MOCCA

outils de CAO
Alliance / Coriolis

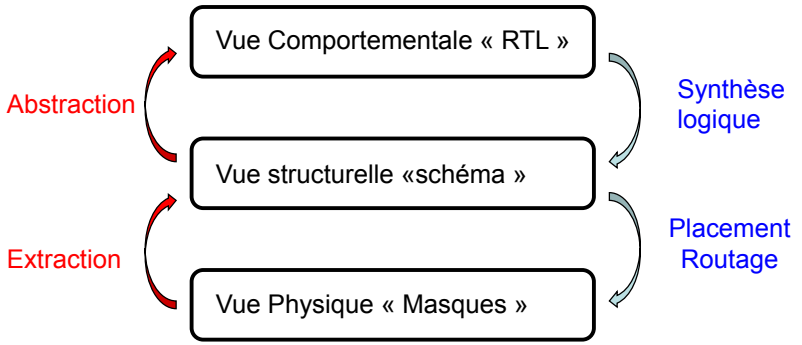
Plan

- 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

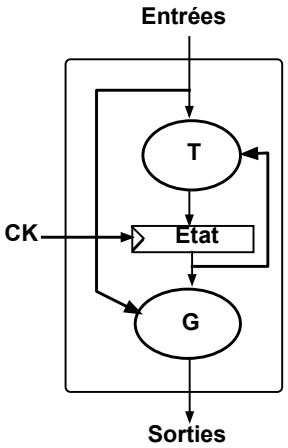
Les trois vues d'un circuit intégré

- 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.

Les trois vues



La vue comportementale RTL



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

Fonction de transition :
NextEtat <= T(Etat, Entrées)

Fonction de génération :
Sorties <= G(Etat, Entrées)

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

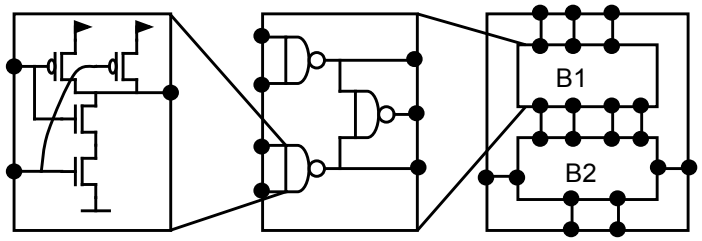
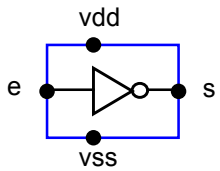


Schéma « transistors »

Schéma « portes logiques »

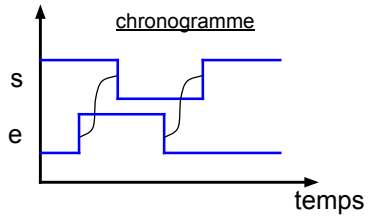
Schéma « blocs »

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 :

