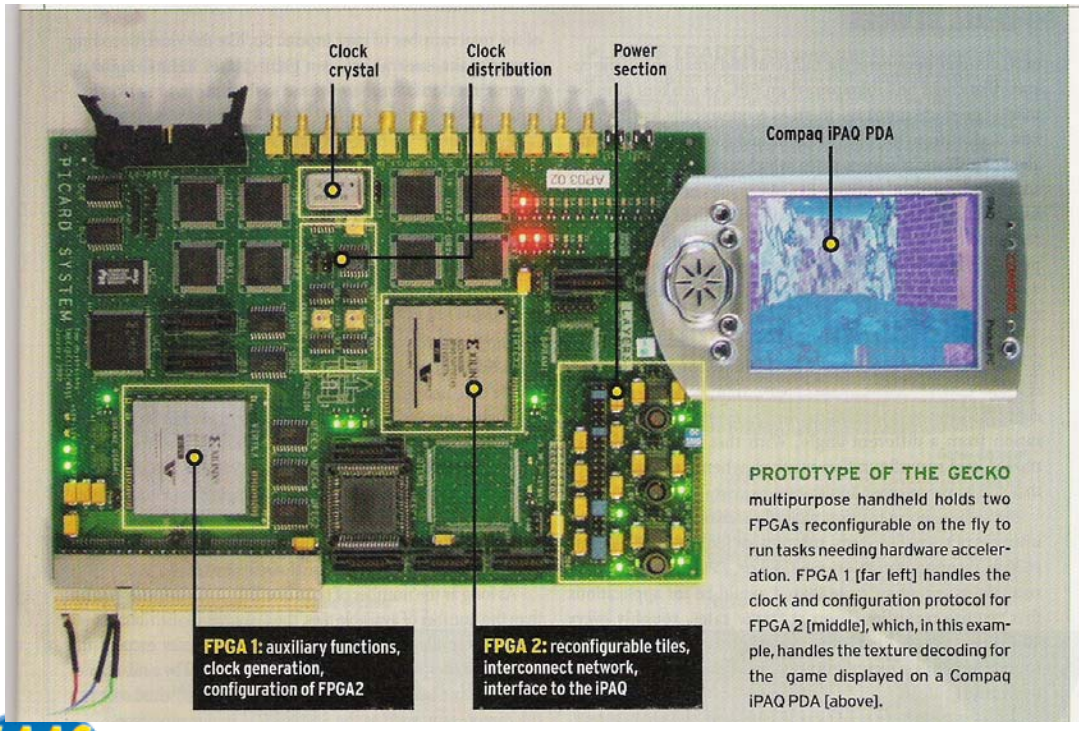
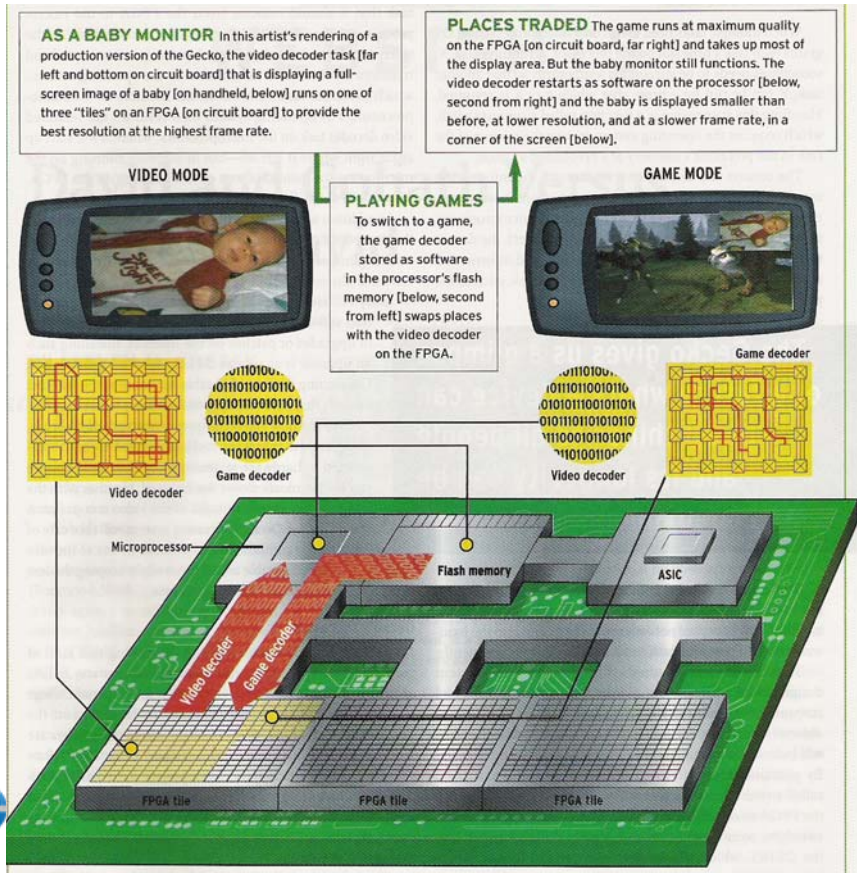


Figure 3. Platform FPGA approach to software-defined radio realization. The high MIPs processing is implemented in the logic fabric, while decision-oriented and non real-time tasks are provided as embedded software running on the Power PC. The multi-Gigabit transceivers could be used for providing connectivity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

DO-IT-All Devices



PROTOTYPE OF THE GECKO multipurpose handheld holds two FPGAs reconfigurable on the fly to run tasks needing hardware acceleration. FPGA 1 [far left] handles the clock and configuration protocol for FPGA 2 [middle], which, in this example, handles the texture decoding for the game displayed on a Compaq iPAQ PDA [above].



Systèmes on Chip - SoC

Daniela Dragomirescu
daniela@laas.fr

Bibliographie

- Pour la conception analogique:
  - P.Gray & R. Meyer - Analysis And Design of Analog Integrated Circuits - Ed. John Wiley & Sons
  - P. Allen & D. Holberg - CMOS Analog Circuit Design - Ed. HRW
- Pour la conception numérique
  - N. Weste & K. Eshraghian - Principles of CMOS VLSI Design - A Systems Perspectives - Ed. Addison-Wesley
- Pour SoC
  - Chang H. et al. - Surviving the SoC Revolution : A Guide To Platform Based Design - Ed. Kluwer Academic Publishers
  - T. Grotker - System Design with SystemC - Ed. Kluwer Academic Publishers
  - C. Chien - Digital Radio Systems on a Chip: A System Approach - Ed. Kluwer Academic Publishers
  - IEEE Design &Test of Computers - sept.-oct. 2001 - Application Specific SoC Multiprocessors

Pourquoi SoC ?

- une porte TTL …CMOS … aujourd'hui 40 millions de transistors sur une puce intégrés dans une technologie de 0.13µm…
- Intel Processeur / Date de production / Fréquence de fonctionnement / Nr. de transistors par puce

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

Méthodologie de conception différente

- VLSI…. ASIC…SoC
- SoC = cohabitation sur silicium des nombreuses fonctions déjà existantes en elles-même: processeurs, traitement de signal, mémoires, bus, convertisseurs, blocs analogiques, communication = Hard + Soft

Pourquoi SoC ?

- Time to market - de plus en plus court
- Systèmes de plus en plus complexe --> temps de conception très longs, incompatible avec le time to market
- Savoir faire nécessaire très grand
- Risc de "bug" - le system ne repondra pas complètement au cahier de charges
  - La vérification est réputée mobiliser plus de la moitié des ressources d'un projet !!!

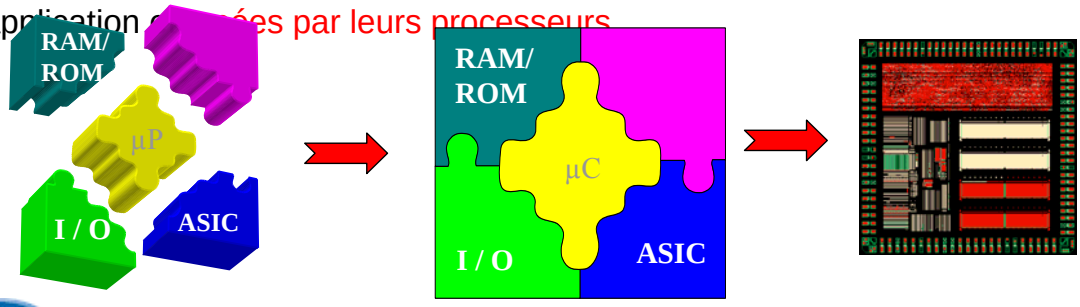
De circuits entièrement dédiés "full custom" on est passée à "standard cells" - blocs virtuels avec des fonctionnalités relativement simples, conçus, testés, fabriqués au par avant

Blocs virtuels réalisant des fonctions complexes - les IPs (Intelectual Property)

Domaines d'applications majeurs des SoC

- Applications qui traitent plusieurs flots des données en même temps
  - Application multiprocesseurs
- Applications multimedia
  - Digital TV - video processing
  - Applications Web sur un PDA ou téléphone mobile GSM
- Routers
- Digital radio on chip (= software radio = radio logiciel)
  - Intégration de la chaîne de transmission/réception d'un téléphone mobile GSM sur une seule puce :
    - Chaîne digitale de traitement des données
    - Conversion analogique -numérique et/ou conversion numérique-analogique
    - Chaîne analogique RF d'emission/réception
    - Amplificateur de puissance ...

- Conception des IPs réutilisables - Design and Reuse - faute de quoi les temps et les coûts de développement deviendront insupportable
- Méthodes d'interconnexions des IP - Création d'un standard VSIA (Virtual Socket Interface Alliance)
- Co-design HW / SW (co-simulation, co-vérification)
  - Software : logiciel embarqué
- Preuve formelle
- Plates-Formes de développements dédiées à des larges secteurs d'application [illegible]ées par leurs processeurs



- Le parasitage de l'analogique par le numérique via le substrat
- Le timing des signaux rendu plus complexe puisque dépendant de la longueur des interconnexions --> pre-routage virtuel dès la synthèse
- La puissance dissipé P_d = nombre des portes actives C_L V_{DD}^2 f
  - Pentium4 - P_d de l'ordre de 30W ! pour un V_{DD}=1.8V et une techno de 0.18µm (C_L de l'ordre de 1fF)
- La quantité énorme des données à gérer

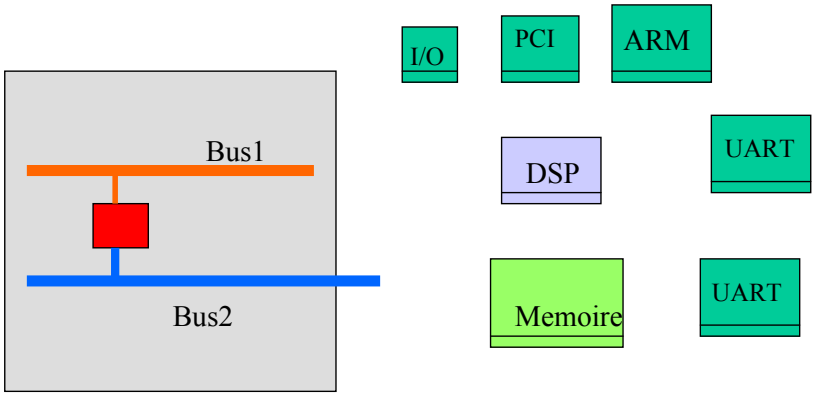
- Approche synthèse à partir d'une spécification (*Specify, Explore and Refine*) – SystemC ou SystemVerilog
- Approche plate-forme (*Configure and Execute*)
- Approche assemblage (*Mix and Match*)
- Approche plate-forme avec IP

- Une façon de permettre aux concepteurs d'améliorer leur productivité est d'augmenter le niveau d'abstraction de la description du système.
- Dans cette approche, le comportement des différents modules et leurs communications sont d'abord décrits avec un langage de haut niveau.
- Puis, le système est divisé, raffiné et synthétisé jusqu'à une implantation physique. L'objectif est d'automatiser ces étapes.
- SoC : SystemC

- Lorsque le concepteur part d'une architecture de base avec un bus et des blocs prédéfinis et modifie ses paramètres pour l'adapter à une application, il utilise une approche plate-forme.

Altera
Excalibur
ARM

Xilinx
AlianceCore



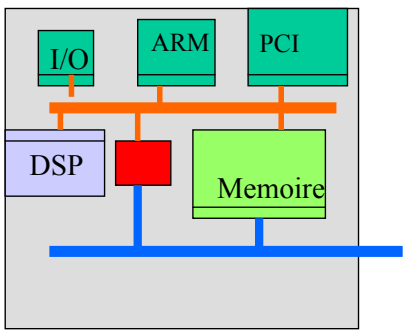
Platform Backbone

Composants avec une interface standard

Platform Based Design

Approche plate-forme (Configure and Execute)

- Lorsque le concepteur part d'une architecture de base avec un bus et des blocs prédéfinis et modifie ses paramètres pour l'adapter à une application, il utilise une approche plate-forme.



