Op Art : tendance qui au sein de l'art cinétique privilégie les effets optiques générateurs d'illusion de mouvement (Larousse).

Version préliminaire

Portée:
De la porte logique aux opérateurs arithmétiques combinatoires.

Pédagogie:
Expérimenter, comprendre, améliorer
complément de notes de cours.

Méthode:
Animer les figures du cours :
Synthèse, simulation, diagramme, algorithme .

Chapitre 1 : *Full-Adder en CMOS*
Chapitre 2 : *Additionneurs*
Chapitre 3 : *Multiplieurs* suivi de *Multiplieurs en "CS"*
Chapitre 4 : *Diviseurs* suivi de *Diviseurs rapides*
Chapitre 5 : *Extracteurs de racine carrée* suivi de *Extracteurs rapides*
Chapitre 6 : Addition *Virgule flottante*
Chapitre 7 : Fonctions élémentaires *exponentielle et logarithme* suivies de *sinus, cosinus et arc tangente*
Chapitre 8 : *Représentation modulaire*
Tous ce cours est imprimable.

Fonction "FA" en CMOS

Technologie CMOS

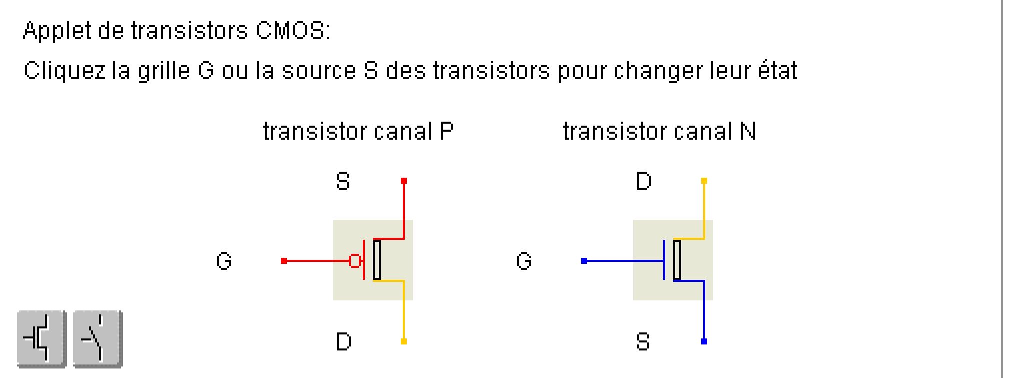
En technologie CMOS *(Métal Oxyde Semiconducteur complémentaire)* on utilise des transistors canal N et des transistors canal P pour réaliser des fonctions logiques. La technologie CMOS est actuellement la technologie dominante du marché. Son principal intérêt par rapport à d'autres technologies comme 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**.

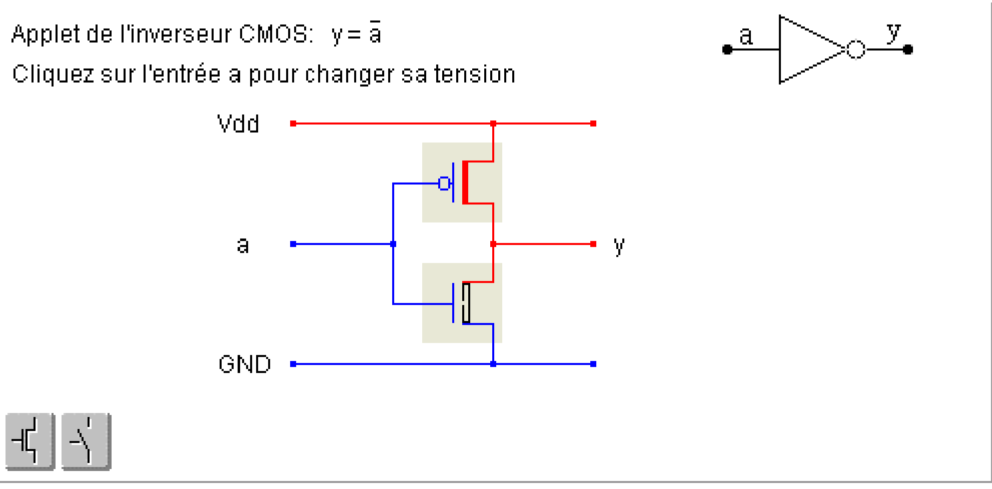


Remarquez que le transistor canal N conduit quand sa grille est au '1' logique, et que le transistor canal P conduit quand sa grille est au '0' logique. Les touches permettent de changer le dessin des transistors.

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'entré 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'.



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 (inévitable 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'entré Cg 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 $Q = Cg * Vdd$). 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 $I = dQ/dt = (Cg * Vdd)/(t_2-t_1)$.

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 :

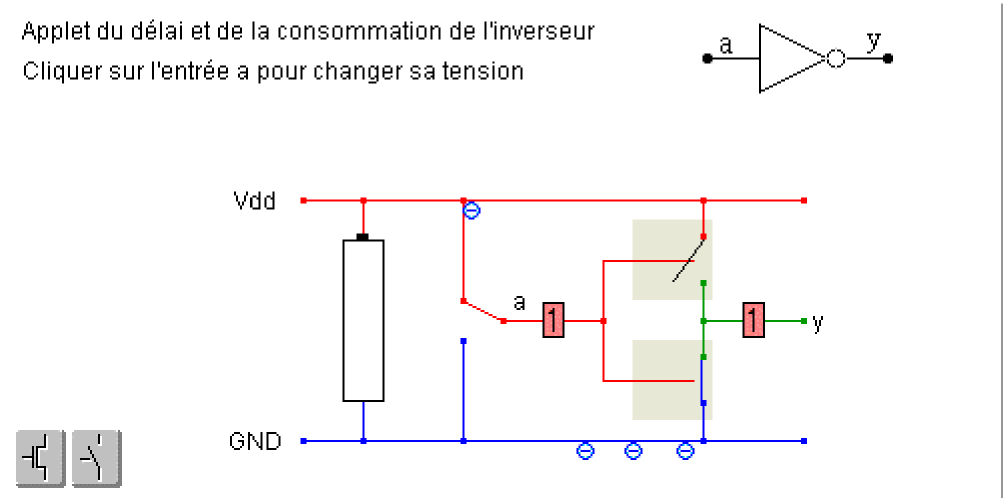
- Un microprocesseur moderne peut contenir 50 millions de transistors, soit environ dix millions de portes. A chaque cycle environ 1% de ces portes commutent.
- Les fréquences d'horloge atteignent 500 MHz (temps de cycle 2 ns) avec une tension d'alimentation Vdd = 3.3V.
- Les fis connectant les portes ont une capacité parasite *Cw* bien plus grande que la capacité de grille *Cg* des portes. Chaque fois qu'une équipotentielle commute, toutes les capacités qui lui sont attachées doivent être chargées ou déchargées : $Ctotal=Cg+Cw$.
- 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 :
$I = (portes\ actives) * (Ctotal*Vdd) / dt = (1\% * 1.000.000) * (1pF*3.3V) / 2ns = 16\ A$

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 ⊖ 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.

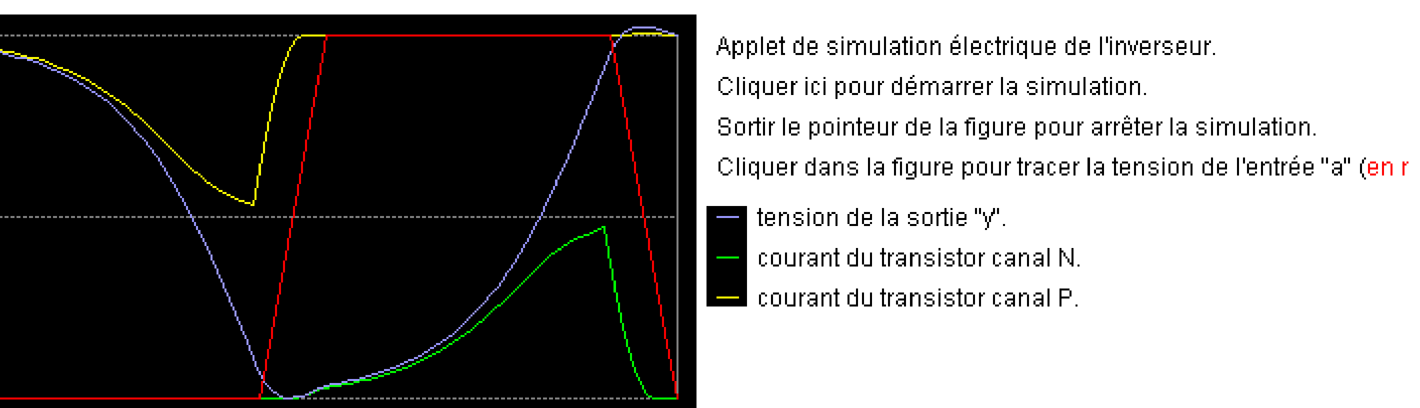
- Si l'entrée a commute, les grilles des deux transistors doivent être chargées ou déchargées. Ceci est illustré par le passage d'un "électron" ⊖ (de charge négative) venant de GND ou bien allant à Vss..
- Quand l'entrée commute, elle passe par des tensions faisant conduire les deux transistors pendant 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".



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.



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.

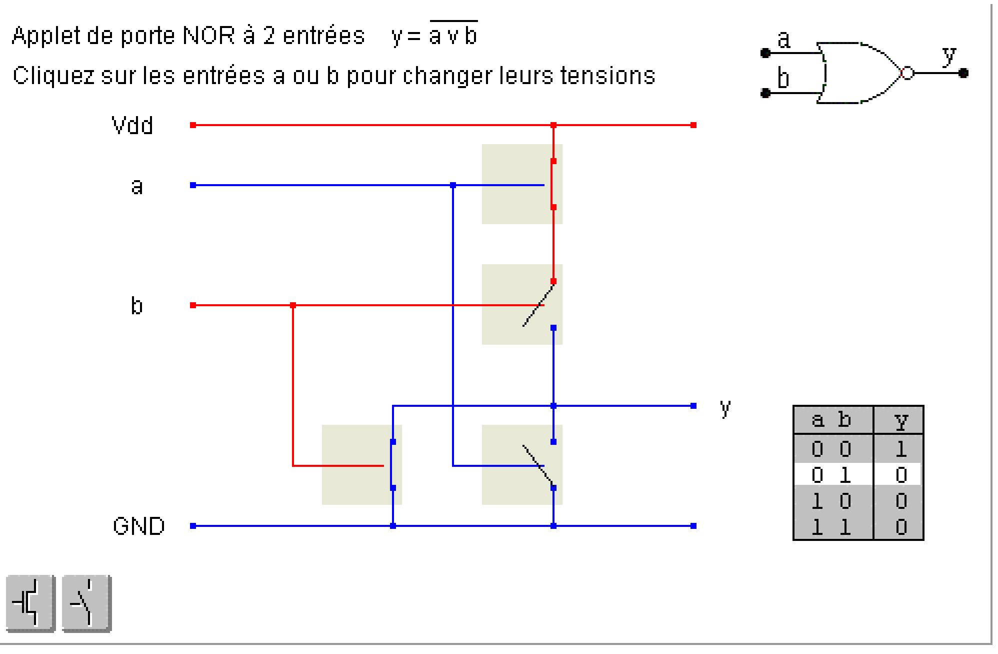
- Cliquer sur le haut de la table reconstruit progressivement cette table.

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

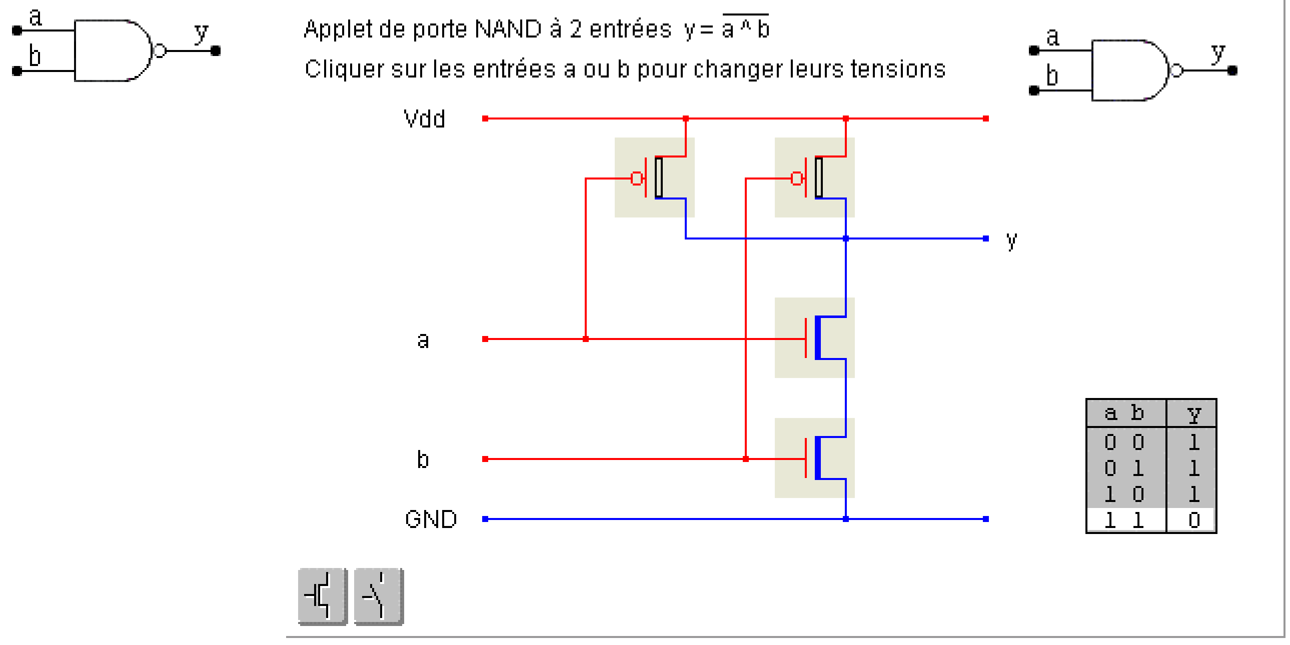
La porte CMOS à 2 entées est l'une des portes les plus simples pour illustrer le qualificatif *complémentaire*: les transistors canal P sont connectés en série alors que les transistors canal N sont connectés en parallèle. Les réseaux de transistors canal N et de transistors canal P sont complémentaires.

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.



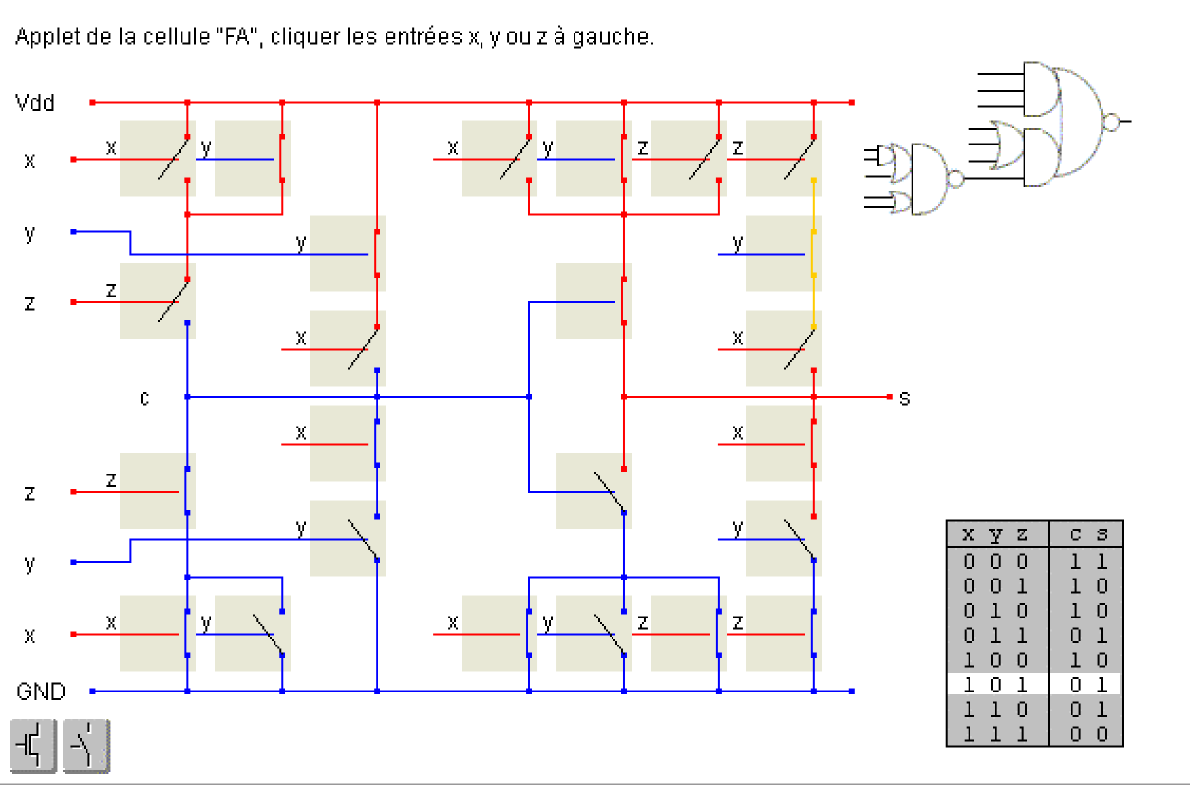
La porte NAND à 2 entrées

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



L'additionneur binaire

La cellule "FA" (Full Adder) 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.

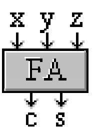


x y z	c s
0 0 0	1 1
0 0 1	1 0
0 1 0	1 0
0 1 1	0 1
1 0 0	1 0
1 0 1	0 1
1 1 0	0 1
1 1 1	0 0

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

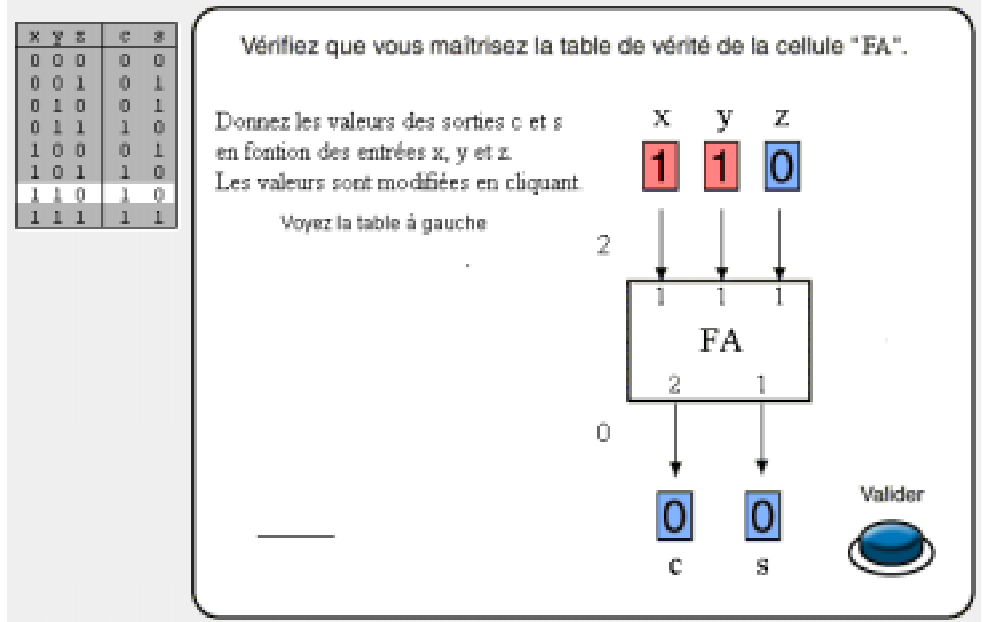
Cellule "FA"



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*.

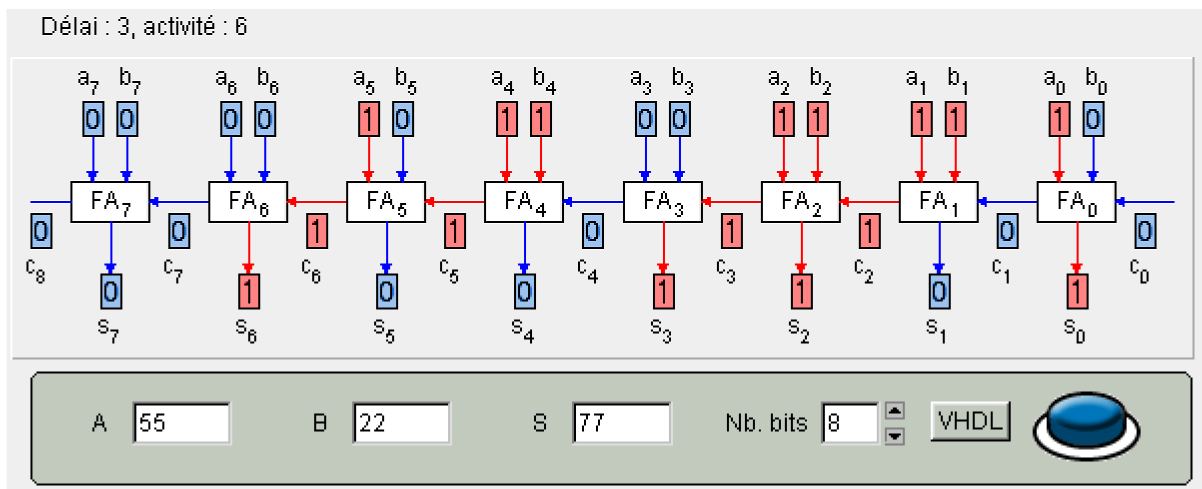
La cellule "FA" est également appelée "réducteur 3 ⇒2" car elle réduit le nombre de bits de 3 à 2 tout en conservant la *valeur numérique*.



x	y	z	c	s
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

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:

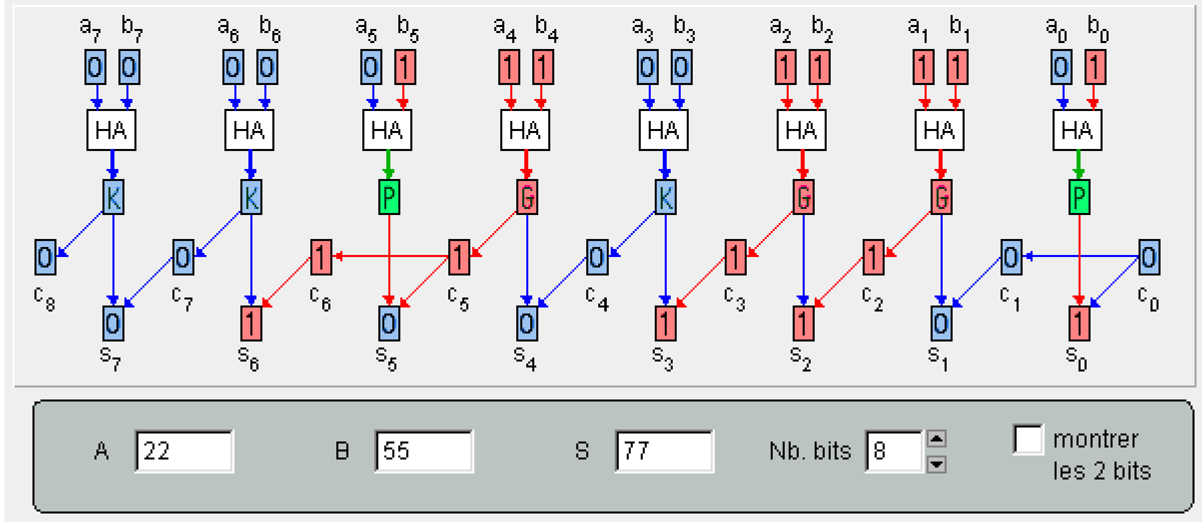
	minimum	moyen	maximum
délai	0	$\log_2(n)$	n
activité	0	3n / 4	$n^2 / 2$

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 :

