Schémas Hiérarchiques Multi-niveaux

version 1.0

Plan

- **Conception structurée**
- **Bibliothèque de cellules précaractérisées**
- **Décomposition structurelle pour « addaccu »**
- **Modèle VHDL structurel**

La vue structurelle

La conception structurée vise deux objectifs :

- exploiter les régularités.
- maîtriser la complexité.

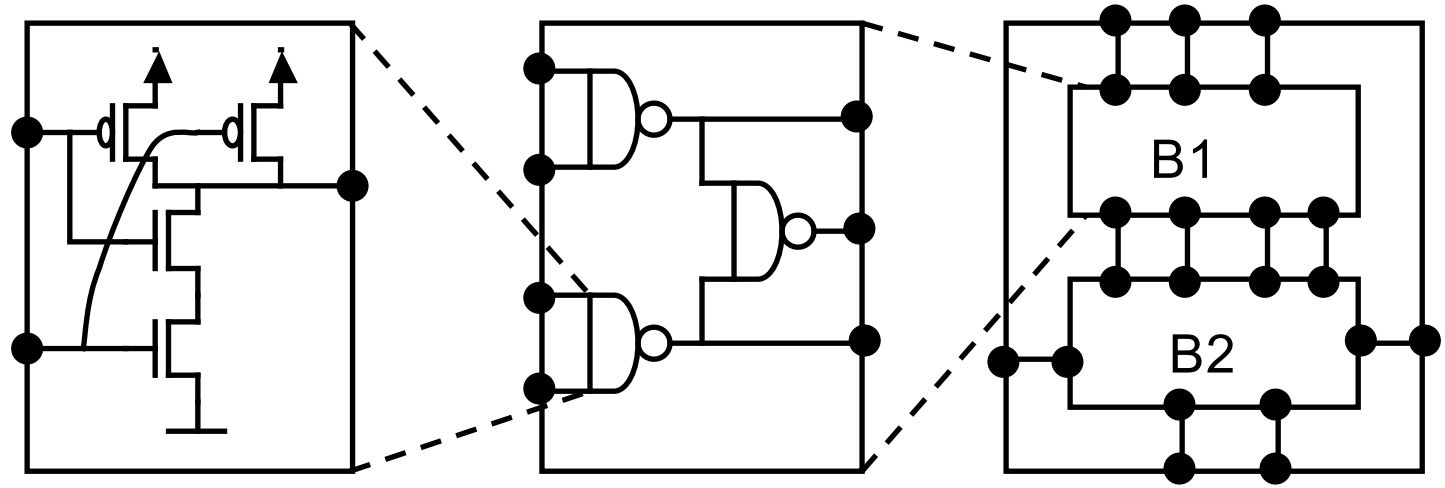


Schéma « transistors »

Schéma « portes logiques »

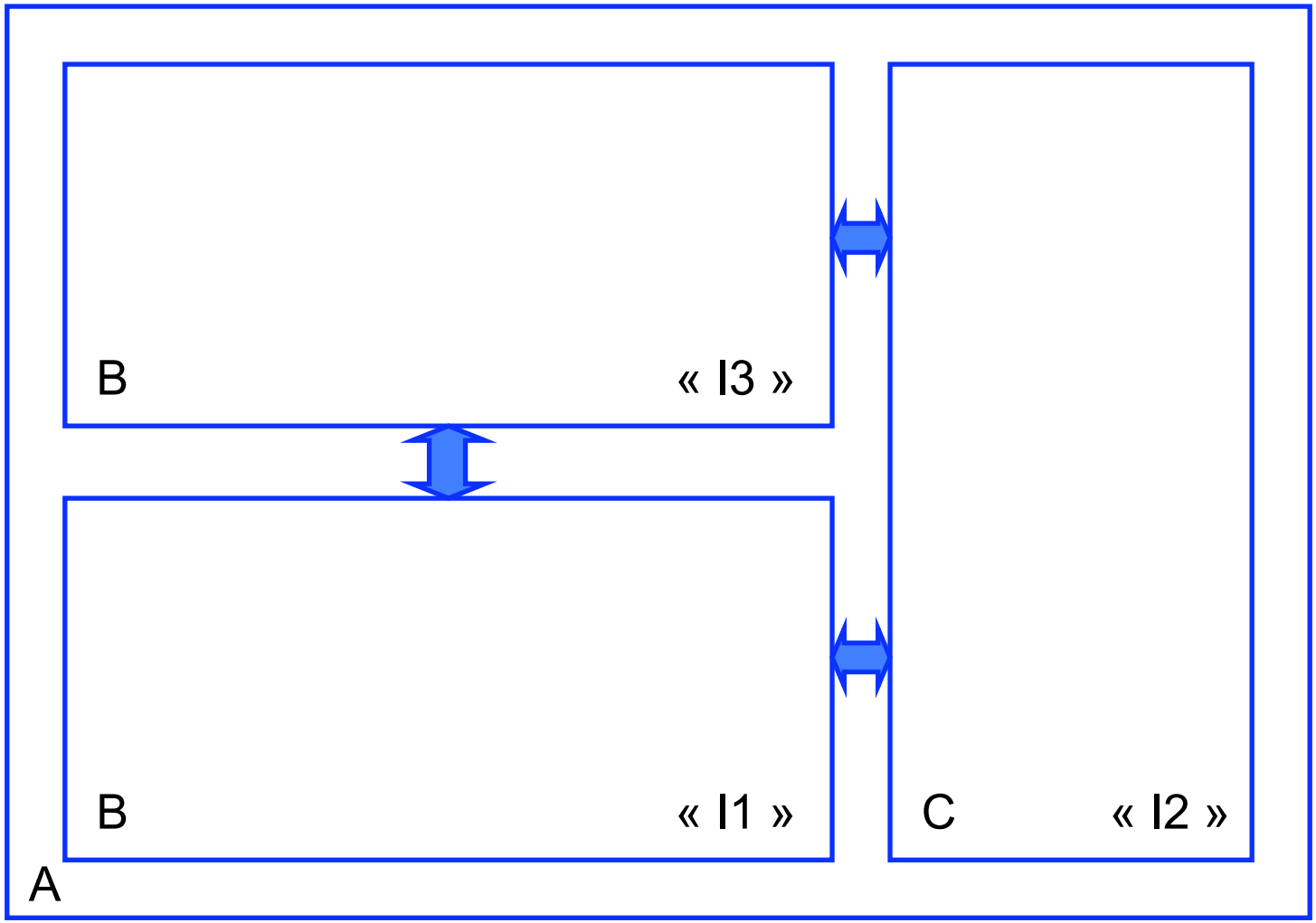
Schéma « blocs »

