Langage de netlist

python
stratus (netlist)

Université Pierre et Marie Curie
Master ACSI
Outils pour la Conception VLSI

Stratus

Vue d'ensemble

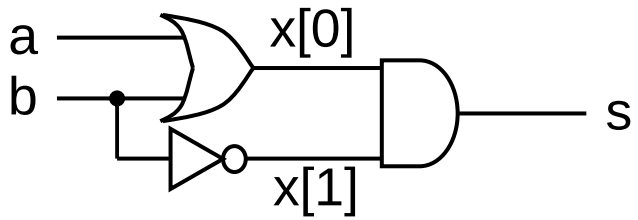
http://www-asim.lip6.fr/recherche/coriolis/doc/en/html/stratus

- Stratus est le langage de contrôle de la chaîne Coriolis
- Coriolis est une chaîne d'outils écrits en C++ permettant le placement et le routage d'un circuit VLSI.
- Le Cœur de Coriolis est la base de donnée Huricane
  - Huricane est une base de données écrite en C++ permettant de représenter et de manipuler de manière intégrée la vue logique, physique et temporelle d'un circuit VLSI.
- Stratus est un module en Python permettant
  - de décrire une netlist de cellules ou de blocs
  - de décrire un placement de cellules ou de blocs
  - de décrire et d'effectuer les étapes de construction automatique d'un circuits VLSI (de la netlist au layout)

Stratus

Description d'une netlist

```
#!/usr/bin/env python
from stratus import *
class Circuit(Model):
    def Interface(self):
        self.a   = SignalIn('a',1)
        self.b   = SignalIn('b',1)
        self.s   = SignalOut('s',1)
        self.vdd = VddIn('vdd')
        self.vss = VssIn('vss')
    def Netlist(self):
        self.x = Signal('x',2)
        Inst('o2_x1','u0',
             map={'i0':self.a,'i1':self.b,'q':self.x[0]
                  'vdd':self.vdd, 'vss':self.vss})
        Inst('inv_x1','u1',
             map={'i0':self.b,'nq':self.x[1],
                  'vdd':self.vdd, 'vss':self.vss})
        Inst('a2_x1','u2',
             map={'i0':self.x[0],'i1':self.x[2],'q':self.s,
                  'vdd':self.vdd, 'vss':self.vss})
c=Circuit('exemple')
c.Interface()
c.Netlist()
c.Save()
```



Stratus

Les signaux

Toutes ces méthodes ont jusqu'à 3 paramètres:

n : nom

a: le nombre de bit (mettre 1 pour un signal simple (1bit))

i: indice debut si !=0 (optionnel et seulement utile pour les vecteurs)

- SignalIn(n,a,i) : Création d'un port d'entrée
- SignalOut : Création d'un port de sortie
- SignalInOut : Création d'un port d'inout
- SignalUnknown : la Création d'un port de direction inconnue
- TriState : Création d'un port trois-état
- CkIn : Création d'un port d'horloge
- VddIn : Création d'alimentation vdd
- VssIn : Création d'alimentation vss
- Signal : Création d'un signal interne

Stratus

Les instances : exemple

instance simple :

```
Inst ( model , name , map = connectmap )
```

```
cin.Alias ( c_temp[0] )   # méthode de rénommage
cout.Alias ( c_temp[4] )
for i in range ( 4 ) :
    Inst ( "fulladder" ,
            map = {  'a' : a[i] ,
                     'b' : b[i] ,
                     'cin' : c_temp[i] ,
                     'sout' : sout[i] ,
                     'cout' : c_temp[i+1] ,
                     'vdd' : vdd , 'vss' : vss } )
```

Les modèles sont pris dans les bibliothèques de cellules précaractèrisées

Stratus

Les générateurs

générateurs

Generate (model, modelname, param = dict)

Prédéfinis

- Buffer
- Multiplexor
- Shifter
- Register
- Constants
- Boolean operations
- Arithmetical operations
- Comparison operations

ou Définis par l'utilisateur.

