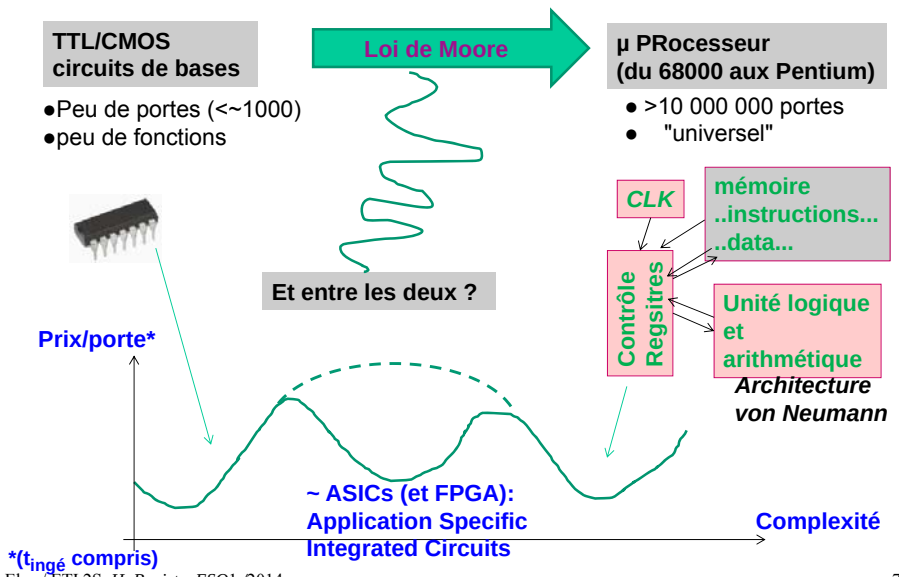
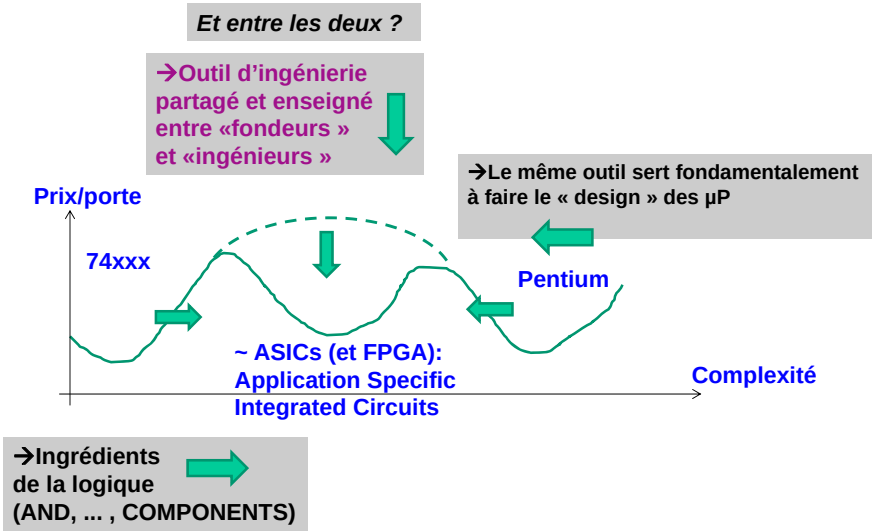
COURS 3 : langage VHDL

- Premier aperçu des niveaux de complexité l'électronique numérique
  - → TTL, ASICs, (micro)processeurs
  - → déroulement temporel, mémoire : « le séquentiel »
    (cours suivant)
- Le langage VHDL
  - → Lien au hardware : PORT, signaux, t_p
  - → Distinctions « Entity »/ « architecture »
  - → Structural / Concurrent : « ambiguité »
  - → [Séquentiel] process
  - → Types, bibliothèques
  - → (pas de syntaxe dans ce cours)