Platform Based Design

- La dernière approche consiste à mélanger et apparier différents IP pour construire une application (“Mix and match”).
- La flexibilité est accrue mais le faible degré d’automatisation peut rendre cette approche moins attrayante.
- SONICS

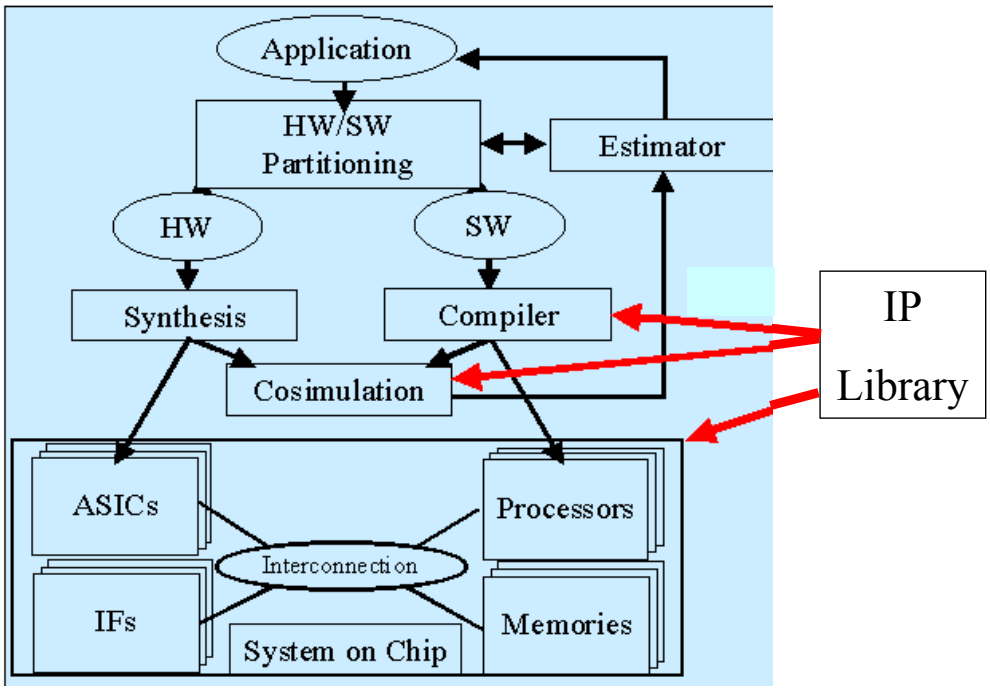
- Les termes “IP-based design” et “Platform-based design” sont souvent utilisés pour désigner un même type de méthodologie. Un concepteur qui combine et apparie différents IP pour construire un système utilise une approche d’assemblage d’IP.
- Cependant, pour le faire, une architecture de base est nécessaire.
- De plus, pour plus de flexibilité, les IP peuvent souvent être configurés. Les deux approches sont donc complémentaires et très souvent utilisées ensemble.
- SOPC : Altera Excalibur NIOS, Xilinx AlinaceCore, CoreGenerator
- SoC : Sonics

Platform Based Design

SoC

- Co-design HW / SW
  - Cadence VCC
  - SystemC ...
- Conception des IPs réutilisables - Design and Reuse

SoC Codesign Flot



IP

- Définition : Un IP est un bloc fonctionnel complexe, qui a été lui même un circuit intégré il y a quelques années et qui a un intérêt pressent ou future pour une organisation[1].

  IP = Core = Virtual Components (VC)

- Les IPs doivent offrir toutes garanties de qualité et de conformité pour qu'ils puissent travailler entre eux au travers de bus, protocoles et logiques de contrôle.

- L'histoire de la conception des SoC à base des IPs a débuté vers le milieu des années 90.
  - Soft IP - sous forme RTL synthétisable (VHDL, Verilog)
  - Hard IP - implémenté physique dans une technologie donnée

- Un IP peut être paramétrable, il n'est pas une boîte noire.

1 - Frank S. Eory, "A Core- Based System-to-Silicon Design Methodology", IEEE Design& Test of Computers, 6/ 1997.

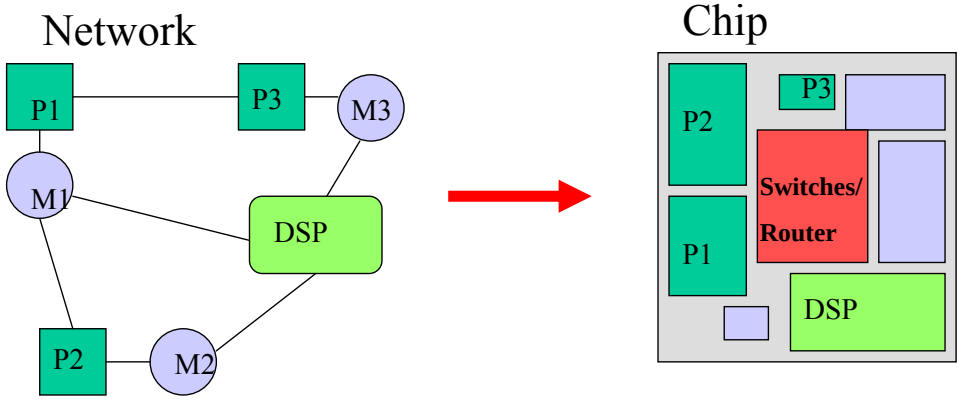
Bibilotheques d'IPs

- Un IP existe sous différentes variantes et configurations, mais toutes vont être décrites par un profil de référence (XML ...)
- Un package IP contient (pour pouvoir suivre toutes les étapes du flot de conception):
  - un model system - description algorithmique ou comportemental
  - un model synthétisable RTL
  - Une netlist post-synthèse
  - Une netlist post-placement-routage
  - Un model physique
- Entreprises
  - Re-use interne
  - Echange externe d'IP
- Création de grandes bases de données (accessibles à travers le Web)- échanges des IPs entre les entreprises
  - http: // www.design-reuse.com
  - http://www.opencore.org

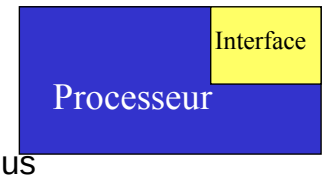
- Exemples des IPs sur le marché (www.sidsa.com - Advanced ARM Emulation):
  - Digital TV SoC for DVB conditional access
  - Sony Digital Camera on Chip (ARM processeur)
  - Bluetooth IP
  - USB IP, PCI IP
  - Ethernet MAC IP
  - Java Accelerator IP
    - RTL Synthétisable
    - Support software pour Java Virtual Machine
- Dolphin - core µC 8051
  - Code VHDL et Verilog synthétisable
  - Code assembleur et simulateur C

- Méthodes d'interconnexions des IP - Création d'un standard **VSIA** (Virtual Socket Interface Alliance)
- **Network on Chip**

- Network on Chip = organiser les ressources disponibles (bloc IP:, µP, DSP, I/O, memoires) de manière qu'elles puissent communiquer efficace
- On s'inspire des techniques utilisés dans les réseaux d'ordinateurs et des techniques de bus pour l'interconnexions de plusieurs cartes (bus PCI)



- Conditions à remplir :
  - Placement de composants et routage - le plus simple possible
  - Possibilité d'utiliser des bloc IP de provenance différente --> fournir une interface standard --> bus



  - Assurer des propriétés déterministe pour les signaux véhicules et l'intégrité du signal --> bus
  - Assurer l'arbitrage du bus

VSIA

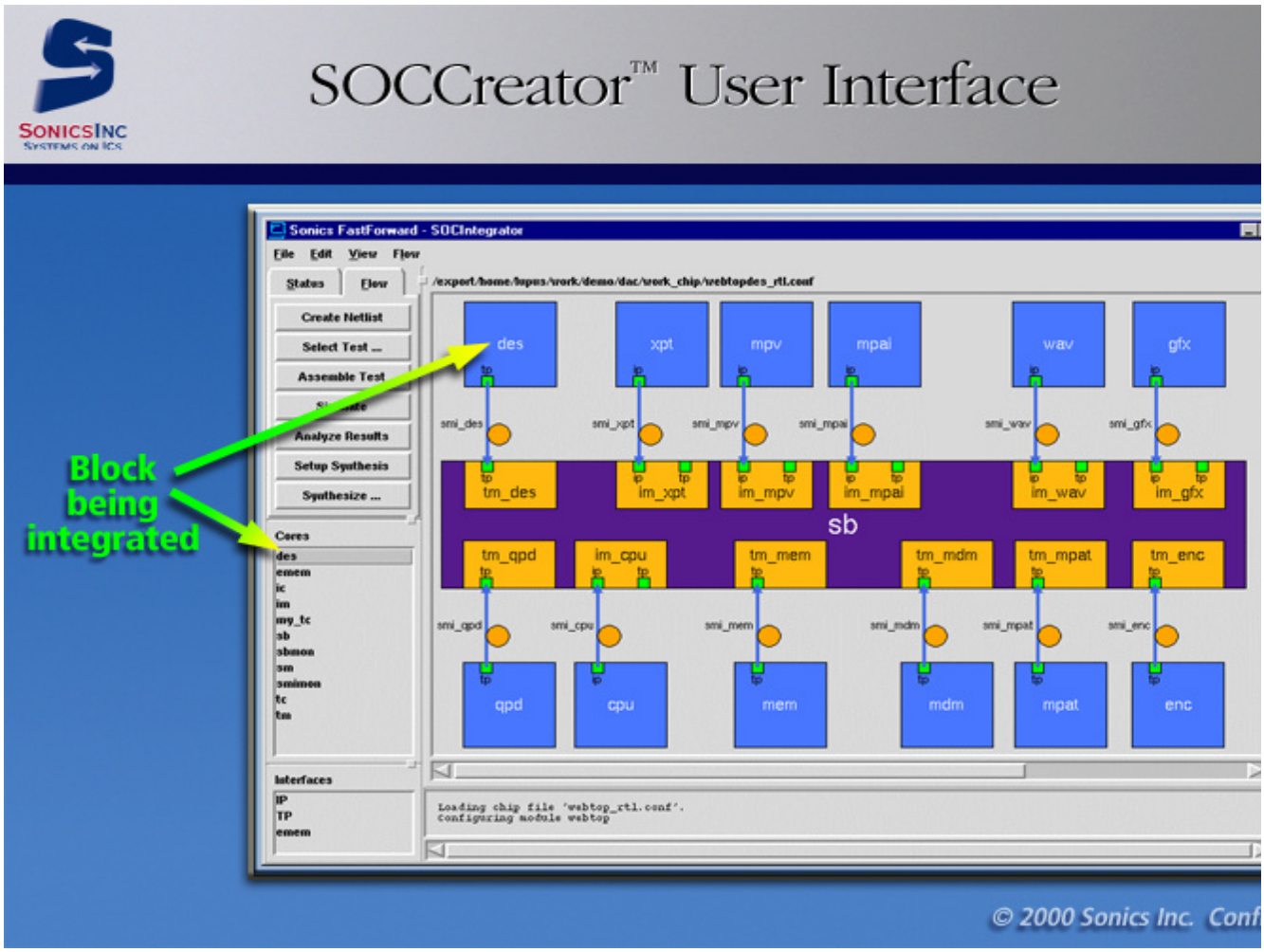
- Plusieurs architectures possibles:
  - CPU centric : bus based architecture
    - Inconvénient : blocage du bus --> blocage du SoC
  - Memory centric : star based architecture
    - Inconvénient : reconception du control mémoire
  - Approche LAN --> solution choisi par SONICS

- Virtual Socket Interface Alliance - http://www.vsia.com - formé en 1996 dans le but d'établir une vision unifié de l'industrie SoC et d'établir les standards techniques nécessaires afin de faire communiqués les IPs entre elles !
- L'effort du VSIA consiste en standardisation :
  - La spécification du bus interne "on-chip bus" = OCB
  - La description du bus interne
  - L'implémentation du bus interne
- VSIA écrit des standards d'interface "open", qui permettent aux IP de se placer facilement dans les "Virtual Sockets" au niveau :
  - Fonctionnel (ex: protocoles)
  - Physique (clock, test, alimentation)

VSIA

- Standardisation de l'architecture du "on-chip bus" = OCB
  - Minimise le re-design des circuits d'interface d'IPs
  - Ajoute des nouvelles fonctions
- Le bus est caractérisé par :
  - Largeur de bande
  - Pipeline (oui/non)
- **Inconvénient : Pas d'outils de développement et/ou d'implémentation**

- Le seul logiciel disponible pour placer et router un SoC
- Utilise des interfaces très ressemblantes à celles de VSIA
- Architecture du réseau sur le chip de type LAN
- Open Core Protocole
  - Synchrone
  - Unidirectionnel, point to point
  - Comportement et timming parametrisable
  - Signaux des données, control et test
  - Contrôle plusieurs flots de données
  - Set de commande extensible
- Mécanisme de signalisation intégré:
  - Bus (fils) dédié aux interruptions et signaux de control du flot de données
- Silicon Backplane
  - Concurrence (jusqu'au 256 threads simultanés) - arbitrage : round Robin , TDM
  - Pipeline - défini par l'utilisateur



Approche plate-forme (avec IP Mélange)