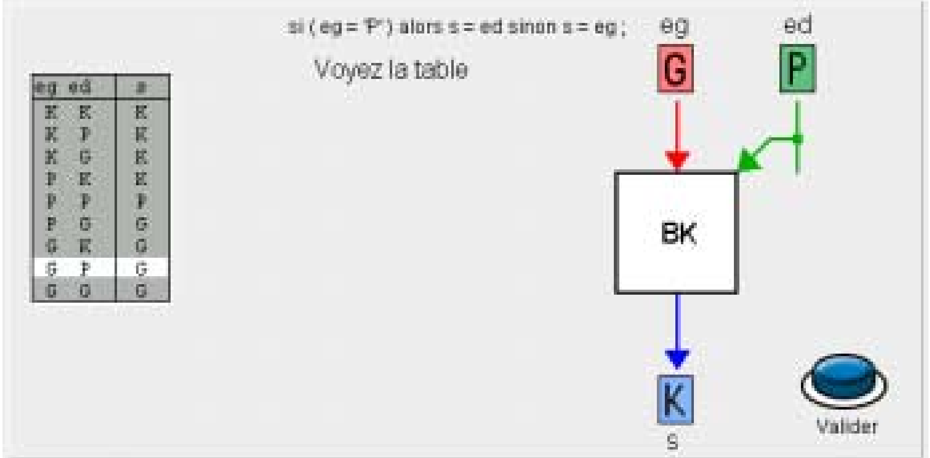
- la retenue c_{i+1} est mise à zéro, cas noté 'K', si $a_i = 0$ et $b_i = 0$
- la retenue c_{i+1} est mise à un, cas noté 'G', si $a_i = 1$ et $b_i = 1$
- la retenue c_{i+1} est propagée, cas noté 'P', si ($a_i = 0$ et $b_i = 1$) ou ($a_i = 1$ et $b_i = 0$). Dans ce cas $c_{i+1} = c_i$. Ce dernier cas,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



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

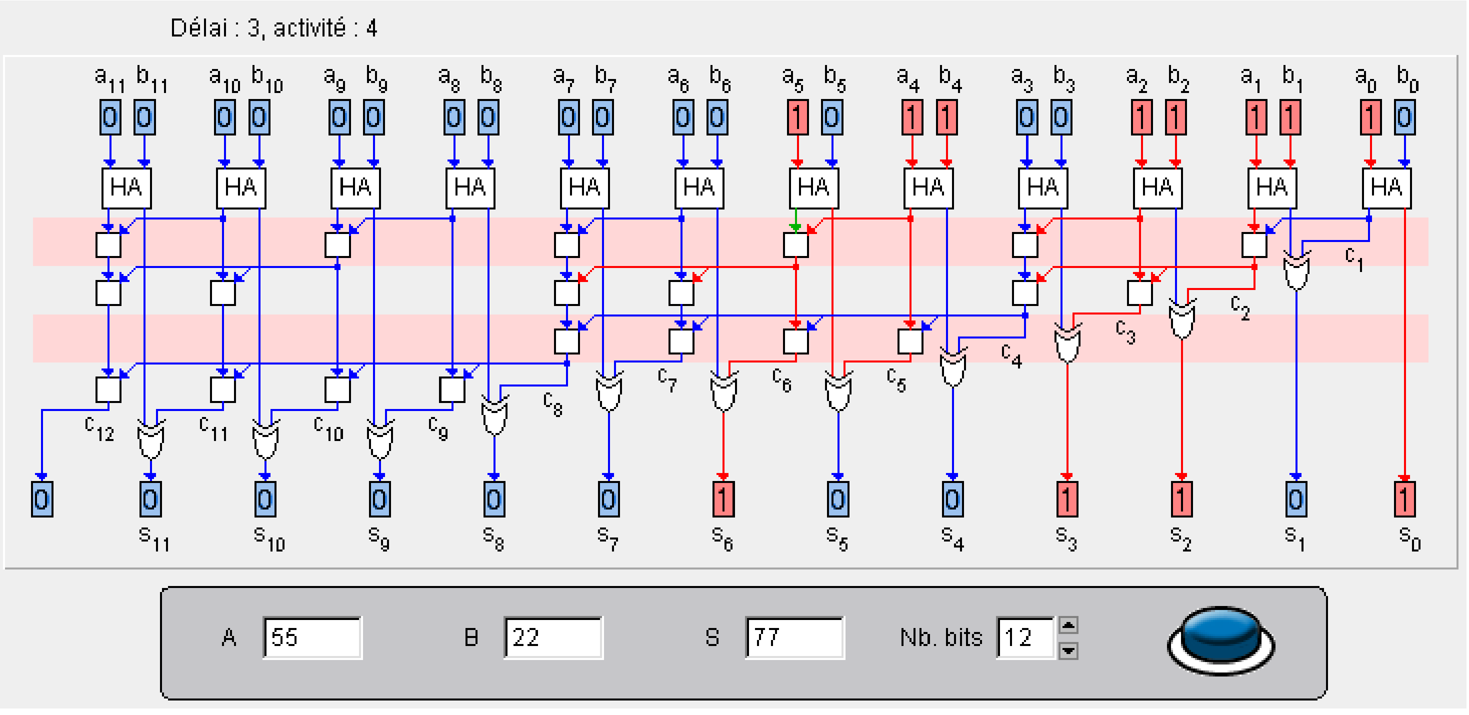
Cellule "BK" (Brent et Kung)

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



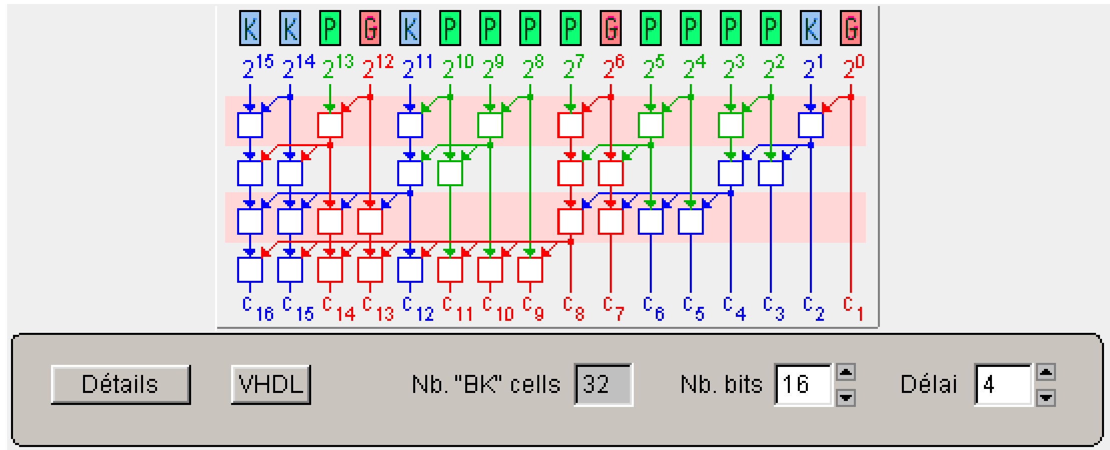
Additionneur de Sklansky

Pour concevoir un additionneur rapide, on va calculer <u>toutes les retenues</u> 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 toujours la meilleure.



Additionneurs rapides (Brent et Kung)

Il y a <u>une seule règle</u> de construction des arbres imbriqués de cellules "BK":
Toute sortie de rang i est reliée à <u>toutes les entrées</u> 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.

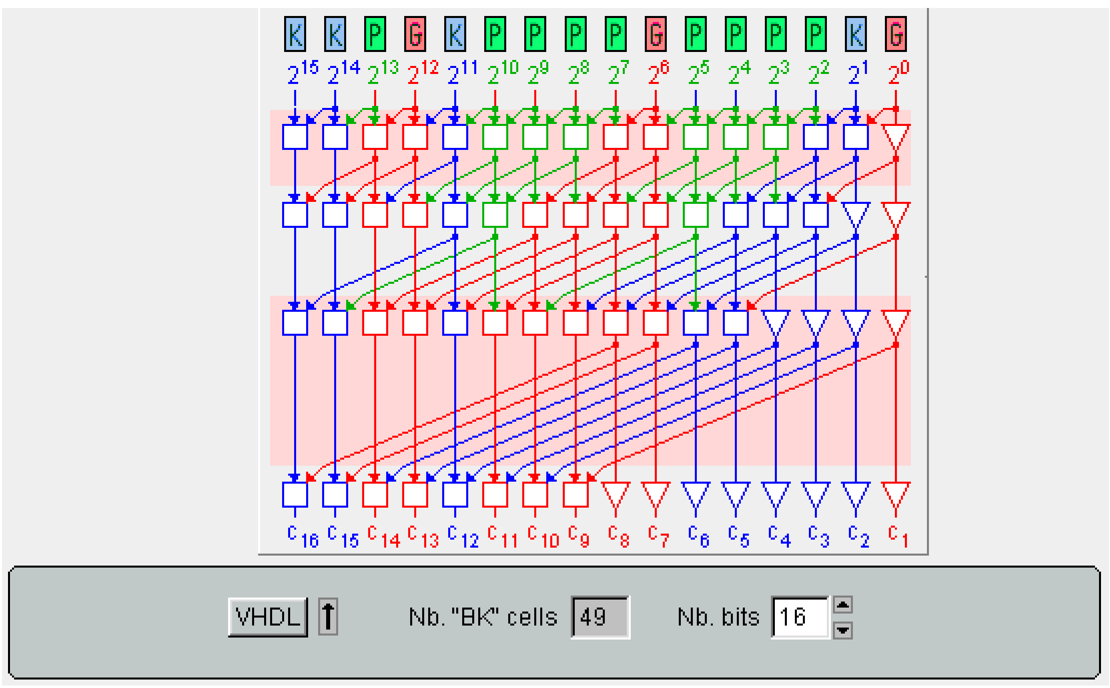


Dans les arbres de cellules "BK" on peut troquer des cellules "BK" contre du délai: moins il y a de délai (cellules "BK" à traverser) de calcul , plus l'additionneur utilise de cellules.

- 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

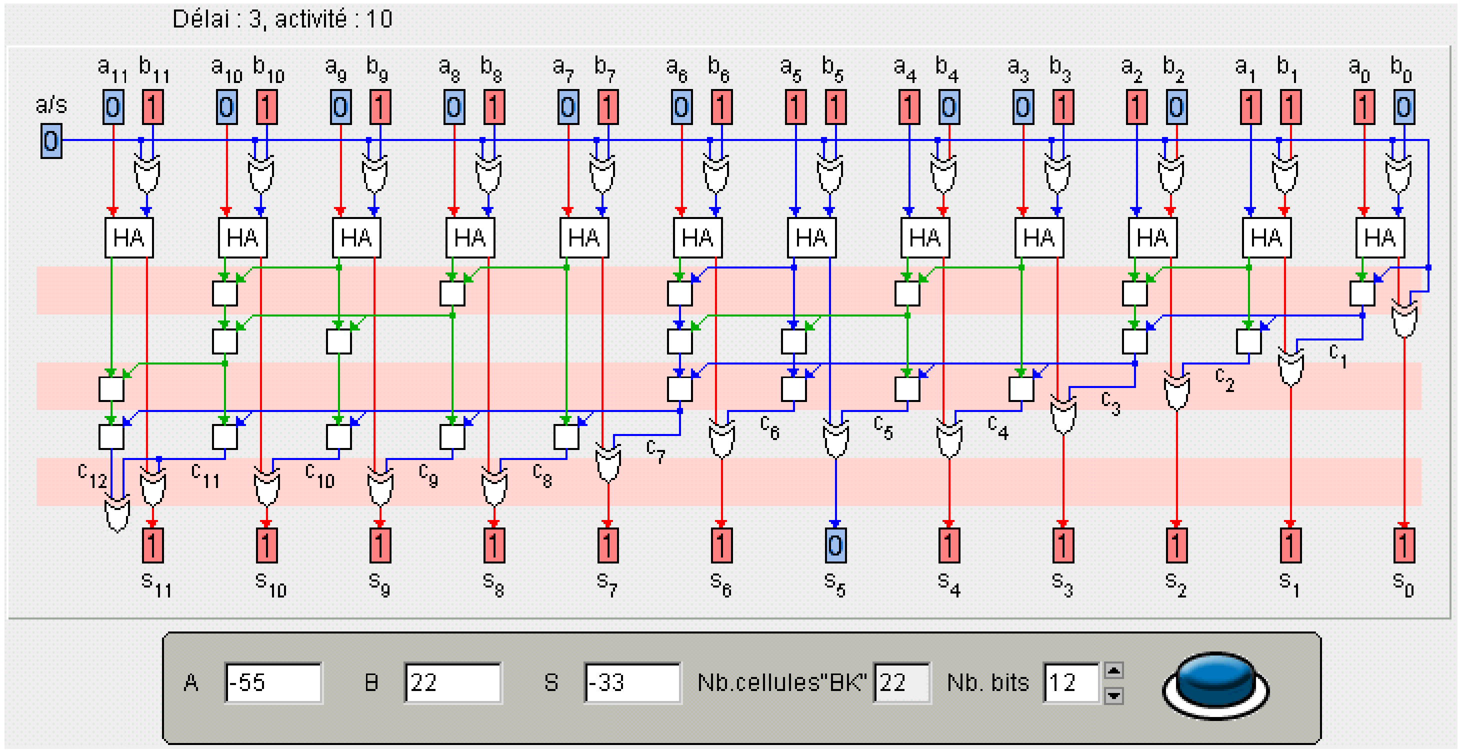
récupérer la description VHDL dans un éditeur de texte, la copier puis coller.

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 le la sortance, il est un peu plus court.

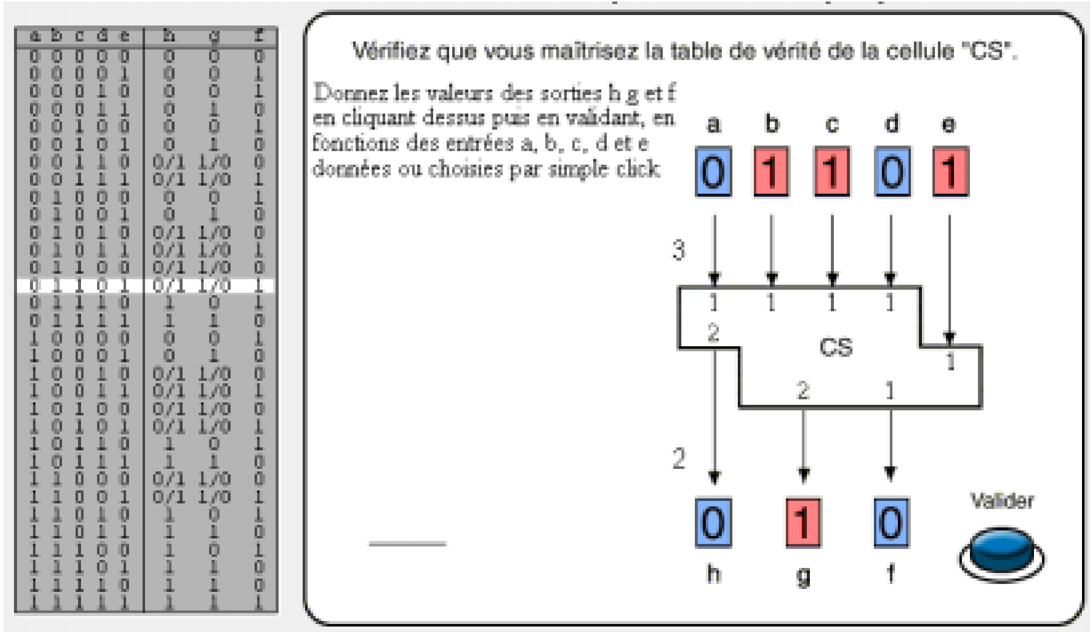


Additionneurs de Ling Dans un additionneur de Ling, les arbres de "BK" calculent une primitive appelée "pseudo retenue". Ceci permet d'éviter le calcul des p_i et g_i, mais impose en revanche un calcul supplémentaire pour obtenir la retenue à partie de la "pseudo retenue". L'astuce est que le délai de 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.

Additionneur/ soustracteur Les additionneurs font spontanément les additions modulo 2^n. Pour obtenir l'opposé -B d'un nombre B, on complémente logiquement tous ses bits puis on lui ajoute 1. L'addition de B et de -B obtenu ainsi donne 2^n c'est à dire 0. On se sert de la retenue entrante c_0 d'un additionneur pour ajouter 1 et de disjonctions pour complémenter chaque bits de B.

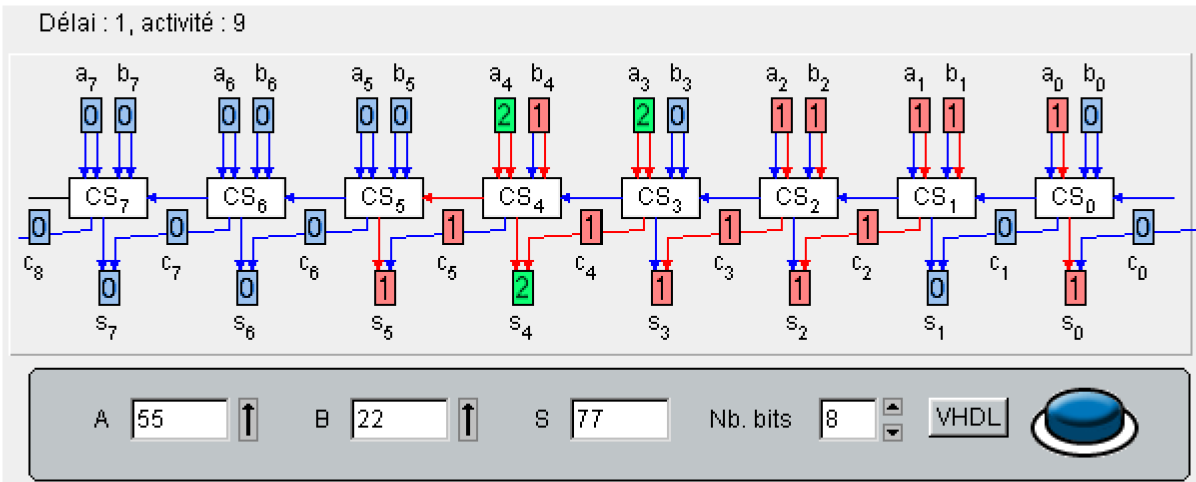


Cellule "CS" Dans la cellule "CS", la somme pondérée des sorties est égale à la somme pondérée des entrées c'est à dire que "a + b + c + d + e = 2*h + 2*g + f ". La cellule "CS" n'est pas seulement un "réducteur 5⇒3", car en plus la sortie "h" ne dépend jamais de l'entrée "e".



a	b	c	d	e	h	g	f
0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	1
0	0	0	1	0	0	0	1
0	0	0	1	1	0	1	0
0	0	1	0	0	0	0	1
0	0	1	0	1	0	1	0
0	0	1	1	0	0/1	1/0	0
0	0	1	1	1	0/1	1/0	1
0	1	0	0	0	0	0	1
0	1	0	0	1	0	1	0
0	1	0	1	0	0/1	1/0	0
0	1	0	1	1	0/1	1/0	1
0	1	1	0	0	0/1	1/0	0
0	1	1	0	1	0/1	1/0	1
0	1	1	1	0	1	0	1
0	1	1	1	1	1	1	0
1	0	0	0	0	0	0	1
1	0	0	0	1	0	1	0
1	0	0	1	0	0/1	1/0	0
1	0	0	1	1	0/1	1/0	1
1	0	1	0	0	0/1	1/0	0
1	0	1	0	1	0/1	1/0	1
1	0	1	1	0	1	0	1
1	0	1	1	1	1	1	0
1	1	0	0	0	0/1	1/0	0
1	1	0	0	1	0/1	1/0	1
1	1	0	1	0	1	0	1
1	1	0	1	1	1	1	0
1	1	1	0	0	1	0	1
1	1	1	0	1	1	1	0
1	1	1	1	0	1	1	0
1	1	1	1	1	1	1	1

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 déterminé 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' .



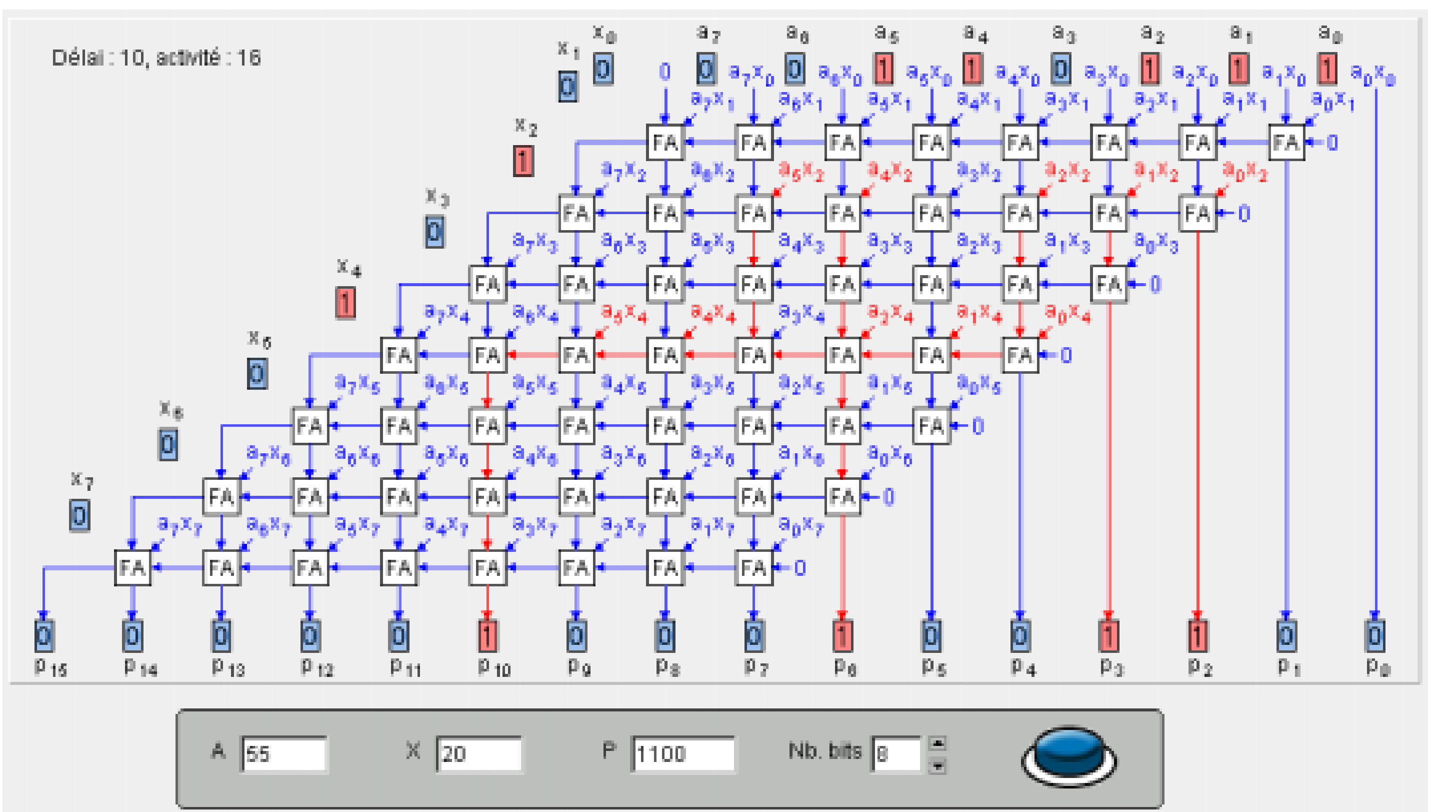
Ce système d'écriture des nombres entiers permet l'addition en un temps court et indépendant du nombre de chiffre des nombres.
Cependant il utilise deux fois plus de bits que la notation standard. En conséquence une même valeur a plusieurs écritures. La flèche verticale à côté des nombres en change l'écriture 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 multiplient 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.

