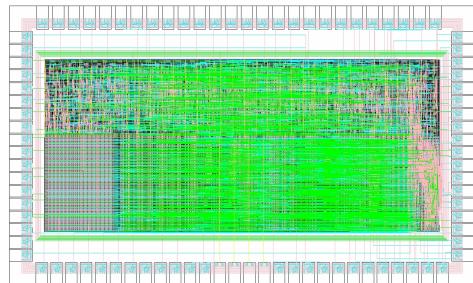


VLSI-2

outils de CAO
Alliance / Coriolis

UPMC / M2 SESI / VLSI2

Exemple : Processeur MIPS 32 bits



UPMC / M2 SESI / VLSI2

introduction

Avertissement

On ne s'intéresse qu'à une classe particulière de circuits,
Les circuits intégrés CMOS **numériques synchrones** :

– Numériques

Les signaux d'entrée et de sortie, ainsi que les variables internes sont représentées par des variables Booléennes ne pouvant prendre que des valeurs discrètes : 0 ou 1 (U, X, ...).

– Synchrones

Les valeurs stockées dans les registres internes du circuit ne peuvent être modifiées qu'à certains instants, définis par les fronts d'un signal particulier, appelé signal d'horloge.

UPMC / M2 SESI / VLSI2

introduction

Plan

- Les trois vues d'un circuits
- Méthodologie de conception
- Modélisation et simulation
- Automate d'états finis synchrones
- Circuits numériques synchrones
- Chaîne de simulation
- Chaîne de synthèse

UPMC / M2 SESI / VLSI2

Les trois vues

UPMC / M2 SESI / VLSI2

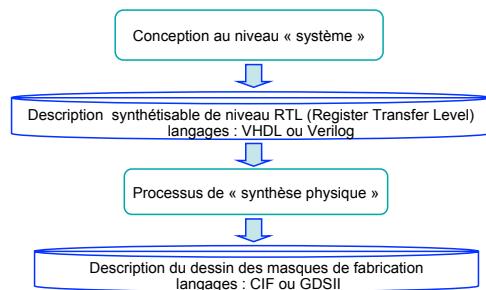
Les trois vues d'un circuit intégré

- Dans la phase de synthèse physique, on peut décrire le composant à trois niveaux d'abstraction :
 - **vue comportementale** (« behaviour ») : Elle permet de simuler et donc d'analyser le comportement, mais ne décrit pas la structure interne du composant modélisé.
 - **vue structurelle** (« net-list ») : elle décrit la structure interne, c'est à dire la façon dont le composant peut se décomposer en une interconnexion de composants plus simples.
 - **vue physique** (« layout ») : elle décrit le dessin des masques de fabrication qui sont utilisés pour graver le silicium.
- Le processus de conception consiste à transformer progressivement la description comportementale en une description physique (utilisable par le fabricant de circuits). Les outils CAO de synthèse physique permettent d'automatiser cette transformation.

UPMC / M2 SESI / VLSI2

flot de conception

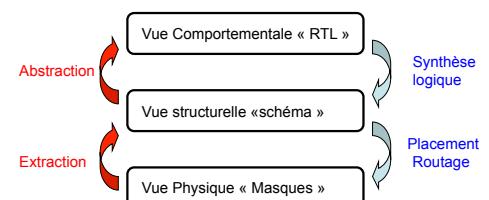
Flot de conception



UPMC / M2 SESI / VLSI2

flot de conception

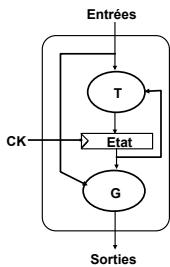
Les trois vues



UPMC / M2 SESI / VLSI2

flot de conception

La vue comportementale RTL



Les registres sont explicités.
Le modèle de référence est celui des Automates d'états synchrones :

Fonction de transition :
 $\text{NextEtat} \Leftarrow T(\text{Etat}, \text{Entrées})$

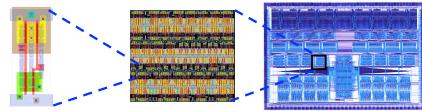
Fonction de génération :
 $\text{Sorties} \Leftarrow G(\text{Etat}, \text{Entrées})$

UPMC / M2 SESI / VLSI2

flot de conception

La vue physique

Elle représente le dessin des masques de fabrication.
C'est une description modulaire et hiérarchique



Une cellule élémentaire

Un bloc fonctionnel

Un circuit complet

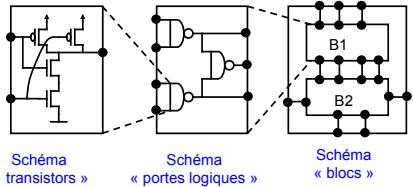
UPMC / M2 SESI / VLSI2

flot de conception

La vue structurelle

C'est une description hiérarchique du schéma d'interconnexion.
Les éléments terminaux de cette description sont

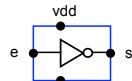
- soit des cellules logiques
- soit des transistors



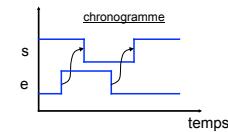
UPMC / M2 SESI / VLSI2

flot de conception

Vue comportementale inverseur CMOS



Ce composant matériel est décrit comme une « boîte noire », dont on ne connaît que l'interface (ports d'entrée/sortie), et le comportement logico-temporel :



UPMC / M2 SESI / VLSI2

LANGAGE VHDL

```

entity inveror is
  -- liste des ports d'entrée/sortie
  port (
    e : in bit;
    s : out bit;
    vss : in bit;
    vdd : in bit
  );
end inveror;
  
```

```

architecture vbe of inveror is
  -- comportement logico-temporel
begin
  s <= not e after 200 ps;
end vbe;
  
```

exemple inverseur

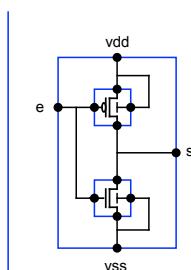
Vue structurelle inverseur CMOS

La structure interne du composant est décrite comme une interconnexion de composants matériels plus simples :
Dans le cas de l'inverseur :

- un transistor N
- un transistor P

Les deux transistors N et P peuvent être considérés comme des interrupteurs contrôlés par la valeur du signal appliquée sur la grille :

Grille	0	1
Transistor N	bloqué	passant
Transistor P	passant	bloqué



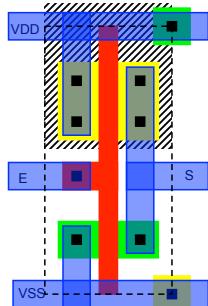
exemple inverseur

UPMC / M2 SESI / VLSI2

UPMC / M2 SESI / VLSI2

Méthodologie de conception

Vue Physique inverseur CMOS



Les 4 ports d'entrée/sortie de l'inverseur (e,s,vdd,vss) sont en métal 1.

La « boîte d'aboutement » définit l'encombrement de la cellule, mais ne correspond pas à une couche physique.

UPMC / M2 SESI / VLSI2

exemple inverseur

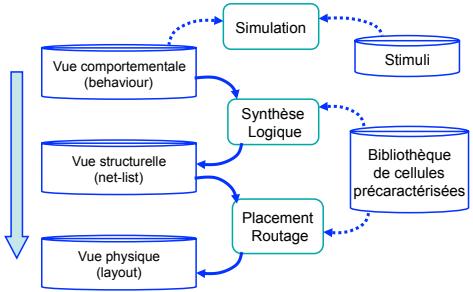
Top-down ou Bottom-up ?

- La méthode générale de conception est une méthode **descendante** (*top-down*). On met en œuvre des techniques de synthèse logique et de placement/routage, en partant de la spécification comportementale pour aboutir au dessin des masques.
- Cependant, pour vérifier que l'implémentation est conforme à la spécification, il faut des étapes d'extraction et/ou d'abstraction qui s'inscrivent dans une stratégie **ascendante** (*bottom up*).
- La définition du schéma par raffinement successif est un processus **descendant**, mais les éléments terminaux du schéma (portes logiques) doivent nécessairement appartenir à une (ou plusieurs) bibliothèques de cellules précaractérisées.
- La méthode de conception est donc à la fois **descendante et ascendante**.

UPMC / M2 SESI / VLSI2

Méthodologie

Conception descendante



UPMC / M2 SESI / VLSI2

Méthodologie

Spécification...

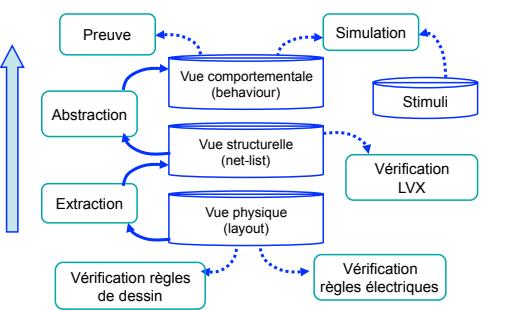
Règle absolue : Il faut spécifier avant de développer...

- Le point de départ de la synthèse physique est donc toujours la **description comportementale**, qui joue le rôle de spécification, pour la synthèse physique, et qui sera utilisée comme référence à toutes les étapes de conception ultérieures.
- Cette description comportementale peut être elle-même générée par des outils de synthèse de plus haut niveau (synthèse d'architecture, compilateurs de silicium), à partir de descriptions plus abstraites.
- Dans ce cours (et dans les TP associés), on écrira la description comportementale « à la main ».

