

Fonction "FA" en CMOS

Technologie CMOS En technologie CMOS (*Métal Oxyde Semiconducteur complémentaire*) on utilise des transistors canal N et des transistors canal P pour réaliser des fonctions logiques. La technologie CMOS est actuellement la technologie dominante du marché. Son principal intérêt par rapport à d'autres technologies comme le NMOS ou le bipolaire est une consommation d'énergie remarquablement faible. En fait les circuits CMOS ont un courant statique (quand ils sont au repos) pratiquement négligeable.

Dans les figures ci-dessous :

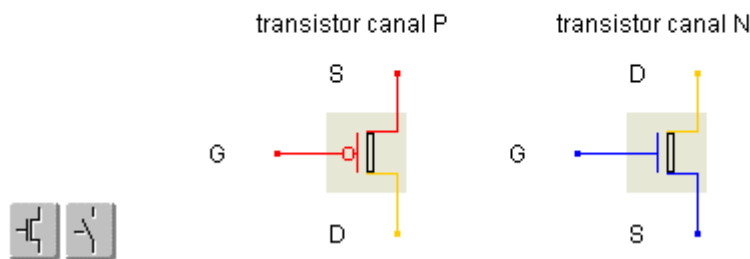
Un '1' logique est représenté électriquement par la tension d'alimentation Vdd (des valeurs courantes de Vdd sont +5V ou +3,3V ou +2,8V) et est coloré en **rouge**.



Un '0' logique correspond à la tension de masse ou GND est coloré en **bleu**.

Une connexion non reliée est colorée en **jaune**.

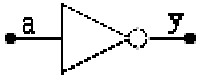
Applet de transistors CMOS:

Cliquez la grille G ou la source S des transistors pour changer leur état



Remarquez que le transistor canal N conduit quand sa grille est au '1' logique, et que le transistor canal P conduit quand sa grille est au '0' logique. Les touches   permettent de changer le dessin des transistors.

L'inverseur CMOS

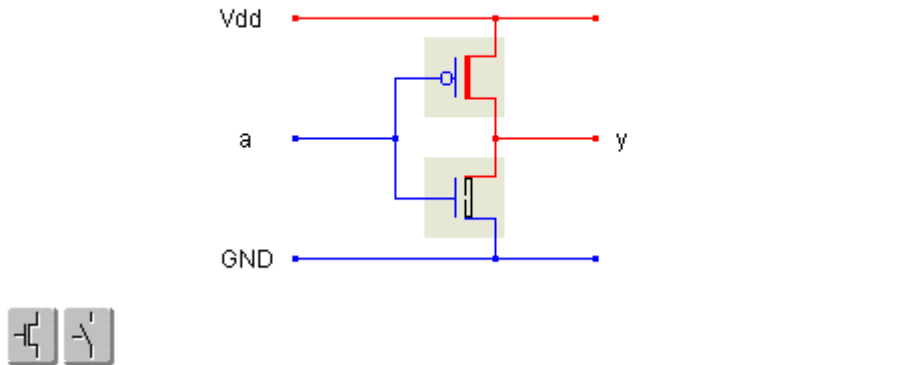


L'inverseur est la porte CMOS la plus fréquente. Il est formé d'un transistor canal N et un transistor canal P, reliés par leurs drains. La figure ci-dessous en illustre le fonctionnement.

Les couleurs sont toujours le rouge pour '1' logique et le bleu pour '0' logique. Une tension d'entrée entre les deux cause un court-circuit en maintenant les deux transistors en conduction. Une telle tension est colorée en **vert**. Cliquer sur l'entrée "a" pour la faire passer de '0' à court-circuit (**vert**), puis à '1', puis de nouveau à '0'.

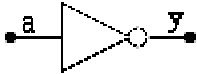
Applet de l'inverseur CMOS: $y = \bar{a}$

Cliquez sur l'entrée a pour changer sa tension



Remarquez que si l'entrée vaut '0' ou '1', un seul transistor conduit.

Délai et consommation de l'inverseur CMOS



Nous venons de voir que l'inverseur n'a pas de dissipation d'énergie sauf lorsqu'il commute. En effet si l'entrée vaut '0' ou '1' il n'y a pas de chemin de conduction entre l'alimentation Vdd et la masse GND. Dans les conditions normales d'utilisation, le courant de court-circuit (inévitables lorsque l'entrée commute) dure très peu de temps, typiquement quelques picosecondes.

La contribution du courant de charge ou décharge des capacités à la consommation est beaucoup plus importante. Les grilles G des transistors forment des capacités. Ces capacités sont d'ailleurs nécessaires au fonctionnement du transistor à effet de champs. Typiquement la capacité d'entrée C_g vaut environ 10 fF. Si l'entrée a de l'inverseur est à reliée à Vdd au temps t_1 , cette capacité est chargée (charge $Q = C_g * V_{dd}$). Si par la suite l'entrée est reliée à GND au temps t_2 la capacité se décharge. Cette décharge produit un courant dans la grille de valeur $I = dQ/dt = (C_g * V_{dd}) / (t_2 - t_1)$.

Bien que le courant de charge/décharge de grille soit faible, le courant total consommé par un circuit intégré complexe peut être important. Prenons un exemple :

- Un microprocesseur moderne peut contenir 50 millions de transistors, soit environ dix millions de portes. A chaque cycle environ 1% de ces portes commutent.
- Les fréquences d'horloge atteignent 500 MHz (temps de cycle 2 ns) avec une tension d'alimentation $V_{dd} = 3.3V$.
- Les fils connectant les portes ont une capacité parasite C_w bien plus grande que la capacité de grille C_g des portes. Chaque fois qu'une équipotentielle commute, toutes les capacités qui lui sont attachées doivent être chargées ou déchargées : $C_{total} = C_g + C_w$.
- La capacité d'un fil d'interconnexion atteint typiquement 1 pF

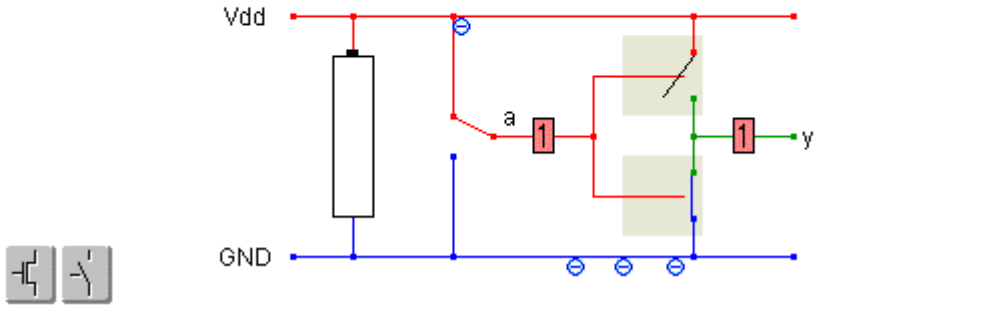
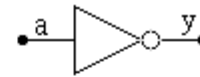
Il est assez difficile d'estimer le courant dû aux courts-circuits, il est en général faible. En revanche le courant résultant de l'activité de commutation est important : $I = (\text{portes actives}) * (C_{total} * V_{dd}) / dt = (1\% * 1.000.000) * (1pF * 3.3V) / 2ns = 16 A$

Enfin le courant de repos dû aux fuites des transistors (pour une circuiterie conventionnelle) est très faible. Une mémoire statique SRAM de 2K*8 bits en CMOS laisse fuir μA au repos.

La figure ci-dessous illustre le courant, ou déplacement d'électrons e^- de l'inverseur CMOS . Si la tension d'entrée demeure à '1' ou à '0', soit le transistor canal P ou le transistor canal N est bloqué et il n'y a pas de courant.

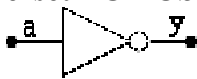
- Si l'entrée a commute, les grilles des deux transistors doivent être chargées ou déchargées. Ceci est illustré par le passage d'un électron e^- (de charge négative) venant de GND ou bien allant à Vss..
- Quand l'entrée commute, elle passe par des tensions faisant conduire les deux transistors pendant un temps très court. Le courant de court circuit résultant est illustré par le passage d'un électron de GND à Vss.
- Enfin la sortie est chargée ou déchargée à travers les transistors. La capacité qui y est attachée stocke deux électrons.

Applet du délai et de la consommation de l'inverseur
 Cliquer sur l'entrée a pour changer sa tension

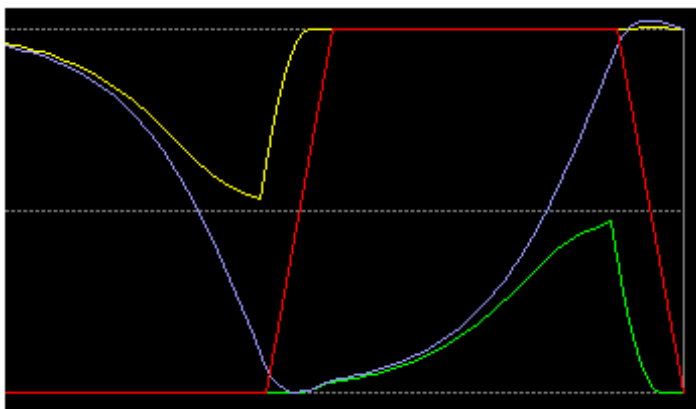


La puissance dissipée par un circuit en logique CMOS conventionnelle est en conséquence directement proportionnelle à la fréquence des commutations, qui est la fréquence de l'horloge.

Simulation électrique de l'inverseur CMOS



Quand vous cliquez dans le chronogramme ci-dessous, vous tracez la tension de l'entrée "a" de l'inverseur (tracé en **rouge** sur le chronogramme). La tension de la sortie "y" est alors calculée (tracée en **bleu**). Le courant traversant le transistor canal N est dessiné en **vert** et celui du transistor canal P en **jaune**. Pour stopper l'applet et figer le dessin, sortez le pointeur de la figure.



Applet de simulation électrique de l'inverseur.

Cliquer ici pour démarrer la simulation.

Sortir le pointeur de la figure pour arrêter la simulation.

Cliquer dans la figure pour tracer la tension de l'entrée "a" (en r

- tension de la sortie "y".
- courant du transistor canal N.
- courant du transistor canal P.

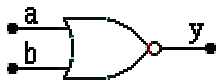
Portes de base NOR et NAND

Nous allons étudier maintenant les portes logiques de base en CMOS: successivement un NOR et un NAND à 2 entrées puis un NAND à 3 entrées. Comme pour toutes les portes CMOS, chaque entrée est connectée à la grille d'un transistor canal N et à la grille d'un transistor canal P.

- **Conventions de couleurs:** Ce sont celles de l'Inverseur. Les connexions à Vdd ('1' logique) sont en rouge, les connexions à GND ('0' logique) sont en bleu, les connexions simultanément à Vdd et GND sont dessinés en **vert**. Enfin les connexions ni à Vdd ni à GND (flottantes) sont en **jaune**. Ces dernières couleurs n'ont pas d'image logique.
- Cliquer près d'une entrée pour changer sa valeur, notez le changement de l'état des transistors.
- La ligne de la table de vérité correspondant à cette combinaison de valeurs d'entrées est indiquée en blanc.

- En cliquant dans la table de vérité, on positionne les entrées aux valeurs de la ligne.
 - Cliquer sur le haut de la table reconstruit progressivement cette table.
- Pour simplifier les applets, seuls des '1' et '0' logiques sont permis en entrée. Il n'est donc pas possible d'entrer des tensions provoquant un court-circuit entre Vdd et GND.

La porte NOR à 2 entrées



La porte CMOS à 2 entrées est l'une des portes les plus simples pour illustrer le qualificatif *complémentaire*: les transistors canal P sont connectés en série alors que les transistors canal N sont connectés en parallèle. Les réseaux de transistors canal N et de transistors canal P sont complémentaires.

Remarquez que si aucun des deux transistors canal P en série ne conduit, leur connexion commune est flottante (**jaune**). Cette valeur non logique ne pose toutefois pas de problème de fonctionnement logique car cette connexion n'est reliée à aucune grille de transistor.

Applet de porte NOR à 2 entrées $y = \overline{a \vee b}$
 Cliquez sur les entrées a ou b pour changer leurs tensions

a	b	y
0	0	1
0	1	0
1	0	0
1	1	0

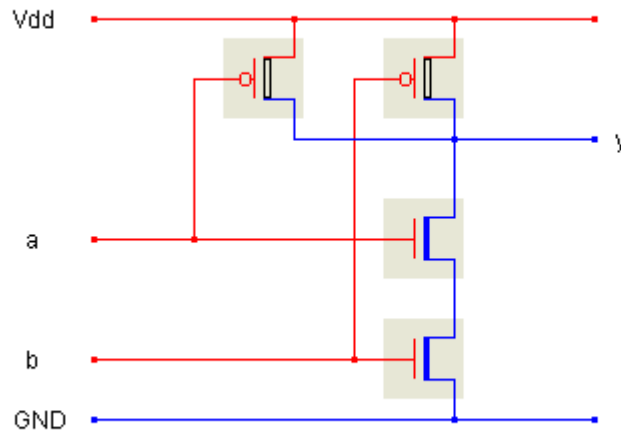
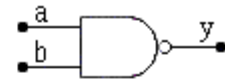
La porte NAND à 2 entrées



Dans le NAND à 2 entrées, les transistors canal P sont reliés en parallèle alors que les transistors canal N sont reliés en série.

Applet de porte NAND à 2 entrées $y = \overline{a \wedge b}$

Cliquer sur les entrées a ou b pour changer leurs tensions

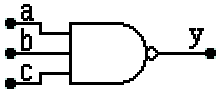


a	b	y
0	0	1
0	1	1
1	0	1
1	1	0



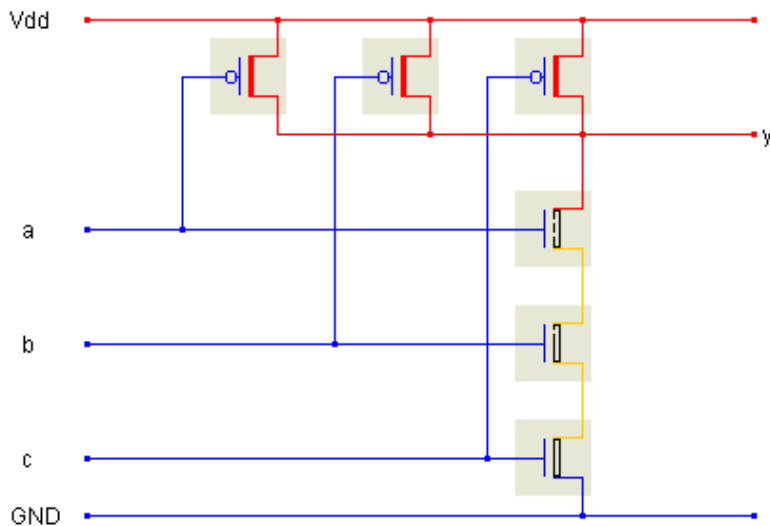
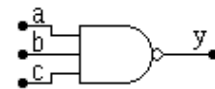
La porte NAND à 3 entrées

La généralisation du schéma d'une porte NOR ou d'une porte NAND à plus de 2 entrées est immédiate. La figure ci-dessous décrit une porte NAND à 3 entrées. Les 3 transistors canal P sont reliés en parallèle et les trois transistors canal N sont reliés en série.



Applet de porte NAND à 3 entrées: $y = \overline{a \wedge b \wedge c}$

Cliquez les entrées a, b ou c pour changer leur tension



a	b	c	y
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

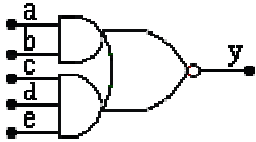


Des portes NAND avec plus de 3 entrées se construisent en suivant le même schéma.

Cependant un grand nombre de transistors en série entraîne un délai de porte insupportablement long.

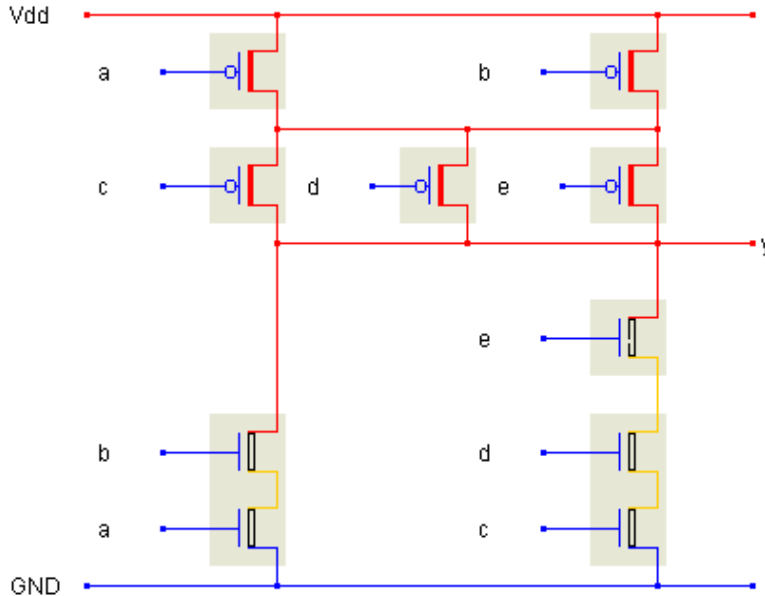
Les nortes On appelle norte complexe des nortes demandant à la fois des symboles ET et des

complexes symboles OU pour leur dessin au niveau logique. Cependant les réseaux de transistors restent raisonnablement simples.



Applet de porte complexe AOI (and-or-inverter): $y = \overline{(a \wedge b) \vee (c \wedge d \wedge e)}$

Cliquez près de l'une des entrées pour changer sa tension



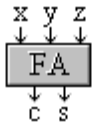
a	b	c	d	e	y
0	0	0	0	0	1
0	0	0	0	1	1
1	0	0	0	0	1
1	0	0	0	1	1
0	1	0	0	0	1
0	1	0	0	1	1
1	1	0	0	0	1
1	1	0	0	1	1
0	0	1	0	0	1
0	0	1	0	1	1
1	0	1	0	0	1
1	0	1	0	1	1
0	1	1	0	0	1
0	1	1	0	1	1
1	1	1	0	0	0
1	1	1	0	1	0
0	0	0	1	0	1
0	0	0	1	1	0
1	0	0	1	0	1
1	0	0	1	1	0
0	1	0	1	0	1
0	1	0	1	1	0
1	1	0	1	0	1
1	1	0	1	1	0
0	0	1	1	0	1
0	0	1	1	1	0
1	0	1	1	0	1
1	0	1	1	1	0
0	1	1	1	0	1
0	1	1	1	1	0
1	1	1	1	0	0
1	1	1	1	1	0



La généralisation à d'autres portes complexes est facile.