chronogramme



LANGAGE VHDL

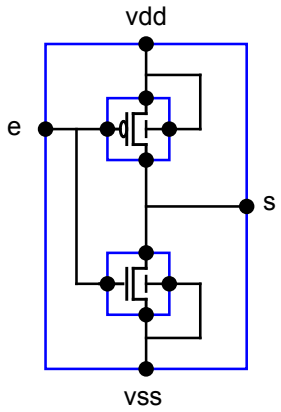
```
entity inversor is
-- liste des ports d'entrée/sortie
port (
        e : in bit ;
        s : out bit ;
        vss : in bit ;
        vdd : in bit
) ;
end inversor ;

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


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

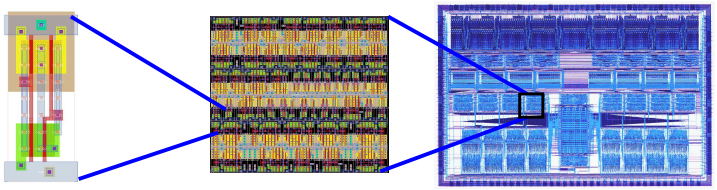
Les deux transistors N et P peuvent être considérés comme des interrupteurs contrôlés par la valeur du signal appliqué sur la grille :

Grille	0	1
Transistor N	bloqué	passant
Transistor P	passant	bloqué



La vue physique

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

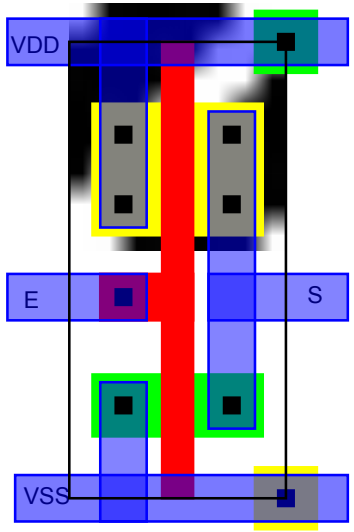


Une cellule élémentaire

Un bloc fonctionnel

Un circuit complet

Vue Physique inverseur CMOS



Dessin des masques

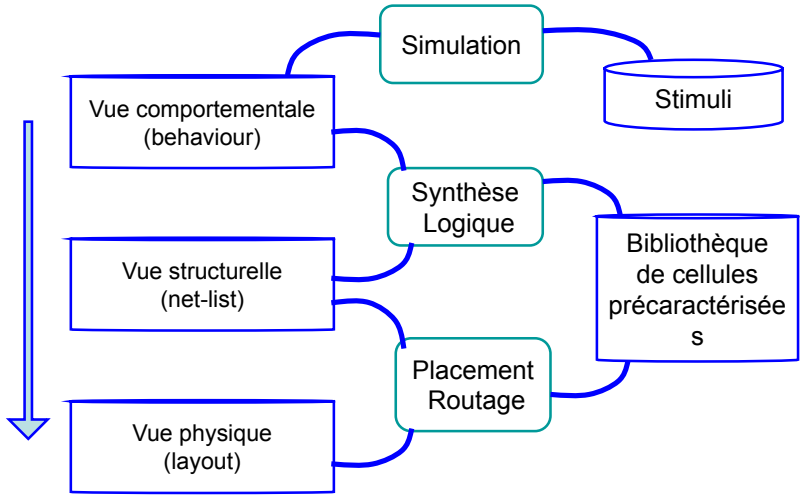
- vert : diffusion N
- jaune : diffusion P
- vert hachuré : caisson N
- rouge : Polysilicium
- bleu : métal 1
- noir : contact (trou l'isolant)

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

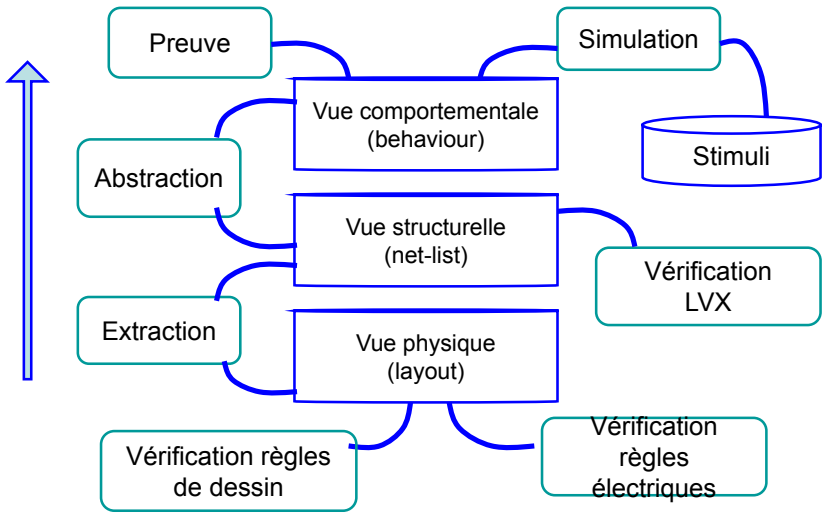
La « boîte d'aboutement » définit l'encombrement de la cellule, mais ne correspond pas à une couche physique.

Méthodologie de conception

Conception descendante



Vérification « ascendante »



Modélisation

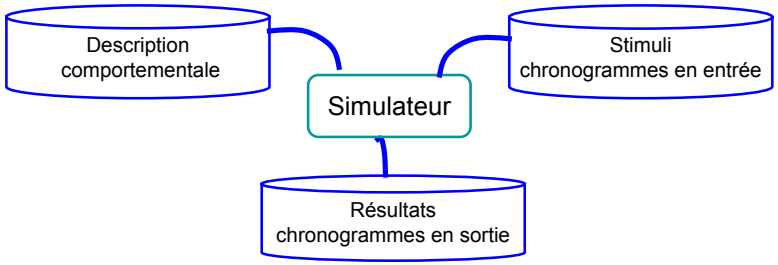
Spécification...

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 TPs associés), on écrira la description comportementale « à la main ».

Validation / Simulation...

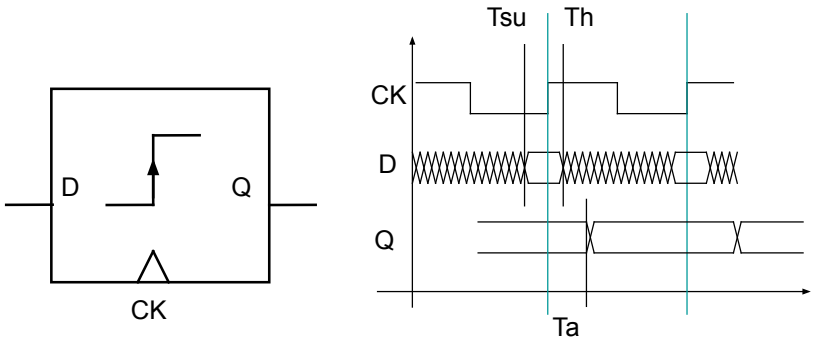
Le principal outil de validation de la description comportementale est la simulation, ce qui nécessite donc de développer les stimuli qui seront appliqués sur le modèle du composant à valider.



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

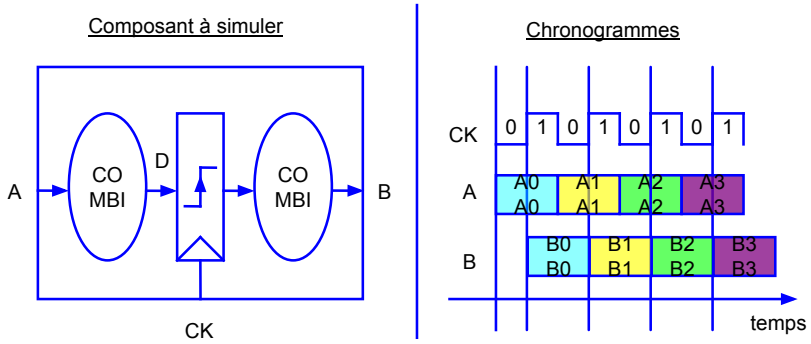
Bascule D (D flip-flop)

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)



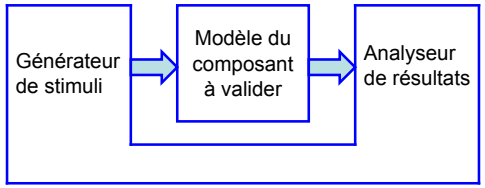
Structure temporelle des stimuli

- 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.



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.

Modélisation comportementale...

1. Niveau « algorithmique »
   - interface défini « au bit près » par une listes 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

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.

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

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...

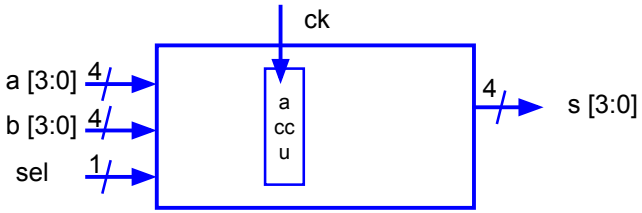
VHDL : assignations concurrentes

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.

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

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.

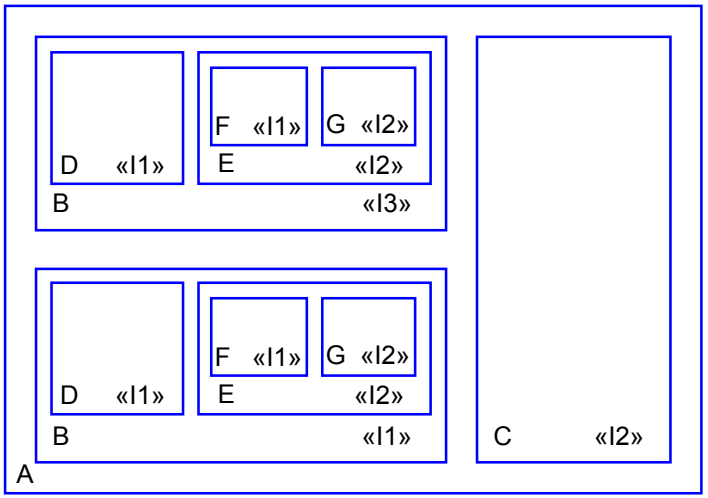
Modélisation VHDL addaccu : Architecture vbe