Du TTL/CMOS au microprocesseur

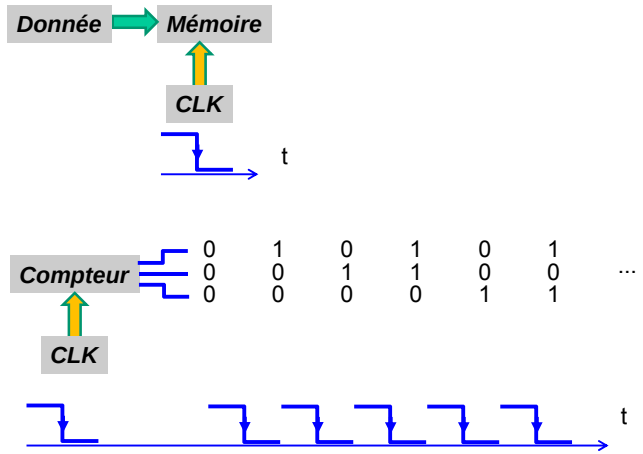


Le VHDL = équipement d'escalade



Le séquentiel : « avant-première »

Le séquentiel version hardware



Le séquentiel : notion délicate ...

Le séquentiel version software ?

while (condition) {action} ?

~ Mode « polling » (interrogation des besoins au passage):
pas de timing garanti car
dépend du t de passage de l'instruction au point considéré !

```
Acheter des timbres
Répondre aux mails
Eplucher une pomme
while (N'y a plus de lait) {demander du lait à ma voisine}
while (Y'a le feu) { appeler 18 , chercher extinteur}
Offrir des fleurs à ma belle-mère
...
```

? C'est grave ?

→C'est le plus dur en VHDL : ne plus penser en « suite d'instruction » !

Définition générale du VHDL

Langage pour définir

Un « méta-composant »

Fait de composants

Ayant des PORTs d'entrée/sortie

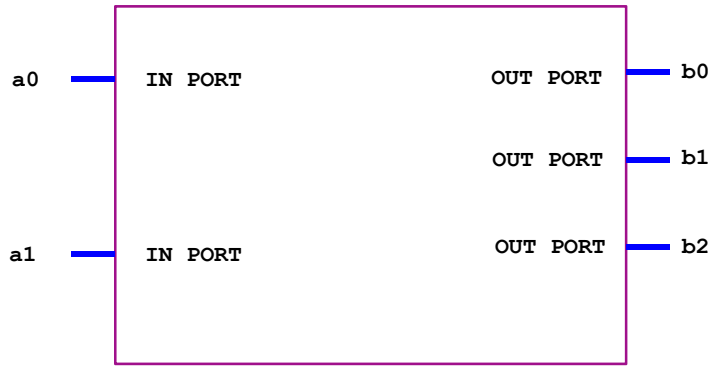
Les composants ayant leur ports

+ Des outils pour décrire « ce qu'il y a dedans »

- **Architecture**
- **Process**
- **Signal**
- **Variable**
- **« standard gates »**

VHDL : ENTITY

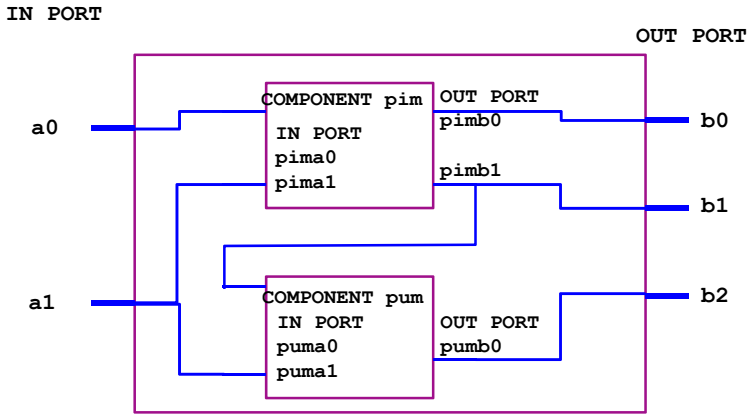
Un « méta-composant » : entity




```
PORT(a1,a2 : IN bits;  b0, b1, b2 : OUT bits)
```

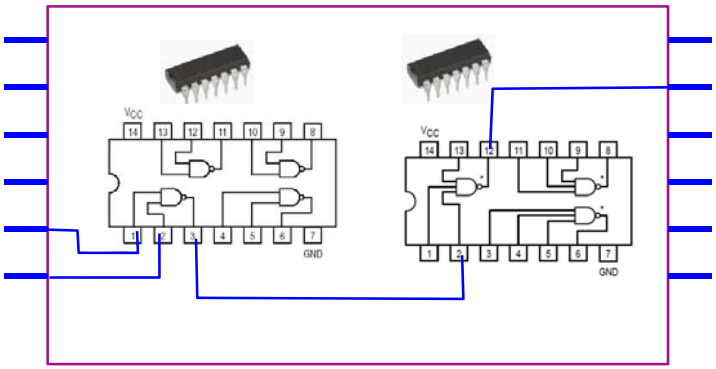

VHDL : COMPONENT

Composants dans une « entity »