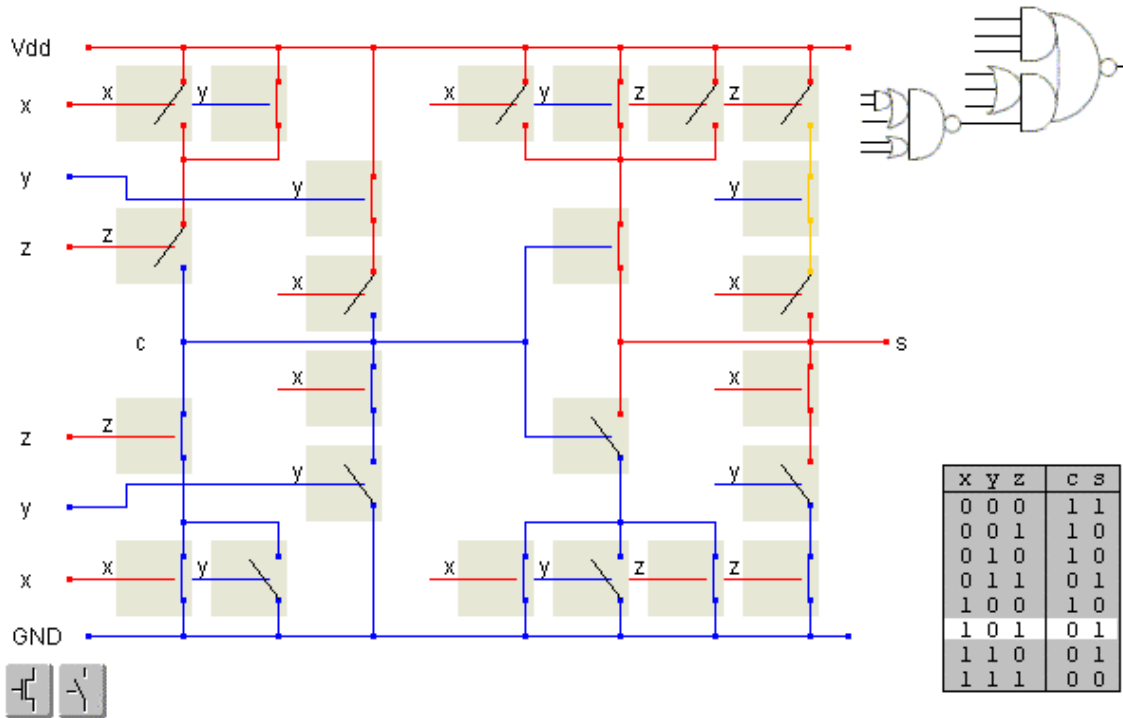
Ainsi la plupart des bibliothèques de portes précaractérisées contiennent des **AOI** (and-or-inverted), **OAI** (or-and-inverted), etc..

L'additionneur binaire



La cellule de "Full Adder" (FA) est formée de deux portes complexes connectées. Elle réalise une égalité arithmétique: la somme pondérée des 3 entrées "x", "y" et "z" a toujours la même valeur que la somme pondérée des deux sorties "c" et "s", c'est à dire que " $x + y + z = 2_*c + s$ ". On peut aisément vérifier cette propriété grâce à la table de vérité.

Applet de la cellule de "Full Adder", cliquer les entrées x, y ou z à gauche.

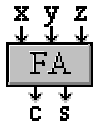


Le réseau de transistors canal P est symétrique au réseau de transistors canal N. Un circuit ayant cette propriété est appelé "miroir". Tous les additionneurs ont cette propriété qui découle d'un lien arithmétique entre le complément logique et le complément arithmétique.

Enfin les sorties de ce circuit sont inversées. Cela découle d'une propriété de la technologie CMOS qui ne permet de faire facilement que les fonctions logiques non croissantes.

Additionneurs

Cellule "FA" Dans la cellule "FA", la somme pondérée des bits en sortie est égale à la somme pondérée des bits en entrée c'est à dire que " $x + y + z = 2*c + s$ ". Les trois bits en entrée ont le même poids, disons 1 pour fixer les idées, le bit sortant "s" a le même poids que "x", "y" et "z" et "c", un poids double (2).



La cellule "FA" conserve la *valeur numérique* comme en électronique les nœuds conservent le *courant*.

La cellule "FA" est également appelée "réducteur $3 \Rightarrow 2$ " car elle réduit le nombre de bits de 3 à 2 tout en conservant la *valeur numérique*.

Vérifiez que vous maîtrisez la table de vérité de la cellule "FA".

Donnez les valeurs des sorties c et s en fonction des entrées x, y et z. Les valeurs sont modifiées en cliquant.

X	0
y	0
z	0

0

1 1 1

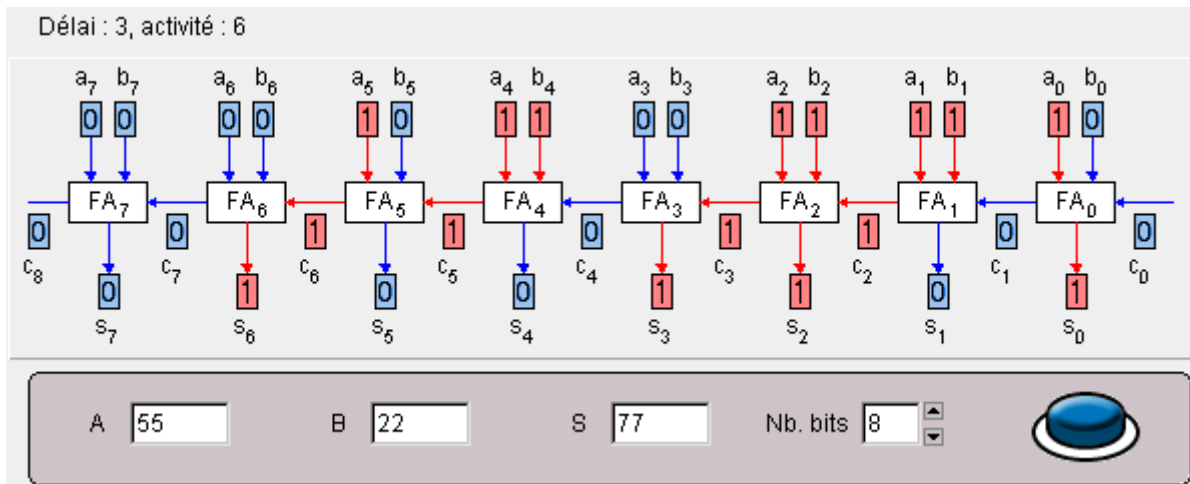
FA

2 1

0

c s

Additionneur à propagation L'addition est de très loin l'opération arithmétique la plus commune des processeurs numériques. D'abord parce que l'addition en tant que telle est très fréquente, ensuite parce que l'addition est la base d'autres opérations arithmétiques comme la multiplication, la division, l'extraction de racines et les fonctions élémentaires. Tout assemblage "cohérent" de cellules "FA" conserve la propriété: *la somme pondérée des bits qui sortent vaut la somme pondérée des bits qui entrent*. Pour obtenir l'additionneur $S = A + B$, il faut que les bits entrant soient ceux des nombres A et B et les bits sortant forment le nombre S.



Performance de l'additionneur à propagation

Soit un additionneur à propagation à travers n cellules "FA", si on suppose que toutes les valeurs de A et B sont équiprobables et indépendantes, les résultats théoriques suivants viennent:

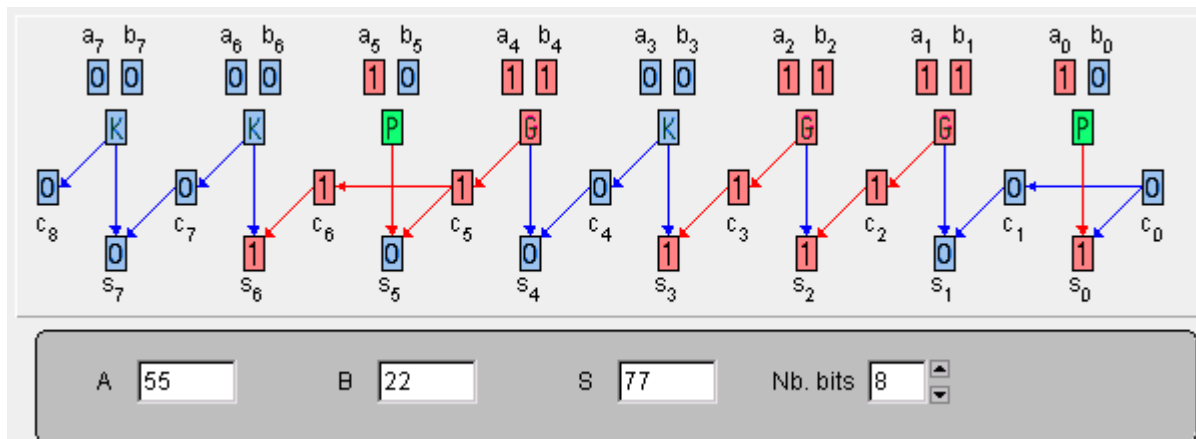
	minimum	moyen	maximum
délai	0	$\log_2(n)$	n
activité	0	$3n / 2$	$n^2 / 2$

Le délai maximum (pire cas) n'est en général pas acceptable. Il est dû au chemin de la retenue qui traverse toutes les cellules "FA". Etudions donc le cheminement de la retenue.

Cheminement de la retenue

Pour chaque cellule "FA" un des 3 cas se produit :

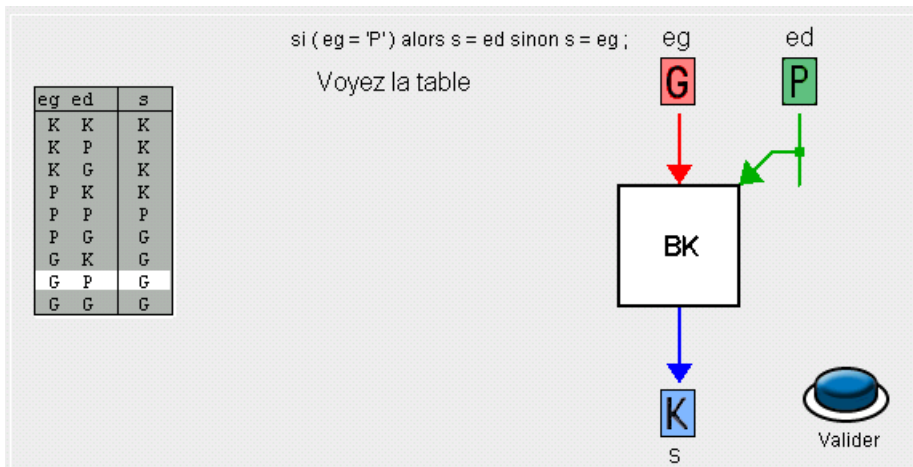
- la retenue c_{i+1} est mise à zéro, cas noté 'K', si $a_i = 0$ et $b_i = 0$
- la retenue c_{i+1} est mise à un, cas noté 'G', si $a_i = 1$ et $b_i = 1$
- la retenue c_{i+1} est propagée, cas noté 'P', si $(a_i = 0$ et $b_i = 1)$ ou $(a_i = 1$ et $b_i = 0)$. Dans ce cas $c_{i+1} = c_i$. Ce dernier cas, défavorable pour le délai, est matérialisé par une flèche horizontale.



Ces trois cas 'K', 'G' et 'P' sont codés sur 2 bits.

Cellule "BK" (Brent et Kung)

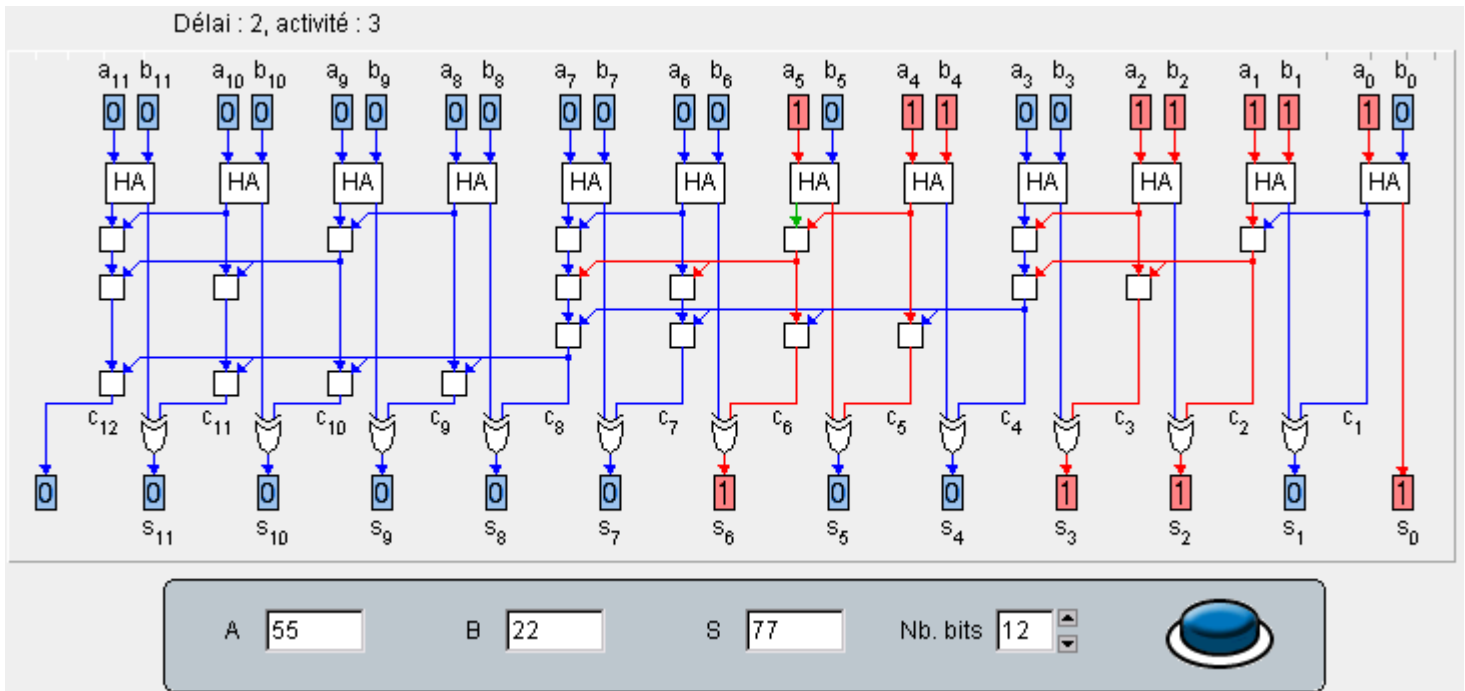
La cellule "BK" permet de calculer la retenue pour un bloc de 2 cellules "FA" ou plus généralement pour 2 blocs de cellules "FA".



Additionneur de Sklanski

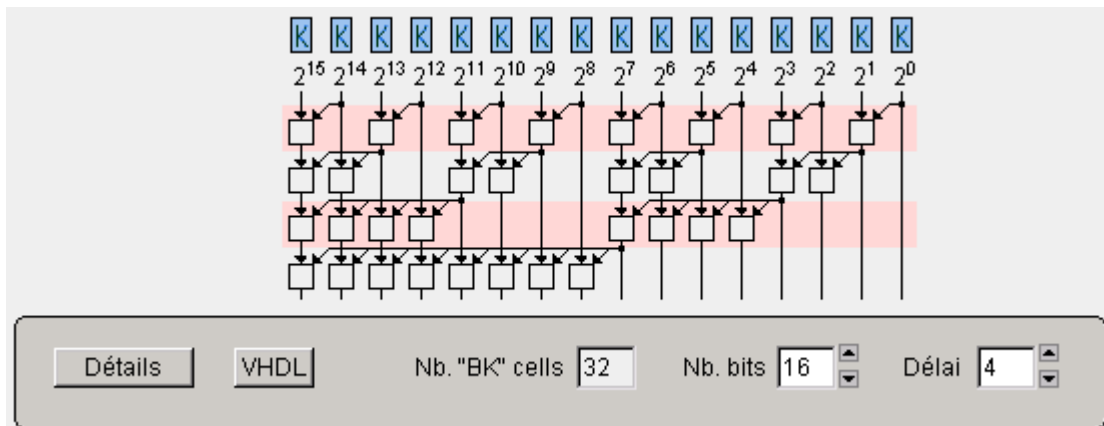
Pour concevoir un additionneur rapide, on va calculer toutes les retenues c_i par des arbres binaires de cellules "BK". L'additionneur de Sklanski construit des additionneurs de nombres de 2 bits 4 bits 8 bits 16 bits 32 bits en assemblant

chaque fois 2 additionneurs de taille inférieure. L'architecture est simple et régulière, mais n'est pas forcément la meilleure.



Additionneurs rapides (Brent et Kung)

Il y a une seule règle de construction des arbres imbriqués de cellules "BK":
 Toute sortie de rang i est relié à toutes les entrées de rang $j \leq i$ par un arbre de cellules "BK", ce qui permet d'entrelacer les arbres des n sorties de très nombreuses façons en fonction du nombre de bit et du délai de l'additionneur.



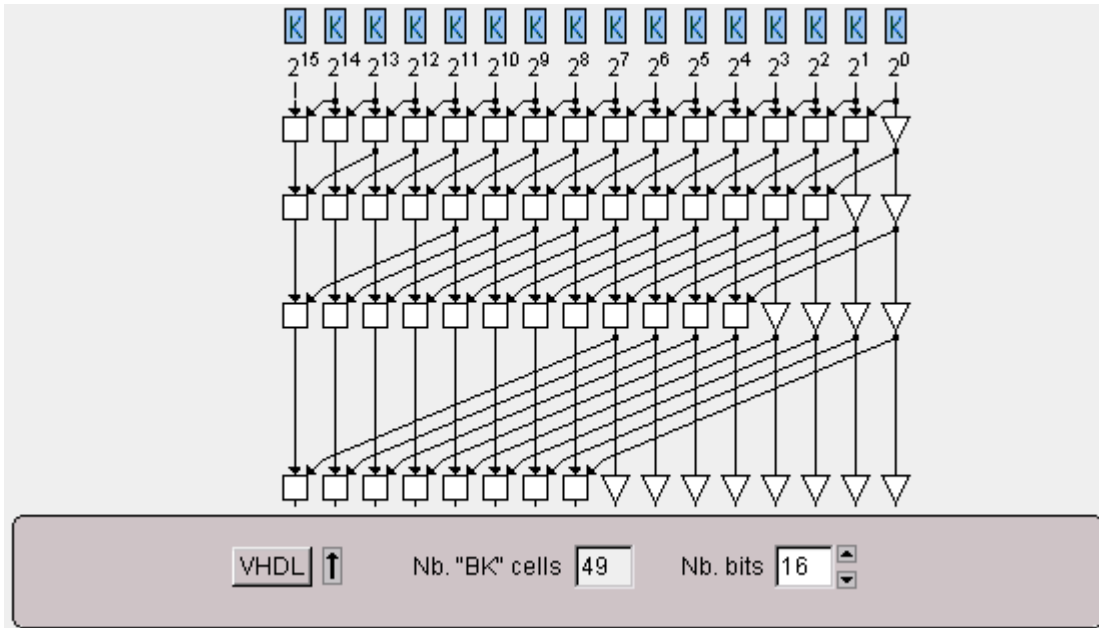
Dans les arbres de cellules "BK" on peut troquer des cellules "BK" contre du délai: moins il y a de délai (cellules "BK" à traverser) de calcul, plus l'additionneur utilise de cellules.