Python

Origines

- Créé par Guido Van Rossum en 1991
- Supporté par la PSF depuis 2001.
  (Python Software Fundation).
- Dérivé du langage ABC (dédié à l'apprentissage).
- Famille des langages de scripts :
  sh-71, awk-77, perl-87, tcl-88, lua-93, ruby-95, …
- Cathégorie "general purpose" multi-paradigme (objet, impératif)
- Usage :
  - Zope, Google, Youtube, BitTorrent, Trac, …
  - DSX (outils de codesign matériel/logiciel de SoCLib)
  - Langage d'extension d'applications (comme stratus)
  - Administration de système
  - un peu de tout …

Python
Documentations (extrait)

- http://www.python.org
  le site officiel
- http://www.afpy.org
  association française des utilisateurs de Python
- http://www.cifen.ulg.ac.be/inforef/swi/python.htm
  un livre disponible en pdf pour apprendre à programmer
- http://diveintopython.adrahon.org/
  un autre livre et plein de ressources
- http://www.iut-orsay.fr/dptmphy/Pedagogie/coursPython.pdf
  slides sur python (pdf et français)

Python
Caractérisques remarquables

Plus

- Langage interprété donc multiplateforme (cf java).
- Langage objet mais utilisable en impératif.
- Syntaxe réduite et intuitive.
- Code clair car compact et indenté.
- Types de base puissants (listes, dictionnaires, …).
- Nombreux modules (graphique, maths, parseurs, …).
- Encapsulation simple d'API compilés en C/C++.
- Gestion d'erreur intégrée et performante.
- Documentation intégrée aux programmes.
- Excellent premier langage de programmation.

Moins

- Plus lent que Java (mais psycho)
- L'absence de compilation retarde la découverte des erreurs (mais pylint)

Python
Philosophie

1. Beau est mieux que laid.
2. Explicite est mieux qu'implicite.
3. Simple est mieux que complexe.
4. Complexe est mieux que compliqué.
5. Plat est mieux qu'emboité.
6. Aéré est mieux que dense.
7. La lisibilité importe.
8. Les cas particuliers ne sont pas assez particuliers pour briser les règles…
9. … même si parfois, la facilité bat la pureté.
10. Les erreurs ne devraient jamais passer inaperçues…
11. … sauf si c'est voulu.
12. Face à une ambiguïté, refuser la tentation de deviner.
13. Il ne devrait exister qu'une et une seule manière évidente de faire quelque chose…
14. … même si cette manière peut ne pas être évidente au début (à moins d'être hollandais).
15. Maintenant est mieux que jamais…
16. … même si jamais est souvent mieux que maintenant tout de suite.
17. Si une implémentation est dure à expliquer, c'est que c'est une mauvaise idée.
18. Si une implémentation est facile à expliquer, c'est que c'est probablement une bonne idée.
19. L'idée du Namespace est une sacrée bonne idée, à exploiter au maximum.

Python

Hello World

Python est interprété et peut être utilisé de façon interactive

```
bop %-) python
Python 2.3.4 (#1, Oct  9 2006, 18:22:22)
[GCC 3.4.5 20051201 (Red Hat 3.4.5-2)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print "Hello World"
Hello World
```

On peut aussi écrire des fonctions et les invoquer

```
bop %-) python
Python 2.3.4 (#1, Oct  9 2006, 18:22:22)
[GCC 3.4.5 20051201 (Red Hat 3.4.5-2)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> def hello(s):
...     print "Hello",s
...
>>> hello("Franck")
Hello Franck
```

Python

script

Un programme Python (script) est (en général) mis dans un fichier pour être exécuté plus tard.

```
bop %-) cat hello.py
def hello(s):
    print "Hello",s

hello("Franck")
bop %-) python hello.py
Hello Franck
```

Une fonction célèbre : Fibonacci

```
bop %-) cat fibo.py
def fibo(n):
    a, b = 0, 1
    for i in range(n):
        a, b = b, a+b
    return a
for i in range(13):
    print "fibo(%d)=%d" % (i, fibo(i))
```

```
bop %-) python fibo.py
fibo(0)=0
fibo(1)=1
fibo(2)=1
fibo(3)=2
fibo(4)=3
fibo(5)=5
…
```

Python

Un script peut être rendu exécutable

```
bop %-) cat fact.py
#!/bin/env python
def facti(n):
    f=1
    while n>1:
      f, n = f*n, n-1
    return f

def factr(n):
    if n<1 :
        return 1
    else:
        return n*factr(n-1)

print "iteratif fact(%d)=%d" % (10,facti(10))
print "recursif fact(%d)=%d" % (10,factr(10))
```

```
bop %-) chmod u+x fact.py
bop %-) fact.py
iteratif fact(10)=3628800
recursif fact(10)=3628800
```