UPMC / M2 SESI / VLSI2

Méthodologie

Vérification « ascendante »



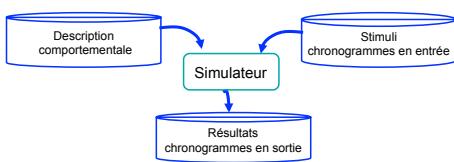
UPMC / M2 SESI / VLSI2

Méthodologie

Modélisation

Validation / Simulation...

Le principal outil de validation de la description comportementale est la simulation, ce qui nécessite donc de développer les stimuli qui seront appliqués sur le modèle du composant à valider.



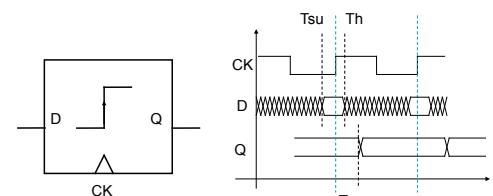
La qualité de la validation dépend de la qualité des stimuli, **la génération des stimuli prend plus de temps que l'écriture du modèle du composant...**

UPMC / M2 SESI / VLSI2

Modélisation : validation

Bascule D (D flip-flop)

Une bascule D permet de « mémoriser » 1 bit
• L'écriture d'une nouvelle valeur a lieu lors du front montant du signal CK
• L'entrée D doit être stable un peu avant (Tsu) et un peu après (Th) le front
• La sortie Q change de valeur au plus une fois par cycle après le front (Ta)



UPMC / M2 SESI / VLSI2

Modélisation : validation

Simulation « Zero-delay »

Au début du processus de conception, on ne connaît pas les temps de propagation des signaux (ces temps dépendent du schéma en portes qui sera généré par la synthèse, ainsi que du placement/routage).

On écrit donc un modèle « zero delay », où les seules informations temporelles sont apportées par les fronts successifs du signal d'horloge, car les temps de propagation dans la logique combinatoire sont supposés infiniment petits par rapport au temps de cycle.

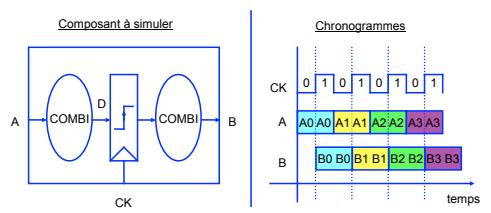
Les stimuli (c'est à dire les chronogrammes des signaux d'entrée) doivent donc être positionnés par rapport aux fronts du signal d'horloge.

UPMC / M2 SESI / VLSI2

Modélisation : validation

Structure temporelle des stimuli

- A chaque port d'entrée est associé un chronogramme.
- Si le composant simulé ne contient que des registres à échantillonnage sur front (bascules D), on utilise un signal d'horloge CK à deux phases.
- Les signaux d'entrée ne doivent pas changer de valeur au moment du front actif du signal CK.

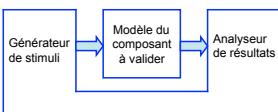


UPMC / M2 SESI / VLSI2

Modélisation : validation

Simulation : Banc de test

- Un banc de test (test-bench) est un modèle comportemental de l'environnement dans lequel sera plongé le composant à valider.
- La technique du banc de test permet d'éviter l'écriture - fastidieuse, voire impossible - des séquences de stimuli, et les - inévitables - erreurs humaines dans la phase d'analyse des résultats de simulation.



Attention : L'écriture du test-bench peut prendre autant de temps que l'écriture du modèle du composant à valider.

Suggestion : Le modèle du test-bench et le modèle du composant peuvent être développés en parallèle par des personnes différentes.

UPMC / M2 SESI / VLSI2

Modélisation : validation

Le niveau « RTL »

- Les descriptions comportementales de type 1 servent d'entrée aux outils de synthèse d'architecture, et ne seront pas considérées ici.
- Les descriptions comportementales de type 2 et 3 sont dites de niveau « RTL » (Register Transfer Level). Une description RTL est assez proche de la réalisation matérielle, puisqu'on peut décrire explicitement - cycle par cycle - la succession des états internes du composant (c'est à dire les valeurs stockées dans les registres).
- Le niveau d'abstraction RTL est fondamental, car il sert d'entrée aux outils de « synthèse physique », qui permettent d'automatiser la génération du dessin des masques (vue physique). Ce sont les différentes étapes de cette « synthèse physique », qui sont analysées dans ce cours.

UPMC / M2 SESI / VLSI2

Modélisation : validation

Modélisation comportementale...

1. Niveau « algorithmique »

- interface défini « au bit près » par une liste de signaux
- pas de signal d'horloge explicite
- pas d'identification des registres
- sémantique séquentielle : un (ou plusieurs) processus

2. Niveau « automate abstrait »

- interface défini « au bit près » par une liste de signaux
- le signal d'horloge est explicite
- les registres sont identifiés, mais les valeurs stockées (états) ne sont pas représentées au bit près.
- sémantique séquentielle : un (ou plusieurs) processus

3. Niveau « data-flow »

- interface défini « au bit près » par une liste de signaux
- le signal d'horloge est explicite
- les registres sont identifiés, et les valeurs stockées sont représentées au bit près.
- sémantique « data-flow » : assignations concurrentes

UPMC / M2 SESI / VLSI2

Modélisation : validation

Langages de description de matériel

On utilise généralement des langages spécialisés, appelés HDL (Hardware Description Language) pour décrire le comportement du matériel :

Différents objectifs :

- simulation logico-temporelle
- synthèse logique
- preuve (?)

Des caractéristiques communes :

- Expression du parallélisme du matériel
- Descriptions mixtes comportementale et structurelles
- support de types spécialisés (vecteurs de bits)
- Représentation explicite du temps

Principaux HDLs :

VHDL, VERILOG, SYSTEMC

Modélisation : validation

Langage VHDL

Un modèle VHDL d'un composant matériel comporte deux parties :

- La partie « Entity » décrit l'interface du composant.
- La partie « Architecture » décrit son comportement, ou sa structure interne.

Il peut exister plusieurs architectures pour un même composant, correspondant à différents niveaux d'abstraction :

- description comportementale « algorithmique »
- description comportementale « automate abstrait »
- description comportementale « data-flow »
- description structurelle

Attention

Le langage VHDL permettant de représenter différents niveaux d'abstraction d'un circuit, l'expression « modèle VHDL » sans qualificatif est extrêmement ambiguë, et doit être évitée...

UPMC / M2 SESI / VLSI2

Modélisation : validation

VHDL : assignations concurrentes

Un modèle VHDL est dit **data-flow** s'il ne comporte que des assignations concurrentes : Le circuit est décrit comme un ensemble de **processus** interconnectés par des signaux, qui s'exécutent en parallèle.

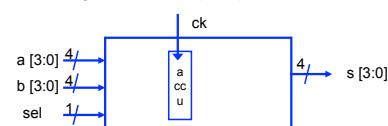
- Chaque assignation correspond à un processus.
- L'ordre d'écriture des assignations concurrentes est sans signification.
- un signal ne peut être assigné qu'une seule fois (assignation unique).
- Une description comportementale de type « data-flow » permet de décrire le parallélisme intrinsèque d'un réseau Booléen.

UPMC / M2 SESI / VLSI2

Modélisation : validation

Exemple : addaccu

Le composant addaccu est décrit comme une boîte « grise » possédant 4 signaux d'entrée : a, b, sel, ck (10 bits au total) et un seul signal de sortie : s (4 bits).



On souhaite le comportement suivant :

- Si (sel == 0) sum <= a + b
- Sinon sum <= accu + b
- s <= sum
- Quand (ck passe de 0 à 1) accu <= sum

UPMC / M2 SESI / VLSI2

Modélisation : validation

Modélisation VHDL addaccu : Entity

```

entity addaccu is
  -- liste des ports d'entrée/sortie
  port (
    a : in std_logic_vector(3 downto 0);
    b : in std_logic_vector(3 downto 0);
    sel : in std_logic;
    ck : in std_logic;
    s : out std_logic_vector(3 downto 0);
    vss : in std_logic;
    vdd : in std_logic
  )
end addaccu;

  • VHDL ne fait pas de distinction entre majuscules et minuscules.
  • Les commentaires commencent par « -- ».
  • Dans l'exemple proposé ici, les mots-clés et les séparateurs du langage VHDL sont en vert.
  
```

UPMC / M2 SESI / VLSI2

Modélisation : validation

Modélisation VHDL addaccu : Architecture vbe

```

Architecture vbe of addaccu is
-- déclaration des signaux
signal r_accu : std_logic_vector ( 3 downto 0 ) ; -- register
signal x      : std_logic_vector ( 3 downto 0 ) ;
signal sum   : std_logic_vector ( 3 downto 0 ) ;
signal r     : std_logic_vector ( 3 downto 0 ) ;

-- description du comportement
begin
  -- selection du deuxième opérande
  x[3 downto 0] <= accu[3 downto 0] when select
    else a[3 downto 0];
  ...