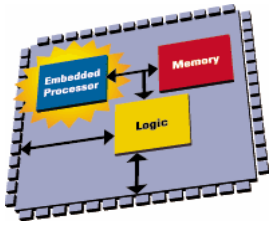
Platform Based Design

D.D

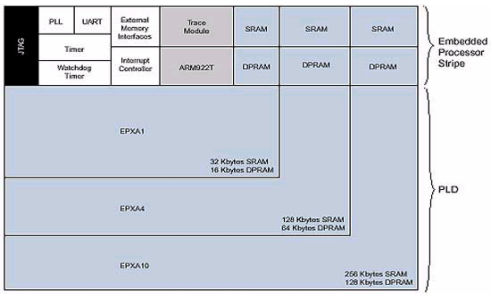
Systems on Programmable Chip -SOPC

- *ALTERA Excalibur - Embedded Processor Sol.*

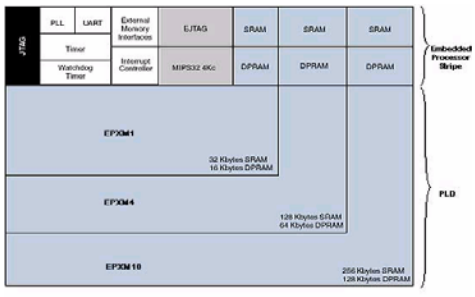
Development system



NIOS



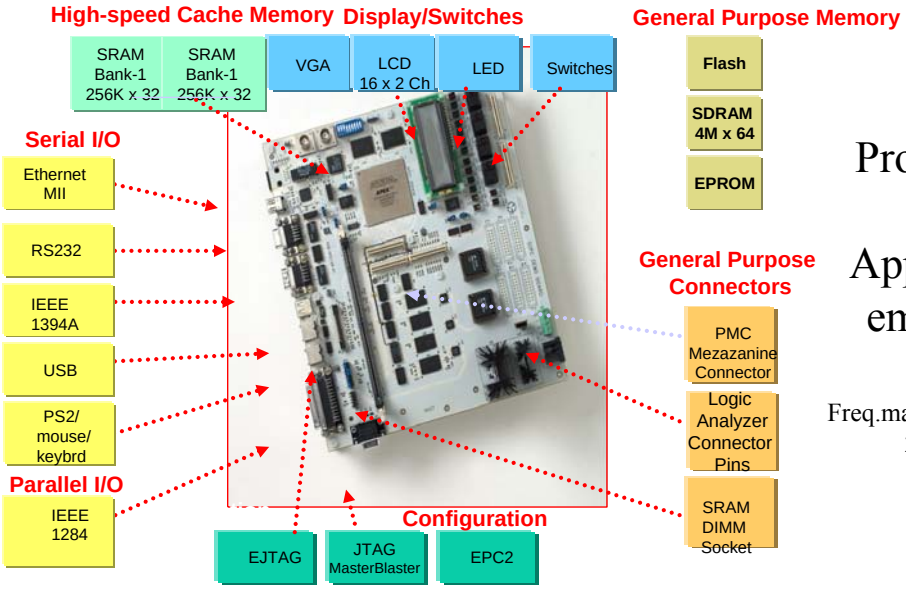
ARM



MIPS

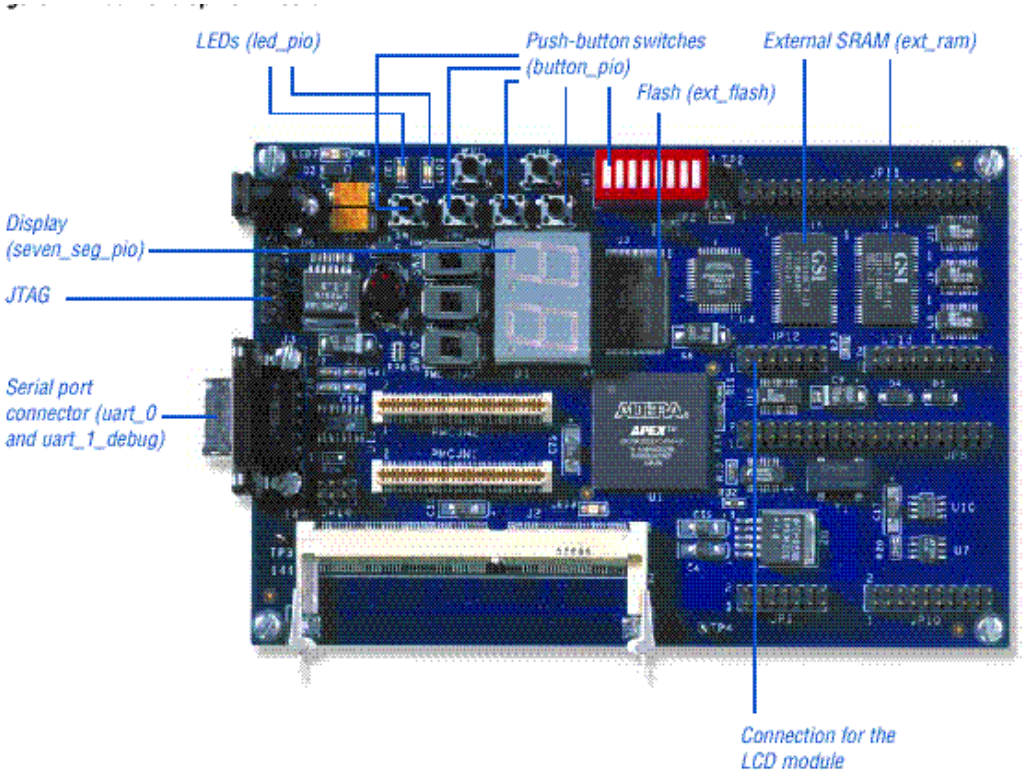
D.D

❖ *SOC Prototyping Board*



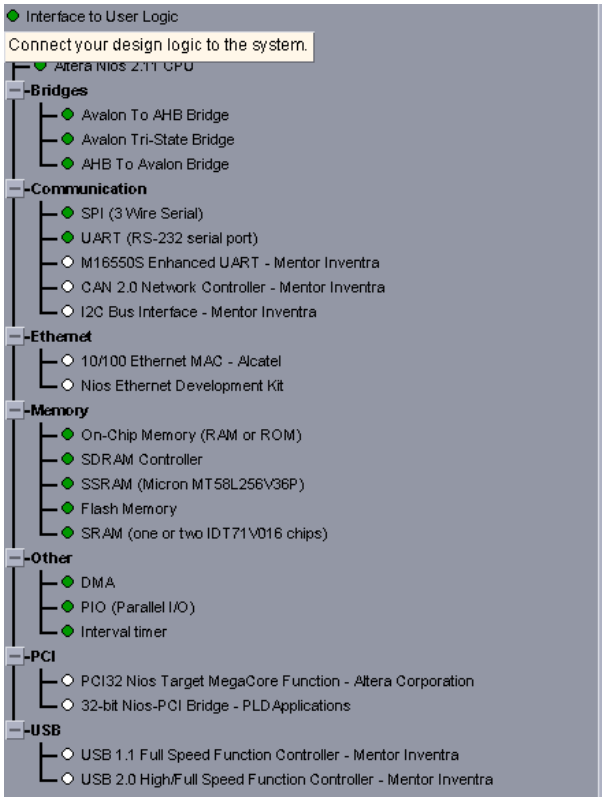
Prototypage
+
Applications embarqués

Freq.max. du processeur
200 MHz



- 1. Création du SoPC en utilisant le logiciel **QUARTUS II**
  - MegaWizard Plug-In Manager --> SOPC Builder
  - Configuration du cœur NIOS et de sa communication avec les périphériques :
    - Mémoire
    - communication avec le PC à travers le RS232
    - les afficheurs 7 segments
  - Autres IP : communication Bluetooth
- 2. Simulation VHDL / Verilog- logiciel ModelSim
- 3. Synthèse logique - logiciel Leonardo Spectrum
- 4. Compilation - synthèse logique tenant compte du composant APEX de la plate-forme Excalibur, placement et routage du SoPC sur le composant APEX. Logiciel **QUARTUS II.**
- 5. Implantation du SoPC dans le composant APEX via le port JTAG
- 6. Programmation du processeur Nios du SoC via la liaison série RS232

SOPC Builder



Quartus - SoPC Builder

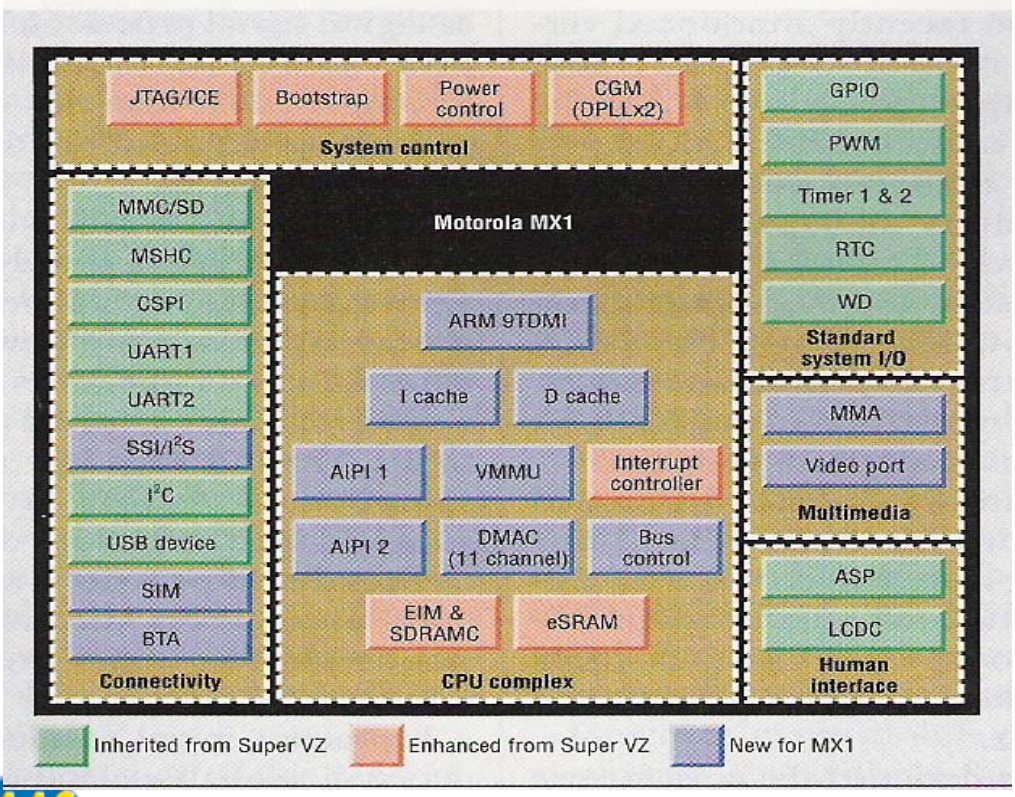
- dispose des IPs simples
- multiplieur
- compteur
- comparateur

- Chaîne de transmission numérique à base de turbo codes
  - Projet étudiants 120h * 6 étudiants
- Transmission d'une image VGA à travers une liaison série RS232
  - Projet étudiants 60h * 3 étudiants

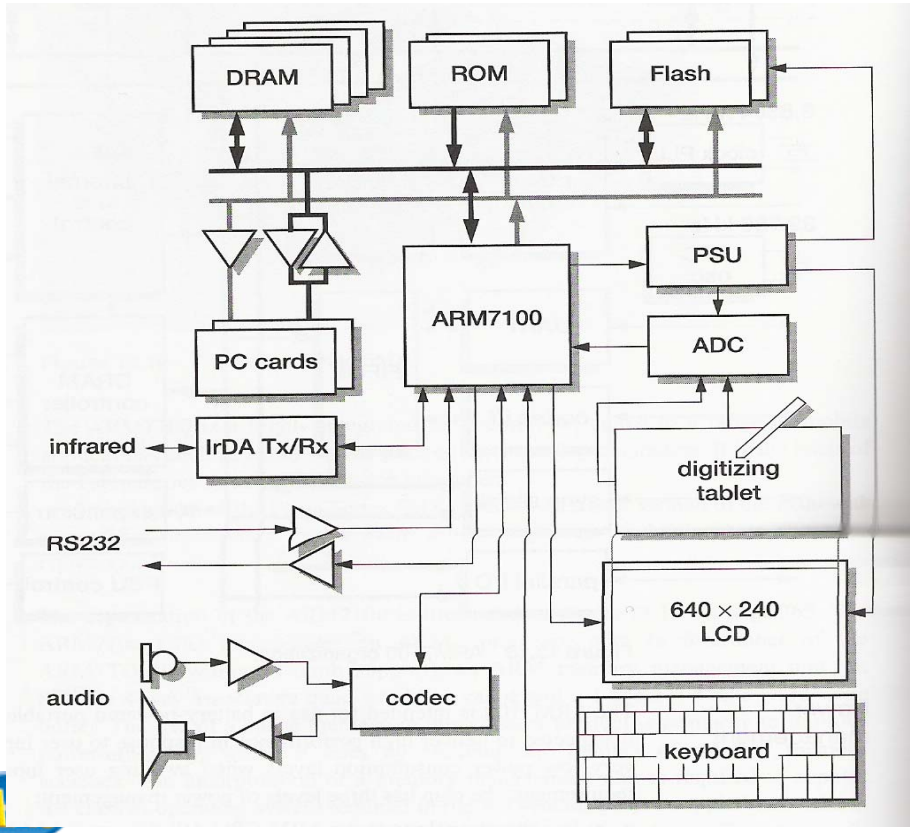
- Pre-distorsion digitale pour la linéarisation des amplificateurs de puissance dans les systèmes de télécommunications mobiles. Application aux systèmes sans fil en modulation QAM

- Modulateur numérique QAM
  - peut-être faisable en TP …
  - **Disponibilités des IP … voir OpenCore**

Applications : PDA



Applications : PDA



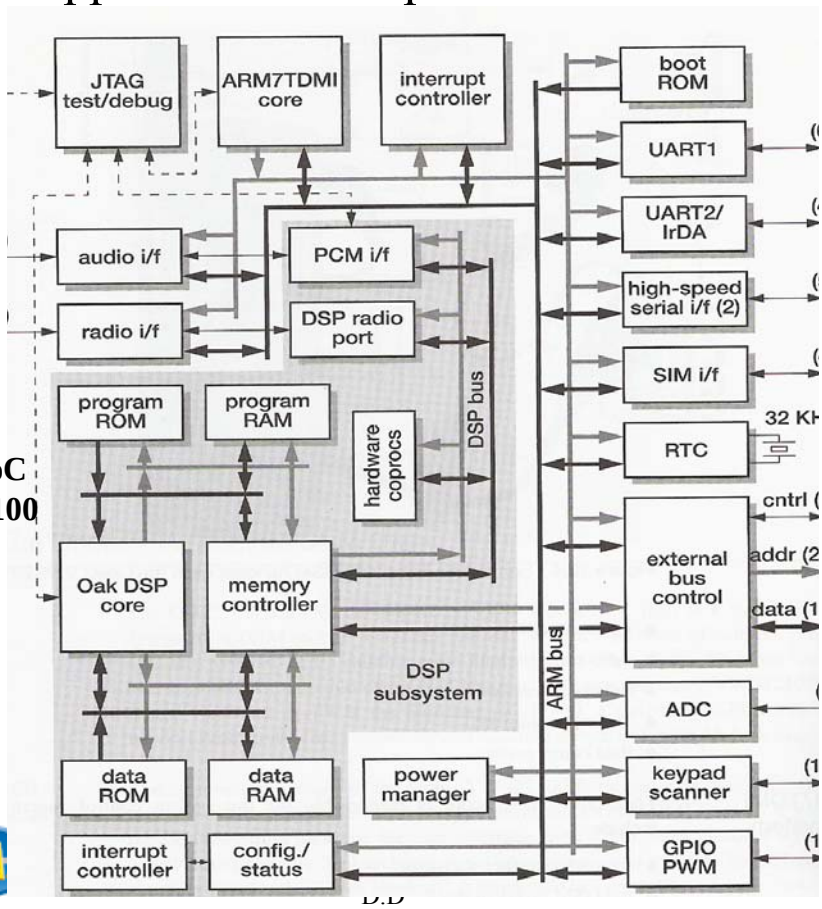
Applications : PDA

- En mode opérationnel complet ARM7 fourni 14MIPS et il consomme 24mA à 3V
- En standby ARM7 consomme 33µW
- En idle mode (CPU arrêté, mais d'autres systèmes fonctionnant) ARM7 consomme 33mW

- Grande flexibilité pour la connexion des nombreuses interfaces conduisant à une architecture très sophistiqué tout en gardant une complexité minimale au niveau de la puce hardware.