Python

Récupération des arguments

```
bop %-) cat fact.py
#!/bin/env python
import sys
def facti(n):
    f=1
    while n>1:
      f, n = f*n, n-1
    return f

def factr(n):
    if n<1 :
        return 1
    else:
        return n*factr(n-1)

n=int(sys.argv[1])
print "iteratif fact(%d)=%d" % (n,facti(n))
print "recursif fact(%d)=%d" % (n,factr(n))
```

```
bop %-) chmod u+x fact.py
bop %-) fact.py 10
iteratif fact(10)=3628800
recursif fact(10)=3628800
```

```
bop %-) fact.py 100
iteratif fact(100)=933262154439441
526816992388562667004907159682
643816214685929638952175999932
299156089414639761565182862536
979208272237582511852109168640
0000000000000000000000
recursif fact(100)=933262154439441
526816992388562667004907159682
643816214685929638952175999932
299156089414639761565182862536
979208272237582511852109168640
0000000000000000000000
```

Python

Types de base

- **int** entier
- **long** entier de précision infini
- **bool** True / False
- **float** nombre réel double précision
- **complex** nombre complexe
- **str** chaîne de caractères ascii
- **unicode** chaîne de caractère unicode (2octets)
- **tuple** séquence invariable d'éléments hétérogènes
- **list** séquence variable d'éléments hétérogènes
- **dict** dictionnaire d'éléments hétérogènes {clé:valeur}

Python
Nombres

Exemples

int 1 2 -4 0x12 0765
en cas de dépassement de capacité,
le type int est changé en type long

long 14L 716726671626L -9919199929919L
la précision est infinie (e.g. essayez fact(998) !)

float 1. 67e-2 1.79769313486231570e308
double précision

complex 2 + 3j
deux flottants double précision

Opérateurs proches du C

(1+2)*3/2, 5**100, 0x12<<2, ~a, a^b, x|1 …

Python
Chaînes

str	"bonjour"	'bonjour'	"l'autre"
	"""sur plusieurs lignes"""		
unicode	u"bonjour"	u'bonjour'	

On peut utiliser les caractères d'échappement du C : \t, \n, …

On peut formater des chaînes :

```
a = 2
"ceci %s une chaîne  avec %d\n" % ('est', a)
```

représente la chaîne :

```
"ceci est une chaîne avec 2\n"
```

Les codes de formatage sont les mêmes qu'en C.

Python
Chaînes

a = "bonjour"

a[0] est 'b'

a[-1] est 'r'

a[i:] est la chaine depuis i jusqu'à la fin

a[i:j] est la chaine de i à j

a[i:j:k] est la chaine de i à j par pas de k

Python
tuples

Séquence invariable hétérogène

(élément, …)

T1 = ('a', 1, 3)

T2 = (b,)

T1[0] est 'a'

- Les tuples sont représentés par des tableaux en mémoire.

Python
Liste

Séquences hétérogènes variable

L1 = [1, 'bonjour', 3.14]

L2 = [1, (1,2), [1,2]]

L1[0] est 1

L1[:1] est [1,'bonjour']

- Les listes sont aussi représentées par des tableaux en mémoire.

Python
Dictionnaires

Dictionnaire

{cle1:val1, cle2:val2, …}

d = {1:12, 'bonjour':34, 3.14:'salut', (1,2):'fin' }

- Les clés des dictionnaires sont *invariables*
- Les listes et les tuples sont ordonnées, pas les dictionnaires.
- D['bonjour'] est 34

Python
Booléens

True vaut 1

False vaut 0

bool(quelque chose) indique si le quelque chose est vrai.