Raffinement progressif du schéma / 1

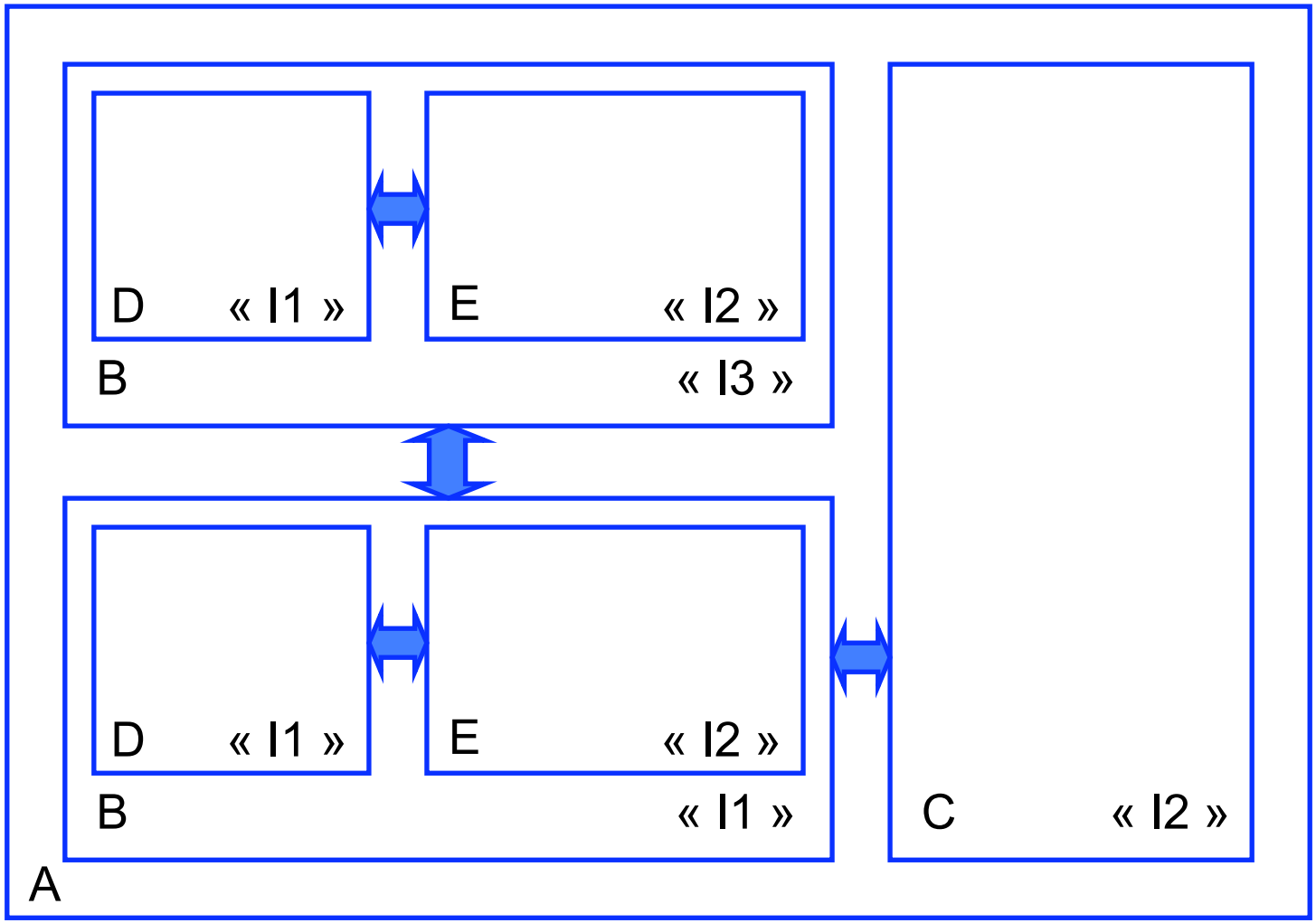
On dispose d'une spécification comportementale
du composant A, et on souhaite obtenir une description
structurelle sous la forme d'un schéma multi-niveaux...