Applications : téléphonie mobile GSM

Puce GSM= SoC
OneC VWS22100

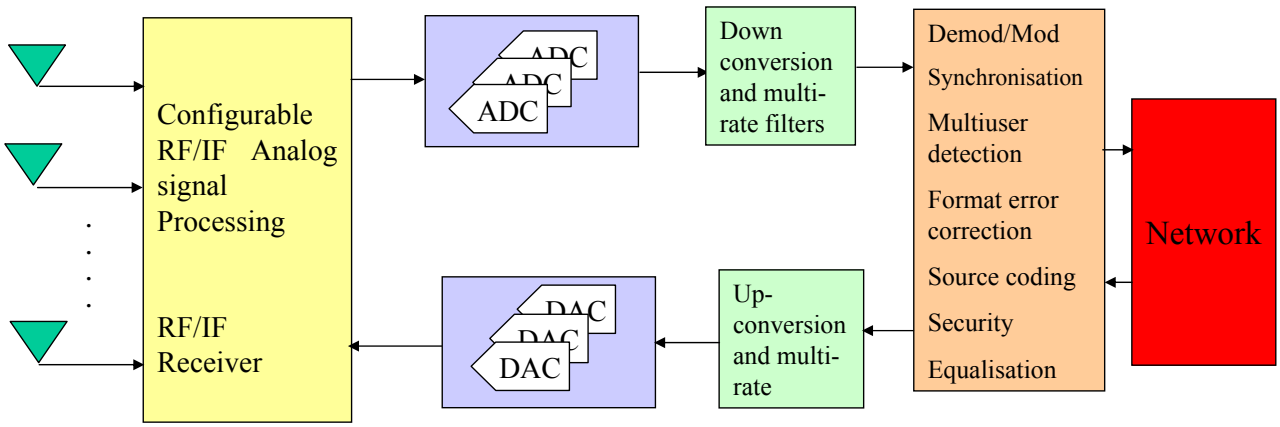


D.D

Applications : téléphonie mobile GSM

- L'architecture du système de traitement de signal pour un tèlephone mobile GSM:
- L'ARM7 –gère l'interface utilisateur et certains couches du protocole GSM
- Le DSP gère le traitement de signal en bande de base aidé par quelques blocks hardware spécifiques
- Le DSP réalise le traitement de signal en temps réel des fonctions suivantes :
  - Le codage de la voix
  - Équalisation
  - Le codage de canal
  - L'annulation de l'écho
  - La suppression du bruit
  - La reconnaissance de la voix
  - La compression des données
- Donc, le flot des données contenant la voix codé et les symboles émis ou reçu sur/ de la interface radio passe directement entre le système DSP et les périphériques.

- L'ARM7 réalise les fonctionnalités suivantes:
  - Le control du logiciel d'interface utilisateur
  - Le protocole GSM
  - La gestion de la puissance consommée
  - La gestion des périphériques
  - Tourner certains applications des données
- Les périphériques sont connectés au microprocesseur ARM et au DSP
- L'ARM
  - établi le gain des amplificateurs
  - Contrôle la puissance du niveau de transmission radio
  - Contrôle les synthétiseurs de fréquence qui vont produire l'appel de la sonnerie dans le cas d'un appel entrant



Radio Logiciel

- Aujourd'hui la demande de systèmes re-configurables rapidement est très grande.
- Exemple :
  - couche radio développé en Europe pour GSM
  - couche radio développé en Europe pour UMTS
  - couche radio développé aux USA pour CDMA2000
  - Ce système peut fonctionner comme un système radio multi-mode dans un environnement qui rempli en même temps des communications CDMA de bande large ou étroite.
- Transition de la 2G à la 3G tout en passant par le 2.5G → stations de base qui support GSM, GPRS, EDGE, UMTS
- Station de base reconfigurable :
  - Upgrade facile
  - Réparation facile de la station de base, même une fois le réseau déployé, en transmettant le nouveau profile émis par le contrôleur réseau par Internet ou par une liaison sans fil.
- A l'avenir 4G:
  - Direct sequence spread spectrum DSSS
  - Orthogonal frequency division multiplexing OFDM
  - Pour un opérateur réseau, la reconfigurabilité de la station de base peut être utilisée pour allouer de manière dynamique les ressources radio

OFDM

- Technologie breveté en 1970
- Principe:
  - Division des données initiales dans des flots parallèles des bits qui ont un débit beaucoup plus faible que les données initiales. Ces flots des bits vont moduler plusieurs porteuses.
- Reconnu récemment comme méthode importante pour des communications sans fil bi-directionnelles avec un très grand débit
- La bande de fréquence est divisée dans un grand nombre de fréquences sous-porteuses. Les fréquences sous-porteuses sont orthogonales →séparation facile par le récepteur sans problèmes d'interférences → diminution du problème de réception muti-trajet
- OFDM → liaison de qualité supérieure et robuste
- Applications:
  - Digital Audio et Video Broadcasting
  - ADSL
  - Wireless ATM Network
  - IEEE 802.11a – hyper LAN et WLAN
    - 802.11a à 5GHz, débit : 54Mbps
    - Largeur de bande d'un canal : 20MHz – 52 sous-porteuses orthogonales

Les FPGA « customized » les chemins des données

- Quels sont les compromis des conception et quelle optimisation doit être faite dans les designs temps réel ?

- Les DSPs ont des architectures ISA (instruction set architecture)
  - Très similaires avec les microprocesseurs traditionnels
- ISA : on place (« map ») l'algorithme dans l'architecture !

- FPGA : on conçoit l'architecture pour l'algorithme !!!
  - Les FPGA profitent aussi de la loi de Moore → augment leur fréquence de fonctionnement
  - Le mécanisme pour être performant est l'exploitation du **grand parallélisme** qui existe dans les algorithmes de traitement de signal.
    - Parallélisme algorithmique – exemple : FFT sur le récepteur
    - Parallélisme fonctionnel – plusieurs décodeurs Viterbi qui fonctionnent en parallèle dans les systèmes multi-user (2Mbps pour chaque user en téléphonie 3G)
  - Exemple : Xilinx Virtex Pro – 556 multiplieurs câblés

Implémentation des fonctions de traitement de signal

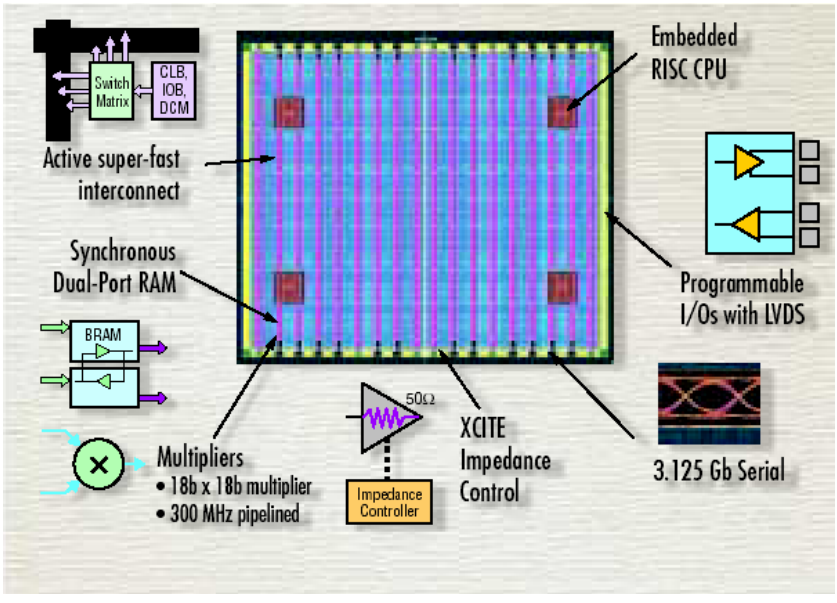
- Actuellement dans les stations de base: DSP et ASIC

- ASIC :
  - Calculs arithmétiques du « radio front-end » (digital down conversion, filtres multi-canal pour les systèmes multi porteuses comme W-CDMA). Ces fonctions dépassent largement la capacité de calcul de tout DSP actuel.
  - Ces ASICs offres une certaine programabilité

- DSP:
  - Fonctions dans le traitement de signal en bande de base, comme le codage –décodage de source

- Fonctions de traitement de signal réalisés par les FPGA
  - Digital down conversion
  - Digital up conversion
  - FFT – temps d'exécution: 1μs à dizaines de μs
  - Multi-rate decimators et interpolators
  - Multi user détection
  - Acquisition des données
  - Implémentation des algorithmes pour DSSS
  - OFDM modulateurs et démodulateurs
  - QAM modulateurs et démodulateurs
    - Égalisation de canal adaptative
    - Boucle à verrouillage de fréquence
    - Boucle de verouillage de la porteuse
  - Implémentation des codeurs –décodeurs (fréquence de fonctionnement >200Mhz)
    - Codeur –décodeur Viterbi –bloc IP fourni par Xilinx
    - Codeur-décodeur Reed-Solomon - bloc IP fourni par Xilinx

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

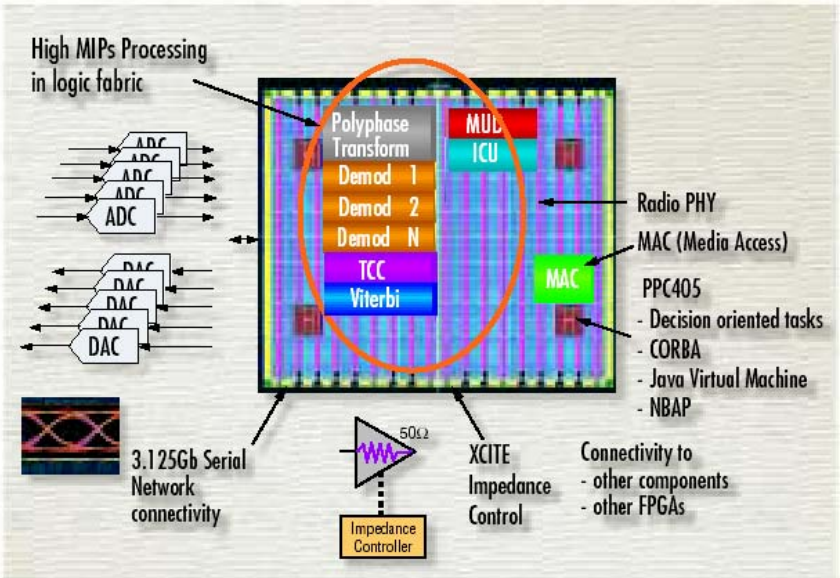


Figure 3. Platform FPGA approach to software-defined radio realization. The high MIPs processing is implemented in the logic fabric, while decision-oriented and non real-time tasks are provided as embedded software running on the Power PC. The multi-Gigabit transceivers could be used for providing connectivity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

- SoC - domaine qui demande des compétences en électronique et informatique --> Nouvelle méthodologie de conception de systèmes
- De plus en plus de plate-formes orientée design (PBD) se développent
- Design d'IP en digital en utilisant HDL --> re-use
- Bibliothèques d'IPs à travers le Web
- …. et l'analogique ? … VHDL-AMS et Verilog - A …
- Communications entre IPs - standardisé -> à améliorer encore
  - **Network on Chip**
- Problèmes à résoudre au cas par cas :
  - Timing
  - Puissance dissipée
- Perspectives :
  - Tools for HW / SW partitioning
  - Création et évolution d'AL (Architecture Description Language)
- Pour de systèmes très performants on va toujours avoir besoin d'Aplication Specific SoC

D.D

ANNEXES

Le microprocesseur ARM

ARM

THE ARCHITECTURE FOR THE DIGITAL WORLD™

Advanced Risc Machine

http://www.arm.com

Le processeur pour applications mobiles

Evolution de processeurs mobiles

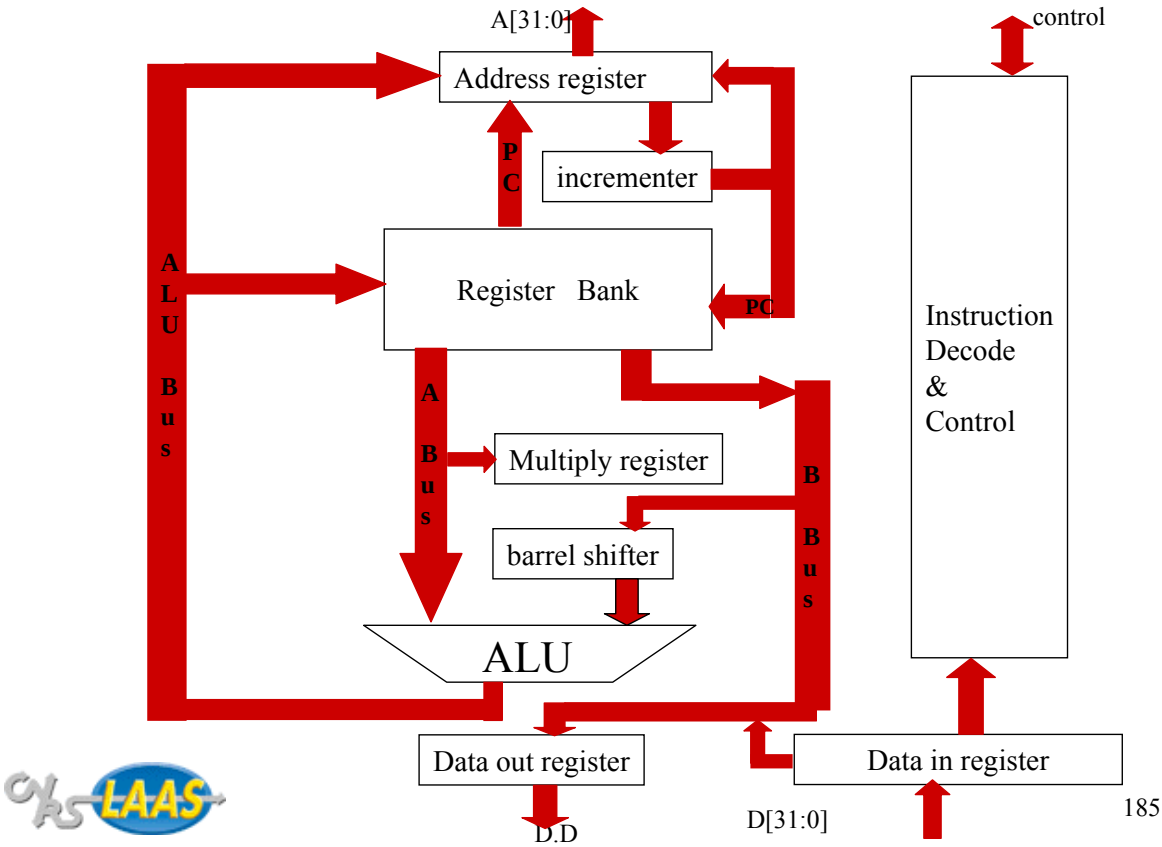
- Dans le passé, les processeurs mobiles fonctionnait à 30 Mhz pour les Palm PDA, à 80MHz pour les téléphones mobiles
- Maintenant les équipements mobiles doivent fournir, par exemple, des accès réseaux sans fil (Wi-Fi, Bluetooth), des services de localisation (GPS)
- Il faut des processeurs plus rapide et en même temps ayant des:
  - dimensions réduits (pour pouvoir être integré dans un équipement qui tient dans la main)
  - **une faible consommation**
  - faible dissipation de chaleur
  - Faible coût
  - mémoire intégré

