Placement & Routage de Cordic

Contents

Avertissement

L'interface entre la partie de contrôle et le chemin de données n'ont pas été finalisées et ne correspondent donc pas exactement. Pour pouvoir faire l'exercice nous nous concentrerons sur le placement/routage de la du chemin de données seul.

Introduction

L'objectif de ce TME est de réaliser l'implantation physique du circuit Cordic, c'est à dire son dessin des masques ou *layout*. Nous le réaliserons en utilisant deux approches différentes pour en effectuer les comparaisons.

1. Synthèse directe du circuit.
2. Découpage explicite en parties contrôle et opératives (ou chemin de données *datapath*). La partie contrôle en synthétisée, mais le chemin de données sera décrit à l'aide du langage de saisie de *netlist* STRATUS.

En première approche, la comparaison se limitera à la surface totale du circuit ainsi que la longueur totale de fil.

Synthèse logique avec Yosys

YOSYS est une alternative beaucoup plus performante à l'ensemble **`boom`**, **`boog`** & **`loon`**.

La première étape consiste à transformer notre description VHDL en une *netlist* de portes logiques. Nous utiliserons pour cela le logiciel YOSYS. Malheureusement, YOSYS ne prend en entrée que du VERILOG et ne génère en sortie que du BLIF (Berkeley Logic Interchange Format).

Il nous faut donc effectuer une traduction *avant* YOSYS puis une autre *après*. Ces différentes étapes sont présentées dans le diagramme suivant:

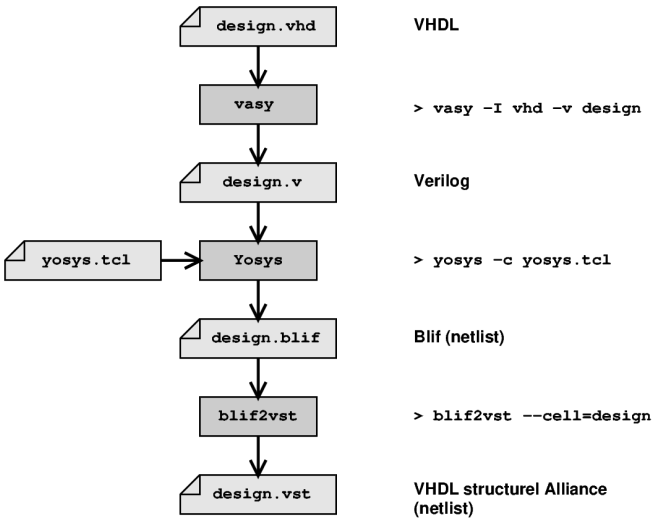

Figure 1: Figure 1 -- Flot de synthèse utilisant YOSYS

Pour automatiser l'ensemble des commandes à donner à YOSYS pour effectuer la synthèse, on passe par un petit script en langage TCL, `yosys.tcl`, dont un exemple est donné ci dessous:

```
set design       cordic_cor
set verilog_file $design.v
set verilog_top  $design
set liberty_file /soc/alliance/cells/sxlib/sxlib.lib
yosys read_verilog          $verilog_file
yosys hierarchy -check -top $verilog_top
yosys synth            -top $verilog_top
yosys dfflibmap -liberty    $liberty_file
yosys abc       -liberty    $liberty_file
yosys clean
yosys write_blif $design.blif
```

Pour l'adapter à vos besoins, simplement éditer la première ligne et changer le nom du *design*.

Note

Nom de fichier et nom du design: le script est écrit de telle façon que le nom du design (de l'ENTITY VHDL) soit le même que le nom du fichier qui le contient. Autrement dit, le fichier `cordic_cor.vhd` doit définir l'ENTITY `cordic_cor`.

Placement & routage du bloc avec CORIOLIS

Avant de pouvoir utiliser CORIOLIS, il est nécessaire d'initialiser l'environnement UNIX:

```
student@pc:~> eval `/soc/coriolis2/etc/coriolis2/coriolisEnv.py`
Using SoC network-wide Coriolis 2 (/soc/coriolis2)
Switching to Coriolis 2.x (Release.Shared)
student@pc:~>
```

Vous pouvez ensuite lancer l'interface graphique **`cgt`**, en indiquant le nom de la *netlist* (i.e. *cell*) à charger:

```
student@pc:~> cgt -V --cell=cordic_cor
```

