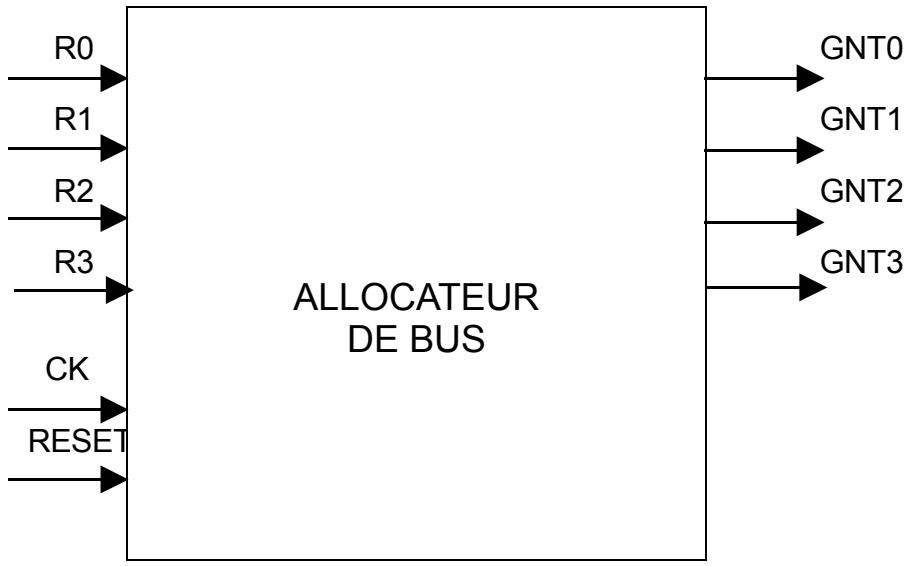
PHY 568

TP « Chaîne de CAO / VLSI »

Le but de ce TP est de réaliser le dessin des masques, en technologie CMOS, d'un composant matériel réalisant un allocateur de bus. Le point de départ est une description comportementale de niveau RTL (Register Transfer Level) écrite en langage VHDL. On utilisera les outils de synthèse, placement, routage, vérification et simulation de la chaîne de CAO/VLSI ALLIANCE, développée par le laboratoire LIP6. On utilisera également la bibliothèque de cellules pré-caractérisées SXLIB.

Cet allocateur de bus est une variante de celui étudié au début du cours, dans le cadre de la PC « automates ». On considère quatre processeurs P0, P1, P2, P3 utilisant le même bus de communication pour accéder à la mémoire. On cherche à réaliser un « allocateur de bus » équitable, se comportant comme un automate de Moore. Chacun des processeurs dispose d'un signal de requête Ri, actif à l'état haut, pour demander à utiliser le bus. Les 4 signaux de requête (R0, R1, R2, R3), sont indépendants et peuvent être actifs simultanément.
En réponse, l'allocateur alloue le bus à l'un des processeurs et l'indique par l'intermédiaire d'un signal Ai actif à l'état haut (A0, A1, A2, A3).
Quand un processeur a fini d'utiliser le bus, il le signale en remettant le signal Ri à zéro. Quand le bus est libéré par un processeur, il ne peut pas être immédiatement re-alloué, et l'allocateur attend au moins un cycle avant de l'allouer de nouveau.
Pour garantir un arbitrage équitable, on utilise un algorithme de priorité tournante de type « round-robin » : le processeur qui obtient le bus devient le moins prioritaire pour la prochaine allocation :

