Conception de circuits numériques et architecture des ordinateurs

Équipe pédagogique :
Marie Badaroux (TD), Julie Dumas (TD+TP), Florence Maraninchi (TD+TP),
Olivier Muller (TD+TP), **Frédéric Pétrot** (CM+TD), Pierre Ravenel (TD+TP)
Eduardo Tomasi Ribeiro (TP), Lionel Rieg (TD+TP),
Sébastien Viardot (TD+TP Apprentis)

Année universitaire 2024-2025

C1	Codage des nombres en base 2, logique booléenne, portes logiques, circuits combinatoires
C2	Circuits séquentiels
C3	Construction de circuits complexes
C4	Micro-architecture et fonctionnement des mémoires
C5	Machines à état
C6	Synthèse de circuits PC/PO
C7	Optimisation de circuits PC/PO
C8	Interprétation d'instructions - 1
C9	Interprétation d'instructions - 2
C10	Interprétation d'instructions - 3
C11	Introduction aux caches

Intérêt

Idem cours précédent, mais en plus rapide et moins cher !

Plan détaillé du cours d'aujourd'hui

Plan

Introduction

Problématique :

Optimisation en temps ou en surface de circuits PC/PO

Toujours :

- simplifier le code et les opérateurs

Optimiser la surface (ce n'est pas notre but) :

- séquentialiser les opérations
- avoir un nombre minimum d'opérateurs, de registres, de multiplexeurs, ...

Optimiser le temps (l'objectif du cours) :

- effectuer plus d'opérations en parallèle
  en respectant toujours les dépendances de données
- limiter le nombre d'états lié à un parcours de l'automate

 non directement lié au nombre total d'états

Optimisations « usuelles »

Existe dans les compilateurs de logiciel :

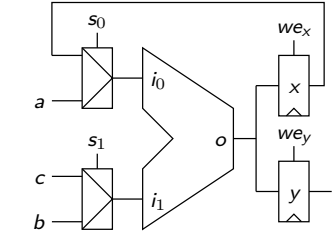
- élimination et propagation de constantes :
  $x := 3; y := x$ donne $y := 3;$
- simplification des opérateurs :
  $y := x/4 \Rightarrow y := x >> 2;$
  $y := x \times 32 \Rightarrow y := x << 5;$
  $y := x \times 15 \Rightarrow y := (x << 4) - x;$
- identification des sous-expressions communes :
  $a := b \times c + g; d := b \times c \times d;$ donne
  $t := b \times c; a := t + g; d := t \times d;$
- ...

Calculs en parallèle : Principe

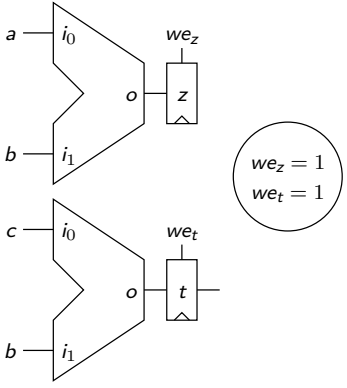
- identifier les dépendances de données
  Exemple 1 : $x := a + b; y := x + c$, attendre 1 cycle que x soit écrit pour calculer y
  Exemple 2 : $z := a + b; t := c + d$, z et t peuvent être écrits dans le même cycle
- déterminer le nombre minimal d'opérateurs pour le parallélisme souhaité :
  Exemple 1 : partage du $+$, 2 cycles
  Exemple 2 : partage du $+$, 2 cycles, ou bien calcul z et t sur 2 $+$ différents, 1 cycle

Calculs en parallèle : implantation

Activation simultanée des we_i
Exemple 1 :



Exemple 2 :



Calculs en parallèle : Exemple

Cordic, again !

```
1  X := x; Y := y; Z := z;
2  for i := 0 to 29 do
3  |   a := X/2^i; b := Y/2^i; c := t[i];
4  |   if Z > 0 then
5  |   |   X := X − b;
6  |   |   Y := Y + a;
7  |   |   Z := Z − c;
8  |   else
9  |   |   X := X + b;
10 |   |   Y := Y − a;
11 |   |   Z := Z + c;
12 |   end if
13 end for
```

- ligne 3 déplaçable derrière 4 et 8
- lignes 5, 6 et 7 peuvent être faites simultanément
  2 soustracteurs, 1 additionneur
- lignes 9, 10 et 11 peuvent être faites simultanément 2 additionneurs, 1 soustracteur
- groupes exclusifs : 3 add/sub