Pour effectuer le placement, sélectionnez dans les menus:
P&R → Place Block
Pour effectuer le routage, sélectionnez dans les menus:
P&R → Route
Dans le terminal d'ou vous avez lancé l'interface graphique, le placeur puis le routeur auront affiché toute une série d'informations intéressantes, en particuler les dimensions du circuit ainsi que la longueur totale de fils. La taille du circuit, en λ , est donnée sur la ligne `<Box 0l 0l 1850l 1850l>`.

```
o  Configuration of ToolEngine<Etesian> for Cell <cordic_cor>
   - Cell Gauge ..................................................... <sxlib>
   - Place Effort ..................................................... 2
   - Update Conf ...................................................... 2
   - Spreading Conf ................................................... 2
   - Routing driven ..................................................... false
   - Space Margin ..................................................... 5%
   - Aspect Ratio ..................................................... 100%
   - Bloat model ...................................................... disabled
o  Creating abutment box (margin:5% aspect ratio:100% g-length:1290.6)
   - Bloat space margin: 0%.
   - <Box 0l 0l 1850l 1850l>
   - GCell grid: [37x37]
```

La lngueur de fil totale est indiquée ici, c'est le *Wire length completion ratio*, exprimé lui aussi en λ .

```
o  Running Negociate Algorithm
   o  Negociation Stage.
      <queue:00012257>
      <event:00023112 remains:00000000>
   o  Repair Stage.
      <repair.queue:00000000>
o  Computing statistics.
   - Processeds Events Total ..................................... 23113
   - Unique Events Total ......................................... 12811
   - # of GCells ................................................. 1369
   - Track Segment Completion Ratio ..................... 100% [12811+0]
   - Wire Length Completion Ratio ....................... 100% [596946+0]
   - Wire Length Expand Ratio ........................... 4.92% [min:568971]
   - Done in ..................................................... 1.82s, 11.4Mb
   - Raw measurements ........................... 1.81689s, +11660Kb/974.3Mb
```

Réalisation avec un chemin de données

Pour décrire le chemin de données (*datapath*), nous allons utiliser le langage de saisie de *netlist* STRATUS, qui est fourni avec CORIOLIS.

- Documentation de Coriolis

La documentation se décompose en deux parties, celle sur STRATUS, le langage lui-même et celle sur DPGEN, les opérateurs de chemin de données.
Le langage STRATUS est un jeu de classes PYTHON, il est donc possible et même recommandé d'utiliser toutes les fonctionalités proposées par ce langage.

Note

STRATUS n'est absolument pas limité à la description de chemin de données. Il peut être utilisé pour tout type de *netlists*.

Votre travail va consister, en partant de la description VHDL et du schéma du chemin de données, à mettre en correspondance chaque élement avec un opérateur donné dans DPGEN. C'est un travail similaire à celui que fait le synthétiseur logique YOSYS.

Convention de nommage pour les signaux (ou *Net*)

Il faut bien distinguer le **nom du signal**, soit la *chaîne de caratère* `'x'` passée en argument à la fonction `Signal()` et **la variable Python** contenant l'*objet* représentant le signal `x`. Pour des raisons de clarté du code, on donne à la *variable* le même nom que celui du signal (la chaîne de caractère). Comme on pourra le voir dans l'exemple 2, la variable PYTHON admet la *notation en tranche* (ou *slice*) qui permet de sélectionner tout ou partie des bits composant un signal (`x[15]` ou `x[7:4]`).

```
x       = Signal( 'x'       , 16 )
x_sra_1 = Signal( 'x_sra_1', 16 )
```

Exemple 1: Multiplexeur 8 vers 1.

La bibliothèque d'opérateur n'offre pas de multiplexeur 8 vers 1, il faut donc en construire un à partir de multiplexeurs 2 vers 1.

Remarques:

1. Comme nous sommes en logique dual-CMOS, les portes complémentées sont plus petites que les celles non-complémentées. On utilisera donc, dans autant de couches que possible, les opérateurs `nmux2`.
2. La commande du multiplexeur est *encodée*, c'est à dire que pour huit entrées, il nous faut une commande de trois bits. La façon dont la partie opérative est construite impose des contraintes sur la partie de contrôle qui doit être capable de fournir les commandes dans le format approprié.

