

VLSI -2

Placement & Routage
Langage Stratus (Python)

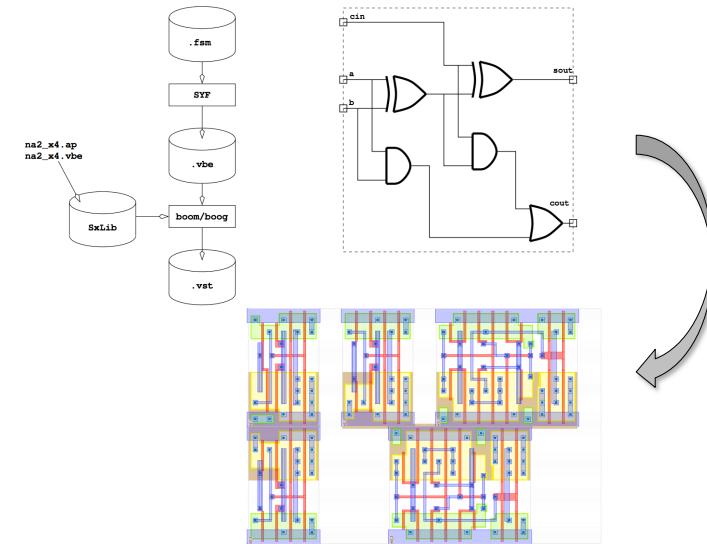
UPMC / M2 SESI / VLSI-2

Plan

- **P & R principles**
- P & R alimentation
- P & R horloge
- Stratus netlist
- Stratus P & R
- Python

UPMC / M2 SESI / VLSI-2

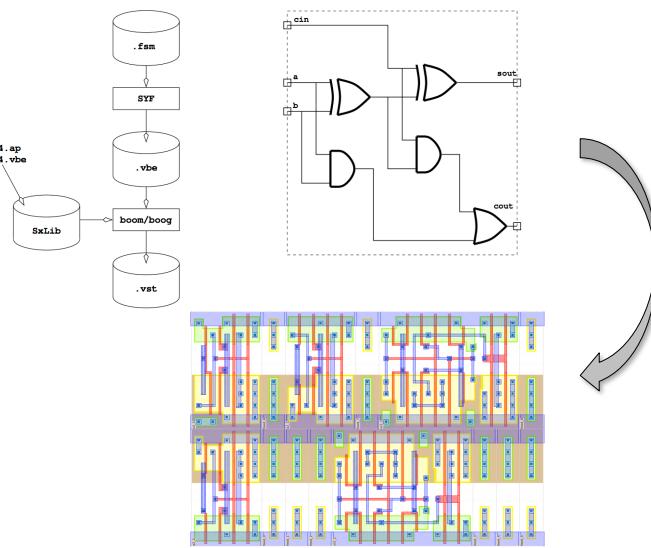
Principe



UPMC / M2 SESI / VLSI-2

Src: Jean-Paul Chaput

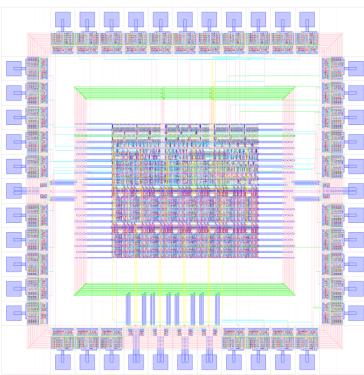
Principe



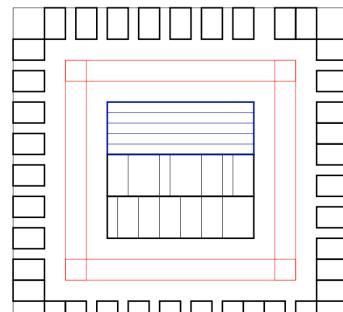
UPMC / M2 SESI / VLSI-2

Src: Jean-Paul Chaput

Objectif



UPMC / M2 SESI / VLSI-2



Src: Jean-Paul Chaput

Méthode générale de P & R

La réalisation suit une approche descendante, comportant 7 étapes :

1. Définition précoce d'un plan de masse`
2. Placement explicite des plots d'entrée-sortie sur les 4 faces.
3. Placement explicite des blocs « durs » (mémoires, chemins de données optimisés) dans la zone réservée au cœur.
4. Définition explicite des réseaux de distribution de alimentations (maillage VDD et VSS)
5. Placement automatique de la logique irrégulière dans les zones inoccupées du cœur.
6. Routage automatique des signaux particuliers (signaux d'horloge ou autres signaux critiques)
7. Routage automatique des signaux logiques autres que les alimentations ou les horloges.

P&R

Automatisation du placement / routage

Lorsque le schéma est défini, la génération du dessin des masques peut être réalisée par des outils automatiques :

- Le **placement** consiste à définir la position de chaque cellule, avec les contraintes suivantes :
 - deux cellules ne peuvent pas se recouvrir (la surface totale est supérieure ou égale à la somme des surfaces des cellules instanciées).
 - un « bon » placement doit viser à minimiser la longueur des fils d'interconnexion de façon à minimiser à la fois les temps de propagation et la consommation.
- Le **routage** consiste à dessiner les fils d'interconnexion entre cellules, avec les contraintes suivantes :
 - Le dessin des fils doit respecter des règles de distance minimale entre fils, pour éviter les court-circuits.
 - Il faut minimiser chaque fois que possible les capacités de couplage entre fils voisins sur une même couche.

