Evolution des méthodes de conception de circuits intégrés

Emmanuel Boutillon

LESTER
Université de Bretagne Sud
Tél : 02 97 87 45 66
Fax : 02 97 87 45 00
emmanuel.Boutillon@univ-ubs.fr

PLAN

Introduction

Circuits numériques spécifiques à une application (ASIC, ASIP)

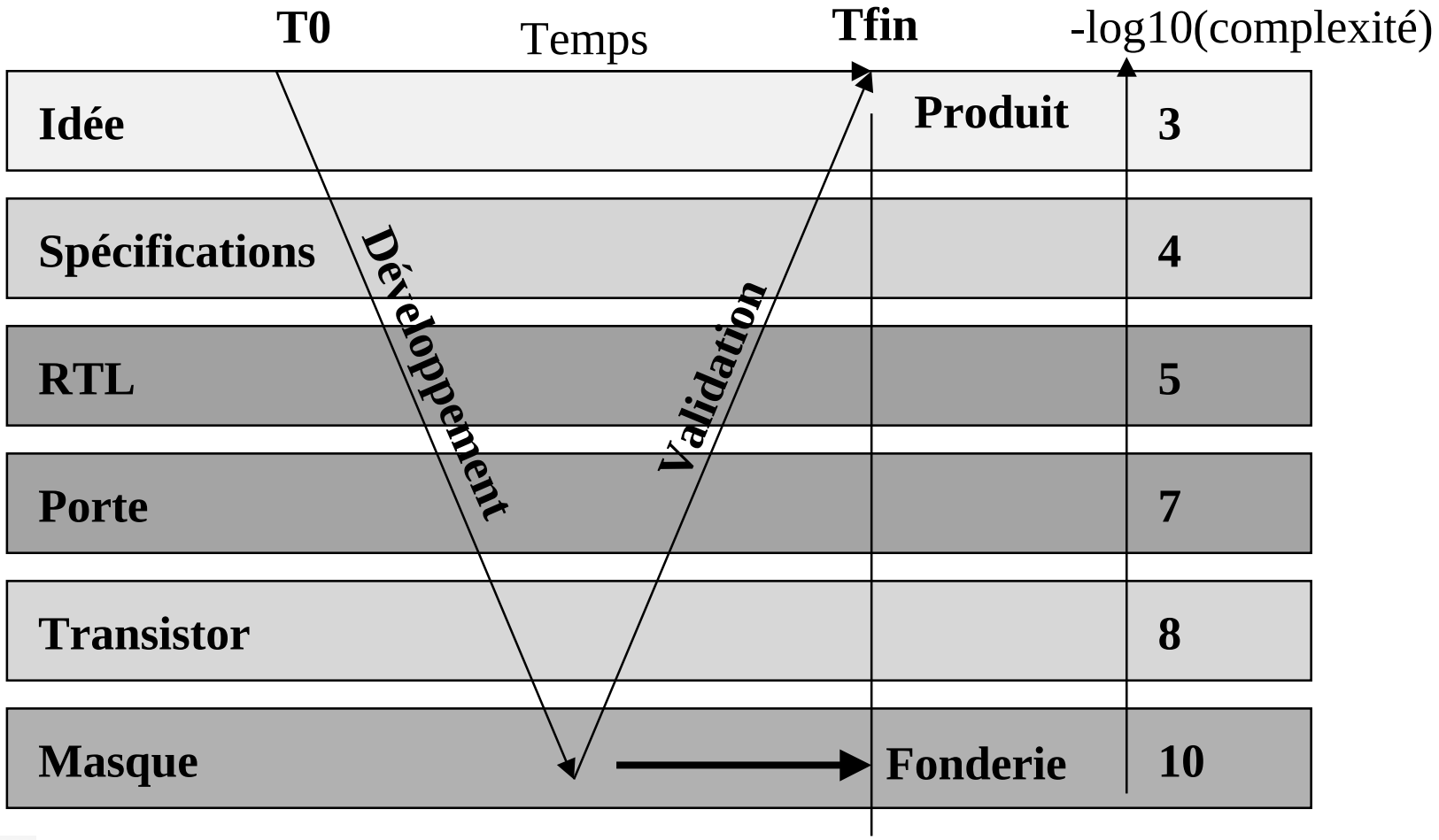
Traitement des images, Communication, Réseaux hauts débits…

Méthodologie de conception « industrielle »

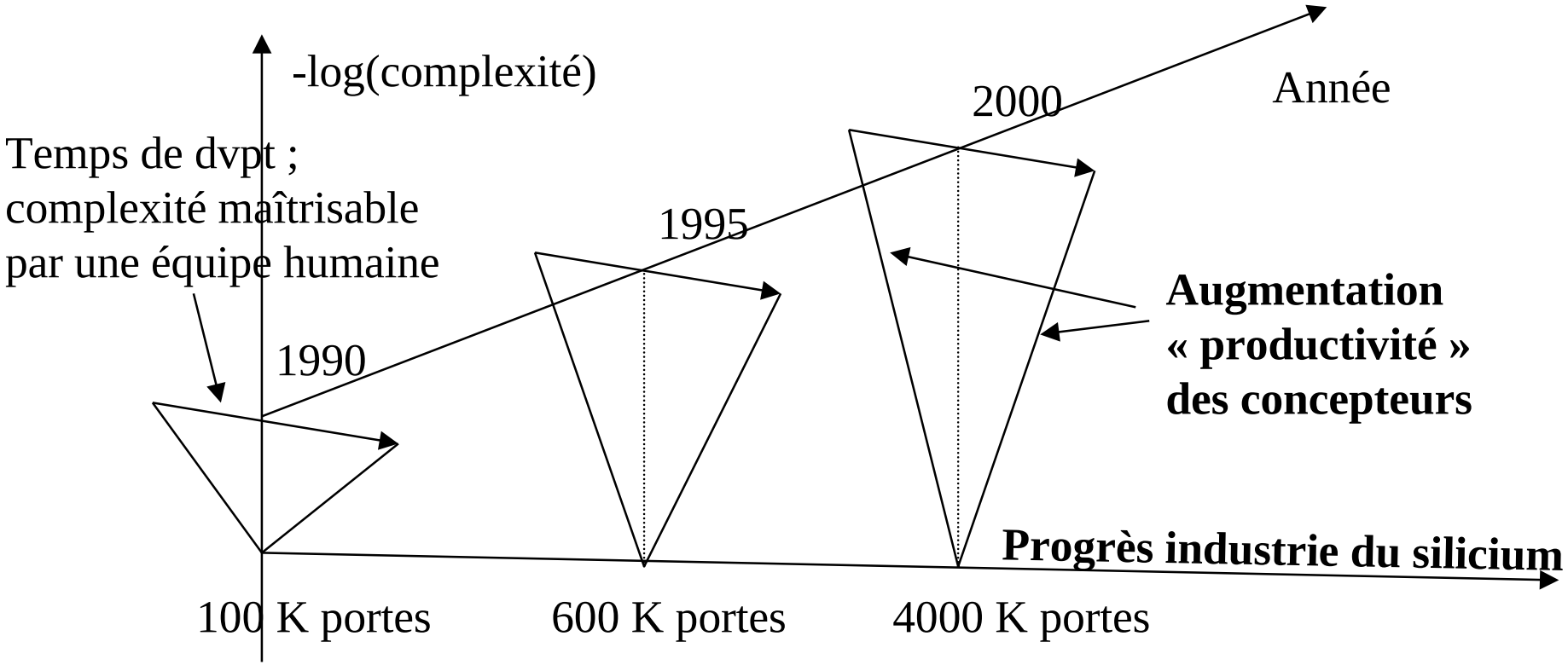
A partir de l'idée … jusqu'à un dessin des masques « garanti » par le fondeur.