Description en langage VHDL

```
x_sra_i <=      x       when i_p=0  -- 0b000
          else x_sra_1 when i_p=1  -- 0b001
          else x_sra_2 when i_p=2  -- 0b010
          else x_sra_3 when i_p=3  -- 0b011
          else x_sra_4 when i_p=4  -- 0b100
          else x_sra_5 when i_p=5  -- 0b101
          else x_sra_6 when i_p=6  -- 0b110
          else x_sra_7 when i_p=7  -- 0b111
```

Implantation matérielle (opérateurs)

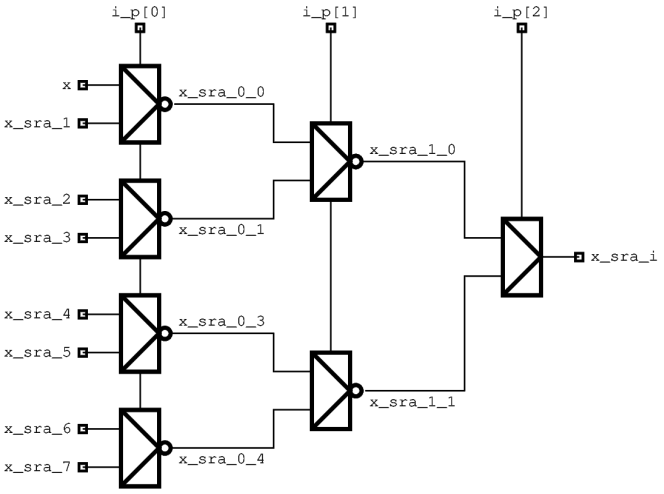

Figure 2: Figure 2 -- Construction d'un multiplexeur 8 vers 1.

Exemple 2: Réorganisation des différents bits d'un signal

Obtenir un signal aux bits *décalés* par rapport à un autre ne correspond pas réellement à du matériel, cela correspond juste à un câblage particulier des fils. Pour l'indiquer au langage STRATUS nous procédons de la façon suivante:

1. Un second signal `x_sra_1` est déclaré.
2. La fonction `Cat()` permet d'assembler un vecteur de bits à partir de bits ou de portion d'autre signaux vectorisés.
3. Enfin, la méthode `Alias()` de l'objet signal permet de dire que celui-çi n'est pas un net indépendant, mais simplement un synonyme ou *alias* d'un autre (utilisation de la *notation en tranche*).

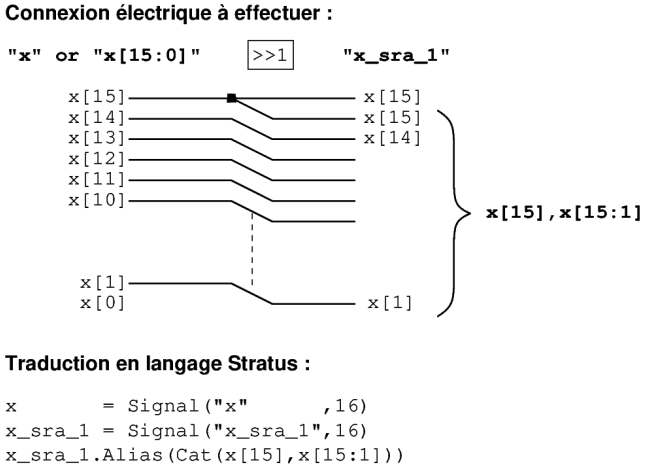

Figure 3: Figure 3 -- Réorganisation des bits d'un signal.

Base du fichier STRATUS du chemin de données

Le script donné ci après vous fourni la base de départ pour décrire le chemin de données. Quelques remarques concernant l'organisation de ce script:

1. Nous définissons une classse dérivée de `Model` (classe de base fournie par STRATUS). Nous devons surcharger les méthodes `Interface()`, `Netlist()` et optionellement `Layout()`.
2. Comme indiqué précedemment, par convention, les objets *signal* (ou *net*) sont stockés dans des variables au nom identique.
3. Pour les instances nous utiliseront un dictionnaire membre de la classe (`self.instances`) pour pouvoir retrouver facilement les instances, qui vont être créé dans `Netlist()` par leur nom dans `Layout()`.

   Nous utiliserons, pour le nommage des instances, une convention similaire à celle des signaux : le nom (et sa clé dans le tableau) seront identique à celle du signal qu'elle génère.
4. Dans la méthode `Layout()`, on se restreindra à un placement sur une seule ligne des opérateurs. On essayera d'optimiser le placement des opérateurs, ce qui, dans notre cas, se résume a changer leur ordonnancement.