- faites varier le nombre de bits et le délai
- vérifier que les arbres suivent la règle de construction en cliquant sur un signal, remarquer que chaque signal a un nom unique composé de 2 chiffres.
- simuler un calcul de retenue en cliquant les touches **K**.
- voir les détails de la construction des arbres avec la touche "Détails"
- voir la description en VHDL de l'additionneur avec la touche "VHDL". Pour récupérer la description VHDL dans un éditeur de texte, la copier puis coller.

Additionneurs de Kogge et Stone

Les arbres binaires de cellules "BK" des additionneurs de Kogge et Stone ne partagent pas. En conséquence la sortance des signaux est réduite au minimum au prix de cellules "BK" plus nombreuses et comme le délai dépend de la sortance il est

Stone un peu plus court.



Additionneurs de Ling Dans un additionneur de Ling, les arbres de "BK" calculent une primitive appelée "pseudo retenue". Ceci permet d'éviter le calcul des p_i et g_i , mais impose en revanche un calcul supplémentaire pour obtenir la retenue à partie de la "pseudo retenue". L'astuce est que le délai de ce calcul supplémentaire est recouvert par le délai des cellules "BK". En conséquence l'additionneur est plus rapide (un peu). La synthèse VHDL de l'applet précédent en tire partie.

Cellule "CS" Dans la cellule "CS", la somme pondérée des sorties est égale à la somme pondérée des entrées c'est à dire que " $a + b + c + d + e = 2 \cdot h + 2 \cdot g + f$ ". La cellule "CS" n'est pas seulement un "réducteur $5 \Rightarrow 3$ ", car en plus la sortie "h" ne dépend jamais de l'entrée "e".

Vérifiez que vous maîtrisez la table de vérité de la cellule "CS".

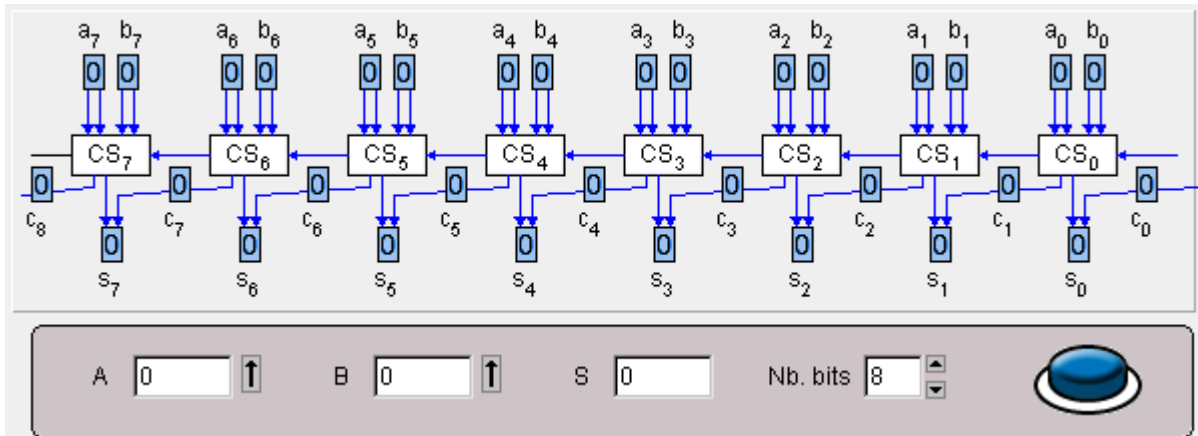
Donnez les valeurs des sorties h g et f en cliquant dessus puis en validant, en fonctions des entrées a, b, c, d et e données ou choisies par simple click

	a	b	c	d	e
0	0	0	0	0	0

Diagram of the CS cell showing inputs a, b, c, d, e and outputs h, g, f. The cell is labeled 'CS' and has a '1' on the right side. The output 'h' is connected to a '0' box, 'g' to a '0' box, and 'f' to a '0' box. A 'Valider' button is present.

Additionneur sans propagation La cellule "CS" ne propage pas la retenue de l'entrée "e" vers la sortie "h". Elle permet donc de réaliser des additionneurs *sans propagation de retenue*. Le nombre de cellules CS nécessaires est imposé par le nombre de chiffres des entiers à ajouter. Cependant chaque chiffre est maintenant représenté sur 2 bits et la valeur du chiffre est

la somme de ces deux bits. Les valeurs possibles des chiffres sont donc '0', '1' et '2' .



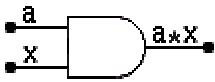
Ce système d'écriture des nombres entiers permet l'addition en un temps court et indépendant du nombre de chiffre des nombres.

Cependant il utilise deux fois plus de bits que la notation standard. En conséquence une même valeur a plusieurs écritures. La flèche verticale **↑** à côté des nombres en change l'écriture sans changer la valeur. Parmi toutes les écritures, celle n'utilisant que des '0' et des '1' est unique.

Multiplieurs

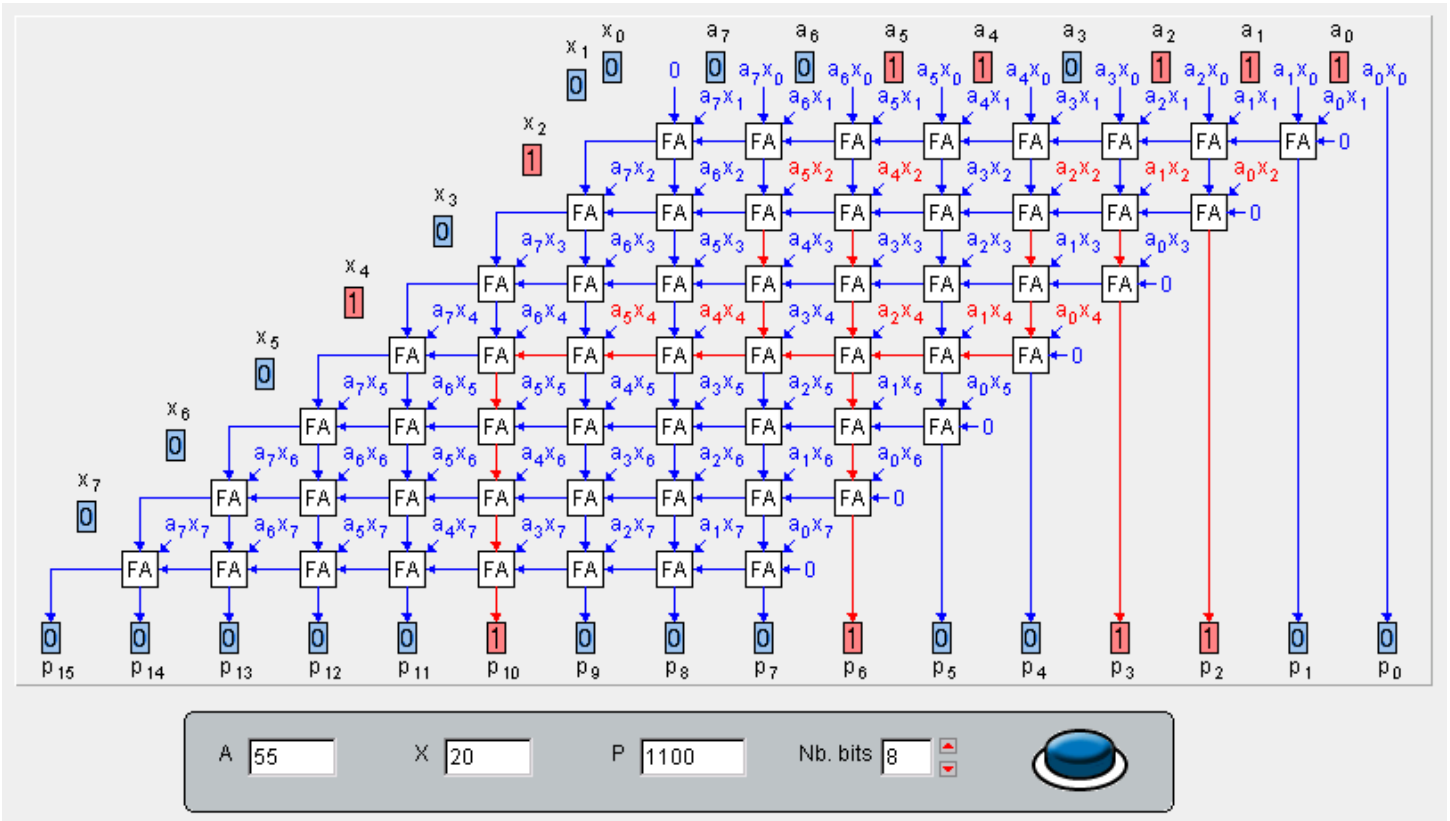
Multiplieur La multiplication vient en second pour la fréquence d'utilisation.

Porte AND Une porte "AND" multiplie 2 bits. Pour multiplier 2 nombres A et X de n bits chacun, on utilise n^2 portes "AND" qui multiplie chaque bit de A par chaque bit de X. La somme pondérée de ces n^2 bits a bien comme valeur le produit $P = A * X$. Cependant cet ensemble de bits n'est pas un nombre, bien que sa valeur se calcule comme celle d'un nombre.



Comme $A < 2^n$ et $X < 2^n$, le produit $P < 2^{2n}$ et s'écrit donc sur $2n$ bits.

Multiplication sans signe Une structure régulière de portes "AND" et de cellules "FA" formant un assemblage "cohérent" permet d'obtenir les produits partiels puis de les réduire pour finalement obtenir le nombre P. Comme chaque cellule "FA" réduit le nombre de bits de 1 exactement (tout en conservant la valeur de P), la quantité de cellules "FA" strictement nécessaire est $n^2 - 2n$ (nombre de bits entrant - nombre de bits sortant). Cependant il y en a ici un peu plus car il entre aussi quelques '0' qu'il faut aussi réduire, plus précisément un '0' par chiffre de X.



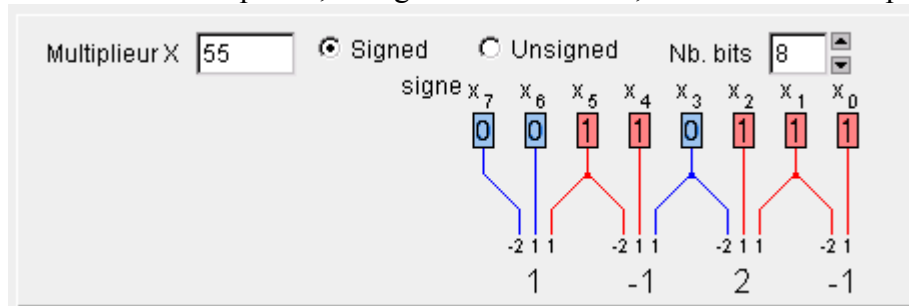
Multiplieurs rapides Plusieurs pistes conduisent à une amélioration de la vitesse:

- Diviser le nombre de bits à réduire en utilisant des grandes bases.
- Utiliser une structure cohérente de cellules "FA" en arbre.
- Utilisation de cellules "CS", dont le pouvoir de réduction double de celui du "FA" permet en outre les arbres binaires.

Codeur de Booth Passer à une base de numération plus grande réduit mécaniquement le nombre de chiffres du multiplieur X.

Voyons la base 4, qui a utilisé deux fois moins de chiffres que la base 2 pour représenter le même nombre. Le "Code de Booth" (ou "BC") est le code de

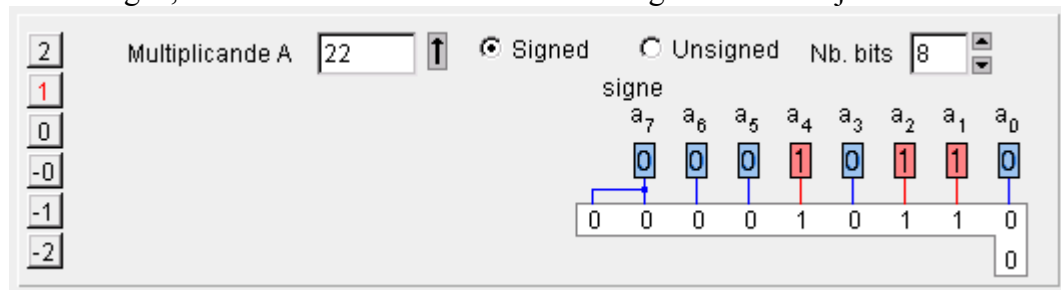
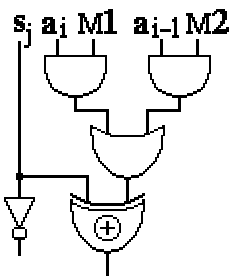
redondance minimale, symétrique, en base 4. Les chiffres $\in \{-2, -1, -0, 0, 1, 2\}$. Le code sur 3 bits adopté ici, en signe/valeur absolue, donne 2 notations pour le zéro.



Cependant les produits partiels sont calculés avec une cellule plus complexe qu'un "AND".

Multiplieur des bits de A par un chiffre "BC"

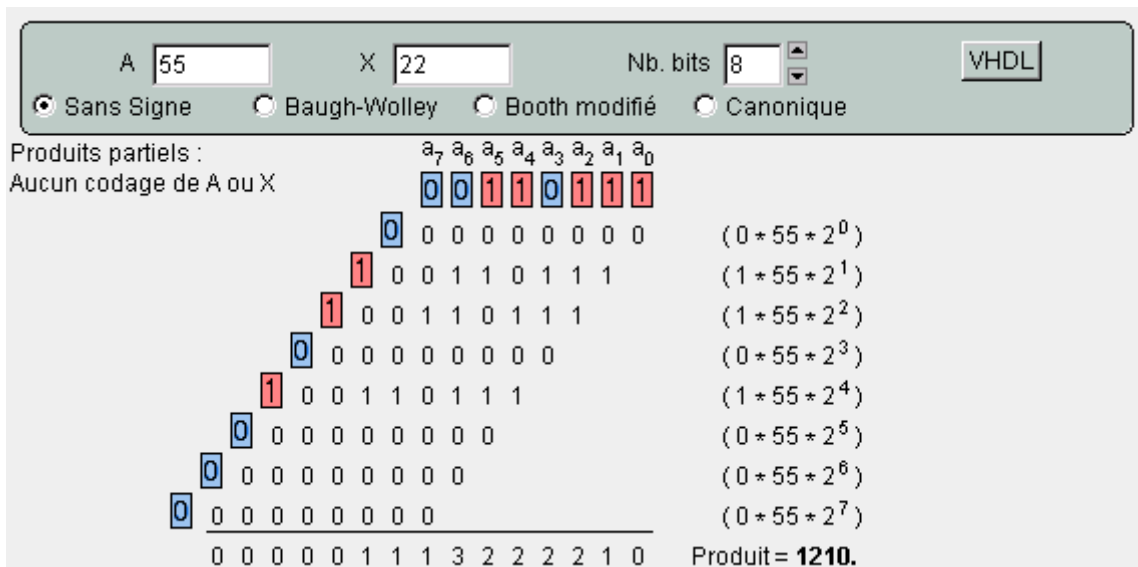
La multiplication par un chiffre "BC" $\in \{-2, -1, -0, 0, 1, 2\}$ rajoute 2 bits à ceux de A: un pour avoir A ou 2A, un autre pour la retenue entrante en cas de soustraction. A étant signé, il faut éventuellement étendre son signe sur le bit ajouté.



La multiplication de A requiert autant de cellule "CASS" que de bit de A plus 1.

Génération des produits partiels

La première étape de la multiplication génère à partir de A et de X des bits dont la somme pondérée vaut le produit P. Le bit de poids fort de P est positif pour la multiplication d'entiers sans signe, et négatif pour la multiplication d'entiers en complément à 2.



Réduction des produits partiels

La deuxième étape de la multiplication réduit les produits partiels de l'étape précédente à deux nombres, en conservant la somme pondérée des bits. Ces deux nombres seront additionnés dans la troisième étape.

La synthèse des arbres de Wallace suit l'algorithme de Dadda, qui garanti le minimum d'opérateurs de réduction. Si de plus on impose d'effectuer les réductions le plus tôt ou le plus tard possible alors la solution est unique et synthétisée toujours de

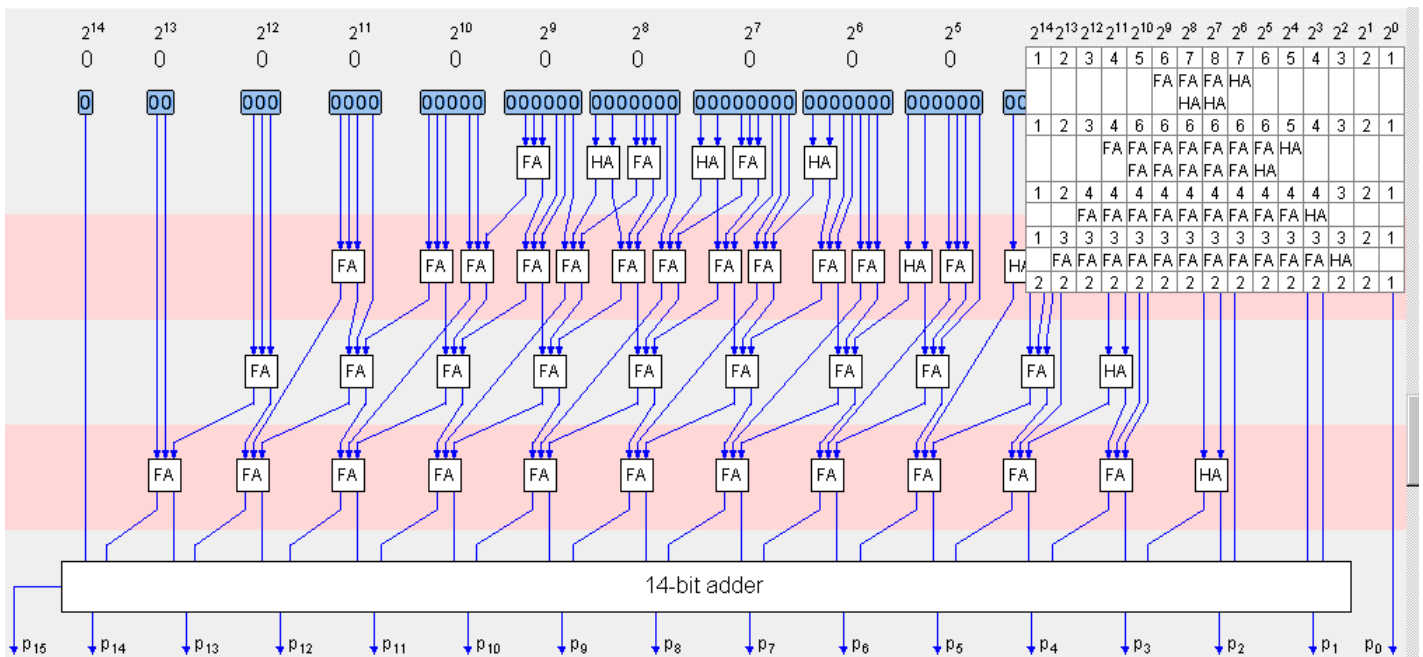
la même façon.

Les deux nombres binaires à ajouter dans la troisième étape peuvent être vus comme un nombre en "CS".