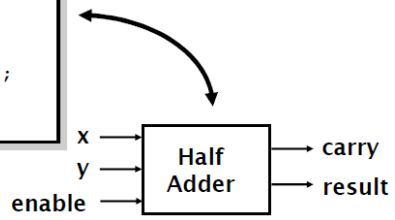
VHDL : C O M P O N E N T

Composants dans une « entity »



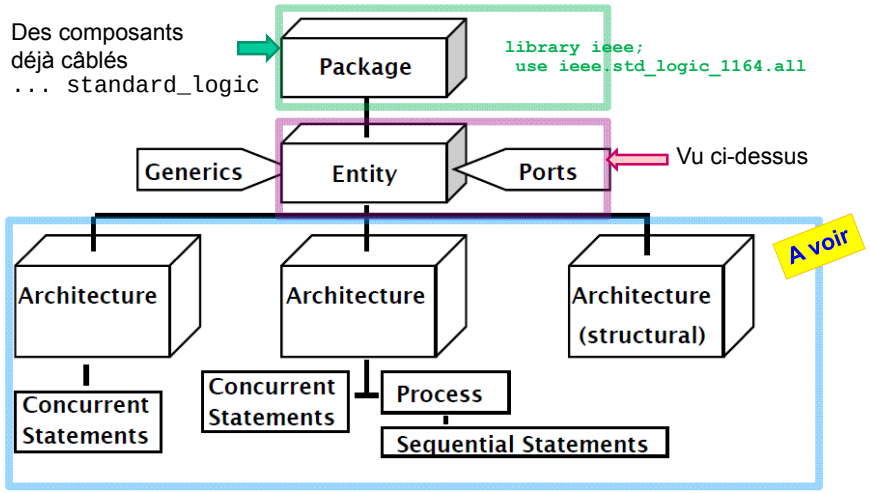
VHDL : ENTITY & PORTS

```
ENTITY half_adder IS

   PORT( x, y, enable: IN bit;
         carry, result: OUT bit);

END half_adder;
```



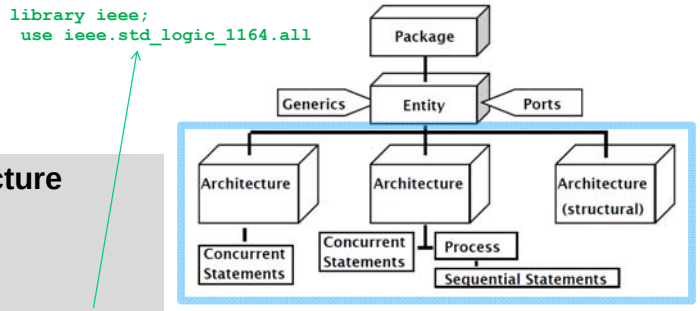
« demi-additioneur »
(sans retenue entrante)

VHDL : UN PETIT COUP DE MACROSCOPE



VHDL : PARTIE « ARCHITECTURE » (~ le dedans)

+ Des outils pour décrire
« ce qu'il y a dedans »



Architecture
Process
Signal
Variable
« standard gates »

VHDL : UN 'PROGRAMME' AVEC ARCHITECTURE

(tiré du classique « demi-additionneur »)

→Action directe sur les ports

```
library ieee;
 USE ieee.std_logic_1164.all;

 -- voici la définition de notre entity
 entity eti2222 is
     port (a, b : in BIT;
           sum, c : out BIT);
 end eti2222;

 architecture gloups33 of eti2222 is

 begin
    -- concurrent statement   porte and
   c <= a and b;
    -- concurrent statement   porte xor
   sum <= a xor b;

 end gloups33;
```