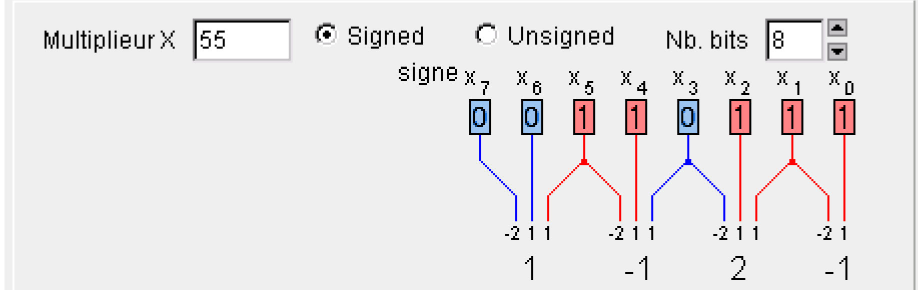


Multiplieurs rapides Plusieurs pistes conduisent à une amélioration de la vitesse:

- 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 utilise 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 ∈ {-2, -1, -0, 0, 1, 2}. Le

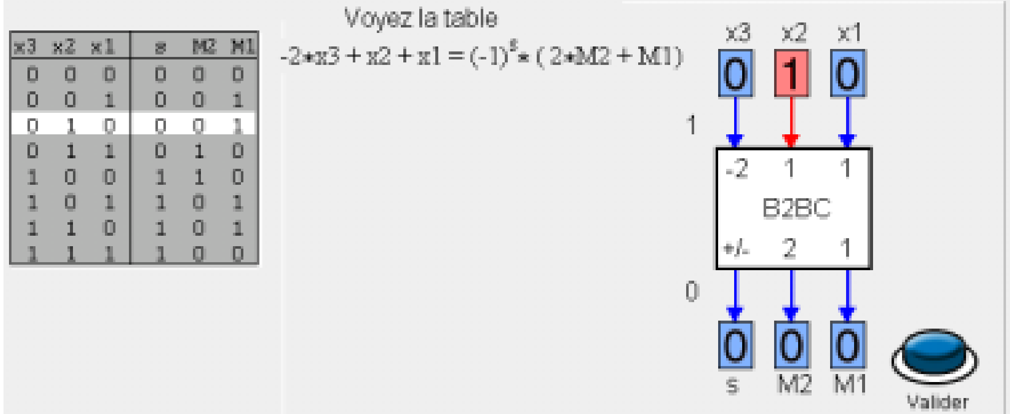
code sur 3 bits adopté ici, en "signe/valeur absolue", donne 2 notations pour le zéro.



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

Cellule du convertisseur de "BC"

Vérifiez que vous maîtrisez les fonctions logiques de la cellule "B2BC" qui convertit un chiffre "BC" en "signe/valeur absolue" pour le calcul de produits partiels. La somme est conservée:

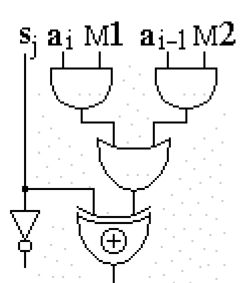


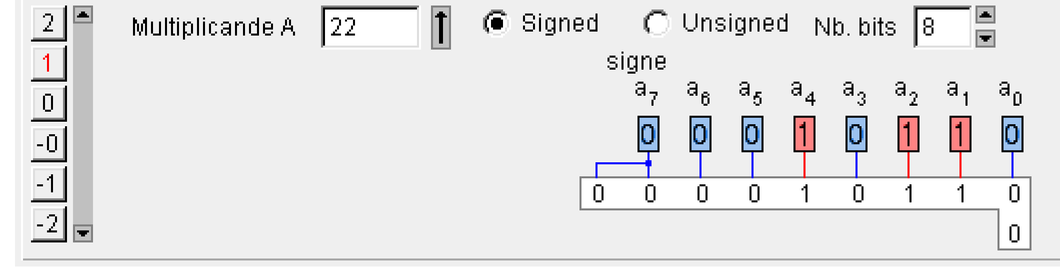
x3	x2	x1	s	M2	M1
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	0	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	0	1
1	1	0	1	0	1
1	1	1	1	0	0

La conversion de X requiert deux fois moins de cellules "B2BC" que de bits de X.

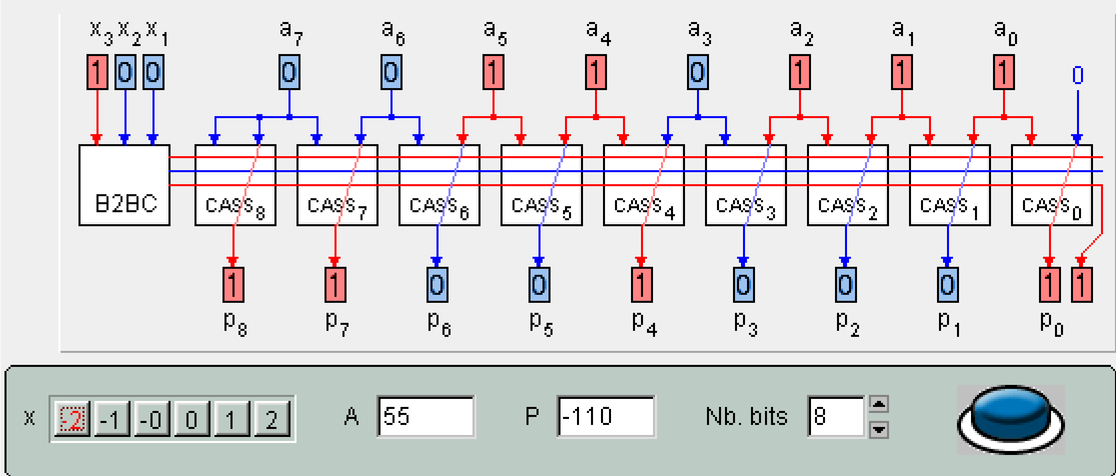
Multiplieur des bits de A par un chiffre "BC"

La multiplication par un chiffre "BC" ∈ {-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.

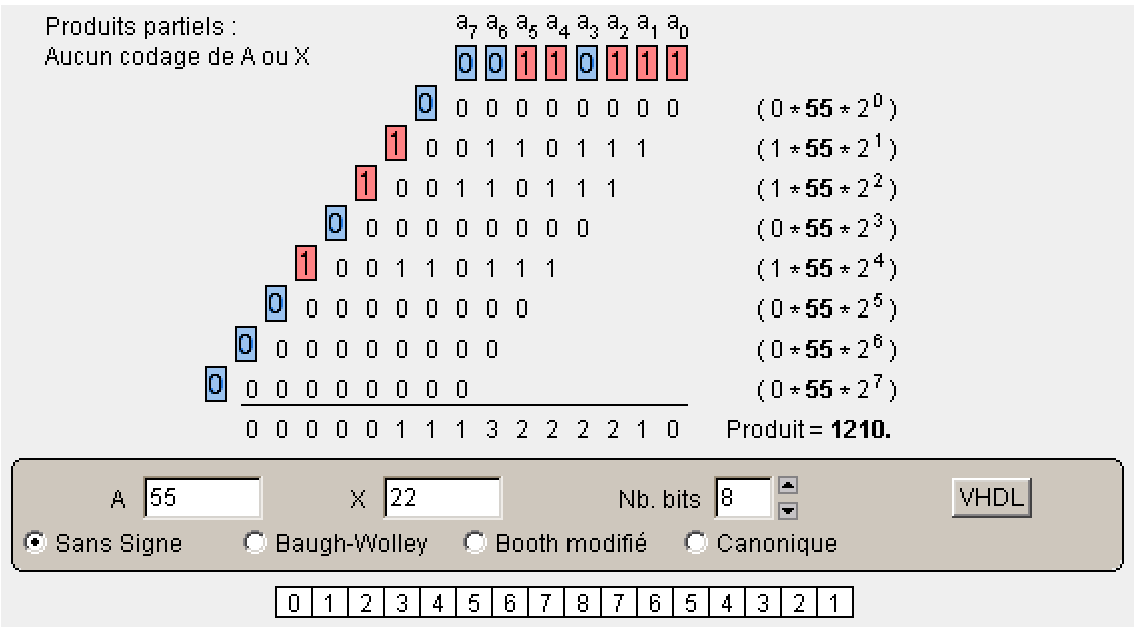




La multiplication de A requiert autant de cellule "CASS" que de bit de A plus 1.



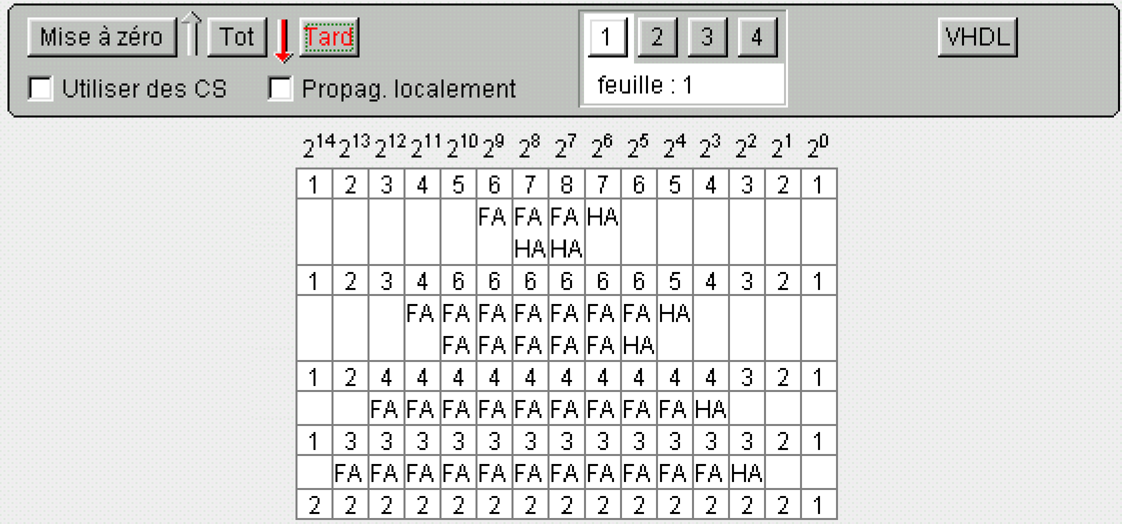
Génération des produits partiels La première étape de la multiplication génère à partie de A et de X des bits dont la somme pondérée vaut le produit P. Le bit de poids fort de P est positif pour la multiplication d'entiers sans signe, et négatif pour la multiplication d'entiers en complément à 2.



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.

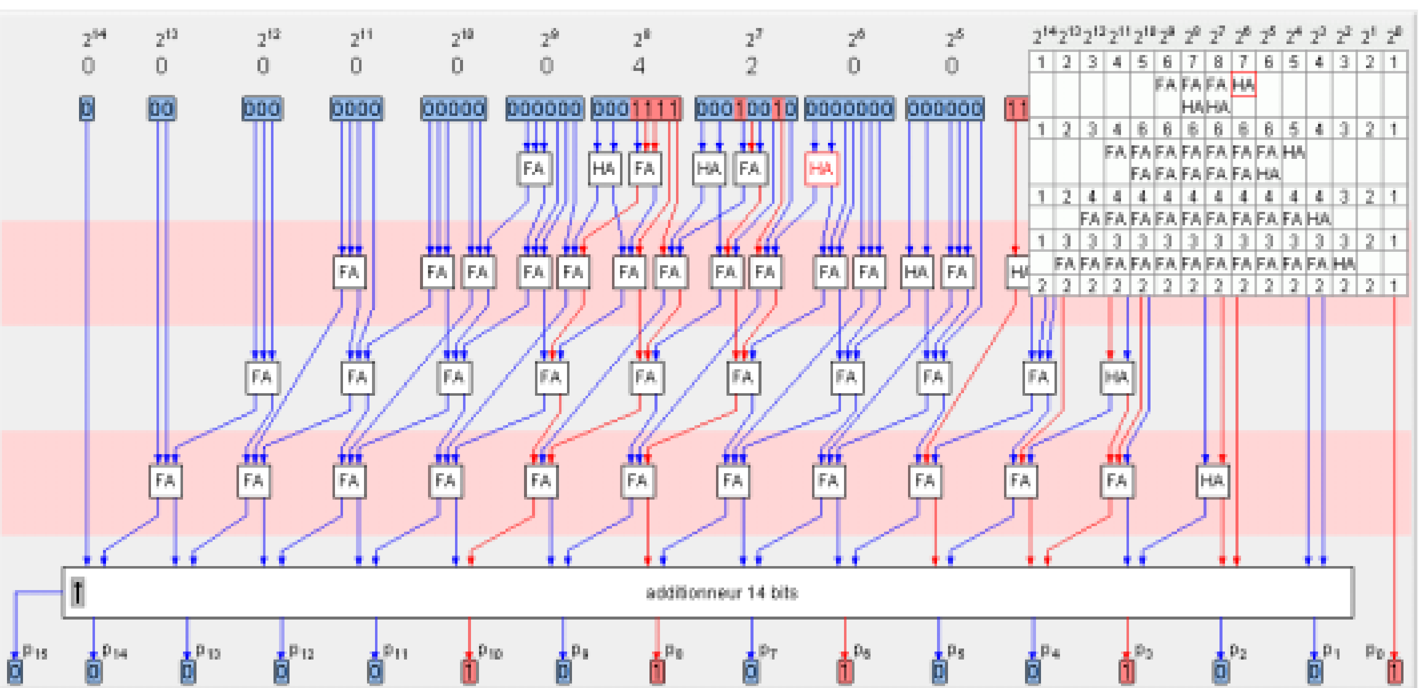
Les deux nombres binaires à ajouter dans la troisième étape peuvent être vus comme un seul nombre en "CS".



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}
1	2	3	4	5	6	7	8	7	6	5	4	3	2	1
					FA	FA	FA	HA						
						HA	HA							
1	2	3	4	6	6	6	6	6	6	5	4	3	2	1
			FA	FA	FA	FA	FA	FA	FA	HA				
				FA	FA	FA	FA	FA	HA					
1	2	4	4	4	4	4	4	4	4	4	4	3	2	1
		FA	FA	FA	FA	FA	FA	FA	FA	FA	HA			
1	3	3	3	3	3	3	3	3	3	3	3	3	2	1
	FA	FA	FA	FA	FA	FA	FA	FA	FA	FA	FA	HA		
2	2	2	2	2	2	2	2	2	2	2	2	2	2	1

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 entent.

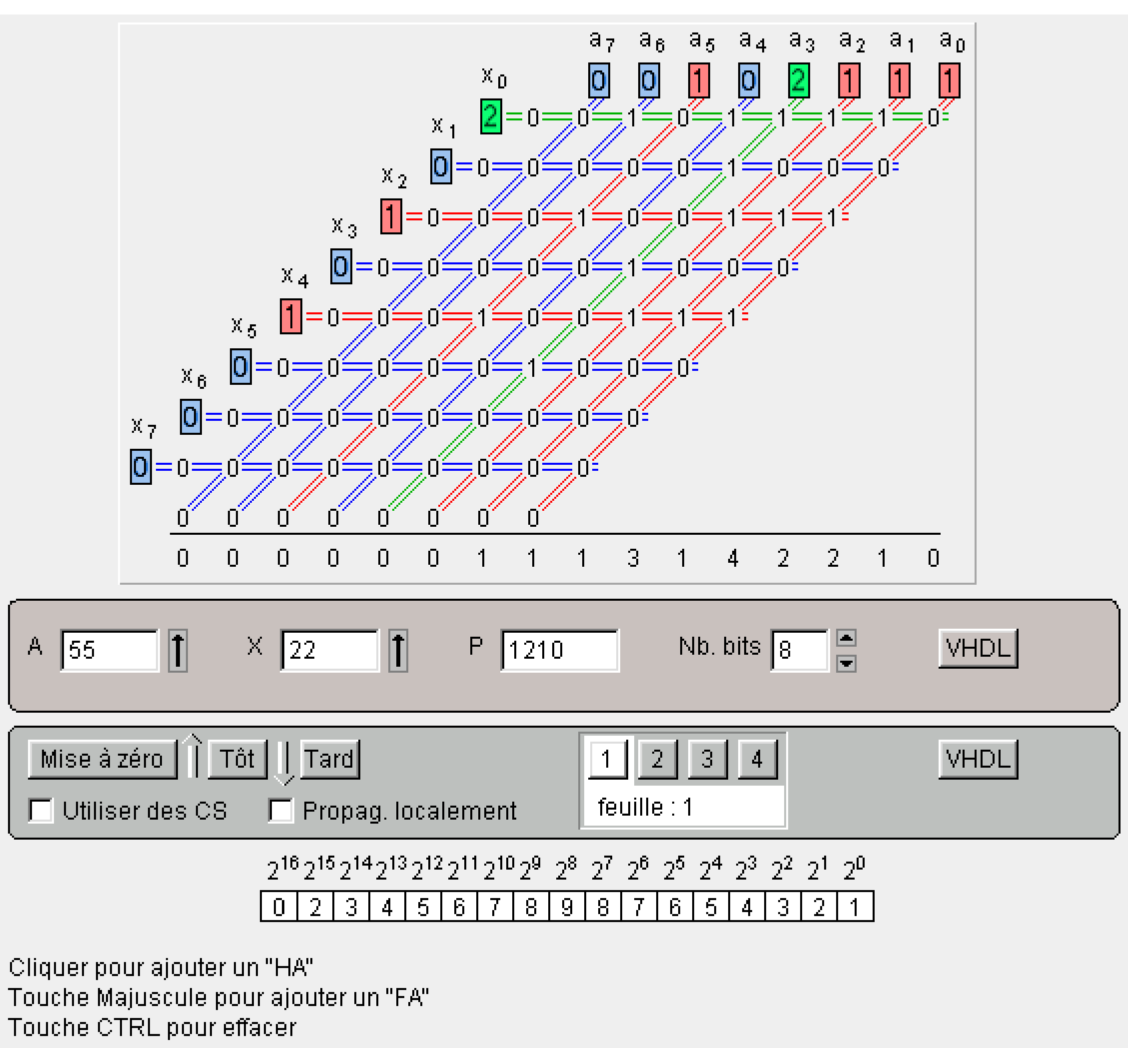


La flèche verticale supprime/rétabli l'additionneur final. Le résultat est en binaire avec l'additionneur et en "CS" sans lui. Le délai de l'addition finale est comparable à celui de l'arbre de réduction.

Produits partiels d'opérandes en "CS"

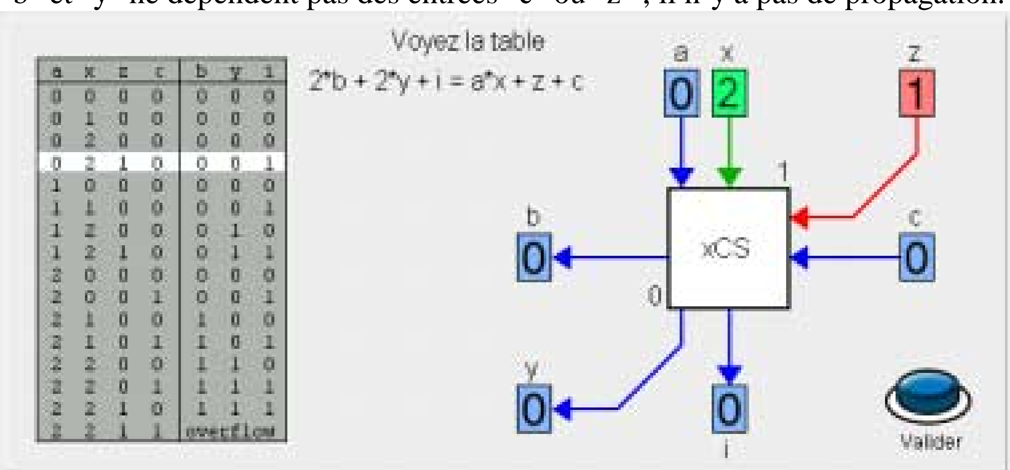
Le multiplieur X et le multiplicande A sont tous les deux en "CS", c'est à dire avec des chiffres $\in \{0, 1, 2\}$.

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'.

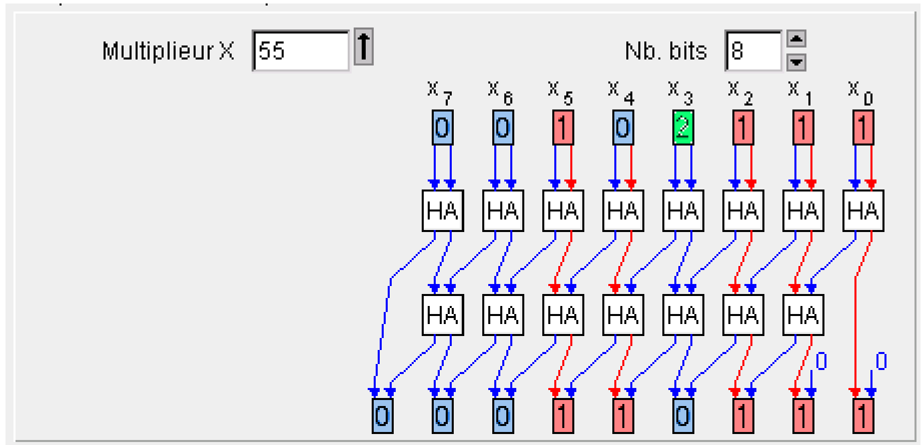


Réduction des produits partiels Les produits partiels de la multiplication de nombres en "CS" sont simplement des bits, réduits de la même façon que pour les multiplieurs précédents.

Cellule "xCS" La cellule "×CS" 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.

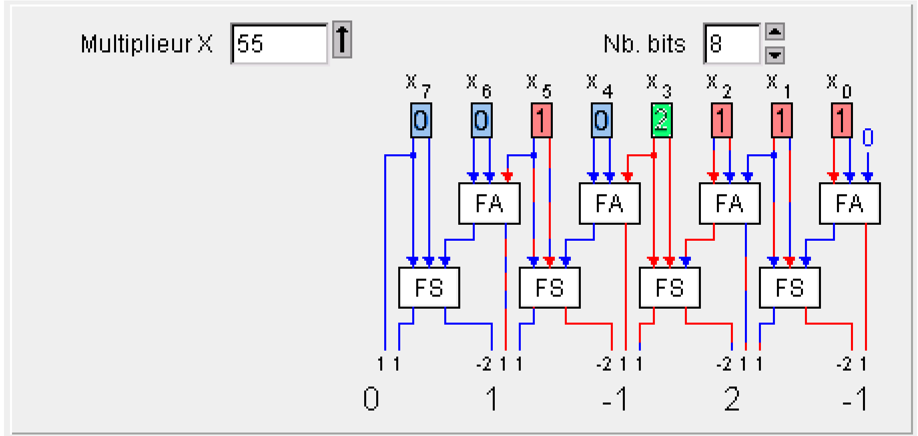


Codeur "CS2CS" Le transcodeur "CS2CS" passe de "CS" à "CS" en garantissant qu'en sortie un '2' est toujours précédé à droite par un '0'.



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 (ou "Code de Booth").



Multiplieurs par des constantes

Multiplication d'un nombre par une série de constantes La transformée de Fourier discrète, la transformée Cosinus ou son inverse, le filtre numérique, etc..., contiennent des multiplications d'une variable X par plusieurs constantes C1, C2, .. Cn. La factorisation de ces constantes permet de réduire le nombre d'additions/soustractions de ces multiplications, parfois très fortement. L'applet ci-dessous calcule Y1 = X∗C1, Y2 = X∗C2, .. Yn = X∗Cn...