Exemple d'arbre de Wallace

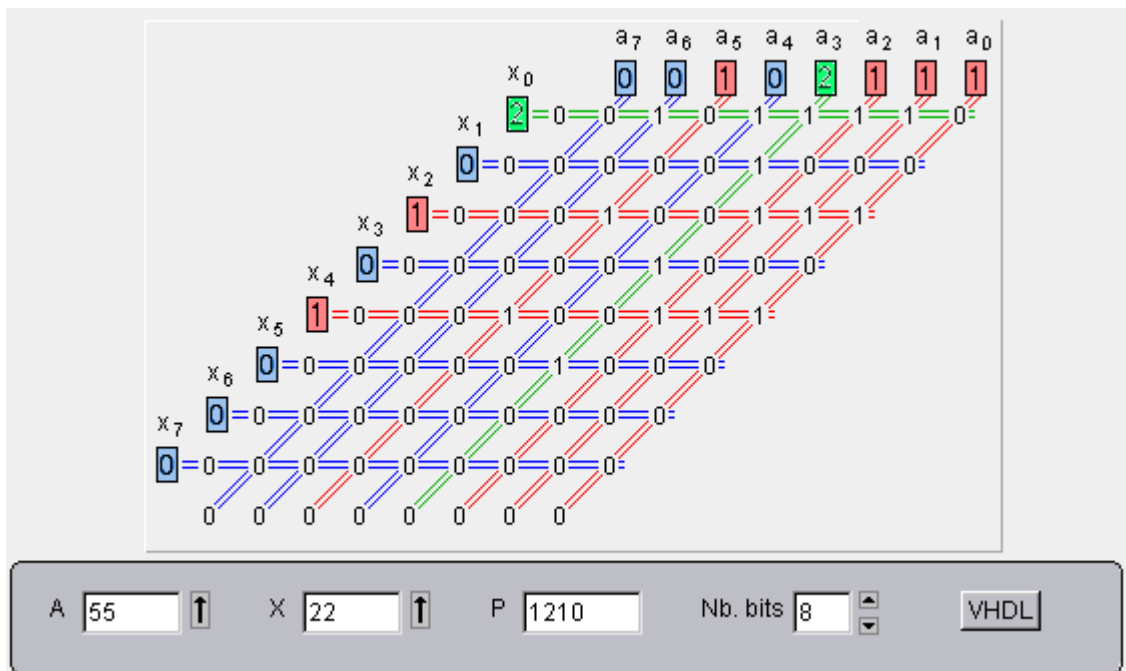
L'applet suivant réduit 8^2 produits partiels (par exemple le produit de deux nombres sans signe de 8 bits chacun). Les arbres de Wallace réduisent "au plus tard" (touche "tard" de l'applet ci-dessus). La somme pondérée des 16 bits qui sortent vaut toujours la somme pondérée des 64 bits qui entrent.



Produits partiels d'opérandes en "CS"

Le multiplicateur X et le multiplicande A sont tous les deux en "CS", c'est à dire avec des chiffres $\in \{0, 1, 2\}$.

On génère des chiffres dont la somme pondérée est égale à $A * X$. Pour assurer que ces chiffres sont des bits (faciles à additionner), il est nécessaire que dans A et dans X tout chiffre '2' soit précédé à droite par un '0'.



Réduction des produits partiels Les produits partiels de la multiplication de nombres en "CS" sont simplement des bits, réduits de la même façon que pour les multiplieurs précédents.

Mise à zéro Tot Tard 1 2 3 4 VHDL
 Utiliser des CS Propag. localement feuille : 1

2^{16}	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	2	3	4	5	6	7	8	9	8	7	6	5	4	3	2	1

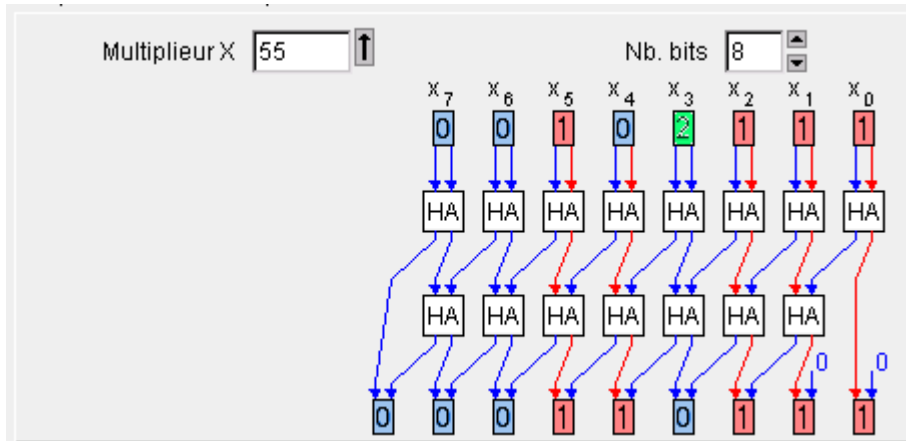
Cliquer pour ajouter un "HA"
 Touche Majuscule pour ajouter un "FA"
 Touche CTRL pour effacer

Cellule "xCS" La cellule "xCS" permet de calculer le produit de deux chiffres a et x en "CS". Son équation arithmétique est $2 * b + 2 * y + i = a * x + z + c$. De plus les sorties "b" et "y" ne dépendent pas des entrées "c" ou "z" ; il n'y a pas de propagation.

Voyez la table $2*b + 2*y + i = a*x + z + c$

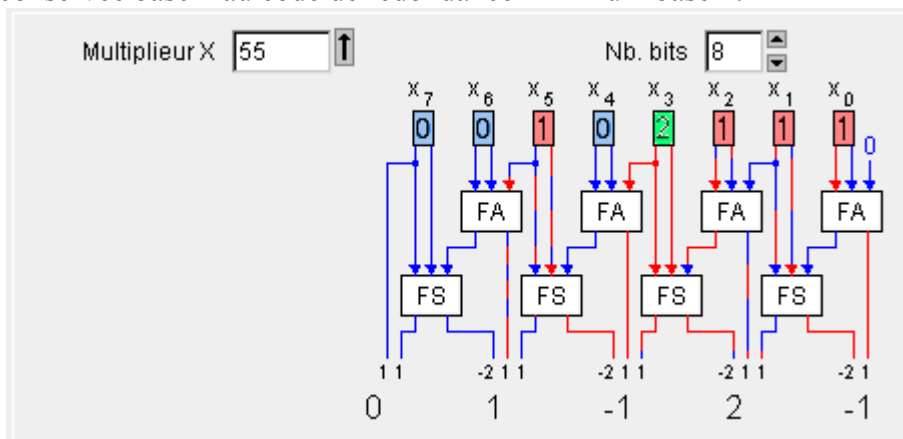
a	x	z	c	b	y	i
0	0	0	0	0	0	0
0	1	0	0	0	0	0
0	2	0	0	0	0	0
0	2	1	0	0	0	1
1	0	0	0	0	0	0
1	1	0	0	0	0	1
1	2	0	0	0	1	0
1	2	1	0	0	1	1
2	0	0	0	0	0	0
2	0	0	1	0	0	1
2	1	0	0	1	0	0
2	1	0	1	1	0	1
2	2	0	0	1	1	0
2	2	0	1	1	1	1
2	2	1	0	1	1	1
2	2	1	1	overflow		

Codeur Le transcodeur "CS2CS" passe de "CS" à "CS" en garantissant qu'en sortie un '2' est "CS2CS" toujours précédé à droite par un '0'.



Cela permet de calculer les produits partiels binaires avec un multiplicande A et un multiplieur X tous les deux en "CS".

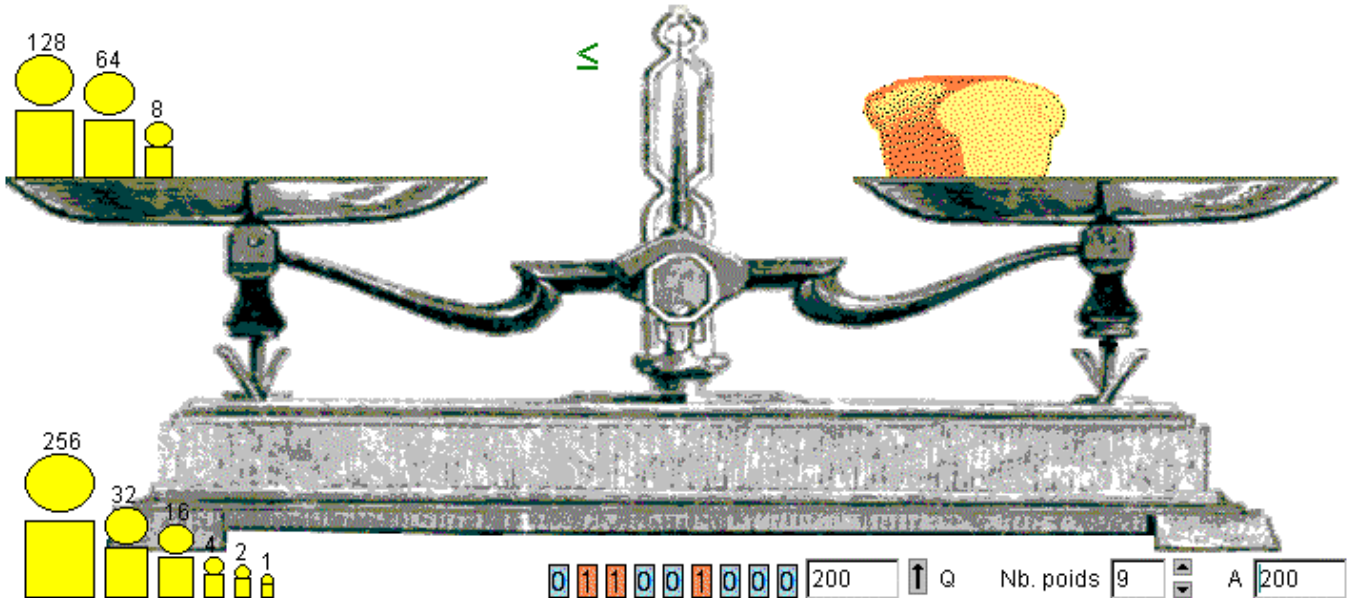
Codeur "CS2BC" Le transcodeur "CS2BC" passe de "CS" à "BC", c'est à dire du code à retenue conservée base 2 au code de redondance minimum base 4.



Diviseurs

Pesée du pain avec restauration sans restauration, pesée RST

On veut calculer $Q = A \div D$. Par un coup de chance extraordinaire on dispose d'une balance, d'un pain de mie dont le poids est justement A et enfin d'une série de poids de valeur $D, 2D, 4D, 8D, \dots, 2^i \cdot D$ marqués respectivement avec $1, 2, 4, 8, \dots, 2^i$. En fait D est un nombre binaire, et $2^i \cdot D$ s'obtient simplement en décalant D . La balance compare la somme des poids sur chacun de ses deux plateaux (\leq ou $>$).



Divisions récurrentes

La division est peu fréquente, cependant comme son délai d'exécution est bien plus grand que celui de l'addition ou de la multiplication, son incidence sur le temps de calcul est substantielle. Il convient donc de soigner la réalisation des diviseurs.

On veut calculer $Q = A \div D$. Ceci est équivalent à $Q * D = A$. Donc si Q et D s'écrivent chacun avec n bits, A s'écrit avec $2n$ bits.

On va construire une suite $Q_n, Q_{n-1}, \dots, Q_2, Q_1, Q_0$ et une suite $R_n, R_{n-1}, \dots, R_2, R_1, R_0$ telles que l'invariant $A = Q_j * D + R_j$ soit respecté pour tout j .

La récurrence est :

- $Q_{j-1} = Q_j + q_{j-1} * 2^{j-1}$
- $R_{j-1} = R_j - q_{j-1} * D * 2^{j-1}$

avec comme conditions initiales :

- $Q_n = 0$
- $R_n = A$.

Quand la récurrence s'arrête, on a $Q = Q_0 = \sum_i q_i * 2^i$. $R = R_0$ est le reste de la division.

A	0000001000100110	= 550	
$D * 2^8$	00010110	= $22 * 2^8$	= 5632 > 550 (pas de débordement)
$R_8 = A$	0000001000100110	= 550	
$R_8 < D * 2^7$	00010110	$q_7 = 0, Q_7 = 0$	
$R_7 = R_8$	0000001000100110	= 550	→
$R_7 < D * 2^6$	00010110	$q_6 = 0, Q_6 = 00$	
$R_6 = R_7$	0000001000100110	= 550	→
$R_6 < D * 2^5$	00010110	$q_5 = 0, Q_5 = 000$	
$R_5 = R_6$	0000001000100110	= 550	→
$R_5 \geq D * 2^4$	00010110	$q_4 = 1, Q_4 = 0001$	↓
$R_4 = R_5 - D * 2^4$	0000000011000110	= $550 - 352 = 198$	↓
$R_4 \geq D * 2^3$	00010110	$q_3 = 1, Q_3 = 00011$	↓
$R_3 = R_4 - D * 2^3$	00000000000010110	= $198 - 176 = 22$	↓
$R_3 < D * 2^2$	00010110	$q_2 = 0, Q_2 = 000110$	
$R_2 = R_3$	00000000000010110	= 22	→
$R_2 < D * 2^1$	00010110	$q_1 = 0, Q_1 = 0001100$	
$R_1 = R_2$	00000000000010110	= 22	→
$R_1 \geq D * 2^0$	00010110	$q_0 = 1, Q_0 = 00011001$	↓
$R_0 = R_1 - D * 2^0$	00000000000000000	= $22 - 22 = 0$	↓
<hr/>			
Reste $R_0 =$	00000000000000000	= 0	
Quotient $Q_0 =$	00011001	= 25	
$Q_0 * D + R_0 =$	$25 * 22 + 0 = 550 + 0 = 550$		

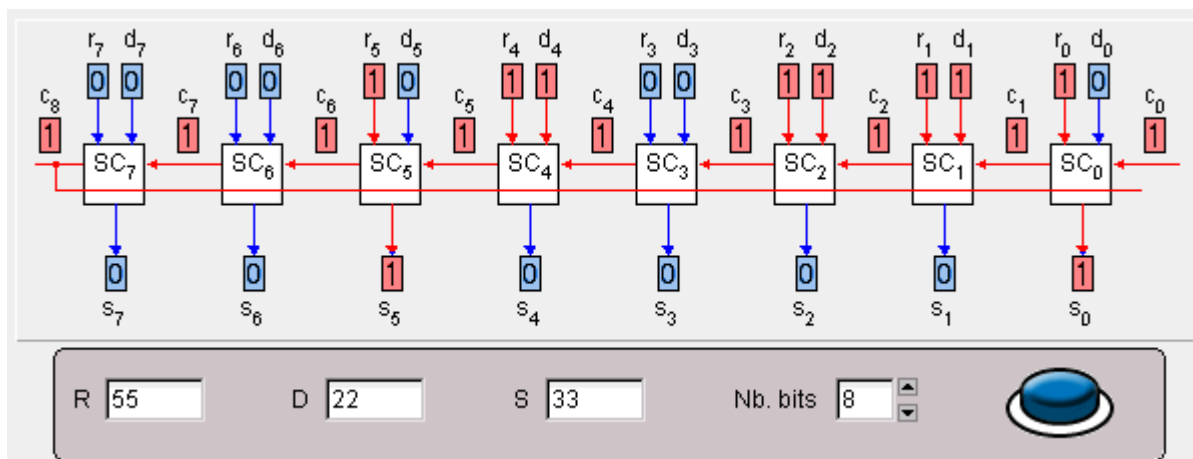
Avec restauration Sans restauration Nb. bits

Soustracteur conditionnel

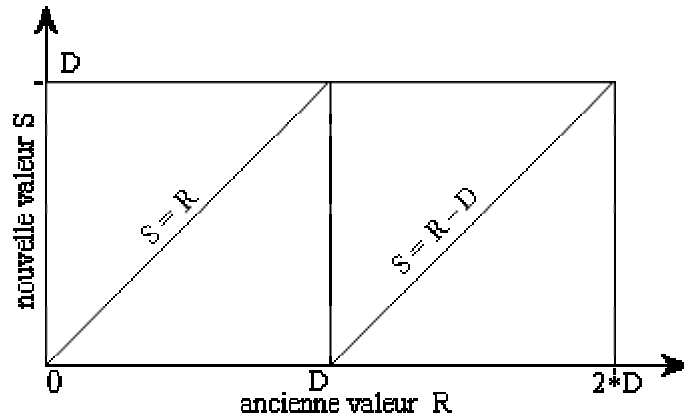
Un "soustracteur conditionnel" donne le résultat S suivant:

si $R < D$ **alors** $S = R$ **sinon** $S = R - D$;

Chaque cellule "SC" calcule le résultat et la retenue de la soustraction $R - D$. Si la retenue sortante (tout à gauche) vaut '1' alors S reçoit le résultat de la soustraction,



La fonction du "soustracteur conditionnel": **si** $R < D$ **alors** $S = R$ **sinon** $S = R - D$, se résume par sa fonction de transfert appelée "diagramme de Robertson". Pour converger, la division impose en outre que $0 \leq R \leq 2 * D$.



Diviseur avec restauration Un diviseur "avec restauration" consiste en une suite de décalages et de tentative de soustractions. Il est formé d'une structure régulière de cellules de soustraction conditionnelle "SC" (soustraction ou rien suivant un bit de condition).

- Diviseurs rapides** Trois approches se combinent pour réaliser des diviseurs rapides:
- Utilisation des additions/soustraction sans propagation de retenue.
 - Préconditionner le diviseur et le dividande pour simplifier la division.
 - Utiliser des grandes bases pour diminuer le nombre d'étapes.