Technologie CMOS, logique complémentaire et synchrone

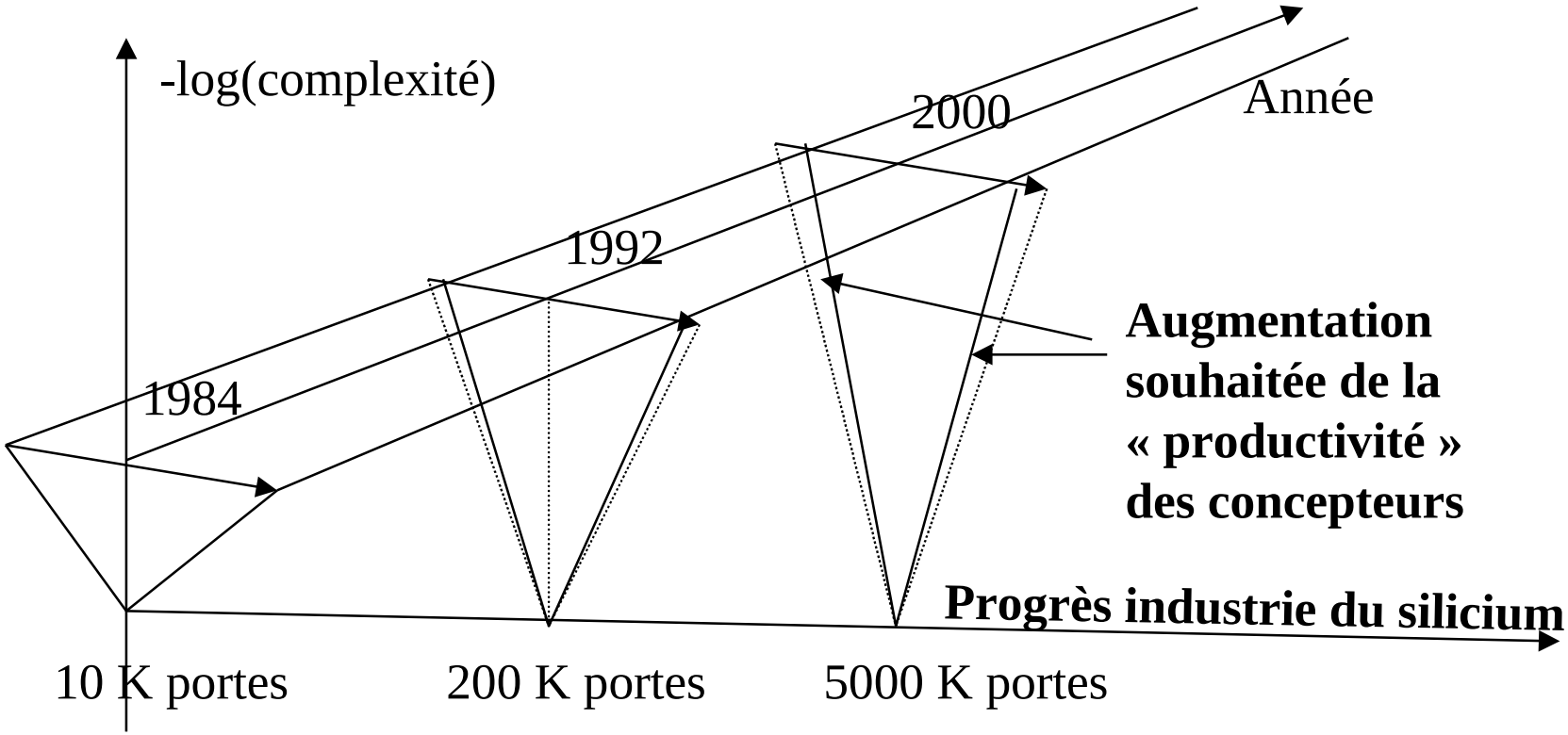
Flot symbolique de conception



Version 3D de la courbe de Moore

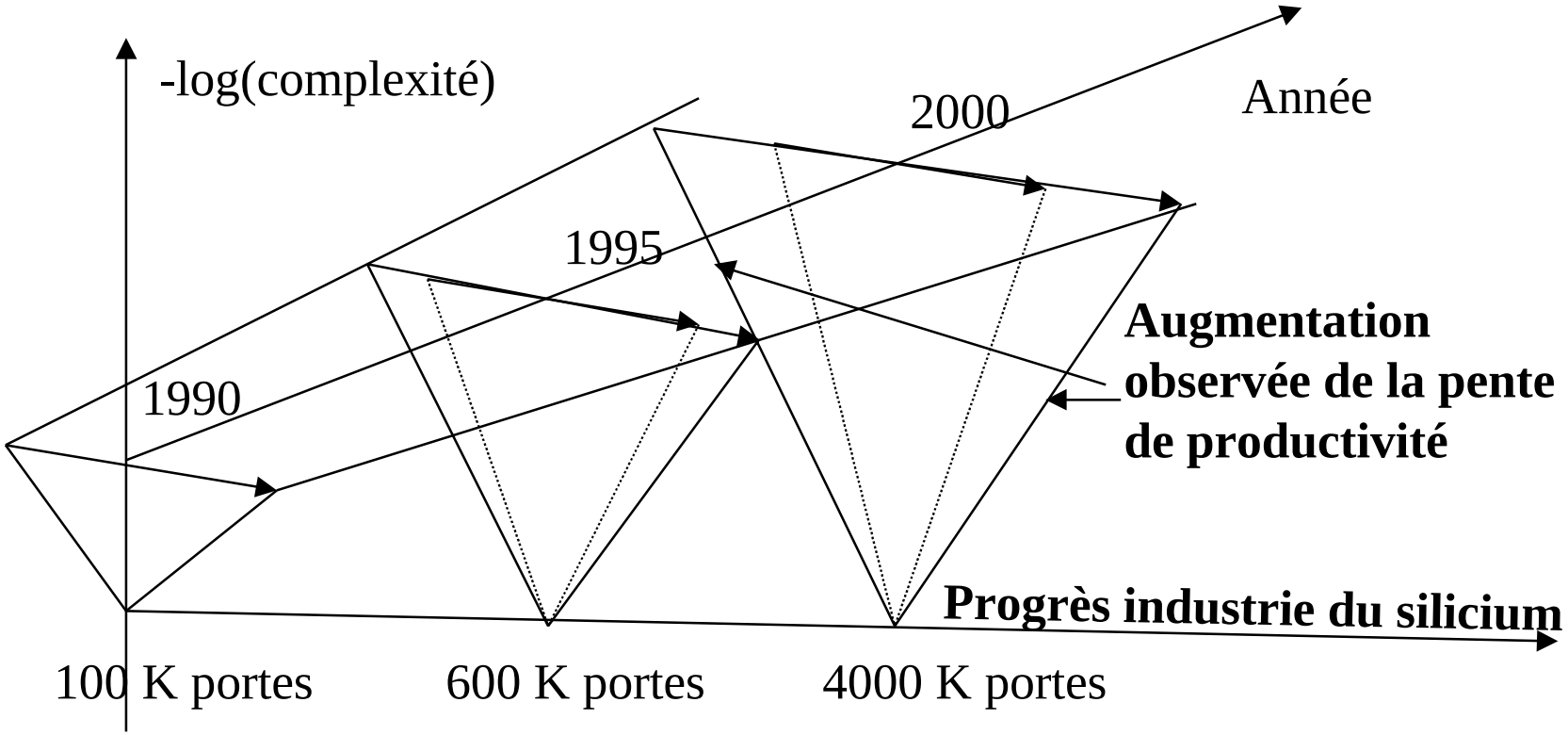


Contrainte du marché...



Temps de conception des circuits de plus en plus court...