Ce script peut être exécuté des différentes façons suivantes:

1. Directement, après l'avoir rendu exécutable (point d'entrée: `__main__`):

```
etudiant@pc:~> chmod u+x cordic_dp.py
etudiant@pc:~> ./cordic_dp.py
```

2. Directement, en passant par l'interpréteur PYTHON (point d'entrée: `__main__`):

```
etudiant@pc:~> python cordic_dp.py
```

3. En utilisant l'argument `script` de **`cgt`** en mode texte (point d'entrée: `ScriptMain()`):

```
etudiant@pc:~> cgt -V --text --script=cordic_dp
```

4. Par le menu **Tools** → **Python Script** de **`cgt`** en mode graphique (point d'entrée: `ScriptMain()`).

```
#!/usr/bin/env python

import sys
from   stratus import *


class Cordic_DP ( Model ):

    def Interface ( self ):
        print 'Cordic_DP.Interface()'
        self.x_p        = SignalIn ( 'x_p'       , 8 )
        self.y_p        = SignalIn ( 'y_p'       , 8 )

       # Interface a completer.
       # ...

        self.ck         = CkIn     ( 'ck'  )
        self.vdd        = VddIn    ( 'vdd' )
        self.vss        = VddIn    ( 'vss' )
        return


    def Netlist ( self ):
        print 'Cordic_DP.Netlist()'

        Generate( 'DpgenNmux2' ,  'nmux2_16b', param={'nbit':16,'behavioral':True
        Generate( 'DpgenMux2'  ,   'mux2_16b', param={'nbit':16,'behavioral':True

        self.instances = {}

        zero_16b = Signal( 'zero_16b', 16 )
        self.instances['zero_16b'] = Inst( 'zero_16b', 'zero_16b'
                                    , map = { 'q'   : zero_16b
                                            , 'vdd' : self.vdd
                                            , 'vss' : self.vss
                                            } )

        x = Signal( 'x', 16 )

        self.instances['x_sra_0_0'] = Inst( 'nmux2_16b', 'x_sra_0_0'
```

```
                                                    , map = { 'cmd' : self.i_p[0]
                                                            , 'i0'  : x
                                                            , 'i1'  : x_sra_1
                                                            , 'nq'  : x_sra_0_0
                                                            , 'vdd' : self.vdd
                                                            , 'vss' : self.vss
                                                            } )

        # Netlist a completer.
        # ...

         return


    def Layout ( self ):
         print 'Cordic_DP.Layout()'

        # X processing.
         Place     ( self.instances['zero_16b'        ], NOSYM, XY( 0, 0 ) )
         PlaceRight( self.instances['x_sra_0_0'       ], NOSYM )

        # Layout a completer.
        # ...

         return


def ScriptMain ( **kw ):
    if kw.has_key('editor') and kw['editor']: setEditor( kw['editor'] )

    cordic_dp = Cordic_DP( "cordic_dp" )

    cordic_dp.Interface()
    cordic_dp.Netlist  ()
    cordic_dp.Layout   ()
    cordic_dp.Save     (LOGICAL|PHYSICAL)
    return 1


if __name__ == "__main__" :
    kw      = {}
    success = ScriptMain( **kw )
    if not success: shellSuccess = 1

    sys.exit( shellSuccess )
```

Assemblage du contrôle et du chemin de données

Ce script effectue les différentes tâches suivantes:

1. Effacer les fichiers **vbe**, **vst** et **ap** issus d'une exécution précédente.
2. Charger le `.blif` de la partie contrôle et la convertir au format **vst** (c'est à dire ce que fait `blif2vst`).
3. Appeler le générateur de chemin de données écrit précedemment automatiquement.
4. Créer un *layout* de l'ensemble avec suffisament d'espace vide pour que le placer puisse y mettre les cellules de la partie contrôle.