UPMC / M2 SESI / VLSI-2

Stratus

Plan

- P & R principes
- **P & R alimentation**
- P & R horloge
- Stratus netlist
- Stratus P & R
- Python

UPMC / M2 SESI / VLSI-2

Le rôle des alimentations

Toute porte logique CMOS duale doit être connectée aux deux alimentations VDD et VSS.

Ces signaux représentent les valeurs logiques 0 et 1, mais ils ont principalement pour rôle d'apporter l'énergie nécessaire au fonctionnement du circuit.

La puissance dissipée instantanée dépend du courant $i(t)$ qui circule entre les bornes VDD et VSS du circuit :

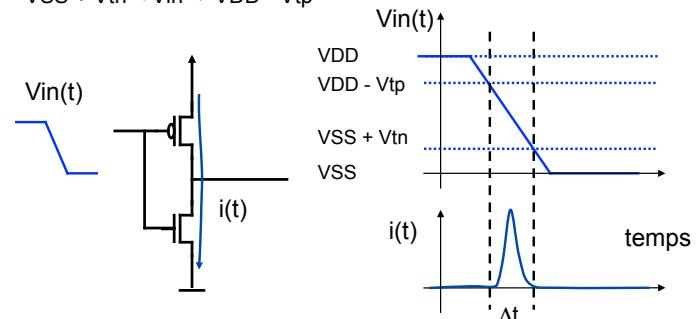
$$P(t) = (VDD - VSS) * i(t)$$

UPMC / M2 SESI / VLSI-2

P&R alimentation

Le courant de court-circuit

A chaque transition du signal d'entrée, les deux transistors de l'inverseur sont simultanément passants durant le temps Δt , lorsque $VSS + Vtn < Vin < VDD - Vtp$



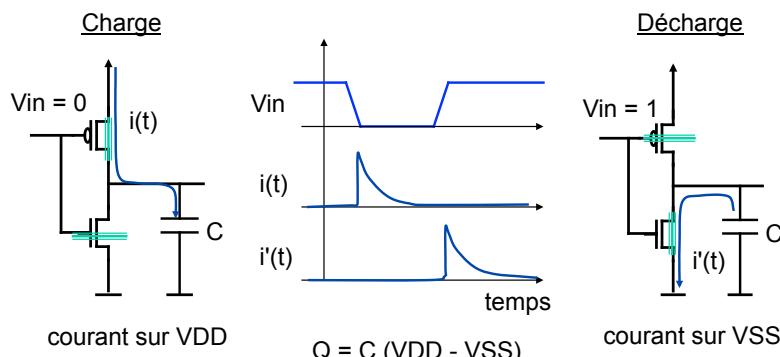
Le courant de court-circuit est purement « parasite » : il ne participe pas à la transmission de l'information. On cherche donc à éviter les fronts « mous », de façon à réduire la durée Δt du court circuit.

UPMC / M2 SESI / VLSI-2

P&R alimentation

Les courants de commutation

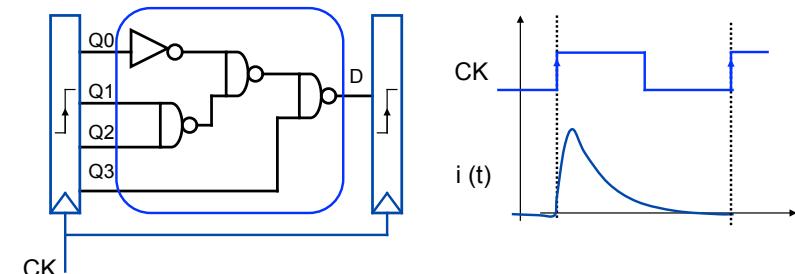
La première cause de consommation d'énergie dans les circuits numériques CMOS est liée à la charge et à la décharge des capacités.



UPMC / M2 SESI / VLSI-2

Dépendance temporelle

Dans les circuits synchrones, le courant dans les réseaux d'alimentation VDD et VSS n'est pas constant, et le bruit d'alimentation est fortement corrélé au signal CK :



Ceci pousse au développement de circuits possédant plusieurs domaines synchrones, pour répartir les appels de courant dans le temps.

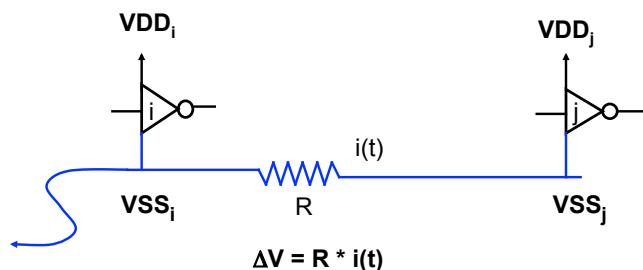
UPMC / M2 SESI / VLSI-2

P&R alimentation

Les chutes de tension résistives ...

La tension constante des signaux VDD et VSS est une abstraction :

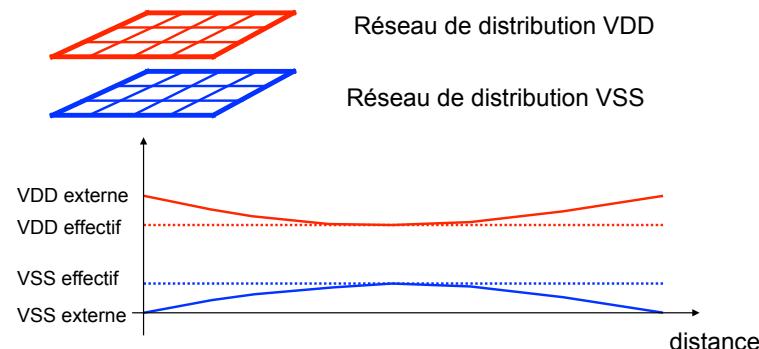
La résistance intrinsèque des fils métalliques réalisant le réseau de distribution VSS entraîne des chutes de tension entre différents points du réseau VSS.