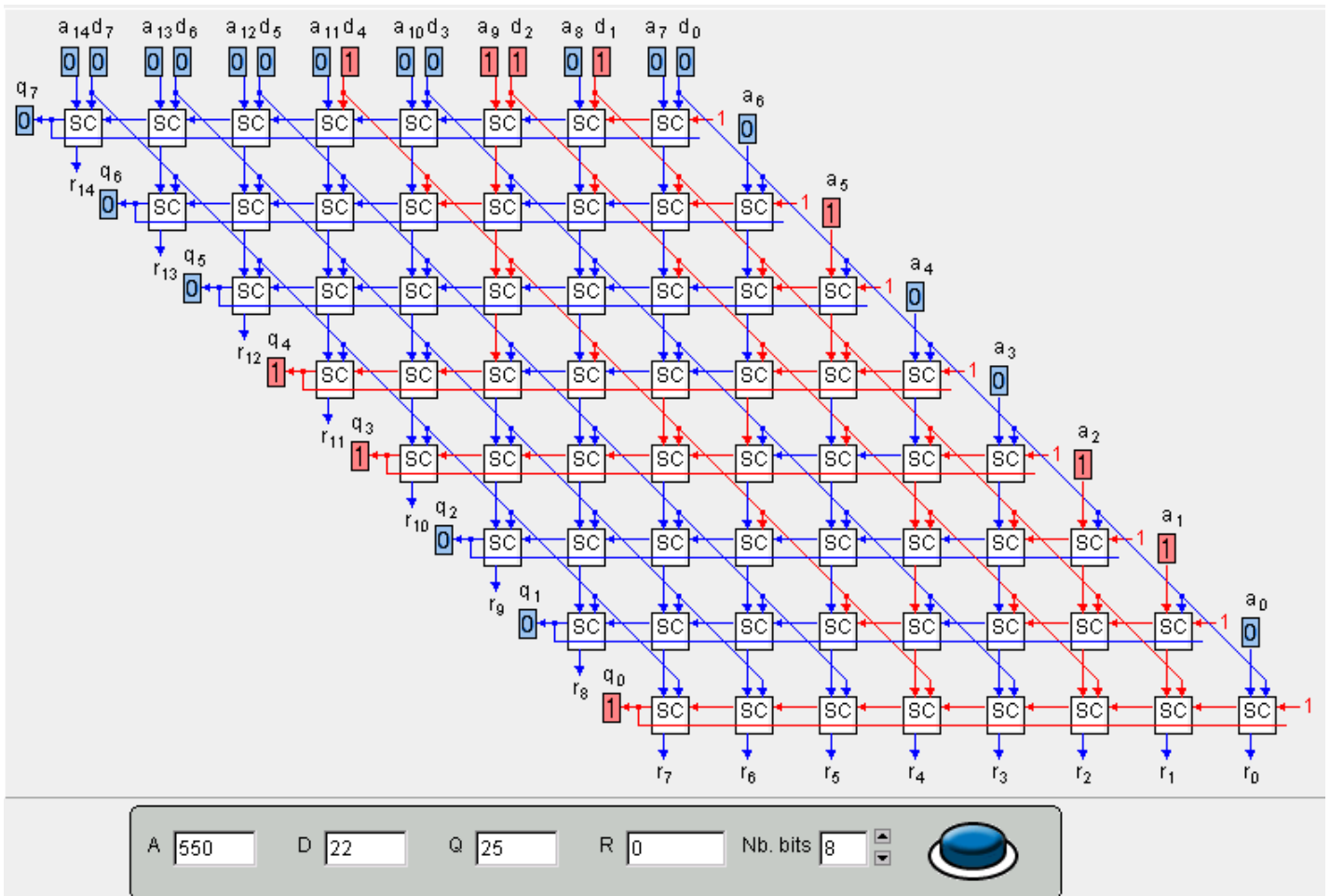
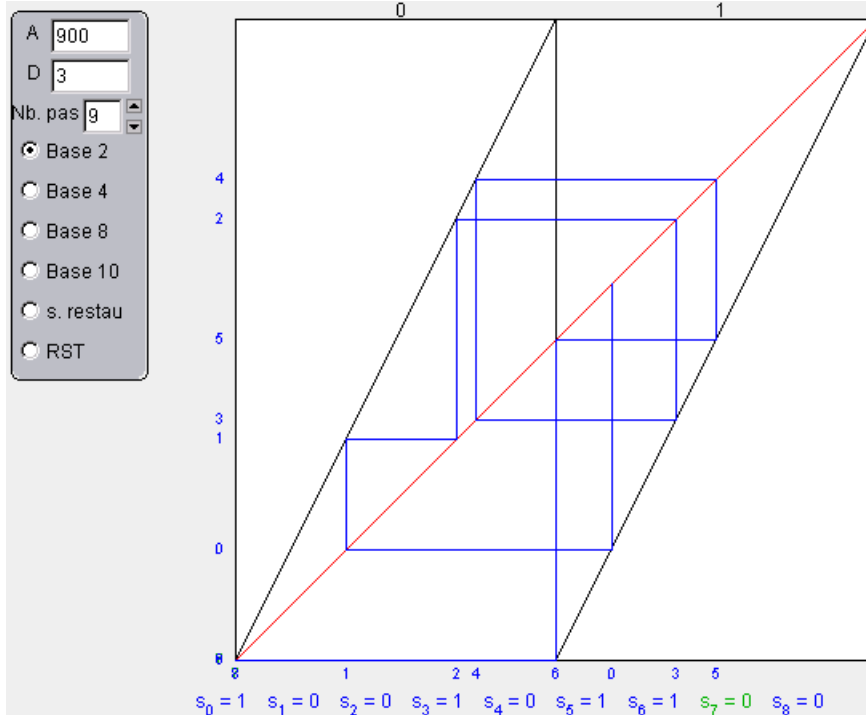


Diagramme de Robertson Pour avoir un diagramme de Robertson carré, on normalise les restes successifs: $(R_j * b^{-j})$ où b est la base de numération.

Les pentes noires représentent la fonction de transfert $R_i \Rightarrow R_{i+1}$. le trait rouge est la

fonction unité qui passe d'une étape à la suivante.

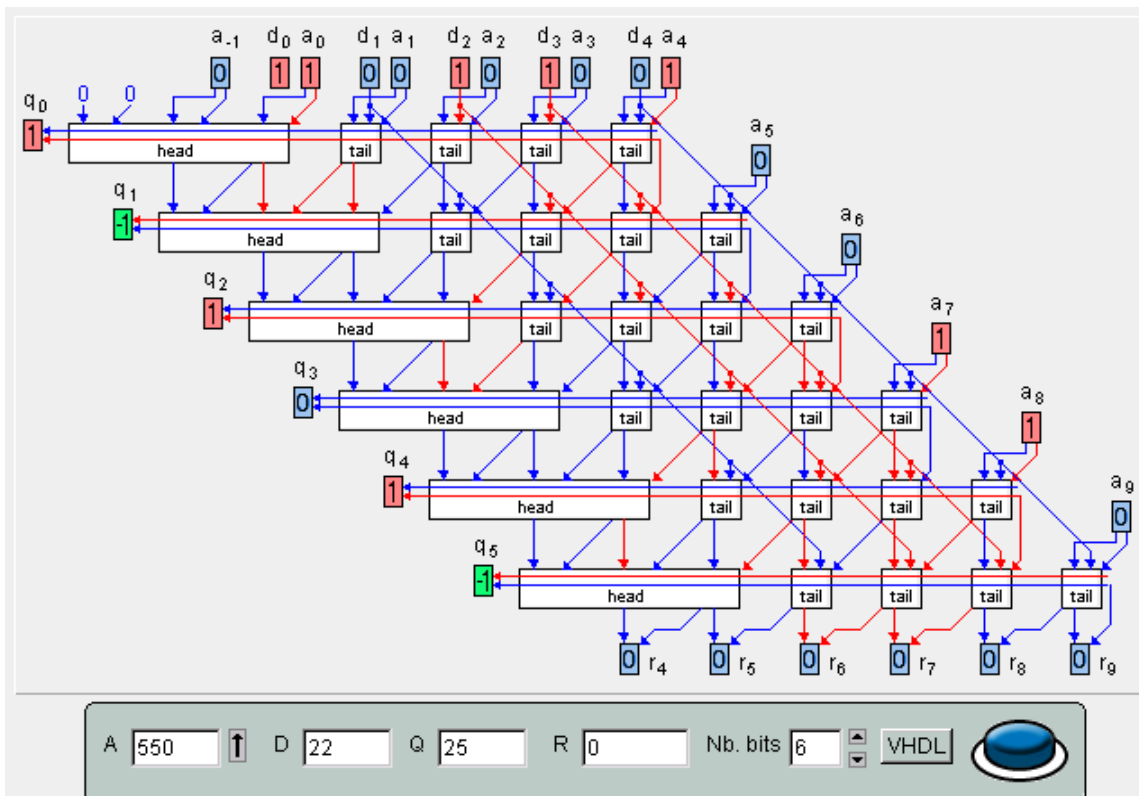


La base 10 nous est familière, elle ne sert ici que comme illustration car elle serait très inefficace en binaire.

Division "SRT" sans propagation de retenue

Pour éviter le délai dû à la propagation de retenue, on utilise ici une suite d'additionneur/soustracteur à emprunt conservé (BS). La cellule "tail", variante de la cellule "SC", exécute suivant les deux bits de commande qui la traversent:

- une addition : $R_{j-1} = R_j + 2^{j-1} * D$
- une soustraction : $R_{j-1} = R_j - 2^{j-1} * D$
- une identité : $R_{j-1} = R_j$



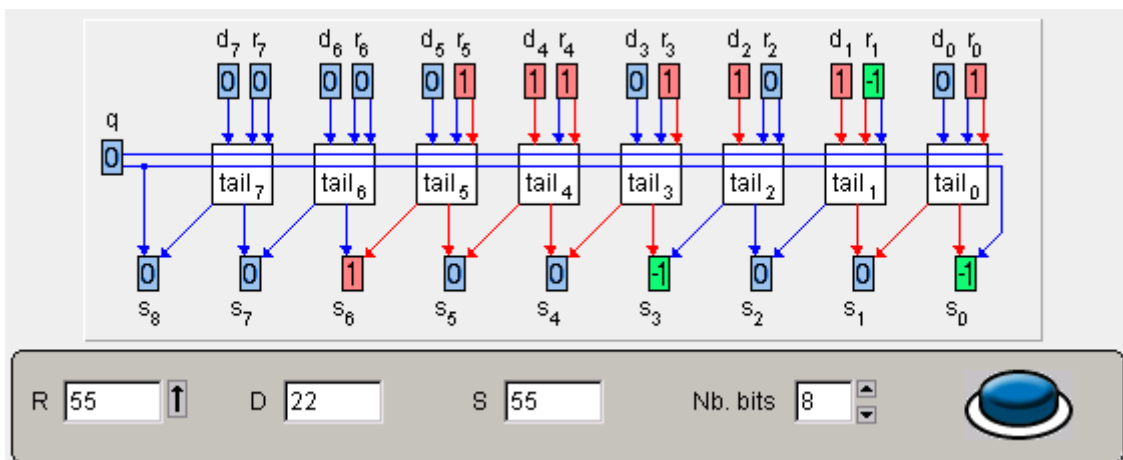
Cette opération est choisie suivant le signe du reste R_j . Pour connaître précisément ce signe il faudrait pouvoir examiner tous les chiffres du reste. On montre qu'il n'est nécessaire d'en examiner que 3. De plus on connaît la position de ces 3 chiffres: celui de gauche est aligné avec le premier bit à "1" du diviseur. Pour éviter que cette position ne se déplace avec les valeurs de D, D est "normalisé", c'est à dire que la position de son premier bit à "1" dans le diviseur est fixé.

**Additionneur/
Soustracteur
conditionnel sans
propagation**

Un "additionneur/soustracteur conditionnel" donne l'un des résultats S suivant:

- si q = '-1' alors $S = R + D$;
- si q = '0' alors $S = R$;
- si q = '1' alors $S = R - D$;

Chaque cellule "tail" réalise l'addition/soustraction de 1 bit. La retenue n'est pas propagée vers la cellule de gauche de la même ligne mais envoyée vers la cellule de la ligne suivante.



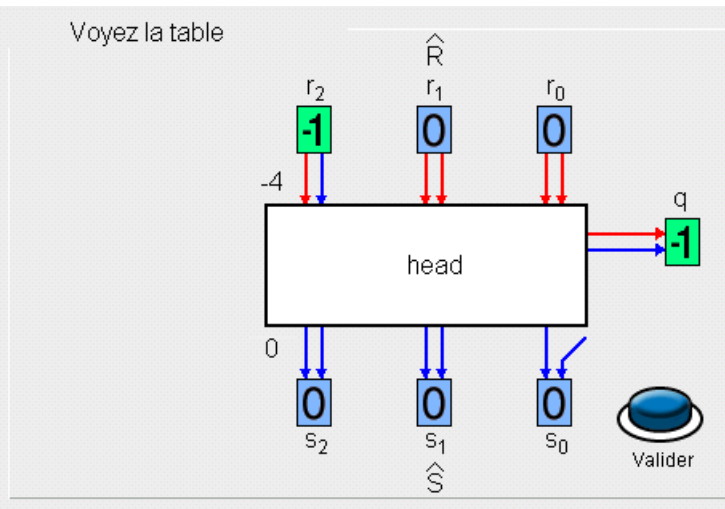
**Cellule de tête du
diviseur SRT**

Soient $\hat{R} = r_2 * 4 + r_1 * 2 + r_0$ et $\hat{S} = s_2 * 4 + s_1 * 2 + s_0$ les valeurs des entrées et des sorties de la cellule de tête.

- si $\hat{R} > 0$ alors $\{ \hat{S} = \hat{R} - 2 ; q = '1' ; \}$
- si $\hat{R} = 0$ alors $\{ \hat{S} = \hat{R} ; q = '0' ; \}$
- si $\hat{R} < 0$ alors $\{ \hat{S} = \hat{R} + 1 ; q = '-1' ; \}$

Lors d'une division (sans débordement), la sortie s_2 est toujours à 0.

r2	r1	r0	s2	s1	s0	q
+1	+1	+1	+1	+1	-1	+1
+1	+1	0	+1	0	0	+1
+1	+1	-1	+1	0	-1	+1
+1	0	+1	+1	0	-1	+1
+1	0	0	0	+1	0	+1
+1	0	-1	0	+1	-1	+1
+1	-1	+1	0	+1	-1	+1
+1	-1	0	0	0	0	+1
+1	-1	-1	0	0	-1	+1
0	+1	+1	0	+1	-1	+1
0	+1	0	0	0	0	+1
0	+1	-1	0	0	-1	+1
0	0	+1	0	0	-1	+1
0	0	0	0	0	0	0
0	0	-1	0	0	0	-1
0	-1	+1	0	0	0	-1
0	-1	0	0	0	-1	-1
0	-1	-1	0	-1	0	-1
-1	+1	+1	0	0	0	-1
-1	+1	0	0	0	-1	-1
-1	+1	-1	0	0	-1	-1
-1	0	+1	0	-1	0	-1
-1	0	0	0	-1	-1	-1
-1	0	-1	-1	0	0	-1
-1	-1	+1	-1	0	0	-1
-1	-1	0	-1	0	-1	-1
-1	-1	-1	-1	-1	0	-1



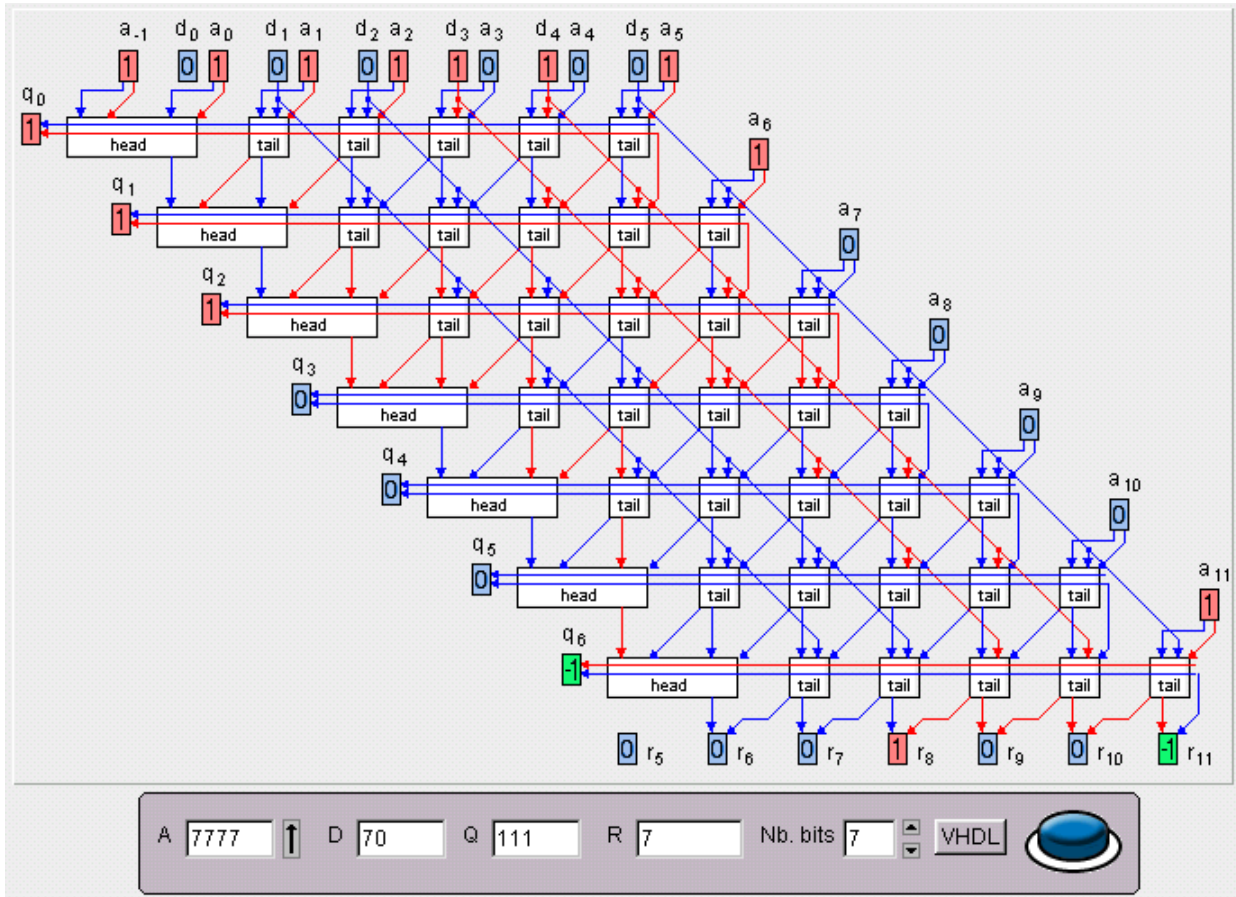
Division "SRT" avec réduction du diviseur

La division "SRT" précédente est simple car le premier bit du diviseur D est toujours "1". Elle se simplifie davantage si les deux premiers bits d_0 et d_1 du diviseur D sont réduits à "1 0" par l'opération :

si d_1 alors $\{ D = D * 3/4 ; A = A * 3/4 ; \}$.

Cette multiplication de A et D par la même constante ne modifie pas le quotient Q, par contre le reste R est aussi multiplié.

Pour un diviseur sur n bits, $2^{n-1} - 1 < D < 2^{n-1} + 2^{n-2}$.

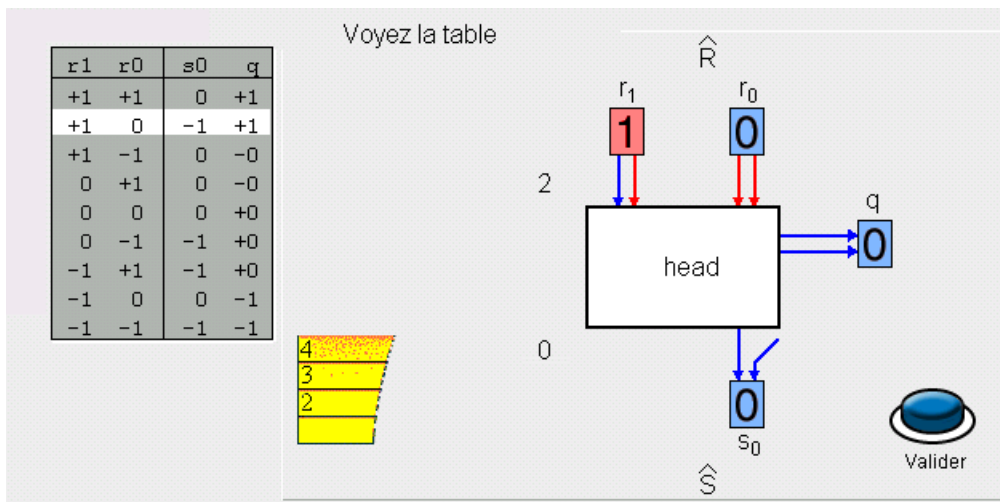


Cellule de tête du diviseur "SRT" avec réduction

Soient $\hat{R} = r_1 * 2 + r_0$ la valeur de l'entrée de la cellule "head".