```

UPMC / M2 SESI / VLSI2

Modélisation : validation

Modélisation VHDL addaccu : Architecture vbe

```

-- calcul de la somme ... et du report
-- sum(3 downto 0) <= b(3 downto 0) xor x(3 downto 0) xor r(3 downto 0) ;
-- r(3 downto 1)    <= ( b(2 downto 0) and x(2 downto 0) ) or
--                  ( b(2 downto 0) and r(2 downto 0) ) or
--                  ( x(2 downto 0) and r(2 downto 0) ) ;
-- r(0) <= '0' ;
sum <= x + b ;

reg : process(ck)
  if ( ck and not ck'stable ) begin
    accu(3 downto 0) <= sum(3 downto 0) ;
  end if;
end process reg;

s(3 downto 0) <= sum(3 downto 0) ;           -- affectation signal de sortie
end vbe ;

```

UPMC / M2 SESI / VLSI2

Modélisation : validation

Table de vérité additionneur 1 bit

$$\begin{array}{r} a_i \\ + b_i \\ + c_i \\ \hline c_{i+1} s_i \end{array}$$

La somme de trois bits de même poids numérique peut prendre trois valeurs : 0 , 1 , 2 , 3
Cette somme peut se recoder sur deux bits c_{i+1} et s_i
(c_{i+1} a un poids numérique double de s_i)

$$s_i = a_i \oplus b_i \oplus c_i$$

$$c_{i+1} = a_i \cdot b_i + c_i \cdot (a_i + b_i)$$

a _i	b _i	c _i	c _{i+1}	s _i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table de vérité

UPMC / M2 SESI / VLSI2

Modélisation : validation

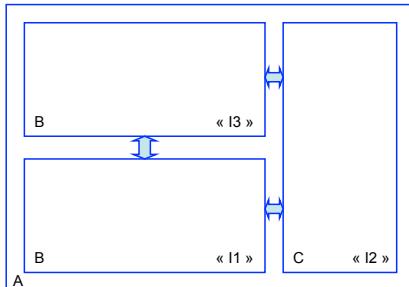
Raffinement progressif du schéma / 1

On dispose d'une spécification comportementale du composant A, et on souhaite obtenir une description structurelle sous la forme d'un schéma multi-niveaux...

A

Modélisation : vue structurelle

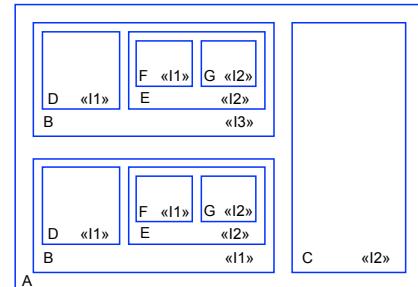
Raffinement progressif du schéma / 2



UPMC / M2 SESI / VLSI2

Modélisation : vue structurelle

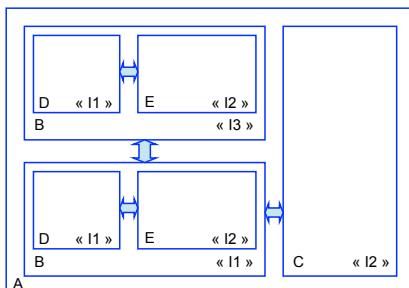
Raffinement progressif du schéma / 4



UPMC / M2 SESI / VLSI2

Modélisation : vue structurelle

Raffinement progressif du schéma / 3

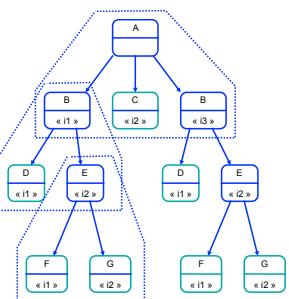


UPMC / M2 SESI / VLSI2

Modélisation : vue structurelle

Arbre d'instanciation

- La racine de l'arbre n'a pas de nom d'instance.
- Les feuilles de l'arbre sont les composants « terminaux ». (ils n'en contiennent pas d'autres)
- Pour que la description soit simulable, il faut disposer d'une description comportementale pour tous les composants terminaux.
- Chaque niveau du schéma hiérarchique multi-niveaux peut être décrit dans un fichier séparé.



UPMC / M2 SESI / VLSI2

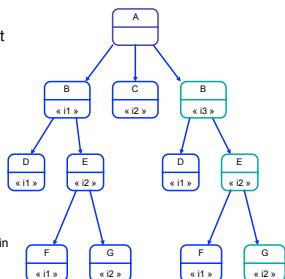
Modélisation : vue structurelle

Occurrences

- On appelle **occurrence** un objet quelconque du schéma multi-niveaux dans son contexte d'instanciation.
- Pour désigner une occurrence, on utilise le **cheminom** :

« A.i3.i2.sig »

A : nom de la figure racine
i3.i2 : nom des instances sur le chemin
sig : nom de l'objet dans la cellule



UPMC / M2 SESI / VLSI2

Modélisation : vue structurelle

Formats de fichier pour la description de net-lists

- Les langages de description de matériel (**VHDL**, **Verilog**, **SystemC**) visent principalement la modélisation comportementale, mais permettent également de décrire des net-lists (description structurelles).
- De nombreux outils CAO, qui utilisent une description structurelle du circuit en entrée, ont défini leur propre format de net-list, qui sont devenus des standards de fait.
exemples : **format.spi** pour SPICE et ELD0 (simulateurs électriques)
format.def pour les outils CADENCE (placement/routage)
- Il existe des formats spécialement définis pour faciliter l'échange d'information entre différentes compagnies.
exemple : **format.edf**

Tous ces formats et langages de description de net-list permettent de représenter la même information : des schémas d'interconnexion hiérarchiques multi-niveaux...

UPMC / M2 SESI / VLSI2

Modélisation : vue structurelle

Outils de saisie de schéma

La plupart des chaînes de CAO commerciales proposent des **éditeurs graphiques interactifs** permettant la saisie de schéma :

Un avantage :

- bonne lisibilité de la documentation

Plusieurs inconvénients :

- complexité limitée
- difficile à maintenir
- non paramétrable

La chaîne Alliance fournit un **langage procédural** permettant la description de « net-lists ». Le langage Stratus est dérivé du langage interprété Python. Il est possible de générer différents formats de fichier à partir d'une description en langage Stratus.

UPMC / M2 SESI / VLSI2

Modélisation : vue structurelle

Bibliothèques de cellules

Le processus de raffinement du schéma est un processus descendant, mais sous contrainte : les composants terminaux font nécessairement partie d'une **bibliothèque de cellules prédefinies... et précaractérisées**.

Pour les **parties régulières** (telles que les chemins de données), le processus de raffinement du schéma peut être réalisé **manuellement**, en utilisant des **macro-cellules optimisées** (exemple : DP_SXLIB).

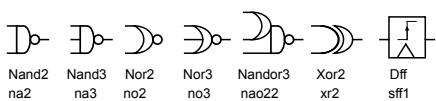
Pour les **parties moins régulières** (telle que la logique de contrôle), il est généralement préférable d'utiliser des **outils de synthèse** automatique (qui sont plus efficaces que le concepteur en termes d'optimisation, et surtout beaucoup plus rapides). Ces outils utilisent généralement des bibliothèques de **cellules de base** (exemple : SXLIB).

UPMC / M2 SESI / VLSI2

Modélisation : bibliothèque de cellules

Cellules de base

Les bibliothèques de cellules peuvent contenir des portes logiques élémentaires (exemple : SXLIB) :



Comme il existe généralement plusieurs puissances pour chaque fonction logique, ces bibliothèques peuvent contenir plusieurs centaines de cellules.

Elles contiennent - **pour chaque cellule** - les vues nécessaires aux outils CAO :

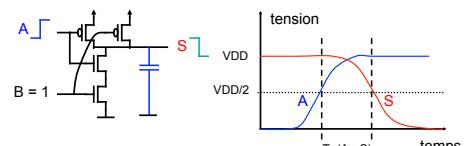
- dessin des masques
- schéma en transistors
- description(s) comportementale(s)

UPMC / M2 SESI / VLSI2

Modélisation : bibliothèque de cellules

Caractérisation temporelle

Les portes logiques sont caractérisées par des temps de propagation : un événement sur une entrée peut créer un événement sur la sortie après un temps $T_p(A \rightarrow S)$



Le temps de propagation dépend de la capacité de charge et de la puissance de la porte (dimensions des transistors).

UPMC / M2 SESI / VLSI2

Modélisation : bibliothèque de cellules

Cellules précaractérisées

- Le dessin des masques** des cellules respecte une série de contraintes topologiques (appelées **gabarit**), permettant l'automatisation du placement et du routage. Cette vue - de même que le schéma en transistors - n'est généralement pas accessible au concepteur.
- Les fichiers de caractérisation** définissent - pour chaque cellule - les informations utiles pour les outils de synthèse automatique et pour les outils de simulation.
 - fonction logique réalisée
 - temps de propagation
 - consommation électrique
 - capacités d'entrée
 - sortance (fan-out)
 - etc.

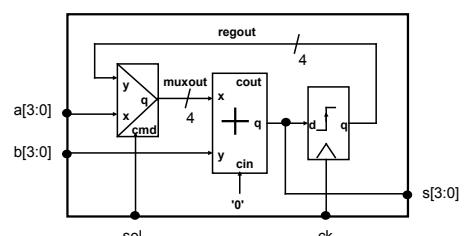
On appelle **Design Kit** l'ensemble des fichiers contenant ces informations de caractérisation d'une bibliothèque de cellule particulière pour une chaîne de CAO particulière.