UPMC / M2 SESI / VLSI-2

P&R alimentation

Dépendance topographique



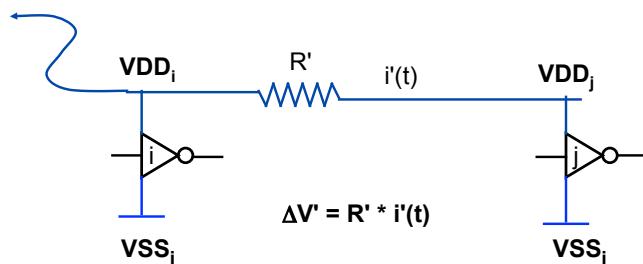
Les plots d'alimentation étant généralement situés à la périphérie, les portes situées au centre du circuit sont les plus mal alimentées.

UPMC / M2 SESI / VLSI-2

P&R alimentation

... sur VSS comme sur VDD

Il en va de même pour le réseau de distribution VDD :



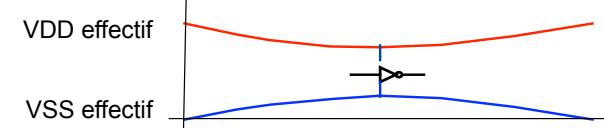
La résistance intrinsèque des fils métalliques a pour effet de baisser la tension VDD et d'augmenter la tension VSS « vues » par une porte interne (par rapport aux valeurs nominales).

UPMC / M2 SESI / VLSI-2

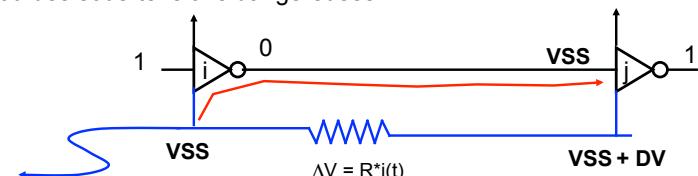
P&R alimentation

Conséquences

- La réduction de la tension d'alimentation effective VDD eff - VSS eff entraîne un **augmentation sensible des temps de propagation**



- Les variations des tensions d'alimentation sont des **causes de bruit** sur les signaux logiques, qui peuvent entraîner des surtensions ou des sous-tensions dangereuses.

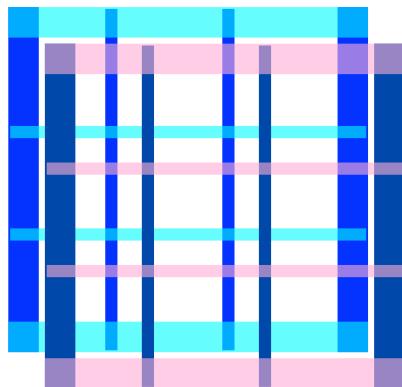


UPMC / M2 SESI / VLSI-2

P&R alimentation

Réseaux maillés

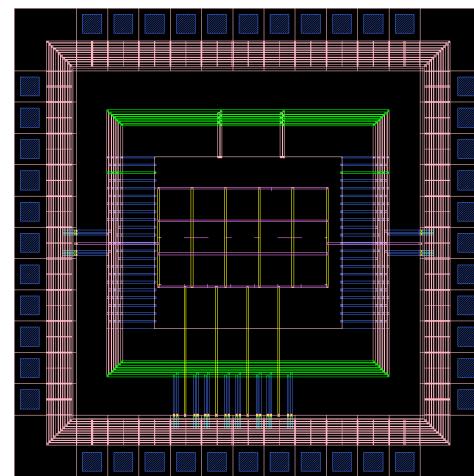
Pour minimiser la résistance entre le centre et la périphérie, on utilise pour des réseaux maillés, pour VDD comme pour VSS :



UPMC / M2 SESI / VLSI-2

P&R alimentation

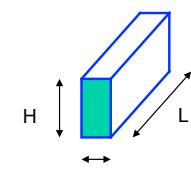
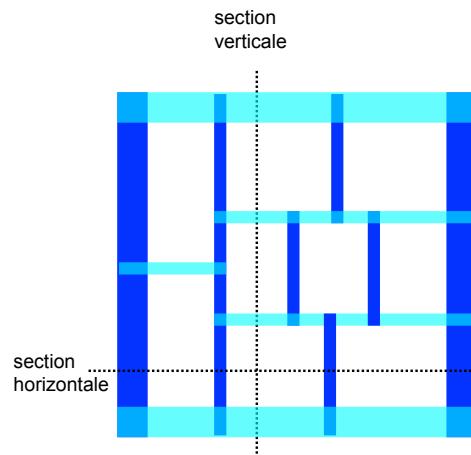
P&R am2901



UPMC / M2 SESI / VLSI-2

P&R alimentation

Dimensionnement



$$R = \rho * L / W * H$$

Règle : la largeur cumulée des fils suivant n'importe quelle section (horizontale ou verticale) doit être supérieure à une valeur minimale prédéfinie.