Version 3D de la courbe de Moore



Augmentation de la productivité des concepteurs

*** Expérience accumulée**

- **Réutilisation (Virtual Component)**
- **Méthodologie (mise en place dans les flots de conception des outils de CAO)**
- **Formalisation des problèmes (Langage de description)**

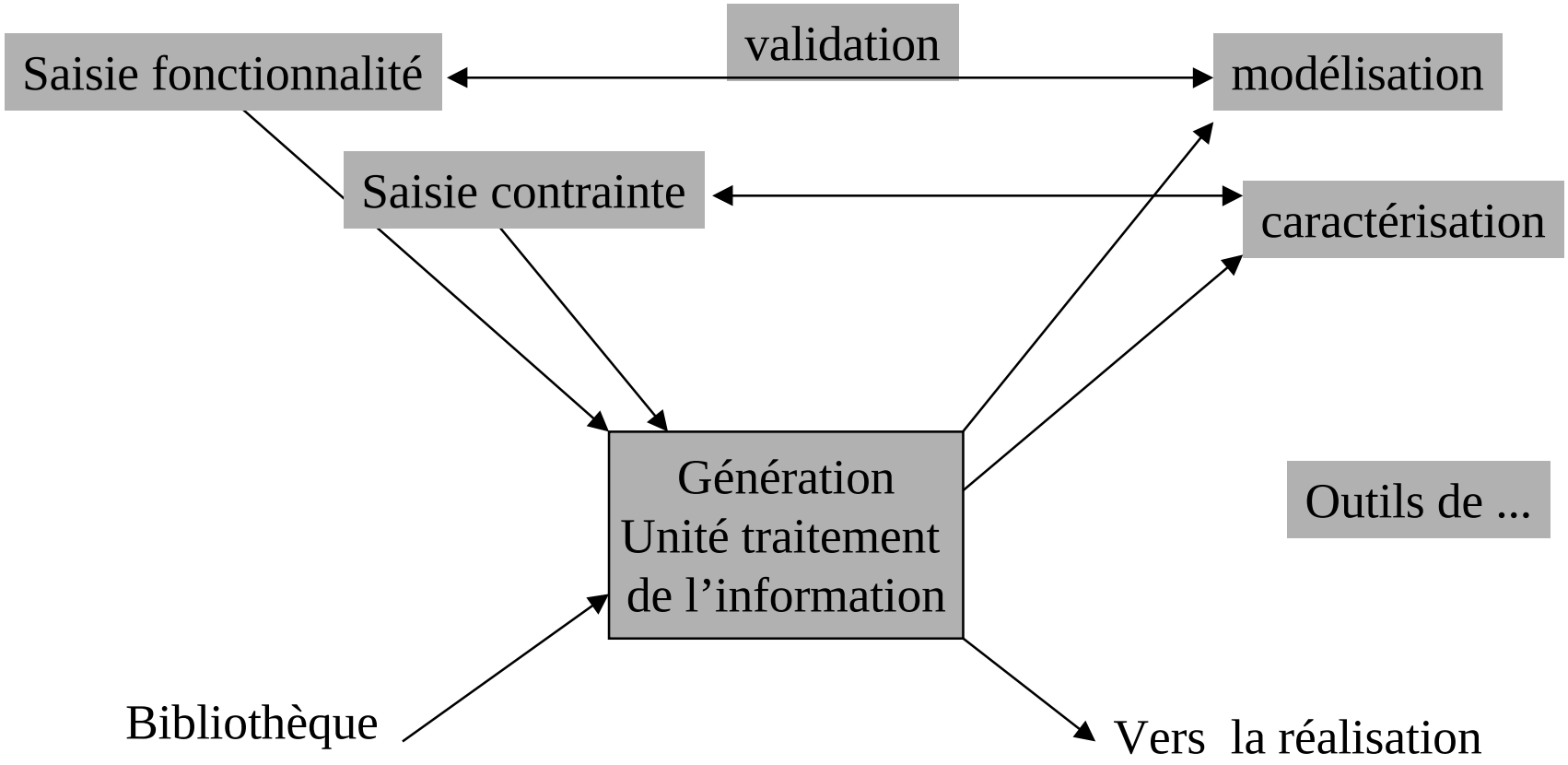
*** Outils informatiques**

- **Quantitatif (+ de puissance de calcul, + de mémoire)**
- **Qualitatif (compilateur, algorithme, programme plus fiable)**

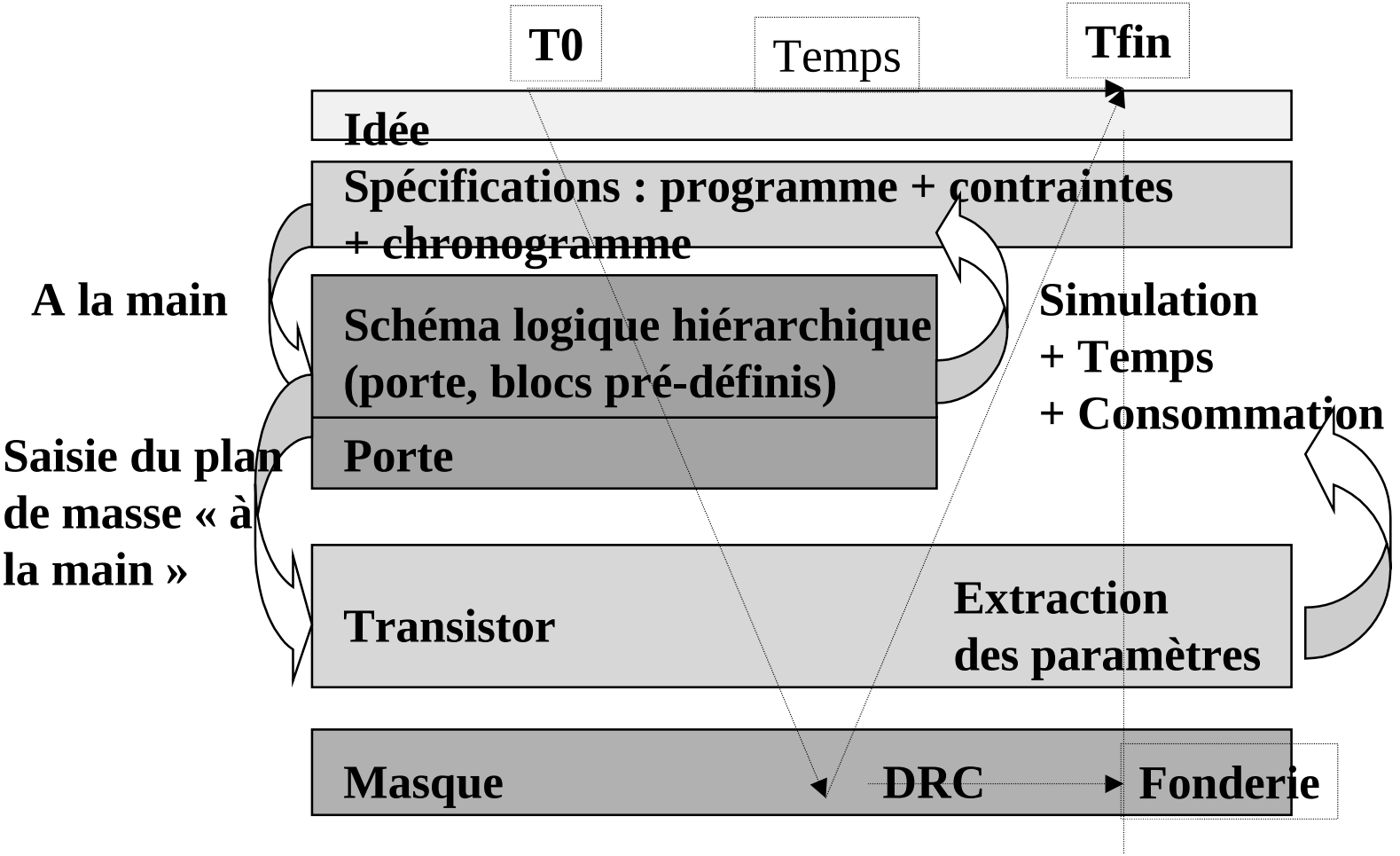
*** Simulateur matériel (simulation de code RTL (Register Transfert Level) et de netlist sur cible FPGA)**

Conception d ’une unité de traitement de l ’information

Spécifications =



Conception dans les années 85-90



Les verrous...

