

# TP2 : Modélisation Structurelle VHDL

1. Objectifs
2. A) Génération procédurale des stimuli
3. B) Description structurelle
4. C) Simulation
5. Compte-rendu

## Objectifs

Le but de cette seconde séance de TP est d'utiliser le langage VHDL pour décrire, puis simuler une description structurelle du composant *addaccu*.

Le but final est d'aboutir à une description structurelle détaillée, c'est à dire un schéma en portes logiques, utilisant une bibliothèque de cellules pré-caractérisées. Ceci sera fait dans le TP3.

Dans ce TP2, nous allons franchir une étape intermédiaire, en décomposant le circuit *addaccu* en trois sous-blocs fonctionnels : le bloc **mux4**, le bloc **adder4**, et le bloc **accu4**.

Un deuxième objectif de ce TP2 est d'introduire le langage de description de stimuli **genpat**.

Commencez par créer un répertoire de travail *tp2* pour archiver les fichiers de ce TP.

## A) Génération procédurale des stimuli

Dans le TP1, vous avez écrit "à la main" le fichier *stimuli.pat* décrivant les valeurs à appliquer sur les entrées du circuit. Cette méthode est assez fastidieuse, et elle est source d'erreurs. Pour faciliter la description des scénarios de simulation, vous pouvez utiliser le langage **genpat**, qui est un ensemble de fonctions écrites en langage C, et qui apporte au concepteur de circuit toute la puissance d'expression du langage C (boucles, expression conditionnelles, etc.) pour décrire les scénarios de simulation.

On rappelle que vous pouvez obtenir des informations détaillées sur n'importe quel outil de la chaîne de CAO *ALLIANCE* en tapant (par exemple) la commande :

```
>man genpat
```

Les noms des fonctions du langage **genpat** sont en majuscules. La fonction la plus importante est la fonction **AFFECT()** qui permet d'assigner une nouvelle valeur à un signal particulier X à une certaine date T. Cette fonction permet donc de spécifier des *événements*.

Il faut donc écrire un fichier *stimuli.c* respectant la syntaxe du langage C, et c'est l'exécution de ce programme C qui générera le fichier *stimuli.pat* utilisable par **asimut**. Vous trouverez ci-dessous un exemple de fichier *stimuli.c*, que vous devez compléter et enrichir.

```
#include <genpat.h>
#include <stdio.h>
/*-----*/
/* Fonction utilitaire */
/*-----*/
char *inttostr(int entier)
{
    char *str;
    str = (char *) mbkalloc (32 * sizeof (char));
```

```

    sprintf (str, "%d",entier);
    return(str);
}
/*-----*/
/* programme principal */
/*-----*/
main ()
{
    int i;
    int time;
    DEF_GENPAT("stimuli");
    /* interface */
    DECLAR ("a", ":2", "X", IN, "3 downto 0", "");
    DECLAR ("b", ":2", "X", IN, "3 downto 0", "");
    DECLAR ("sel", ":2", "B", IN, "", "");
    DECLAR ("ck", ":2", "B", IN, "", "");
    DECLAR ("s", ":2", "X", OUT, "3 downto 0", "");
    DECLAR ("vdd", ":2", "B", IN, "", "");
    DECLAR ("vss", ":2", "B", IN, "", "");
    /* événements */
    AFFECT ("0", "sel" , "0b0");
    AFFECT ("0", "vdd" , "0b1");
    AFFECT ("0", "vss" , "0b0");
    time = 0;
    for (i=0; i<4; i++) {
        AFFECT ( inttostr(time), "ck", "0b0" );
        AFFECT ( inttostr(time), "a", inttostr(i));
        AFFECT ( inttostr(time), "b", "0xF");
        time += 10000;
        AFFECT ( inttostr(time), "ck", "0b1" );
        time += 10000;
    }
    SAV_GENPAT ();
}

```

Pour générer le fichier *stimuli.pat* il faut lancer la commande :

```
>genpat stimuli
```

Vérifiez que le fichier *stimuli.pat* généré correspond à ce que vous attendez en utilisant l'outil de visualisation de chronogrammes **xpat**.

```
>xpat stimuli
```

Validez votre fichier de stimuli en simulant son exécution sur le modèle comportemental du composant *addaccu* provenant du TP1 :

```
>asimut -b -zd addaccu stimuli result
```

L'option **-zd** signifie que vous souhaitez que **asimut** effectue une simulation zéro-délay : même si la description comportementale contient des constructions **AFTER**, celles-ci ne seront pas prises en compte.

## B) Description structurelle

On va maintenant décrire le composant *addaccu* comme l'instanciation de trois blocs fonctionnels : le bloc **mux4**, le bloc **adder4**, et le bloc **accu4**, dont les interfaces et les comportements sont prédéfinis.



- Le bloc **adder4** est un additionneur 4 bits, avec report entrant et report sortant.

- Le bloc `mux4` est un multiplexeur 4 bits qui sélectionne un mot parmi 2.
- Le bloc `accu4` est un registre 4 bits à échantillonnage sur front montant de CK.

Vous pouvez consulter le modèle comportemental data-flow de chacun de ces blocs en cliquant sur le nom du bloc.

Vous devez maintenant écrire en VHDL la description structurelle du composant `addaccu`. Le fichier VHDL comportera l'extension `.vst` (Vhdl SStructuel)

Bien que le langage VHDL permette en principe de décrire un composant matériel en "mélangeant" dans une même description des assignations concurrentes et des instanciations, le simulateur **asimut** impose la règle suivante:

- Une description comportementale data-flow (de type `.vbe`) ne contient que des assignations concurrentes.
- Une description structurelle (de type `.vst`) ne contient que des instanciations d'autres composants.

La construction VHDL qui permet d'instancier un composant dans un autre est la construction "PORT MAP", mais pour plus de précisions, vous avez intérêt à consulter le man du format `.vst` :

```
>man vst
```

## C) Simulation

Le simulateur **asimut**, comme tous les simulateurs VHDL est capable de simuler aussi bien une description comportementale (telle que le fichier `addaccu.vbe`) qu'une description structurelle (telle que le fichier `addacu.vst`), à condition que les modèles comportementaux des blocs instanciés soient disponibles.

Vous devez donc créer dans le répertoire `tp2` les trois fichiers `adder4.vbe`, `mux4.vbe`, et `accu4.vbe`. Vous pouvez pour cela importer les modèles fournis ci-dessus par simple copier/coller.

Dans le cas d'une description structurelle, certains blocs instanciés peuvent être eux-mêmes décrits de façon structurelle. On parle alors de description hiérarchique "multi-niveaux", et le nombre de niveaux peut être quelconque. Il faut donc indiquer au simulateur quels sont les blocs "terminaux", pour lesquels il existe une description comportementale. Le simulateur **asimut** trouve cette information dans le fichier `CATAL`. Ce fichier est un fichier texte contenant les noms des blocs terminaux (un composant par ligne), suivi de la lettre C :

```
adder4  C
accu4   C
mux4    C
```

Ce fichier `CATAL` doit se trouver dans le répertoire de travail.

Vous pouvez maintenant appliquer sur cette description structurelle les stimuli définis dans la partie A :

```
>asimut addaccu stimuli result
```

L'absence de l'option `-b` indique au simulateur qu'il s'agit d'une description structurelle, et qu'il doit utiliser le fichier `CATAL`.

Vous devez obtenir les mêmes résultats de simulation que dans la partie "simulation zero-delay" du TP1.

## Compte-rendu

Il vous est demandé un rapport d'une page, au format `.pdf`. Vous joindrez les fichiers `stimuli.c` et `addaccu.vst` en annexe.