UPMC / M2 SESI / VLSI-2

P&R alimentation

Plan

- P & R principes
- P & R alimentation
- **P & R horloge**
- Stratus netlist
- Stratus P & R
- Python

UPMC / M2 SESI / VLSI-2

La difficulté

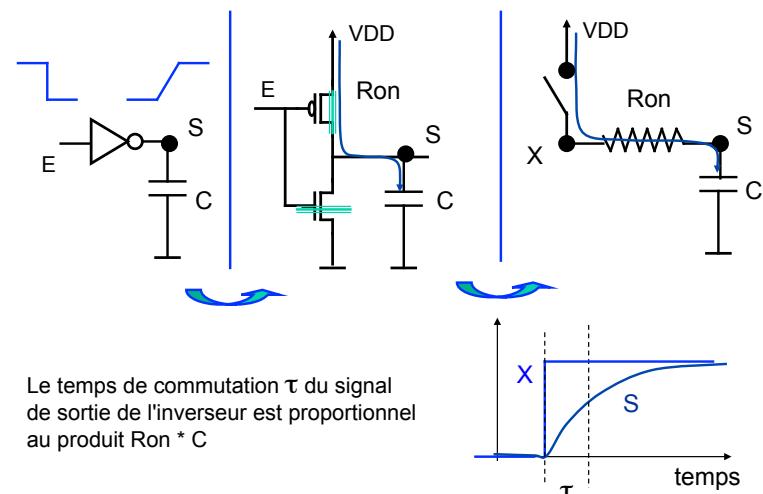
- Un circuit synchrone peut contenir des milliers de bascules, et chaque bascule possède une capacité d'entrée.
- Les bascules sont généralement réparties sur toute la surface de la puce, et les fils de distribution du signal d'horloge sont nécessairement très longs, ce qui représente une grosse capacité.
- On souhaite que le front du signal d'horloge soit le plus raide possible, pour minimiser l'imprécision due aux dispersions des seuils des transistors.

Il faut donc **amplifier** le signal d'horloge externe pour générer les signaux d'horloge internes.

UPMC / M2 SESI / VLSI-2

P&R horloge

Temps de commutation

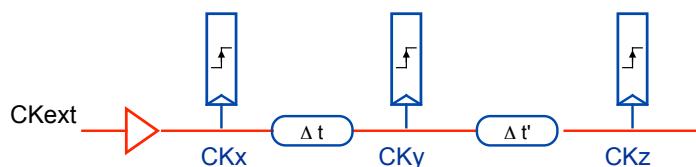


Le temps de commutation τ du signal de sortie de l'inverseur est proportionnel au produit $R_{on} * C$

UPMC / M2 SESI / VLSI-2

P&R horloge

Le skew



Le signal de synchronisation CK est une abstraction, car le réseau de distribution du signal CK n'est pas parfait :

- Les deux principales causes du skew sont :
- les résistances intrinsèques des fils,
 - les amplificateurs intermédiaires

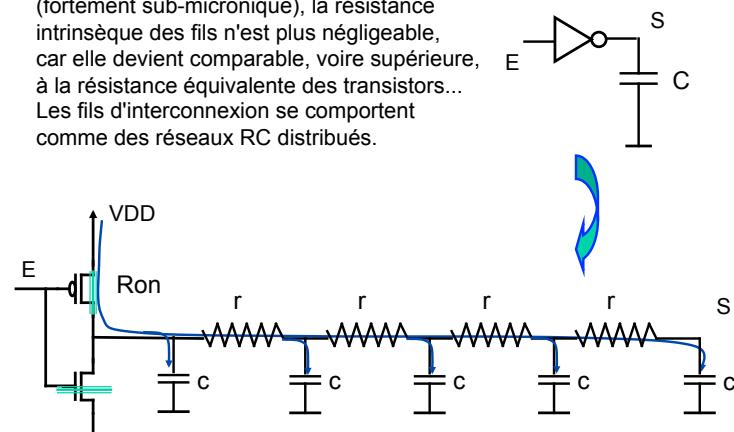
Le principal objectif du réseau de distribution d'horloge est donc de minimiser la valeur du skew.

UPMC / M2 SESI / VLSI-2

P&R horloge

La résistance des longs fils

Dans les procédés de fabrication DSM (fortement sub-micronique), la résistance intrinsèque des fils n'est plus négligeable, car elle devient comparable, voire supérieure, à la résistance équivalente des transistors... Les fils d'interconnexion se comportent comme des réseaux RC distribués.



UPMC / M2 SESI / VLSI-2

P&R horloge

Les techniques de distribution d'horloge

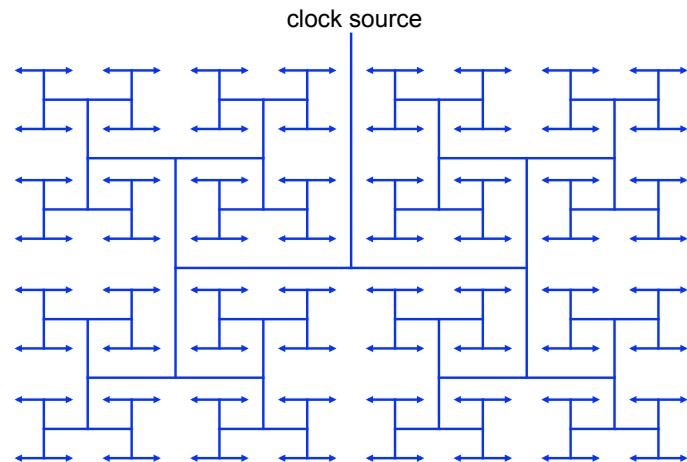
Il existe plusieurs techniques pour minimiser le skew :

- arbre d'amplificateurs équilibrés
- réseau maillé faiblement résistif
- approche GALS (Globalement Asynchrone Localement Synchrone)