Nb. const. 3 2717 2726 2723 VHDL

$Y1 = X*C1 = X*\mathbf{2717} = X*2^{11} + X*2^{9} + X*2^{7} + X*2^{4} + X*2^{3} + X*2^{2} + X*2^{0}$

$Y2 = X*C2 = X*\mathbf{2726} = X*2^{11} + X*2^{9} + X*2^{7} + X*2^{5} + X*2^{2} + X*2^{1}$

$Y3 = X*C3 = X*\mathbf{2723} = X*2^{11} + X*2^{9} + X*2^{7} + X*2^{5} + X*2^{1} + X*2^{0}$

Pas 0 : coût = 16 additions

$Y1 = X*C1 = X*\mathbf{2717} = X*2^{11} + X*2^{9} + X*2^{7} + X*2^{5} - X*2^{2} + X*2^{0}$

$Y2 = X*C2 = X*\mathbf{2726} = X*2^{11} + X*2^{9} + X*2^{7} + X*2^{5} + X*2^{3} - X*2^{1}$

$Y3 = X*C3 = X*\mathbf{2723} = X*2^{11} + X*2^{9} + X*2^{7} + X*2^{5} + X*2^{2} - X*2^{0}$

Pas 1 : coût = 12 additions et 3 soustractions

$Y1 = X*C1 = X*\mathbf{2717} = -X*2^{2} + X*2^{0} + Y4*2^{5}$

$Y2 = X*C2 = X*\mathbf{2726} = X*2^{11} + X*2^{9} + X*2^{7} + X*2^{5} + X*2^{3} - X*2^{1}$

$Y3 = X*C3 = X*\mathbf{2723} = X*2^{2} - X*2^{0} + Y4*2^{5}$

$Y4 = X*C4 = X*\mathbf{85} = X*2^{6} + X*2^{4} + X*2^{2} + X*2^{0}$

Pas 2 : coût = 10 additions et 2 soustractions

$Y1 = X*C1 = X*\mathbf{2717} = -X*2^{2} + X*2^{0} + Y4*2^{5}$

$Y2 = X*C2 = X*\mathbf{2726} = Y4*2^{3} + Y5*2^{1}$

$Y3 = X*C3 = X*\mathbf{2723} = X*2^{2} - X*2^{0} + Y4*2^{5}$

$Y4 = X*C4 = X*\mathbf{85} = X*2^{6} + X*2^{4} + X*2^{2} + X*2^{0}$

$Y5 = X*C5 = X*\mathbf{1023} = X*2^{10} - X*2^{0}$

Pas 3 : coût = 7 additions et 2 soustractions

$Y1 = X*C1 = X*\mathbf{2717} = -X*2^{2} + X*2^{0} + Y4*2^{5}$

$Y2 = X*C2 = X*\mathbf{2726} = Y4*2^{3} + Y5*2^{1}$

$Y3 = X*C3 = X*\mathbf{2723} = X*2^{2} - X*2^{0} + Y4*2^{5}$

$Y4 = X*C4 = X*\mathbf{85} = Y6*2^{2} + Y6*2^{0}$

$Y5 = X*C5 = X*\mathbf{1023} = X*2^{10} - X*2^{0}$

$Y6 = X*C6 = X*\mathbf{17} = X*2^{4} + X*2^{0}$

Pas 4 : coût = 6 additions et 2 soustractions

$Y1 = X*C1 = X*\mathbf{2717} = Y4*2^{5} - Y7*2^{0}$

$Y2 = X*C2 = X*\mathbf{2726} = Y4*2^{3} + Y5*2^{1}$

$Y3 = X*C3 = X*\mathbf{2723} = Y4*2^{5} + Y7*2^{0}$

$Y4 = X*C4 = X*\mathbf{85} = Y6*2^{2} + Y6*2^{0}$

$Y5 = X*C5 = X*\mathbf{1023} = X*2^{10} - X*2^{0}$

$Y6 = X*C6 = X*\mathbf{17} = X*2^{4} + X*2^{0}$

$Y7 = X*C7 = X*\mathbf{3} = X*2^{2} - X*2^{0}$

Pas 5 : coût = 4 additions et 3 soustractions

$Y1 = X*C1 = X*\mathbf{2717} = Y4*2^{5} - Y7*2^{0}$

$Y2 = X*C2 = X*\mathbf{2726} = Y4*2^{3} + Y5*2^{1}$

$Y3 = X*C3 = X*\mathbf{2723} = Y4*2^{5} + Y7*2^{0}$

$Y4 = X*C4 = X*\mathbf{85} = Y6*2^{2} + Y6*2^{0}$

$Y5 = X*C5 = X*\mathbf{1023} = X*2^{10} - X*2^{0}$

$Y6 = X*C6 = X*\mathbf{17} = X*2^{4} + X*2^{0}$

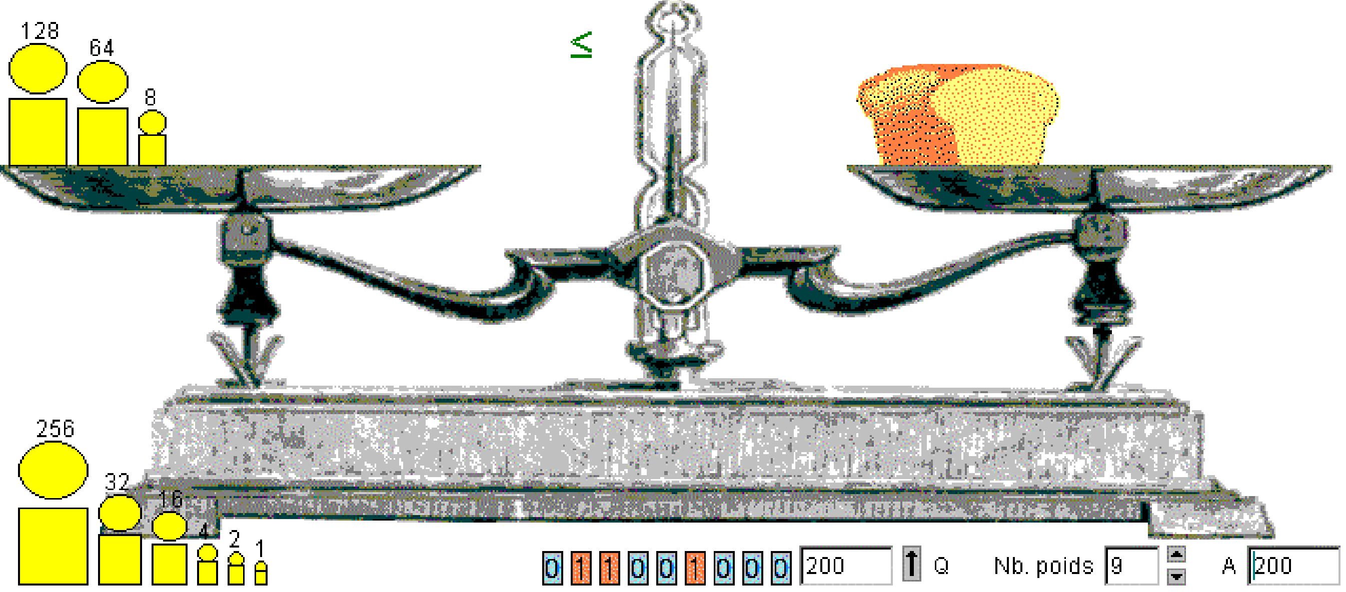
$Y7 = X*C7 = X*\mathbf{3} = X*2^{1} + X*2^{0}$

Pas 6 : coût = 5 additions et 2 soustractions

Diviseurs

Pesée du pain avec restauration sans restauration, pesée RST

On veut calculer Q = A ÷ 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, ... 2^i*D marqués respectivement avec 1, 2, 4, 8, ... 2^i.
En fait D est un nombre binaire, et 2^i*D s'obtient simplement en décalant D de i positions. La balance compare la somme des poids sur chacun de ses deux plateaux (≤ ou >).



Divisions récurrentes

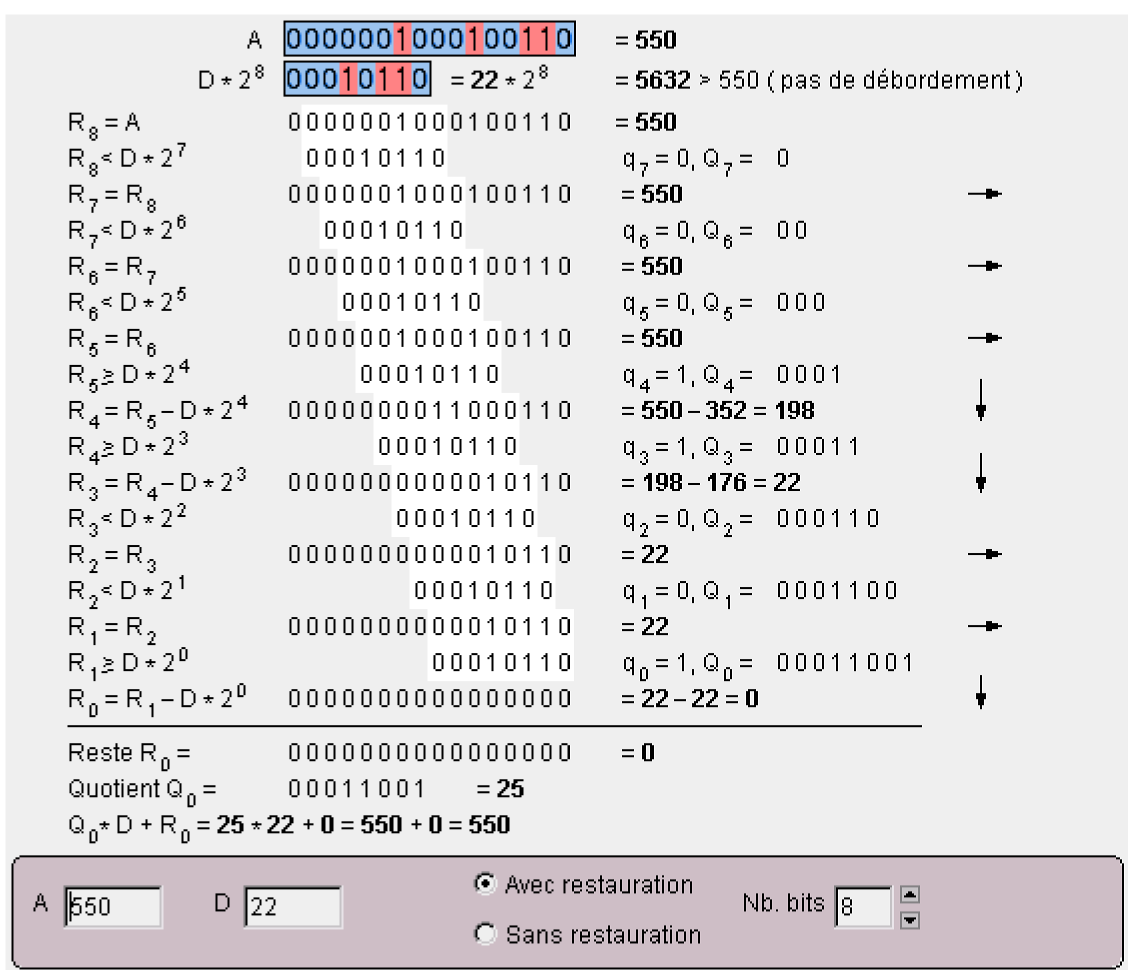
La division est peu fréquente, cependant comme son délai d'exécution est bien plus grand que celui de l'addition ou de la multiplication, son incidence sur le temps de calcul est substantielle. Il convient donc de soigner la réalisation des diviseurs.
On veut calculer Q = A ÷ D. Ceci est équivalent à 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 $Q_n, Q_{n-1}, \ldots Q_2, Q_1, Q_0$ et une suite $R_n, R_{n-1}, \ldots R_2, R_1, R_0$ 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 :

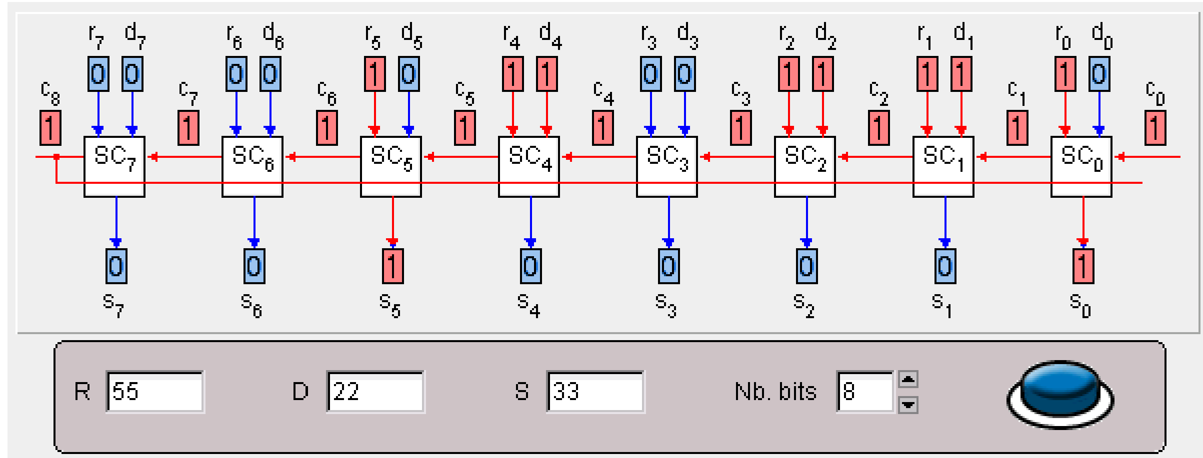
- $Q_n = 0$
- $R_n = A$.

Quand la récurrence se termine, on a $Q = Q_0 = \sum_{i=0}^{n} q_i * 2^i$. $R = R_0$ est le reste de la division.

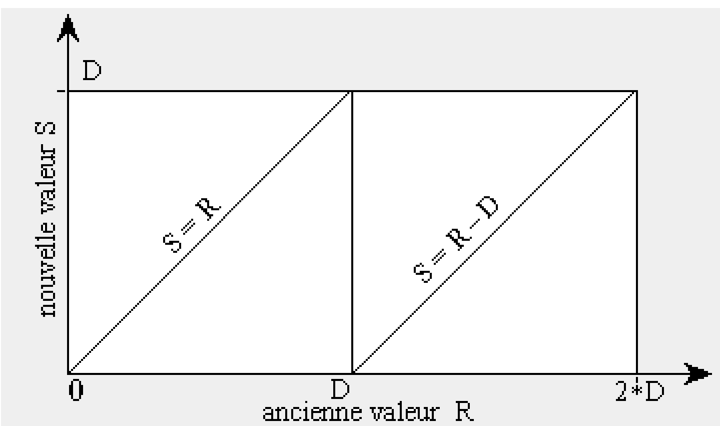


Soustracteur conditionnel

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 $0 \leq R \leq 2*D$.

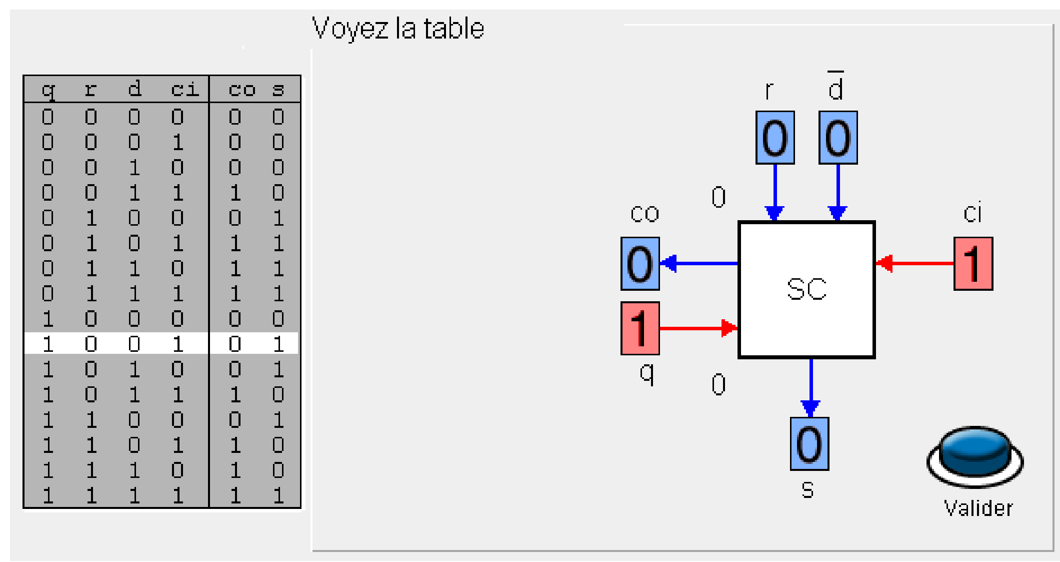


Cellule "SC" du soustracteur conditionnel

Vérifiez que vous maîtrisez les fonctions logiques de la cellule "SC" :

si q = 0 **alors** { co = majorité (r, $\overline{d}$, ci) ; s = r } // identité

sinon { co = majorité (r, $\overline{d}$, ci) ; s = r ⊕ $\overline{d}$ ⊕ ci } // soustraction



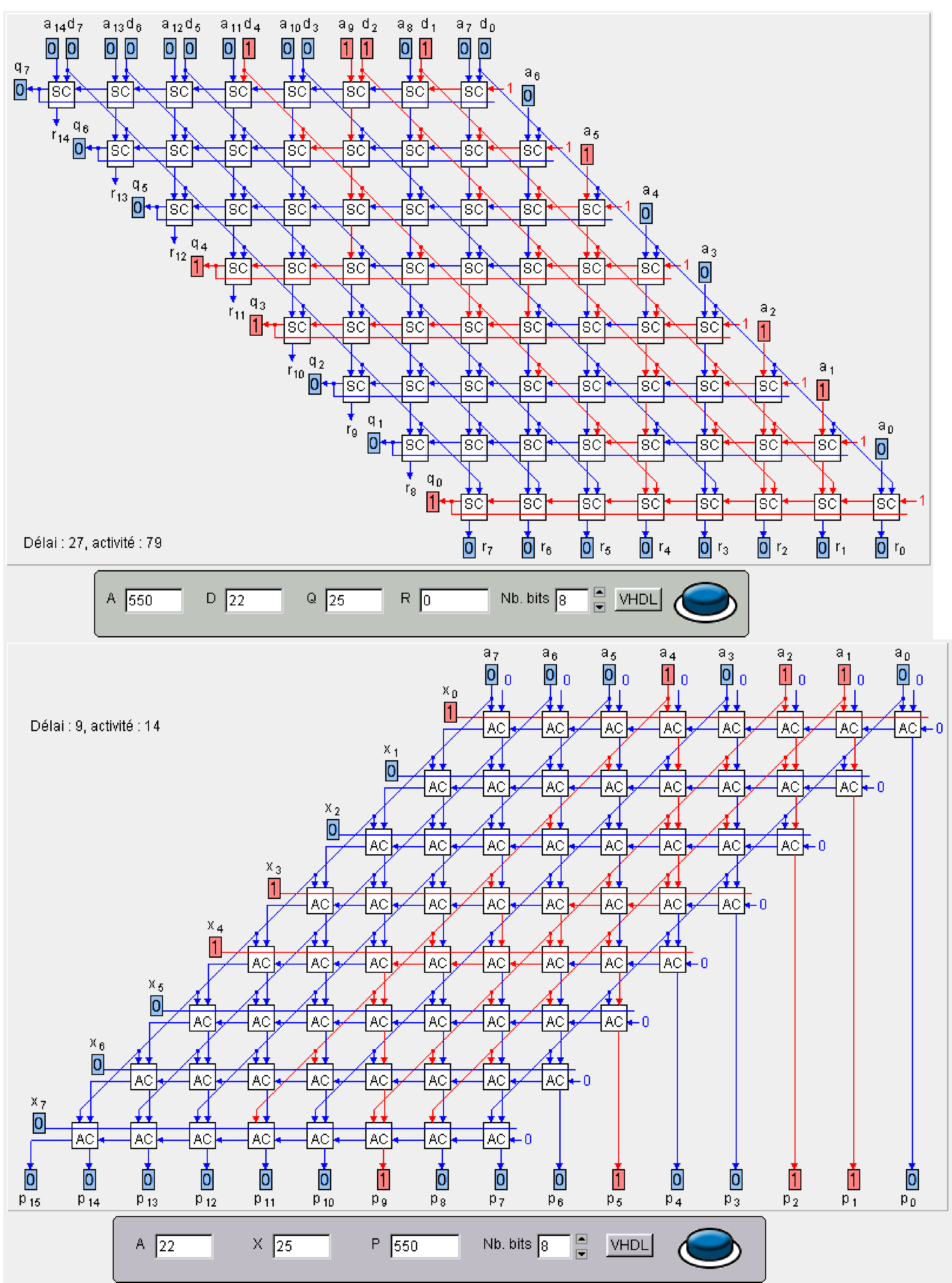
q	r	d	ci	co	s
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	1	1	0
0	1	0	0	0	1
0	1	0	1	1	1
0	1	1	0	1	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	0	1	0	1
1	0	1	0	0	1
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	1	1

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).

Inverse de la division

Si on retourne le diviseur "avec restauration" tête en bas et replace les soustracteurs conditionnels "SC" par des additionneurs conditionnels "AC" (identité ou addition selon x_i) on obtient l'opérateur de l'inverse de la division c'est à dire la multiplication.


Délai : 27, activité : 79
A 550
D 22
Q 25
R 0
Nb. bits 8
VHDL
Délai : 9, activité : 14
A 22
X 25
P 550
Nb. bits 8
VHDL

Diviseur sans restauration

Un diviseur "sans restauration" consiste en une suite de décalages et d'additions ou de soustractions. Il est formé d'une structure régulière de cellules d'addition/soustraction "AS" (addition ou soustraction en fonction de q).

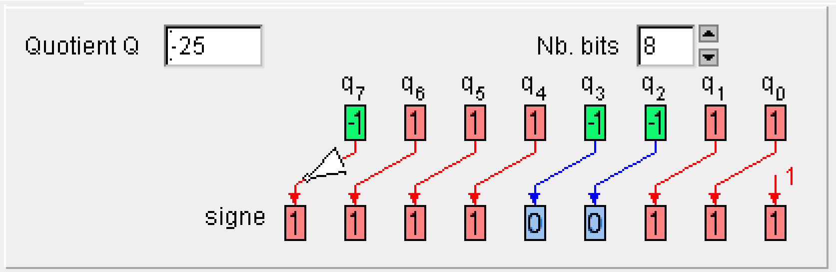
- **si** R < 0 **et** D < 0 **alors** {S = R – D ; q = '1'}
- **si** R < 0 **et** D ≥ 0 **alors** {S = R + D ; q = '$\overline{1}$ '}
- **si** R ≥ 0 **et** D < 0 **alors** {S = R + D ; q = '$\overline{1}$ '}
- **si** R ≥ 0 **et** D ≥ 0 **alors** {S = R – D ; q = '1'}

Délai : 29, activité : 144

A 550 D -22 Q -25 R 0 Nb. bits 8 VHDL

Avantage et inconvénient du diviseur sans restauration

Le diviseur sans restauration a l'avantage de la compatibilité de A et de D avec le complément à 2. Cependant pour que le résultat soit unique on convient que le reste R soit toujours du même signe que le dividende A. Une correction finale est alors nécessaire pour que le diviseur sans restauration obéisse à cette convention. Enfin le quotient Q est écrit avec les chiffres '$\overline{1}$ ' et '1' , il faut donc le convertir en binaire, ce qui est très simple.



Diviseurs rapides

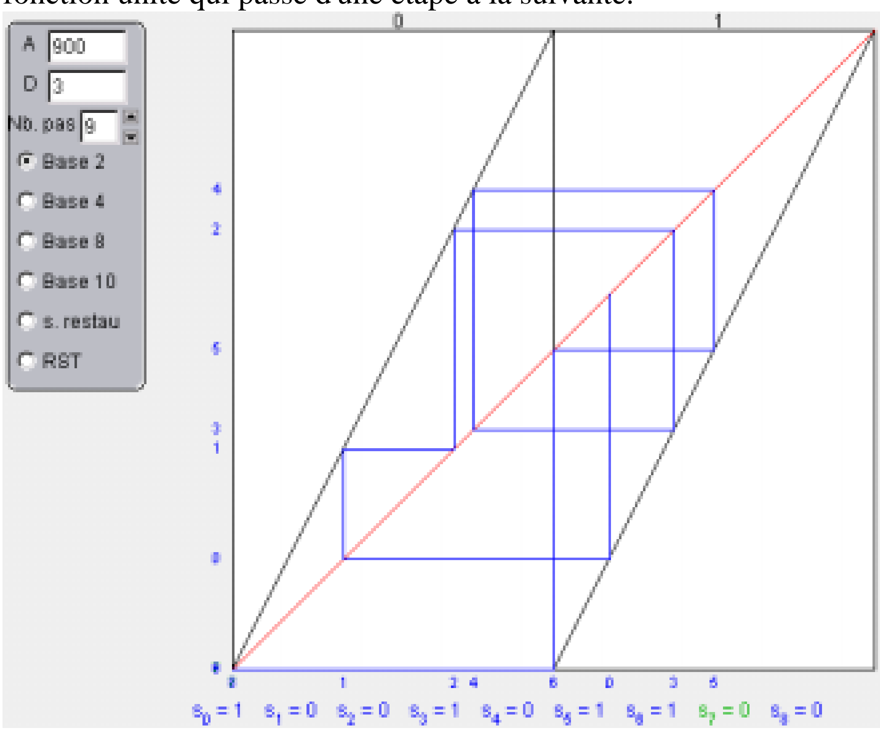
Trois approches se combinent pour réaliser des diviseurs rapides:

- Utiliser des additions/soustraction sans propagation de retenue.
- Préconditionner le diviseur et le dividende 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 $R_j \Rightarrow R_{j-1}$, le trait rouge est la fonction unité qui passe d'une étape à la suivante.