UPMC / M2 SESI / VLSI2

Modélisation : bibliothèque de cellules

Décomposition « addaccu »

Le composant addaccu peut se décomposer en trois blocs :



UPMC / M2 SESI / VLSI2

Modélisation : exemple addaccu

Raffinement progressif du schéma

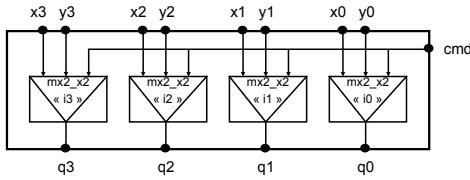
- Pour valider ce premier découpage structurel, on peut écrire des modèles comportementaux pour chacun des trois blocs mux4, adder4, et accu4... et re-simuler, en utilisant le même test-bench que celui qui a permis de valider le modèle comportemental du composant addaccu.
- L'étape suivante consiste à décomposer les 3 blocs mux4, adder4 et accu4, en définissant pour chaque bloc un schéma utilisant les cellules de base disponibles dans la bibliothèque de cellules précaractérisées. Il peut être utile de définir des sous-blocs intermédiaires.
- Le résultat final est un schéma hiérarchique multi-niveaux, qui possède une structure d'arbre, dont la racine est le composant « addaccu », et dont les feuilles sont des éléments de la bibliothèque de cellules.

UPMC / M2 SESI / VLSI2

Modélisation : exemple addaccu

Multiplexeur « mux4 » : structure interne

Puisqu'il existe une cellule multiplexeur 1 bit dans la bibliothèque SxLib, (cellule « mx2_x2 »), on peut décomposer le bloc « mux4 » en 4 cellules, identifiées par un « nom d'instance », qui doit être unique :

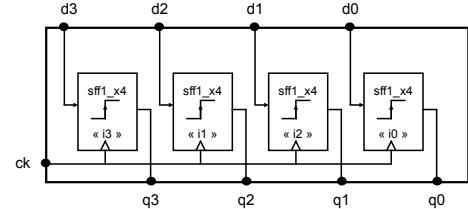


UPMC / M2 SESI / VLSI2

Modélisation : exemple addaccu

Accumulateur « accu4 » : structure interne

Puisqu'il existe une cellule bascule D dans la bibliothèque SxLib, (cellule « sff1_x4 »), on peut décomposer le registre 4 bits « accu4 » en 4 cellules identiques :

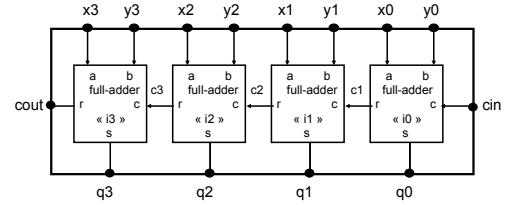


UPMC / M2 SESI / VLSI2

Modélisation : exemple addaccu

Additionneur « adder4 » : structure interne

Il existe de nombreux schémas de réalisation d'additionneurs. La plus simple est la structure « ripple adder » (additionneur à retenue propagée) qui utilise des additionneurs 1 bit (ou « full-adder ») :

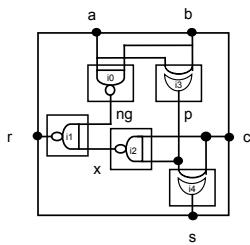


UPMC / M2 SESI / VLSI2

Modélisation : exemple addaccu

Schéma possible pour le full-adder

On peut réaliser un additionneur 1 bit (fa) en interconnectant 3 cellules na2_x1 (NAND à 2 entrées), et 2 portes xr2_x1 (XOR à 2 entrées) :



UPMC / M2 SESI / VLSI2

Modélisation : exemple addaccu

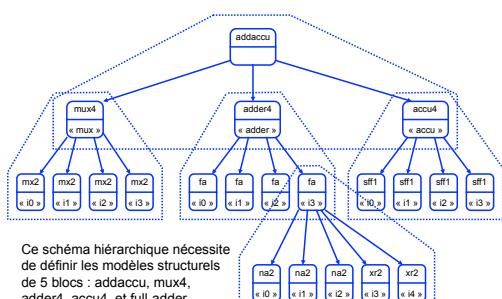
Modèle VHDL « addaccu.vst » / 1

```
page 1
entity addaccu is
-- liste des ports d'entrée/sortie
port(
    cmd : in std_logic_vector(3 downto 0);
    x : in std_logic_vector(3 downto 0);
    y : in std_logic_vector(3 downto 0);
    q : out std_logic_vector(3 downto 0);
    vss : in std_logic;
    vdd : in std_logic);
component mux4
port(cmd : in std_logic;
      x : in std_logic_vector(3 downto 0);
      y : in std_logic_vector(3 downto 0);
      q : out std_logic_vector(3 downto 0);
      vss : in std_logic;
      vdd : in std_logic);
end mux4;
component adder4
port(cin : in std_logic;
      x : in std_logic_vector(3 downto 0);
      y : in std_logic_vector(3 downto 0);
      q : out std_logic_vector(3 downto 0);
      cout : out std_logic;
      vss : in std_logic;
      vdd : in std_logic);
end adder4;
```

UPMC / M2 SESI / VLSI2

Modélisation : exemple addaccu

Arbre d'instanciation



UPMC / M2 SESI / VLSI2

Modélisation : exemple addaccu

Modèle VHDL « addaccu.vst » / 2

```
page 3
component accu4
port(ck : in std_logic;
      d : in std_logic_vector(3 downto 0);
      q : out std_logic_vector(3 downto 0);
      vss : in std_logic;
      vdd : in std_logic);
-- déclaration des signaux
signal muxout std_logic_vector(3 downto 0);
signal regout std_logic_vector(3 downto 0);
-- description de la net-list
begin
    mux : mux4 port map( cmd => sel,
                          x => a,
                          y => regout,
                          q => muxout,
                          vss => vss,
                          vdd => vdd );
    adder : adder4 port map( x => muxout,
                             y => b,
                             q => s,
                             cin => vss ,
                             vss => vss ,
                             vdd => vdd );
    accu : accu4 port map( ck => ck ,
                           d => s,
                           q => regout ,
                           vss => vss ,
                           vdd => vdd );
end vst;
```

UPMC / M2 SESI / VLSI2

Modélisation : exemple addaccu

page 2

```
component mux4
port(cmd : in std_logic;
      x : in std_logic_vector(3 downto 0);
      y : in std_logic_vector(3 downto 0);
      q : out std_logic_vector(3 downto 0);
      vss : in std_logic;
      vdd : in std_logic);
end mux4;
component adder4
port(cin : in std_logic;
      x : in std_logic_vector(3 downto 0);
      y : in std_logic_vector(3 downto 0);
      q : out std_logic_vector(3 downto 0);
      cout : out std_logic;
      vss : in std_logic;
      vdd : in std_logic);
end adder4;
```

Modélisation : exemple addaccu

page 4

Automate d'états finis synchrones

UPMC / M2 SESI / VLSI2

Automate d'états finis synchrones

La théorie des automates est une méthode de représentation extrêmement générale du comportement d'un système matériel numérique synchrones.

Le comportement de n'importe quel système synchrone, (simple compteur 4 bits, ou microprocesseur 32 bits complet) peut - en principe - être représenté par ce modèle.

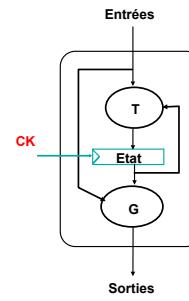
Le comportement d'un système complexe est souvent décrit en interconnectant des automates plus simples.

Il existe une méthode systématique permettant de construire le schéma en portes logiques réalisant le comportement défini par un automate abstrait.

UPMC / M2 SESI / VLSI2

automate d'états finis synchrones

Principe Général



Tout système numérique synchrone peut être modélisé par un automate :

Fonction de transition :

$$\text{NextEtat} \leqslant T(\text{Etat}, \text{Entrées})$$

Fonction de génération :

$$\text{Sorties} \leqslant G(\text{Etat}, \text{Entrées})$$

UPMC / M2 SESI / VLSI2

automate d'états finis synchrones

Représentation abstraite

- Dans la définition générale d'un automate, les fonctions de transition et de génération sont définies pour :
 - un ensemble de valeurs symboliques pour les entrées
 - un ensemble de valeurs symboliques pour les sorties
 - un ensemble de valeurs symboliques pour les états

... mais le code binaire associé à chacune de ces valeurs n'est pas défini.

- En pratique, les signaux d'entrée et de sortie sont très souvent définis au niveau Booléen : n bits d'entrée correspondent à 2^n valeurs possibles pour les entrées. Idem pour les sorties.

Seules les valeurs stockées dans les registres ne sont pas définies au niveau Booléen : le codage des états n'est pas explicite.

UPMC / M2 SESI / VLSI2

automate d'états finis synchrones

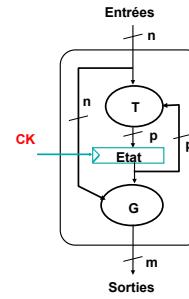
Initialisation

- Puisqu'un automate contient un état interne, il faut définir un état initial pour obtenir un comportement totalement déterministe.
- Les automates synchrones possèdent très souvent un signal d'entrée particulier (signal NRESET), qui n'agit que sur la fonction de transition : lorsque le signal NRESET est actif (état bas), on force la valeur de l'état initial dans le registre d'état lors du prochain front du signal d'horloge...
 - quel que soit l'état actuel de l'automate
 - quelle que soit la valeur des autres entrées.

UPMC / M2 SESI / VLSI2

automate d'états finis synchrones

Fonctions de génération et de transition



Si on a :

- n bits d'entrée E_i
- m bits de sortie S_j
- p bits mémorisés R_k

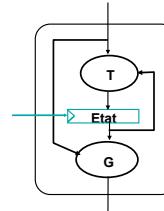
Il faut définir :

- p fonctions booléennes dépendant de (n+p) variables pour la transition
 $NR_k = T_k(E_i, R_k)$
- m fonctions booléennes dépendant de (n+p) variables ou la génération
 $S_j = T_j(E_i, R_k)$

UPMC / M2 SESI / VLSI2

automate d'états finis synchrones

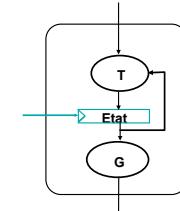
Automates de Moore et de Mealy



Cas général :

Automate de Mealy

UPMC / M2 SESI / VLSI2



Cas particulier : La fonction de génération ne dépend que de l'état !