PACKAGE

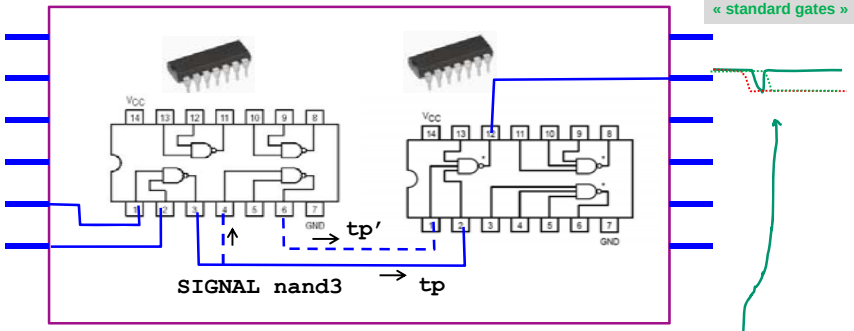
ENTITE: PORTS

ARCHITECTURE

```
variante : (a, b : in std_logic     --etc...
```


VHDL : SIGNAL

Architecture
Process
Signal
Variable
« standard gates »



SIGNAL

→ correspondra à un fil physique

→ Concrétise dans la réalité des « intermédiaires de calcul »

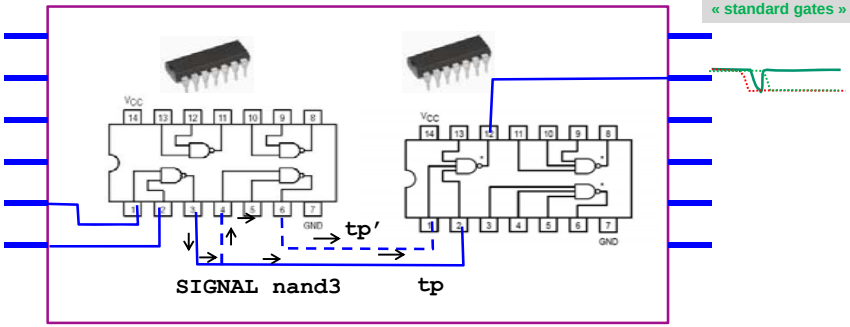
```
SIGNAL nand3:BIT ;
```

→ Un délai (t_p) spécifique pourra/devra être associé → Scénarii de sortie

→ Un signal peut avoir plusieurs bits, ou être un autre TYPE que bit, par exemple un INTEGER

VHDL : SIGNAL

Architecture
Process
Signal
Variable
« standard gates »



DIRE QUI SE CONNECTE A QUOI

VHDL : UN 'PROGRAMME' AVEC SIGNAL

Architecture
Process
Signal
Variable
« standard gates »

→Action des ports sur les signaux, puis des signaux sur les ports

```
library ieee;
 USE ieee.std_logic_1164.all;

 -- voici la définition de notre entity
 entity eti4444 is
     port (a,b,c: in ...; s0,s1: out...);
 end eti4444;

 architecture gloups55 of eti4444 is
 signal q2,q1,q0 unsigned (2 down to 0)
 begin
    q0 <= a and b
    q1 <= c xor q0
    q2 <= a xor q1

    s0 <= q0 or q1
    s1 <= q2 xor b
 end gloups55;
```