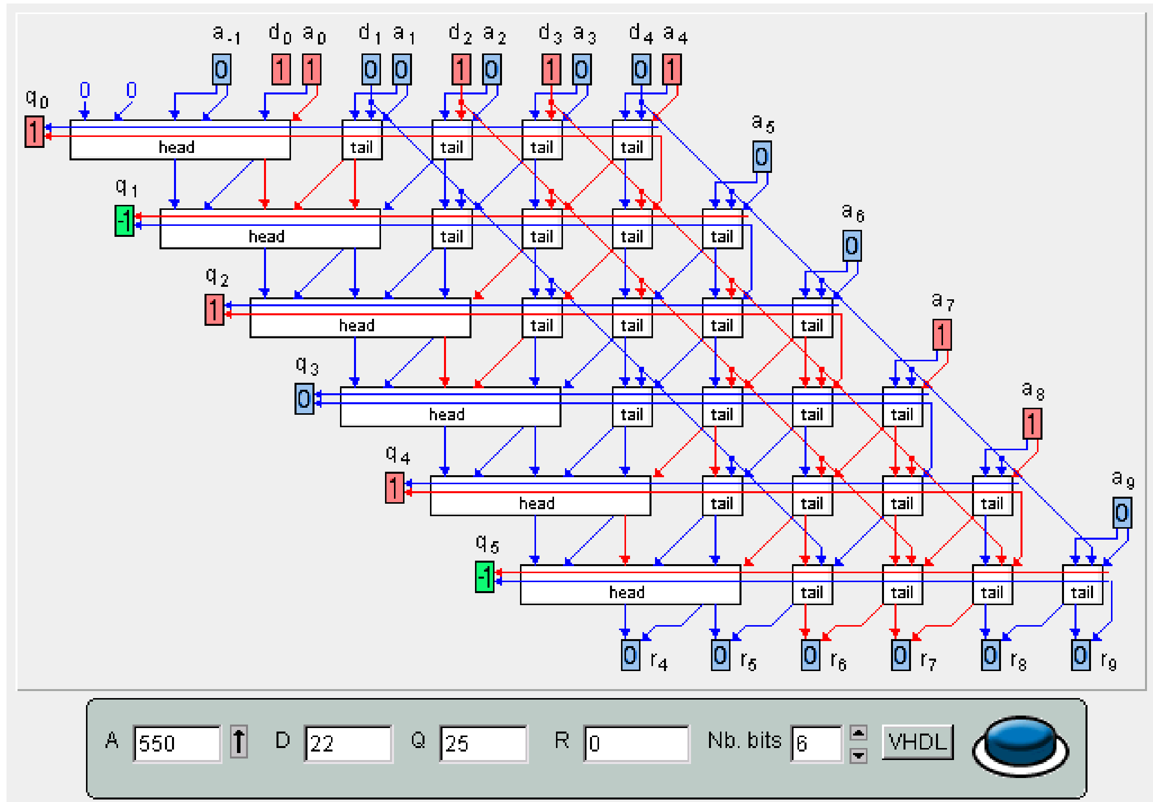


La base 10 nous est familière, elle ne sert ici que comme illustration car elle serait très inefficace codée en binaire.

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 : $R_{j-1} = R_j$



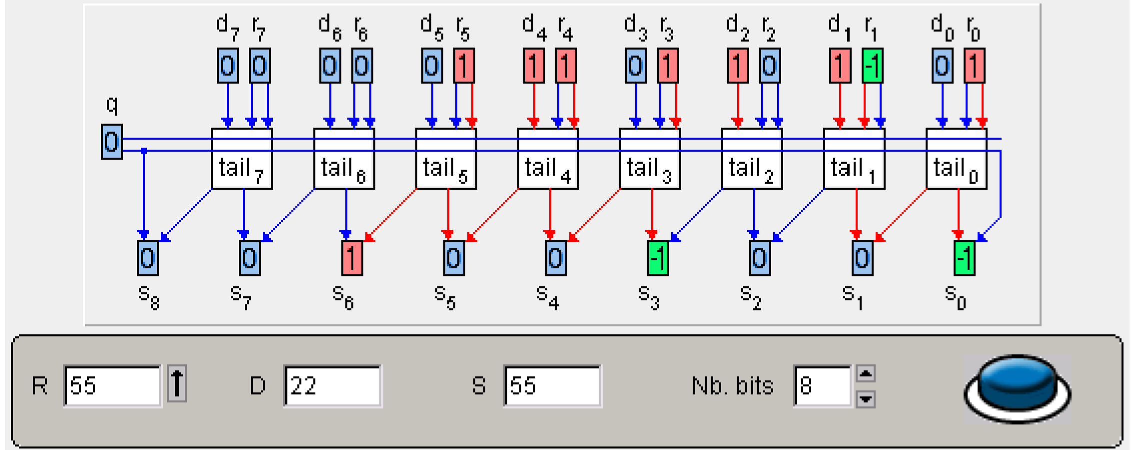
Cette opération est choisie suivant le signe du reste R_j. Pour connaître précisément ce signe il faudrait pouvoir examiner tous les chiffres du reste. On montre qu'il n'est nécessaire d'en examiner que 3. De plus on connaît la position de ces 3 chiffres: celui de droite est aligné avec le premier bit 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

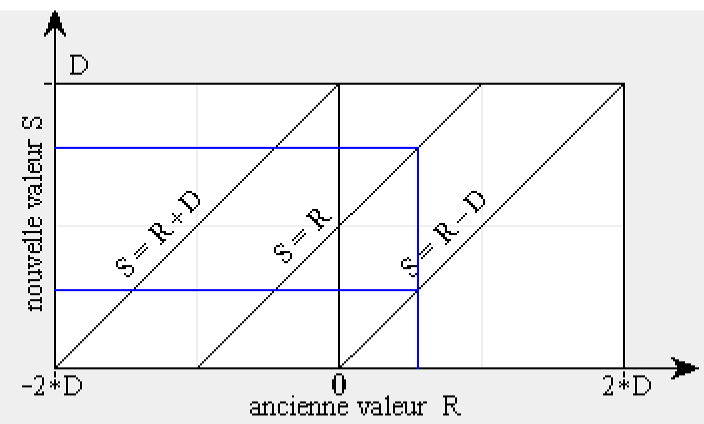
Un "additionneur/soustracteur conditionnel" donne l'un des résultats S suivant:

- **si** q = '$\overline{1}$' **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.



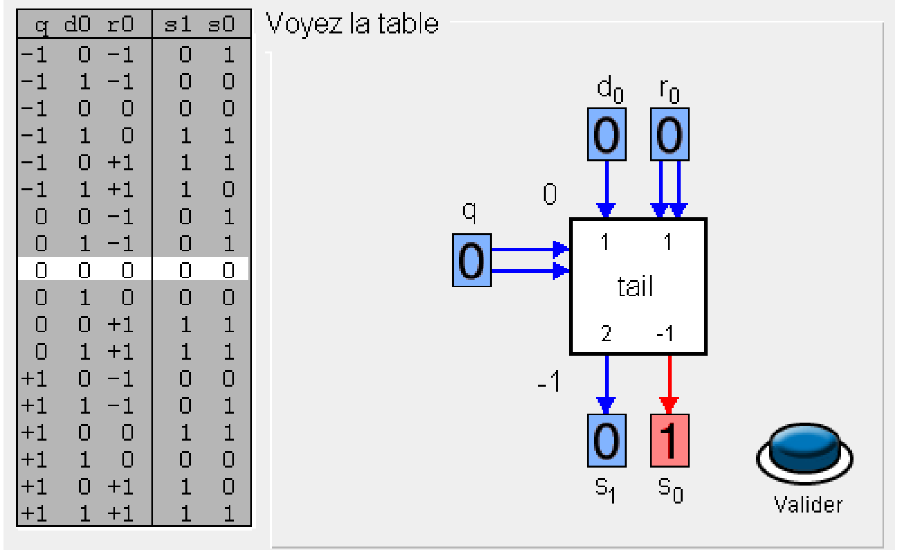
La fonction de l' "additionneur/soustracteur conditionnel" se résume par sa fonction de transfert appelée "diagramme de Robertson". Pour converger, la division impose en outre que -2*D ≤ R ≤ 2*D. Si -D ≤ R ≤ 2 alors S a deux solutions.



Cellule "tail" du diviseur "SRT"

Vérifiez que vous maîtrisez les fonctions logiques de la cellule "tail" .

- **si** q = '$\overline{1}$ ' **alors** $2*s_1 - s_0 = d_0 + r_0$; // addition
- **si** q = '0' **alors** $2*s_1 - s_0 = r_0$; // identité
- **si** q = '1' **alors** $2*s_1 - s_0 = \overline{d_0} + r_0$; // soustraction



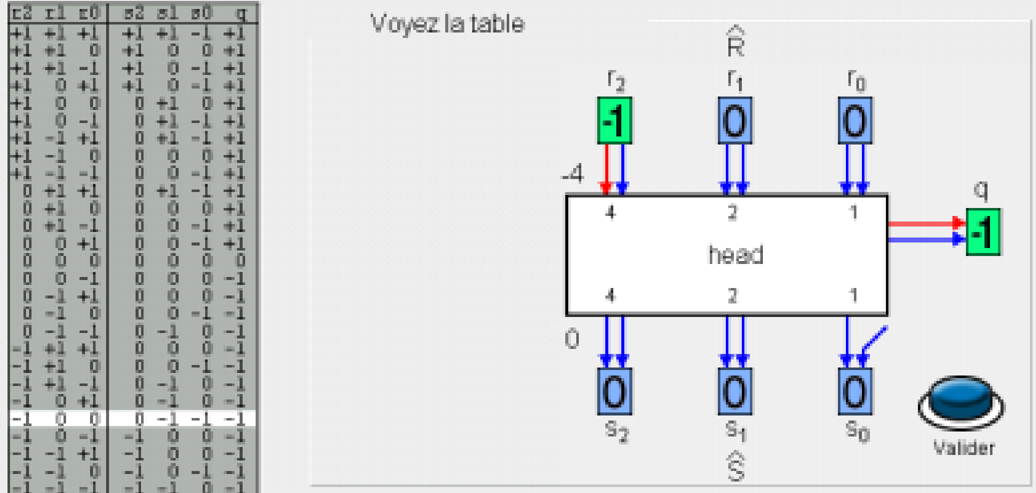
q	d0	r0	s1	s0
-1	0	-1	0	1
-1	1	-1	0	0
-1	0	0	0	0
-1	1	0	1	1
-1	0	+1	1	1
-1	1	+1	1	0
0	0	-1	0	1
0	1	-1	0	1
0	0	0	0	0
0	1	0	0	0
0	0	+1	1	1
0	1	+1	1	1
+1	0	-1	0	0
+1	1	-1	0	1
+1	0	0	1	1
+1	1	0	0	0
+1	0	+1	1	0
+1	1	+1	1	1

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 respectivement 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 = '$\overline{1}$ '; }

Lors d'une division (sans débordement), la sortie s_2 est <u>toujours</u> à 0. Cependant le débordement est utilisé dans la "double division".



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	-1	0	-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

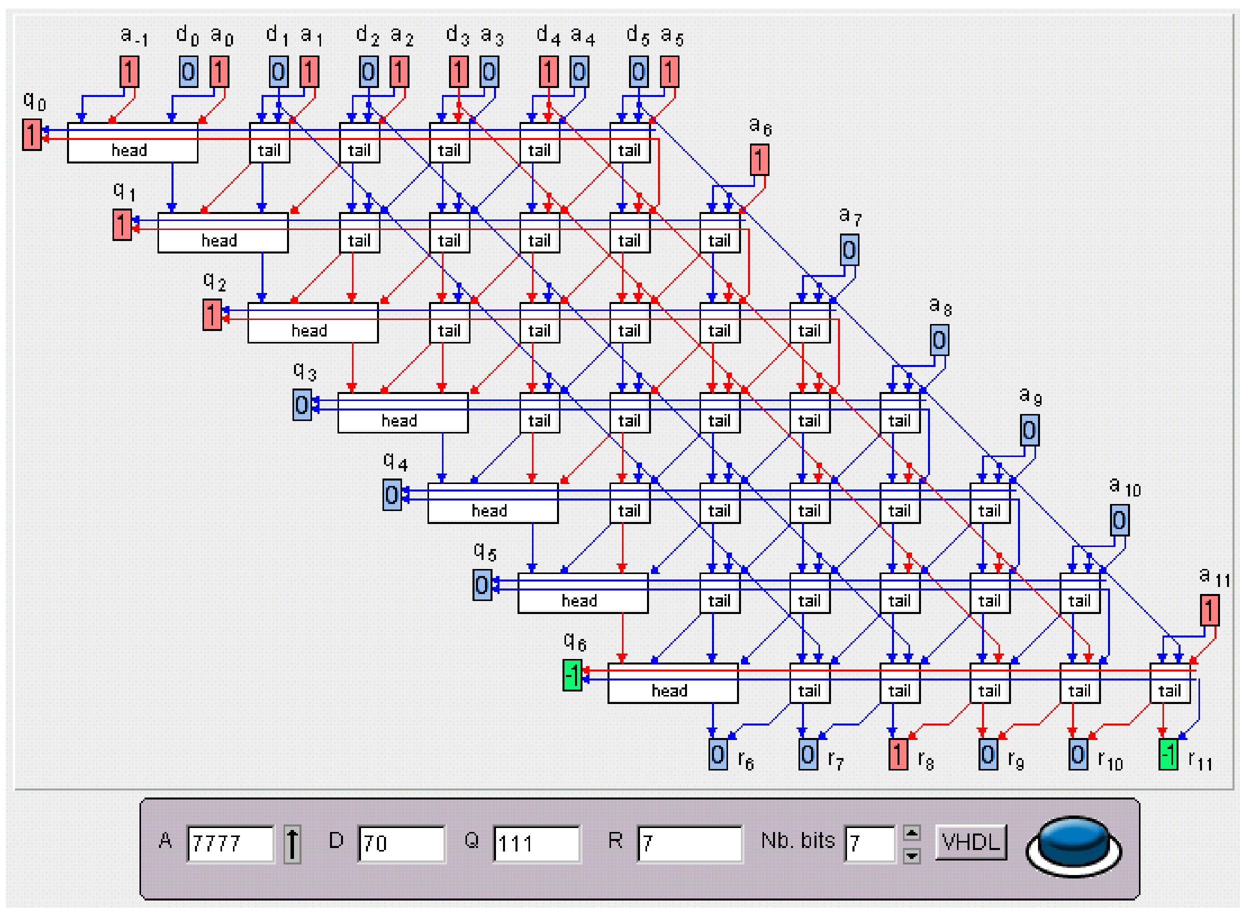
Division "SRT" avec réduction du diviseur

La division "SRT" précédente est simple car le premier bit du diviseur D est toujours '1'. Elle se simplifie davantage si les <u>deux premiers bits</u> 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, $2^{n-1} - 1 < D < 2^{n-1} + 2^{n-2}$.

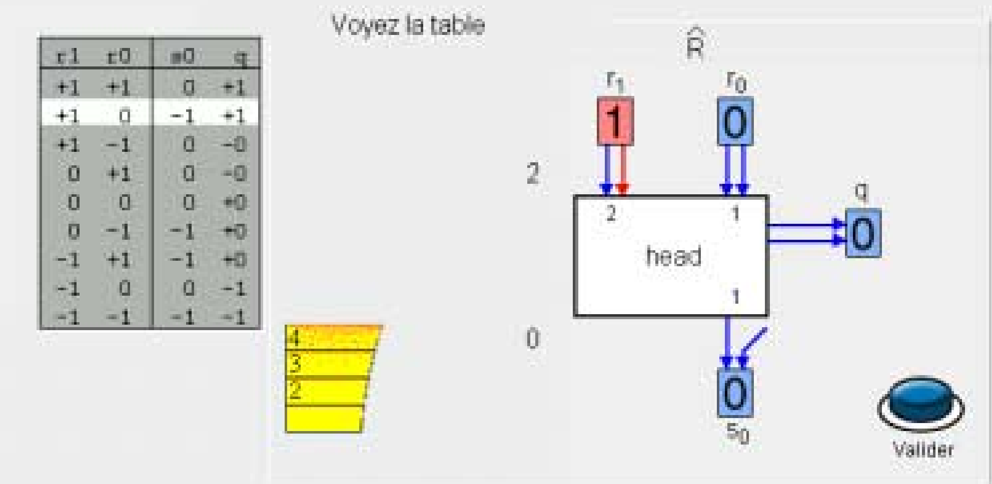


Cellule de tête du diviseur "SRT" avec réduction

Soient $\hat{R} = r_1 * 2 + r_0$ la valeur de l'entrée de la cellule de tête "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* { $s_0 =$ '$\bar{1}$' ; $q = -0$; }
- **si** $\hat{R} = -1$ **alors** { $s_0 = -1$; $q = +0$; }
- **si** $\hat{R} < -1$ **alors** { $s_0 = \hat{R} + 2$; $q =$ '$\bar{1}$' ; }

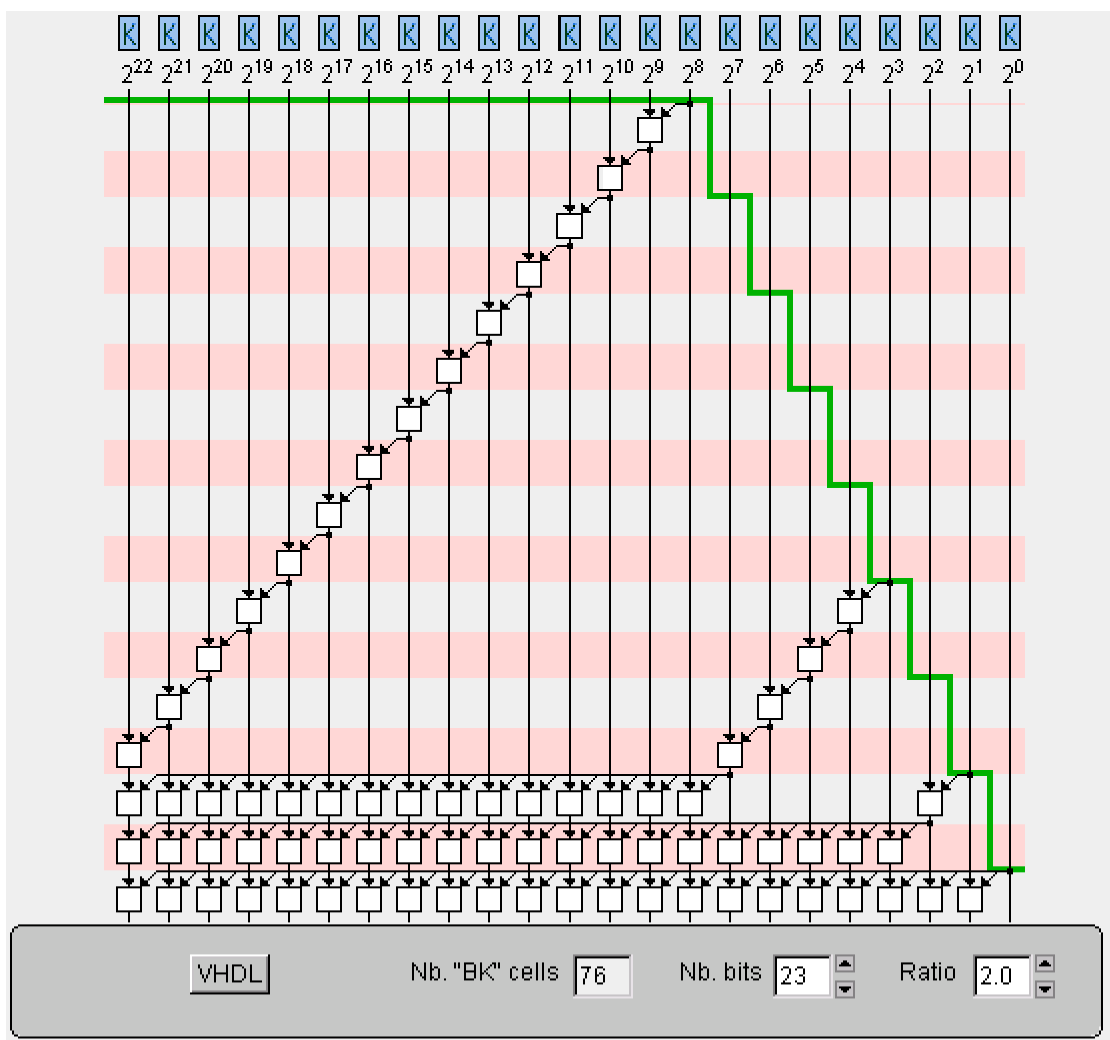
La différence entre les deux écritures de 0 pour q : "- 0" et "+ 0" importe.



r1	r0	s0	q
+1	+1	0	+1
+1	0	−1	+1
+1	−1	0	−0
0	+1	0	−0
0	0	0	+0
0	−1	−1	+0
−1	+1	−1	+0
−1	0	0	−1
−1	−1	−1	−1

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.

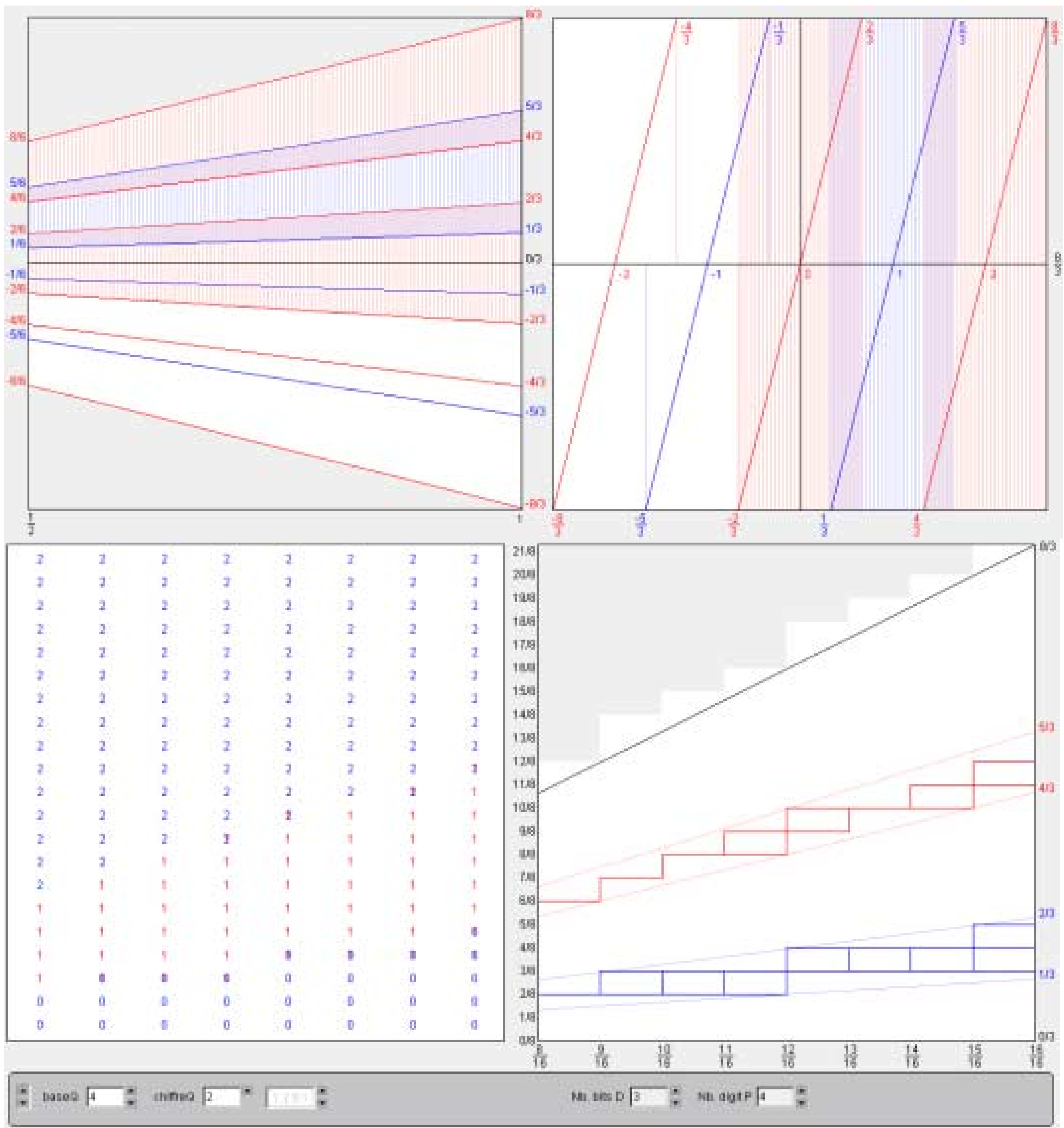
Soit "Ratio" le rapport entre le délai de la cellule de tête du diviseur SRT et le délai de la cellule "BK".



