Arithmétique binaire

M. Combacau
combacau@laas.fr

Université Paul Sabatier
LAAS-CNRS

November 10, 2024

Objectif

Comprendre comment sont effectués les calculs dans un processeur informatique

Introduction

Arithmétique

Étude des propriétés de l'ensemble des nombres rationnels ($\mathbb{Q}$)

- Définition légèrement modifiée dans ce cours
- $\rightarrow$ les opérations de calcul de base (+,-,x,/)
- Basé sur la logique combinatoire
- Réalisation complète pour les opérations arithmétiques combinatoires
- Principes et algorithmes associés pour les opérations non combinatoires (vues en S6 en DES)

Addition binaire de deux bits (1)

Comme pour les nombres en base 10, l'addition de deux mots est réalisée en se basant sur l'addition de deux digits (bits ici)

- soit a et b deux variables booléennes interprétées comme la valeur $a_0 \times 2^0$ (abus de langage)
- la somme de ces deux valeurs donne 0, 1 ou 2
- le résultat est codé sur deux bits $r_1 s_0$
- Table de vérité

a_0	b_0	r_1	s_0
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

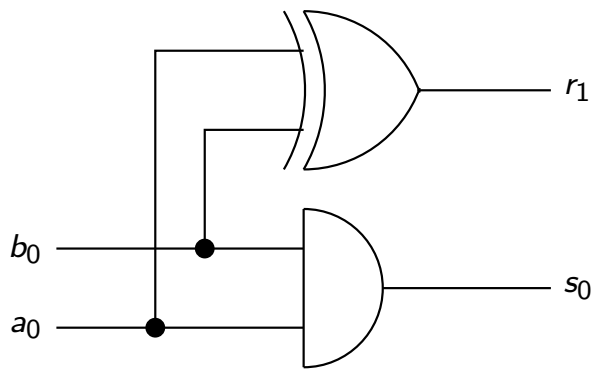
on reconnaît immédiatement :

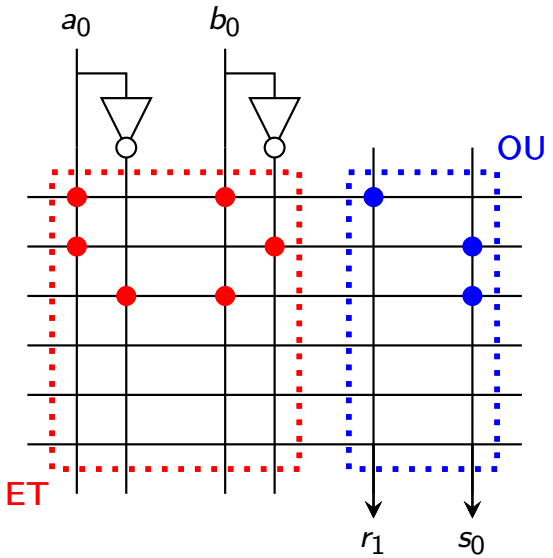
$$\begin{cases} s_0 = \overline{a_0}.b_0 + a_0.\overline{b_0} = a_0 \oplus b_0 \\ r_1 = a_0.b_0 \end{cases}$$

Addition binaire de deux bits (2)

$$\begin{cases} s_0 &= \overline{a_0}.b_0 + a_0.\overline{b_0} = a_0 \oplus b_0 \\ r_1 &= a_0.b_0 \end{cases}$$

D'où les logigrammes qui permettraient sa réalisation





Cette réalisation est dite "demi-additionneur"

Addition binaire de deux bits (3)

- Addition n bits, $\Rightarrow$ propager retenue bits i vers bits $i+1$.
- Additionneur complet $\Rightarrow$ 3 bits en entrée $(r_i,\ a_i,\ b_i)$
- Table de vérité