1) Passage d'un format de description à l'autre

2) Niveau d'abstraction du point d'entrée des outils de CAO

3) Problèmes de vérification

PLAN

1. Introduction
2. Langages de description de matériel
3. La synthèse d ’architecture
4. La preuve formelle
4. Conclusion

Langage de description matériel

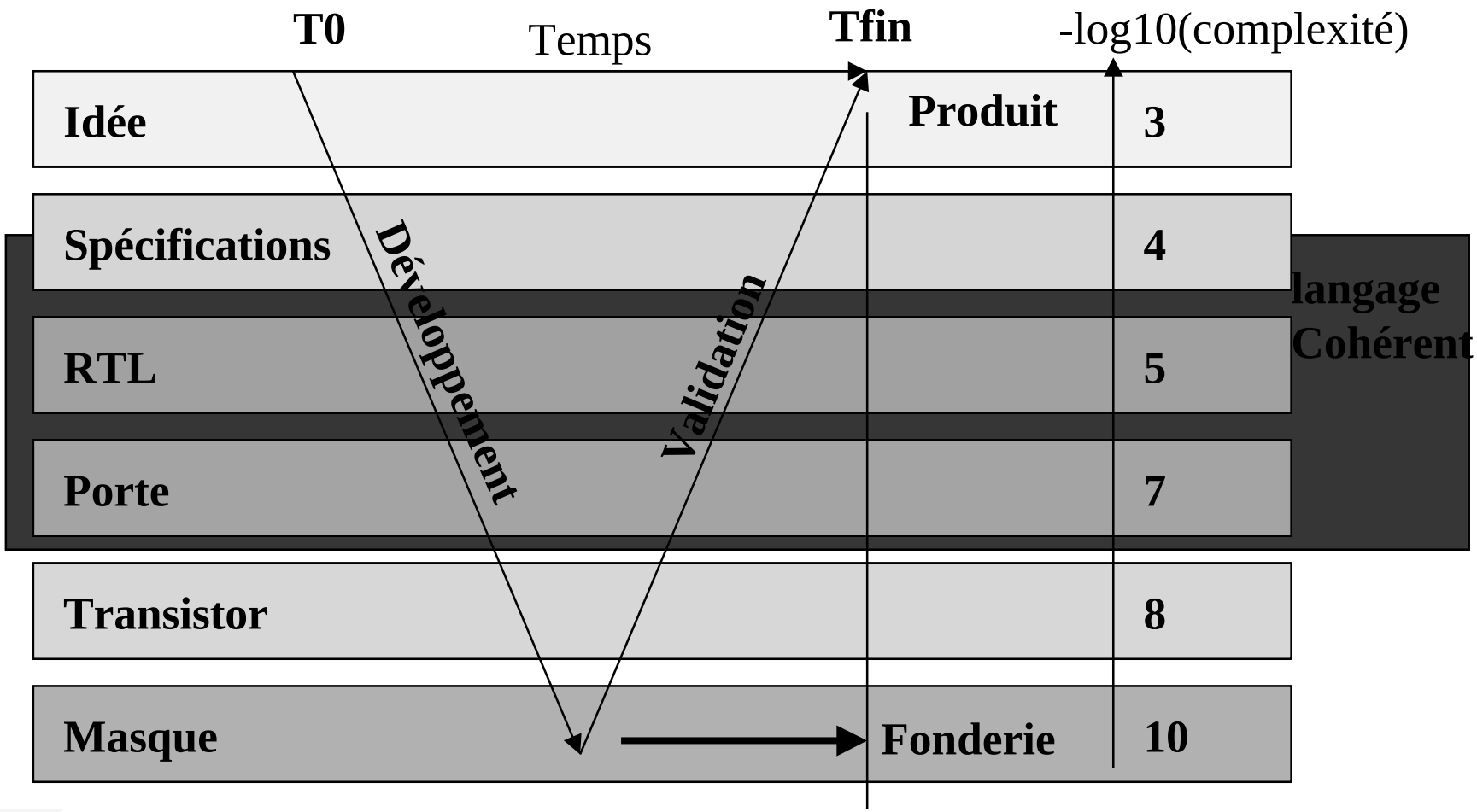
Langages permettant

- de décrire un système comme assemblage de composants (avec la notion de hiérarchie).
- de simuler des processus s'exécutant en parallèle
- de gérer plusieurs niveaux de description d'un même objet.
  - comportemental
  - transfert de registres (RTL)
  - structurel

De plus, ces langages peuvent servir de point d'entrée aux synthétiseurs logiques.

Gain de productivité très important.

Flot symbolique de conception



Exemple de code rtl synthétisable

```
--  Test if P picture and bwd prediction
IF (RF.Picture_Frame = P) AND (RF.Motion_decision = bwd) THEN
   IF RF.Picture_Struct = Frame THEN
    ------------------------------------------------------
    -- Case of frame picture: If not_coded, the MB should be
    -- skipped exept if the MB is first or last in the slice
    ------------------------------------------------------
    IF NOT MB_coded(RF.CBP, chroma)
       AND (RF.MB_start_of_slice OR RF.MB_end_of_slice) THEN

       RF.Motion_decision := fwd;
       MV(fwd_top_X) := 0; MV(fwd_top_Y) := 0;
    END IF;
```

=> Les synthétiseurs rtl acceptent un haut niveau d'abstraction

Fichier de contrainte

Le synthétiseur reçoit, en plus du code rtl, un fichier de contraintes qui lui indique :

- les contraintes de temps

- les contraintes de surface

- un modèle d'estimation des capacités de routage

- l'ordre séquentiel des synthèses et des optimisations à effectuer (fruit de l'expérience)

mais ils ne sont pas exempts de bugs, par exemple...

X un enregistrement (X.a entier et X.b un booléen)
Dans un processus, se trouvent les lignes suivantes :

Si (condition 1) appel d'une fonction ayant X comme argument
et modifiant la valeur de X.a
Si (condition 2) appel d'une autre fonction ayant X comme
argument et modifiant la valeur de X.b

=> Après synthèse, si condition 1 et condition 2 sont vraies, seul X.b est modifié

Résultat : dans de très rares cas, il y a non conformité entre la simulation du rtl et de la netlist synthétisée.