```
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 (sel = 0)
                             else  a[3 downto 0];
           ...
```


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)                          -- écriture dans le registre
              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 ;
```

Netlist hiérarchique

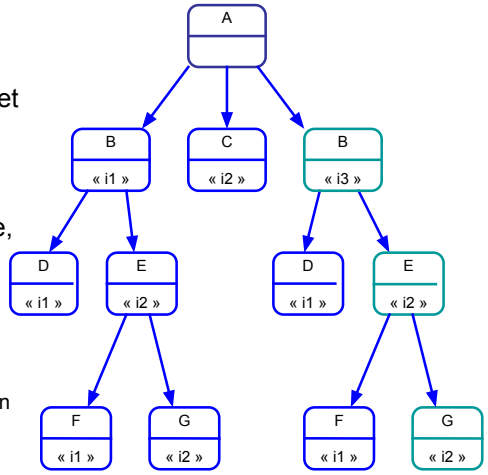


Arbre d'instanciation et Occurences

- La racine de l'arbre n'a pas de nom d'instance.
- On appelle occurence un objet quelconque du schéma multi-niveaux dans son contexte d'instanciation.
- Pour désigner une occurence, on utilise le cheminom :

« A.i3.i2.i2.sig »

A : nom de la figure racine
i3.i2.I2 : nom des instances sur le chemin
sig : nom de l'objet dans la cellule



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 paramètrable

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.

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 ELDO (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 .edif

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

Bibliothèques de cellules

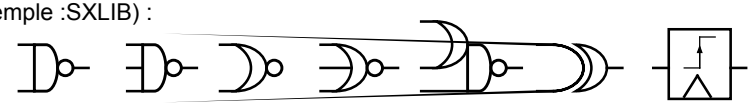
Le processus de raffinement du schéma est un processus descendant, mais sous contrainte : les composants terminaux font nécessairement partie d'une bibliothèque de cellules prédéfinies… et précaractérisées.

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).

Cellules de base

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



Nand2 Nand3 Nor2 Nor3 Nandor3 Xor2
Dff
na2 na3 no2 no3 nao22 xr2 sff1

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)

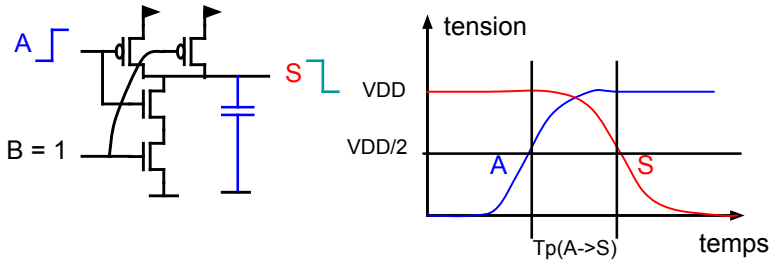
Cellules précaractérisées

- 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.

Caractérisation temporelle

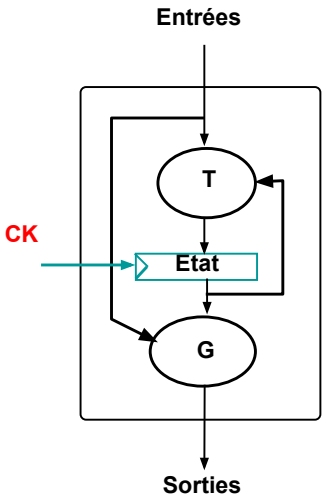
Les portes logiques sont caractérisées par des temps de propagation : un événement sur une entrée peut créer un événement sur la sortie après un temps Tp(A->S)



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

Automate d'états finis synchrones

Principe Général



Tout système numérique synchrone peut être modélisé par un automate :

Fonction de transition :

NextEtat <= T(Etat, Entrées)

Fonction de génération :

Sorties <= G(Etat, Entrées)

Automate d'états finis synchrones

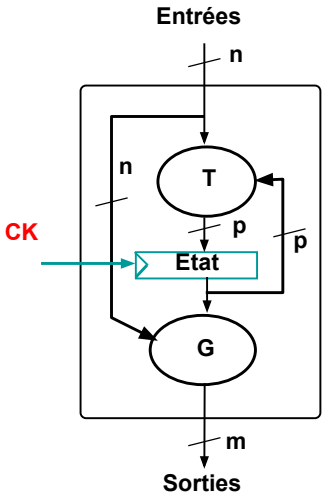
La théorie des automates est une méthode de représentation extrêmement générale du comportement d'un système matériel numérique synchrone.

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.

Fonctions de génération et de transition



Si on a :
- n bits d'entrée Ei
- m bits de sortie Sj
- p bits mémorisés Rk

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

$$S_j = \Gamma j(E_i, R_k)$$

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 associée 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.