UPMC / M2 SESI / VLSI-2

P&R horloge

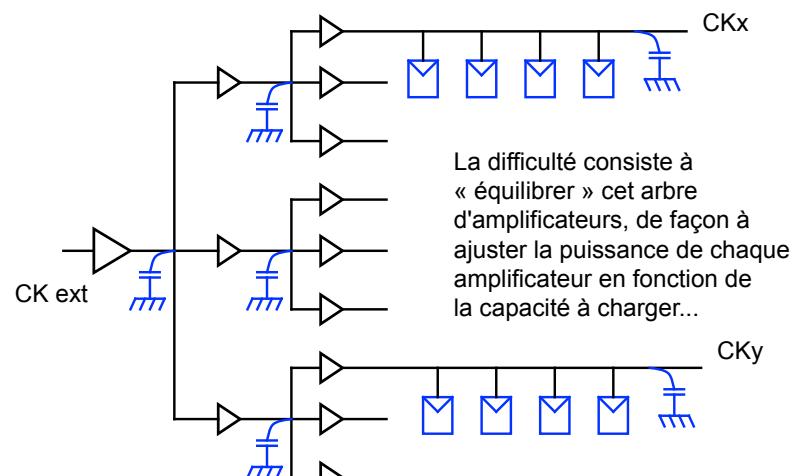
Topologie en H



UPMC / M2 SESI / VLSI-2

P&R horloge

Arbres d'amplificateurs

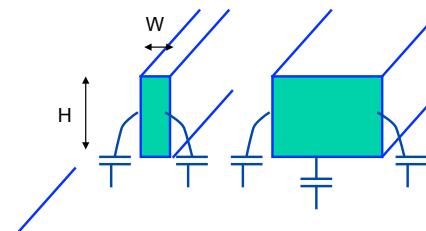


UPMC / M2 SESI / VLSI-2

P&R horloge

Réseaux maillés / a

Quand la surface du domaine synchrone n'est pas trop importante, on peut utiliser un réseau maillé, en utilisant des fils de largeur **non** minimale (on minimise la résistance des fils en acceptant d'augmenter un peu la capacité).



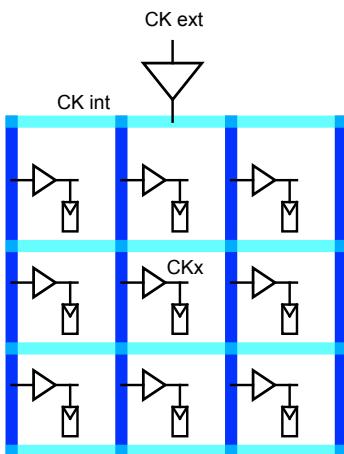
Les fils ayant une hauteur H plus grande que la largeur minimale W, la capacité du fil n'est pas proportionnelle à la surface...

Attention : si la résistance diminue, la capacité augmente, le skew ne change pas, mais la consommation augmente.

UPMC / M2 SESI / VLSI-2

P&R horloge

Réseaux maillés / b



- On essaie de rendre le réseau maillé correspondant au signal CKint aussi équipotentiel que possible, sur toute la surface du domaine synchrone.
- On introduit un étage d'amplificateurs locaux, entre le signal synchrone CKint et les signaux CKx connectés aux bascules.
- La puissance de chaque amplificateur local doit être ajustée, en fonction du nombre de bascules connectées au signal CKx.

UPMC / M2 SESI / VLSI-2

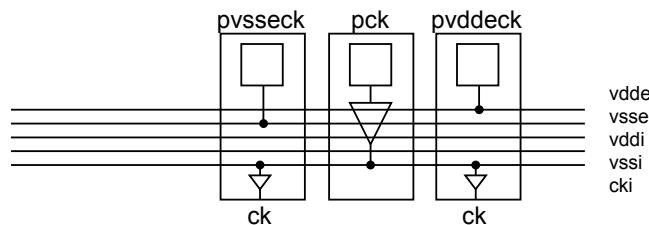
P&R horloge

Plan

- P & R principes
- P & R alimentation
- P & R horloge
- **Stratus netlist**
- Stratus P & R
- Python

UPMC / M2 SESI / VLSI-2

Plots horloge Alliance



UPMC / M2 SESI / VLSI-2

P&R horloge

Vue d'ensemble

<http://www-soc.lip6.fr/recherche/cian/coriolis/documentation/>

- **Stratus est le langage de contrôle de la chaîne Coriolis**
- Coriolis est une chaîne d'outils écrits en C++ permettant la description de la netlist, le placement et le routage d'un circuit VLSI.
- **Le Cœur de Coriolis est la base de donnée Hurricane**
 - Hurricane 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 :**
 1. décrire une netlist de cellules ou de blocs
 2. décrire des patterns (comme genpat) et d'invoquer Asimut
 3. décrire un placement de cellules ou de blocs
 4. décrire et d'effectuer les étapes de construction automatique d'un circuits VLSI (de la netlist au layout)