- si $\hat{R} > 1$ alors $\{ s_0 = \hat{R} - 3 ; q = +1 ; \}$
- si $\hat{R} = 1$ alors $\{ s_0 = 0 ; q = -0 ; \}$
- si $\hat{R} = 0$ alors $\{ s_0 = 0 ; q = +0 ; \}$ ou bien $\{ s_0 = -1 ; q = -0 ; \}$
- si $\hat{R} = -1$ alors $\{ s_0 = -1 ; q = +0 ; \}$
- si $\hat{R} < -1$ alors $\{ s_0 = \hat{R} + 2 ; q = -1 ; \}$

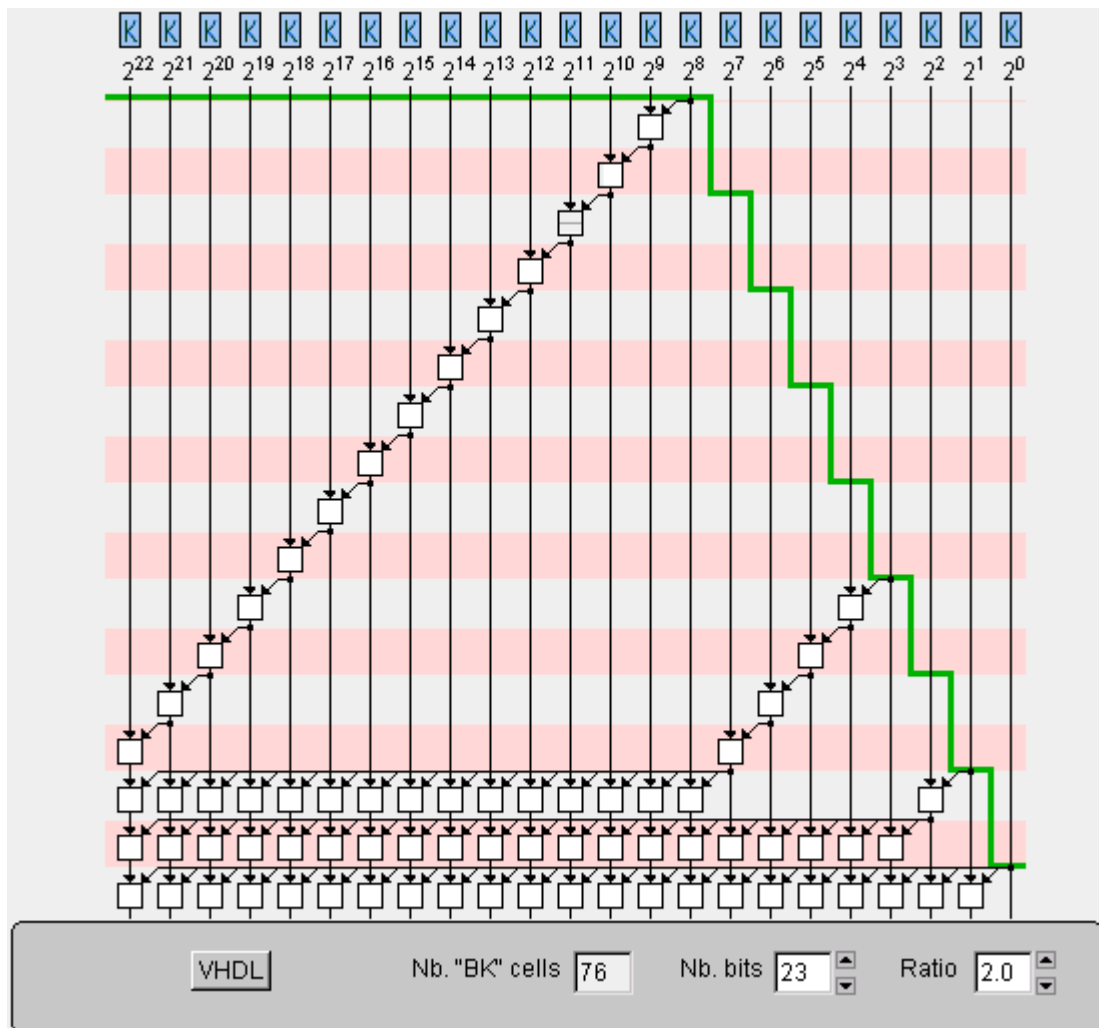
La différence entre les deux écritures de 0 pour q : "- 0" et "+ 0" importe.



Convertisseur de quotient

Le quotient Q du diviseur SRT est en notation redondante. Sa conversion en notation conventionnelle passe par un additionneur (en fait un soustracteur). Comme les chiffres q du quotient sont calculés séquentiellement (poids forts d'abord), la conversion peut être menée en même temps que le calcul des chiffres q .

Soit "Ratio" le rapport entre le délai de la cellule de tête du diviseur SRT et le délai de la cellule "BK".

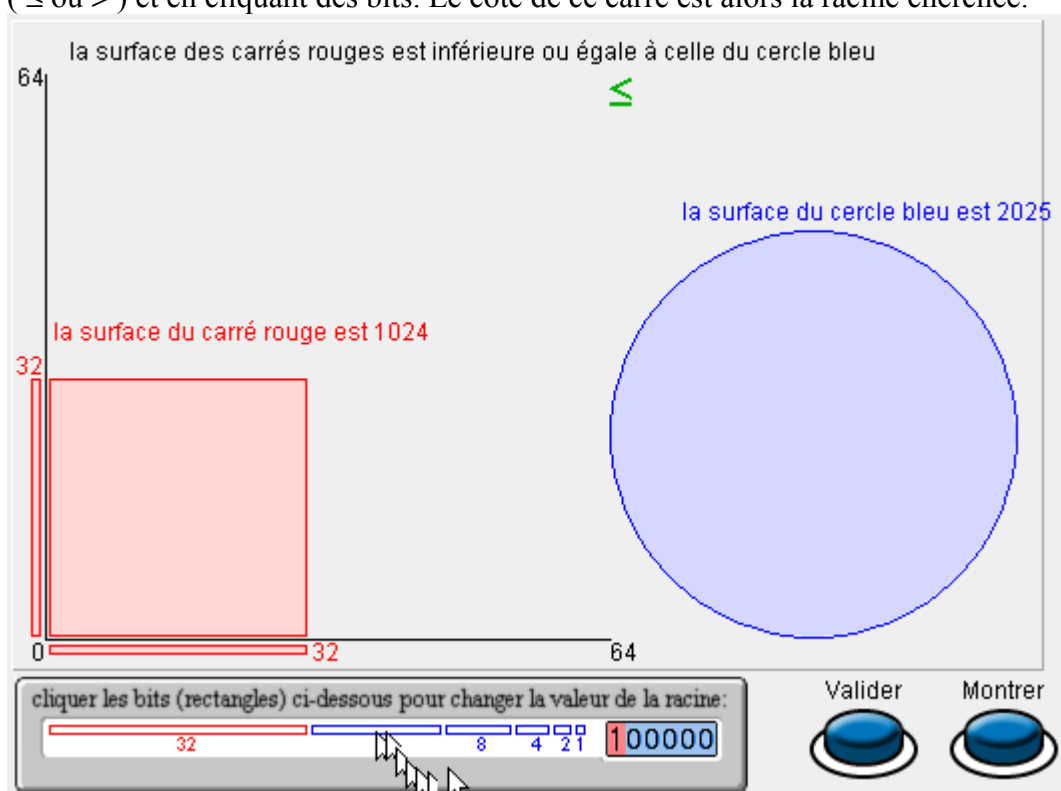


Extracteur de racine carrée

Extraction de racine carrée L'extraction de racine carrée est relativement peu fréquente. Cependant elle intervient dans les distances euclidiennes et dans les moindres carrés. L'opérateur d'extraction s'apparente au diviseur et tout ce que l'on sait de la division rapide s'applique à la racine carrée. Souvent le même opérateur rapide exécute soit la division, soit l'extraction de racine carrée, les collisions étant trop rares pour justifier deux opérateurs par ailleurs coûteux.

Algorithme d'extraction de racine carrée Dans le dessin ci-dessous, la surface de chaque rectangle rouge représente le poids de un bit. Seuls les bits à '1' sont dessinés. La surface totale est donc la somme pondérée de ces bits.

Le jeu consiste à trouver un carré de surface égale à un nombre donné, nombre dont la valeur est représentée par la surface d'un cercle bleu, en observant un bit de test (\leq ou $>$) et en cliquant des bits. Le coté de ce carré est alors la racine cherchée.



Extracteur de racine carrée On veut calculer $Q = \sqrt{A}$. Ceci est équivalent à $Q = A \div Q$. Donc si Q s'écrit sur n bits, A s'écrit avec $2n$ bits.

On va construire une suite $Q_n, Q_{n-1}, \dots, Q_2, Q_1, Q_0$ et une suite $R_{2n}, R_{2n-2}, \dots, R_4, R_2, R_0$ telles que l'invariant $A = Q_j * Q_j + R_{2j}$ soit respecté pour tout j .

La récurrence est :

- $Q_{j-1} = Q_j + q_{j-1} * 2^{j-1}$
- $R_{2j-2} = R_{2j} - q_{j-1} * 2^{j-1} * (2 * Q_j + 2^{j-1})$

avec comme conditions initiales :

- $Q_n = 0$
- $R_{2n} = A$.

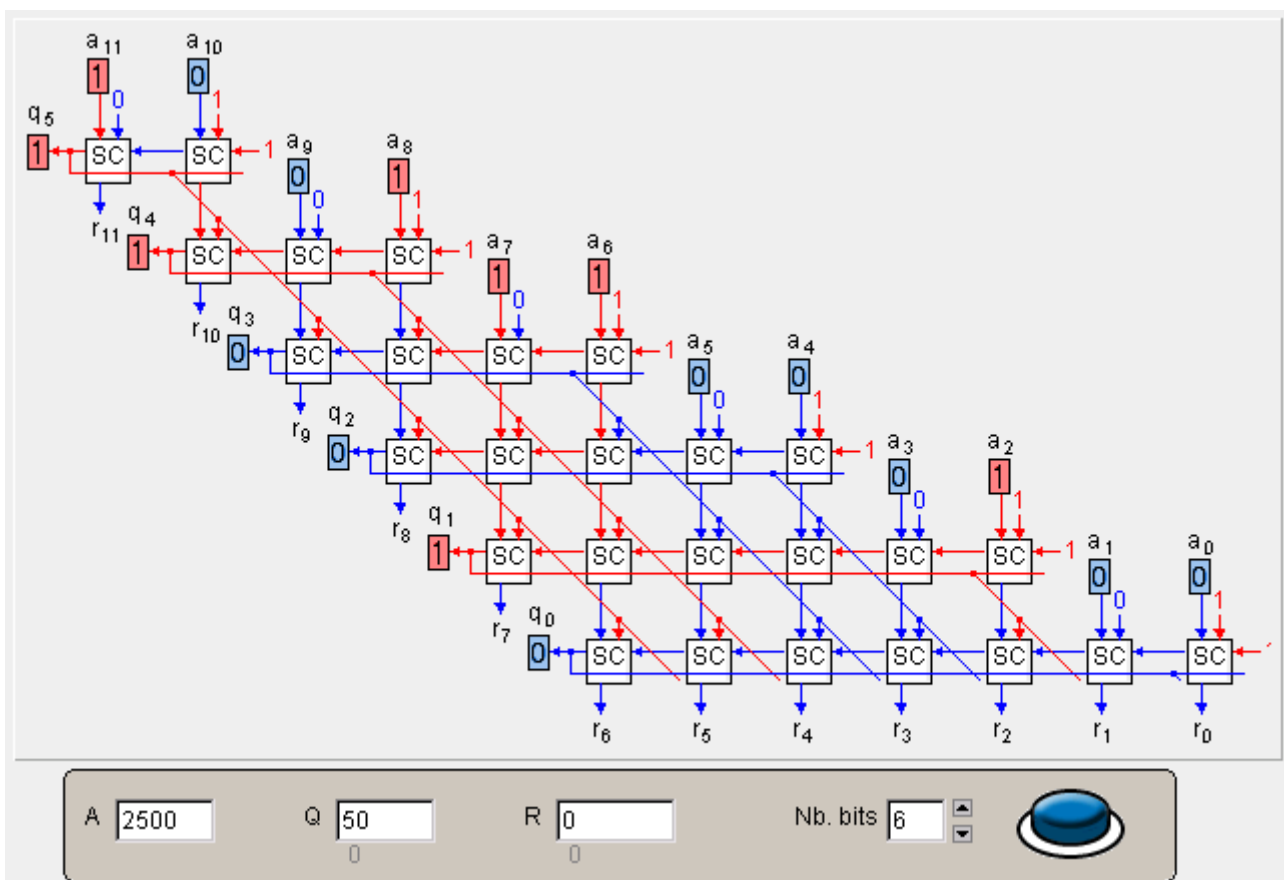
Quand la récurrence s'arrête, on a $Q = Q_0 = \sum_i q_i * 2^i$.

$R = R_0$ est le reste de l'extraction de racine.

	100111000100	
$R_{12} = A$	100111000100	= 2500
Soust. condition.	01	$q_5 = 1, Q_5 = 1$
$R_{10} = R_{12} - Q_5$	010111000100	= 2500 - 1024 = 1476
Soust. condition.	101	$q_4 = 1, Q_4 = 11$
$R_8 = R_{10} - Q_4$	000011000100	= 1476 - 1280 = 196
Soust. condition.	1101	$q_3 = 0, Q_3 = 110$
$R_6 = R_8$	000011000100	= 196
Soust. condition.	11001	$q_2 = 0, Q_2 = 1100$
$R_4 = R_6$	000011000100	= 196
Soust. condition.	110001	$q_1 = 1, Q_1 = 11001$
$R_2 = R_4 - Q_1$	000000000000	= 196 - 196 = 0
Soust. condition.	1100101	$q_0 = 0, Q_0 = 110010$
$R_0 = R_2$	000000000000	= 0
<hr/>		
Racine $Q_0 =$	110010	= 50
Reste $R_0 =$	0000000	= 0
$Q_0^2 + R_0 =$	$50 * 50 + 0 =$	2500 + 0 = 2500

A
 Avec restauration
 Sans restauration
 Nb. bits

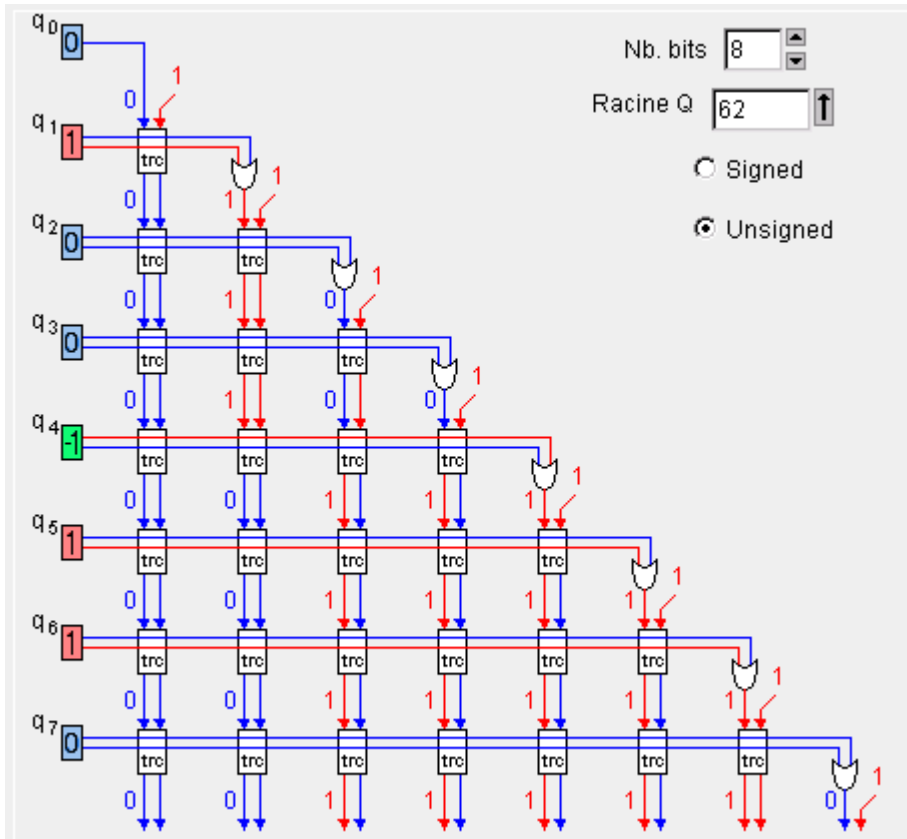
Réalisation L'extracteur de racine carrée avec restauration utilise les mêmes soustracteurs conditionnels "SC" que le diviseur avec restauration.



Extracteur de racine carrée On cherche à s'affranchir de la propagation de retenue en utilisant la notation "BS", les mêmes cellules "head" et "tail" et une architecture similaire à celle de la division

racine carrée rapide rapide. On se heurte à trois difficultés en cherchant à utiliser ce diviseur pour extraire des racines.

Convertisseur de racine La première difficulté est le rebouclage de la racine. Comme le diviseur, l'extracteur de racine carrée rapide fourni des racines partielles Q_j en notation "BS". Utiliser les cellules "head" et "tail" que le diviseur rapide exige une racine partielle en notation binaire conventionnelle. On pourrait utiliser un soustracteur pour la conversion de "BS" à binaire de chaque Q_j mais ce serait coûteux en temps et en circuit. Le convertisseur ci-dessous utilise une cellule "trc" à 4 entrées et 2 sorties dérivées de la cellule "BK"



Cellule du convertisseur de racine Vérifiez que vous maîtrisez les fonctions logiques de la cellule "trc" de conversion de notation "BS" vers notation binaire conventionnelle. L'entrée "si" est un bit du Q_j , l'entrée "ci" indique que la retenue se propage à la position de cette cellule. La retenue est utilisée pour la soustraction. Elle correspond au 'P' de la cellule "BK".

- si $q_j = -1$ alors { $so = si \oplus ci$; $co = 0$ } //soustraction (somme – retenue), retenue tuée
- si $q_j = 0$ alors { $so = si$; $co = ci$ } //somme inchangée, retenue propagée
- si $q_j = 1$ alors { $so = si$; $co = 0$ } //somme inchangée, retenue tuée

q_j	s_i	c_i	s_o	c_o
+1	0	0	0	0
+1	0	1	0	0
+1	1	0	1	0
+1	1	1	1	0
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	1	1
-1	0	0	0	0
-1	0	1	1	0
-1	1	0	1	0
-1	1	1	0	0

Voyez la table

Extracteur de racine carrée sans propagation de retenue

L'extracteur rapide utilise les mêmes cellules que le diviseur rapide pour effectuer à chaque itération une des opérations arithmétiques :

- si $q_j = '-1'$ alors $R_{2j-2} = R_{2j} + 2^j * Q_j - 2^{2j-1}$ // addition
- si $q_j = '0'$ alors $R_{2j-2} = R_{2j}$ // identité
- si $q_j = '1'$ alors $R_{2j-2} = R_{2j} - 2^j * Q_j - 2^{2j-1}$ //soustraction

Chaque cellule "head" détermine un q_j grâce au signe d'un approximant \hat{R}_{2j} du reste courant R_{2j} .