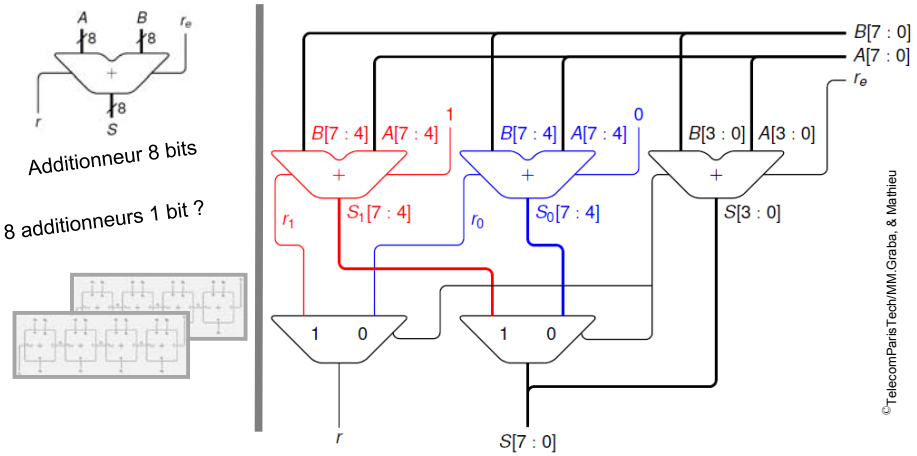
PACKAGE

ENTITE: PORTS

ARCHITECTURE

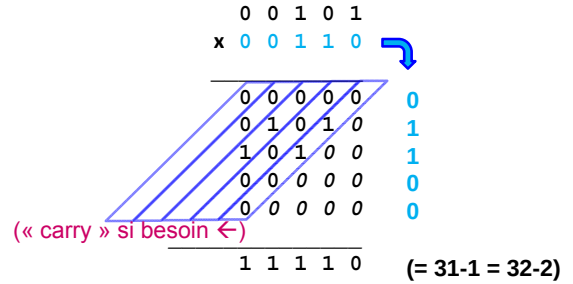
(pas que VHDL) : STRATEGIES DE TIMING

→Architecture à ~100 portes pour une opération simple...
Qu'a-t-on pu optimiser par rapport à une version + simple ?



(pas que VHDL) : STRATEGIES DE TIMING

→La question du timing dans la multiplication ?



→Divers choix, *(les derniers flirtant les limites du combinatoire vers le séquentiel)*
Attendre le temps maximum par précaution ?
Prévenir dès que c'est fini ?
Prévenir avant pour certains cas particuliers « prévisibles »
(« 0 »...) ?

VHDL : LES TYPES

langage très « typé »

→ ~ Pas traités dans ce cours

- `bit`
- `bit_vector` + de quoi le nombre de bits (<0:2> et qui est le LSB)
- `std_logic`
- `std_logic_vector`
- `integer` existe en version signée et non signée

+ des réels, des temps (intrinsèque à VHDL : la nanoseconde), ...

→ ~ Règles d'écriture ...

```
a="00"   -- variable de 2 bits ('0' ou '1' pour 1 bits)
a='0' & '0'   -- concaténation
```


VHDL « STRUCTURAL »: CONSEQUENCES « inattendues »

- Une entité → Plusieurs architectures possibles
  [on s'en fiche si fonctions & délais corrects/compatibles !]

→ Par exemple $y=(a+b)\times c$
ou $y= (a\times c)+(b\times c)$ [distributivité] : deux modes de calcul

Ou encore $a\times b=((a+b)^2-(a-b)^2)/4$:
pas besoin de multiplication générale pour multiplier

- L'ordre du programme « structural » NE DIT RIEN de la logique temporelle
  Toutes les opérations logiques combinatoires décrites sont « simultanées »!

→ A part les délais (qui sont à programmer par/pour les spécialistes),
il faut s'efforcer de « repenser en parallèle »

VHDL : Exemple simultanéité

```
SIGNAL x,y,a,b,c:BIT

begin

   b <= x and y
   c <= b
   b <= a

end
```

conflit !
`b` n'est pas « libéré » par `c<=b`
mais seulement « connecté/attribué »

VHDL : QUARTUS

- Avec Quartus
  - L'aspect structurel → **Graphique**
  - L'aspect algorithmique → **textuel Vhdl**
- VHDL est un langage de description
- 3 types de descriptions :
  - Par flots de données :
    - Association de portes logiques
    - `S <= A and B and (not C);`
  - Comportementale :
    - Par description d'un algorithme
    - Utilisation de when ..., de process()
  - Structurelle :
    - Par association de composants (components) et instanciation

→Dans *Quartus* : associer des schémas blocs (.bdf)

VHDL : SEQUENTIEL/PROCESS

mise en place

Architecture
Process
Signal
Variable
« standard gates »

```
architecture gloups55 of eti4444 is

signal a,b,c : integer

begin

   process (a,b,c)
   begin
    variable x,y :integer

       x:= a+b
       c <= x+1;

   end process


end gloups55;
```

Sensitivity list

Variable

Sensitivity list:

Possibilité de sensibilité
en général et de
sensibilté aux changements
en particulier

sur tous signaux & ports d'entrée

Variables:

Dans le « process »
traitement très différent des signaux
...