bool (n'importe quoi) est True

sauf 0, 0L, 0.0, ( ), [ ], { }

Python
Variables

Une variable est une référence sur une valeur

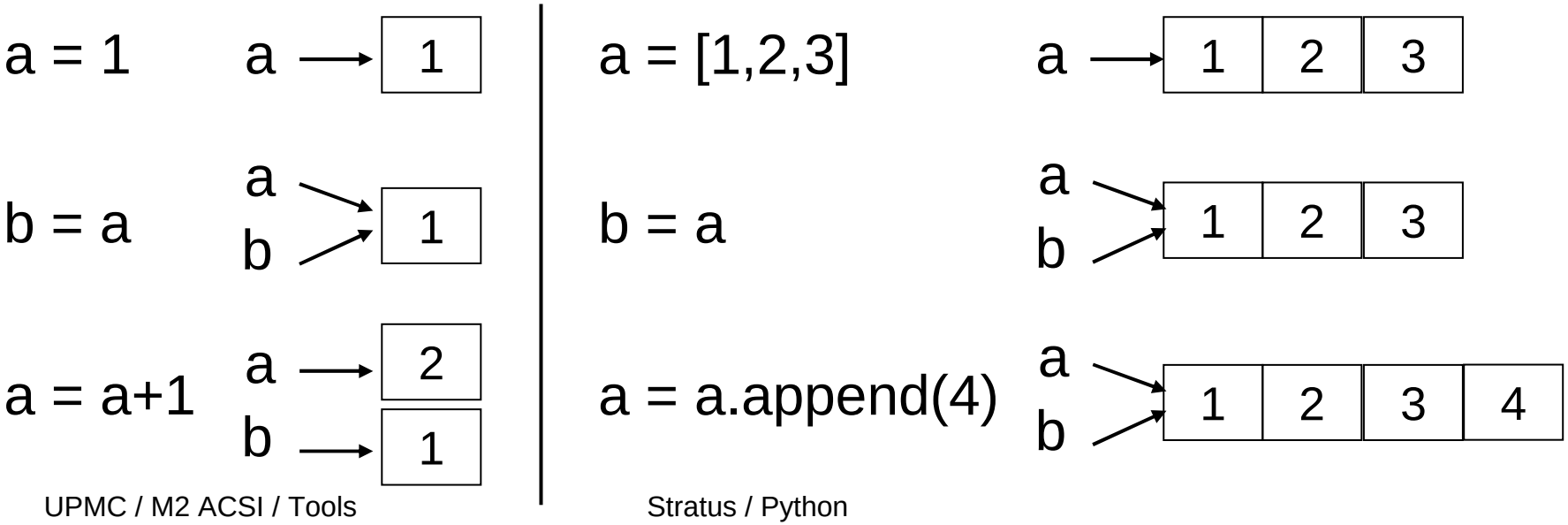
- Les variables n'ont pas besoin d'être déclarées.
- Les variables prennent automatiquement le type de la valeur attribuée.

```
a = 1           # a est un int
b = 2           # b est un int
c = a + b       # c est un int
a = 'bonjour'   # a est une chaîne
b = a + 3       # ERREUR
b = a * 3       # là c'est bon !
```

Python

Valeurs invariables (inmutables)

- les nombres (int, long, float, complex), les chaines de caractères et les tuples sont invariables
  - ➔ une fois créés en mémoire, ils ne sont jamais modifiés.
- Les listes et les dictionnaires sont variables (mutables)
  - ➔ une fois créés, ils vont pouvoir évoluer.



Python

Structures de contrôle

while condition**:**
    instructions
[**else:**
    instructions]

for element **in** sequence**:**
    instructions
[**else**:
    instructions]

break sort de la boucle sans executer **else:**

continue passe au nouvel élément de la séquence

pass représente une séquence d'instructions nulles

if condition1**:**
    instructions
[**elif** condition2**:**
    instructions]
[**else:**
    instructions]

Exemples:

```
for i in range(10) :
    print "i=",i

for (x,y) in ((3,5),(a,b)) :
    print x*y
```

Python
Opérateurs

affectation

```
a = 1
a,b = 1, 2      a prend 1, b prend 2
t = (a,b)       () pas obligatoire
d,e = t         d prend 1, e prend 2
```

Opérateurs

```
+ - * / % ** >> << | & ~ ^
and  or  not
==  !=  <  >  <=  >=
in   not in
is   is not
```

exemples

```
if a < b < c:
        instructions
```

"a" * 3 donne "aaa"

['a'] * 3 donne ['a', 'a', 'a']

['a'] + ['b'] donne ['a', 'b']