Automate de Moore

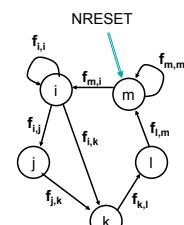
automate d'états finis synchrones

Représentation graphique d'un automate

- Les nœuds (i) représentent les états
- Les arcs (i,j) représentent les transitions

Chaque arc (i,j) est étiqueté par une expression Booléenne f_{i,j} ne dépendant que des signaux d'entrée, et définissant la condition de transition.

Dans le cas d'un automate de Moore, les signaux de sortie ne dépendent que de l'état : chaque nœud est donc étiqueté par la valeur des signaux de sortie.



UPMC / M2 SESI / VLSI2

automate d'états finis synchrones

Automate déterministe

Pour qu'un automate possède un comportement déterministe, il faut respecter les conditions suivantes :

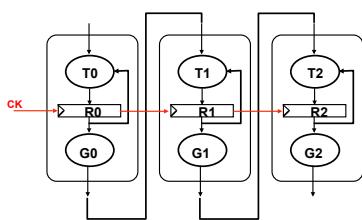
- Existence d'un mécanisme matériel d'**initialisation** permettant de forcer l'automate dans un état initial connu.
 - Condition d'**orthogonalité** : pour tout état i , et pour toute configuration des entrées, il y a un seul état successeur de l'état i .
- Pour tout état i , $\sum_j f_{i,j} = 1$
- Condition de **complétude** : pour toute configuration des entrées, il y a toujours un état successeur de l'état i .
- Pour tout état i , $\sum_j f_{i,j} = 1$

UPMC / M2 SESI / VLSI2

automate d'états finis synchrones

Automates communicants

Un système numérique synchronisé est souvent conçu comme un ensemble d'automates « simples » fonctionnant en parallèle, et communiquant entre eux : les entrées d'un automate sont les sorties d'un autre automate.

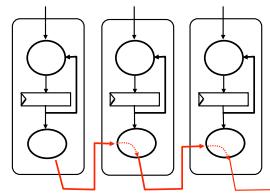


UPMC / M2 SESI / VLSI2

automate d'états finis synchrones

Inconvénient des Automates de mealy

Dans un automate de mealy, Il existe une dépendance combinatoire entre les entrées et les sorties !



ceci introduit des chaînes longues traversant plusieurs composants.

Il devient impossible de caractériser le comportement temporel de chaque automate indépendamment des autres.

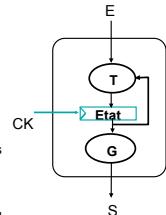
UPMC / M2 SESI / VLSI2

automate d'états finis synchrones

Exemple 1 : Détection d'une séquence

Problème à résoudre :

- On cherche à réaliser un dispositif matériel possédant un seul bit d'entrée E et une seul bit de sortie S .
- L'entrée E est échantillonnée à chaque cycle.
- La sortie S passe à 1 pendant un cycle chaque fois que l'entrée E a la valeur 1 pendant 3 cycles consécutifs.



Remarque: Il n'y a pas d'entrée de type RESET !

UPMC / M2 SESI / VLSI2

automate d'états finis synchrones : exemple 1

1) représentation graphique

On devine un état initial :



Ici l'état dans lequel l'automate n'a pas vu E à la valeur 1 depuis longtemps

Signification des états :

- Z : le dernier bit reçu valait 0

UPMC / M2 SESI / VLSI2

automate d'états finis synchrones : exemple 1

1) représentation graphique

Signification des états :

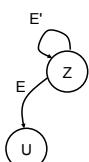
- Z : le dernier bit reçu valait 0
- U : les 2 derniers bits reçus valaient 0,1
- D : les 3 derniers bits reçus valaient 0,1,1

UPMC / M2 SESI / VLSI2

automate d'états finis synchrones : exemple 1

1) représentation graphique

notation : $E' \Leftrightarrow \neg E$



Signification des états :

- Z : le dernier bit reçu valait 0
- U : les 2 derniers bits reçus valaient 0,1

UPMC / M2 SESI / VLSI2

automate d'états finis synchrones : exemple 1

1) représentation graphique

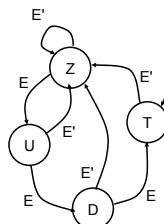
Signification des états :

- Z : le dernier bit reçu valait 0
- U : les 2 derniers bits reçus valaient 0,1
- D : les 3 derniers bits reçus valaient 0,1,1
- T : les 3 derniers bits reçus valaient 1,1,1

UPMC / M2 SESI / VLSI2

automate d'états finis synchrones : exemple 1

1) représentation graphique



Signification des états :

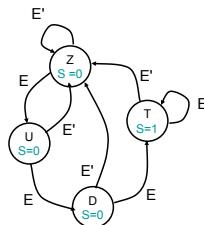
- Z : le dernier bit reçu valait 0
- U : les 2 derniers bits reçus valaient 0,1
- D : les 3 derniers bits reçus valaient 0,1,1
- T : les 3 derniers bits reçus valaient 1,1,1

On reboucle sur des états connus → fin

UPMC / M2 SESI / VLSI2

automate d'états finis synchrones : exemple 1

1) représentation graphique



La valeur de la sortie S ne dépend que de l'état.

UPMC / M2 SESI / VLSI2

automate d'états finis synchrones : exemple 1

2) encodage des états

Etat	X1	X0	Y0	Y1	Y2	Y3
« Z »	0	0	1	0	0	0
« U »	0	1	0	1	0	0
« D »	1	0	0	0	1	0
« T »	1	1	0	0	0	1

code binaire
(sur 2 bits)

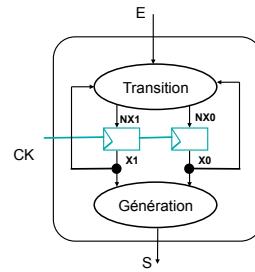
code « one hot »
(un bit par état)

Il y a plusieurs types de code possibles, et pour chaque type on peut avoir un très grand nombre d'encodages.
Dans la suite de l'exemple, on utilisera le code binaire ci-dessus.

UPMC / M2 SESI / VLSI2

automate d'états finis synchrones : exemple 1

2) Encodage des états



Deux fonctions Booléennes pour la fonction de transition :

$$NX0 = T0(X1, X0, E)$$

$$NX1 = T1(X1, X0, E)$$

Une fonction Booléenne pour la fonction de génération :

$$S = G(X1, X0)$$

UPMC / M2 SESI / VLSI2

automate d'états finis synchrones : exemple 1

3) table de vérité

X1	X0	E	NX1	NX0	S
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1	1	1	1

Variables d'entrée

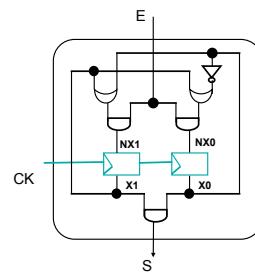
Fonction de Transition

Fonction de Génération

UPMC / M2 SESI / VLSI2

automate d'états finis synchrones : exemple 1

5) Synthèse logique



On traduit les expressions Booléennes représentant la fonction de transition et la fonction de génération en un schéma en portes.

Pour cet exemple, il faut 3 portes « and », 2 portes « or », un inverseur et 2 bascules « D ».

UPMC / M2 SESI / VLSI2

automate d'états finis synchrones : exemple 1

4) Simplification des expressions

Fonction de transition :

$$NX1 = E \cdot (X1 + X0)$$

$$NX0 = E \cdot (X1 + X0')$$

Fonction de génération :

$$S = X1 \cdot X0$$

UPMC / M2 SESI / VLSI2

automate d'états finis synchrones : exemple 1

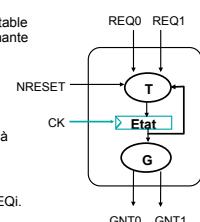
On cherche à réaliser un allocateur de bus équitable entre 2 utilisateurs (respectant une priorité tournante ou « round robin »).

Le signal NRESET initialise l'automate dans un état où le bus n'est pas alloué.

Les deux requêtes REQ0 et REQ1 sont actives à l'état haut, et indépendantes.

L'utilisateur possédant le bus signale la fin de l'utilisation en forçant la valeur 0 sur le signal REQi.