Approches de conception pour les microprocesseurs

- Complex Instruction Set Computers --> CISC
  - code dense, compilateur simple
  - set d 'instruction puissant, format variable, multi-word (octet)
  - exécution multi-cycle, fréquence d 'horloge lente (?)
- Reduce Instruction Set Computers
  - fréquence d 'horloge rapide, coût de développement faible (?)
  - facilité d'implémentation dans une nouvelle technologie
  - instruction simples, format fixe, compilateur complex

L'organisation du processeur ARM

- Le premier processeur ARM a été développé dans une technologie de 3 microns en 1983-1985
- Cette présentation se réfère principalement à l'architecture de ARM 6/7, développé entre 1990 - 1995.
- ARM s'est focalisé sur l'augmentation des performances en créant des cœurs de micro-processeurs et un environnement de développement qui permet aux acheteurs de changer facilement de technologie
- ARM optimise les instructions, la répartition de l'horloge

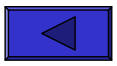


L'architecture du microprocesseur ARM

- 2 blocs principales :
  - le chemin des données (datapath)
  - le décodeur
- un bank de registres : r0 à r15
  - 2 port de lecture A-bus et B- bus
  - un port d 'écriture - le bus ALU
  - des port additionnels d 'écriture-lecture pour le Program counter = r15
- barrel shifter - shift/rotate pour la 2ème opérande
- ALU
- registre d 'adresses - il garde l 'adresse du PC ou l 'adresse de l 'opérande

L'architecture du microprocesseur ARM

- Registres de données - garde les données à lire/écrire de la / dans la mémoire
- Le décodeur d 'instructions décode le code machine des instructions pour produire les signaux de control pour le chemin des données
- Dans les instructions exécutés dans un seul cycle d 'horloge, les valeurs des données sont lues sur le bus A et B et le résultat donnée par l 'ALU est écrit dans le bank de registres
- La valeur du PC dans le registre d'adresses est incrémenté et copié dans r15 et le registre d'adresses --> cette opération permet le pre-fetch



L'architecture du microprocesseur ARM

- Pipelining
  - ARM a 3 étages de pipeline :
  - FETCH - fetch le code d'instruction de la mémoire dans le pipeline d 'instructions
  - Décodage - décodage de l 'instruction pour obtenir les signaux de contrôle pour le chemin des données
  - Exécute: lecture des registres, déplacement, résultats de l 'ALU générés et inscrits dans le bank de registres
  - les résultats de chaque étage de pipeline sont stokes dans de registres --> la période d 'horloge est plus courte que sans pipeline
  - 3 instructions différents peuvent occupés chaque étage du pipeline
  - latence = 3 cycles (on a besoin de 3 cycles pour compléter une instruction)
  - une fois le pipeline rempli, le processeur effectué une instruction à chaque cycle d 'horloge

L'architecture du microprocesseur ARM

- Principales caractéristiques:
  - architecture « load-store »
    - instructions de « load-store » de plusieurs registres dans un seul cycle d 'horloge
  - instruction de longueur fixe --> 32 bit
  - dans le format de l 'instruction, il y a 3 adresses : 2 adresses pour les opérandes et une adresse pour le résultat
  - exécution conditionnelle de TOUTES les INSTRUCTIONS
  - instruction de déplacement (shift) de n-bits prévu dans l 'ALU et exécutable dans un seul cycle d 'horloge
  - interfaçable avec les instructions du co-processeur

ARM - registres et organisation de la mémoire

- 15 registres
- R0 à R14 - registres d 'use général (32 bits)
- R15 - program counter (PC)
- registres pour le fonctionnement en mode system - utilisés pour les interruptions, le « call system »
- Current Program Status Register (CPSR) - registre de drapeaux (flags)

- Maximum 2^{32} octets de mémoire
- un mot = 32bits, un demi-mot=16bits

ARM - l ’organisation d ’I/O

- Il manipule TOUS les périphériques d ’entrées/ sorties (comme l ’imprimante, le hard-disk, le réseau, uart) comme des composants « cartographié » dans la mémoire
- 2 types d ’interruptions : normal et rapide
- Accès DMA câblé hardware pour assurer un transfère des données rapide (high bandwidth)

Outils de développement ARM

- Compilateur de C pour ARM - ANSI C, standard, rapide, intégré
- ARM assembleur
- Linker
  - prend un ou plusieurs fichier object (donnée par le compilateur de C ou par l ’assembleur) et il les combine dans un programme exécutable
  - il ressoude les références symboliques
- ARM debugger - contrôle complet sur l exécution et la visualisation des registres
- ARMulator - emulateur

Exécution conditionnelle de toutes les instructions

- On peut rajouter des conditions à TOUTES les instructions
  - ; if ((a==b) &&(c==d) then e:=e+1)
  - CMP r0,r1 ; r0 =a et r1=b
  - CMPEQ r2,r3 ; r2=c et r3=d
  - ADDEQ r 4, r4, #1 ; e:=e+1
- observons que si la première comparaison trouve des opérandes non-égaux, la 2ème et la 3ème instruction ne sont pas exécutes
- le « and » logique et la clause « if » sont implémentés en faisant la 2ème comparaison conditionnelle
- l exécution conditionnelle est efficace si la séquence conditionnelle est composée de 3 instructions ou moins. Sinon, il faut faire des boucles conditionnelles habituelles

Déplacement (shift)

- Dans toutes les instruction, on peut appliquer à la 2ème opérande une opération de déplacement
  - ADD r3, r2, r1, LSL #3 ; r3:=r2+8*r1
  - LSL --> logical shift left avec le nombre spécifiés de bits
- ceci est une seule instruction ARM , qui s 'effectue dans un seul cycle d 'horloge !!!
- Le déplacement est intégré dans l 'ALU

Architecture « LOAD- STORE »

- ARM peut chargé / stocké TOUT sous-set de registres dans une seule instruction, exécuté dans un seul cycle d 'horloge
- l 'instruction de « load/store » de plusieurs registres peut être utilisé pour implémenter une mémoire LIFO = la pile
- pour la pile
  - r13 est utilisé comme pointer du top de la pile
  - la pile augment en décrémentant les adresses mémoire
- ARM n'a pas d 'instruction PUSH

ARM et JAVA

- La « portabilité » du langage JAVA est très importante dans le marché des applications mobiles car ce marché contient de nombreuses platformes et microprocesseurs différentes.
  - Java peut augmenter la productivité des programmateurs et permettre aux vendeurs l'accès aux différentes fournisseurs.
  - La majorité des applications Java tournes avec un « overhead » minimal spécifique au microprocesseur.
- ARM a implémenté la machine virtuelle J2ME (Java 2 Platform Micro Edition's)
- ARM interprète JAVA directement en hardware !!!
  - Réduction du temps d'accès mémoire
  - Économie de puissance consommée et dissipée
  - Les applications tournent beaucoup plus vite !!!

Introduction Générale au Langage SystemC

Roberto Reyna
Daniela Dragomirescu

Références

- Bhasker, J. A SystemC Primer. Star Galaxy Publishing, June 2002.
- Open SystemC Initiative Website: www.systemc.org
- SystemC Language Reference Manual. Inclus dans la version SystemC 2.0.1 à télécharger de http://www.systemc.org.
- SystemC User Guide. Inclus aussi dans la version SystemC 2.0.1
- Cette présentation est basée sur la présentation de Forte Design Systems http://www.forteds.com. Vous y trouverez une présentation plus détaillée en anglais

Historique

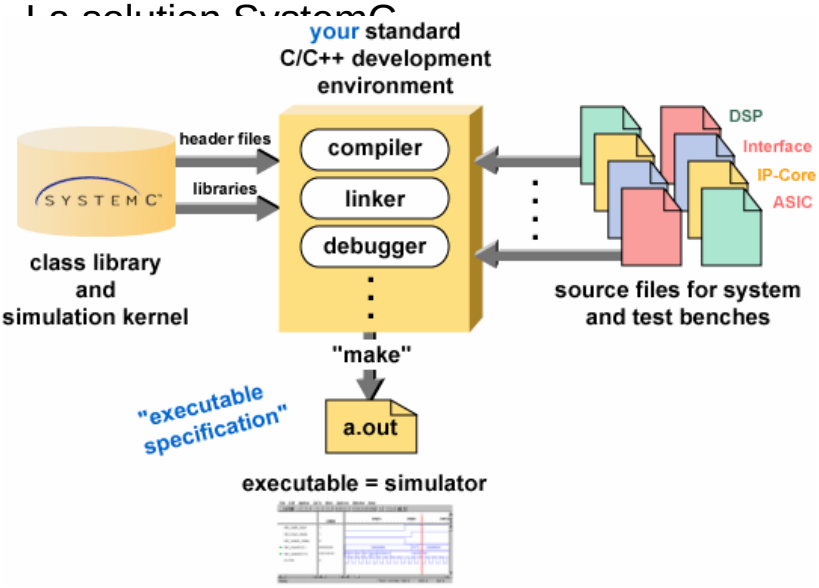
- Le langage SystemC a été proposé comme solution pour la modélisation des composants logiciels et matériels d'un système en utilisant le langage C++
- La version 0.9 a été publié en 1999 par l'initiative OSCI (Open SystemC Initiative)
- SystemC avait été proposé au départ par Synopsys, Frontier Design et CoWare
- Les deux principales releases ont été publiées en janvier 2000 (Version 1) et en juillet 2002 (version 2)
- http://www.systemc.org

Les limitations de C++

- C++ peut modéliser des composants logiciels
- Mais pour le matériel, C++ possède trois limitations qui ont été «corrigés» en SystemC
- La notion du temps
  - En C++ aucune notion d'événements temporaires n'existent
- Le comportement concurrentiel du matériel
  - Le matérielle et de nature concurrentielle, il fonctionnent en parallèle, en C++ ces notions n'existent pas
- Les types du langage ne sont pas adaptés aux spécifications matérielles
  - Les types existant en C++ ne sont pas adaptés pour le matériel, par exemple il n'y pas un type pour déclarer un état de haute impédance

La solution SystemC

- SystemC est une extension de C++
  - La syntaxe reste la même
  - On utilise les modules pour l'encapsulation et la gestion de la hiérarchie (l'équivalent des entités en VHDL)



- Comme c'est du C++, on utilise les environnements de C/C++
- Il faut juste rajouter la librairie et le noyau de simulation
- On peut utiliser les outils classique pour l'affichage de chronogrammes

Les images appartiennent à Forte Design Systems

Les Modèles

- Modèle système architectural
- Modèle performance système
- Modèle au niveau transaction
- Modèle fonctionnel
- Modèle au niveau système
- Modèle de synthèse comportementale
- Modèle fonctionnel au niveau bus
- Modèle au niveau de transfert de registres RTL
- Modèle au niveau de portes

Les Modèles et VHDL

- En VHDL on parle plutôt de description:
  - Le modèle au niveau RTL et le modèle au niveau de portes de base sont les mêmes Description RTL et description de portes logiques (utilisé surtout après synthèse)
  - Le modèle de synthèse comportementale et l'équivalent de la description comportementale en VHDL
  - Le modèle au niveau système peut être assimilé à la description architecturale en VHDL
  - Le modèle fonctionnel au niveau bus peut être vu comme l'équivalent d'une description en flot de données VHDL mais qui n'est pas utilisé pour la synthèse dans le cas SystemC

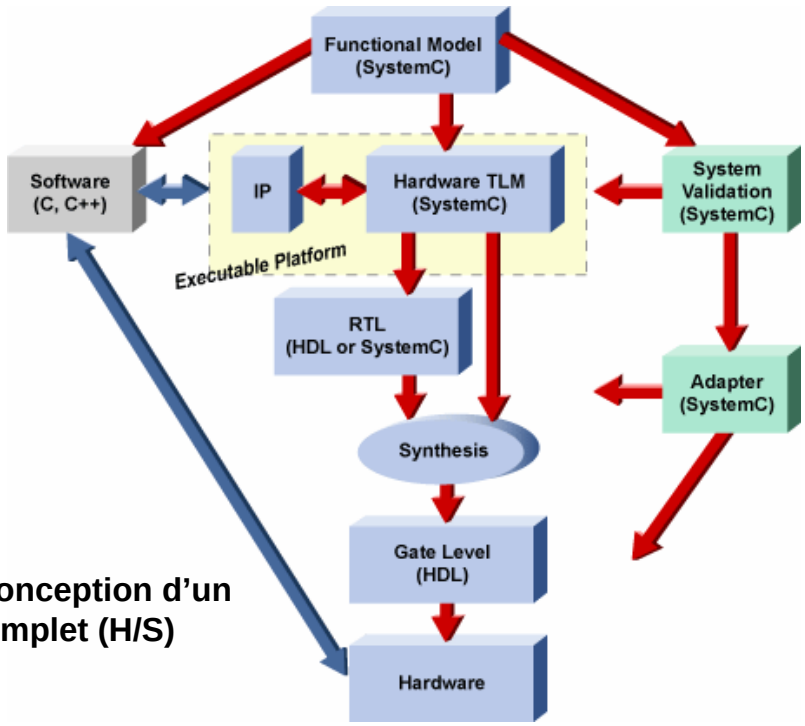
Les modèles de plus haut niveau 1

- SystemC est donc un langage pour modéliser le matériel et le logiciel, on a donc des modèles supplémentaires
- Les deux modèles de plus haut niveau servent principalement à la modélisation des composants logiciels
  - Modèle système architectural : utile pour l'évaluation d'architectures et pour la détermination des algorithmes. On utilise pas la notion de temps dans ce type de modèle
  - Modèle performance système : utile pour la modélisation haut-niveau de la performance. On peut ou pas utiliser les notions de temps dans ce type de modèle