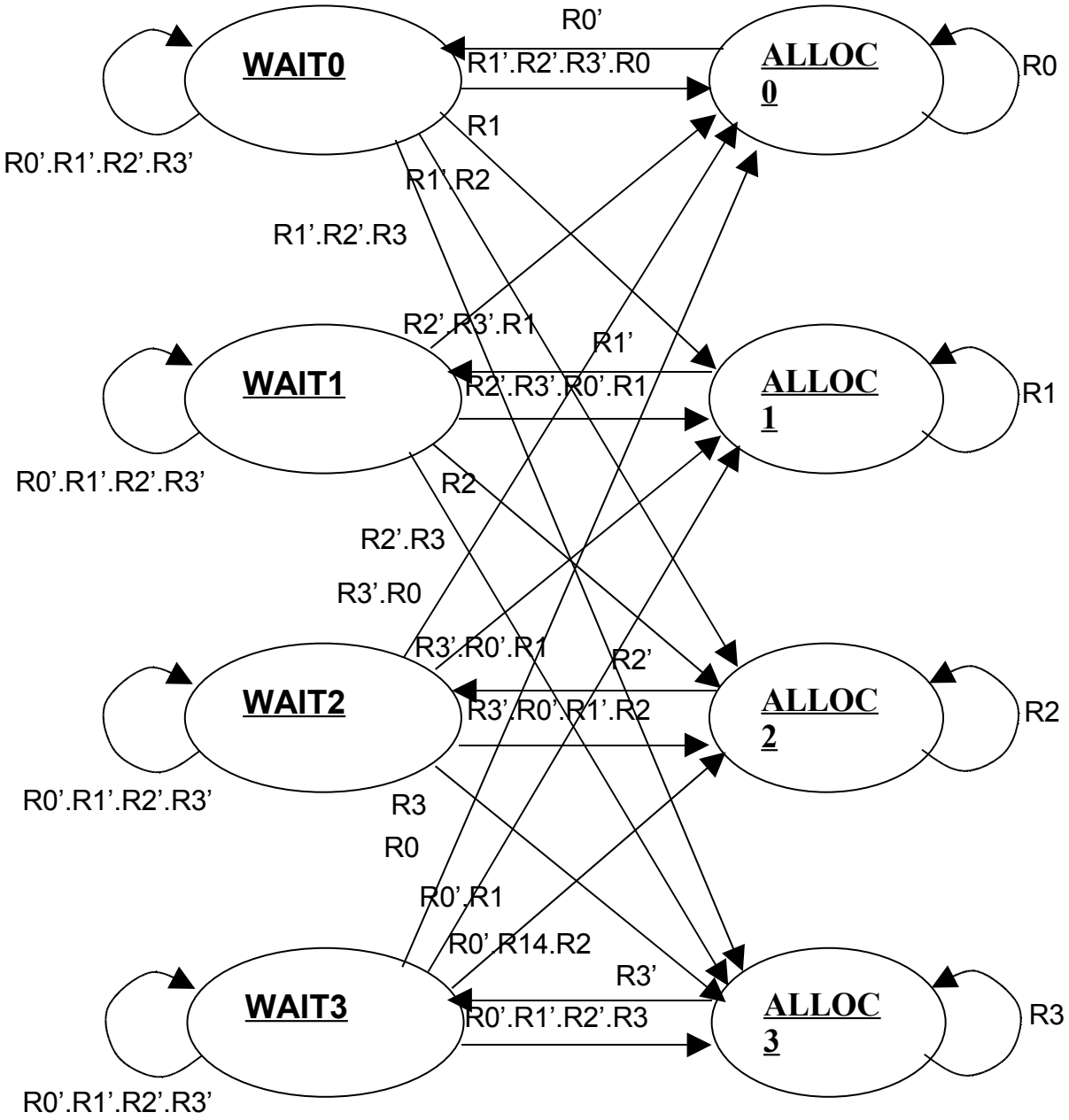
- si P0 vient d'obtenir le bus la priorité est P0 < P1 < P2 < P3
- si P1 vient d'obtenir le bus la priorité est P1 < P2 < P3 < P0
- si P2 vient d'obtenir le bus la priorité est P2 < P3 < P0 < P1
- si P3 vient d'obtenir le bus la priorité est P3 < P0 < P1 < P2



On définit les huit états de l'automate de la façon suivante :

- WAIT0 : L'allocateur attend une requête, P0 est le moins prioritaire

- ALLOC0 : Le bus est alloué à P0, l'allocateur attend la libération du bus.
- WAIT1 : L'allocateur attend une requête, P1 est le moins prioritaire
- ALLOC1 : Le bus est alloué à P1, l'allocateur attend la libération du bus.
- WAIT2 : L'allocateur attend une requête, P2 est le moins prioritaire
- ALLOC2 : Le bus est alloué à P2, l'allocateur attend la libération du bus.
- WAIT3 : L'allocateur attend une requête, P3 est le moins prioritaire
- ALLOC3 : Le bus est alloué à P3, l'allocateur attend la libération du bus.



Pour tous les outils logiciels utilisés dans ce TP, il existe des manuels en ligne, qu'il est fortement recommandé de consulter.

Par ailleurs, il faut définir le répertoire de travail, qui sera utilisé par les différents outils de la chaîne ALLIANCE pour ranger les fichiers générés. Il est recommandé de créer un répertoire spécial pour ce TP.