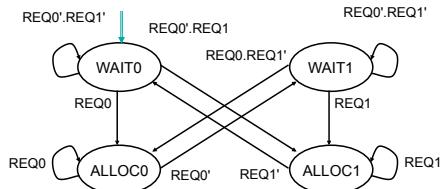
Les deux signaux GNT0 et GNT1 ne peuvent être actifs en même temps, et il y a toujours un cycle non alloué ($GNT0 = GNT1 = 0$) entre deux allocations



UPMC / M2 SESI / VLSI2

automate d'états finis synchrones : exemple allocateur

représentation graphique



Signification des états :

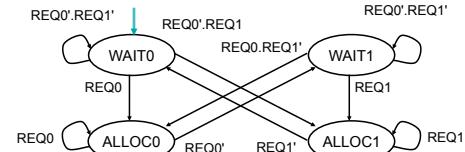
- WAIT0 : bus non alloué / utilisateur 0 prioritaire
- WAIT1 : bus non alloué / utilisateur 1 prioritaire
- ALLOC0 : bus alloué à l'utilisateur 0
- ALLOC1 : bus alloué à l'utilisateur 1

UPMC / M2 SESI / VLSI2

automate d'états finis synchrones : exemple allocateur

A : codage one-hot /1

- En codage one-hot le registre d'état a autant de bits qu'il y a d'états, un bit par état.
- Définir la fonction de transition consiste à définir l'expression de chaque état.
- Le codage étant connu, il est possible de décrire l'automate directement en vbe



UPMC / M2 SESI / VLSI2

automate d'états finis synchrones : exemple allocateur

Trois méthodes

A : One Hot
1 bit par état

B : Codage manuel
Le choix du codage est fait par le concepteur

C : Codage automatique
Le choix du codage est réalisé par un algorithme

UPMC / M2 SESI / VLSI2

automate d'états finis synchrones : exemple allocateur

A : codage one-hot /2

```
entity allocateur is
    -- liste des ports d'entrée-sortie
    port ( ck      : in std_logic;
           req0   : in std_logic;
           req1   : in std_logic;
           nreset : in std_logic;
           gnt0   : out std_logic;
           gnt1   : out std_logic );
    end allocateur;

    -- fonction de transition
    process (ck) begin
        if (ck and ck'event) then
            if (nreset = '0') then
                r_wait0 <= '1'; r_wait1 <= '0';
                r_alloc0 <= '0'; r_alloc1 <= '0';
            else
                r_wait0 <= s_wait0; r_wait1 <= s_wait1;
                r_alloc0 <= s_alloc0; r_alloc1 <= s_alloc1;
            end if;
        end if;
    end process reg;
```

UPMC / M2 SESI / VLSI2

automate d'états finis synchrones : exemple allocateur

B : encodage des états

Etat	X1	X0
« WAIT0 »	0	0
« WAIT1 »	0	1
« ALLOC0 »	1	0
« ALLOC1 »	1	1

codage binaire
(sur 2 bits)

UPMC / M2 SESI / VLSI2

automate d'états finis synchrones : exemple allocateur

B : simplification des expressions

Sans le NRESET:

```
NX1 = X1'.REQ1 + S1'.REQ0 + X0.REQ1 + X0'.REQ0
NX0 = REQ0'.REQ1 + X0.REQ1 + X1'.X0.REQ0' + X1.X0'.REQ0'
GNT0 = X1.X0
GNT1 = X1.X0'
```

Avec le NRESET:

```
NX1 = NRESET.(X1'.REQ1 + S1'.REQ0 + X0.REQ1 + X0'.REQ0)
NX0 = NRESET.(REQ0'.REQ1 + X0.REQ1 + X1'.X0.REQ0' + X1.X0'.REQ0')
GNT0 = X1.X0
GNT1 = X1.X0'
```

UPMC / M2 SESI / VLSI2

automate d'états finis synchrones : exemple allocateur

B : table de vérité (sans NRESET)

	X1	X0	REQ0	REQ1	NX1	NX0	GNT1	GNT0
WAIT0	0	0	0	0	0	0	0	0
	0	0	0	1	1	1	0	0
	0	0	1	0	1	0	0	0
	0	0	1	1	1	0	0	0
	0	1	0	0	0	1	0	0
	0	1	0	1	1	1	0	0
	0	1	1	0	1	0	0	0
	0	1	1	1	1	1	0	0
WAIT1	1	0	0	0	0	1	0	1
	1	0	0	1	0	1	0	1
	1	0	1	0	1	0	0	1
	1	0	1	1	1	0	0	1
	1	1	0	0	0	0	1	0
	1	1	0	1	1	1	1	0
	1	1	1	0	0	0	1	0
	1	1	1	1	1	1	1	0
ALLOC0	1	0	1	0	0	1	0	1
	1	0	1	1	1	0	0	1
	1	0	1	0	1	0	0	1
	1	0	1	1	1	0	0	1
	1	1	0	1	1	1	1	0
	1	1	0	1	1	1	1	0
ALLOC1	1	1	1	1	1	1	1	0

UPMC / M2 SESI / VLSI2

automate d'états finis synchrones : exemple allocateur

C : Modèle VHDL allocateur : architecture fsm

```
entity allocateur is
    -- liste des ports d'entrée-sortie
    port ( ck      : in std_logic;
           req0   : in std_logic;
           req1   : in std_logic;
           nreset : in std_logic;
           gnt0   : out std_logic;
           gnt1   : out std_logic );
    end allocateur;

    -- définition du type enuméré
    type etat_type is (wait0, wait1, alloc0, alloc1);

    -- déclaration des signaux
    signal present, futur : etat_type;
    signal futur_nxt : etat_type;
    -- directives pour la synthèse :
    -- pragma current_state present
    -- pragma next_state futur
    -- pragma clock ck
    begin
        -- processus « update »
        process ( ck ) begin
            if (ck and ck'event) then
                if (nreset = '0') then
                    present <= alloc0;
                    futur_nxt <= wait0;
                else
                    case present is
                        when wait0 => if (req0 = '1') then futur_nxt <= alloc0;
                                         else futur_nxt <= valid0;
                                         end if;
                        when wait1 => if (req1 = '1') then futur_nxt <= alloc1;
                                         else futur_nxt <= valid1;
                                         end if;
                        when alloc0 => if ((req0 = '1') or (req1 = '1')) then futur_nxt <= wait1;
                                         else futur_nxt <= alloc0;
                                         end if;
                        when alloc1 => if ((req0 = '1') or (req1 = '1')) then futur_nxt <= wait0;
                                         else futur_nxt <= alloc1;
                                         end if;
                    end case;
                end if;
            end if;
        end process;
    end process;

    -- fonction de génération
    if (present = alloc0) then gnt0 <= '1'; else gnt0 <= '0';
    end if;
    if (present = alloc1) then gnt1 <= '1'; else gnt1 <= '0';
    end if;
end process;
end fsm;
```

UPMC / M2 SESI / VLSI2

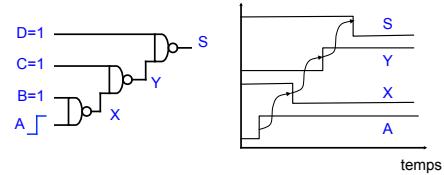
automate d'états finis synchrones : exemple allocateur

Circuits numériques synchrones

UPMC / M2 SESI / VLSI2

Propagation des événements

Un opérateur combinatoire est une structure **orientée** :
Les événements se propagent des entrées vers les sorties !

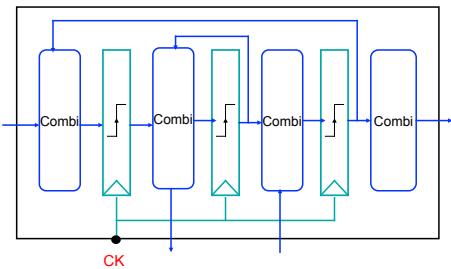


=> Il ne doit pas y avoir de boucle dans un bloc combinatoire

UPMC / M2 SESI / VLSI2

circuits numériques synchrones

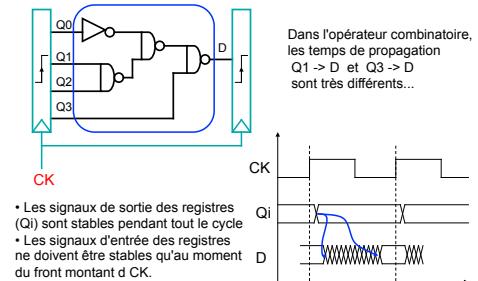
Circuits numériques synchrones



UPMC / M2 SESI / VLSI2

circuits numériques synchrones

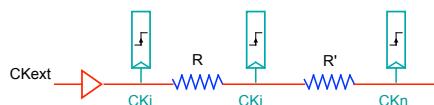
Les temps de propagation



UPMC / M2 SESI / VLSI2

circuits numériques synchrones

Le skew



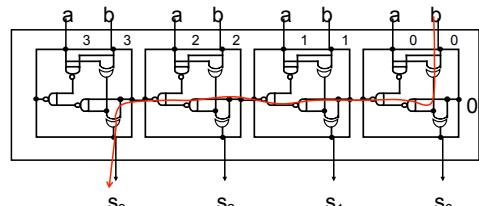
Le signal de synchronisation CK est une abstraction.
Le réseau physique de distribution du signal CK n'est pas parfait : les résistances et capacités intrinsèques des fils, ainsi que les amplificateurs intermédiaires introduisent des déphasages entre les signaux Cki qui parviennent aux registres.

Définition : le skew est un majorant de la valeur absolue du déphasage entre deux signaux d'horloge Cki et Ckj

UPMC / M2 SESI / VLSI2

circuits numériques synchrones

Chaîne longue additionneur 4 bits

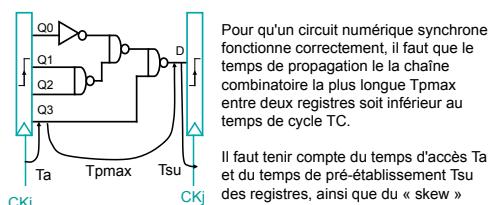


La chaîne longue traverse 6 portes « nand » et deux portes « xor »

UPMC / M2 SESI / VLSI2

circuits numériques synchrones

Chaînes longues



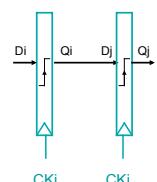
Tout opérateur combinatoire doit respecter la condition suivante :

$$Ta(CK \rightarrow Q) + Tpmax(Q1 \rightarrow D) + Tsu(D \rightarrow CKj) < TC - skew$$

UPMC / M2 SESI / VLSI2

circuits numériques synchrones

Chaînes courtes



Un dysfonctionnement de type « chaîne courte » se produit lorsque le temps de propagation minimal entre deux registres est plus court que le déphasage entre les deux signaux d'horloge (skew).