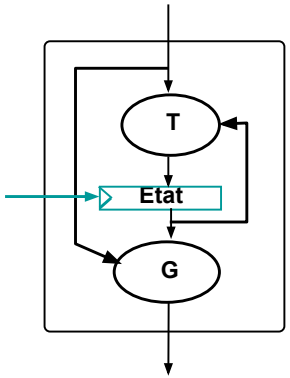
Initialisation

- 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...

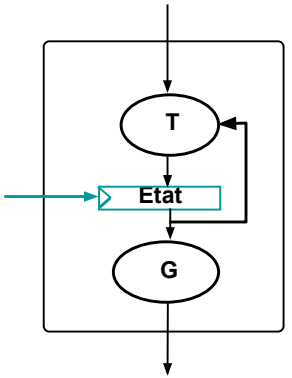
  - quel que soit l'état actuel de l'automate
  - quelle que soit la valeur des autres entrées.

Automates de Moore et de Mealy



Cas général :

Automate de Mealy



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

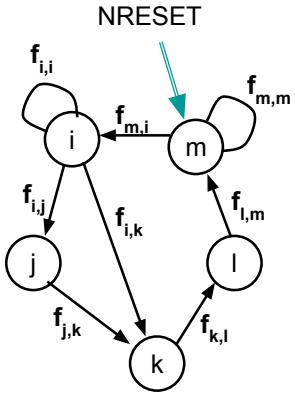
Automate de Moore

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 $\mathbf{f_{i,j}}$ 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.



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.

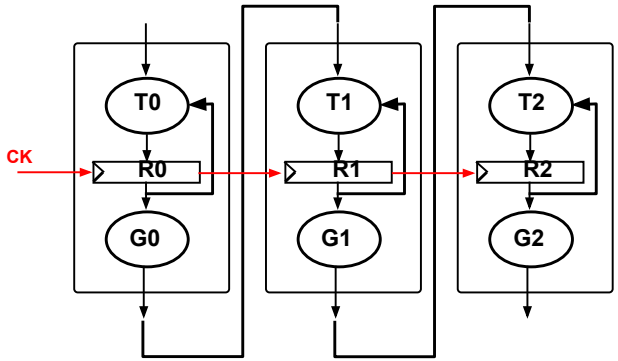
Pour tout état i, $f_{i,j} \cdot f_{i,k} = 0$ si j différent de k

- Condition de complétude : pour toute configuration des entrées, il y a toujours un état successeur de l'état i.

Pour tout état i, $\Sigma_j f_{i,j} = 1$

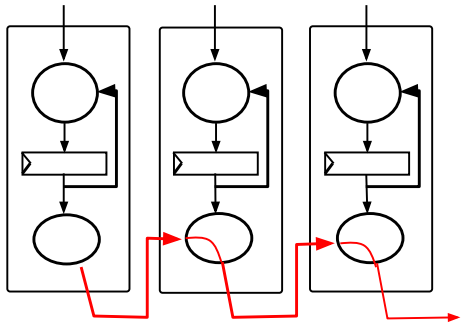
Automates communicants

Un système numérique synchrone 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.



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.

Exemple : Allocateur de bus

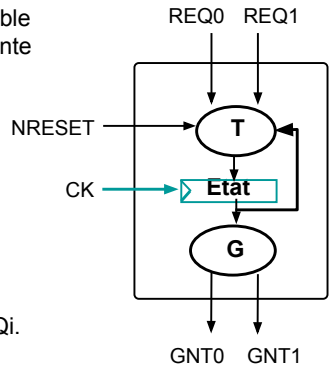
On cherche à réaliser un allocateur de bus équitable entre 2 utilisateurs (respectant une priorité tournante ou « round robin »).

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

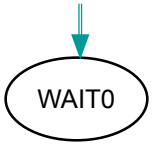
Les deux requêtes REQ0 et REQ1 sont actives à l'état haut, et indépendantes.

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



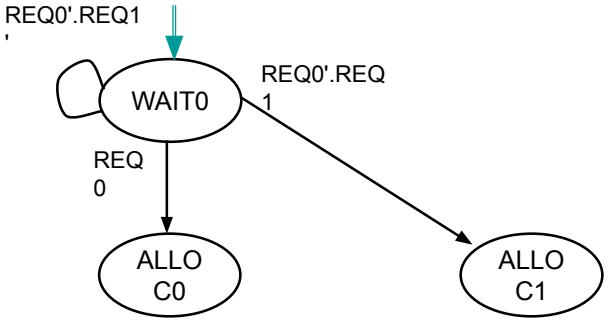
représentation graphique



On devine un état initial :

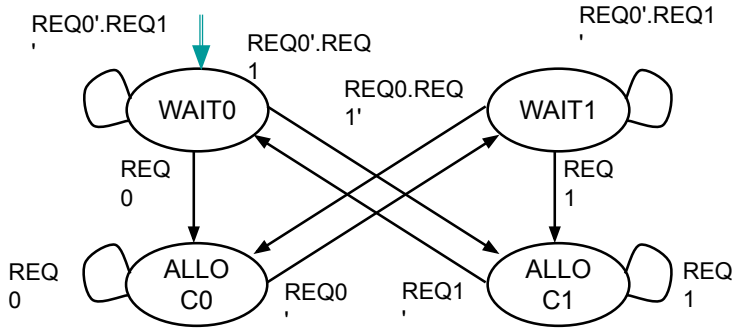
Ici l'état WAIT0 dans lequel l'automate
attend une requête alors que c'est l'utilisateur 0
qui a eu le bus en dernier

représentation graphique



Pour l'état de départ,
On étudie toutes les combinaisons de sortie
et on créé de nouveaux états.
Puis on recommence pour chaque nouvel état.