VHDL : SEQUENTIEL/PROCESS

les variables

Architecture
Process
Signal
Variable
« standard gates »

→ **On n'a droit aux variables que dans les process**

→ ***Ce ne sont que des « auxiliaires de calcul »***

→ **Affectation instantanée**
- **Pas de gestion de délais,**
- **pas d'attente d'un évènement pour valider.**

→ **Commodité surtout pour le programmeur !**

Par exemple (provisoire):
→ le produit (a<2:0>)× (b<2:0>) est-il multiple de 3 ?
(résultat entre 0 et 49)
On a besoin de la réponse (1 ou 0)
mais pas des signaux intermédiaires afférents, ceux du produit...
« qu'il se débrouille pour le faire dans un coin » !
(le compil connait Karnaugh mieux que nous)
(m'enfin on rase pas gratis)

VHDL : SEQUENTIEL/PROCESS

les signaux

Architecture
Process
Signal
Variable
« standard gates »

→ Sont traités séquentiellement dans les process

→ *Ne sont affectés qu'à la fin du process*

VHDL : SEQUENTIEL/PROCESS

Dépendance temporelle dans le process

Architecture
Process
Signal
Variable
« standard gates »

→ mots-clés du type pour expliciter le vrai « quand... »

```
wait until(clk='1');
wait until rising_edge(clk);
```

→ toute structure de choix

```
process (A,B,C,D,SEL)
      begin
          case SEL is
             when "00" =>   S <= A ;
             when "01" =>   S <= B ;
             when "10" =>   S <= C ;
             when others => S <= D ;
          end case;
      end process;
```

→ Attention à l'ambiguité du langage courant:
une fonction combinatoire s'exprime avec « quand » !
« quand tous les bits sont à 0, alors... »

VHDL : SEQUENTIEL/PROCESS

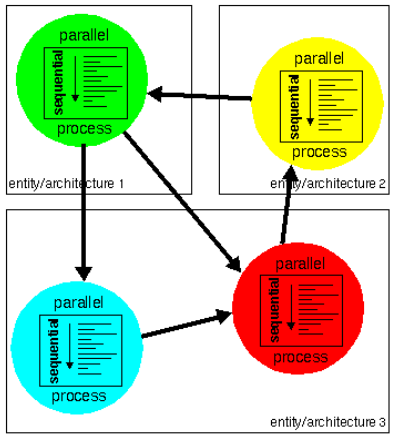
Dépendance temporelle dans le process

Architecture
Process
Signal
Variable
« standard gates »

→ Parallélisme des entités/architectures

→ **Execution séquentielle dans les process**

A l'utilisateur de prendre soin des listes de sensibilité (les flèches noires)



VHDL : SEQUENTIEL/PROCESS vs. STRUCTURAL

Architecture
Process
Signal
Variable
« standard gates »

```
entity eti6666 is
   port (A, B, C, X : in    bit_vector (3 downto 0);
                 YYY : out   bit_vector (3 downto 0);
                 ZZZ : out   bit_vector (3 downto 0)  );
end eti6666;
architecture  gloups77  of eti6666  is
begin

   -- Concurrent version of conditional signal assignment
   YYY <= A  when X = "1111"   else   B when X > "1000"   else   C;
   -- Equivalent sequential statements
   process (A, B, C, X)
   begin
       if  (X = "1111")  then
         ZZZ <= A;
       elsif  (X > "1000")  then
         ZZZ <= B;
      else
         ZZZ <= C;
       end if;
   end process;

end  gloups77;
```

Les résultats `YYY` et `ZZZ`
sont les mêmes... !
Toutefois le timing pourrait différer.

VHDL : SURTOUT NE PAS CONCLURE !

Suivant votre appréciation :

- Mal nécessaire pour gérer certaines choses
  (comme les fonctions d'onde,la majorité des ingénieurs fini par s'en passer !)

- Informe sur la réalité d'un process/ d'un processeur
  (et nous re-forme notre cerveau)
  - Parallélisme ~ puissance cachée de l'algorithmique
  - Nous sommes victimes de notre perception
    « algo = recette à appliquer ».
  - Cette recette connait deux obstacles
    → algorithmique récursive
    → algorithmique parallèle
  - Force de VHDL : aucun n'est un obstacle !
  - Des architectures complexes de « logique » généralisée restent la clé des progrès futur

On apprend la « loi de la panne »:

"Ce n'est qu'après qu'on a pensé à tout qu'on trouve à quoi on n'a pas pensé"

Un peu plus sur Von Neumann vs. Réseaux de Neurones vs. Reservoir Computing si j'ai le temps (?)