Il faut ici également tenir compte du temps d'accès Ta du registre source et du temps de maintien Th du registre destination.

Tout opérateur combinatoire doit respecter la condition suivante :

$$Ta(CK \rightarrow Q) + Tpmin(Q \rightarrow D) > Th(D \rightarrow CKj) + skew$$

UPMC / M2 SESI / VLSI2

circuits numériques synchrones

Le triple rôle des registres

Les registres ont une triple fonction :

- **Mémorisation** : stocker une valeur qui sera utilisée plus tard. Ceci impose que l'écriture dans les registres soit conditionnelle.
- **Synchronisation** : assurer que toutes les données d'un même opérateur combinatoire sont simultanément disponibles.
- **Stabilisation** : garantir que les signaux d'entrée des opérateurs combinatoires sont stables pendant toute la durée du cycle.

UPMC / M2 SESI / VLSI2

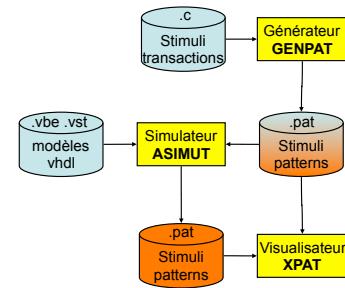
circuits numériques synchrones

Chaîne de simulation

UPMC / M2 SESI / VLSI2

simulation

Chaine de simulation



UPMC / M2 SESI / VLSI2

simulation

format PAT

- Le format de fichier .pat utilisé par Asimut permet de décrire les patterns appliqués et les résultats attendus.
- Le format .pat contient :
 - L'interface du circuit
 - La séquence des patterns
 - Les actions sur le simulateur
- Documentation : [man 5 pat](#)

UPMC / M2 SESI / VLSI2

simulation

Exemple de .pat

```

in      A (0 to 15) X;
in      B (0 to 15) X;
in      Cin;
out     Cout;
signal  S (0 to 15) X;
register Accu.A (0 to 15) X;

begin
  < 0 ns > pattern_0 : F0F0 0A0A 1 ?0 ?FAFA ?6DE7;
  < +10 ns > pattern_1 : OF0F F6F0 0 + **** ?54FC;
end;
  
```

UPMC / M2 SESI / VLSI2

simulation

Séquence de patterns

```

begin
  [< date >] [étiquette] : valeurs_des_signaux : []
  ...
end;

• date :: [+]
  • unité :: ps | ns | us | ms
  • [+] :: délai depuis le précédent pattern

• étiquette :: sert à s'y retrouver dans la séquence de patterns

• valeurs_des_signaux :: 
  entrées = 0 | 1 | + | - | 00110 | 0256 | DEAD
  sorties prédites = ?0 | ?1 | ?+ | ?- | ?00110 | ?0256 | ?DEAD
  sorties non prédites = " | ***

• [] :: permet d'ajouter une ligne blanche dans le fichier produit
  
```

UPMC / M2 SESI / VLSI2

simulation

Interface .pat

mode	nom de l'entrée-sortie	format	[spy]	[:]
• in	• nom	• B		
• out	• group (nom1, nom2, ...)	• X		
• inout	• chemi-nom pour les signaux	• O		
• signal	ou registres internes	• rien		
• register	.vbe : root.nom			
	.vst : nom			
		instance.nom		

Attention

Exemple

```

in      A (0 to 15) X;
in      B (0 to 15) X;
in      Cin ::;
out     Cout;
signal  S (0 to 15) X;
register Accu.A (0 to 15) X;
  
```

Les noms de vecteurs de signaux ont le format VHDL

UPMC / M2 SESI / VLSI2

simulation

Actions sur le simulateur

- Initialisation d'un registre
 - il est possible d'initialiser à tout moment un registre à une valeur quelconque, au format vhdl : registre <= value;
- sauvegarde de l'état des signaux
 - placé entre deux patterns : save;
- commentaires du fichier
 - précédés par -- Ils sont simplement ignorés
 - précédés par # ils sont recopiés intacts dans le fichier de sortie.

UPMC / M2 SESI / VLSI2

simulation

Le simulateur ASIMUT

Documentation : `man asimut`
`asimut [options] [root_file] [pattern_file] [result_file]`

options essentielles

- b si le fichier à simuler est uniquement comportemental (.vbe)
- c asimut fait seulement une lecture du modèle
- i val initialise tous les signaux à val (0 ou 1)
- zd force la simulation sans délai

autres options

- f file initialise tous les signaux à partir du fichier file (sauvé par la commande save)
- inspect inst_name produit un fichier de pattern avec les signaux à l'interface de inst_name.
- core file produit un fichier .cor avec l'état de tous les signaux dès la première erreur.

UPMC / M2 SESI / VLSI2

simulation

fichier CATAL

- Le fichier CATAL permet d'indiquer quelles sont les feuilles de l'arborescence dans le cas de la simulation d'une netlist.
 - Les noms présents dans le fichier ont un modèle comportemental.
 - format
- ```
bloc1 C
bloc2 C
bloc3 C
```

UPMC / M2 SESI / VLSI2

simulation

## variables d'environnement

- MBK\_CATA\_LIB répertoires contenant les descriptions et les patterns
- MBK\_WORK\_LIB répertoire de travail avec les descriptions et les patterns et où sont écrits les fichiers produits
- MBK\_CATAL\_NAME nom du fichier catalogue (placé dans MBK\_WORK\_LIB)
- MBK\_IN\_LO extension (type) des fichiers netlist (al ou vst)
- VH\_MAXERR nombre maximum d'erreurs autorisées avant l'arrêt de la simulation

UPMC / M2 SESI / VLSI2

simulation

## Langage GENPAT

- Documentation : `man genpat`
  - Le but est d'exprimer dans un langage procédural, les transactions sur les signaux.
  - C'est une bibliothèque de fonctions C:
    - pour définir l'interface
    - et les transactions
    - pour générer un fichier de patterns au format .pat
  - Le langage C permet l'écriture de boucles et de fonctions
  - Un script permet de lancer le compilateur C puis l'exécution du programme :
- ```
> genpat [-v] [-k] file
```

UPMC / M2 SESI / VLSI2

simulation

API Genpat (essentiel)

- DEF_GENPAT("nom") définit le nom de fichier dans lequel les patterns seront placés.
- SAV_GENPAT() sauve le fichier sur disque.
- DECLAR("ident","nb_space","format","mode","size","option") déclare le nom du signal

ident	nom du signal
nb_space	nombre d'espaces entre les colonnes,
format	format d'affichage (B,O,X),
type	IN, OUT, INOUT, SIGNAL, REGISTER
size	rien, X to Y, Y downto X
option	rien, S
- AFFECT("pattern_date","ident","value") définit une transaction sur le signal

pattern_date	date absolue de la transaction, ou "+nombre" date relative au dernier AFFECT() ou INIT()
ident	nom du signal
value	nombre dans la base définie par format

UPMC / M2 SESI / VLSI2

simulation

API Genpat : Exemple

```
#include <stdio.h>
#include "genpat.h"

fonction transformant
un entier en
chaîne de caractères
{
    char *itoa(int entier) {
        char *str = malloc (32);
        sprintf(str, "%d",entier);
        return(str);
    }

fichier vecteurs.pat
main () {
    int i, cur_vect = 0;
    DEF_GENPAT("vecteurs");
}

déclaration de l'interface
et des signaux internes
{
    DECLAR ("a", "2", "X", IN, "0 to 3");
    DECLAR ("b", "2", "X", IN, "0 to 3");
    DECLAR ("s", "2", "X", OUT, "0 to 3");

    for (i=0; i<16; i++) {
        for (j=0; j<16; j++) {
            AFFECT (itoa(cur_vect), "a", itoa(i));
            AFFECT (itoa(cur_vect), "b", itoa(j));
            cur_vect++;
        }
    }
    SAV_GENPAT ();
}
```

UPMC / M2 SESI / VLSI2

simulation

API Genpat (complément)

- INIT ("pattern_date", "ident", "value") identique à AFFECT mais pour initialiser un registre
- SETTUNIT("unit") définit l'unité de temps utilisée pour les dates : "ps", "ns", "ms", "us".
- LABEL("ident") affecte une étiquette pour le pattern courant (donc celui qui vient d'être affecté)
- ARRAY ("ident", "...", "nb_spa", "fmt", "mode", "option", "ident_group", 0)
 - déclare un groupe de signaux (aggrégat) de même type.
 - ident, ... noms des signaux composant le groupe (MSB en premier)
 - nb_spa nombre d'espaces entre les colonnes,
 - fmt format d'affichage (B,O,X),
 - type IN, OUT, INOUT, SIGNAL, REGISTER
 - option rien, S
 - ident_group nom du groupe

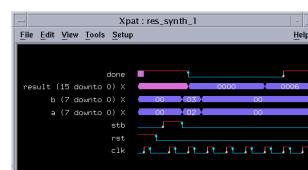
UPMC / M2 SESI / VLSI2

simulation

Le visualisateur de patterns XPAT

XPAT est l'un des nombreux outils de visualisation
 - avantage : fonctionnement simple et intuitif.
 - inconvénient : peu de fonctions

xpat [-l file]



UPMC / M2 SESI / VLSI2

visualiseurs

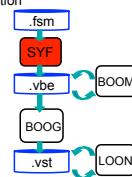
Chaîne de synthèse

UPMC / M2 SESI / VLSI2

La synthèse d'automate

Les outils de synthèse d'automate (exemple SYF) appliquent la méthode générale vue précédemment :

- Construction du graphe représentant l'automate abstrait
- Choix d'un codage pour les états
- Construction des fonctions de transition et de génération
- Simplification des expressions Booléennes
- Génération du réseau Booléen



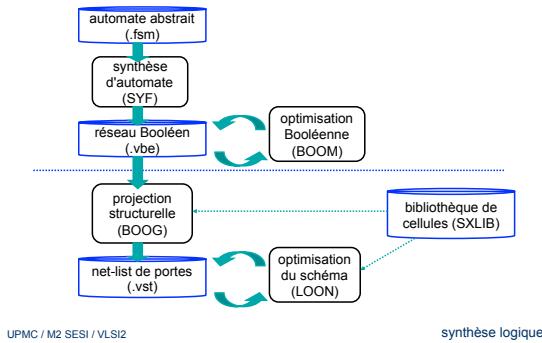
La principale intervention du concepteur porte sur le choix d'un type de codage.