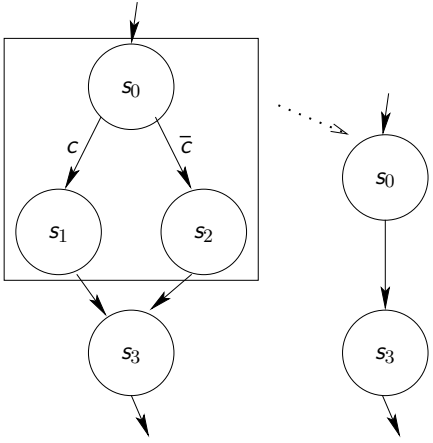
Calculs en parallèle : Exemple

Ré-écriture du code
Notation écriture concurrente : $(p, q, ..) := (f(x, y, ...), g(u, v, ...), ...)$
Ex : $(p, q) := (q, p)$ échange p et q

1 $(X, Y, Z) := (x, y, z)$;
2 **for** $i := 0$ **to** 29 **do**
3 | **if** $Z > 0$ **then**
4 | | $(X, Y, Z) := (X - \frac{Y}{2^i}, Y + \frac{X}{2^i}, Z - t[i])$;
5 | **else**
6 | | $(X, Y, Z) := (X + \frac{Y}{2^i}, Y - \frac{X}{2^i}, Z + t[i])$;
7 | **end if**
8 **end for**

Câblage des conditions : principes

Regrouper plusieurs états en 1 seul :



Pourquoi :

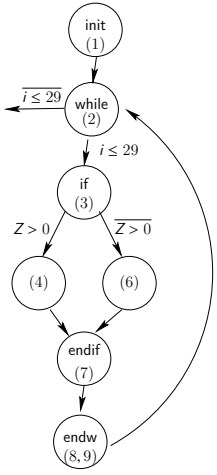
- gagne des cycles
- simplifie l'automate

Comment :

- en commandant les s_i et we_j directement dans la PO

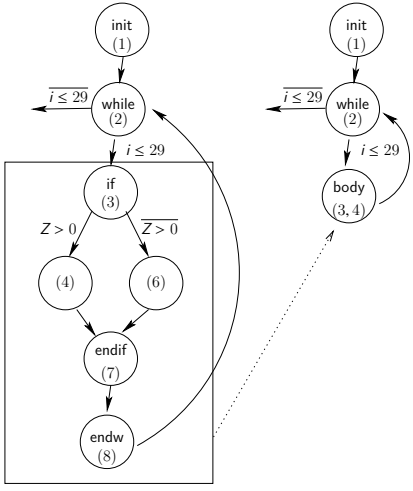
Câblage des conditions : Exemple

1 $(X, Y, Z, i) := (x, y, z, 0)$;
2 **while** $i \leq 29$ **do**
3 | **if** $Z > 0$ **then**
4 | | $(X, Y, Z) := (X - \frac{Y}{2^i}, Y + \frac{X}{2^i}, Z - t[i])$;
5 | **else**
6 | | $(X, Y, Z) := (X + \frac{Y}{2^i}, Y - \frac{X}{2^i}, Z + t[i])$;
7 | **end if**
8 | $i := i + 1$;
9 **end while**