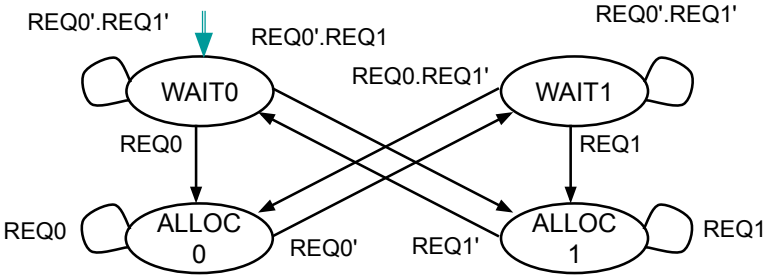
représentation graphique



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

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



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 ;

architecture fsm of allocateur is
-- déclaration des états
signal
  s_wait0,  r_wait0, -- attente req0 prioritaire
  s_wait1,  r_wait_0,-- attente req1 prioritaire
  s_alloc0, r_alloc0,-- bus alloué par req0
  s_alloc1, r_alloc1 -- bus alloué par req1
  : std_logic;

begin

 -- fonction de génération
   gnt0 <= r_alloc0;
   gnt1 <= r_alloc1;
```

```
-- fonction de transition
    s_wait0  <= (r_wait0 and not req0 and not req1)
             or (r_alloc1 and not req1);
    s_wait1  <= (r_wait1 and not req0 and not req1)
             or (r_alloc1 and not req0);
    s_alloc0 <= (r_wait0 and req0)
             or (r_alloc0 and req0);
    s_alloc1 <= (r_wait1 and req1)
             or (r_alloc1 and req1);

-- mise à jour du registre d'état
   reg : 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;
end fsm;
```


B : 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 ;
architecture fsm of allocateur is

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

-- déclaration des signaux
signal present, futur : 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
       present <= futur;
        end if ;
   end process;
```

```
-- processus « combinatoire »
process ( present, req0, req1, nreset ) begin
   -- fonction de transition
   if (nreset = '0') then   futur <= wait0 ;
   else
     case present is
     when wait0 => if (req0) then futur <= alloc0 ;
                elsif (req1) then futur <= alloc1 ;
                end if ;
     when wait1 => if (req1) then futur <= alloc1 ;
                elsif (req0) then futur <= alloc0 ;
                end if ;
     when alloc0 => if ((req0)='0') then futur <= wait1;
                end if ;
     when alloc1 => if ((req1)='0') then futur <= wait0;
                end if ;
     end case ;
   end if ;

-- 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;
```


Circuits numériques synchrones

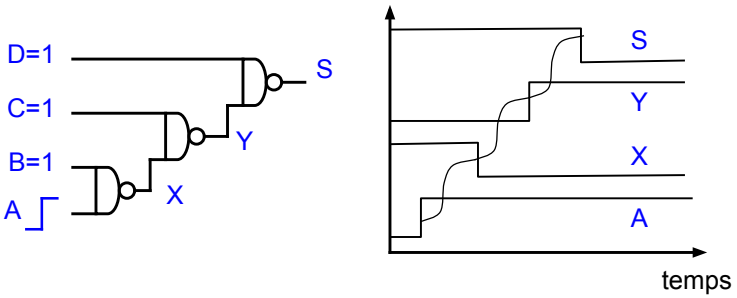
Le triple rôle des registres

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.

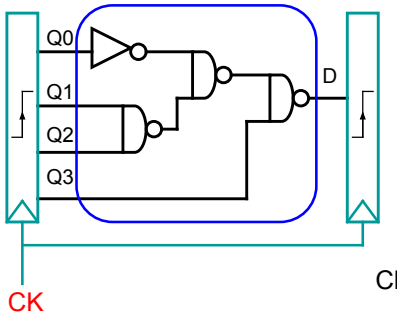
Propagation des événements

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

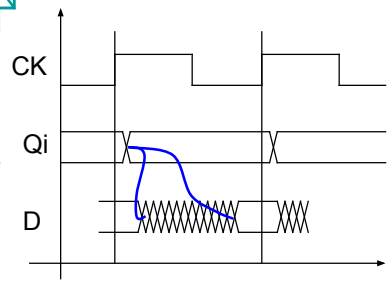


=> Il ne doit pas y avoir de boucle dans un bloc combinatoire

Les temps de propagation

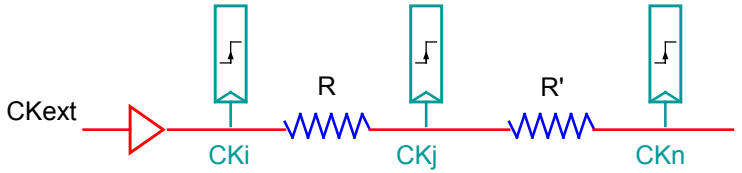


Dans l'opérateur combinatoire,
les temps de propagation
Q1 -> D et Q3 -> D
sont très différents...



- Les signaux de sortie des registres (Qi) sont stables pendant tout le cycle
- Les signaux d'entrée des registres ne doivent être stables qu'au moment du front montant d CK.

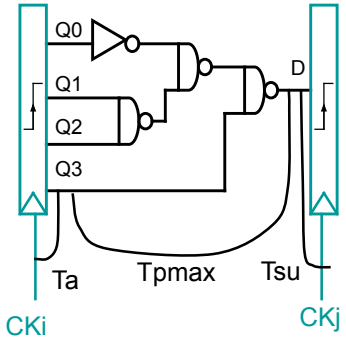
Le skew



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

Chaînes longues



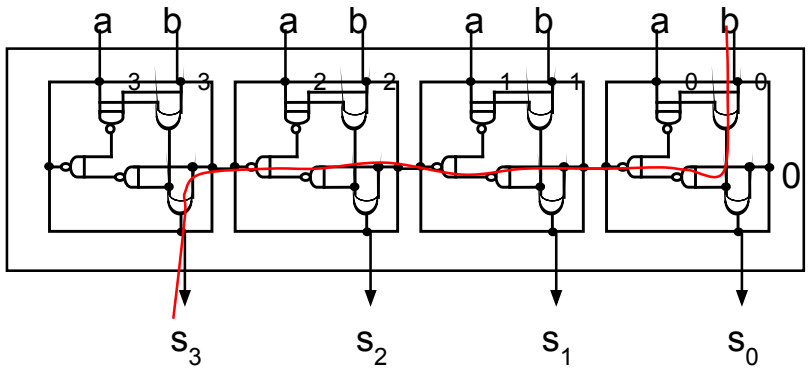
Pour qu'un circuit numérique synchrone fonctionne correctement, il faut que le temps de propagation le la chaîne combinatoire la plus longue Tpmax entre deux registres soit inférieur au temps de cycle TC.