Conception d'un diviseur Les chiffres du quotient Q sont redondants et symétriques. Ils sont définis par la base de numération de Q et la valeur maximale du chiffre Q.

Le choix du nombre de chiffres du diviseur D et du reste partiel P pris en

compte détermine si on peut tracer des frontières séparant les chiffres de Q.



Le bouton à gauche passe à l'étape suivante ou revient à l'étape précédente.

1- diagramme de Robertson, donnant le reste partiel suivant en fonction du reste partiel P courant , sans tenir compte explicitement de D

2- PD-plot symétrique tracé pour les valeurs de D dans [1/2 , 1 [

3- demi PD-plot continu, moitié supérieure du précédent. La moitié inférieure s'obtiendrait en changeant les signes.

4- demi PD-plot à discrétiser, le nombre de bits de D fixe le pas en abscisse, le nombre de chiffres de P fixe le pas en ordonnée.

Ce choix détermine si on peut tracer des frontières pour séparer les chiffres de Q.

5- demi-table de vérité du demi PD-plot.

Extracteur de racine carrée

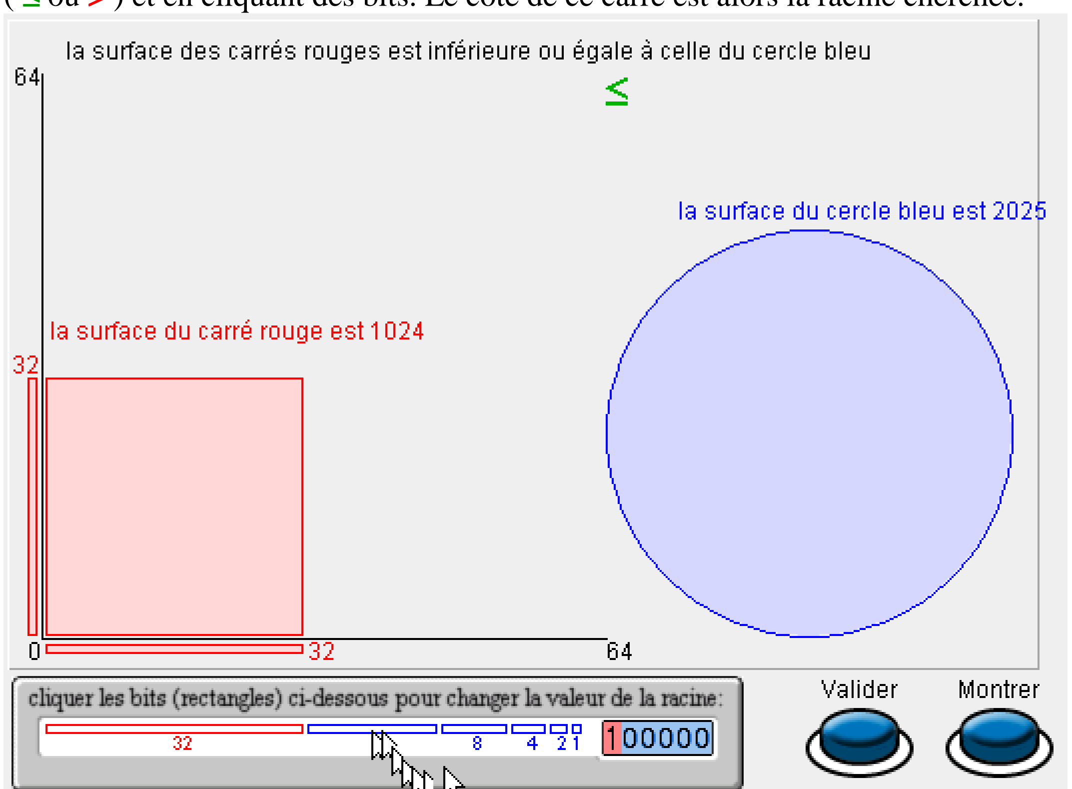
Extraction de racine carrée

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 dessinée 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 (≤ ou >) et en cliquant des bits. Le côté 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 $Q_n, Q_{n-1}, \ldots Q_2, Q_1, Q_0$ et une suite $R_{2n}, R_{2n-2}, \ldots R_4, R_2, R_0$ telles que l'invariant $A = Q_j * Q_j + R_{2j}$ soit respecté pour tout j.

La récurrence est :

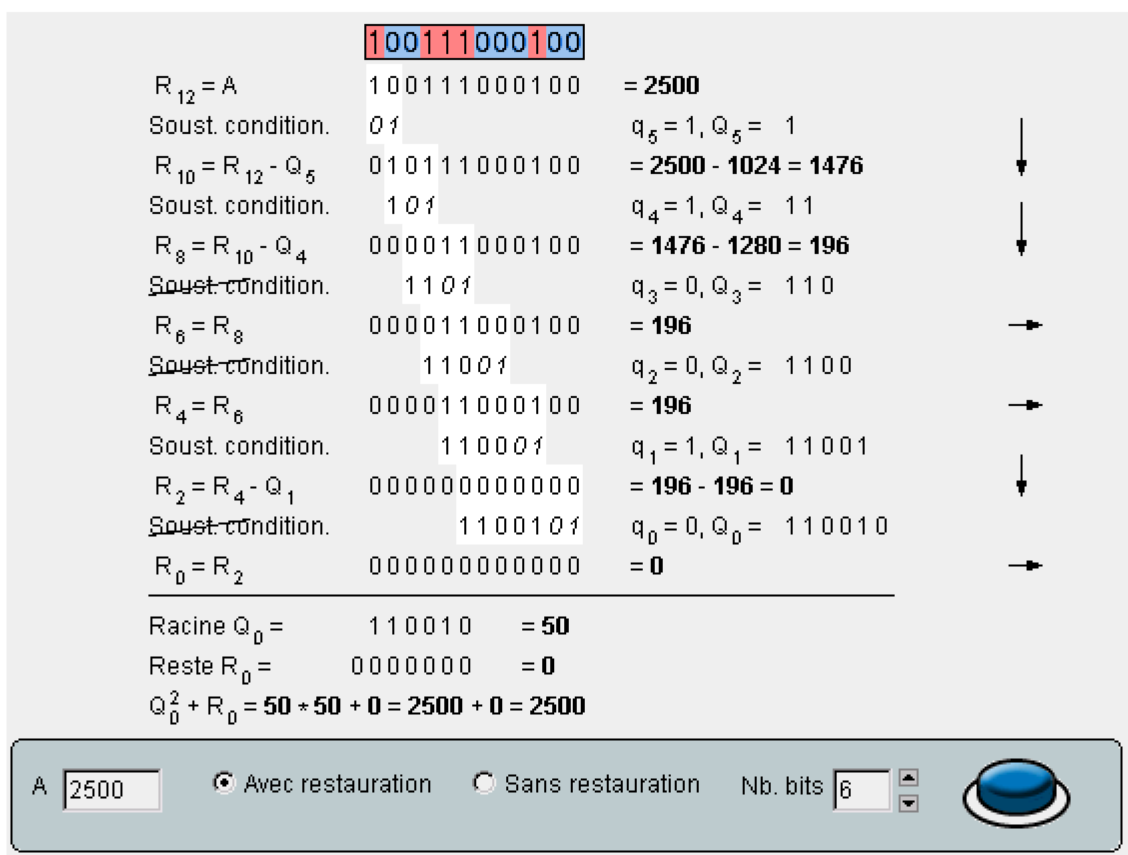
- $Q_{j-1} = Q_j + q_{j-1} * 2^{j-1}$
- $R_{2j-2} = R_{2j} - q_{j-1} * 2^{j-1} * (2 * Q_j + 2^{j-1})$

avec comme conditions initiales :

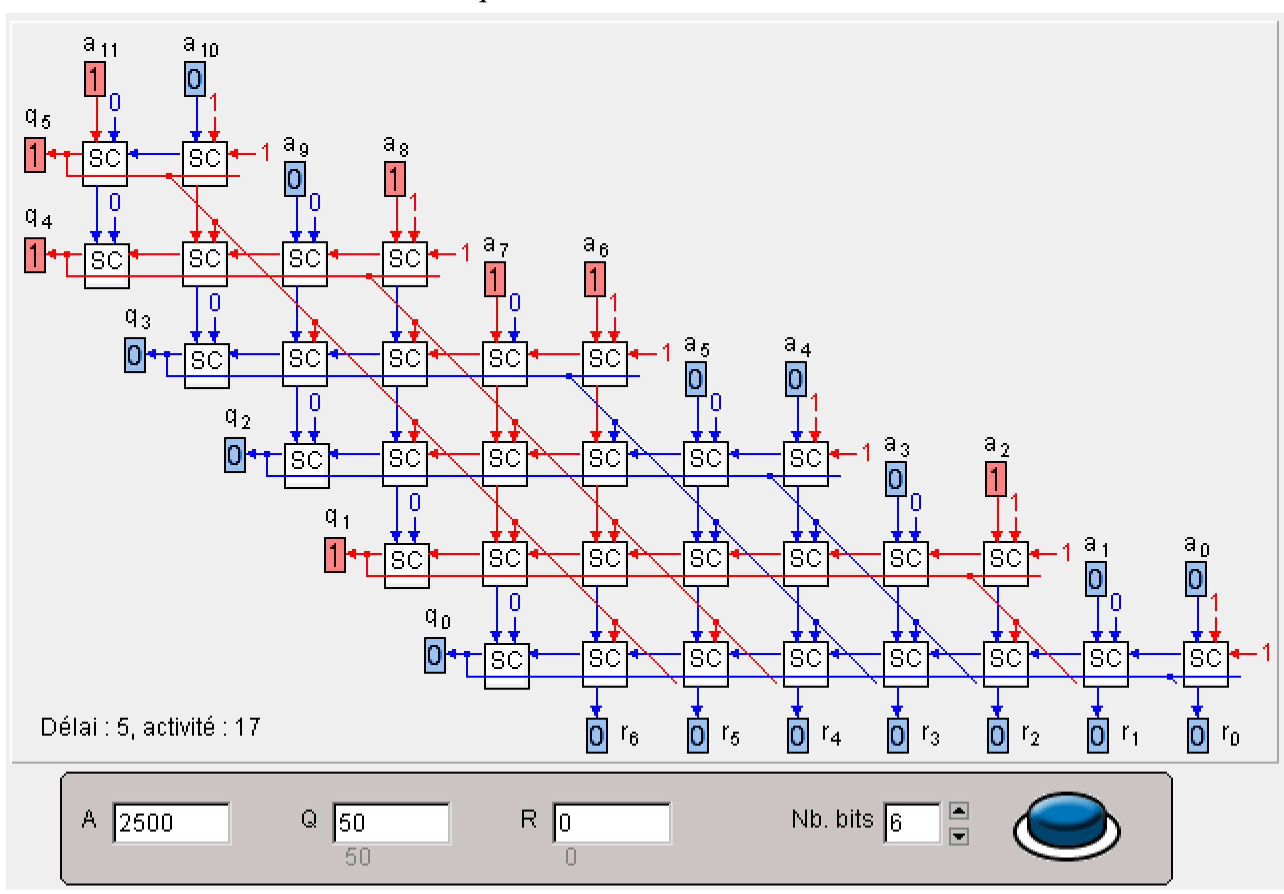
- $Q_n = 0$
- $R_{2n} = A$.

Quand la récurrence se termine, on a $Q = Q_0 = \sum_{i=0}^{n} q_i * 2^i$.

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



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

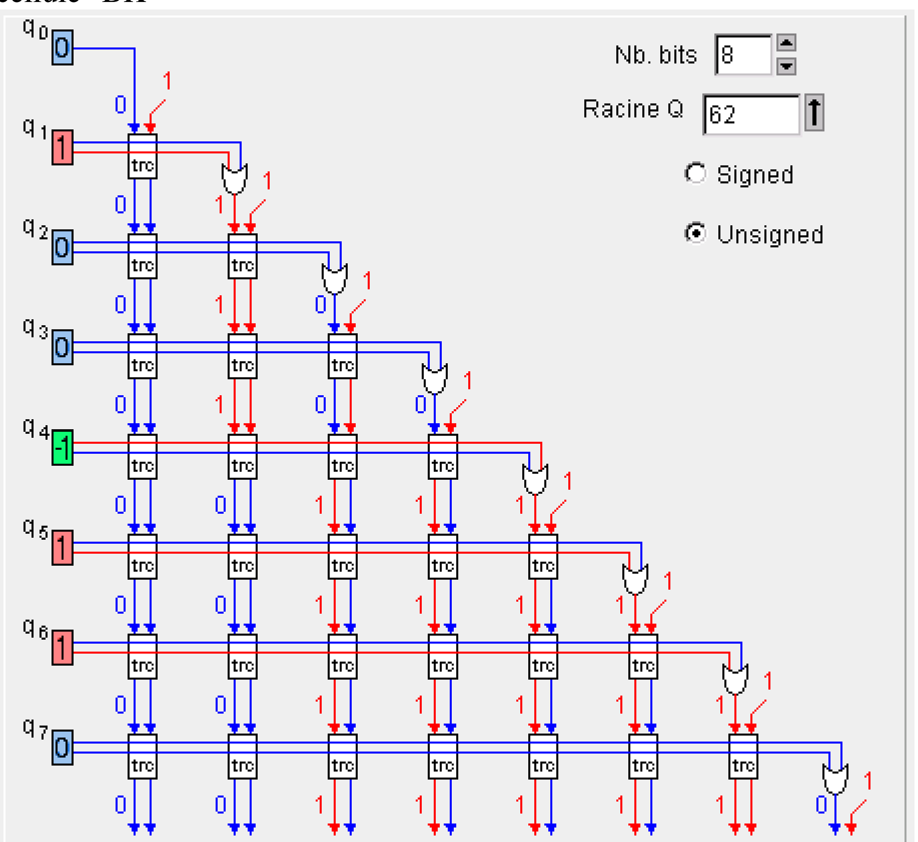


Extracteur de racine carrée rapide On cherche à s'affranchir de la propagation de retenue en utilisant la notation "BS", les mêmes cellules "head" et "tail" et une architecture similaire à celle de la division rapide. On se heurte à trois difficultés en cherchant à utiliser ce diviseur pour extraire

rapide des racines.

Convertisseur de racine

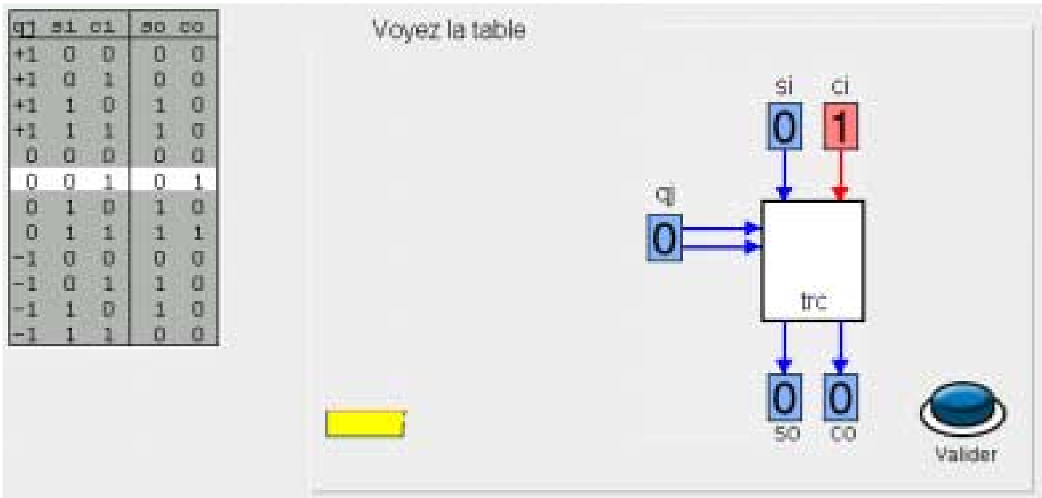
La première difficulté est le rebouclage de la racine. Comme le diviseur, l'extracteur de racine carrée rapide fourni des racines partielles Q_j en notation "BS". Utiliser les cellules "head" et "tail" que le diviseur rapide exige une racine partielle en notation binaire conventionnelle. On pourrait 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 = '$\overline{1}$' **alors** { so = si ⊕ 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



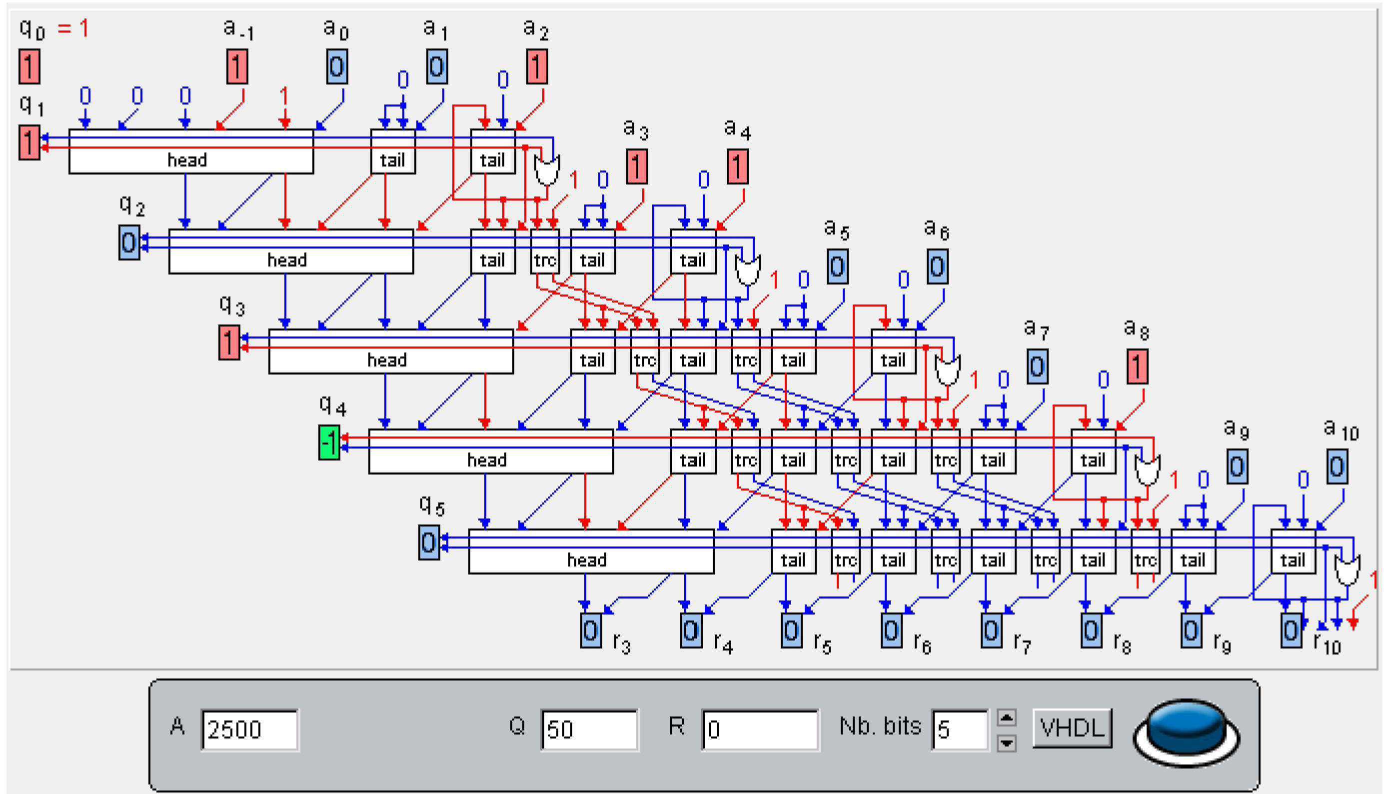
qj	si	ci	so	co
+1	0	0	0	0
+1	0	1	0	0
+1	1	0	1	0
+1	1	1	1	0
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	1	1
-1	0	0	0	0
-1	0	1	1	0
-1	1	0	1	0
-1	1	1	0	0

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 = '$\bar{1}$' **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 = '$\bar{1}$' 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".



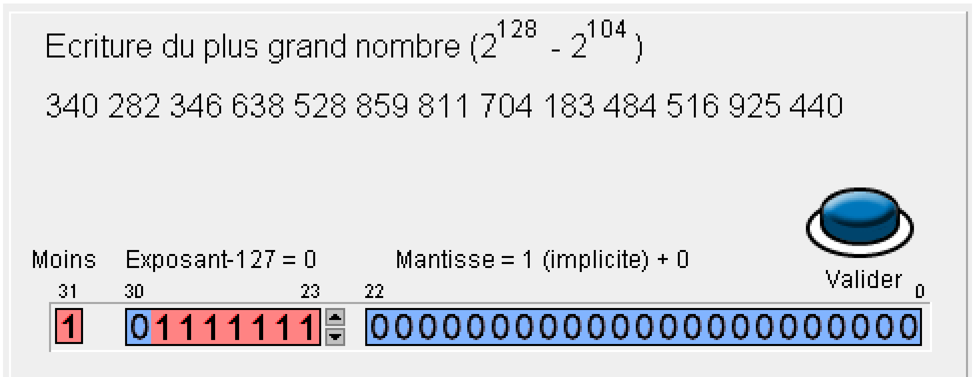
Addition en Virgule Flottante

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.

Vérifiez que vous vous maîtrisez ce format en donnant l'écriture (32 bits) des nombres proposés.