Avant de commencer, il faut copier dans le répertoire de travail les 4 fichiers (qui sont sur le site Web):

- allocateur.fsm
- processeur.vbe
- systeme.vst
- stimuli.pat

Q1) Synthèse d'automate

Le fichier « allocateur.fsm » contient une description comportementale de type « Register Transfer Level » de l'allocateur de bus, en langage VHDL. Cette description est délibérément incomplête. Complétez, en utilisant votre éditeur de texte préféré, et en exploitant le graphe ci-dessus, le fichier « allocateur.fsm » qui vous est fourni.

Synthétisez cet automate en utilisant l'outil SYF, pour génèrer un fichier « allocateur.vbe », qui est une description VHDL comportementale de type « assignations concurrentes ». Il est recommandé de consulter le manuel en ligne de l'outil SYF avant de lancer la synthèse. On utilisera l'option –o permettant de choisir le codage « one-hot » (pour encoder les états sur huit bits).

- **man syf**
- **syf –o allocateur allocateur**

Editez le fichier « allocateur.vbe » de façon à identifier le registre d'état de ce composant. On pourra pour cela consulter le manuel décrivant le format .vbe.

> man vbe

Q2) Simulation VHDL

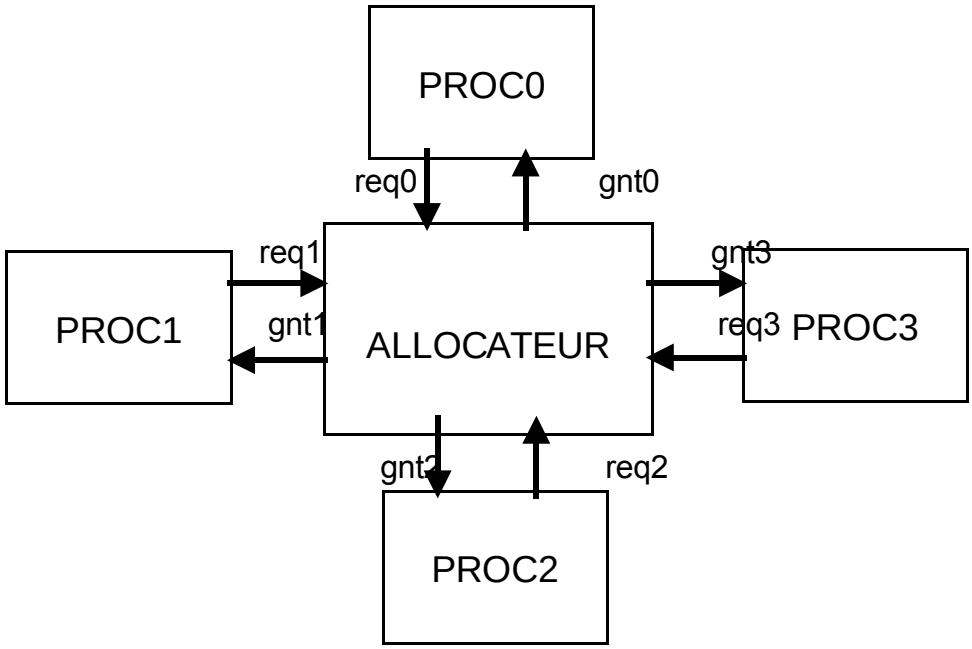
On cherche maintenant à valider ce fichier « allocateur.vbe » par simulation. On utilise pour cela le simulateur ASIMUT, et le test-bench « systeme.vst ».
Un test-bench est construit en interconnectant le composant à valider (ici l'allocateur de bus) avec un ensemble d'autres composants dont le comportement est connu. Le fichier « systeme.vst contient donc une description VHDL structurelle d'un petit système comportant 4 processeurs identiques, connectés à l'allocateur de bus. Editez le fichier « systeme.vst » pour voir comment on écrit une description structurelle en VHDL, et pour identifier les noms des signaux qu'on cherchera à observer au cours de la simulation.

> man vst

Le comportement du composant processeur est décrit dans le fichier « processeur.vbe », qui vous est fourni. Il s'agit d'un modèle de processeur très simplifié, puisque chaque processeur se contente de répéter indéfiniment la séquence suivante comportant trois états:

1. demande d'allocation du bus, par activation du signal REQ, jusqu'à obtention du signal GNT. Le nombre de cycles d'attente est à priori quelconque, et dépend évidemment du bon vouloir de l'allocateur.
2. utilisation du bus pendant un cycle.