A

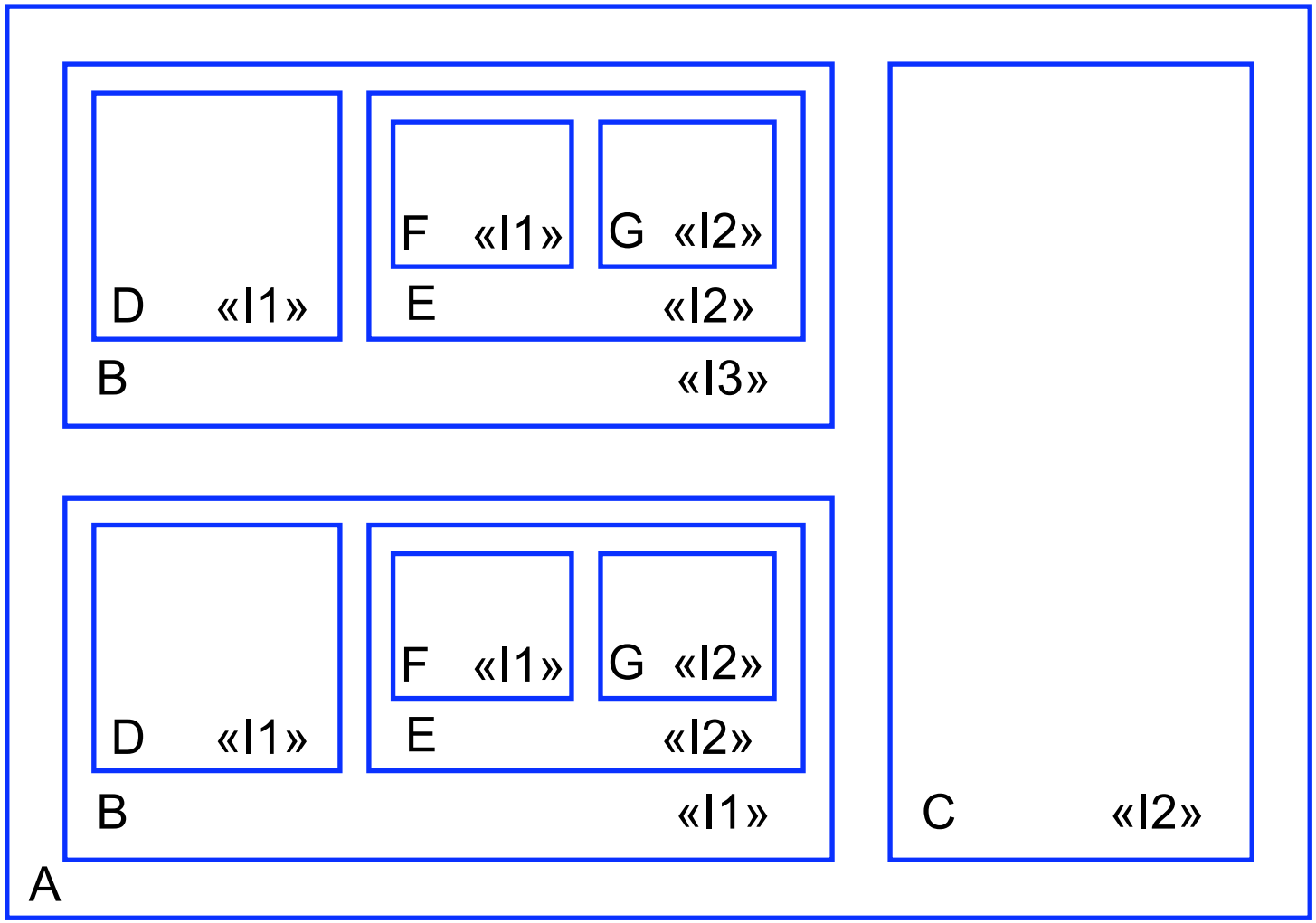
Raffinement progressif du schéma / 2



Raffinement progressif du schéma / 3



Raffinement progressif du schéma / 4



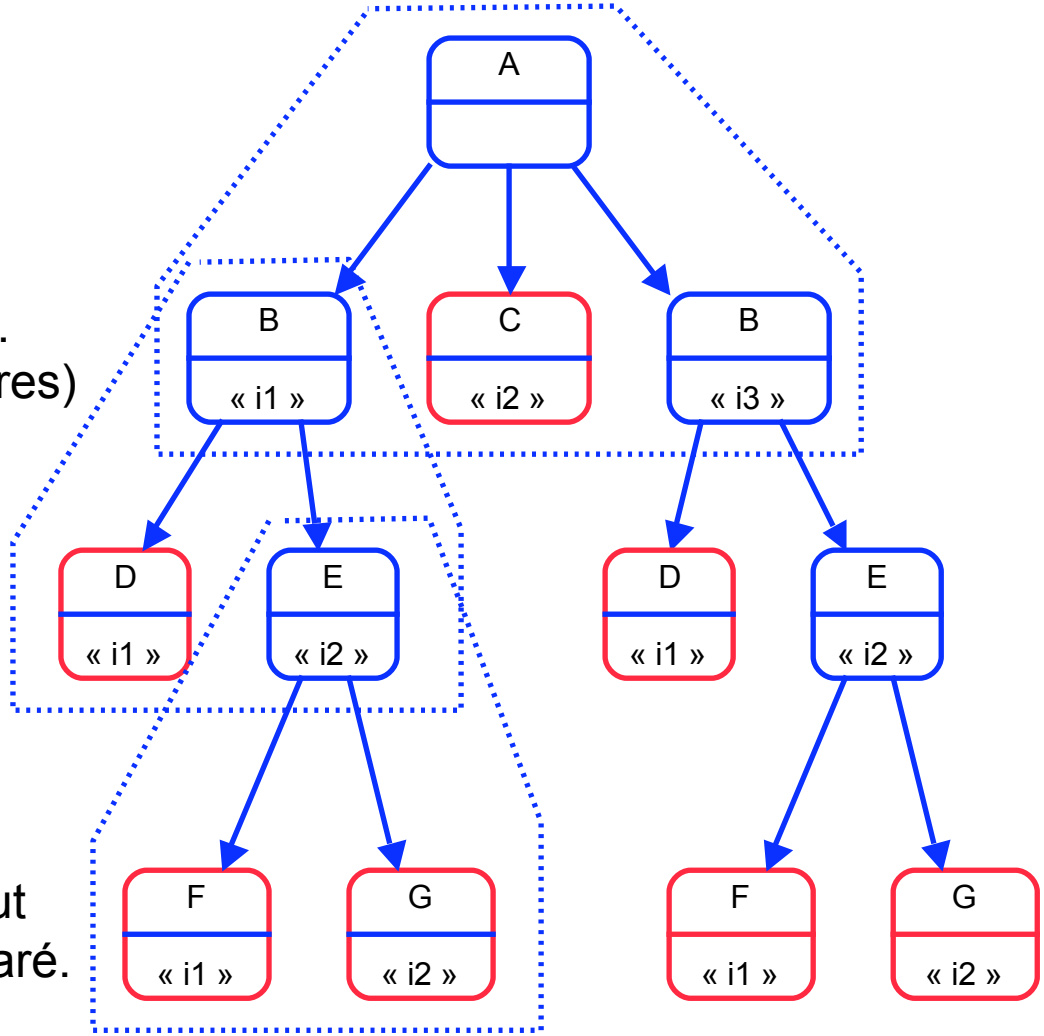
Arbre d'instanciation

- 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.

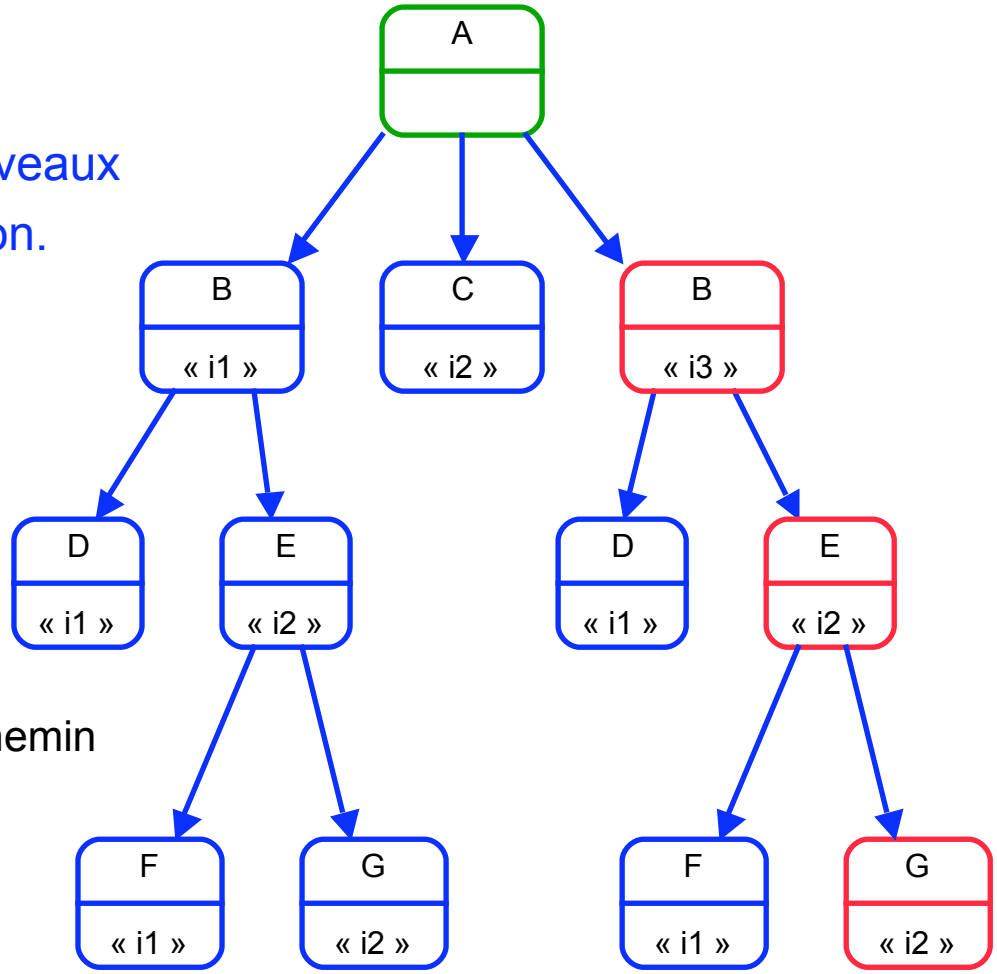


Occurences

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 interprêté 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 langagesde description de net-list permettent de représenter la même information : des schémas d'interconnexion hiérarchiques multi-niveaux...