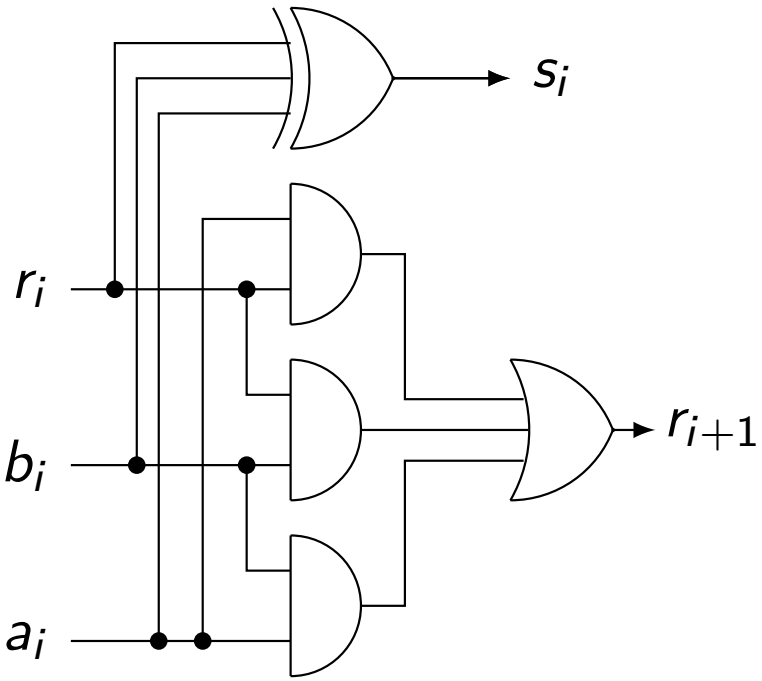
r_i	a_i	b_i	r_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

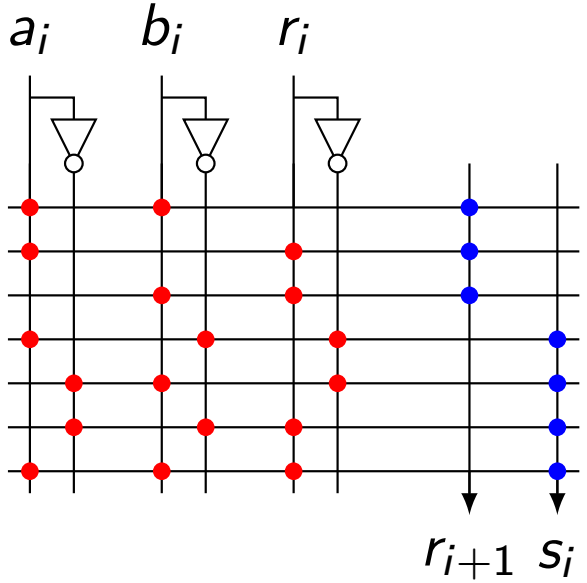
On trouve (méthode au choix)

$$\begin{cases} s_i & = & a_i \oplus b_i \oplus r_i \\ r_{i+1} & = & r_i.a_i + r_i.b_i + a_i.b_i \end{cases}$$

Addition binaire de deux bits (4)

D'où une réalisation par portes et réseaux logiques

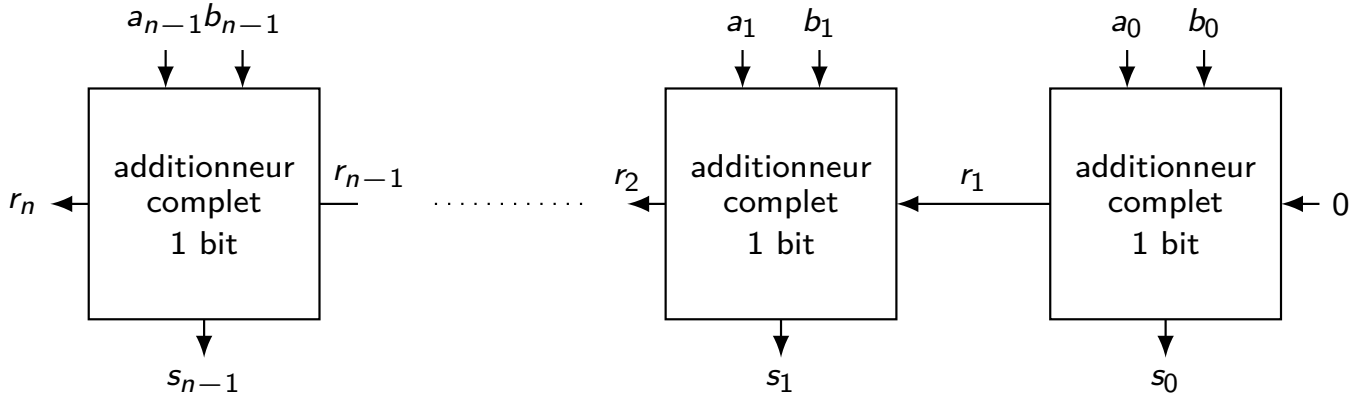




Mise en œuvre de l'additionneur complet

Addition binaire de n bits (1)