La deuxième difficulté par rapport à la division est dans la soustraction de 2^{2j-1} quand $q_j = -1$ ou $q_j = 1$. On utilise pour cette soustraction de un bit l'entrée négative de la cellule "tail" en poids faible de chaque ligne.

La troisième difficulté limite en fait le domaine d'utilisation. En effet tous les Q_j doivent commencer par un "1" en poids fort (implicite). Cette condition n'est pas réalisée si les deux premiers bits du radicande A sont nuls tous les deux. On soustrait ce "1" de A à la première ligne grâce à une entrée négative de "head".

Addition en Virgule Flottante

Format des nombres en virgule flottante

Trois champs composent l'écriture binaire des nombres en virgule flottante. Le signe S (1 bit), l'exposant E (8 bits) et la mantisse M, ou significande (23 bits).
 La valeur du nombre est $(-1)^S * 2^{(E - 127)} * (1 + M / 8388608)$. Cependant si $E = 0$, le nombre dénormalisé vaut $(-1)^S * 2^{(-126)} * (M / 8388608)$ et si $E = 255$, la valeur est infinie.

Ecriture de plus grand dénormalisé (1.1754942 E-38)

0.000 000 000 000 000 000 000 000 000 000 117 549 42

Réponse fausse, essayer encore une fois


Exposant-127 = 0

31 30 23 22

0 01111111 001000000000000000000000

Mantisse = 1 + (1/8)

0 01111111 001000000000000000000000



Valider 0

Addition et soustraction

Les réels étant codés en "signe/valeur-absolue", un seul bit permet de changer le signe d'un opérande. En conséquence le même opérateur effectue indifféremment l'addition ou la soustraction suivant les signes des opérandes. L'addition/soustraction de deux réels $S = A + B$ est plus complexe que la multiplication ou la division.

Elle se déroule en 4 étapes:

- Alignement des mantisses si les exposants de A et B sont différents
- Addition ou soustraction des mantisses alignées
- Renormalisation de la somme S si elle n'est pas normalisée

Arrondi de la somme S

L'alignement produit un bit de garde et un bit collant utilisés pour l'arrondi.

A 55

B 5.5

31 30 23 22 0

0 1000100 101110000000000000000000

31 30 23 22 0

0 1000001 011000000000000000000000

A = + 1.101110000000000000000000 * 2⁵ = 55.0

B = + 1.011000000000000000000000 * 2² = 5.5

1 - Alignement des mantisses de A et de B

A = + 1.101110000000000000000000000 * 2⁵ = 55.0 (A inchangé dans l'alignement)

B = + 0.001011000000000000000000000 * 2⁵ = 5.5 (B décalé de 3 positions à droite)

2 - Addition des mantisses alignées

S = + 0.111100100000000000000000000 * 2⁵ = 60.5

3 - Renormalisation de la mantisse de S

S = + 1.111001000000000000000000000 * 2⁵ = 60.5

4 - Arrondi de la mantisse de S

S = + 1.111001000000000000000000 * 2⁵ = 60.5

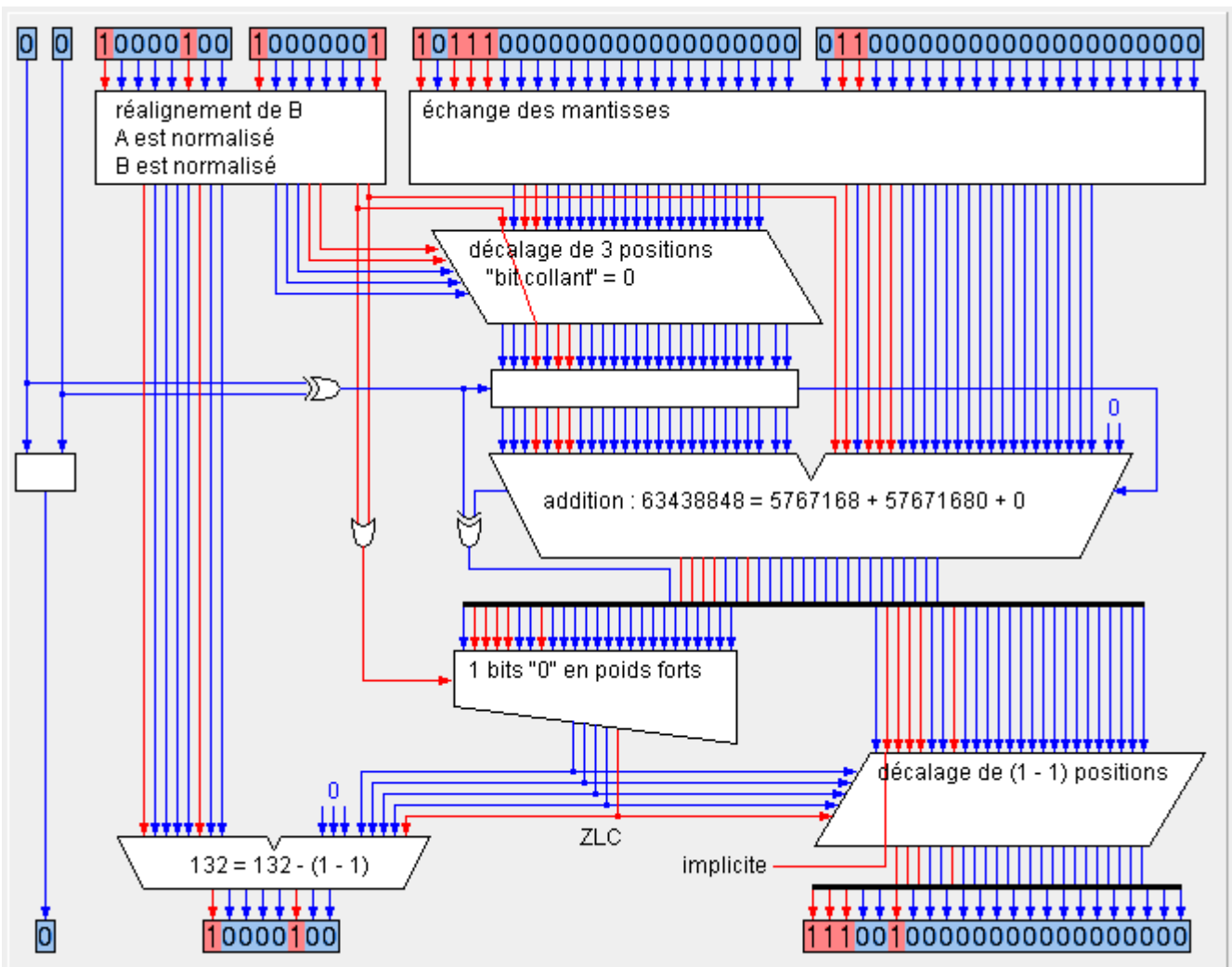
31 30 23 22 0

0 1000100 111001000000000000000000

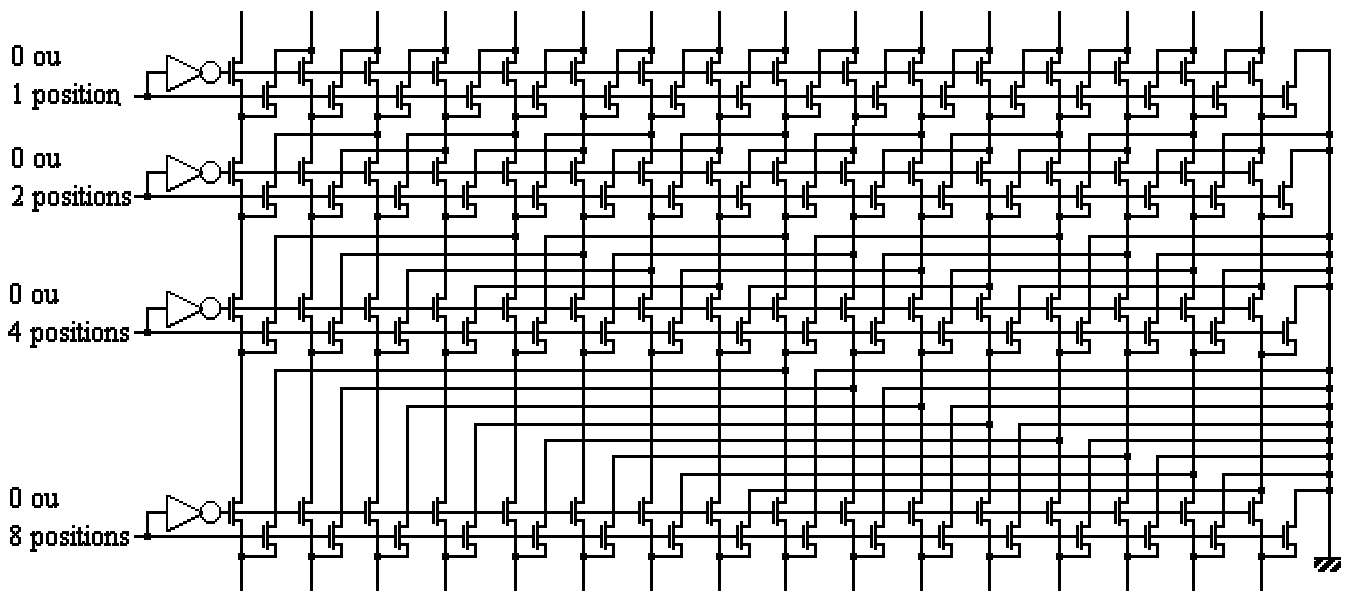
Additionneur/ soustracteur

Un additionneur flottant est formé des blocs ci-dessous:
 Bloc1: sort plus grand exposant (8 bits). sort la distance des exposants (5 bits). sort le

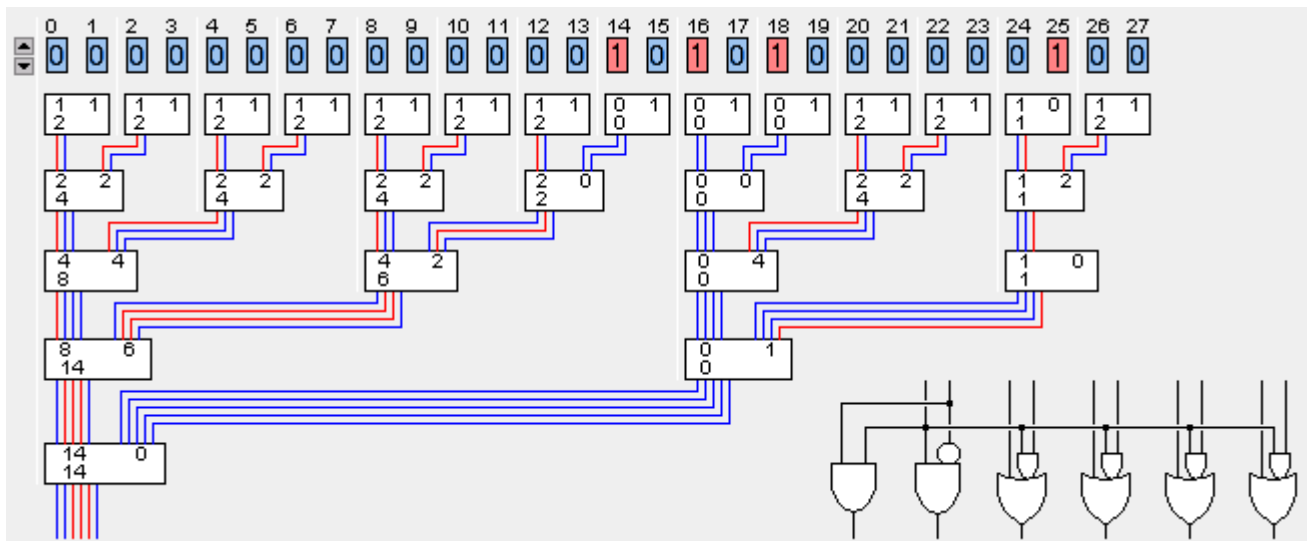
soustracteur bit implicite du plus petit opérande et le bit implicite du plus grand opérande.
 Bloc 2: sort à gauche la mantisse du plus petit opérande (23 bits), sort à droite la mantisse du plus grand opérande (23 bits).
 Décaleur 1: décale vers la droite la mantisse du plus petit, ajoute un bit de garde et un bit collant; total 26 bits.
 Complémenteur: fait sur commande le complément logique en vue d'une soustraction.
 Additionneur 1: additionne les deux entrées et la retenue, sort un résultat arrondi et une retenue.
 Compteur de zéros en tête: la sortie ZLC compte nombre de "0" si le résultat est normalisé et vaut "1" autrement.
 Décaleur 2: décale vers la gauche ($ZLC - 1$). Le bit sortant poids fort est perdu ("1" implicite).
 Additionneur 2: soustrait du plus grand exposant ($ZLC - 1$).



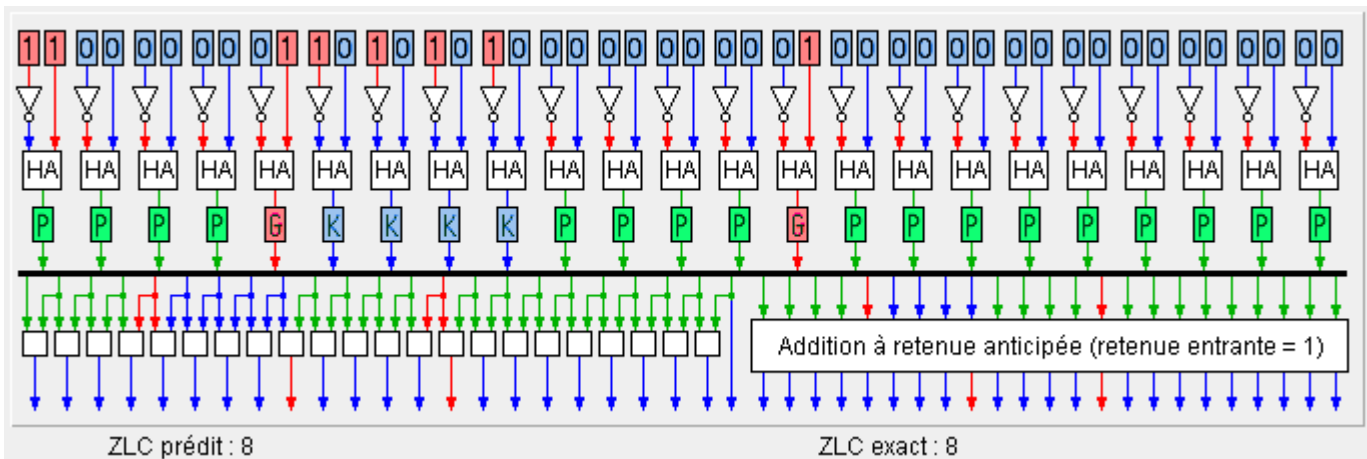
Additionneur de réels rapide Pour l'addition en virgule flottante, il faut effectuer des additions d'entiers, des décalages paramétrables (à droite pour l'alignement, à gauche pour la renormalisation) et un comptage des zéros en poids forts. Nous savons effectuer l'addition en temps $\log_2(n)$. Le décalage paramétrable est également en temps $\log_2(n)$.



Compteur de zéros en tête (ZLC) Un arbre binaire décompte le nombre de '0' en poids forts de la chaîne de bits S par dichotomie. Si la taille des sous-chaînes est une puissance de 2, alors il n'y a pas besoin d'additionneur mais seulement de multiplexeurs. En fait seule la taille de la sous-chaîne de gauche doit être une puissance de 2, la taille de la chaîne de droite doit simplement être inférieure ou égale à celle de gauche.



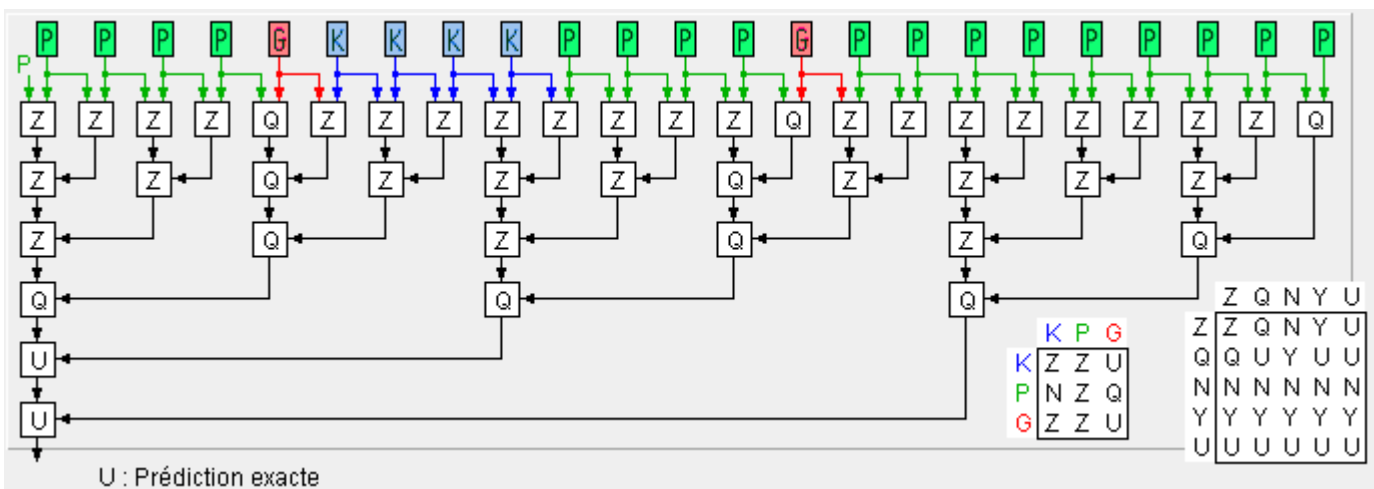
Prédiction de zéros en tête A partir de A et B on peut construire une chaîne de bits ayant (à 1 près) le même nombre de zéros en tête que le résultat de la soustraction $A - B$ sans effectuer la soustraction . En entrée d'un ZLC, cette chaîne prédit le nombre de décalage. Si le résultat du décalage a encore un zéro en tête, il faut le décaler de une position. Dans le cas contraire, ce résultat est normalisé.



La prédiction est exploitable si A est normalisé et B inférieur ou égal à A, ce qui est le cas dans une soustraction de mantisses. Alors les zéros en tête viennent de la séquence de retenue 'P'* 'G' 'K*', formée d'un certain nombre (éventuellement nul) de 'P' suivi de un seul 'G' puis d'un certain nombre de 'K'. Le prédicteur sort un '0' pour toute paire dans cette séquence: 'P' 'P' ; 'P' 'G' ; 'G' 'K' et 'K' 'K' et sort un '1' pour toute paire hors de la séquence.