Plan

- Conception structurée
- Bibliothèque de cellules précaractérisées
- Décomposition strcuturelle pour « addaccu »
- Modèle VHDL structurel

Bibliothèques de cellules

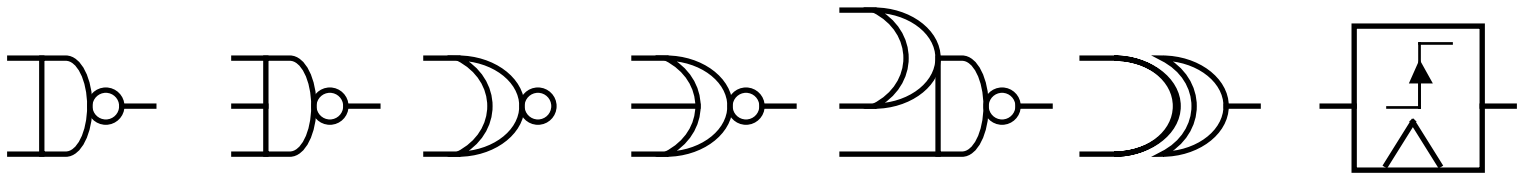
Le processus de raffinement du schéma est un processus descendant, mais sous contrainte : les composants terminaux font nécessairement partie d'une bibliothèque de cellules prédéfinies… et précaractérisées.

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

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

Elles contiennent - pour chaque cellule - les différentes vues nécessaire aux outils CAO :

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

Cellules précaractérisées

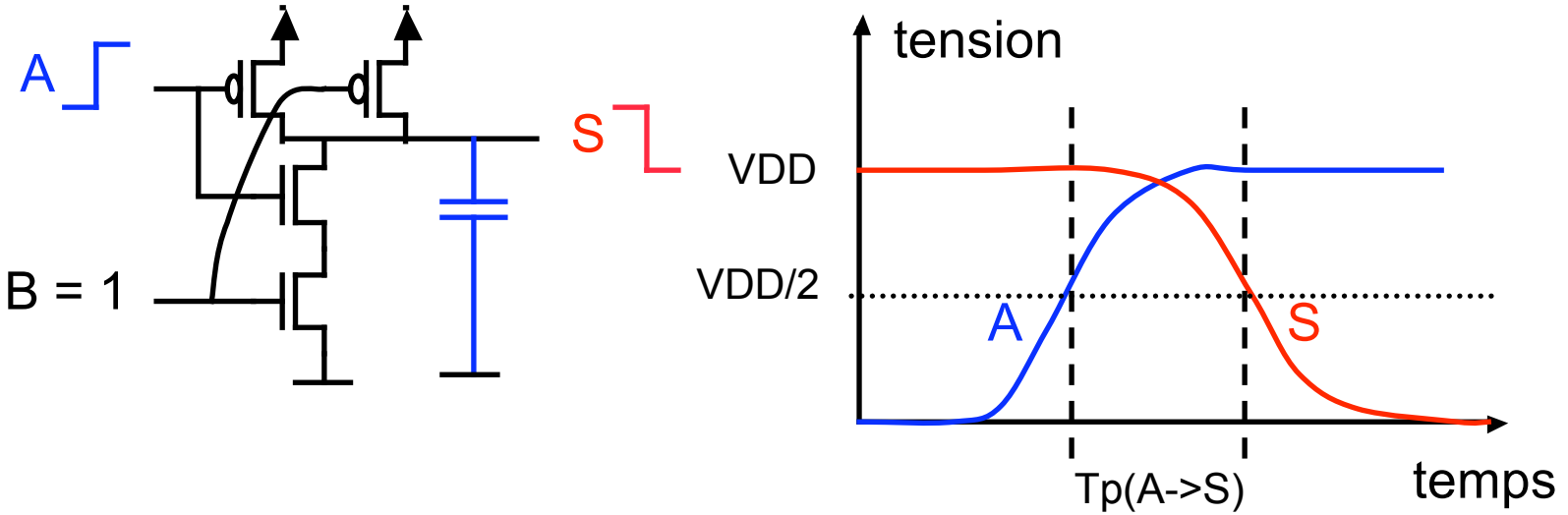
- Le dessin des masques des cellules respecte une série de contraintes topologiques (appelées gabarit), permettant l'automatisation du placement et du routage. Cette vue - de même que le schéma en transistors - n'est généralement pas accessible au concepteur.

- 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)



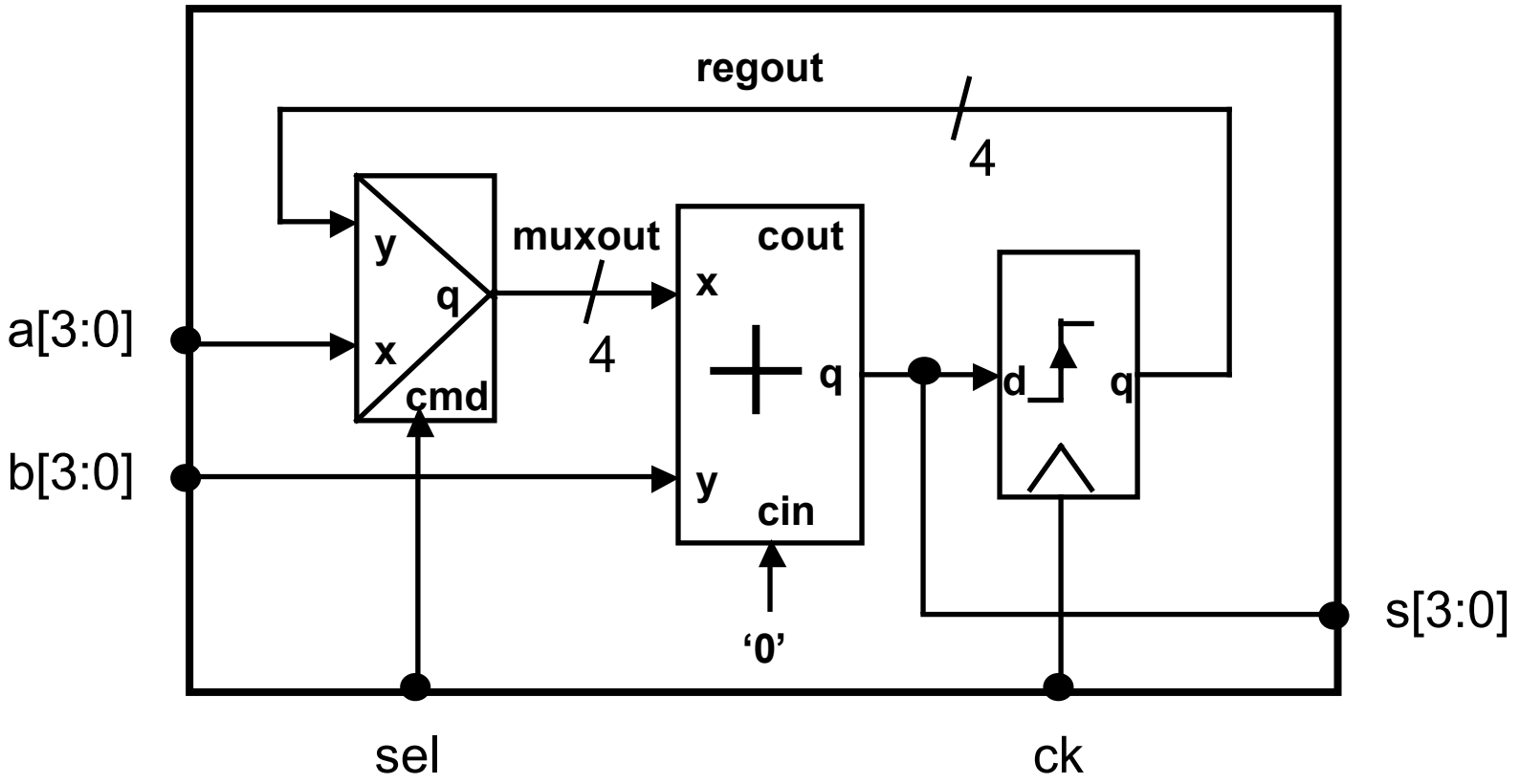
Le temps de propagation dépend de la capacité de charge et de la puissance de la porte (dimensions des transistors).