Les modèles de plus haut niveau 2

- SystemC est donc un langage pour modéliser le matériel et le logiciel, on a donc des modèles supplémentaires
- Les deux modèles suivants servent à modéliser le matériel à un plus haut niveau qu'avec VHDL ou Verilog:
  - Modèle au niveau transaction : utile pour modéliser une plate-forme exécutable; on l'utilise que pour le matériel
  - Modèle fonctionnel : pour la modélisation à un niveau inférieur au MNT et comprend l'architecture et les performances mais au niveau fonction

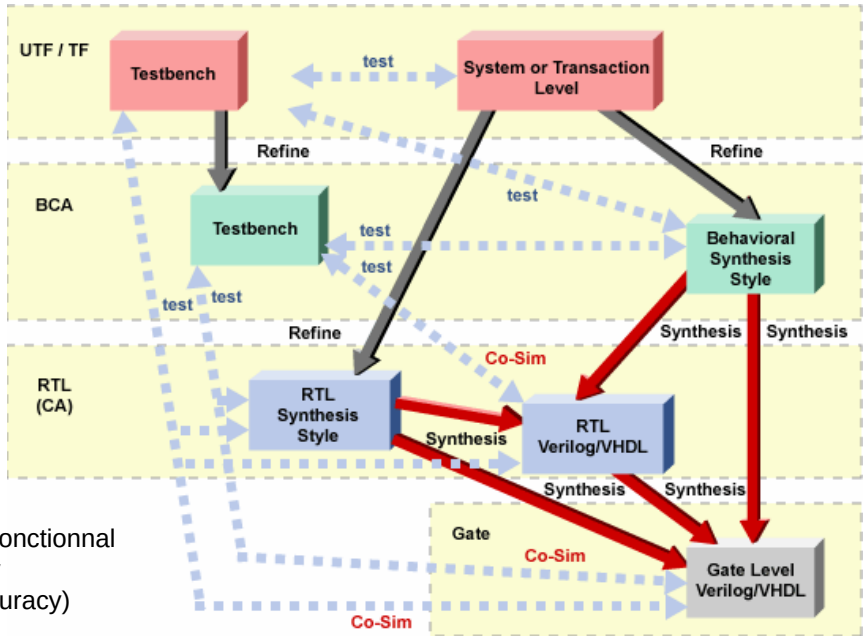
Le flot de conception SystemC



Le flot de conception d'un système complet (H/S)

Le flot de conception SystemC

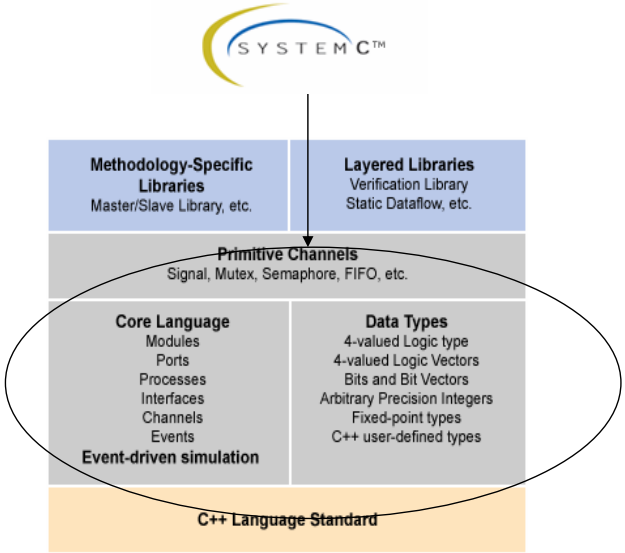
- **La partie matériel et le test**
- **On teste à différents niveaux**
- **Si le fichier de test est en VHDL il faut de la Co-simulation**



UTF/TF UnTimed/Timed Fonctionnal
BCA : Bus Cycle Accuracy
RTL(CA) : RTL (Cycle Accuracy)

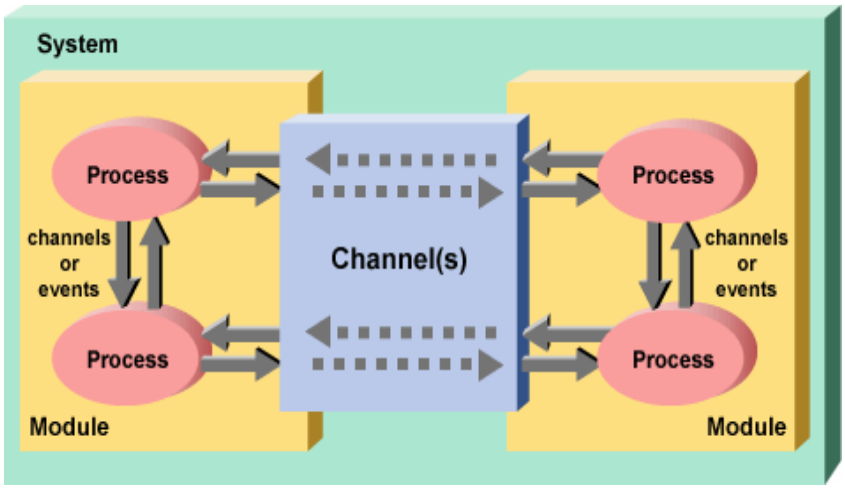
Architecture du langage SystemC

- La base du langage c'est C++
- Les librairies, méthodologies et standards de conception ne font pas partie de SystemC
- Le cœur du langage :
  - une simulateur événementiel
  - Modules et ports pour la structure du modèle
  - Interfaces et canaux pour modéliser la communication
  - Le typage propre SystemC pour modéliser le matériel
  - Les canaux primitives (signaux, FIFO)

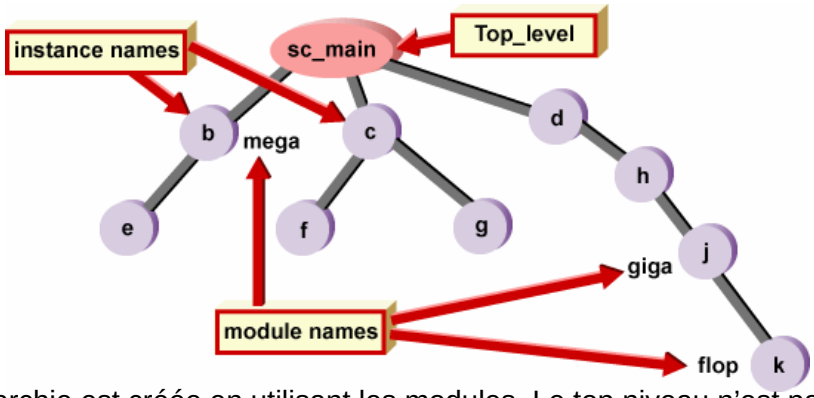


Un Système SystemC

- Un système SystemC est composé d'un ensemble de Modules
- Les modules contiennent de processus
- Les processus communiquent entre eux à travers de canaux ou des événements
- La communication entre modules se fait aussi à travers de canaux

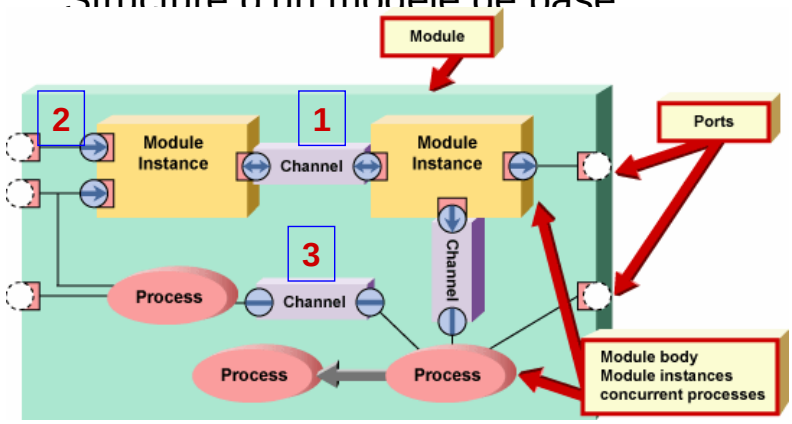


Un système SystemC



- La hiérarchie est créée en utilisant les modules. Le top niveau n'est pas un module mais la fonction sc_main()
- Les modules sont instanciés à l'intérieur de sc_main et à l'intérieur d'autres modules
- Dans la figure par exemple, l'instance k du module flop est instancié à l'intérieur de l'instance j du module giga

Structure d'un modèle de base

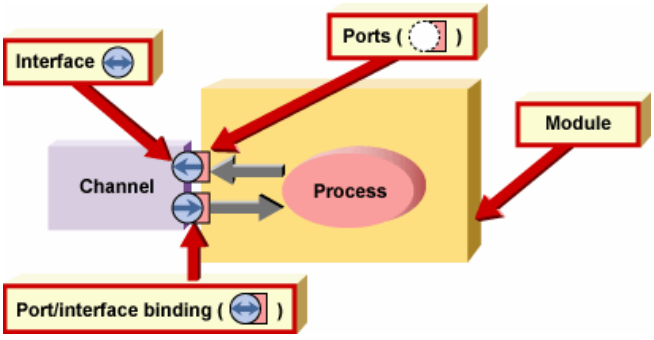


- Un module peut représenter un système, une carte, un circuit, etc.
- Les ports représentent l'interface, les connecteurs, les pins du circuits, etc., en fonction du type de module représenté
- Les modules à un même niveau sont interconnectés avec des canaux
- La connexion entre les instances des modules « père » et « enfant » se fait à travers les ports
- Les processus communiquent à travers de canaux également

1
2
3

Structure d'un modèle de base

- Trois éléments sont utilisés pour la communication :
  - Les interfaces
    - Ensemble des méthodes d'accès
    - Purement fonctionnel, aucune info sur le type d'implémentation
    - Ils sont liés aux ports et définissent ce que l'on peut faire à travers eux
  - Les canaux
    - Implémentent une ou plusieurs méthodes de l'interface



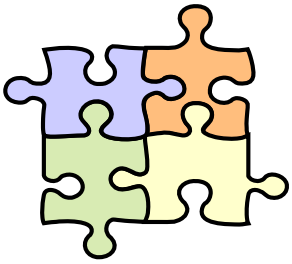
  - Les ports
    - Le port est l'objet à travers lequel les modules (et donc les processus) ont accès au méthodes des canaux
    - Un processus accède le canal en appliquant les méthodes d'interface à un port
    - Les ports peuvent être liés à multiples canaux

Les canaux

- Il y a deux types de canaux:
  - Primitifs
    - Sans structure visible
    - Ne contiennent pas de processus
    - Ne peuvent pas avoir accès à d'autres canaux
  - Hiérarchiques
    - Possèdent une structure
    - Contiennent de processus, ports et instances d'autres canaux ou modules
    - Peuvent accéder à d'autres canaux

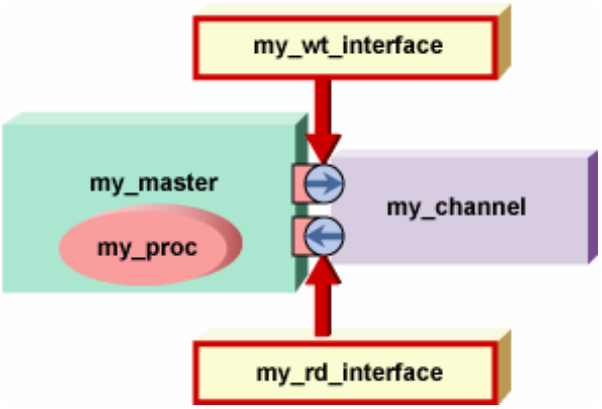
sc_signal<T>
sc_signal_rv<N>
sc_fifo<T>
sc_mutex
sc_sémaphore
sc_buffer<T>

Canaux primitifs



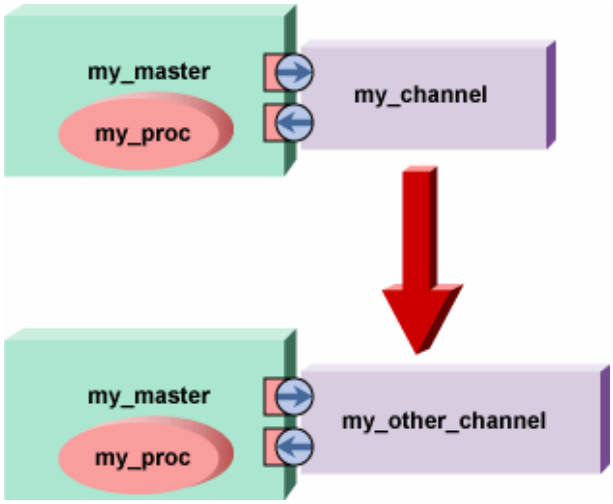
Exemple

- my_proc est un processus dans le module my_master
- L'interface my_wt_interface défini la méthode (fonction) wt() pour accéder à un canal à travers le port auquel il est lié
- L'interface my_rd_interface également défini une méthode (fonction) appelée rd() pour accéder à un canal
- L'implémentation de rd() et wt() se trouve dan le canal my_channel
- Quand my_proc appelle rd() ou wt() le code qui est exécuté se trouve dans my_channel

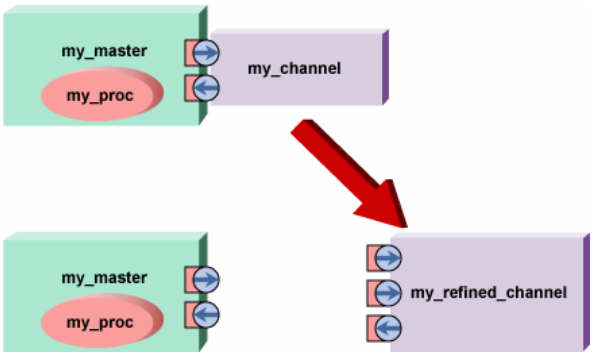


Exemple d'amélioration

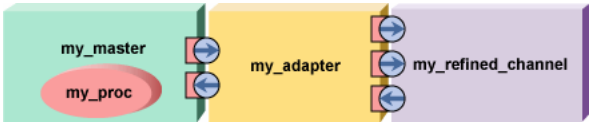
- Le canal my_channel peut être remplacé par my_other_channel avec une implémentation différente
- Condition : my_other_channe doit aussi implémenter rd() et wt()
- Dans ce cas my_proc dans my_master ne requiert d'aucune modification