UPMC / M2 SESI / VLSI-2

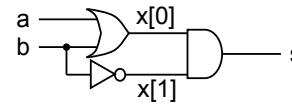
Stratus

Description d'une netlist

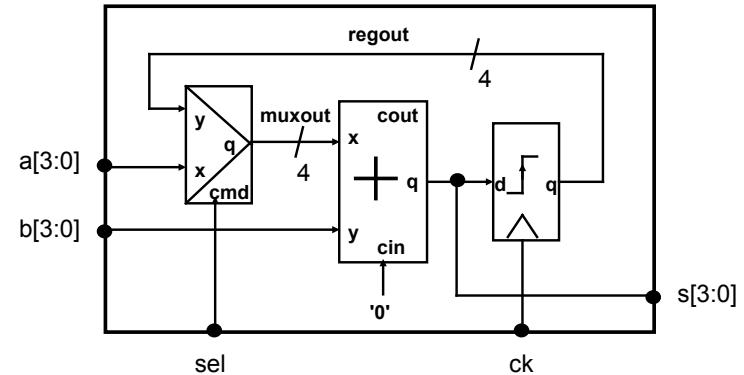
```
#!/usr/bin/env python
from stratus import *
class Circuit(Model):
    def Interface(self):
        self.a = SignalIn('a',1)
        self.b = SignalIn('b',1)
        self.s = SignalOut('s',1)
        self.vdd = VddIn('vdd')
        self.vss = VssIn('vss')
    def Netlist(self):
        self.x = Signal('x',2)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

UPMC / M2 SESI / VLSI-2

Stratus



addaccu



UPMC / M2 SESI / VLSI-2

Stratus

syntaxe de base

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 : 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

instance simple :

- Inst (model , name , map = connectmap)

UPMC / M2 SESI / VLSI-2

Stratus

Les générateurs

générateurs

Generate (model, modelname, param = dict)

Prédefinis

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

ou Définis par l'utilisateur.

UPMC / M2 SESI / VLSI-2

Stratus

Création d'un générateur

On souhaite décrire un générateur d'addaccu n bits, on décrit trois générateurs : mux, add, accu.

ici le mux (dans mux.py), on peut faire de même pour add et accu

```
#!/usr/bin/env python
from stratus import *
# instantiation des cellules
def Netlist (self):
    for i in xrange(self.n):
        Inst ( "mx2_x2"
              , map = { 'i0' : self.i0[i]
                        , 'il' : self.il[i]
                        , 'cmd': self.cmd
                        , 'q' : self.s[i]
                        , 'vdd': self.vdd
                        , 'vss': self.vss })
```

```
# definition du bloc mux
class mux (Model):
    # declaration des connecteurs
    def Interface (self):
        # recuperation du parametre
        self.n = self._param('nbit');
        # connecteurs
        self.i0 = SignalIn ( "i0" , self.n )
        self.il = SignalIn ( "il" , self.n )
        self.cmd = SignalIn ( "cmd" , 1 )
        self.s = SignalOut( "s" , self.n )
        self.vdd = VddIn ( "vdd" )
        self.vss = VssIn ( "vss" )
```

UPMC / M2 SESI / VLSI-2

Stratus

création de addaccu

Dans un fichier testaddaccu.py, on crée le modèle addaccu

```
#!/usr/bin/env python
from stratus import *
from addaccu import addaccu
iaddaccu=addaccu('iaddaccu', {'nbit': Param('n')})
iaddaccu.Interface()
iaddaccu.Netlist()
iaddaccu.Save()
```

UPMC / M2 SESI / VLSI-2

Stratus

addaccu

```
# instantiation des cellules
def Netlist (self):
    generate('mux.mux','mux%s'%self.n,
             param={'nbit':self.n})
    generate('add.add','add%s'%self.n,
             param={'nbit':self.n})
    generate('accu.accu','accu%s'%self.n,
             param={'nbit':self.n})
```

```
# definition du bloc addaccu
class addaccu (Model):
    # declaration des connecteurs
    def Interface (self):
        # recuperation du parametre
        self.n = self._param('nbit');
        # connecteurs
        self.a = SignalIn ( "a" , self.n )
        self.b = SignalIn ( "b" , self.n )
        self.sel = SignalIn ( "sel" , 1 )
        self.ck = SignalIn ( "ck" , 1 )
        self.s = SignalOut( "s" , self.n )
        self.vdd = VddIn ( "vdd" )
        self.vss = VssIn ( "vss" )
        idem pour add et pour accu
```

UPMC / M2 SESI / VLSI-2

Stratus

Plan

- P & R principes
- P & R alimentation
- P & R horloge
- Stratus netlist
- **Stratus P & R**
- Python

UPMC / M2 SESI / VLSI-2

API Stratus

<u>Description of a layout</u>	<u>Place and Route</u>	<u>Virtual library</u>
<u>Instanciation facilities</u>		
- Place	- PlaceSegment	- Buffer
- PlaceTop	- PlaceContact	- Multiplexor
- PlaceBottom	- PlaceRef	- Shifter
- PlaceRight	- GetRefXY	- Register
- PlaceLeft	- PlaceCentric	- Constants
- SetRefIns	- PlaceGlu(e)	- Boolean operations
- DefAb	- FillCell	- Arithmetical operations
- ResizeAb	- PadNorth,...	- Comparison operations
	- AlimVerticalRail,
	- AlimConnectors	
	- PowerRing	
	- RouteCk	

UPMC / M2 SESI / VLSI-2

Placement Explicite - 2

```
def Layout (self):

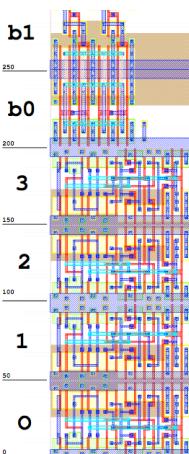
    for i in range ( 8 ) :
        if i == 0 : Place ( self.ram_reg[i], NOSYM, XY ( 0, 0 ) )
        else      : PlaceRight ( self.ram_reg[i], NOSYM )
    PlaceRight ( self.ram_ntsa[1], NOSYM )
    PlaceRight ( self.ram_ntsb[1], NOSYM )

    for i in range ( 8, 16 ) :
        if i == 8 : Place ( self.ram_reg[i], NOSYM, XY ( 0, 300 ) )
        else      : PlaceRight ( self.ram_reg[i], NOSYM )
    PlaceRight ( self.ram_ntsa[1], NOSYM )
    PlaceRight ( self.ram_ntsb[1], NOSYM )