=> Problème très difficile à identifier

PLAN

1. Introduction
2. Langage de spécification matériel
3. La synthèse d ’architecture
4. La preuve formelle
5. Conclusion

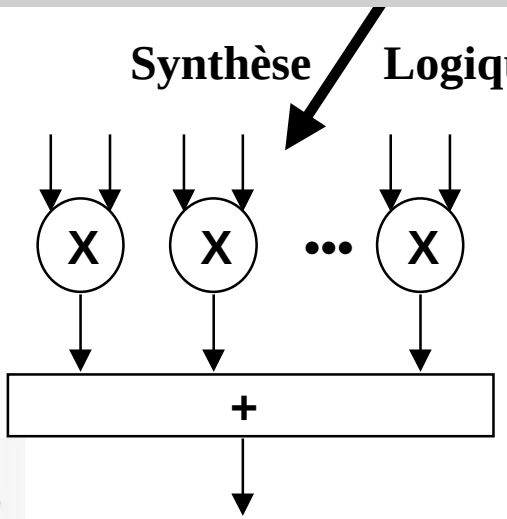
Synthèse d'architecture

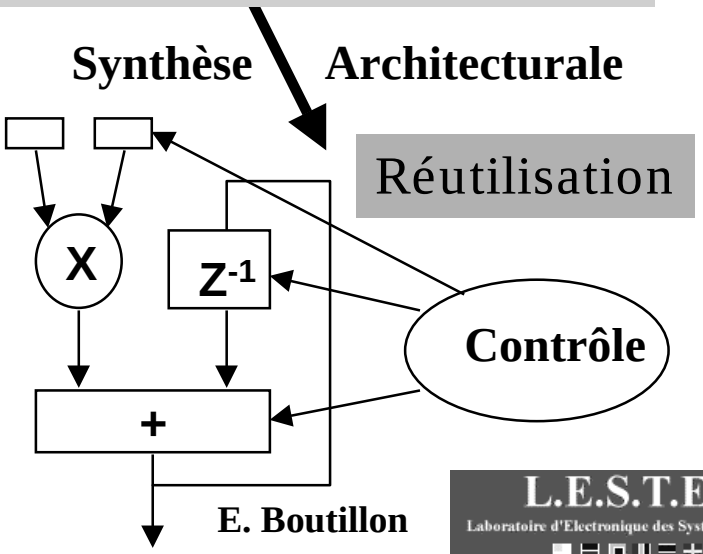
- Synthèse d'architecture (HLS)
  - ⇨ Synthèse RTL :
    - Part d'une description RTL ; le comportement est fixé à chaque cycle d'horloge ;
    - Repose sur la synthèse logique pour optimiser le circuit ;
    - Génère une liste de cellules standards
  - ⇨ Synthèse HLS :
    - Part d'une description comportementale abstraite (algorithmique)
    - Génère une description RTL
    - Exploite les propriétés de la spécification (concurrence, hiérarchie,…)

Synthèse d'architecture

```
-- X et Y sont supposés être des tableaux d'entiers de dimension N
-- S est un signal entier compatible avec la définition de la variable Z

P_calcul : process
        variable Z : integer16;
begin
      wait until H_cycle = '1';        -- conditions d'horloge
      Z := 0;                  -- initialisation
      for i in X'range loop            -- N itérations
         Z := Z+ X(i) * Y(i);          -- multiplications accumulations
      end loop;
      S <= Z;                          -- sortie du résultat synchrone de l'horloge H_cycle
end process P_calcul;
```





E. Boutillon
L.E.S.T.E.R
Laboratoire d'Electronique des Systèmes TEmps Réel

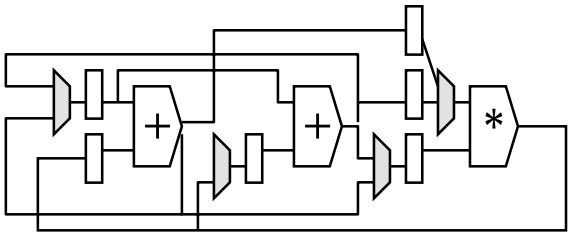
Spécifications

- Langage de programmation conventionnel
  - ⇨ Pascal, C
- Langage de description du matériel
  - ⇨ VHDL, SpecC, SystemC, Rosetta, HadwareC, Superlog
- Description graphique
  - ⇨ SpecCharts, StateMate, StateCharts, SDL, Petri

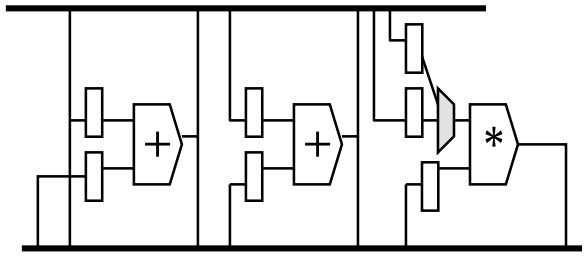
Après compilation ces spécifications aboutissent à une représentation interne.

Modèle d'architecture

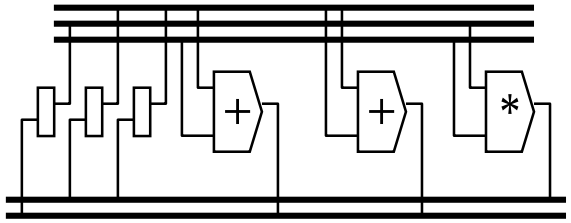
- Modèles d'architecture : unités de traitement



Modèle à base de multiplexeurs :
+ : Les registres sont partagés ;
- : Les chemins sont « point à point »



Modèle à base de bus :
+ : Les chemins sont « communs »
- : Les registres sont « localisés »
i.e. en entrée des opérateurs



Modèle à registres généraux :
+ : Les registres sont partagés
- : Les bus peuvent être nombreux
dépend du nombre d'opérateurs

Le modèle de structure contraint l'utilisation des ressources

Exemple de spécification (Gaut)

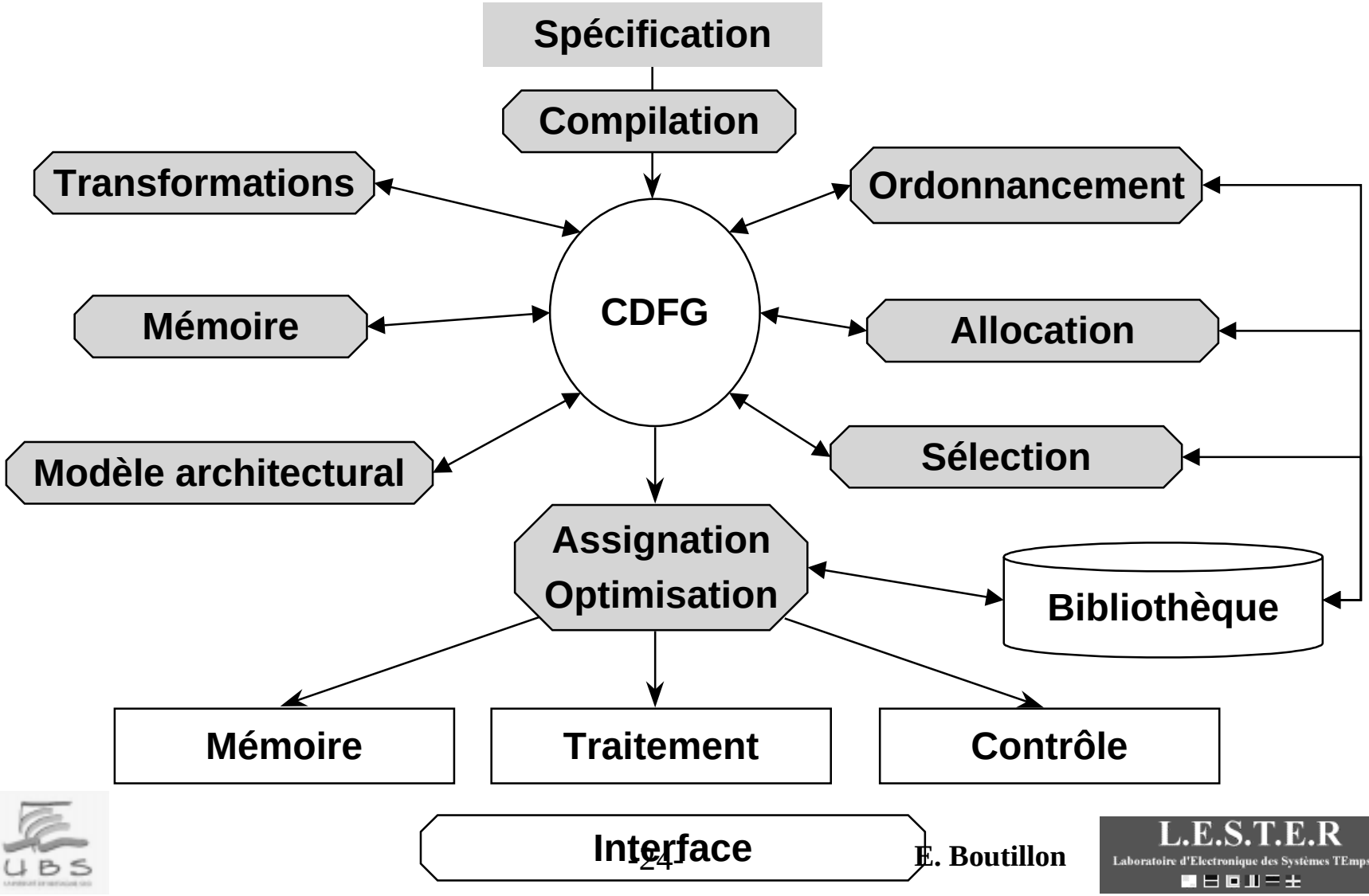
- VHDL

```
ENTITY fir IS
   PORT (xn:IN INTEGER; yn:OUT INTEGER);
END fir;

ARCHITECTURE behavioral OF fir IS
   ...
BEGIN
   PROCESS
       VARIABLE H,x:      vecteur;
       VARIABLE tmp:      INTEGER;
   BEGIN
       tmp := xn * H(0);
       FOR i IN 1 TO N-1 LOOP
             tmp := tmp + x(i) * H(i);
       END LOOP;
       yn <= tmp;
       FOR i IN N-1 DOWNTO 2 LOOP
             x(i) := x(i-1);
       END LOOP;
       x(1) := xn;

       WAIT FOR cadence;
   END PROCESS;
END behavioral;
```