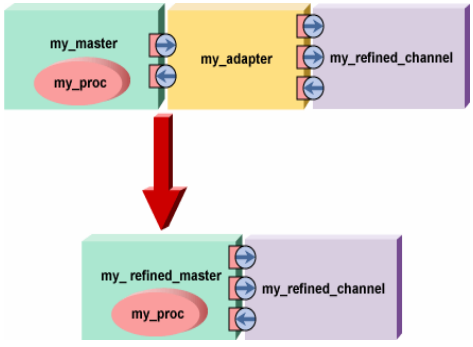
Amélioration indépendante



1. Remplacement du canal



2. Insertion d'un adaptateur



3. Intégration de l'adaptateur dans le module

Événement et sensibilité dynamique

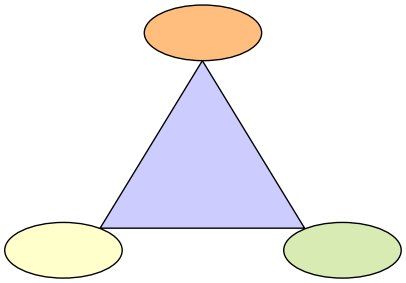
- L'événement (sc_event) est l'objet de synchronisation de base de SystemC
- Les processus sont synchronisés par un événement
- Les canaux utilisent les événement pour bloquer
- La liste de sensibilité est constituée des événements qui font déclencher les processus
- SystemC supporte deux type de liste de sensibilité
  - Statique : la liste est établie avant le début de la simulation
  - Dynamique : la liste peut être établie pendant la simulation
- Un processus peut attendre (wait()) l'occurrence d'un événement

Modèle du temps

- Le temps est un type entier non signé de 64 bits; les unités sont énumérés:
  - SC_FS,SC_PS,SC_NS,SC_US,SC_MS,SC_SEC
- Le type sc_time a été créé pour spécifier les valeurs de temps
- Syntaxe :
  - // time_value of type unit64
  - sc_time var_name(time_value, time_unit);
- Exemples :
  - sc_time t(20, SC_NS);
  - //var t of type sc_time with value of 20ns
  - sc_time r_time(1000, SC_NS);
- Quelques fonctions :
  - sc_stop() arrête la simulation et sc_start() renvoi le contrôle à sc_main
  - sc_time_stamp() renvoie un objet temps avec le temps de simulation
  - sc_simulation_time() renvoie un objet double avec le temps de simulation

SystemC Scheduler

- La simulation est base sur les événement
- Les processus sont exécutés et leurs sortie mis à jour en fonction des événements
- Le scheduler est chargé de gérer tous les événements et les mises à jours de canaux
- Les différentes phases
  - Initialisation
  - Évaluation
  - Répéter l'évaluation pour tous les processus
  - Mis à jour
  - Déterminer quel processus est prêt et revenir à l'évaluation
  - S'il n'y a plus de notification d'événement temporel la simulation se termine
  - Sinon, avancer au plus prochain événement dans le temps
  - Déterminer quels processus deviennent actif et revenir à l'évaluation



Déclaration de types

- Entiers signés et non signés. La précision est fixé dans la déclaration (64 bits ou moins)
  - Syntaxe
    - sc_int<length> variable_name, variable_name, ...;
    - sc_uint<length> variable_name, variable_name, ...;
  - Exemple
    - sc_int<5> a; // a est un entier signé de 5 bits
    - sc_uint<44> d; // d est un entier non signé de 44 bits
    - sc_int<3> c;
    - a = 13; // a prends 01101, a[4] = 0, a[3] = 1, ..., a[0] = 1
- Entiers signé et non signés de précision arbitraire (64 bits ou plus)
  - Syntaxe
    - sc_bigint<length> variable_name, variable_name, ...;
    - sc_biguint<length> variable_name , variable_name, ...;

Déclaration de

Values:	
'0'	logical zero or false (equivalent to bool false)
'1'	logical one or true (equivalent to bool true)
'Z'	high impedance
'X'	unknown

- Type logique
  - Syntaxe
    - sc_logic variable_name, variable_name
- Tableau de valeurs logiques
  - Syntaxe
    - sc_lv<length> variable_name, variable_name, ... ;
  - Exemple
    - sc_lv<38> a; // 38-bit bit vector
    - sc_lv<4> b;
    - sc_logic c;
    - b = "ZZZZ";

Les types à virgule flottante

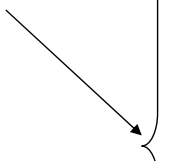
- Pour modéliser les opérations arithmétiques à plus haut niveau, quand on les passe en matériel ils sont codés à virgules fixes
  - Syntaxe pour les types de précision arbitraire
    - sc_fixed<wl, iwl, q_mode, o_mode, n_bits> object_name, ... ;
    - sc_ufixed<wl, iwl, q_mode, o_mode, n_bits> object_name, ... ;
      - * wl: longueur totale du mot, nombre de bits utilisés
      - * iwl: longueur de la partie entière, nombre de bits (.).
      - * q_mode: mode de quantization
      - * o_mode: mode de l'overflow
      - * n_bits: nombre de bits de saturation utilisé par le mode d'overflow
  - Example
    - sc_ufixed<8,4,SC_RND, SC_WRAP> val;
      - // val serait xxxx.xxxx
      - // arrondi au chiffre représentable le plus proche
      - // wrap on overflow
  - Syntaxe pour les types de précision fixe
    - sc_fixed_fast<wl, iwl, q_mode, o_mode, n_bits> object_name, object_name, ... ;
    - sc_ufixed_fast<wl, iwl, q_mode, o_mode, n_bits> object_name, object_name, ... ;

Comment sélectionner ?

- Pour avoir une performance optimale lors de la simulation il faut suivre les règles suivantes
  - Utiliser le typage C++ natif lors qu'il est possible
    - Bool pour les valeurs binaires d'un seul bit
    - Unsigned char pour les valeurs binaires de 8 bits
    - Unsigned short pour les valeurs binaires de 16 bits
  - Utiliser sc_uint et sc_int pour
    - Les données de moins de 64 bits et avec des valeurs binaires (0 ou 1)
    - Pour les opérations logiques et arithmétiques
  - Utiliser sc_logic et sc_lv pour
    - Les données avec quatre valeurs ('0', '1', 'Z', 'X')
  - Utiliser sc_bigint et scbiguint
    - Les données de plus de 64 bits
  - Utiliser les types à virgule fixe pour
    - Les opérations arithmétiques à virgule fixe

Les interfaces

- Définissent l'ensemble de méthodes d'accès
  - sc_signal_in_if
  - sc_signa_inout_if
  - sc_fifo_in
  - sc_fifo_inout_if
  - sc_mutex_if
  - sc_semaphore_if



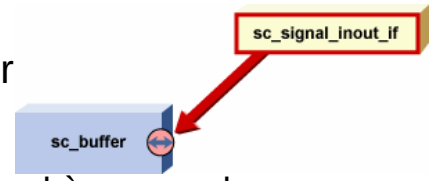
- bool **event**();
- const T& **get_data_ref** () const
- bool **negedge** () const
  - **Type** bool **and** sc_logic **only**
- const sc_event& **negedge**() const
  - **Type** bool **and** sc_logic **only**
- const sc_event& **posedge**() const
  - **Type** bool **and** sc_logic **only**
- bool **posedge** () const
  - **Type** bool **and** sc_logic **only**
- const T& **read** () const
- Const sc_event& **value_changed_event** () const

Le canal sc_signal

- sc_signal implémente l'interface sc_signal_inout_if
- Il s'agit des signaux, utilisés au niveau RTL
- On les utilise pour la communication point-à-point ou multipoints
- Les signaux sont non résolus
  - Une seule source d'écriture
  - Par contre, plusieurs demandeurs de lecture
- Le signal contient une seule valeur, qui peut avoir la taille définit par l'utilisateur
- Comme dans VHDL le signal garde un historique
  - Current_value, new_value, old_value
- Syntaxe
  - sc_signal<T> signal_name, signal_name, ... ;
- Exemple

```
SC_MODULE (module_name) {
sc_signal<int> d ;
sc_signal<char> e ;
sc_signal<sc_int<10> > f;
// rest of module not shown
} ;
```

Le canal sc_buffer



- Se comporte exactement comme sc_signal à une seule exception
  - value_changed_event est toujours activé quand l'objet est accédé en écriture même si la valeur reste inchangé
  - Syntaxe
    - sc_buffer<T> buffer_name, buffer_name, . . . ;

Le canal sc_signal_rv

- Se comporte comme sc_signal<sc_lv<w> > mais en version résolu
- Ceci permet qu'il y ait plusieurs sources d'écriture
- C'est utilisé essentiellement pour modéliser les bus de données
- Syntaxe

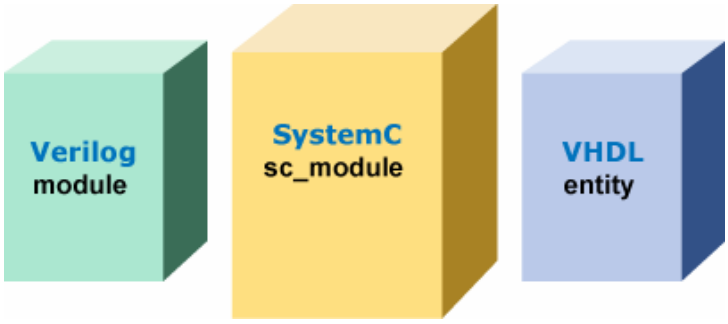
```
sc_signal_rv<W> signal_name, signal_name, ... ;
   * W est le nombre de bits
   * chaque bit peut avoir la valeur '0', '1', 'X' ou 'Z'
   * Il n'y pas de restriction de taille
   * Il faut utiliser l'opérateur [ ] pour accéder à chaque bit individuellement
```

- Exemple

```
SC_MODULE (module_name) {
sc_signal_rv<8> d ;
sc_signal_rv<44> e ;
sc_signal_rv<16 > f;
// le reste du module n'est pas montré} ;
```

Resolve	X	0	1	Z
X	X	X	X	X
0	X	0	X	0
1	X	X	1	1
Z	X	0	1	Z

sc_module

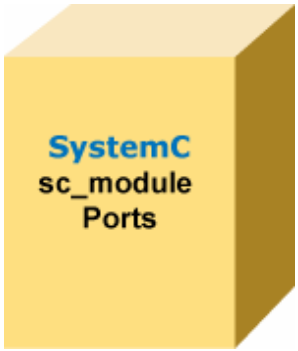


- Le sc_module est l'équivalent de l'entité en VHDL
- Tous les modules doivent être décrit avec
  - un fichier d'entête (module_name.h)
  - un fichier d'implémentation (module_name.cpp)
- Syntaxe
  - SC_MODULE (module_name) { // body of module
  } ;

Les ports

- Sont crées à partir de la classe sc_port et lié à une interface
- Syntaxe
  - sc_port<interface_type, N> port_name, port_name,... ;
    - N est le nombre de canaux que peuvent se connecter
- Exemple

```
SC_MODULE(my_module) {
// port in_p a deux canaux connectés
sc_port<sc_signal_in_if<int>,2> in_p;
sc_port<sc_signal_inout_if<int> > out_p;
// corps du module
};
```



Les constructeur du module

- Le module a un constructeur puisqu'il s'agit d'une classe C++
- Le constructeur crée (instancie) et initialise un module et sa structure de données avec de valeurs connus
- Les processus également sont enregistrés dans un constructeur
- SystemC contient une macro pour l'appel du constructeur sc_ctor avec un seul paramètre : le nom du module
- Exemple

```
SC_MODULE(module_name) {
  // Ports, internal signals, processes, other methods
  // Constructor
  SC_CTOR(module_name) /* : liste d'initialisation */ {
    // process registration
    // declarations of sensitivity lists
    // module instantiations
    // port connection declarations
  }
};
```

Exemple

```
// transmit.h

SC_MODULE(transmit) {
        sc_in<packet_type>      tpackin;        // input port
        sc_in<bool>             timeout;        // input port
        sc_out<packet_type>     tpackout;       // output port
        sc_inout<bool>          start_timer;    // output port
        sc_in<bool>             clock;          // input port

        int buffer;
        int framenum;
        packet_type packin, tpackold;
         packet_type s;
        int retry;
        bool start;

        void send_data();
        int get_data_fromApp();

        // Constructor
        SC_CTOR(transmit) {
                SC_METHOD(send_data); // Method Process
                sensitive << timeout;
                sensitive_pos << clock;
                framenum = 1;
                retry = 0;
                start = false;
                buffer = get_data_fromApp();
        }
};
// transmit.
```

```
// transmit.cc
#include "transmit.h"
int transmit::get_data_fromApp() {
        int result;
        result = rand();
        cout <<"Generate:Sending Data Value = "<<result << "\n";
    return result;
}
void transmit::send_data() {
        if (timeout) {
            s.info = buffer;
            s.seq = framenum;
            s.retry = retry;
            retry++;
        tpackout = s;
        start_timer = true;
        cout <<"Transmit:Sending packet no. "<<s.seq            << "\n";
} else {
        packin = tpackin;
        if (!(packin == tpackold)) {
                if (packin.seq == framenum) {
                        buffer = get_data_fromApp();
                        framenum++;
                        retry = 0;
                }
        tpackold = tpackin;
        s.info = buffer;
        s.seq = framenum;
        s.retry = retry;
        retry++;
        tpackout = s;
        start_timer = true;
        cout <<"Transmit:Sending packet no. "<<s.seq            << "\n";
        }
}
}
```


Packet_type

```
// packet.h file
#ifndef PACKETINC
#define PACKETINC
#include "systemc.h"
struct packet_type {
        long info;
        int seq;
        int retry;
        inline bool operator == (const packet_type& rhs) const
        {
                return (rhs.info == info && rhs.seq ==          seq &&
                rhs.retry == retry);
        }
};

extern
void sc_trace(sc_trace_file *tf, const packet_type& v, const sc_string&
        NAME);

#endif
```

```
// packet.cc file
#include "packet.h"
void sc_trace(sc_trace_file *tf, const packet_type& v,
const sc_string& NAME) {
sc_trace(tf,v.info, NAME + ".info");
sc_trace(tf,v.seq, NAME + ".seq");
sc_trace(tf,v.retry, NAME + ".retry");
}
```