    SetRefIns ( self.ram_reg[8] )
    PlaceTop ( self.inv_ra,           NOSYM )
    PlaceRight ( self.inv_rb,         NOSYM )
    PlaceRight ( self.mx2_ops0,       NOSYM )
```

UPMC / M2 SESI / VLSI-2

Placement Explicite - 1



```
def Netlist ( self ) :
    # Create both the "dff_4bits"
    # netlist (.vst) and placement (.ap).
    Generate ( "DpgenSff"
               , "dff_4bits"
               , param = { 'nbit' : 4
                          , 'physical': True} )

    # [...]

    for i in range ( 16 ) :
        # Instanciate Register part.
        self.ram_reg[i] = Inst ( "dff_4bits", "ram_reg%ld" % i
                               , map   = { 'wen': self.b_w[i]
                                         , 'ck' : self.ram_ck[i]
                                         , 'io' : ram_d
                                         , 'q' : ram_q[i]
                                         , 'vdd': self.vdd
                                         , 'vss': self.vss
                                         } )
        # [...]
```

UPMC / M2 SESI / VLSI-2

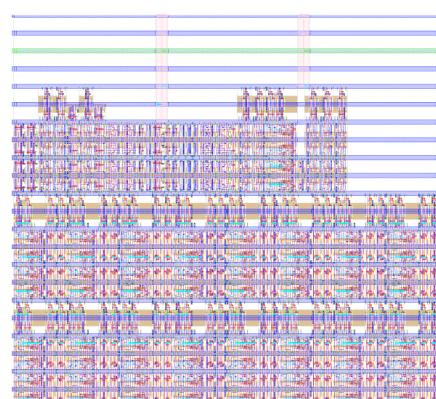
Assemblage du cœur

```
def Layout ( self ) :
    Place( self.Dpt,NOSYM, XY(0,0) )

    ResizeAb( 0, 0, 0, 200 )

    AlimVerticalRail ( 80 )
    AlimVerticalRail ( 160 )
    AlimHorizontalRail( 20 )

    AlimConnectors()
    return
```



UPMC / M2 SESI / VLSI-2

Assemblage du circuit

```
def Layout ( self ) :
    DefAb( XY(0, 0), XY(3000, 3000) )
    self.View()

    PlaceCentric(self.Core)
    self.View()

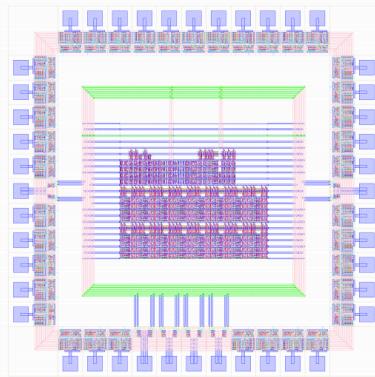
    PadNorth( self.p_i[3], self.p_i[4], self.p_i[5], self.p_b[0], [...] )
    PadWest ( self.p_ck, self.p_d[0], self.p_d[1], self.p_d[2], [...] )
    PadSouth( self.p_cin, self.p_np, self.p_ng, self.p_vssick0, [...] )
    PadEast ( self.p_y[3], self.p_signe, self.p_noe, self.p_a[0], [...] )

    PowerRing( 3 )
    self.View()

    PlaceGlue( self.Core )
    self.View()

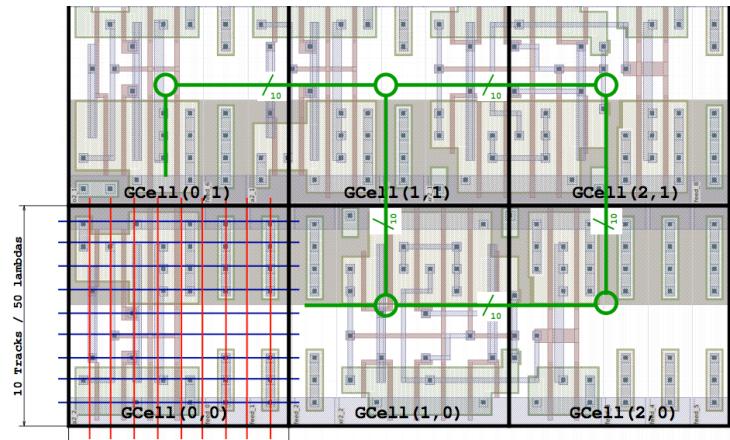
    RouteCk( self.ckc )
    self.View()

