TP4 : Le chemin de données de l'AMD2901

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

Afin de profiter de cette régularité, on génère la liste de signaux sous forme d'opérateurs vectoriels (ou

colonnes) via les macro-fonctions de l'outil Stratus.

Cela permet d'économiser de la place en utilisant plusieurs fois le même matériel. Par exemple, le NOT d'un mux de n bits est instancié une seule fois pour ces n bits...

1 Exemple de description avec Stratus

Considérons le circuit suivant : a Voici la structure du chemin de données correspondante : a[3]

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
                                            }
                                     )
```

Ce premier fichier définit votre circuit, enregistrez-le sous le nom "circuit.py". Il faut maintenant créer un autre fichier pour instancier votre circuit :

```
#!/usr/bin/env python
from stratus import *
from circuit import circuit
# creation du circuit
mon_circuit = circuit ( "mon_circuit" )
# creation de l?interface
mon_circuit.Interface ()
# creation de la netlist
mon_circuit.Netlist ()
# sauver les fichiers mon_circuit.vst
mon_circuit.Save ()
```

Enregistrez-le sous le nom "test.py". Changez les droits du fichier afin de le rerendre executable :

```
> chmod +x test.py
```

Puis executez le fichier :

```
>./test.py
```

Si tout se passe bien, vous obtenez le fichier "mon_circuit.vst", dans le cas contraire, et mises à part des erreurs de syntaxe, il se peut que votre environnement soit mal configuré pour Stratus.

Consultez la doc au format html

"file *:/asim/coriolis/share/doc/en/html/stratus/index.html" afin de vous renseigner sur les variables d'environnement à positionner.*

Lorsque vous avez obtenu le fichier, passez à la section "Description de la partie chemin de données".

Note : Stratus étant issu du langage Python, il faut apporter une grande importance à l'indentation.

Un bon conseil, n'utilisez pas de tabulations (ou alors configurez vos éditeurs pour qu'ils transforment automatiquement les tabulations en espaces).

2 Description du chemin de données

Les schémas correspondant à la liste de signaux à réaliser vous sont fournis en annexes.

Compléter le fichier "amd2901_dpt.py"

puis créer le fichier "test_amd2901_dpt.py"

correspondant, pour l'exécuter en utilisant le modus operandi ci-dessous.

NOTE : la ram est déjà construite.

Générer la liste de signaux .vst à partir du fichier .py en lancant 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
```

3 Rapport

Il s'agit simplement de décrire votre travail fait en TP.

Décrivez les différentes étapes menant à la netlist finale pour le digicode et l'Amd2901

(le compteur 5 bits vous est épargné).

Quelles sont les deux manières de concevoir une netlist ? Quels avantages y a-t-il à faire des colonnes d'opérateurs pour le data-path ?...

Inutile de faire un roman. Soyez clairs et concis ! Les répertoires, fichiers et logins devront être mentionnés dans le rapport ainsi que vos noms de binômes. N'oubliez pas de mettre les droits en lecture !