Anatomie d'un outil de synthèse



Technique de synthèse

- **Compilation**
  - Analyser le code de la spécification, paralléliser et transformer le code afin d ’obtenir une représentation interne exploitable
- **Sélection**
  - Sélectionner dans une bibliothèque d'opérateurs les composants dont les caractéristiques répondent aux contraintes de l'application.
- **Allocation**
  - Déclarer la quantité exacte d'opérateurs que l'on désire utiliser, ainsi que les intervalles de temps où ils sont utilisables.
- **Ordonnancement**
  - Affecter une date d'exécution à chaque opération de l'application.
- **Assignation**
  - Affecter une opération à un opérateur.

Compilation et transformations

- Propagation des constantes

Avant : j := 4
k := j*2

Après : j := 4
k := 8

- Elimination des expressions communes

Avant : j := b + sinx*b
k := (sinx*b)*2

Après : temp := sinx*b
j := b + temp
k := temp * 2

- Elimination des boucles des invariants

Avant : Pour i depuis 1 à 10 faire
a(i) := x + y
fait

Après : temp := x + y
Pour i depuis 1 à 10 faire
a(i) := temp
fait

- Remplacement d'instructions

Avant : Pour i depuis 1 à 10 faire
a(i) := i*4
fait

Après : t := 0
Pour i depuis 1 à 10 faire
a(i) := t; t := t + 4
fait

Compilation et transformations

- Déroulage de boucles (loop unrolling)
  - ⇨ Permet d'augmenter le parallélisme dans la boucle
  - ⇨ Déroulage total (implicite) dans la plupart des cas
  - ⇨ Déroulage partiel :

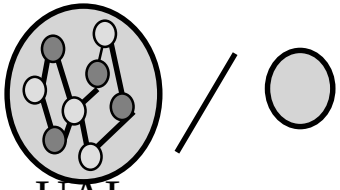
Avant :
```
Pour i depuis 1 à 100 faire
    a(i) := a(i) + b(i)
    i := i + 1
fait
```

Après :
```
Pour i depuis 1 à 100 faire
    a(i) := a(i) + b(i)
    a(i+1) := a(i+1) + b(i+1)
    i:=i+2
fait
```

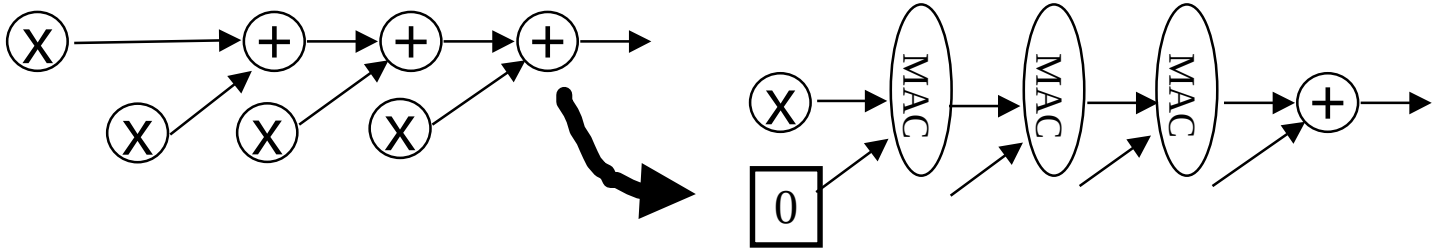
S'il y a 2 additionneurs dans l'unité de traitement, le temps d'exécution est divisé par 2.

Transformations

- Fonctionnelles

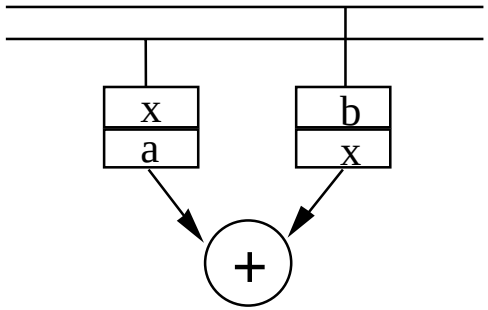


  - ⇨ Réalisation algorithmique ou matérielle
    - Exemple Multiplieur câblé ou Multiplieur µprogrammé sur UAL en utilisant e.g. l'algorithme de Booth
  - ⇨ Remplacer une (ou des) opérations par une (ou des) opération équivalente.
    - Exemple multiplication par une constante -> add/décalage
    - MAC



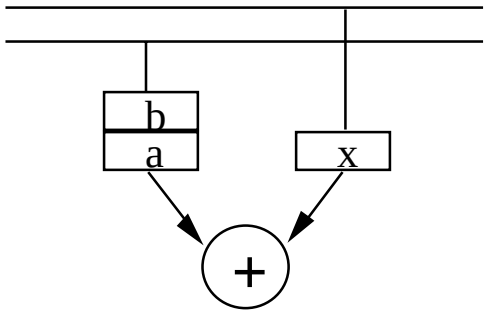
Transformation

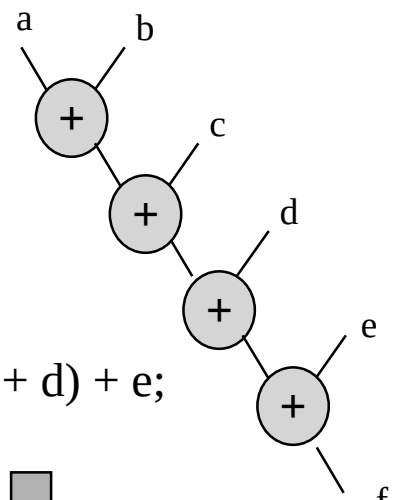
- Commutativité, Associativité



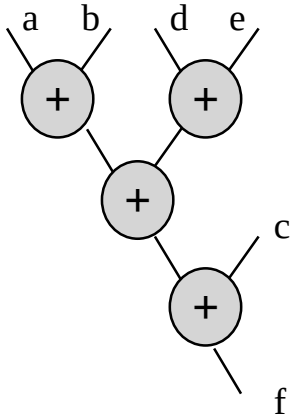
y = a.x
z = x.b

y = a.x
z = b.x





f = (((a + b) + c) + d) + e;



f = (a + b) + (d + e) + c;