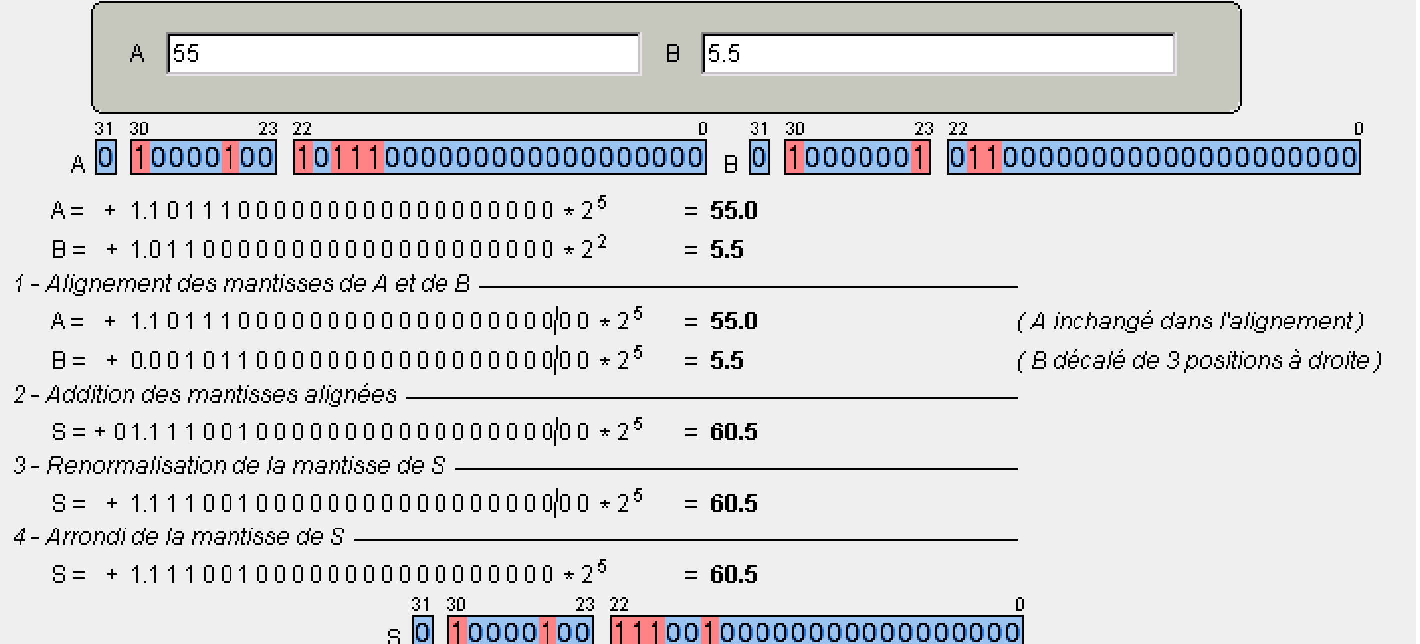
Addition et soustraction

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:

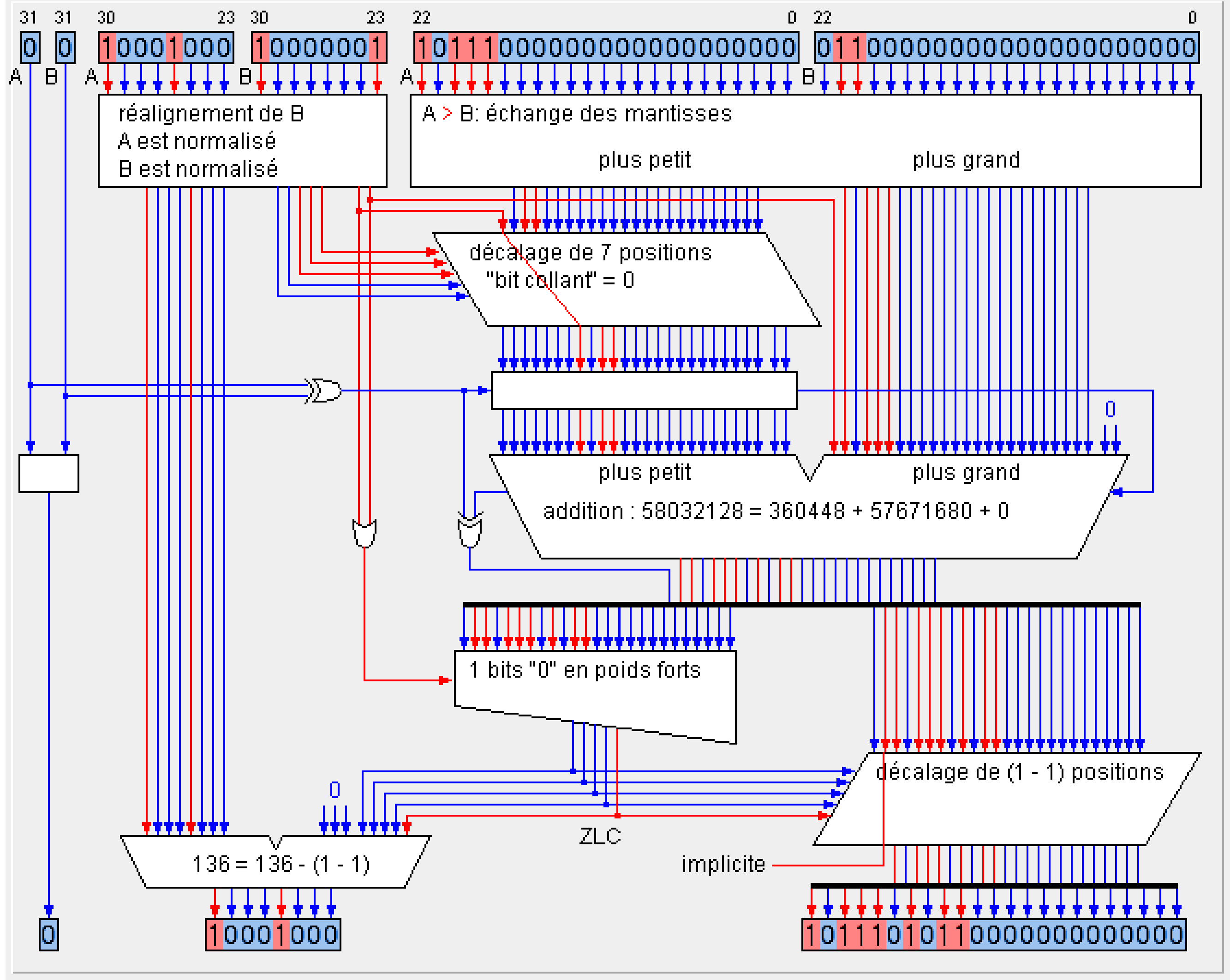
- Alignement des mantisses si les exposants de A et B sont différents
- Addition ou soustraction des mantisses alignées
- Renormalisation de la somme S si elle n'est pas normalisée
- Arrondi de la somme S

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



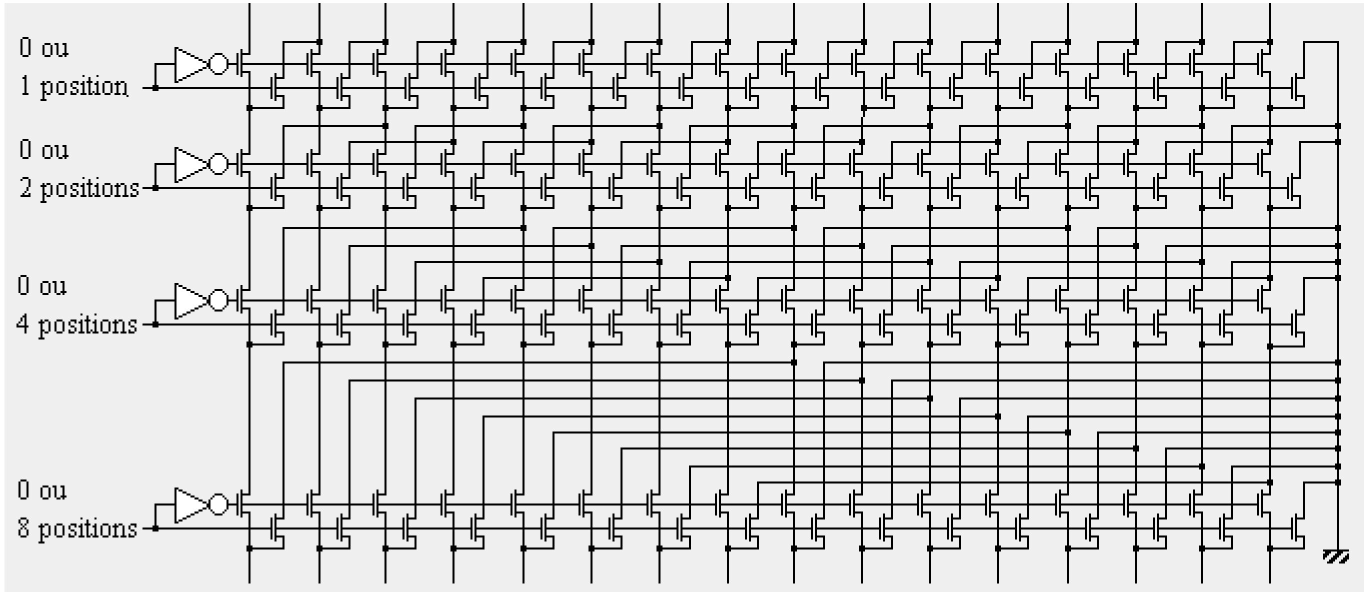
Additionneur/ soustracteur

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 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' en poids forts si le résultat n'est pas 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).



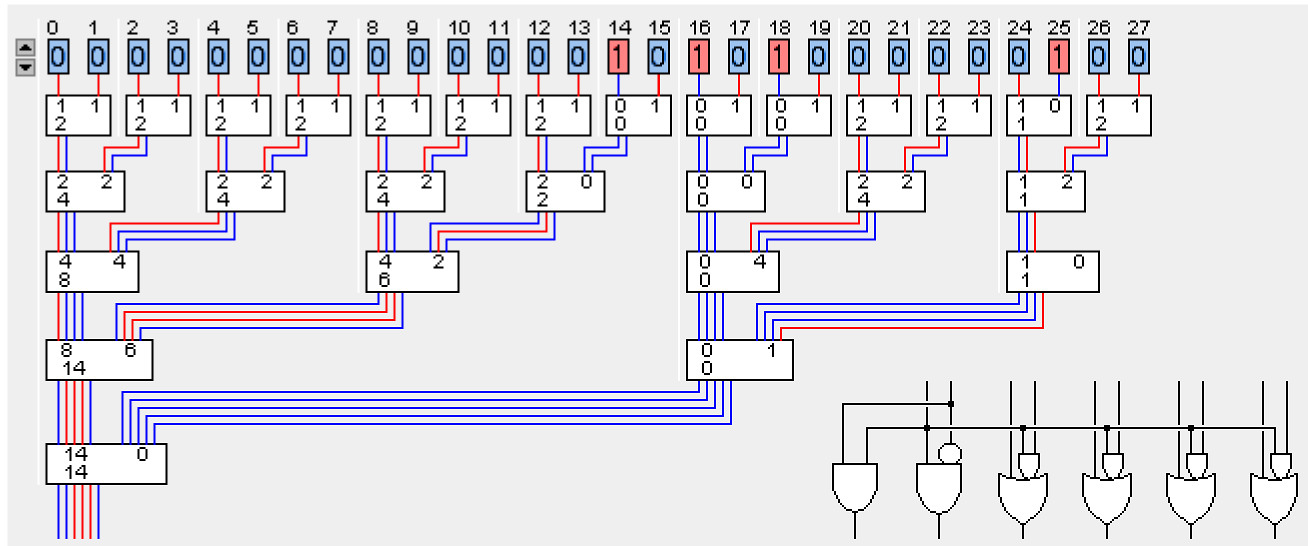
Additionneur de réels rapide

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)

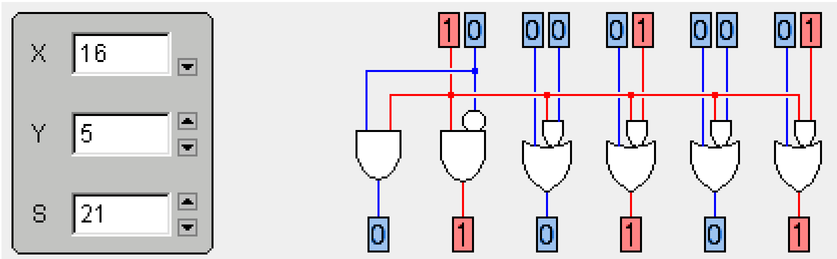
Un arbre binaire décompte le nombre de '0' en poids forts de la chaîne de bits S par dichotomie. Si la taille des sous-chaînes est une puissance de 2, alors il n'y a pas besoin d'additionneur mais seulement de multiplexeurs. En fait seule la taille de la sous-chaîne de gauche doit être une puissance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



Cellule de compteur de zéros en tête

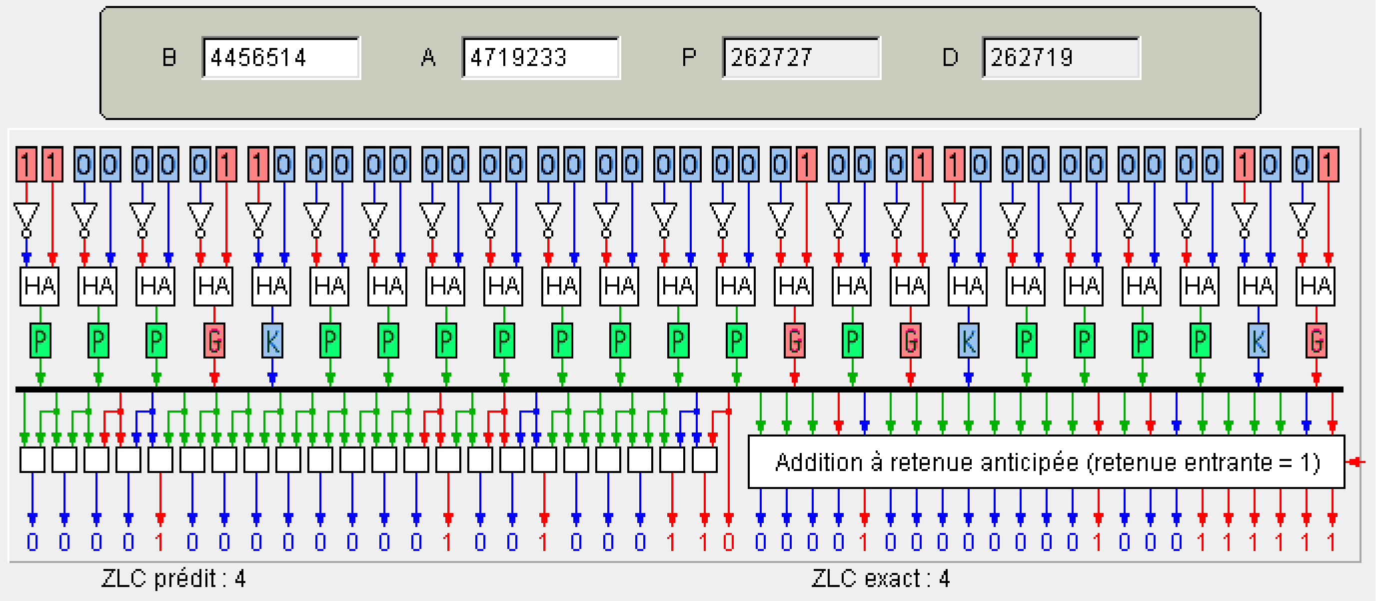
Cette cellule combine le nombre de '0' en poids forts de chacune de deux chaînes de longueur 16 pour obtenir le nombre de '0' en poids forts de la concaténation des deux chaînes.

- **si** X < 16 **alors** S = X **sinon** S = 16 + Y



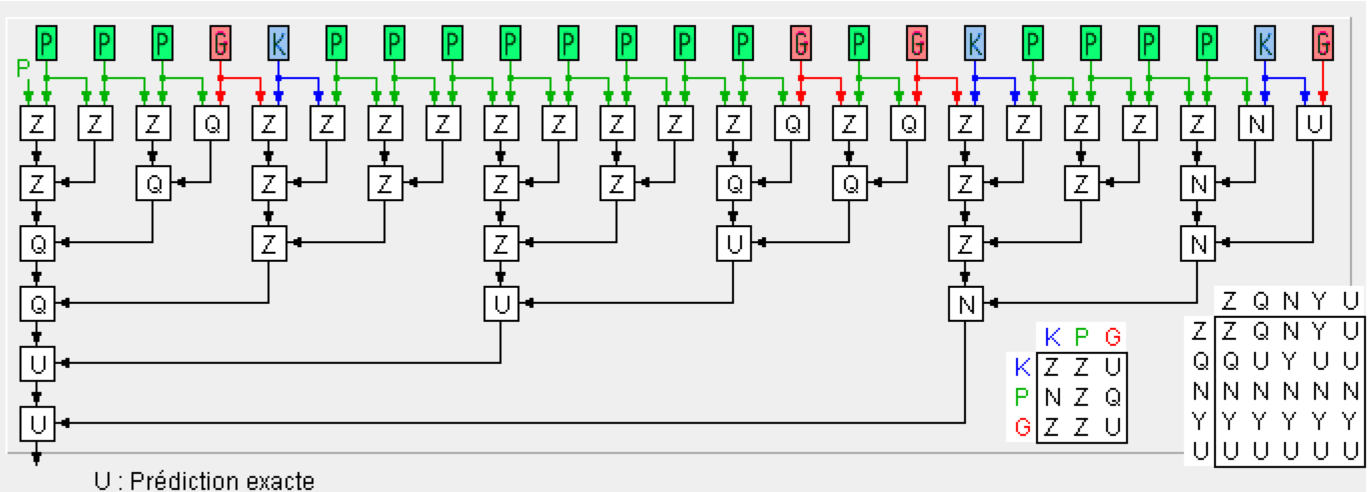
Prédiction de zéros en tête

A partir des mantisses A et B on peut construire une chaîne de bits P ayant (à 1 près) le même nombre de zéros en tête que le résultat de la différence D = 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 décaler ce résultat 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.



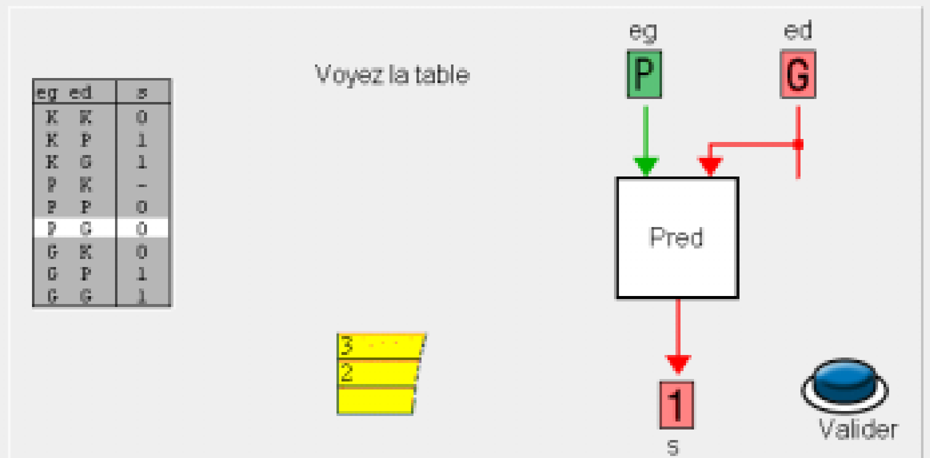
Ajustement de zéros en tête

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



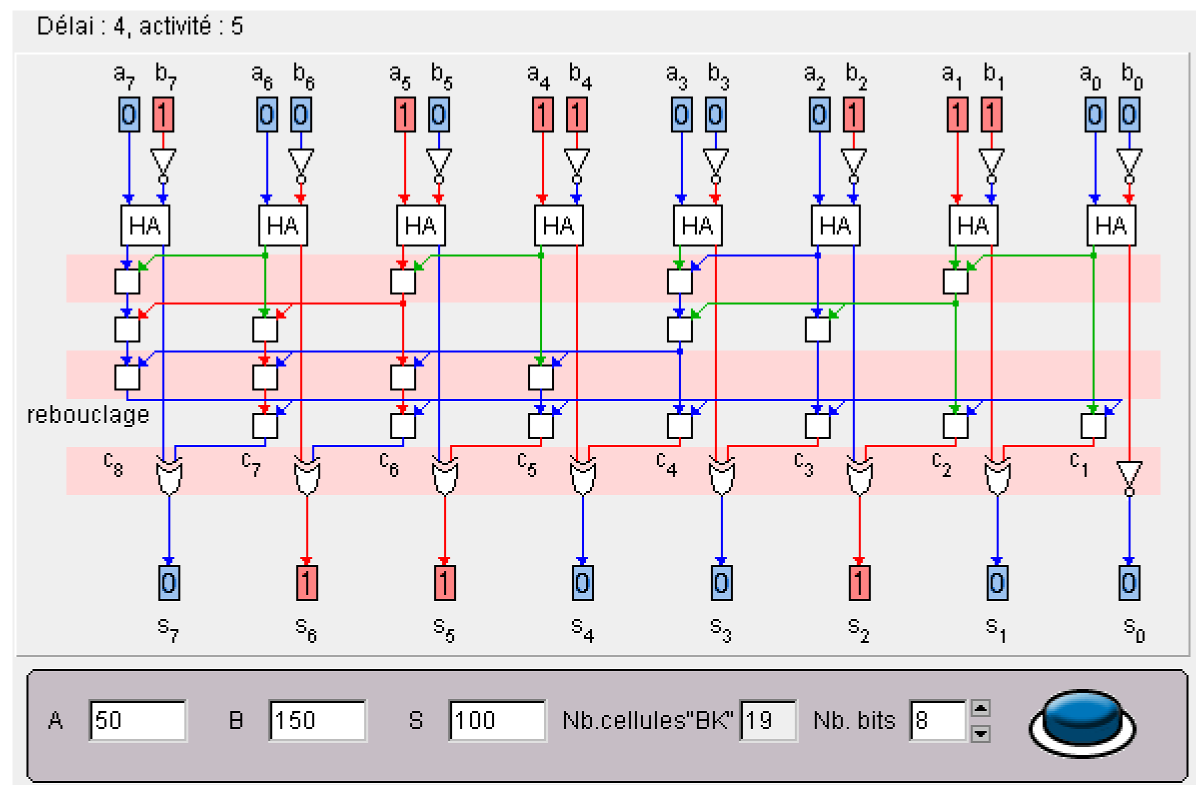
Z indique une chaîne 'K'* 'P'*
Q indique une chaîne 'P'* 'G' 'K'* 'P'* (contenant un seul 'G')
N indique une chaîne commençant par 'P'* 'K'
Y indique une chaîne commençant par 'P'* 'G' 'K'* 'P'* 'K', c'est à dire Q suivi de N.
U marque toute autre chaîne.

Cellule de prédiction La cellule de prédiction des '0' génère un '1' à la fin de la chaîne 'P'* 'G' 'K'* et des '0' à l'intérieur de celle-ci.



Valeur absolue de la différence On a besoin de la valeur absolue de la différence des exposants pour commander le décaleur 1. En cas de soustraction en virgule flottante on calcule également la valeur absolue de la différences des mantisses. Avec quatre légères modifications l'additionneur de Sklanski rend S = │ A – B │, valeur absolue de la différence.

- insérer des inverseurs pour complémenter B (ou bien A)
- modifier la cellule "BK" calculant c_n pour éliminer la valeur 'P' au profit de 'G' pour le signal "rebouclage"
- insérer une ligne de cellules "BK" avec "rebouclage" en entrée droite
- modifier ces dernières cellules pour complémenter logiquement le résultat $A + \overline{B}$ ou bien lui ajouter 1 en fonction du signal "rebouclage".



si $A + \overline{B} \geq 2^n$ **alors** $S = A + \overline{B} + 1$ **sinon** $S = \overline{A + \overline{B}}$

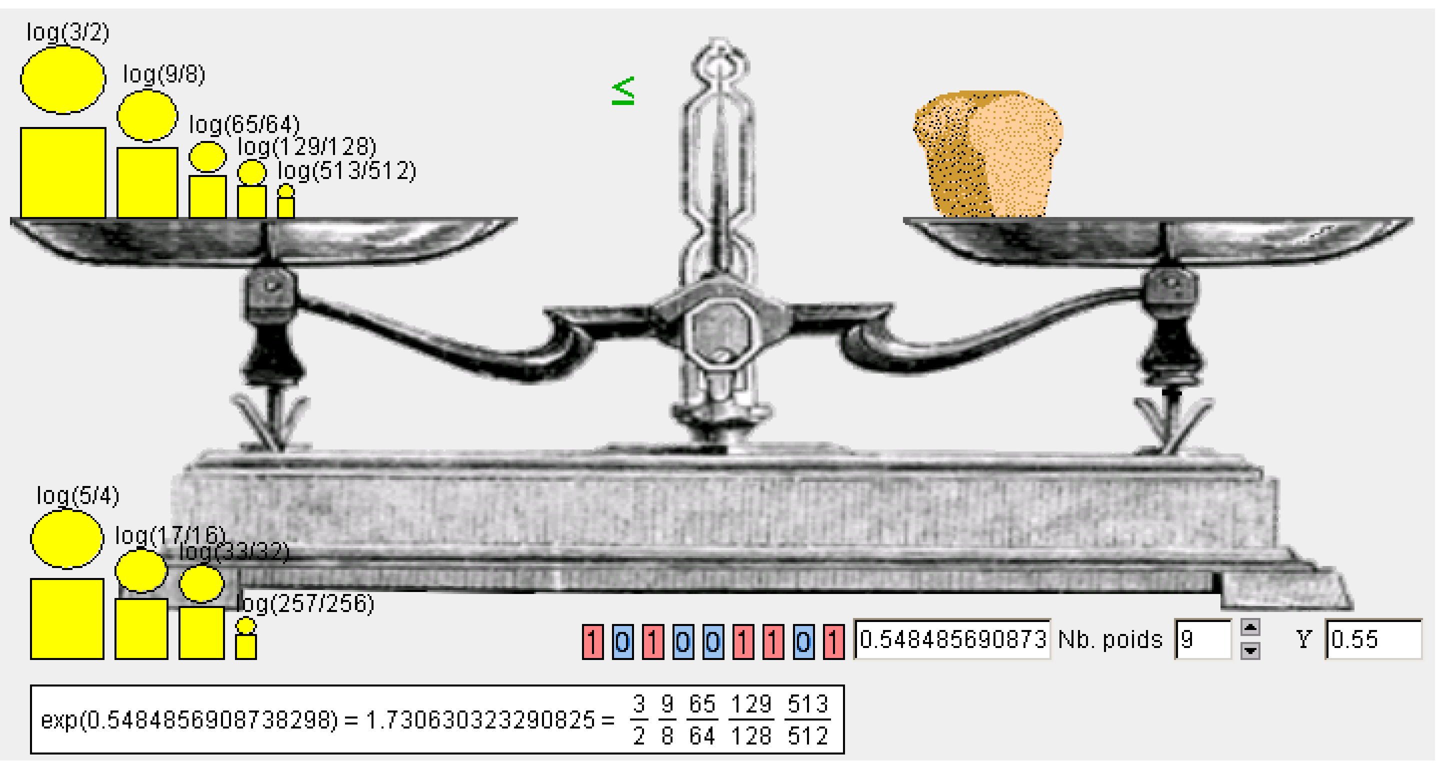
Fonctions élémentaires

Fonctions élémentaires On réalise les fonctions Exponentielle, Logarithme, Sinus, Cosinus et Arc tangente avec des additions/soustractions et des décalages fixes. Le décalage fixe est de coût et délai nul si l'opérateur est câblé. Les additions/soustractions sont sans propagation de retenue, donc à délai constant.