3. libération du bus par désactivation du signal REQ pendant un cycle, et retour à l'étape 1.



Ce test-bench possède 4 ports externes : VDD, VSS, CK, RESET, sur lesquels il faut appliquer des stimuli. Le fichier « stimuli.pat » permet de définir les stimuli à appliquer sur les entrées du système, et permet également de définir quels signaux internes on souhaite observer. Consultez le manuel du format de fichier « .pat », et visualisez le contenu de ce fichier en utilisant l'outil XPAT.

- **man pat**
- **xpat**

Créez dans votre répertoire un fichier texte portant le nom CATAL et contenant le nom des deux composants instanciés dans le système.
Lancez la simulation sous ASIMUT, en utilisant l'option –zd (simulation en « zéro délay »). On utilise également le format « .pat » pour stocker les résultats de la simulation dans le fichier « result.pat».

- **man asimut**
- **asimut – zd systeme stimuli result**
- **xpat**

Quelle est la fréquence d'accès au bus pour un processeur particulier, dans ce contexte où tous les processeurs sont en compétition pour l'accès au bus ?

Q3) Synthèse logique

On cherche maintenant à générez automatiquement un schéma en portes logiques pour le composant « allocateur », en utilisant les cellules précaractérisées de la bibliothèque SXLIB. Consultez le contenu de la bibliothèque de cellules SXLIB, puis visualisez quelques cellules sous l'éditeur graphique GRAAL. Pour cela, il faut définir le chemin d'accès au répertoire contenant les fichiers décrivant les cellules de la

bibliothèque SXLIB. On utilise la variable d'environnement MBK_CATA_LIB pour définir ce répertoire.
On visualisera successivement les cellules na2_x1, na2_x4 et sff1_x4.
Quelles sont les caractéristiques topologiques communes à toutes les cellules ?
A quoi correspond l'extension du nom de la cellule ?

- **man sxlib**
- **graal**

Pour la synthèse, on utilisera successivement les outils BOOM, PROOF et BOOG.

L'outil BOOM prend en entrée une description comportementale VHDL de type « assignations concurrentes » (fichier allocateur.vbe), et génère une autre description comportementale de même type. Il optimise le réseau Booléen sous-jacent à cette description comportementale, en utilisant des techniques de factorisation et de simplification visant à minimiser le nombre de littéraux.
Attention : commencez par donner un nom différent au fichier de sortie, de façon à comparer les deux descriptions (avant et après optimisation).

- **man boom**
- **boom allocateur allocateur_opt**

Utilisez l'outil PROOF pour vérifier que les deux descriptions sont équivalentes, avant de relancer l'optimisation, en donnant au fichier de sortie le même nom que celui du fichier d'entrée.

- **man proof**
- **proof –d allocateur allocateur_opt**
- **boom allocateur allocateur**

L'outil BOOG prend en entrée une description comportementale VHDL de type « assignations concurrentes » (fichier allocateur.vbe), et génère une description structurelle VHDL instanciant des cellules SXLIB (fichier allocateur.vst).

- **man boog**
- **boog allocateur**

Visualisez le schéma ainsi généré en utilisant l'outil de visualisation XSCH.

> xsch

Validez le schéma ainsi généré, en simulant avec ASIMUT le test-bench « systeme.vst », en applicant les mêmes stimuli, mais en utilisant cette fois la description structurelle « allocateur.vst » au lieu de la description comportementale « allocateur.vbe ».
Il faut pour cela modifier le fichier CATAL. Le fichier CATAL contient les noms de tous les composants pour lesquels il existe une description comportementale. Si on supprime la ligne « allocateur » dans le fichier CATAL, le simulateur utilisera la description structurelle du composant l'allocateur, ainsi que les descriptions comportementales des cellules SXLIB, (fichiers du type « na2_x1.vbe »).
Visualisez avec l'outil XPAT le fichier résultant « result_s.pat, et vérifiez que le schéma en portes logiques de l'allocateur a le même comportement que la spécification comportementale initiale.

- **asimut –zd systeme stimuli result_s**

- **xpat**

Q4) Placement/Routage

On cherche maintenant à générer le dessin des masques (vue physique) du composant « allocateur », à partir du schéma en portes.