Plan

- **Conception structurée**
- **Bibliothèque de cellules précaractérisées**
- **Décomposition structurelle pour « addaccu »**
- **Modèle VHDL structurel**

Décomposition « addaccu »

Le composant addaccu peut se décomposer en trois blocs :



Raffinement progressif du schéma

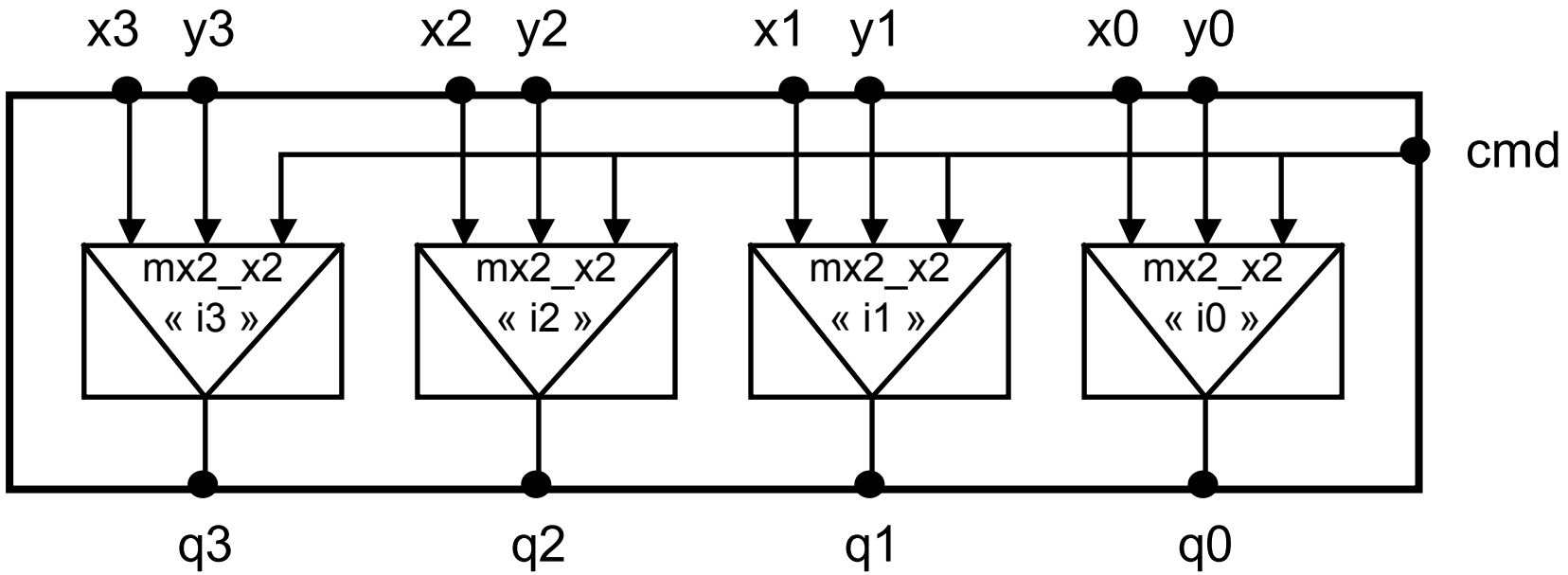
- Pour valider ce premier découpage structurel, on peut écrire des modèles comportementaux pour chacun des trois blocs mux4, adder4, et accu4...
  et re-simuler, en utilisant le même test-bench que celui qui a
  permis de valider le modèle comportemental du composant addaccu.

- 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.

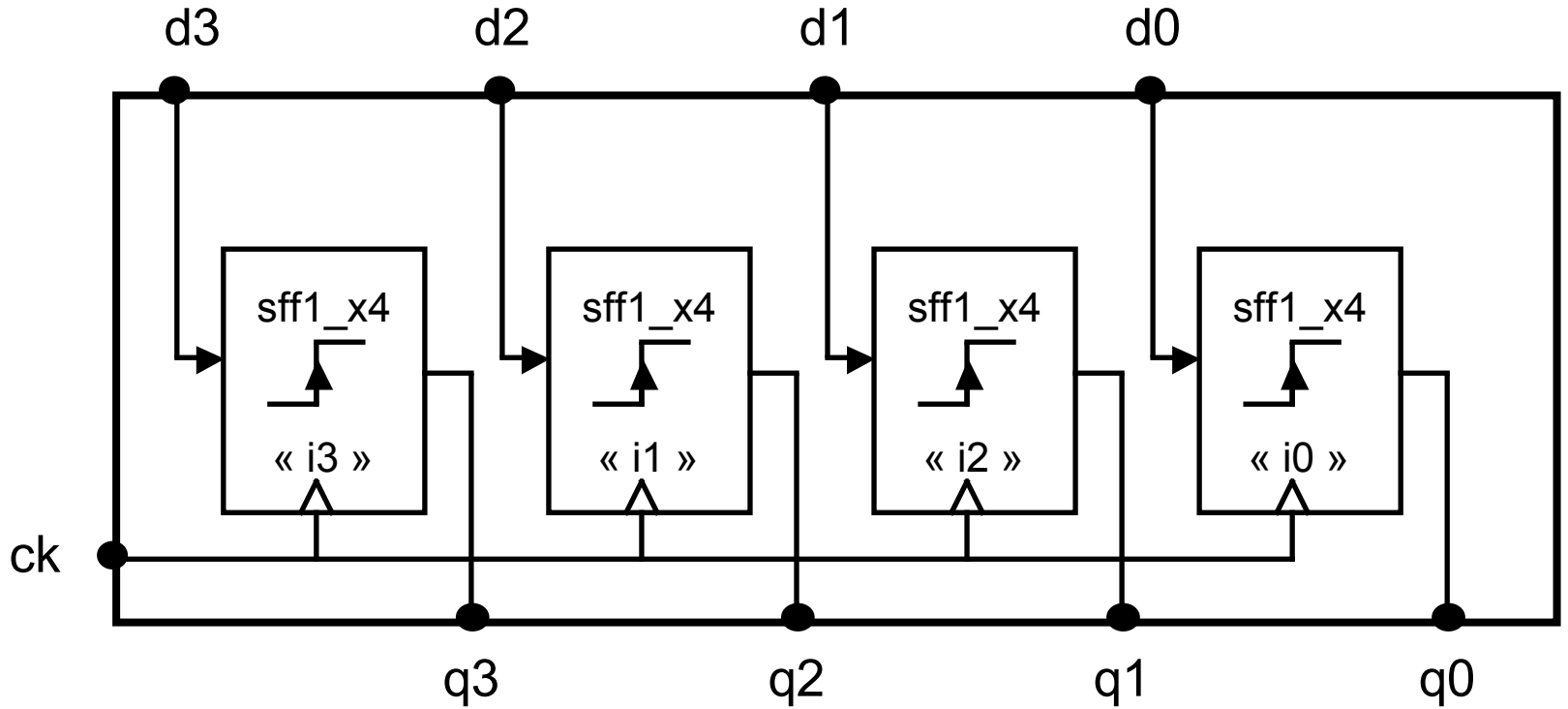
Multiplexeur « mux4 » : structure interne