Elle repose sur la mise en parallèle de n additionneurs complets



Additionneur conçu pour les entiers positifs

Addition de deux nombres négatifs en ca2

$A < 0$ et $B < 0$, codés respectivement par $2^n - |A|$ et $2^n - |B|$.

$$\left\| \begin{array}{rl} S & = 2^n - |A| + 2^n - |B| \\ & = 2^n + (2^n - |A + B|) \end{array} \right.$$

$\Rightarrow$ code de $-A - B$ avec $r_n = 1$

Addition binaire de n bits (2)

Addition de $A \geq 0$ et $B < 0$ en ca2

$$\left\| \begin{array}{ll} S & = A + 2^n - |B| \\ & = 2^n + (A - |B|) \end{array} \right.$$

$\Rightarrow$ code de $A - |B|$ avec r_n

1. $|A| \geq |B|$, $S = A - B \geq 0$ retenue $r_n = 1$
2. $|A| < |B|$, $S = |A| - |B| < 0$ retenue $r_n = 0$

$\Rightarrow$ soustracteur $A - B$ $(B > 0)$ si on remplace B par ca2(B)

Exemple

2	0010	(code de 2)
-3	1101	(code de 16-3=13)
S	1111	(code de -1)

Temps de calcul (1)

Un opérateur logique $\rightarrow$ temps de propagation $= \delta_t$ (approximation)

- Calcul de $r_1 = 2 \times \delta_t$, calcul de $s_0 = \delta_t$
- Calcul de $r_2 = (2+2) \times \delta_t$, calcul de $s_1 = 2 \times \delta_t + \delta_t$
- ...
- Calcul de $r_i = (i \times 2) \times \delta_t$, calcul de $s_{i-1} = i \times \delta_t + \delta_t = (2i+1)\delta_t$

Propagation de retenue $\Rightarrow$ dépendance linéaire de la durée à n
Durée de l'addition $\Rightarrow$ limite la performance du calculateur !
Exemple sur 64 bits : si $\delta_t = 2ns \Rightarrow s_{63}$ calculé en $127ns$

Solution : calcul en parallèle de toutes les retenues (retenue anticipée)

Temps de calcul (2)

Retenue anticipée (pour r_2)

On sait que

$$\begin{cases} r_1 &= a_0.b_0 + a_0.r_0 + b_0.r_0 \\ r_2 &= a_1.b_1 + a_1.r_1 + b_1.r_1 \end{cases}$$

D'où

$$\begin{aligned} r_2 &= a_1.b_1 + a_1.(a_0.b_0 + a_0.r_0 + b_0.r_0) + b_1.(a_0.b_0 + a_0.r_0 + b_0.r_0) \\ &= a_1.b_1 + a_1.a_0.b_0 + a_1.a_0.r_0 + a_1.b_0.r_0 + b_1.a_0.b_0 + b_1.a_0.r_0 + b_1.b_0.r_0 \end{aligned}$$

Cette expression de r_2 peut être réalisée en "deux couches" (ET suivi de OU) sur réseau logique programmable (calcul en $2 \times \delta_t$)

De même

$$\begin{aligned} s_1 &= a_1 \oplus b_1 \oplus r_1 \\ &= \overline{a_1}.\overline{b_1}.r_1 + \overline{a_1}.b_1.\overline{r_1} + a_1.\overline{b_1}.\overline{r_1} + a_1.b_1.r_1 \\ &= \overline{a_1}.\overline{b_1}.(a_0.b_0 + a_0.r_0 + b_0.r_0) + \overline{a_1}.b_1.\overline{(a_0.b_0 + a_0.r_0 + b_0.r_0)} \\ &\quad + a_1.\overline{b_1}.\overline{(a_0.b_0 + a_0.r_0 + b_0.r_0)} + a_1.b_1.(a_0.b_0 + a_0.r_0 + b_0.r_0) \end{aligned}$$