Utilisez l'outil de placement automatique OCP pour générer le placement des cellules constituant le composant allocateur. Cet outil prend en entrée le fichier « allocateur.vst » décrivant le schéma d'interconnexion entre les cellules, ainsi que le dessin des masques de chaque cellule instanciée dans le schéma (fichiers du type « na2_x1.ap »). Il génère un fichier « allocateur_nr.ap », contenant une description du dessin des masques du composant allocateur. A ce stade, les cellules sont aboutées les unes aux autres, mais les fils d'interconnexion ne sont pas dessinés. Visualisez sous l'éditeur graphique GRAAL le résultat du placement.

- **man ocp**
- **ocp allocateur allocateur_nr**
- **graal allocateur_nr**

Utilisez l'outil de routage NERO pour générer automatiquement le routage (c'est à dire le dessin des fils d'interconnexion entre cellules). Cet outil prend en entrée le fichier « allocateur.vst » décrivant le schéma d'interconnexion entre les cellules, ainsi que le fichier « allocateur_p.ap », décrivant le placement des cellules, et les fichiers contenant le dessin des masques des cellules (fichiers du type « na2_x1.ap »). Il génère un fichier « allocateur.ap », contenant le dessin des masques final du composant allocateur (c'est à dire avec les fils d'interconnexion). Visualisez sous l'éditeur graphique GRAAL le résultat du routage.

- **man nero**
- **nero –p allocateur_nr allocateur allocateur**
- **graal allocateur**

1. Utilisez la fonction PEEK (dans le menu TOOLS de GRAAL) pour visualiser le dessin des masques, jusqu'aux transistors contenus dans les cellules.
2. Utilisez la fonction EQUI (dans le menu TOOLS de GRAAL), pour visualiser le signal d'horloge.
3. Utilisez la fonction DRUC (dans le menu TOOLS de GRALL) pour vérifier que les outils de placement/routage automatique n'ont pas introduit de violation des règles de dessin.
4. Utilisez les fonctions d'édition graphique de GRAAL (dans le menu EDIT de GRAAL) pour modifier un fil d'interconnexion et introduire volontairement une erreur de dessin (par exemple deux fils Metal2 distants de moins de 3 lambdas), et vérifiez que cette erreur est détectée.

Q5) Extraction et vérification

On souhaite maintenant vérifier le résultat du placement/routage. Pour cela, on utilise l'outil d'extraction COUGAR, qui prend en entrée une description physique représentant le dessin des masques (fichier « allocateur .ap ») et génère une description structurelle extraite du dessin des masques .
Si la description physique est structurée (ce qui est le cas de l'allocateur), le schéma extrait est lui-même structuré, et on obtiendra un schéma en portes logiques.

- **man cougar**
- **cougar allocateur allocateur_x**

Pour vérifier que l'outil de routage automatique n'a pas créé de court-circuit , et pour s'assurer que tous les signaux sont bien connectés, on utilise l'outil de comparaison de net-list LVX. Cet outil prend en entrée deux description structurelles (généralement la net-list obtenue par synthèse logique, et la net-list extraite du dessin des masques) et vérifie qu'elles sont isomorphes.