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 :



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 :



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 ») :

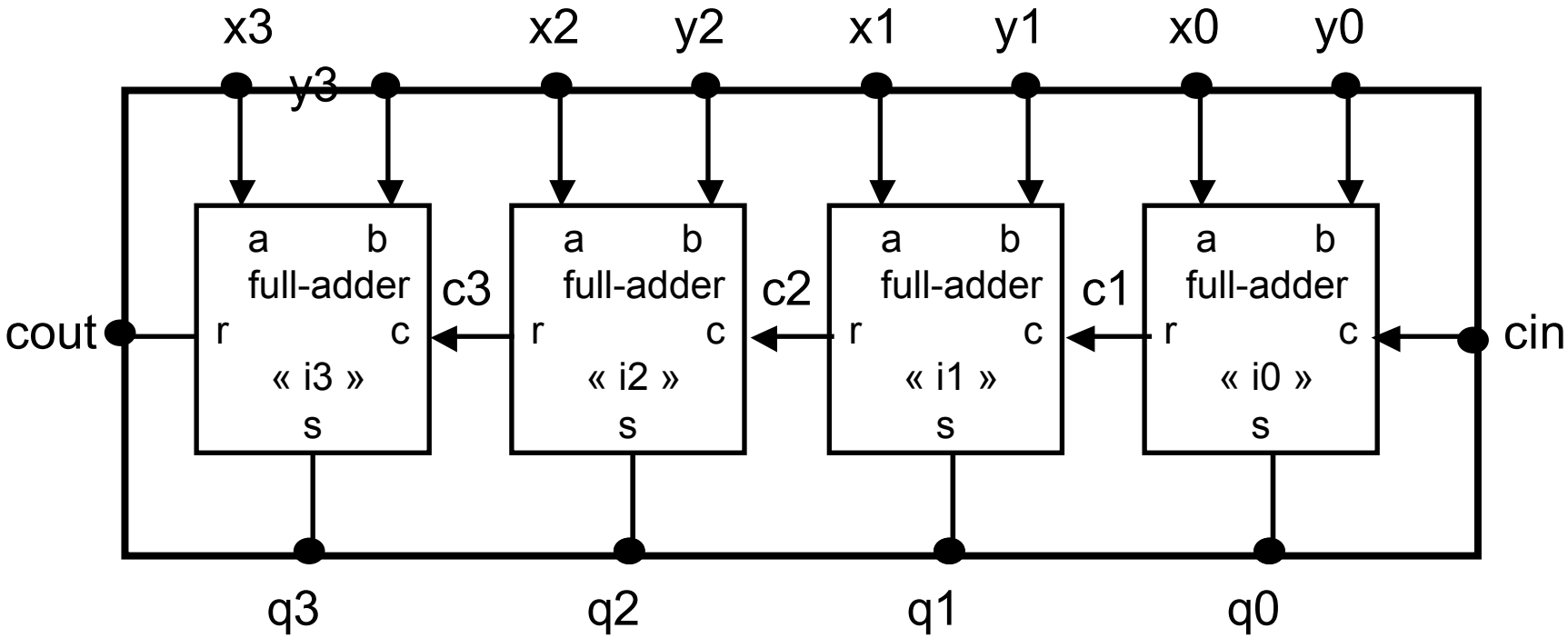


Table de vérité additionneur 1 bit

$$
\begin{array}{rl}
 & a_i \\
+ & b_i \\
+ & c_i \\
\hline
 & r_i \;\; s_i
\end{array}
$$

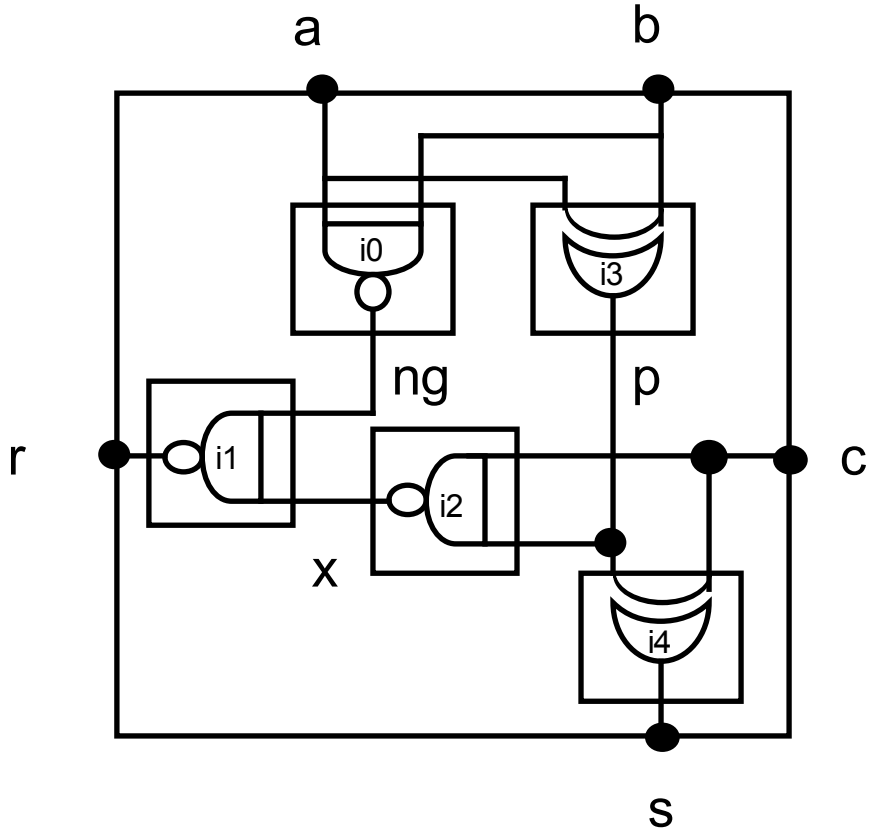
La somme de trois bits de même poids numérique peut prendre trois valeurs : 0 , 1 , 2 , 3
Cette somme peut se recoder sur deux bits r_i et s_i
(r_i a un poids numérique double de s_i)