Note
Ce fichier génére un équivalent exact à `cordic_net.vhd`, mais directement au format **vst**.

```
#!/usr/bin/env python

import sys
import symbolic.cmos
from   Hurricane import DbU
import CRL
import plugins.RSavePlugin
from   stratus   import *


class Cordic_Net ( Model ):

    def Interface ( self ):
        print 'Cordic_Net.Interface()'
        self.raz        = SignalIn ( 'raz'       , 1  )

        self.wr_axy_p   = SignalIn ( 'wr_axy_p'  , 1  )
        self.a_p        = SignalIn ( 'a_p'       , 10 )
        self.x_p        = SignalIn ( 'x_p'       , 8  )
        self.y_p        = SignalIn ( 'y_p'       , 8  )
        self.wok_axy_p  = SignalOut( 'wok_axy_p' , 1  )

        self.rd_nxy_p   = SignalIn ( 'rd_nxy_p'  , 1  )
        self.nx_p       = SignalOut( 'nx_p'      , 8  )
        self.ny_p       = SignalOut( 'ny_p'      , 8  )
        self.rok_nxy_p  = SignalOut( 'rok_nxy_p' , 1  )

        self.ck         = CkIn     ( 'ck'  )
        self.vdd        = VddIn    ( 'vdd' )
        self.vss        = VddIn    ( 'vss' )
        return


    def Netlist ( self ):
        print 'Cordic_Net.Netlist()'

        get      = Signal( 'get'     , 1 )
        calc     = Signal( 'calc'    , 1 )
```

```
        mkc      = Signal( 'mkc'     , 1 )
        place    = Signal( 'place'   , 1 )
        a_lt_0   = Signal( 'a_lt_0'  , 1 )
        quadrant = Signal( 'quadrant', 2 )
        i        = Signal( 'i'       , 3 )

        self.instances = {}

        self.instances['ctl'] = Inst( 'cordic_ctl', 'ctl'
                                    , map = { 'ck'         : self.ck
                                            , 'raz'        : self.raz
                                            , 'wr_axy_p'   : self.wr_axy_p
                                            , 'a_p'        : self.a_p
                                            , 'wok_axy_p'  : self.wok_axy_p
                                            , 'rd_nxy_p'   : self.rd_nxy_p
                                            , 'rok_nxy_p'  : self.rok_nxy_p
                                            , 'calc_p'     : calc
                                            , 'mkc_p'      : mkc
                                            , 'place_p'    : place
                                            , 'a_lt_0_p'   : a_lt_0
                                            , 'i_p'        : i
                                            , 'quadrant_p' : quadrant
                                            , 'vdd'        : self.vdd
                                            , 'vss'        : self.vss
                                            } )

        self.instances['dp'] = Inst( 'cordic_dp', 'dp'
                                   , map = { 'ck'         : self.ck
                                           , 'x_p'        : self.x_p
                                           , 'y_p'        : self.y_p
                                           , 'nx_p'       : self.nx_p
                                           , 'ny_p'       : self.ny_p
                                           , 'get_p'      : get
                                           , 'calc_p'     : calc
                                           , 'mkc_p'      : mkc
                                           , 'place_p'    : place
                                           , 'i_p'        : i
                                           , 'quadrant_p' : quadrant
                                           , 'a_lt_0_p'   : a_lt_0
                                           , 'n_xy_cmd'   : i          # THIS IS
                                           , 'vdd'        : self.vdd
                                           , 'vss'        : self.vss
                                           } )
        return


    def Layout ( self ):
        print 'Cordic_Net.Layout()'

        Place( self.instances['dp'], NOSYM, XY( 0, 0 ) )
        ResizeAb( 0, 0, 0, DbU.fromLambda( 100.0 ) )

        return
```

```
def ScriptMain ( **kw ):
    for entry in os.listdir('.'):
      if os.path.isfile(entry):
        if    entry.endswith('.vst') \
           or entry.endswith('.vbe') \
           or entry.endswith('.ap'):
          print 'Removing previous run file "%s".' % entry
          os.unlink( entry )

    views   = CRL.Catalog.State.Logical|CRL.Catalog.State.VstUseConcat
    ctlCell = CRL.Blif.load( 'cordic_ctl', CRL.Catalog.State.Logical|CRL.Catalog
    ctlKw   = {'cell':ctlCell, 'views':views}
    plugins.RSavePlugin.ScriptMain( **ctlKw )

    if kw.has_key('editor') and kw['editor']: setEditor( kw['editor'] )

    buildModel( 'cordic_dp', DoNetlist|DoLayout, className='Cordic_DP' )

    cordic_net = Cordic_Net( "cordic_net" )

    cordic_net.Interface()
    cordic_net.Netlist   ()
    cordic_net.Layout    ()
    cordic_net.Save      (LOGICAL|PHYSICAL)
    return 1


if __name__ == "__main__" :
    kw      = {}
    success = ScriptMain( **kw )

    shellSuccess = 0
    if not success: shellSuccess = 1

    sys.exit( shellSuccess )
```

Travail demandé

Effectuer les placements & routages par synthèse complète puis avec un chemin de données, comparer les résultats. Le travail à rendre est le fichier PYTHON de génération du chemin de données ainsi qu'une page expliquant vos conclusions.