Le module transmit possède les ports d'e/s suivants :

```
sc_in<sc_lv<16> >       tpackin;   // input port
sc_in<bool>             timeout;   // input port
sc_out<sc_lv<16> >      tpackout;          // output port
sc_inout<bool>                  start_timer;        // output port
sc_in<bool>             clock;             // input port

int buffer;
int framenum;
packet_type packin, tpackold;
 packet_type s;
int retry;
bool start;
// Déclaration de fonction dans le fichier .h et implémenté dans .cpp
void send_data();
int get_data_fromApp();
```


Le constructeur

```
// Constructor
SC_CTOR(transmit) {
    SC_METHOD(send_data); // Method Process
    sensitive << timeout;
    sensitive_pos << clock;
    framenum = 1;
    retry = 0;
    start = false;
    buffer = get_data_fromApp();
}
```

L'implémentation des méthodes

```
int transmit::get_data_fromApp() {
        int result;
        result = rand();
        cout <<"Generate:Sending Data Value = "<<result << "\n";
      return result;
}
```

La méthode get_data_fromApp() est une méthode local et non pas un processus

Elle sert à générer la paquet de données qui sera envoyé par le module transmit en utilisant la fonction random de C++ rand ()

Le processus

```
void transmit::send_data() {
        if (timeout) {
            s.info = buffer;
            s.seq = framenum;
            s.retry = retry;
            retry++;
        tpackout = s;
        start_timer = true;
        cout <<"Transmit:Sending packet no. "<<s.seq << "\n";
} else {
        packin = tpackin;
        if (!(packin == tpackold)) {
                if (packin.seq == framenum) {
                                buffer = get_data_fromApp();
                                framenum++;
                                retry = 0;
                }
        tpackold = tpackin;
        s.info = buffer;
        s.seq = framenum;
        s.retry = retry;
        retry++;
        tpackout = s;
        start_timer = true;
        cout <<"Transmit:Sending packet no. "<<s.seq << "\n";
        }
}
```

Main.cc

```
// main.cc
#include "packet.h"
#include "timer.h"
#include "transmit.h"
#include "channel.h"
#include "receiver.h"
#include "display.h"
int sc_main(int argc, char* argv[]) {
sc_signal<packet_type> PACKET1, PACKET2, PACKET3,
PACKET4;
sc_signal<long> DOUT;
sc_signal<bool> TIMEOUT, START;
sc_clock CLOCK("clock", 20); // transmit clock
sc_clock RCLK("rclk", 15); // receive clock
transmit t1("transmit");
t1.tpackin(PACKET2);
t1.timeout(TIMEOUT);
t1.tpackout(PACKET1);
t1.start_timer(START);
t1.clock(CLOCK);
d1 <<DOUT;
timer tm1("timer");
tm1 <<START<<TIMEOUT<<CLOCK.signal();
```

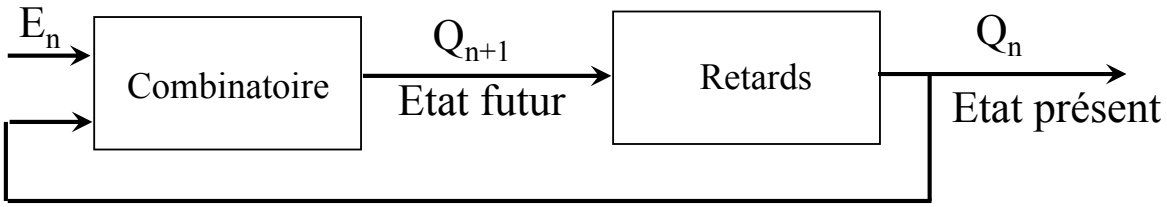
```
// tracing:
// trace file creation
sc_trace_file *tf = sc_create_vcd_trace_file
("simplex");
// External Signals
sc_trace(tf, CLOCK.signal(), "clock");
sc_trace(tf, TIMEOUT, "timeout");
sc_trace(tf, START, "start");
sc_trace(tf, PACKET1, "packet1");
sc_trace(tf, PACKET2, "packet2");
sc_trace(tf, PACKET3, "packet3");
sc_trace(tf, PACKET4, "packet4");
sc_trace(tf, DOUT, "dout");
sc_start(10000);
sc_close_vcd_trace_file(tf);
return(0);
}
```


Compléments de cours

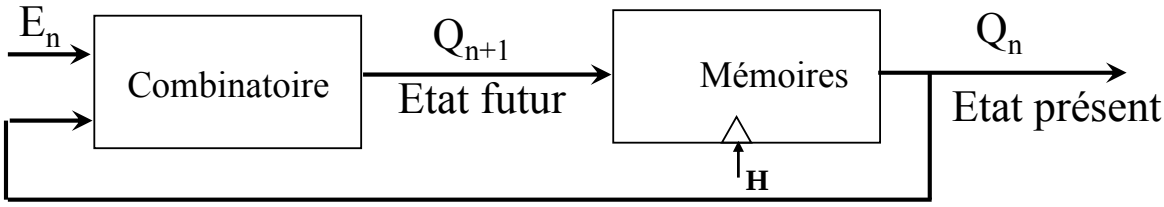
Exercices corrigés

Systèmes séquentiels
Synchrone et Asynchrone

- Système asynchrone ☞ Plus rapide mais de conception complexe



- Système synchrone ☞ Conception moins complexe

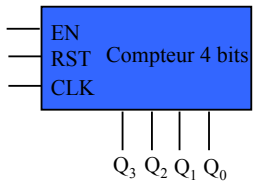


Exercice : Synthèse d'un compteur 4 bits modulo 10

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

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

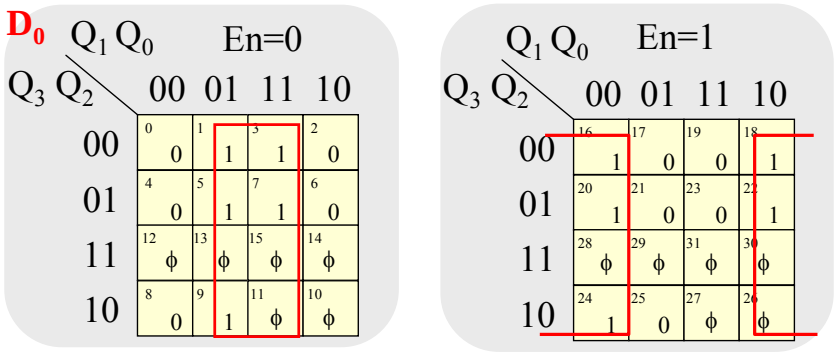
RST = '1' -RAZ
RST = '0' - comptage



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

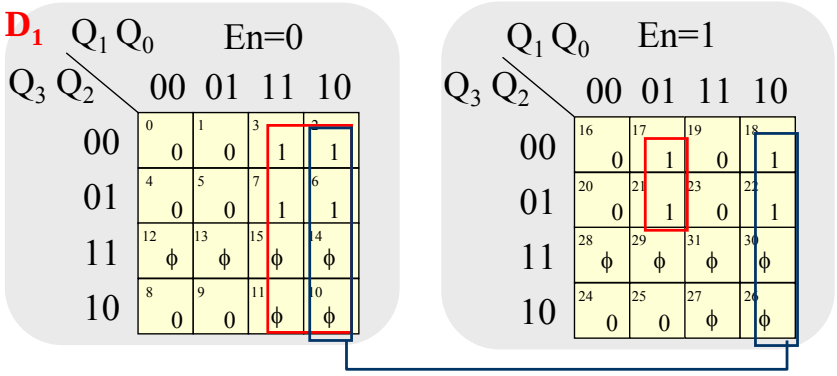
Sorties des bascules D — **état futur des entrées des bascules D pour EN='1'** — **état futur des entrées des bascules D pour EN='0'**

Exercice : Synthèse d'un compteur 4 bits modulo 10



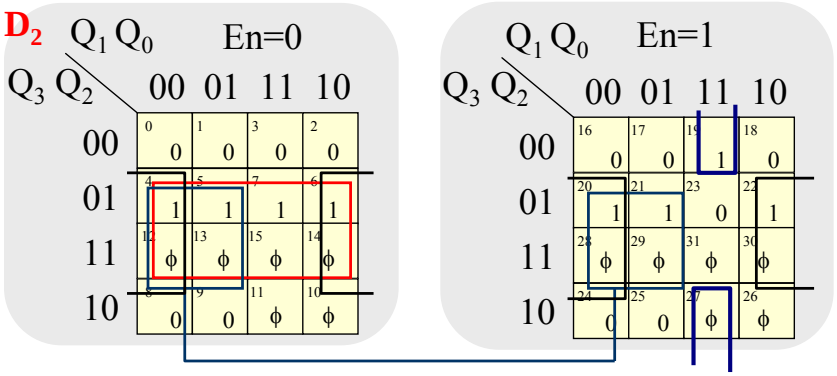
$$D_0 = Q_0 \oplus E_n$$

Exercice : Synthèse d'un compteur 4 bits modulo 10



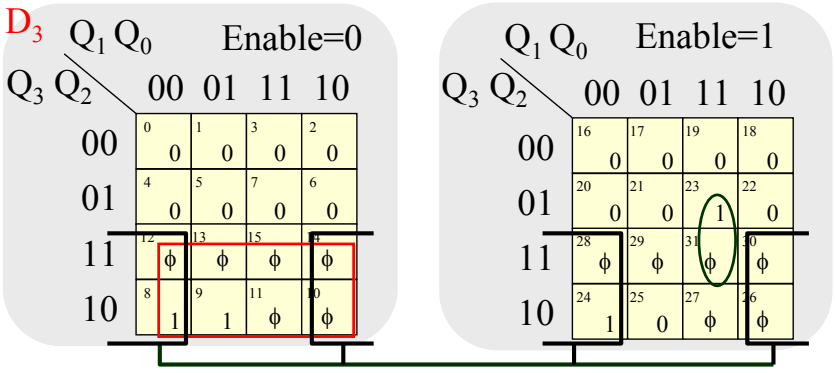
$$D_1 = \overline{E}_n\, Q_1 + Q_1\overline{Q}_0 + E_n\overline{Q}_3\overline{Q}_1 Q_0$$

Exercice : Synthèse d'un compteur 4 bits modulo 10



$$D_2 = \overline{E}_n\, Q_2 + Q_2\overline{Q}_1 + Q_2\overline{Q}_0 + E_n\overline{Q}_2 Q_1 Q_0$$

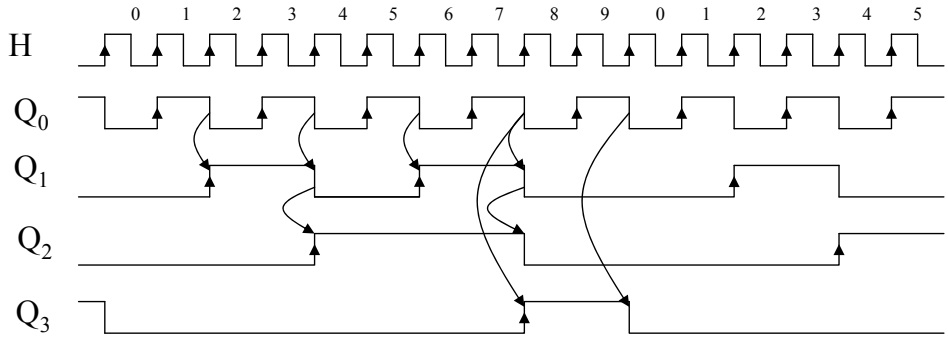
Exercice : Synthèse d'un compteur 4 bits modulo 10



$$D_3 = \overline{E}_n\, Q_3 + Q_3\overline{Q}_0 + E_n Q_2 Q_1 Q_0$$

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

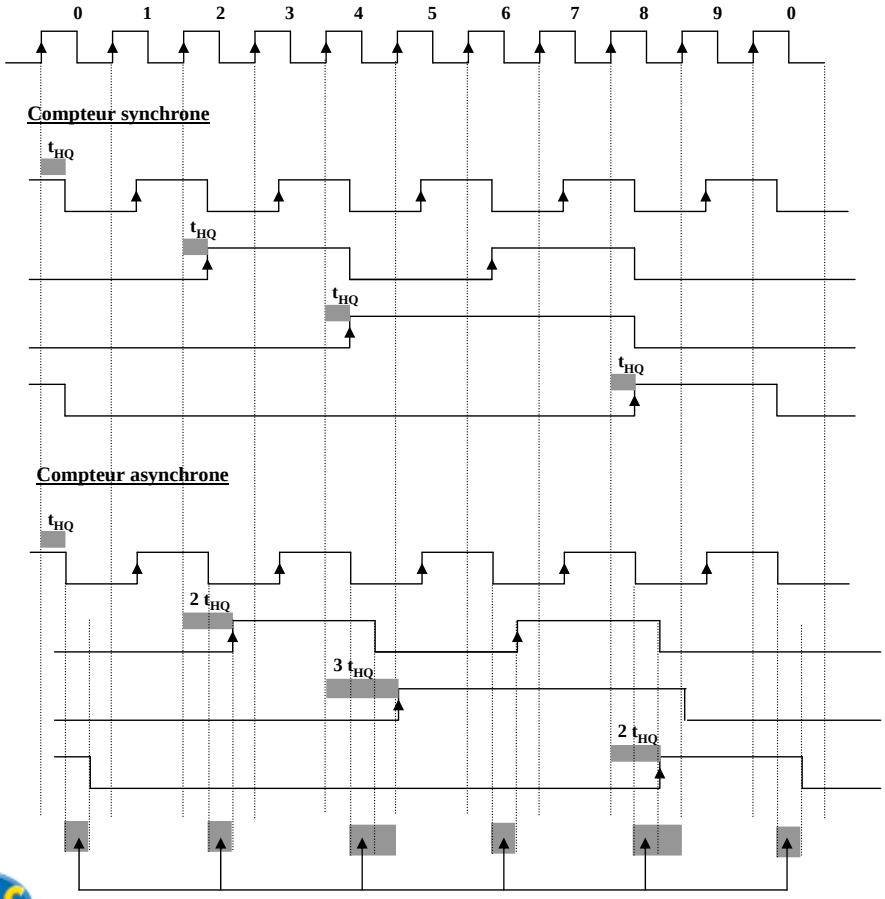
- Les bascules ne sont plus reliées à la même horloge
- Choisir le signal de commande le plus adapté



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

$D_0 = \overline{Q_0}$
$D_1 = \overline{Q_3}\overline{Q_1}$
$D_2 = \overline{Q_2}$
$D_3 = Q_2Q_1$

D.D



Etats aléatoires

D.D

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;
```


Régistre 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;
```