a_i	b_i	c_i	r_i	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

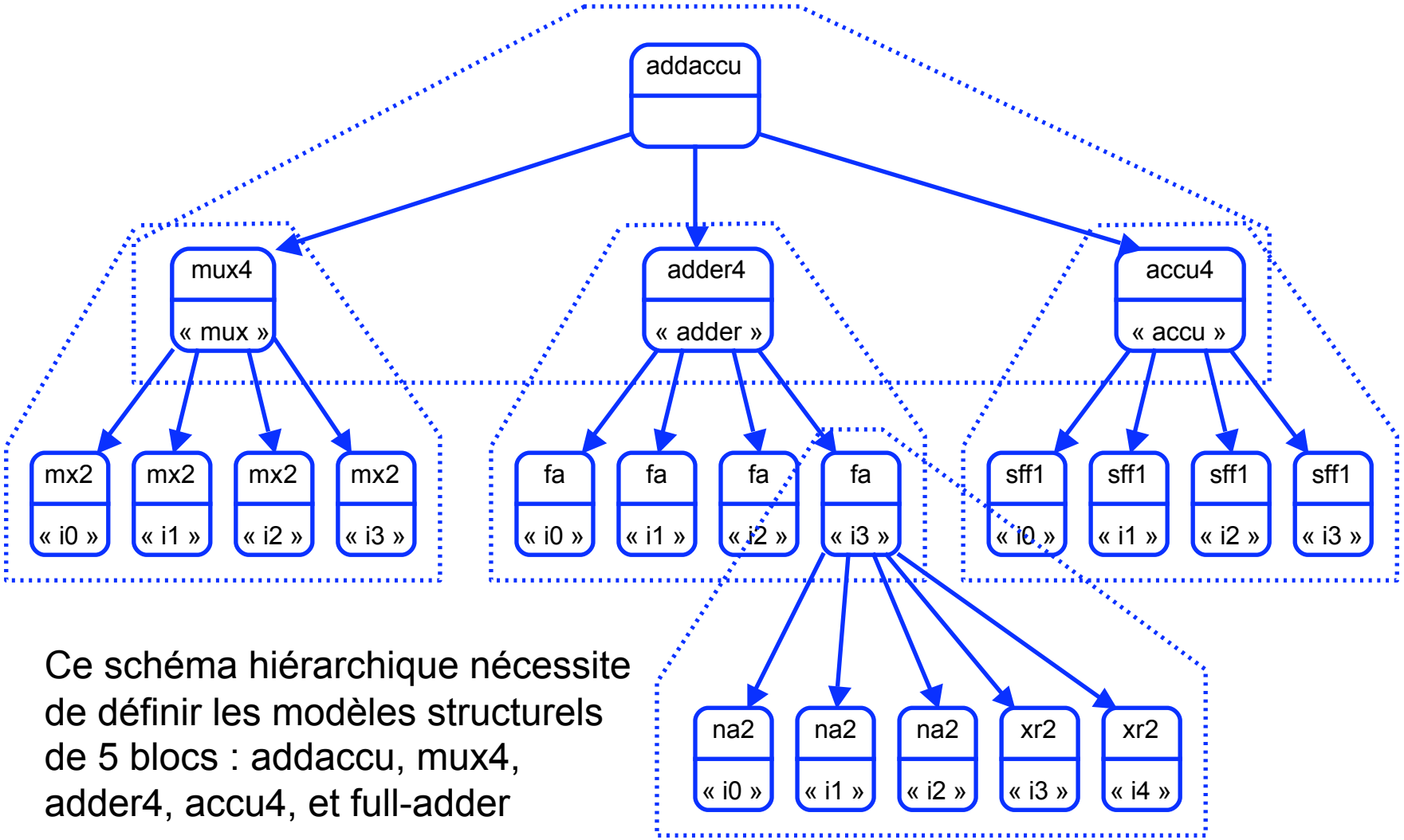
Table de vérité

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) :