Python
Organisation du code

Le code de Python est structuré en

fonctions
séquence d'instructions conditionnelles paramétrables

classes
ensembles de fonctions (méthodes) et de variables (attributs)

modules
fichiers regroupant variables, fonctions et classes

paquets
répertoires regroupant des modules

Python
Définition de fonctions

def nom (param):
    instructions

return rend une valeur qui peut être quelquonque

param :
    (a1, a2, a3=def_val, *args, *kw)
- a1, a2, a3 sont trois paramètres standards
- a3 a une valeur par défaut et peut ne pas être passé en paramètre
- *args est un tuple qui va contenir tous les paramètres non nommés
- *kw est un dictionnaire qui va contenir tous les paramètres nommés non standard

Python
Exemple de fonctions

Une fonction est définie par une liste de paramètres et une séquence d'instructions

- Les fonctions peuvent avoir un nombre variable de paramètres.
- L'association paramètre-valeur peut être explicite.
- Le passage des paramètres se fait par référence.

```
def test (a1, a2, a3=3, *args, *kw):
    print "a1=%d a2=%d a3=%d" % (a1, a2, a3)
    print "args=%s\n kw=%s" % (str(args), str(kw))
```

test(10,20)
a1=10 a2=20 a3=3
args=()
kw={}
test(10,20,30)
a1=10 a2=20 a3=30
args=()
kw={}
test(10,20,30,40,50)
a1=10 a2=20 a3=30
args=(40, 50)
kw={}
test(10,20,30,40,50,fin=70)
a1=10 a2=20 a3=30
args=(40, 50)
kw={'fin': 70}

Python
Définition de classes

Une classe est un ensemble de variables (attributs) et de fonctions (méthodes)

- Une classe définit un espace de nom qui structure le code.
- *__method__* représente une méthode dont l'usage est prédéfini, elle permet en particulier la surcharge des opérateurs.
  - __init__ est invoquée à la création de l'objet
  - __str__ est invoquée par print
  - __add__ permet de définir l'addition de deux objets…

```
class NomClass:
    attrclass = Vdefaut
    def __init__(self, p1, p2):
        self.attrobjet = p1
        self.__attrprive = p2
    def __str__(self):
        return "ac=%s ao=%s ap=%s" % \
                (attrclass, self.attrobjet,
                 self.attrprive)
    def m1(self, a1):
        self.attrobjet += a1

i = NomClass(34, 67)
i.attrobjet = 23
NomClass.attrclass =1
print i
```

Python
Méthodes

En Python tout est objet et on peut appliquer des méthodes à chaque objet
Il y en a des dizaines !

Quelques méthodes de chaines

s.capitalize()
met en majuscules

s.join(sequence)
fabrique une chaine en concaténant les éléments de sequence entre lesquels on met S.

s.strip()
élimine les espaces de début et fin

s.find(sub, start=0,end=sys.maxint)
rend le plus petit index dans s où se trouve sub

Quelques méthodes de list

l.append(x)
ajoute x en fin de liste

l.reverse()
renverse une liste

l.remove(x)
efface la première occurrence de x

Quelques méthodes de dictionnaires

d.has_key(k)
rend True si d contient la clé k

d.values()
rend une liste avec toutes les valeurs de d

d.pop(k[,x])
retire la clé k de d si elle existe sinon retourne x

d.copy()
rend une copie de d

Python
accès aux entrées/sorties

Terminal

- entrée d'une donnée depuis le clavier
  a = input ("message : ") *typage automatique*
  a = raw_input(message : ") force la saisie texte
- sortie sur l'écran
  print "mess1" mess1 + CR
  print "mess2", mess1 sans CR
  print "format" % (valeurs) mess formaté

Fichiers

- f1 = open('filename', 'r') de même 'w', 'a'
- f1.close() écrit sur le disque et ferme le fichier
- f1.write(s) / print s >> f1 écrit la chaîne s dans f1
- f1.read() / f1.read(n) lit tout le fichier ou n octets
- f1.readline() / f1.readlines() lit une ligne ou toutes les lignes%%%

Conclusion

- Vous n'êtes pas oblige de connaître Python, même si c'est préférable parce c'est une beau langage.
- Vous allez peut-être devoir lire la doc de stratus (coté netlist pour le moment)

Que du bonheur !