Diminution de Tcc

peut diminuer le coût en FUs

++ utilisation des ressources

mais ++composants interconnexions

E. Boutillon

L.E.S.T.E.R
Laboratoire d'Electronique des Systèmes TEmps Réel

Sélection et allocation

- Sélection : extraire de la bibliothèque de composants un jeu de composants.
- Allocation : déterminer le nombre d'opérateurs de chaque type minimum à utiliser dans le circuit.
- Objectif : minimiser le coût global.
- Problèmes : une même fonction (addition par exemple) peut avoir plusieurs instances avec différent compromis vitesse-surface-consommation

Ordonnancement

- Affectation d'une date d'exécution à chaque opération
- Deux types
  - ⇨ Temps contraint ou Ressource contrainte
  - ⇨ Précède l'allocation Suit l'allocation
- Techniques
  - ⇨ Constructives (transformations, glouton, left edge);
  - ⇨ Polynomiales ou Linéaires (listes, orientés par les forces, ASAP, ALAP);
  - ⇨ Exponentielles (branch & bound, ILP);
  - ⇨ Stochastiques (Monte-Carlo, recuit simulé)
  - ⇨ Auto- organisation avec réseaux de neurones

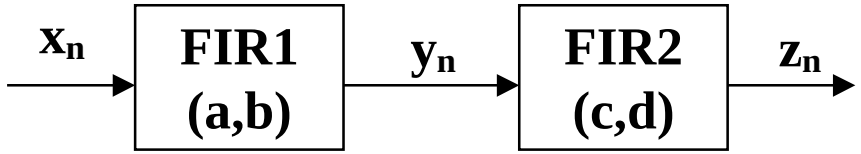
Outils de synthèse logique

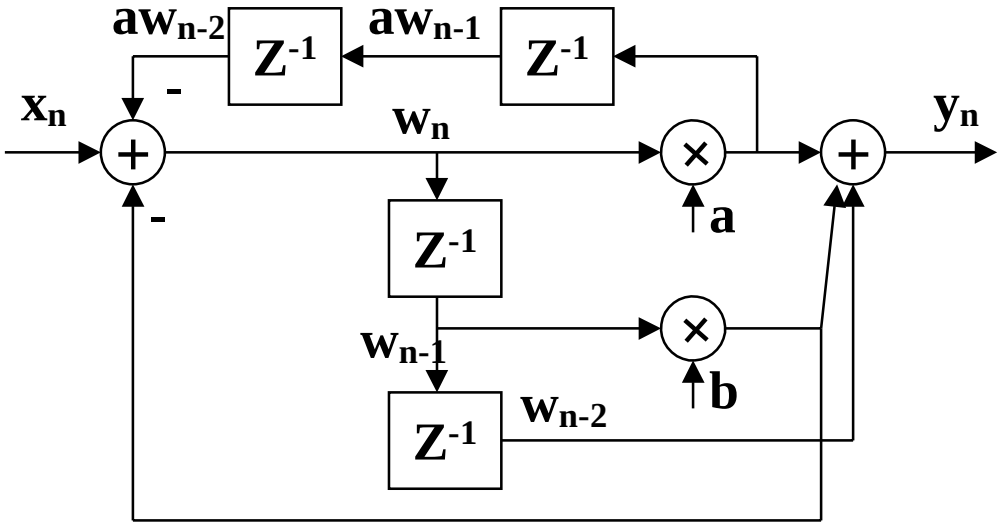
- Outils de recherche et techniques sont mûres
  - ⇨ Cathedral, Hyper, Amical, Phideo, Mimola, Caddy, Gaut...
- Outils commerciaux disponibles
  - ⇨ Monet (Mentor-Graphics)
  - ⇨ Behavioral Compiler (Synopsys) *CoCentric*
  - ⇨ Synthesia, Visual Architect (Cadence)
  - ⇨ DASYS (Summit Design http://sd.com)

GAUT : http://lester.univ-ubs.fr:8080
HYPER : http://infopad.eecs.berkeley.edu/~hyper
CATHEDRAL : http://www.imec.be
AMICAL : http://tima-cmp.imag.fr/tima/sls/amical/amical.html

Exemple de synthèse d'architecture

Application : cascade de deux FIR "passe-tout" d'ordre 2.





Equations :

$w_n = x_n - bw_{n-1} - aw_{n-2}$

$y_n = aw_n + bw_{n-1} + w_{n-2}$

$u_n = y_n - du_{n-1} - cu_{n-2}$

$z_n = cu_n + du_{n-1} + u_{n-2}$

4 multiplications et 4 additions par échantillon d'entrée

E. Boutillon

L.E.S.T.E.R
Laboratoire d'Electronique des Systèmes TEmps Réel

Solution « RTL » avec 4 cycles d ’horloge par échantillon

Détermination du matériel minimal

- **1 multiplieur**
- **2 additionneurs/soustracteurs**

Détermination du séquencement

cycle 0 : $bw_{n-1} = b \times w_{n-1}$ $\quad u_{n-1} = x + d \times w_{n-2} + c \times w_{n-3}$

cycle 1 : $cu_{n-1} = d \times u_{n-1}$ $\quad w_n = x + b \times w_{n-1} + a \times w_{n-2}$

cycle 2 : $aw_n = a \times w_n$ $\quad z_{n-1} = c\, u_{n-1} + d \times u_{n-2} + u_{n-3}$

cycle 3 : $dw_{n-1} = d \times u_{n-1}$ $\quad y_n = a\, w_{n-1} + b \times w_{n-1} + w_{n-2}$

Détermination du nombre minimal de registres

- **durée de vie des variables**

Ecriture du code VHDL niveau RTL + Synthèse.

=> Lent, possibilité de faire des erreurs.

En VHDL pour Monet v44 (mentorgraphics)

```
main : process
variable w0,w1,w2,u0,u1,u2,x0,y0,z0:integer;
begin
  primary : loop wait until port_clk_in='1';
     exit primary when port_reset_in='1';
  x0:= port_x_in;
  w2:= w1; w1:= w0; u2:= u1; u1:= u0; -- Vieillissement
-- Calcul de la sortie du premier puis deuxieme filtre
  w0:= x0 + F(A*w2) - F(B*w1);  - F(X) = troncature de X
  y0:= F(A*w0) + F(B*w1) + w2;
  u0 := y0 + F(C*u2) - F(D*u1);
  z0 := F(C*u0) + F(D*u1) + u2;

for count in 1 to LATENCY-1 loop wait until port_clk_in='1';
exit primary when port_reset_in='1'; end loop;
  port_z_out <= z0;
end loop primary; -- puis instruction du reset
```

Analyse du premier résultat

Contraintes : 4 cycles d'horloge, 20 ns de temps de cycles

- Détecte 6 multiplications à effectuer par cycle symbole

- Si a, b, c et d ne sont pas sur le même nombre de bits
  => 1 multiplieur par constante !

- Instancie 2 multiplieurs, 7 registres.
- Calcul en 3 cycles d'horloge
- Complexité totale de 12 000 portes

Remarque : temps de conception = 2 heures
temps de synthèse comportemental = 3 mn

Autre style d'écriture

```
w0 := x0 + aw2 - bw1;
aw0 := F(A*w0);
y0 := aw0 + bw1 + w2;
bw0 := F(B*w0);
u0 := y0 + cu2 - du1;
cu0 := F(C*u0);
z0 := cu0 + du1 + u2;
du0 := F(D*u0);
```

Nouvelle écriture du code qui injecte une connaissance sur le système.

- Instancie 1 multiplieur, 18 registres.
- Calcul en 4 cycles d'horloge
- Complexité totale de 14 000 portes

Conclusion : Spécification rapide, résultat dépendant du style de spécification
Manque de maturité des outils de synthèse d'architecture

Avenir de la synthèse d'architecture

La synthèse d'architecture est-elle un problème intrinsèquement difficile ?