Il faut tenir compte du temps d'accès Ta et du temps de pré-établissement Tsu des registres, ainsi que du « skew »

Tout opérateur combinatoire doit respecter la condition suivante :

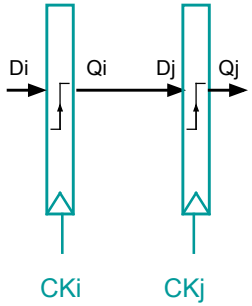
$$Ta(CK->Qi) + Tpmax(Qi->D) + Tsu(D->CKj) < TC - skew$$

Chaîne longue additionneur 4 bits



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

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).

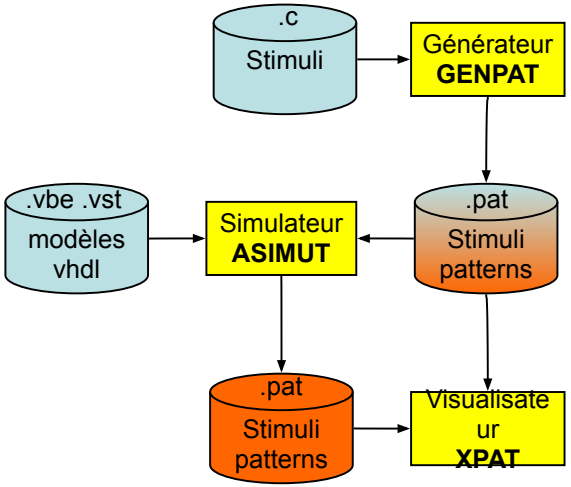
Il faut ici également tenir compte du temps d'accès Ta du registre source et du temps de maintien Th du registre destination.

Tout opérateur combinatoire doit respecter la condition suivante :

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

Chaîne de simulation

Chaine de 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`**

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 : 0F0F F6F0 0  +  **** ?54FC;

end;
```


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

Les noms de vecteurs de signaux ont le format VHDL

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;
```


Séquence de patterns

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

- **date** :: [+] *nombre_entier unité*
  - 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

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

-i file initialise tous les signaux à partir du fichier **file** (sauvé par la 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.

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

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

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
```

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 : "nombre" 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

API Genpat : Exemple

fonction transformant un entier en chaine de caractères

fichier vecteurs.pat

déclaration de l'interface et des signaux internes

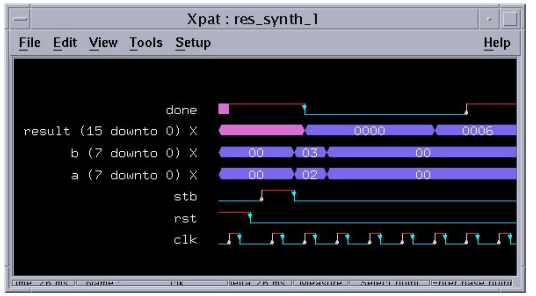
boucles imbriquées produisant 256 patterns

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

char *itoa(int entier)   {
    char * str = malloc (32);
    sprintf (str, "%d",entier);
    return(str);
}

main ()  {
    int i, j, cur_vect = 0;

    DEF_GENPAT("vecteurs");

    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 ();
}
```


Le visualisateur de patterns XPAT

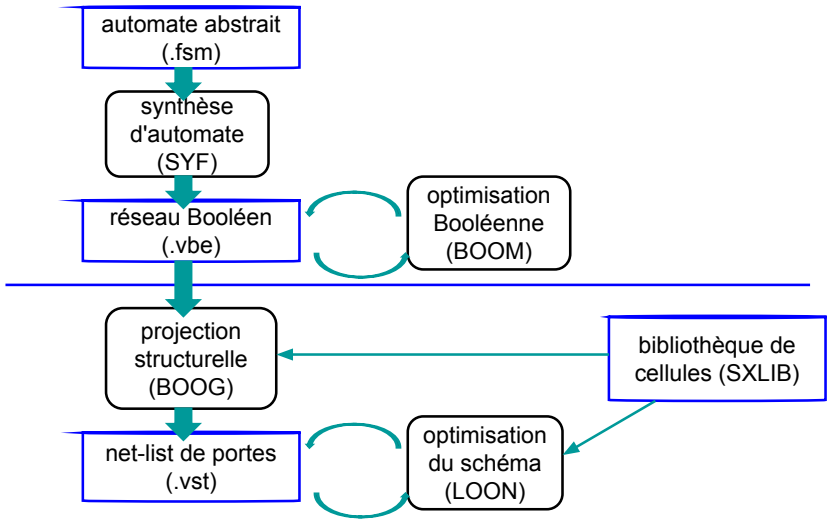
XPAT est l'un des nombreux outils de visualisation
- avantage : fonctionnement simple et intuitif.
- inconvenient : peu de fonctions

`xpat [-l file]`



Chaîne de synthèse

Les étapes de la synthèse Alliance



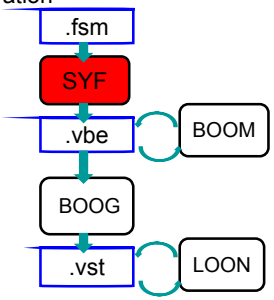
La synthèse d'automate

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.

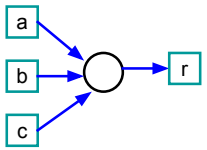
Contrairement à l'intuition, le codage
« one-hot » donne très souvent
de bons résultats !



Optimisation Booléenne / a

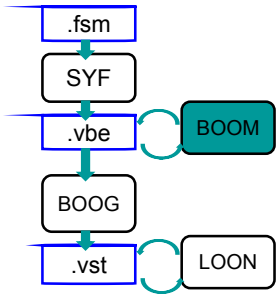
Les outils d'optimisation Booléenne (exemple : BOOM) cherchent
à « simplifier » le réseau Booléen. Cette optimisation étant indépendante du
procédé de fabrication choisi, la fonction de coût est le « nombre de littéraux ».