Contrairement à l'intuition, le codage « one-hot » donne très souvent de bons résultats !

UPMC / M2 SESI / VLSI2

synthèse logique

Les étapes de la synthèse



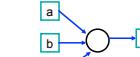
UPMC / M2 SESI / VLSI2

synthèse logique

Optimisation Booléenne / a

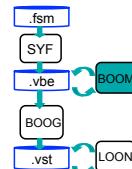
Les outils d'optimisation Booléenne (exemple : BOOM) cherchent à « simplifier » le réseau Booléen. Cette optimisation étant indépendante du procédé de fabrication choisi, la fonction de coût est le « nombre de littéraux ».

L'optimisation locale vise la simplification de l'expression Booléenne associée à un noeud particulier du réseau Booléen.



forme canonique : 12 littéraux
 $r \leq a.b.c + a'.b'.c + a'.b.c + a.b.c$

forme optimisée : 6 littéraux
 $r \leq a.b + a.c + b.c$



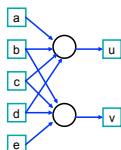
UPMC / M2 SESI / VLSI2

synthèse logique

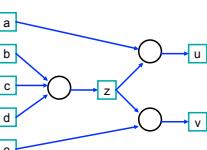
Optimisation Booléenne / b

L'optimisation globale utilise des techniques de factorisation, qui peuvent modifier la structure du réseau Booléen :

Forme initiale :
 $u \leq a.(b.c + b'.d)$
 $v \leq (b.c + b'.d) + e$



Forme factorisée :
 $z \leq b.c + b'.d$
 $u \leq a.z$
 $v \leq z + e$



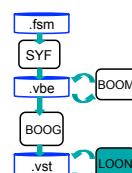
synthèse logique

UPMC / M2 SESI / VLSI2

Optimisation du schéma / a

Les outils d'optimisation de schéma (exemple: LOON) visent principalement l'optimisation des performances temporelles. Ils cherchent à réduire les temps de propagation sur les chemins critiques du bloc synthétisé.

- Les deux principales techniques sont
 - l'utilisation de portes de puissance
 - l'insertion de buffers
- Pour optimiser les performances temporelles sans trop augmenter la surface totale du bloc, seules les portes logiques et les signaux se trouvant sur un chemin critique doivent être modifiés.
- L'analyse des chemin critique dépend des temps d'arrivées des signaux sur les ports d'entrée, et des temps requis sur les ports de sortie



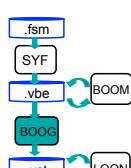
UPMC / M2 SESI / VLSI2

synthèse logique

Projection structurelle

Les outils de projection structurelle (exemple : BOOG) transforment une expression Booléenne associée à un noeud du réseau Booléen en un schéma en portes logiques.

- Ces outils s'appuient sur une bibliothèque de cellules précaractérisées.
- Le traitement est local : chaque expression Booléenne est traitée indépendamment.
- Ils utilisent des techniques de reconnaissance de forme pour reconnaître des sous-expressions.
- Ils exploitent les informations de caractérisation associées aux cellules pour optimiser
 - la surface totale du bloc synthétisé
 - les performances temporelles

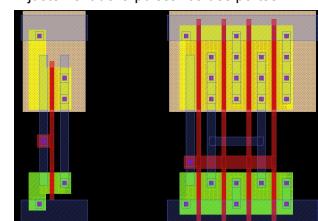


synthèse logique

UPMC / M2 SESI / VLSI2

Optimisation du schéma / b

Ajustement de la puissance des portes :



inv_x1 : WN = 5 / WP = 10

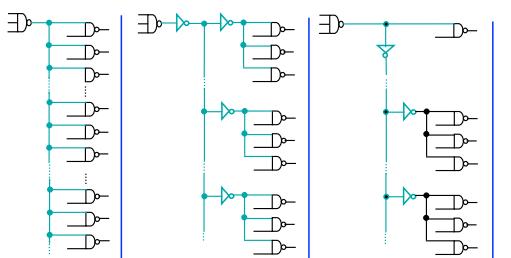
inv_x8 : WN = 40 / WP = 80

synthèse logique

UPMC / M2 SESI / VLSI2

Optimisation du schéma / c

Insertion d'arbres de buffers (en cas de fanout très grand)

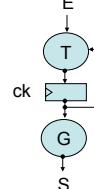


UPMC / M2 SESI / VLSI2

synthèse logique

syf

syf prend une machine d'états finis décrite en vhdl, choisit un codage et produit un modèle au format vbe.

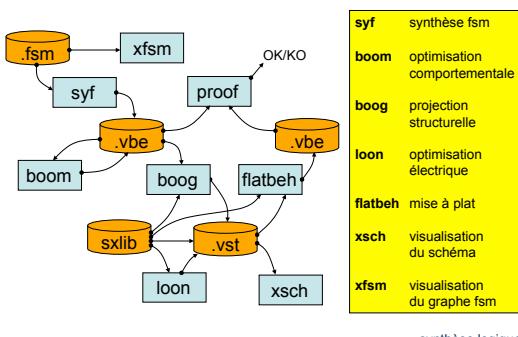


- principe de fonctionnement
- le comportement est décrit en vhdl
 - un process **Transition** définissant l'état futur en fonction de l'état courant et des entrées
 - un process **Génération** définissant les sorties en fonction de l'état courant
 - **syf** choisit un codage (plusieurs algos)
 - **syf** détermine les expressions de tous les bits du registre d'état et tous les bits de sorties

UPMC / M2 SESI / VLSI2

synthèse logique

Flot des outils de synthèse Alliance

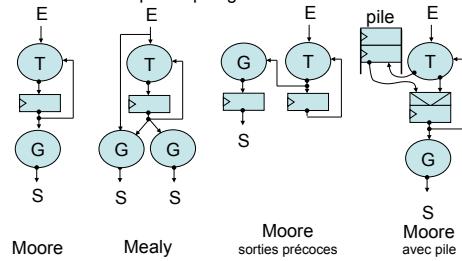


UPMC / M2 SESI / VLSI2

synthèse logique

syf

Quelques topologies d'automates.

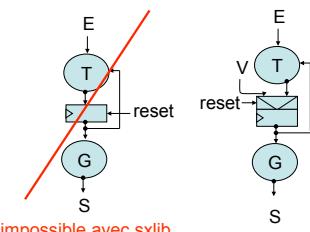


UPMC / M2 SESI / VLSI2

synthèse logique

syf

L'initialisation peut être ou ne pas être synchronisée

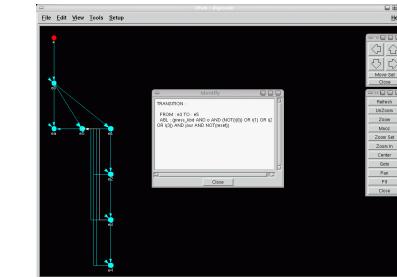


UPMC / M2 SESI / VLSI2

synthèse logique

x fsm

x fsm représente le graphe d'états d'un automate.



UPMC / M2 SESI / VLSI2

visualiseurs

syf

Codage d'un automate d'un FSM

syf -j[a|m|o|u|r [-CDEOPRSTV] input [output]

paramètres:

- j : trois algos de codage (asp, jedi, mustang)
- o : codage one-hot
- u : codage utilisateur dans le fichier input.enc
- r : liste des couples: noms_d'état code_hexa
- C : vérifie la complétude et l'orthogonalité
- E : sauve l'encodage (syntaxe de -u)
- P : ajoute un scanpath
- R : utilise une ROM et un micro séquenceur
- V : verbose mode

environnement

MBK_WORK_LIB: répertoire de sortie

UPMC / M2 SESI / VLSI2

synthèse logique

lax

Fichier de paramétrisation des outils de synthèse.

Les informations présentes sont

- le type d'optimisation général (délai ou surface)
- le degré d'optimisation (plus ou moins d'effort)
- le retard en ns de certaines entrées
- les sorties qu'il faut optimiser en priorité
- les signaux auxiliaires à conserver
- les charges capacitives sur les entrées et les sorties (en FF)
- les impédances des entrées (en ohms)
- le nombre de buffers à ajouter sur certaines entrées

UPMC / M2 SESI / VLSI2

synthèse logique