    return
```



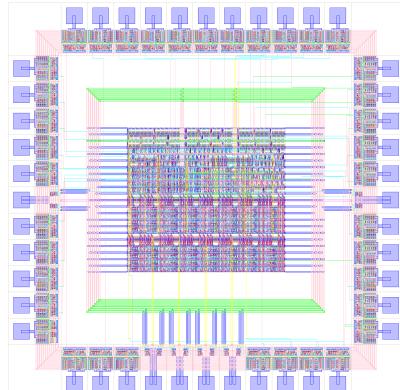
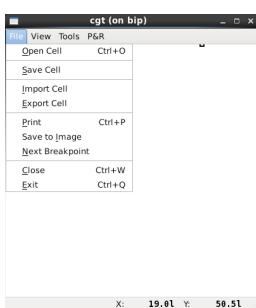
UPMC / M2 SESI / VLSI-2

Routage Global et Détailé



UPMC / M2 SESI / VLSI-2

Outil cgt



UPMC / M2 SESI / VLSI-2

Plan

- P & R alimentation
- P & R horloge
- Stratus netlist
- Stratus P & R
- Python

UPMC / M2 SESI / VLSI-2



- 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, ...
- Catégorie "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 ...

UPMC / M2 SESI / VLSI-2

Python

Documentations (extrait)

- <http://www.python.org>
le site officiel
- <http://www.afpy.org>
association française des utilisateurs de Python
- <http://inforef.be/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)

UPMC / M2 SESI / VLSI-2

Python

Caractéristiques 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)

UPMC / M2 SESI / VLSI-2

Python

les principes

1. Beau est mieux que laid.
2. Explicite est mieux qu'implicite.
3. Simple est mieux que complexe.
4. Complex 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.

UPMC / M2 SESI / VLSI-2

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
```

UPMC / M2 SESI / VLSI-2

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))
```

UPMC / M2 SESI / VLSI-2

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
...
```

UPMC / M2 SESI / VLSI-2

Python

Récupération des arguments

```
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
52681699238856266700490715968
26438162146859296389521759999
32299156089414639761565182862
53697920827223758251185210916
86400000000000000000000000000000

bop %-) fact(100)=933262154439441
52681699238856266700490715968
26438162146859296389521759999
32299156089414639761565182862
53697920827223758251185210916
86400000000000000000000000000000
```

UPMC / M2 SESI / VLSI-2

Python

Types de base

Types simples:

- int entier
- long entier de précision infini
- bool True / False
- float nombre réel double précision
- complex nombre complexe

Conteneurs:

- str chaîne de caractères ascii
- unicode chaîne de caractère unicode (2octets)
- tuple séquence invariable d'éléments hétérogènes ('bonjour', 3)
- list séquence variable d'éléments hétérogènes [34, 'salut', 2.0]
- dict dictionnaire d'éléments hétérogènes {clé:valeur}

UPMC / M2 SESI / VLSI-2

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 1.7976931348623157e308 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 ...

UPMC / M2 SESI / VLSI-2

Python

Chaînes

str	"bonjour" """sur plusieurs lignes"""	'bonjour l'autre'
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.

UPMC / M2 SESI / VLSI-2

Python

Traitement de chaînes

```
a = "bonjour"
a[0] est 'b'
a[-1] est 'r'
a[i:] est la chaîne depuis i jusqu'à la fin
a[i:j] est la chaîne de i à j
a[i:j:k] est la chaîne de i à j par pas de k
```

UPMC / M2 SESI / VLSI-2

Python

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.

UPMC / M2 SESI / VLSI-2

Python

Dictionnaires

- Dictionnaire

{cle1:val1, cle2:val2, ...}

```
d = {'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'] vaut 34
- a = dict()
a['v'] = 23

UPMC / M2 SESI / VLSI-2

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.

UPMC / M2 SESI / VLSI-2

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, (), [], {}

UPMC / M2 SESI / VLSI-2

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 !
```

UPMC / M2 SESI / VLSI-2

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]

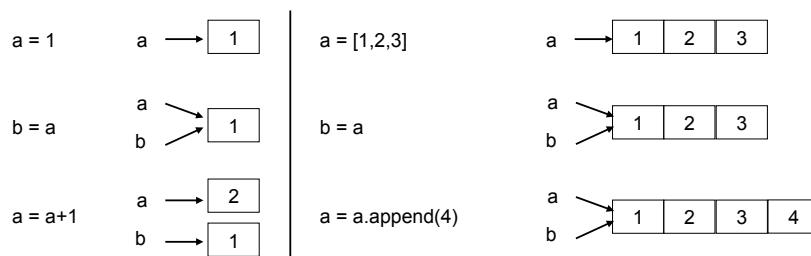
Exemples:

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

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

Valeurs invariables (inmutables)

- les nombres (int, long, float, complex), les chaînes 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.



UPMC / M2 SESI / VLSI-2

Python

Opérateurs

affectation

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

exemples

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

```
"a" * 3 donne "aaa"
['a'] * 3 donne ['a', 'a', 'a']
['a'] + ['b'] donne ['a', 'b']
```

Opérateurs

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

UPMC / M2 SESI / VLSI-2

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

UPMC / M2 SESI / VLSI-2

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.

```
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}

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))
```

UPMC / M2 SESI / VLSI-2

Python

Définition de fonctions

```
def nom (param):
    instructions

return rend une valeur qui peut être quelconque

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
```

UPMC / M2 SESI / VLSI-2

Python

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 = 0 # Vdefaut
    def __init__(self, p1, p2):
        self.attrobjet = p1
        self.__attrprive = p2
    def __str__(self):
        return "ac=%s ao=%s ap=%s"\ \
            (NomClass.attrclass
             ,self.attrobjet
             ,self.__attrprive)
    def m1(self, a1):
        self.attrobjet += a1
i = NomClass(34, 67)
i.attrobjet = 23
NomClass.attrclass = 1
print i
ac=1 ao=23 ap=67
```

UPMC / M2 SESI / VLSI-2

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 chaînes

s.capitalize()
met en majuscules
s.join(sequence)
fabrique une chaîne en concaténant les éléments de séquence entre lesquels on met la chaîne 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

UPMC / M2 SESI / VLSI-2

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"
 - print "mess2",
 - print "format" % (valeurs)mess1 + CR
mess1 sans CR
mess formaté

Fichiers

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

Python

UPMC / M2 SESI / VLSI-2