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



forme canonique : 12 littéraux
r <= a.b.c' + a.b'.c + a'.b.c + a.b.c

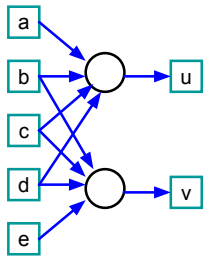
forme optimisée : 6 littéraux
r <= a.b + a.c + b.c



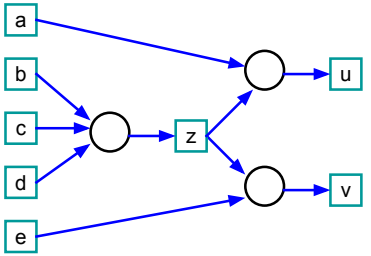
Optimisation Booléenne / b

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

Forme initiale :
u <= a.(b.c + b'.d)
v <= (b.c + b'.d) + e



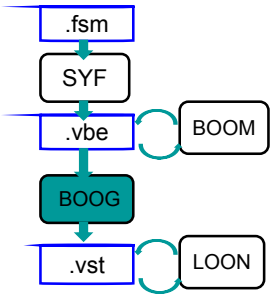
Forme factorisée :
z <= b.c + b'.d
u <= a.z
v <= z + e



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.

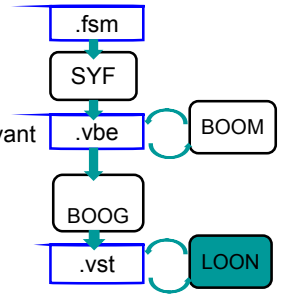
- Ces outils s'appuient sur une bibliothèque de cellules précaractérisées.
- Le traitement est local : chaque expression Booléenne est traitée indépendamment.
- Ils utilisent des techniques de reconnaissance de forme pour reconnaître des sous-expressions.
- Ils exploitent les informations de caractérisation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



Optimisation du schéma / a

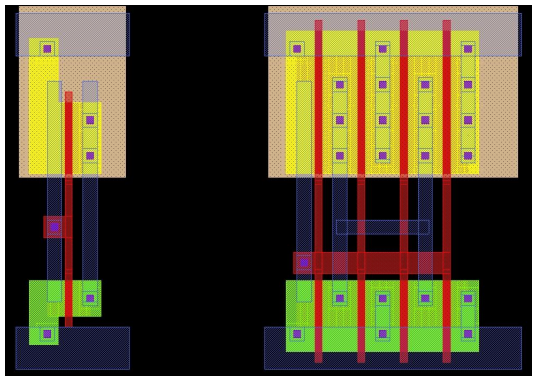
Les outils d'optimisation de schéma (exemple: LOON) visent principalement l'optimisation des performances temporelle. Ils cherchent à réduire les temps de propagation sur les chemins critiques du bloc synthétisé.

- 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



Optimisation du schéma / b

Ajustement de la puissance des portes :



inv_x1 : WN = 5 / WP = 10 inv_x8 : WN = 40 / WP = 80

Optimisation du schéma / c

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

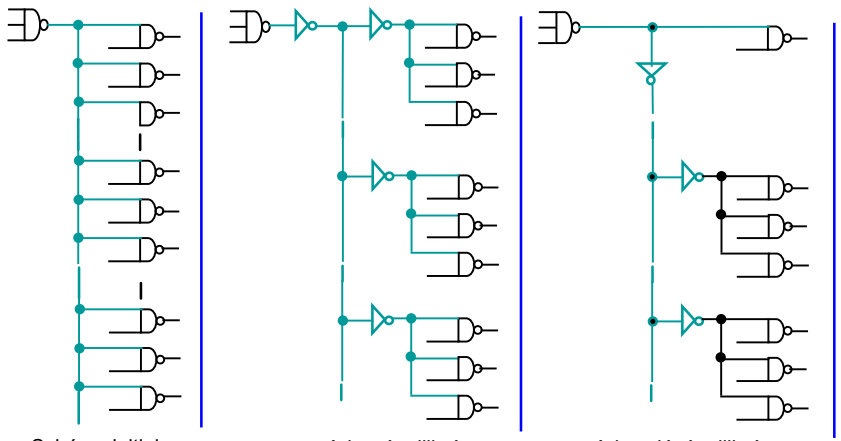
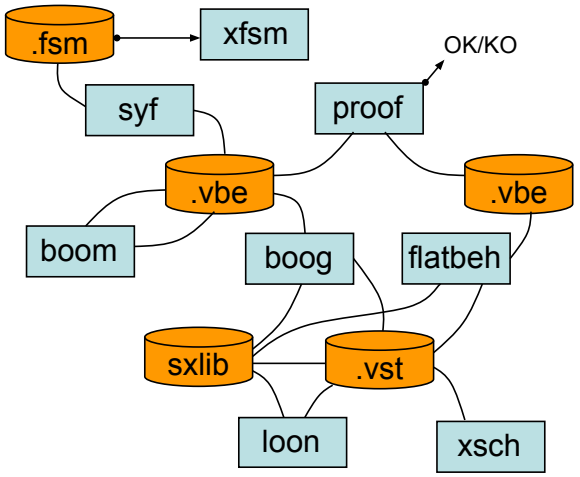