Temps de calcul (3)

Retenue anticipée (pour r_2)

$$
\begin{aligned}
s_1 &= \overline{a_1}.\overline{b_1}.(a_0.b_0 + a_0.r_0 + b_0.r_0) \\
&\quad + \overline{a_1}.b_1.\overline{(a_0.b_0 + a_0.r_0 + b_0.r_0)} \\
&\quad + a_1.\overline{b_1}.\overline{(a_0.b_0 + a_0.r_0 + b_0.r_0)} \\
&\quad + a_1.b_1.(a_0.b_0 + a_0.r_0 + b_0.r_0) \\
&= \overline{a_1}.\overline{b_1}.a_0.b_0 + \overline{a_1}.\overline{b_1}.a_0.r_0 + \overline{a_1}.\overline{b_1}.b_0.r_0 \\
&\quad + \overline{a_1}.b_1.\overline{a_0}.\overline{b_0} + \overline{a_1}.b_1.\overline{a_0}.\overline{r_0} + \overline{a_1}.b_1.\overline{b_0}.\overline{r_0} \\
&\quad + a_1.\overline{b_1}.\overline{a_0}.\overline{b_0} + a_1.\overline{b_1}.\overline{a_0}.\overline{r_0} + a_1.\overline{b_1}.\overline{b_0}.\overline{r_0} \\
&\quad + a_1.b_1.a_0.b_0 + a_1.b_1.a_0.r_0 + a_1.b_1.b_0.r_0
\end{aligned}
$$

Réalisable sur réseau logique programmable

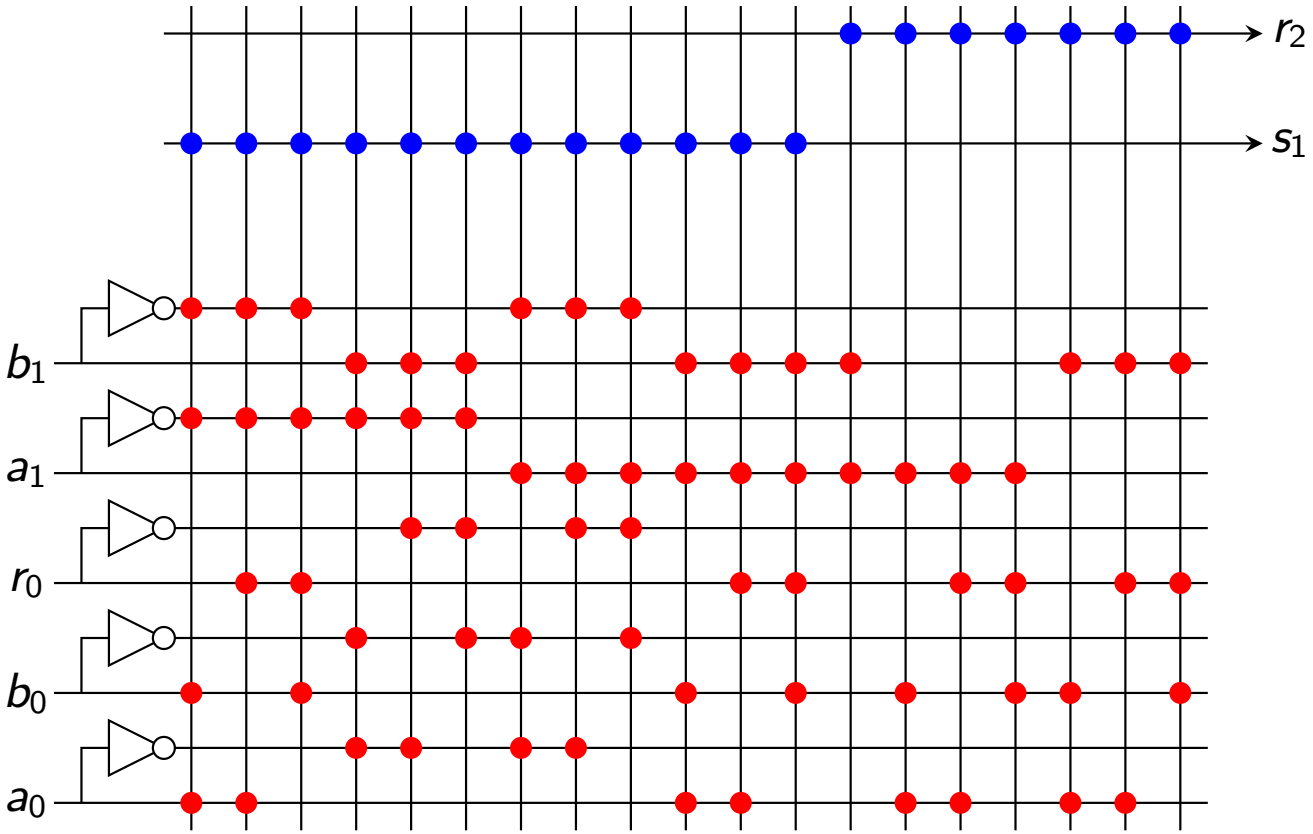
temps de reponse $3 \times \delta_t$ ($NON \rightarrow ET \rightarrow OU$)