Ce prédicteur ne tient pas compte de la propagation de retenue, d'où une possibilité de bits faussement prédits. Cependant seul le dernier bit dans la séquence 'P'* 'G' 'K*' peut être faussement prédit.

Ajustement de zéros en tête Ce circuit rend 'Y' si la prédiction du nombre de zéros en tête est fautive, donc trop petite de 1. La prédiction est fautive si la séquence de retenues commence par 'P'* 'G' 'K'* 'P'* 'K'.

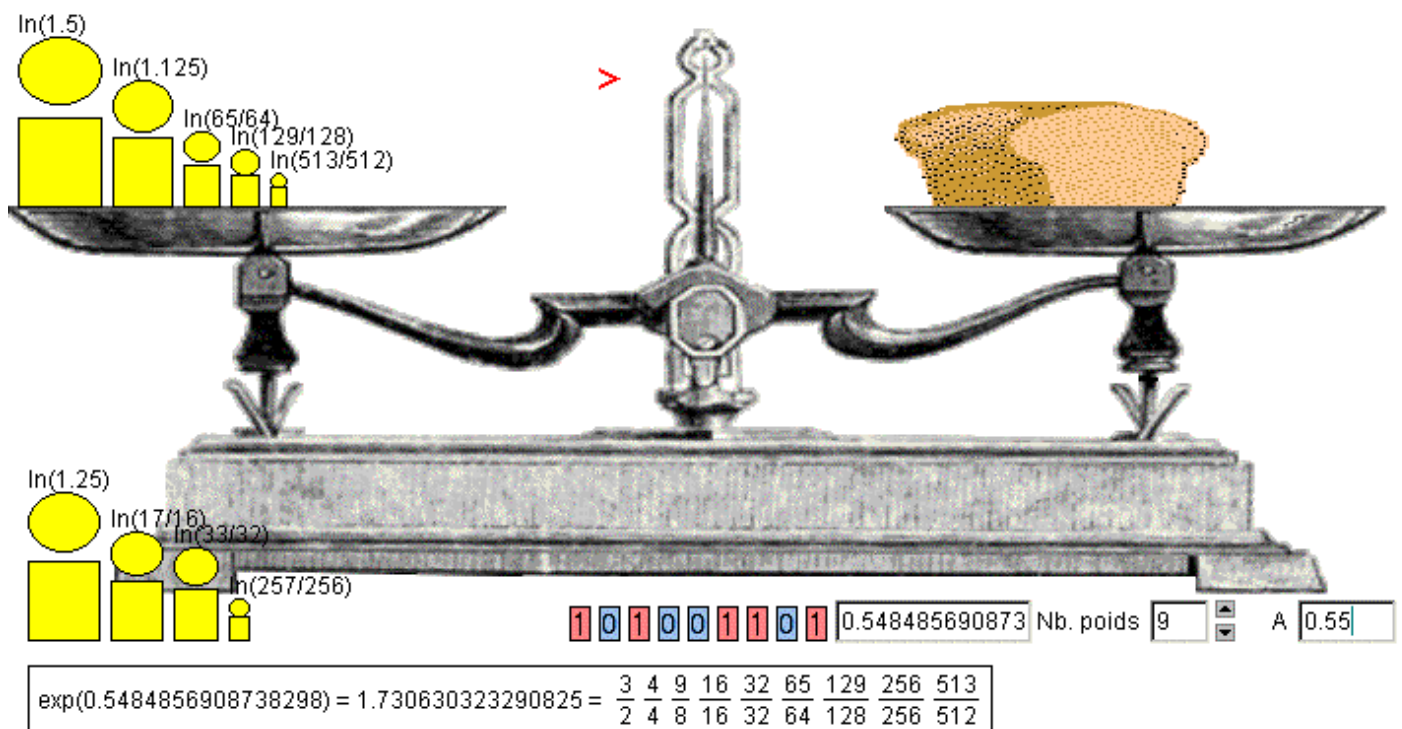


- Z indique une chaîne 'K'* 'P'*
- Q indique une chaîne 'P'* 'G' 'K'* 'P'*
- N indique une chaîne commençant par 'P'* 'K'
- Y indique une chaîne commençant par 'P'* 'G' 'K'* 'P'* 'K', c'est à dire Q suivi de N.
- U marque toute autre chaîne.

Fonctions élémentaires

Fonctions élémentaires Réaliser les fonctions Sinus, Cosinus, Arc tangente, Exponentielle et Logarithme avec des additions/soustractions et des décalages. Le décalage est de coût et délai nul si l'opérateur est câblé.

De la pesée du pain à l'exponentielle On veut calculer $\exp(A)$. On dispose d'une balance, d'un pain dont le poids est justement A et enfin d'une série de poids de valeur $\log(1 + 2^{-i})$. La pesée nous donne le résultat cherché sous forme d'un produit de rationnels $(2^i + 1) / 2^i$. Chaque multiplication se réduit à une addition et un décalage. Un poids sur le plateau de droite (celui du pain) change sa valeur en $-\log(1 - 2^{-i})$. Grâce à ça, peut peser avec restauration, sans restauration ou "SRT".



Diviseur sans propagation pour l'exponentielle On remplace la balance par un [diviseur "SRT"](#). Le produit de rationnels est donné par la concaténation du quotient (à gauche) et du reste final (en bas). En effet, pour les valeurs élevées de i , $\log(1 \pm 2^{-i})$ devient très proche de 1. Si le diviseur est très proche de 1, alors le reste devient une approximation acceptable du quotient.

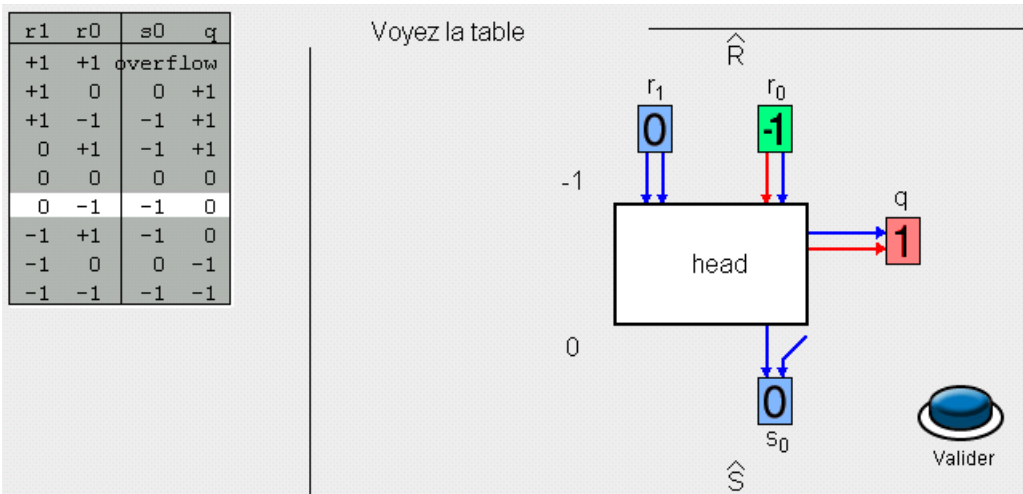
L'applet donne tous les restes successifs. Le dividande est dans] -1 , +1 [.

Conversion de notation La suite de multiplications conditionnelles par $(1 + 2^{-i})$ ou par $(1 + 2^{-i})$ nécessite une seule propagation de retenue finale grâce à des additionneurs en "CS" et des décalages câblés.

Les additions sont tronquées à $2n$ chiffres, dont 2 avant la virgule. Le troisième chiffre poids fort (à gauche) est le signe. Bien que les résultats partiels soient tous positifs ou nuls, exécuter des soustractions en "CS" entraîne un signe parfois non résolu. Le résultat final doit être traduit de "CS" en binaire par une addition (avec propagation).

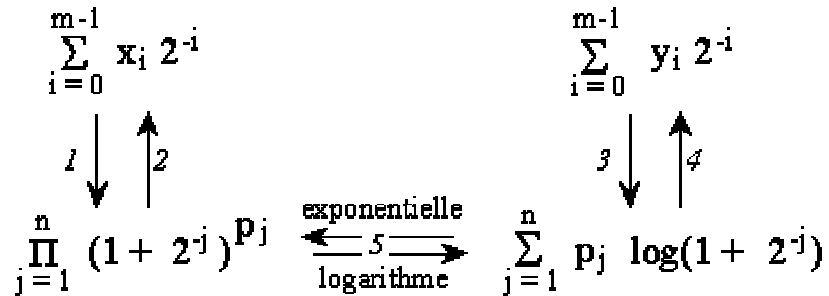
Opération d'une tranche de diviseur "SRT" pour l'exponentielle Chaque q_j est choisi par une cellule "head" en fonction de \hat{R}_j , somme pondérée des 2 chiffres poids forts r_1 et r_0 de l'écriture de R_j .

- si $\hat{R}_j > 0$ alors $\{ q_j = '1' ; s_0 = \hat{R}_j - 2 ; R_{j+1} = R_j + \log(1 - 2^{-j}) \}$ // soustraction
- si $\hat{R}_j = 0$ ou $\hat{R}_j = -1$ alors $\{ q_j = '0' ; s_0 = \hat{R}_j ; R_{j+1} = R_j + 0 \}$ // identité
- si $\hat{R}_j < -1$ alors $\{ q_j = '-1' ; s_0 = \hat{R}_j + 2 ; R_{j+1} = R_j + \log(1 + 2^{-j}) \}$ // addition



Logarithme et exponentielle

$$X = \prod_{j=1}^n (1 + 2^{-j})^{p_j} \iff \log(X) = \sum_{j=1}^n p_j \log(1 + 2^{-j})$$



Opérateur de calcul de Logarithme ou d'Exponentielle (c'est le même) avec des additions/soustraction (c'est la même opération), des décalages et des constantes. Les constantes sont ici $\log(1 + 2^{-i})$ et $-\log(1 - 2^{-i})$ et les chiffres $\in \{ '-1', '0', '1' \}$. Cette liberté dans le choix des chiffres, qui fait défaut pour Sinus et Cosinus, permet éventuellement de calculer plus vite avec des additions/soustractions sans propagation

Logarithme (X) Exponentielle (Y) Mise à zéro Nb. bits : 12

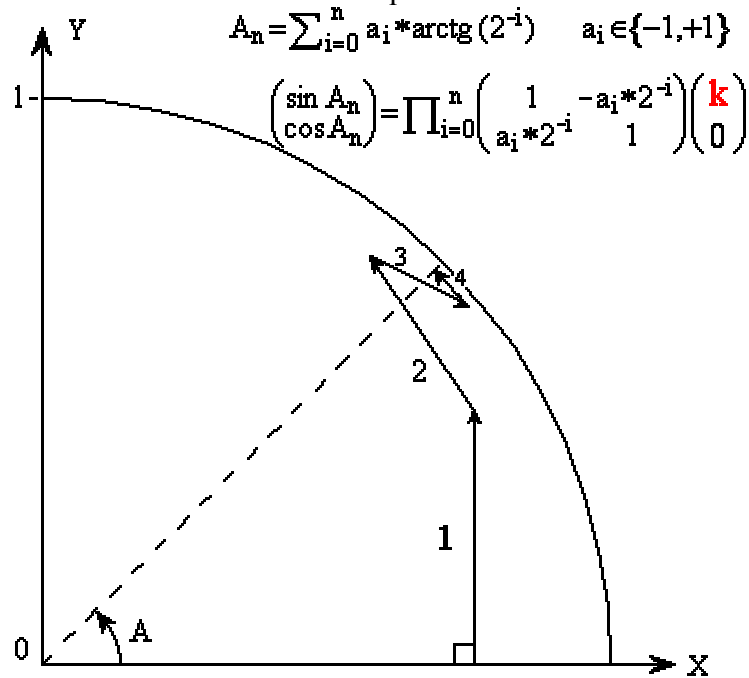
$X_0 = 0.852671$ $Y_0 = 0$
 $X_i \rightarrow 1$ $Y_i \rightarrow \log(X_0)$

↑	0.110110100100	0
0	=0.110110100100	=0.000000000000
0	+	-
1	=0.110110100100	=0.000000000000
0	+	-
2	=0.110110100100	=0.000000000000
0	+	-
3	=0.110110100100	=0.000000000000
1	+0.000110110101	-0.000111100010
4	=0.111101011010	=0.000111100010
0	+	-
5	=0.111101011010	=0.000111100010
1	+0.000001111011	-0.000001111110
6	=0.111110101010	=0.001001100000
0	+	-
7	=0.111110101010	=0.001001100000
1	+0.000000100000	-0.000000100000
8	=0.111111101010	=0.001010000000
0	+	-
9	=0.111111101010	=0.001010000000
1	+0.000000001000	-0.000000001000
10	=0.111111111010	=0.001010001000
0	+	-
11	=0.111111111010	=0.001010001000
1	+0.000000000010	-0.000000000010

$\log(X_0)$ trouvé = 0.0010100010100000 = **-0.15869140625**
 $\log(X_0)$ exacte = 0.0010100011001110 = **-0.15938150342898558**

Calcul de Sinus et Cosinus

Soit un vecteur V_i d'extrémité (x_i, y_i) . Une "pseudoRotation" de V_i d'un angle $\text{arctg}(2^{-i})$ donne V_{i+1} : $x_{i+1} = x_i - y_i * 2^{-i}$ et $y_{i+1} = y_i + x_i * 2^{-i}$. En décomposant un angle A en une somme pondérée d' $\text{arctg}(2^{-i})$, une suite de "pseudoRotations" calcule les coordonnées du vecteur d'angle A qui sont les valeurs $\sin(A)$ et $\cos(A)$ cherchées. Pour les "pseudoRotations" on n'a effectué que des additions/soustractions.

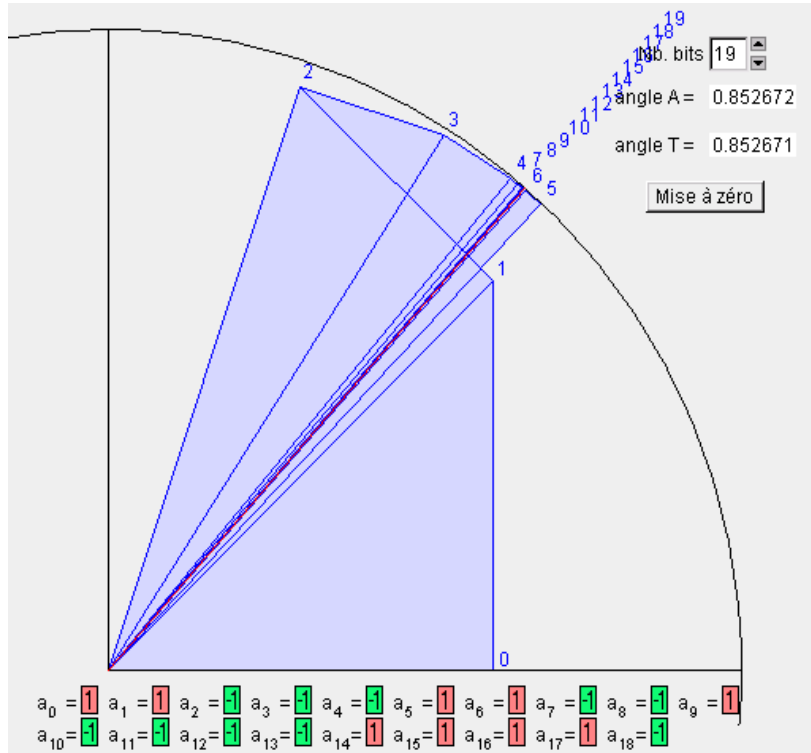


La constante k Chaque "pseudoRotation" de $\text{arctg}(2^{-i})$ entraîne un allongement du vecteur de $\sqrt{1+2^{-2i}}$, car ce n'est pas une rotation mais un déplacement de l'extrémité du vecteur sur un vecteur perpendiculaire. Pour compenser par avance le produit des allongements d'une suite de "pseudoRotations", le vecteur de départ est $(x_0 = k, y_0 = 0)$. Pour n assez grand, k vaut environ 0,60725. Pour que k soit une constante, l'écriture dans la base $\text{arctg}(2^{-i})$ utilise des chiffres $\in \{-1, +1\}$.

$$k = \frac{1}{\prod_{i=0}^n \sqrt{1+2^{-2i}}}$$

Décomposition d'un angle

Quel est le domaine des angles $A = \sum_{i=0}^n a_i \text{arctg}(2^{-i})$ et quelle précision espérer de cette écriture ? L'angle A est la valeur cherchée et l'angle T la valeur atteinte par la suite de "microRotations". Cliquer dans la figure pour changer A . Les valeurs sont affichées en radian. La touche "Mise à zéro" laisse faire 'à la main' la conversion de A dans la base $\text{arctg}(2^{-i})$.



Application numérique Le "Nombre de bits" fixe à la fois la précision des calculs et le nombre de pas. En cliquant flèche verticale ↑ on change la présentation. La touche "Mise à zéro" permet le contrôle 'à la main' de la convergence.

Sin(A) Cos(A) Arctg(Y) Mise à zéro Nb. bits : 12

$A_0 = 0.852671$ $X_0 = k$ (câblé) $Y_0 = 0$
 $A_i \rightarrow 0$ $X_i \rightarrow \cos(A_0)$ $Y_i \rightarrow \sin(A_0)$

↑	0.110110100100	0.100110110111	
0	=0.110110100100	=0.100110110111	0.000000000000
1	+0.110010010001	-0.000000000000	0.100110110111
1	=0.000100010100	=0.100110110111	0.100110110111
2	+0.011101101011	-0.010011011100	0.010011011100
2	=0.011001010111	=0.010011011011	0.111010010011
3	+0.001111101011	-0.001110100101	0.000100110111
3	=0.001001101100	=0.100010000000	0.110101011100
4	+0.000111111101	-0.000110101100	0.000100010000
4	=0.000001101111	=0.101000101100	0.110001001100
5	+0.000100000000	-0.000011000101	0.000010100011
5	=0.000010010001	=0.101011110001	0.101110101001
6	+0.000010000000	-0.000001011101	0.000001011000
6	=0.000000010001	=0.101010010100	0.110000000001
7	+0.000001000000	-0.000000110000	0.000000101010
7	=0.000000010111	=0.101001100100	0.110000101011
8	+0.000000010000	-0.000000011000	0.000000010101
8	=0.000000001111	=0.101001111100	0.110000010110
9	+0.000000001000	-0.000000001100	0.000000001010
9	=0.000000000001	=0.101010001000	0.110000001100
10	+0.000000000100	-0.000000000110	0.000000000101
10	=0.000000000011	=0.101010000010	0.110000010001
11	+0.000000000010	-0.000000000011	0.000000000011
11	=0.000000000011	=0.101010000101	0.110000001110
11	+0.000000000010	-0.000000000010	0.000000000001

$\cos(A_0)$ trouvé = $X_{12} = 0.1100000011010000 = 0.753173828125$
 $\cos(A_0)$ exacte = $0.110000001100011 = 0.7530405381207532$
 $\sin(A_0)$ trouvé = $Y_{12} = 0.1010100001110000 = 0.657958984375$
 $\sin(A_0)$ exacte = $0.1010100001110000 = 0.6579741240708531$