Schéma initial Arbre équilibré Arbre déséquilibré

Flot des outils de synthèse Alliance



syf synthèse fsm

boom optimisation comportementale

boog projection structurelle

loon optimisation électrique

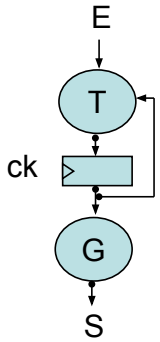
flatbeh mise à plat

xsch visualisation du schéma

xfsm visualisation du graphe fsm

syf

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

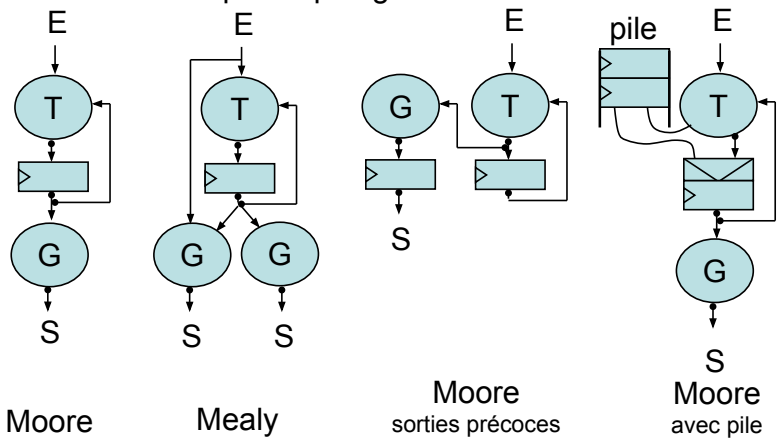


principe de fonctionnement

- le comportement est décrit en vhdl
  - un process ***Transition*** définissant 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

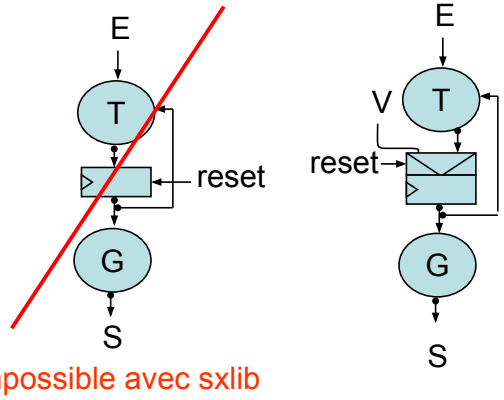
syf

Quelques topologies d'automates.



syf

L'initialisation peut être ou ne pas être synchronisée



syf

Codage d'un automate d'un FSM

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

paramètres:

-j -a -m : trois algos de codage (asp, jedi, mustang)
-o : codage one-hot
-u : codage utilisateur dans le fichier input.enc
liste des couples: noms_d'état code_hexa
-r : codage random

-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

boom

Optimisation Booléenne

boom [-VTOAP] [-l num] [-d num] [-i num] [-a num]
[-sjbgpwtmorn] input [output]

paramètres

-V : verbose
-A : procède à des optimisations locales en conservant
la majorité des signaux internes.
-P : lit le fichier input.boom contenant
- les signaux à conserver dans le fichier produit
- les expressions à conserver
-l val : effort de 0 (faible) à 3 (fort)
-d val : type d'optimisation de 0 (délai) à 100 (surface)
-i val : nb d'itérations pour l'algorithme
-a val : amplitude pendant le réordonnancement des bdd
-sjbgpwtmorn : algorithme choisi

environnement

MBK_WORK_LIB répertoire de sortie

boog

boog prend une description comportementale
et produit une netlist de cellules précaractérisées.

- **boog** ne sait traiter que des expressions produites par **boom.**
- **boog** n'utilise que des portes à une sortie et de faible sortance.
- **boog** connaît les caractéristiques des portes
grâce à des génériques dans les vbe des portes
- La projection d'un ET à 8 entrées

s <= a and b and c and d and e and f and g and h

8 and à 2 entrées

s <= a and (b and (c and (d and (e and (f and (g and h))))))

2 nand à 4 entrées + 1 nor à 2 entrées

s <= not(not (a and b and c and d) or not (e and f and g and h))

- **boog** pourrait se contenter de :
  - nand 2, nor 2, basculeD, inverseur, xor
- **boog** n'est pas déterministe !

boog

Projection structurelle

boog [-hmxold] input output [lax_file]

paramètres

-h : help
-m val : optimisation de 0 (surface) à 4 (delai)
-x val : génération d'un fichier de coloration des signaux
0 (chemin critique), 1 (dégradé en fonction du délai)

environnement

MBK_CATA_LIB liste des répertoires contenant les fichiers sources
MBK_TARGET_LIB répertoire de la bibliothèque cible
MBK_OUT_LO format de la netlist de sortie
MBK_WORK_LIB répertoire de sortie

loon

Optimisation électrique locale

loon [-hmxlo] input_file output_file [lax_file]

paramètres

-h	: help
-m val	: optimisation de 0 (surface) à 4 (délai)
-x val	: génération d'un fichier de coloration des signaux 0 (chemin critique), 1 (dégradé en fonction du délai)

environnement

MBK_CATA_LIB	liste des répertoires contenant les fichiers sources
MBK_TARGET_LIB	répertoire de la bibliothèque cible
MBK_IN_LO	format de la netlist d'entrée
MBK_OUT_LO	format de la netlist de sortie
MBK_WORK_LIB	répertoire de sortie

flatbeh

flatbeh réalise la mise à plat d'une netlist et produit le modèle comportemental.

flatbeh root_structural_file [output_file]

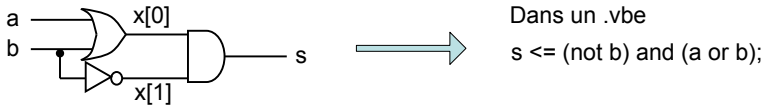
paramètre

CATAL	fichiers contenant les cellules feuilles

environnement

MBK_CATA_LIB	liste des répertoires contenant les fichiers sources
MBK_IN_LO	format de la netlist d'entrée
MBK_OUT_LO	format de la netlist de sortie
MBK_WORK_LIB	répertoire de sortie

Dans un .vst



Dans un .vbe

s <= (not b) and (a or b);

*(on aura intérêt à utiliser **boom** pour simplifier le résultat)*

proof

proof compare formellement deux descriptions comportementales et affiche les différences.

- Les deux descriptions doivent avoir
  - les mêmes entrées/sorties
  - les mêmes registres
- Chaque description est représentée par une structure ROBDD (Reduced Oriented Binary Decision Diagram)
- La représentation d'une expression Booléenne par un ROBDD est canonique
  ⇒ si les représentations sont superposables, c'est qu'elles sont égales

xsch

xsch représente une netlist par un schéma