Si oui : elle permet de concevoir rapidement des parties non critiques

Avenir certain : synthèse d'architecture dédiée à un type de problème

Il s'agit de mettre la connaissance en architecture pour un problème donné dans un outil dédié.

=> généralisation de l'IP

PLAN

1. Introduction
2. Langage de spécification matériel
3. La synthèse d ’architecture
4. La preuve formelle
5. Conclusion

Preuve d'équivalence

Preuve d'équivalence entre deux systèmes A et B

- **Mêmes unités de mémorisation**
- **Logique booléenne différente**

Possibilité de prouver l'équivalence entre les deux circuits.

Utilisation :

- **Vérification de la synthèse/RTL**
- **Vérification résultat optimisation d'une netlist**
- **Vérification de l'insertion d'un arbre d'horloge**
- **Vérification de l'insertion de matériel de test (entrée SCAN mise à zéro).**

Preuve d ’équivalence

Outils disponibles :

Mentorsgraphic : formalpro

http://www.mentorgraphics.com/formalpro/overview.html

Verplex : Tuxedo™ LEC

http://www.verplex.com/

Synopsys : Formality

http://www.synopsys.com/products/verification/verification.html

Maturité des outils : 4 heures pour prouver l’équivalence de deux circuits d’un million de portes.

Preuve formelle

Preuve de comportement :
Equivalence fonctionnelle séquentielle entre deux circuits.

Exemple : prouver l'équivalence (du point de vue entrée/sortie) entre deux machines à états.

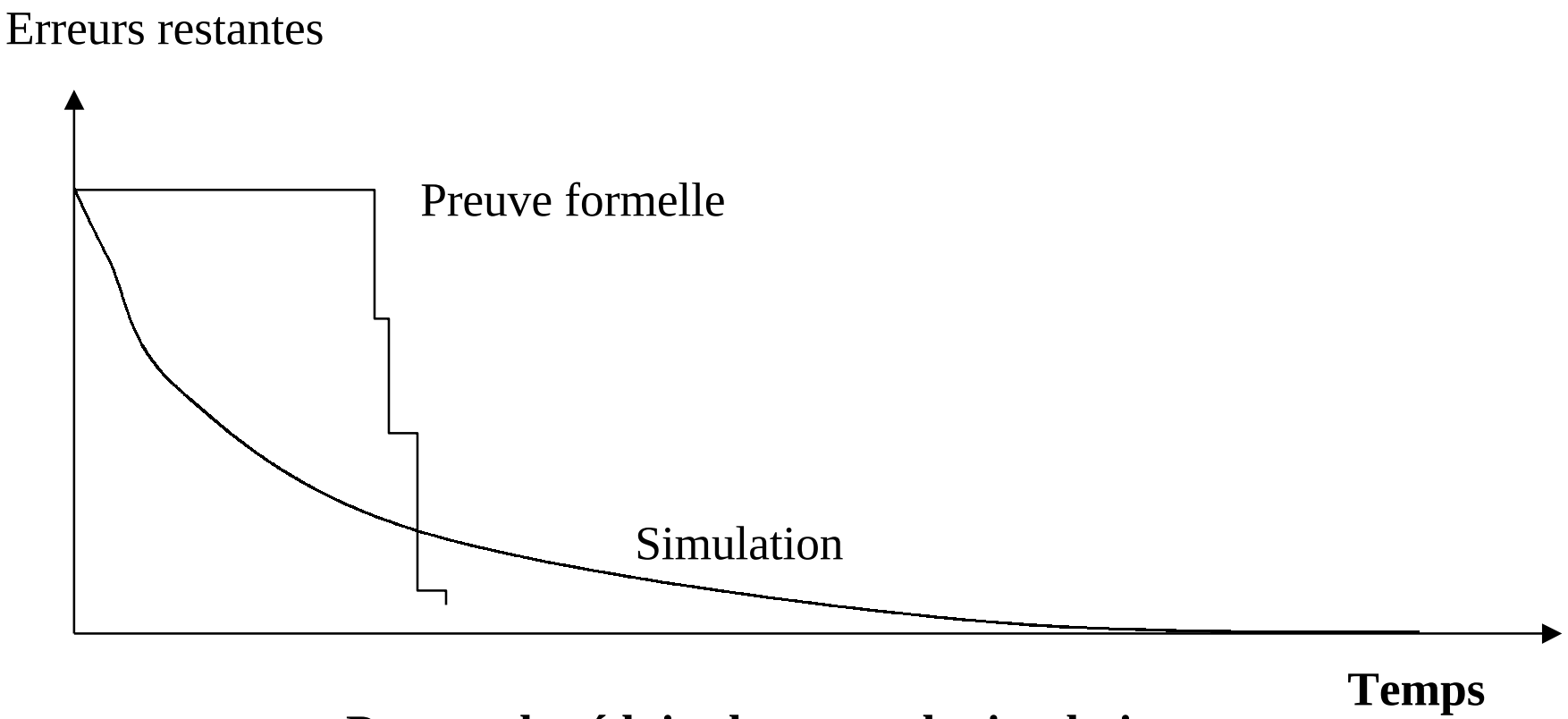
Machine A : 5 états, 3 registres
Machine B : 5 états, 5 registres (codage One Hot)

Prouver une propriété sur un circuit donné

« Il n'y aura jamais les signaux A et B égaux à 1 en même temps »
« si le signal E passe à 1, alors, le signal S passera à 0 »

=> Donne une séquence contre-exemple si la propriété est fausse

Correction des erreurs



Permet de réduire le temps de simulation

Limitation de la preuve formelle

*** Se limite à la vérification des théorèmes testés.**

*** Vérification possible uniquement sur des problèmes de petites tailles (< 100 registres).**

Exemple : contrôleur de SDRAM

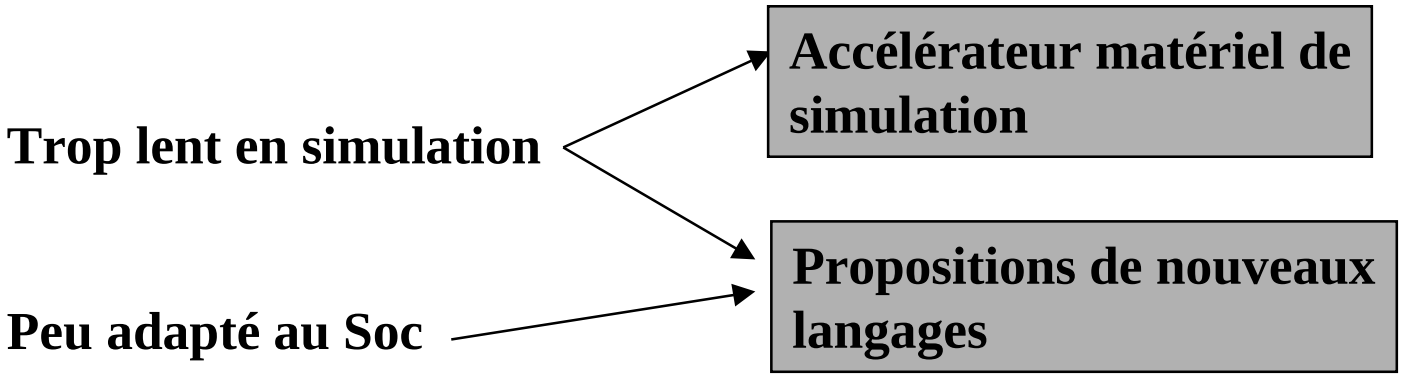
*** Difficulté à être utilisée par les concepteurs de circuits car formalisme mathématique complexe.**

De nombreux travaux en cours...

Avenir de la preuve formelle ?

Conclusion

Avenir des langages de description de matériel



Synthèse d'architecture

- **Pour du "quick and dirty"**
- **Outils dédiés pour une classe d'algorithme (IP)**

Preuve formelle

- **A suivre...**