TP4 : Le chemin de données du circuit AM2901

Le chemin de données est formé de la logique régulière du circuit.

Afin de profiter de cette régularité, on utilise les opérateurs vectoriels de la bibliothèque **DP_SXLIB**. Cela permet d'optimiser le schéma en utilisant plusieurs fois le même matériel. Par exemple, les amplificateurs des signaux de commande d'un multiplexeur sur n bits sont partagés par les n bits...

On utilise le langage **STRATUS** pour décrire la structure (i.e. le schéma) du chemin de données.

1 Exemple de description avec Stratus

Considérons le circuit suivant :

Voici la structure du chemin de données correspondante :

Chacune des portes occupe une colonne, une colonne permettant de traiter un ensemble de bits pour un même opérateur. La première ligne représente le bit 3, la dernière le bit 0.

Le fichier **Stratus** correspondant est le suivant :

```
#!/usr/bin/env python
from stratus import *
# definition de la cellule
class circuit ( Model ):
  # declaration des connecteurs
  def Interface ( self ):
    self.a = SignalIn ( "a" , 4 )
    self.b = SignalIn ( "b" , 4 )
    self.c = SignalIn ( "c" , 4 )
    self.v = SignalIn ( "v" , 1 )
    self.cout = SignalOut ( "cout", 1 )
    self.s = SignalOut ( "s" , 4 )
    self.cmd = SignalIn ( "cmd" , 1 )
    self.vdd = VddIn ( "vdd" )
    self.vss = VssIn ( "vss" )
  # instanciation des operateurs
  def Netlist ( self ):
    # declaration des signaux internes
    d_aux = Signal ( "d_aux", 4 )
    e_aux = Signal ( "e_aux", 4 )
    ovr = Signal ( "ovr" , 1 )
    # generation
    Generate ( "DpgenNand2", "instance_nand2_4bits"
             , param = { ?nbit? : 4 }
             )
    # instanciation
    self.instance_nand2_4bits = Inst ( "instance_nand2_4bits"
                                     , map = { ?i0? : Cat ( self.v
                                                          , self.v
```

```
                                                    , self.v
                                                    , self.v )
                                                , ?i1? : self.a
                                                , ?nq? : d_aux
                                                , ?vdd? : self.vdd
                                                , ?vss? : self.vss
                                                }
                                        )
        Generate ( "DpgenOr2", "instance_or2_4bits"
                 , param = { ?nbit? : 4 }
                  )
        self.instance_or2_4bits = Inst ( "instance_or2_4bits"
                                       , map = { ?i0? : d_aux
                                               , ?i1? : self.b
                                               , ?q? : e_aux
                                               , ?vdd? : self.vdd
                                               , ?vss? : self.vss
                                               }
                                        )
        Generate ( "DpgenAdsb2f", "instance_add2_4bits"
                 , param = { ?nbit? : 4 }
                 )
        self.instance_add2_4bits = Inst ( "instance_add2_4bits"
                                        , map = { ?i0? : e_aux
                                                , ?i1? : self.c
                                                , ?q? : self.s
                                                , ?add_sub? : self.cmd
                                                , ?c31? : self.cout
                                                , ?c30? : ovr
                                                , ?vdd? : self.vdd
                                                , ?vss? : self.vss
                                                }
                                         )
```

Comme vous pouvez le voir, il n'a pas été nécessaire de générer *à la main* les différents blocs de ce circuit, ceux ci sont fournis par la bibliothèque *DP_SXLIB*. Il suffit alors de les générer (fonction **Generate** comme montré dans le TP précédent).

Pour connaitre les générateurs de colonne dont vous disposez, consultez la documentation fournie :

?https://www-asim.lip6.fr/recherche/coriolis/doc/en/html/dpgen/index.html

Consultez également la documentation de **Stratus** en cas de besoin :

?https://www-asim.lip6.fr/recherche/coriolis/doc/en/html/stratus/index.html

2 Description du chemin de données dans l'AM2901

Le chemin de données de l'Am2901 peut être schématisé par la figure ci-dessous.

3 Travail à faire

- Etant donné le fichier description en vbe du chemin de données de l'Amd2901, compléter le fichier description en stratus du chemin de données de l'Amd2901.
- Créer le fichier "test_amd2901_dpt.py" correspondant. NOTE : la ram est déjà construite.

- Lancer le fichier :

```
> ./test_amd2901_dpt.py
```

- Valider la liste de signaux de la même manière que pour la partie contrôle.
- Supprimer le fichier **CATAL** et simuler le circuit avec **asimut**.

```
> asimut -zerodelay amd2901_chip pattern resultat
```