Quelle que soit l'expression, il est toujours possible de revenir à une forme somme de produits

temps de réponse $3 \times \delta_t$ indépendant de i

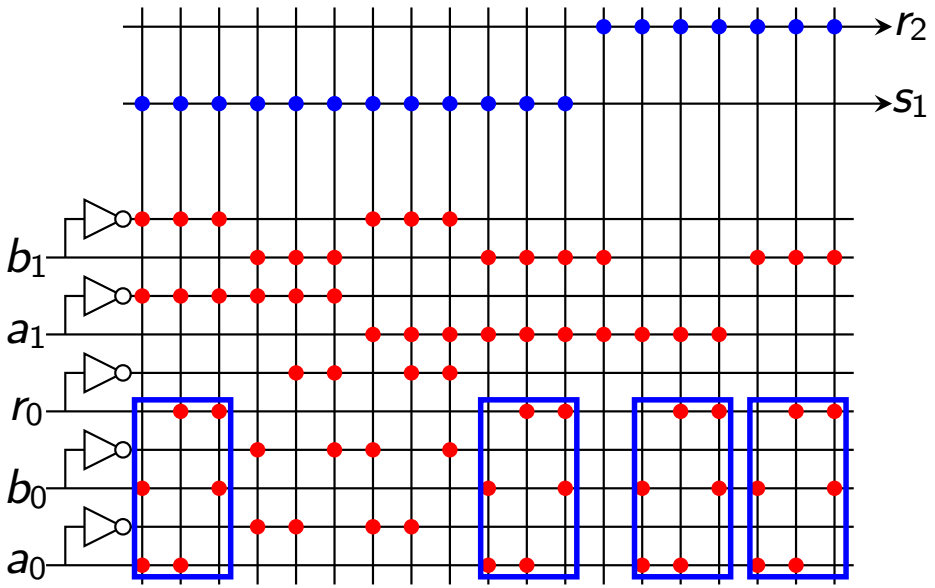
Temps de calcul (4)

Réalisation de s_2 et r_2



Quelques conclusions

- Retenue anticipée plus rapide $\leftrightarrow$ circuit plus complexe
- Calculs identiques faits plusieurs fois en parallèle



- Le calcul de la retenue r_2 n'a pas à être câblé
- Complexité tout à fait abordable pour des logiciels

Quelques conclusions (2)

- Retenue r_n à câbler !
- C'est un indicateur de **débordement** de capacité
- Additionneur n bits mis en œuvre dans tous les processeurs

Cet additionneur va servir pour mettre en œuvre la soustraction $A - B$ d'une manière très simple.
Il suffit de complémenter l'entrée B de l'additionneur et d'ajouter $+1$ au résultat (r_0) pour obtenir la soustraction $A - B$.
Nous y reviendrons lors de la construction d'une UAL (Unité Arithmétique et Logique)