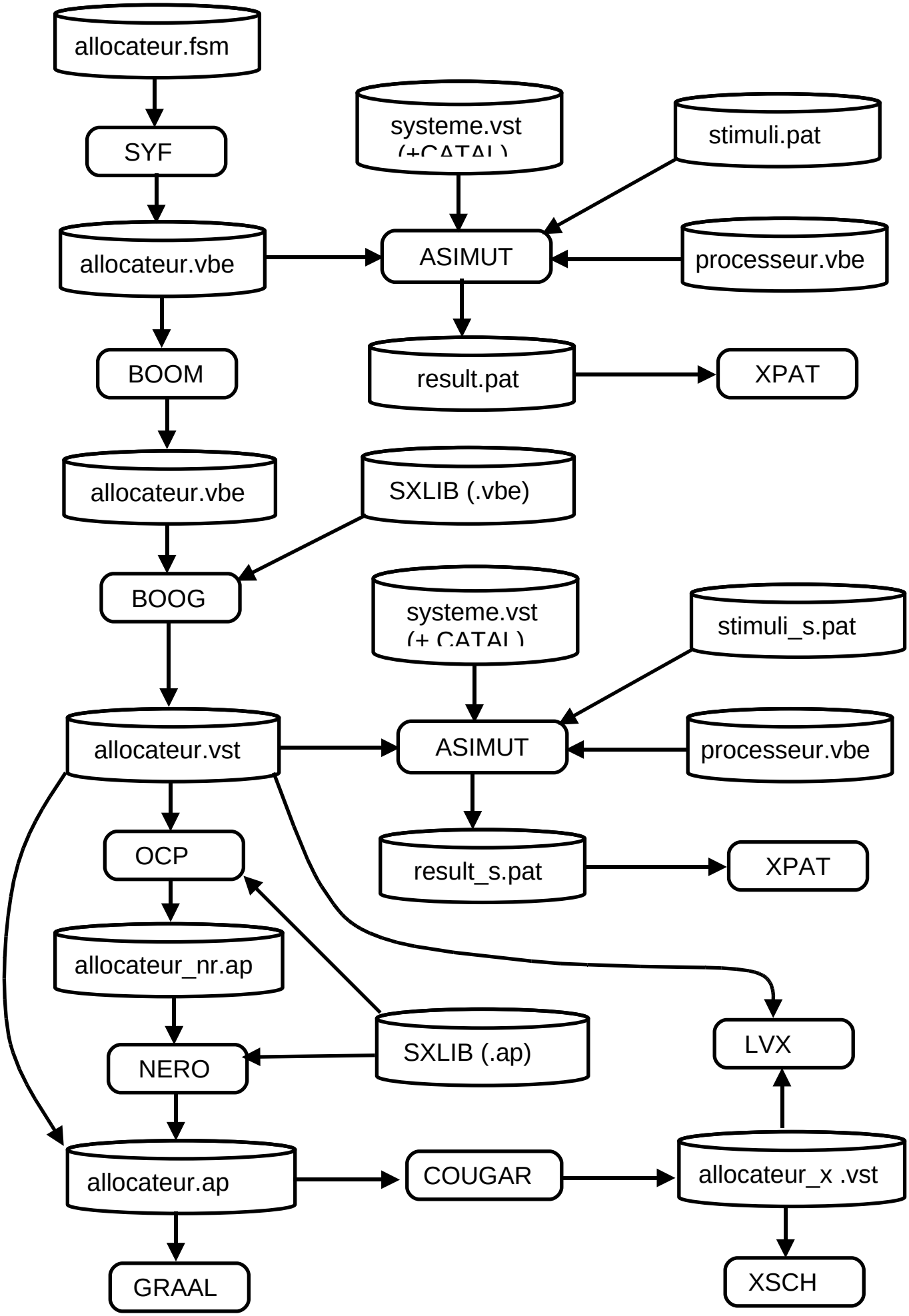
- **man lvx**
- **lvx vst vst allocateur allocateur_x**

On peut également demander à l'extracteur de générer un schéma en transistors, en utilisant l'option -t. On peut aussi demander à l'extracteur de calculer la capacité électrique des fils d'interconnexion, en utilisant l'option –ar. Il faut pour cela définir les paramètres du procédé de fabrication par la variable d'environnement RDS_TECHNO_NAME, qui définit le nom du fichier contenant les valeurs des paramètres.
Le langage VHDL n'étant pas adapté pour la représentation de ces schémas électriques au niveau transistors et capacités, on utilise le format SPICE pour représenter ce schéma extrait du dessin des masques.
Les outils de la chaîne de CAO ALLIANCE sont capables de lire et de générer différents formats de fichiers pour représenter la vue structurelle. Pour définir le format de sortie, on utilise la variable d'environnement MBK_OUT_LO.
Il faut également définir le nom du fichier contenant les modèles de transistors dans la variable d'environnement MBK_SPI_MODEL. Relancez l'extraction pour obtenir un schéma extrait au niveau transistors et capacités.

- **cougar –ac –t allocateur allocateur_t**

Quel est le nombre total de transistors contenus dans le composant allocateur ?

Pour finir, on peut simuler ce schéma en transistors en utilisant le simulateur SPICE. Il faut simplement ajouter les générateurs de tension pour les signaux VDD et VSS, et les formes d'ondes pour les signaux CK et RESET.


allocateur.fsm
SYF
allocateur.vbe
BOOM
allocateur.vbe
BOOG
allocateur.vst
OCP
allocateur_nr.ap
NERO
allocateur.ap
GRAAL
systeme.vst
(+CATALI)
stimuli.pat
ASIMUT
processeur.vbe
result.pat
XPAT
SXLIB (.vbe)
systeme.vst
(+ CATALI)
stimuli_s.pat
ASIMUT
processeur.vbe
result_s.pat
XPAT
SXLIB (.ap)
LVX
COUGAR
allocateur_x .vst
XSCH