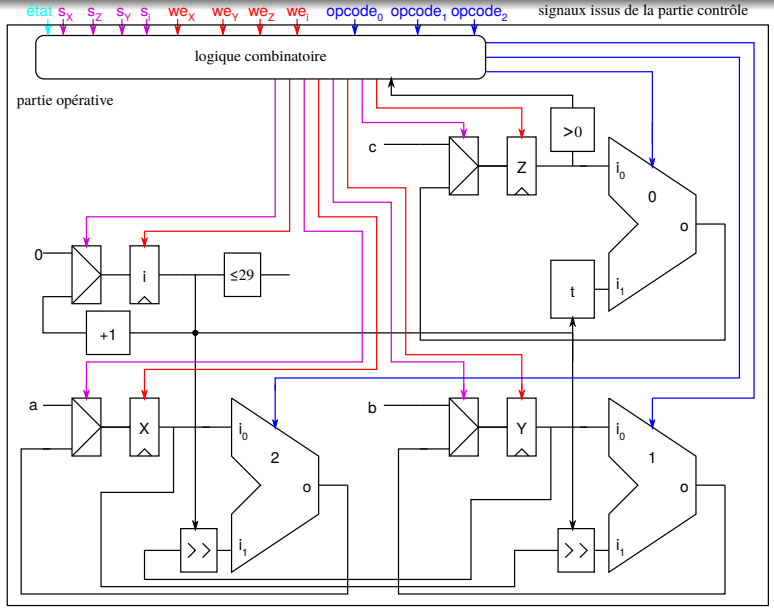


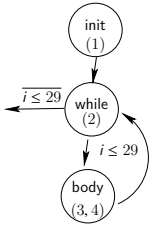
Câblage des conditions : Exemple

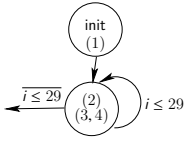
1 $(X, Y, Z, i) := (x, y, z, 0)$;
2 **while** $i \leq 29$ **do**
3 $\quad (X, Y, Z, i) := Z > 0 ? (X - \frac{Y}{2^i}, Y + \frac{X}{2^i}, Z - t[i], i + 1)$
4 $\quad\quad\quad\quad\quad\quad\quad\quad : (X + \frac{Y}{2^i}, Y - \frac{X}{2^i}, Z + t[i], i + 1)$;
5 **end while**



Câblage des conditions : partie opérative







Câblage des conditions : analyse

PC :

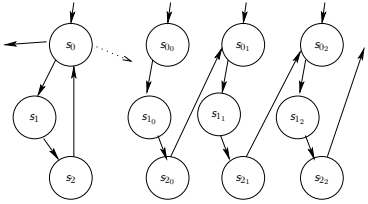
- gain en nombre d'états : $4 = 7 - 3$ états
- gain sur le nombre d'états par itération : de 5 à 2 états
- simplification de Θ et Γ : chemin critique dans PC plus court

PO :

- selecteurs et autorisation d'écriture :
  calculés comme une fonction de la condition cablée et des signaux issus de la PC
- plus de logique et d'opérateurs arithmétique
- chemin critique dans PO potentiellement plus long

Câblage des boucles : principes

Dérouler une boucle n fois sur le silicium



Pourquoi :

- suprime ou simplifie l'automate
- met en évident du parallèlisme entre itérations
- choix performance/coût silicium

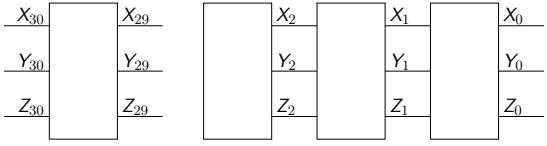
Comment :

- nécéssite le câblage des conditions
- nécéssite la réplication du matériel
- le déroulement peut être partiel

Câblage des boucles : Exemple

X_i, Y_i, Z_i : signaux intermédiaires
t_i : constantes indépendantes

$(X, Y, Z) := (a, b, c)$;
$(X_1, Y_1, Z_1) := Z > 0?(X - Y, Y + X, Z - t_0) : (X + Y, Y - X, Z + t_0)$;
$(X_2, Y_2, Z_2) := Z_1 > 0?(X_1 - Y_1 >> 1, Y_1 + X_1 >> 1, Z_1 - t_1) : (X_1 + Y_1 >> 1, Y_1 - X_1 >> 1, Z_1 + t_1)$;
$(X_3, Y_3, Z_3) := Z_2 > 0?(X_2 - Y_2 >> 2, Y_2 + X_2 >> 2, Z_2 - t_2) : (X_2 + Y_2 >> 2, Y_2 - X_2 >> 2, Z_2 + t_2)$;
$\ldots$;
$(X_{29}, Y_{29}, Z_{29}) := Z_{28} > 0?(X_{28} - Y_{28} >> 28, Y_{28} + X_{28} >> 28, Z_{28} - t_{28}) : (X_{28} + Y_{28} >> 28, Y_{28} - X_{28} >> 28, Z_{28} + t_{28})$;
$(X_{30}, Y_{30}, Z_{30}) := Z_{29} > 0?(X_{29} - Y_{29} >> 29, Y_{29} + X_{29} >> 29, Z_{29} - t_{29}) : (X_{29} + Y_{29} >> 29, Y_{29} - X_{29} >> 29, Z_{29} + t_{29})$;



Câblage des boucles : analyse

PO :

- optimisations possibles car variable de boucle devient constante
  décalage ≡ choix de fils
  multiplication par constante plus simple, *cf* optim compilo
- potentiellement :
  beaucoup plus d'opérateurs
  chemin critique plus long

En pratique : on ne déroule pas les boucles, on les pipeline ! Voir l'excellent cours de 2A !