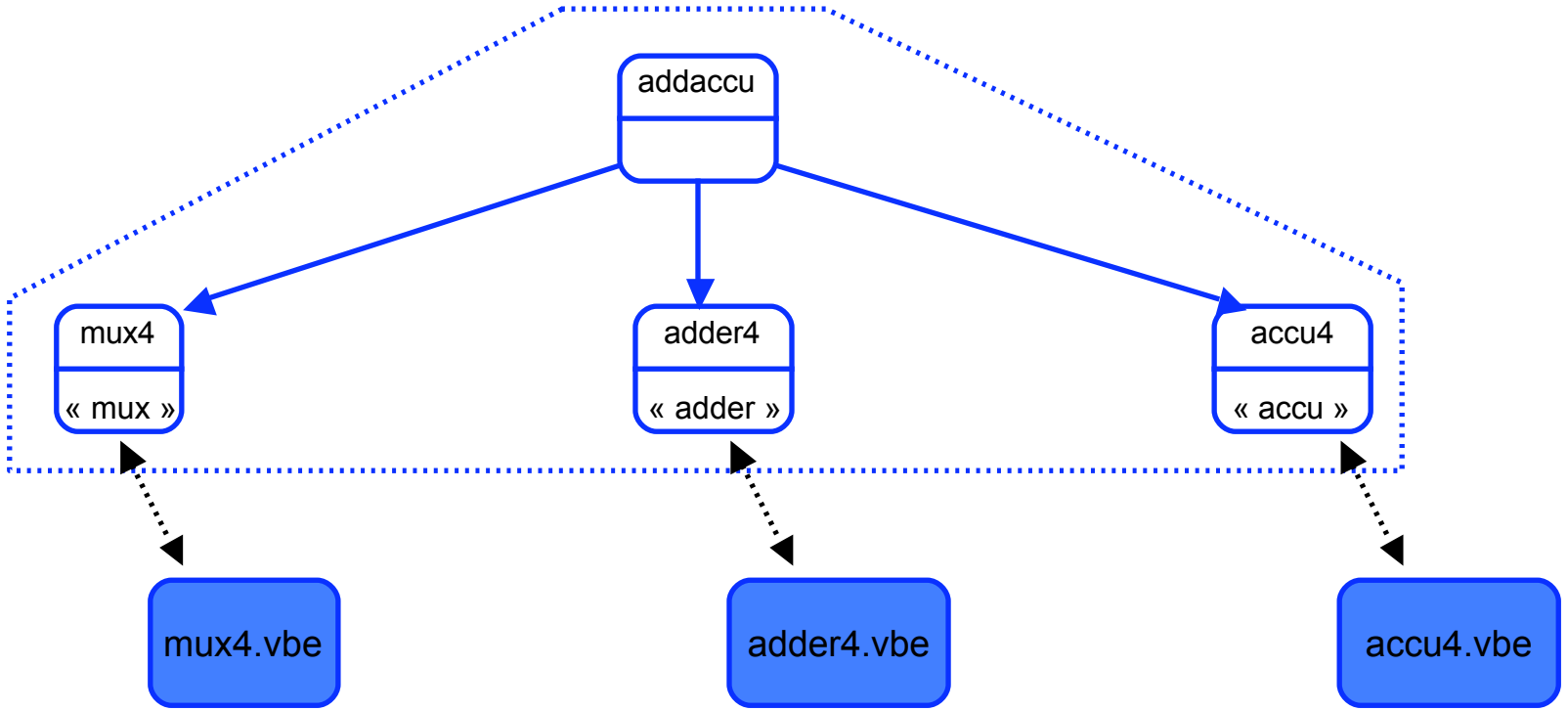
Arbre d'instanciation



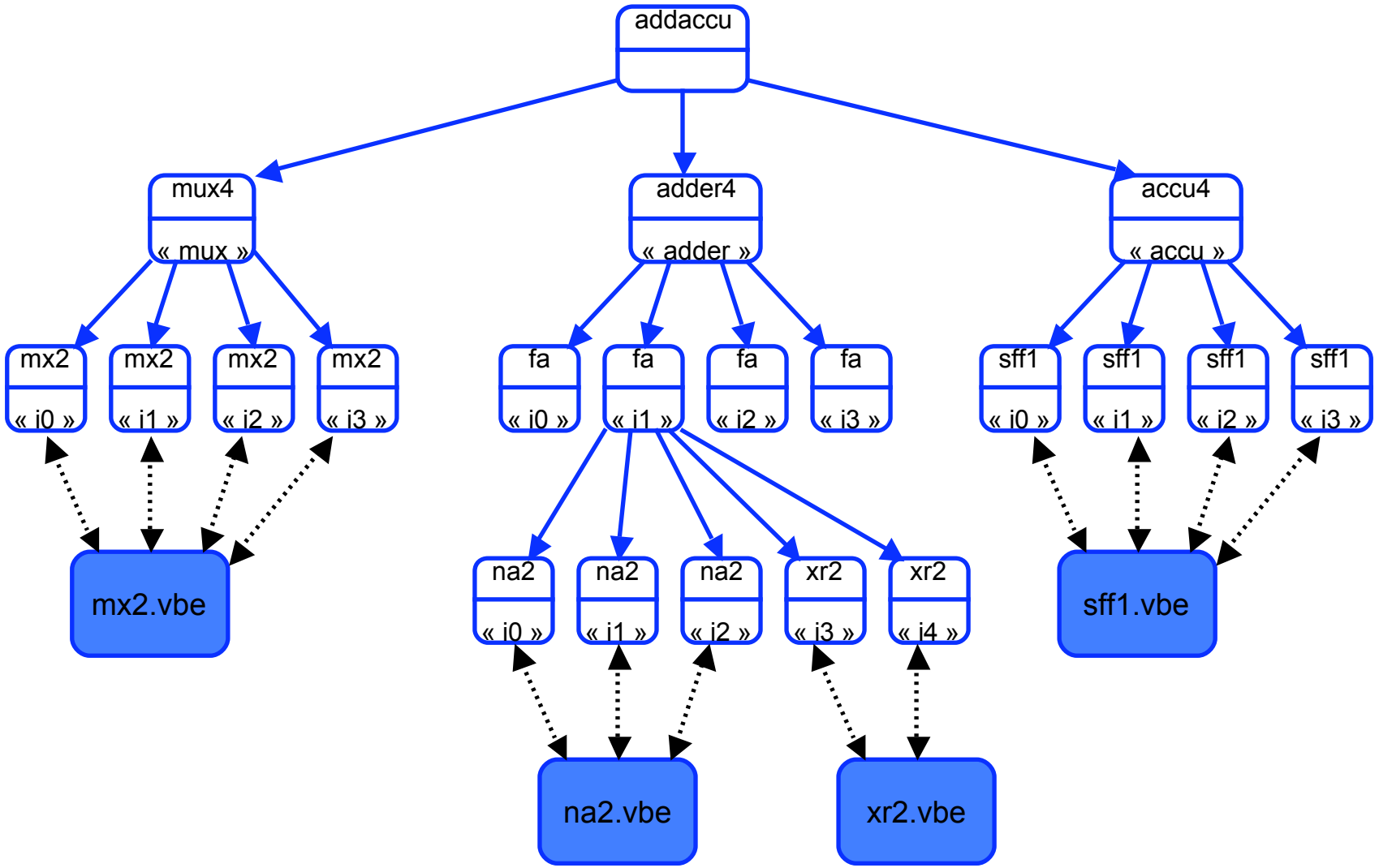
Ce schéma hiérarchique nécessite de définir les modèles structurels de 5 blocs : addaccu, mux4, adder4, accu4, et full-adder

Simulation / 1

Une description structurelle hiérarchique multi-niveaux peut être simulée, à condition de disposer d'un modèle comportemental pour chacune des « feuilles » de l'ar(bre d'instanciation :



Simulation / 2



Plan

- **Conception structurée**
- **Bibliothèque de cellules précaractérisées**
- **Décomposition structurelle pour « addaccu »**
- **Modèle VHDL structurel**

Modèle VHDL « addaccu.vst » / 1

```
entity addaccu is

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

page 1

```
architecture vst of addaccu is

-- déclaration des modèles
component mux4
port(cmd   : in bit ;
     x     : in bit_vector (3  downto 0) ;
     y     : in bit_vector (3  downto 0) ;
     q     : out bit_vector (3  downto 0) ;
     vss   : in bit ;
     vdd   : in bit ) ;

component adder4
port(cin   : in bit ;
     x     : in bit_vector (3  downto 0) ;
     y     : in bit_vector (3  downto 0) ;
     q     : out bit_vector (3  downto 0) ;
     cout  : out bit ;
     vss   : in bit ;
     vdd   : in bit ) ;
```

page 2

Modèle VHDL « addaccu.vst » / 2

```
component accu4
port(ck      : in bit ;
     d       : in bit_vector (3  downto 0) ;
     q       : out bit_vector (3  downto 0) ;
     vss     : in bit ;
     vdd     : in bit ) ;

-- déclaration des signaux
signal muxout bit_vector (3  downto 0) ;
signal regout bit_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 ) ;
```

page 3

```
adder : adder4 port map( x => muxout ,
                         y => b ,
                         y => regout ,
                         q => s ,
                         cin => vss ,
                         vss => vss ,
                         vdd => vdd ) ;
accu : accu4 port map( ck => ck ,
                       d => s ,
                       q => regout ,
                       vss => vss ,
                       vdd => vdd ) ;

end vst ;
```

page 4