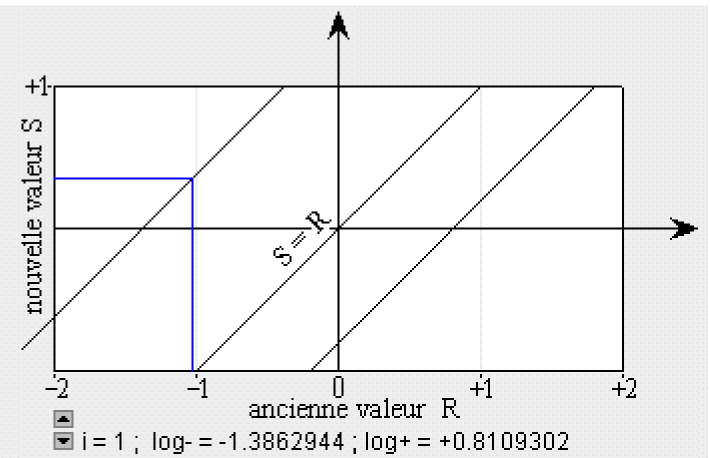
	coût	délai max
addition/soustraction/décalage	n^2	n
lecture en table (ROM)	$n * 2^n$	$\log_2(n)$

Une lecture en table (ROM) serait plus rapide, mais la taille de la table, donc le coût, croît exponentiellement avec le nombre de bits de précision requis. Toutefois le partitionement de la table en réduit la taille.

De la pesée du pain à l'exponentielle On veut calculer exp (Y). On dispose d'une balance, d'un pain dont le poids est justement Y et enfin d'une série de poids de valeur $\log(1 + 2^{-i})$. La pesée nous donne le résultat cherché sous forme de multiplications de rationnels $(2^i + 1) / 2^i$.
Chaque multiplication se réduit en fait à une addition et un décalage.
Un poids sur le plateau de droite (celui du pain) change sa valeur en $-\log(1 - 2^{-i})$, cela facilite le calcul de l'exponentielle

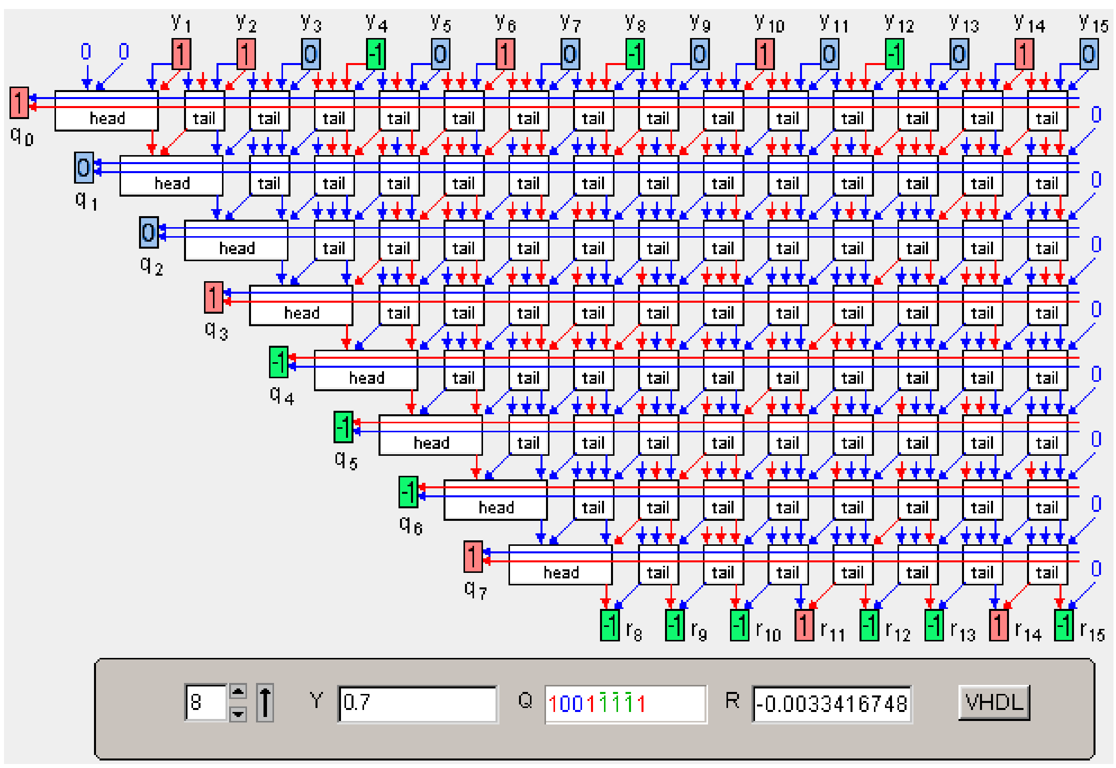


Division sans propagation pour l'exponentielle On remplace la balance par un diviseur "SRT" dont voici le "diagramme de Robertson".



Diviseur "SRT" pour l'exponentielle

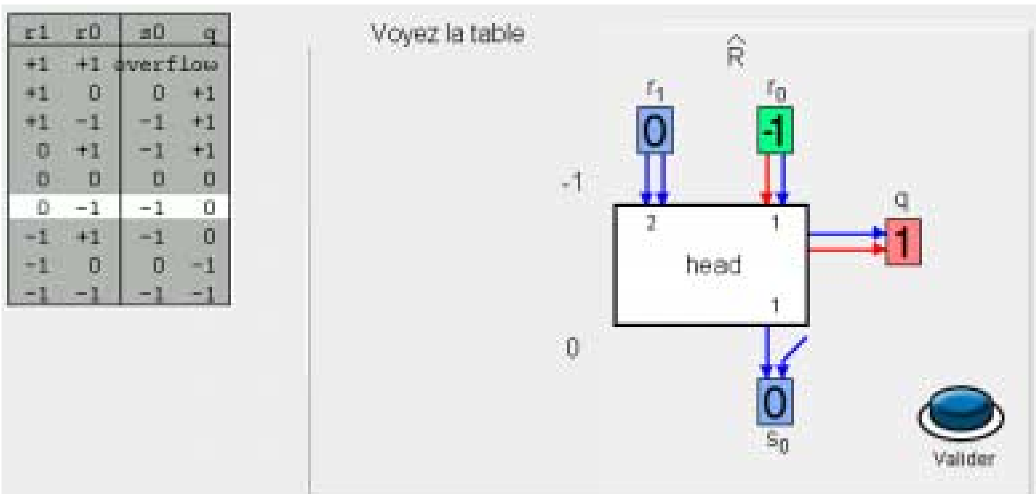
Le dividende Y (en haut) est dans l'intervalle] -1 , +1 [Les constantes $\log(1 + 2^{-i})$ et $-\log(1 - 2^{-i})$ entrant dans les cellules "tail" sont câblées. Il y a donc 4 variantes de la cellule "tail" fonction des valeurs de 2 bits.



Opération d'une tranche de diviseur "SRT" pour l'exponentielle

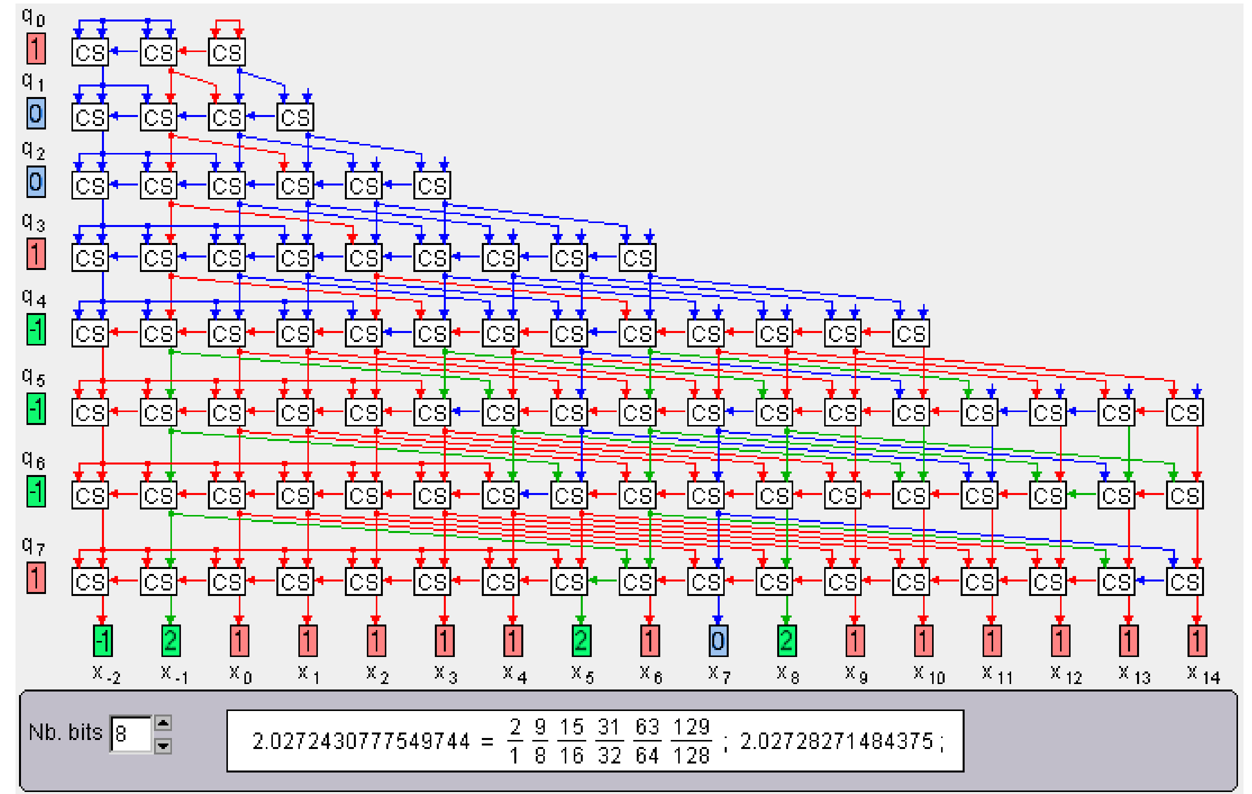
Chaque q_j est choisi par une cellule "head" en fonction de $\hat{R}_j$, somme pondérée des 2 chiffres poids forts r_1 et r_0 de l'écriture de R_j.

- **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 = '$\bar{1}$' ; $s_0 = \hat{R}_j + 2$; $R_{j+1} = R_j + \log(1 + 2^{-j})$ } //addition

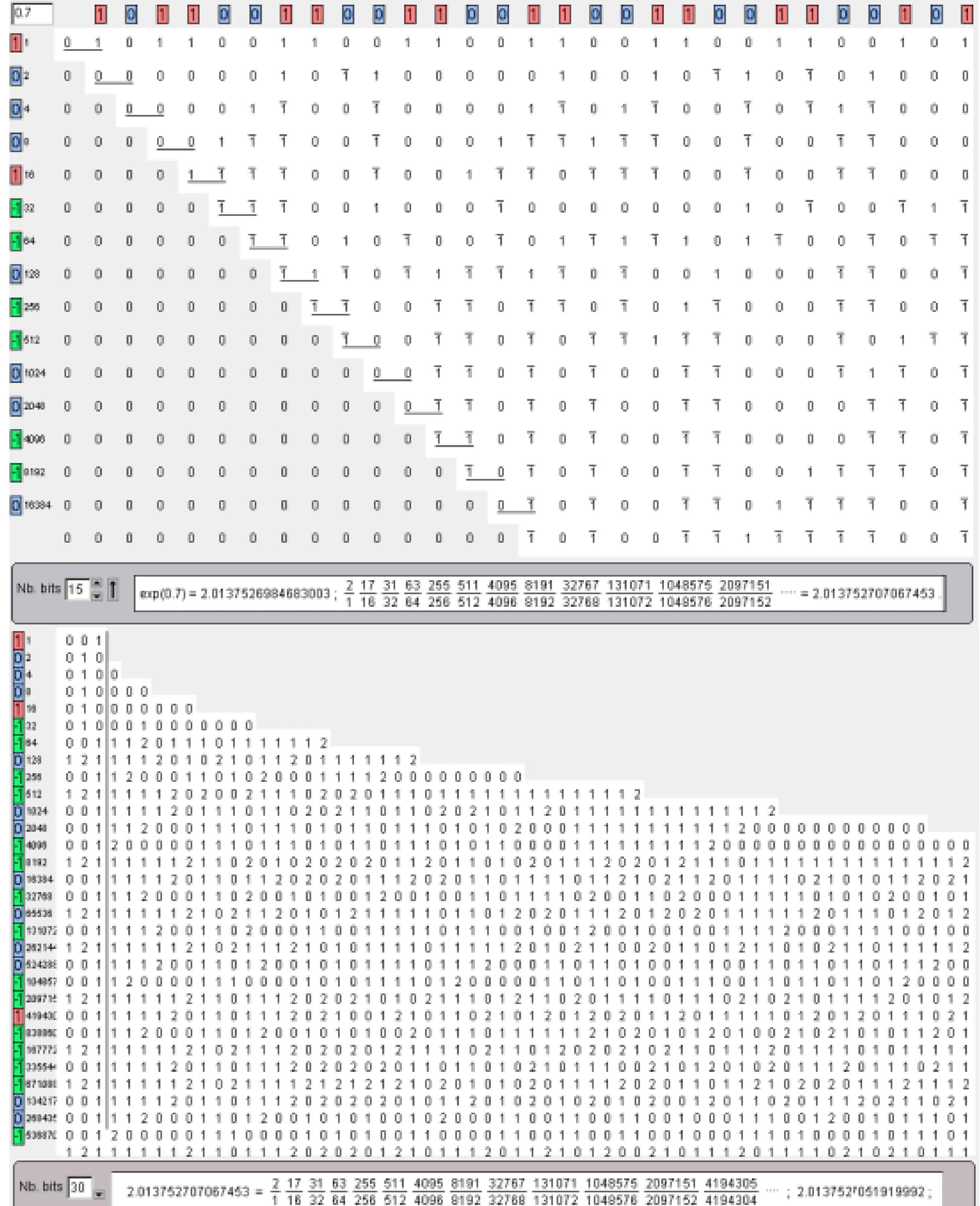


r1	r0	s0	q
+1	+1	overflow	
+1	0	0	+1
+1	-1	-1	+1
0	+1	-1	+1
0	0	0	0
0	-1	-1	0
-1	+1	-1	0
-1	0	0	-1
-1	-1	-1	-1

Suite de multiplications La suite de multiplications conditionnelles par 1, 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 en "CS" sont tronquées à 2 n chiffres, dont 2 avant la virgule. Le troisième chiffre poids fort (tout à gauche) est le signe. Bien que les résultats partiels soient tous positifs ou nuls, exécuter des soustractions en "CS" entraîne un signe qui doit être étendu lors du décalage. Le résultat final (en bas) doit être traduit de "CS" en binaire par une addition (avec propagation). La fenêtre du bas permet de comparer le produit "vraie" des multiplications (sans troncature) au produit avec troncature.



Exemple numérique d'exponentielle Les tableaux ci-dessous montrent les restes partiels du diviseur "SRT" ("BS") puis les produits partiels ("CS") pour le calcul d'exponentielle. La fenêtre du bas permet de comparer la valeur "vraie" de l'exponentielle au produit de multiplications tronquées par $(1 + 2^{-i})$ ou par $(1 - 2^{-i})$.


0.7
Nb. bits 15
exp(0.7) = 2.01375269984683003 ; 2/1 17/16 31/32 63/64 255/256 511/512 4095/4096 8191/8192 32767/32768 131071/131072 1048575/1048576 2097151/2097152 .... = 2.013752707067453
Nb. bits 30
2.013752707067453 = 2/1 17/16 31/32 63/64 255/256 511/512 4095/4096 8191/8192 32767/32768 131071/131072 1048575/1048576 2097151/2097152 4194305/4194304 .... ; 2.013752705191999

Extension du domaine Le circuit précédent fonctionne pour Y dans] -1 , +1 [. Pour l'exponentielle d'un nombre Y quelconque, on écrit $Y = Q*\log(8) + R$, où Q est le quotient entier de la division de Y par log(8) et $R < \log(8) < 1$. Alors $\exp(Y) = 8^{Q} * \exp(R) = 2^{3Q} * \exp(R)$. Comme $\exp(R) < 1$, il est calculable par le circuit précédent.

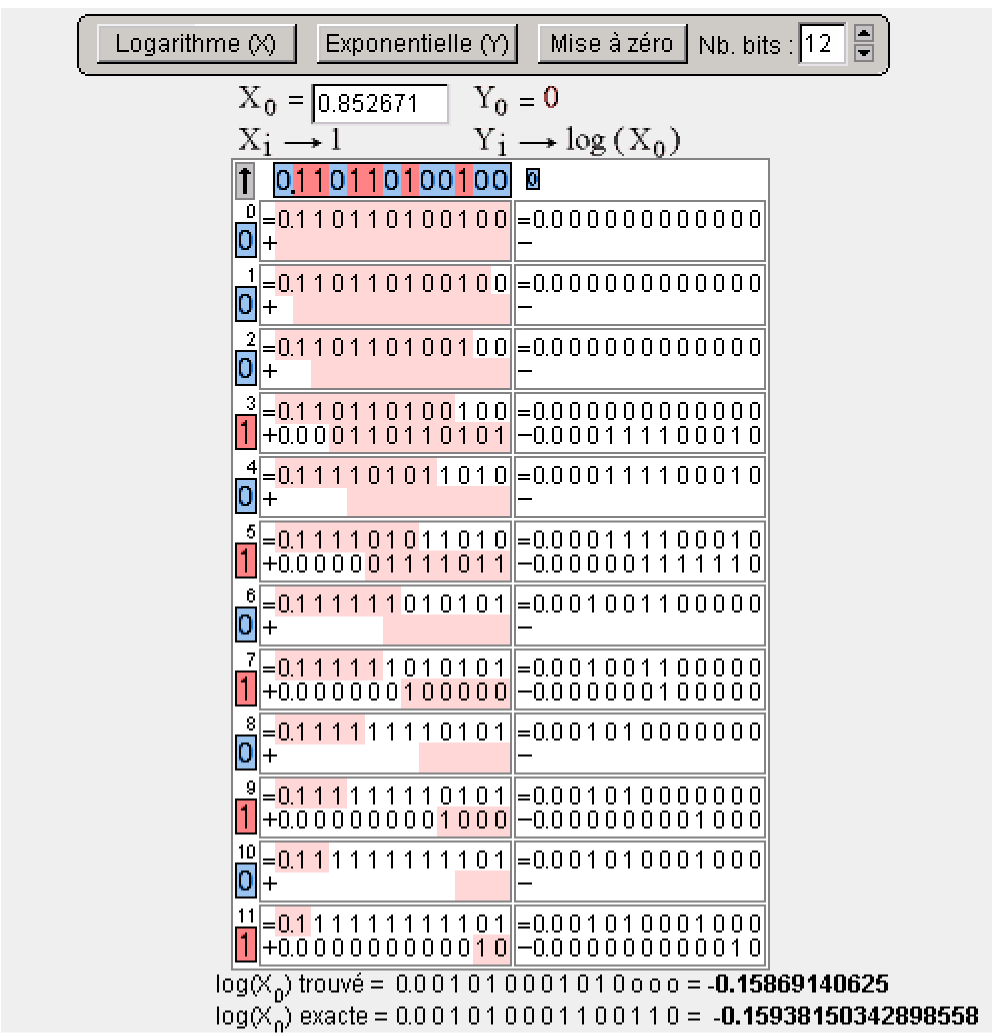
Logarithme et exponentielle

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

$$\sum_{i=0}^{m-1} x_i 2^{-i} \quad \overset{1}{\downarrow} \overset{2}{\uparrow} \qquad\qquad \sum_{i=0}^{m-1} y_i 2^{-i} \quad \overset{3}{\downarrow} \overset{4}{\uparrow}$$

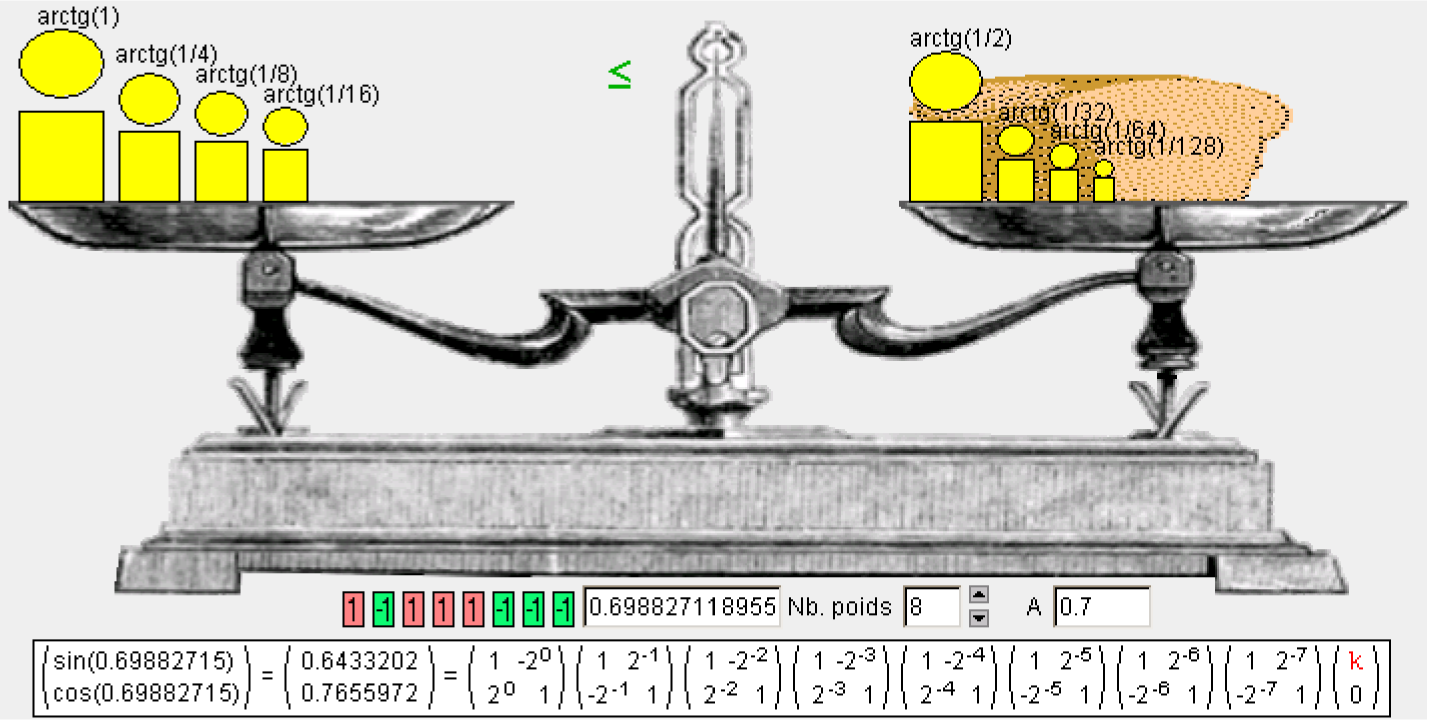
$$\prod_{j=1}^{n} (1 + 2^{-j})^{p_j} \quad \underset{\text{logarithme}}{\overset{\text{exponentielle}}{\overset{5}{\rightleftarrows}}} \quad \sum_{j=1}^{n} p_j \log(1 + 2^{-j})$$

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 $\log(1 + 2^{-i})$ et $-\log(1 - 2^{-i})$ et les chiffres $\in \{$ '$\overline{1}$' , '0' , '1' $\}$. Cette liberté dans le choix des chiffres, qui fera défaut pour Sinus et Cosinus, permet éventuellement de calculer plus vite avec des additions/soustractions sans propagation



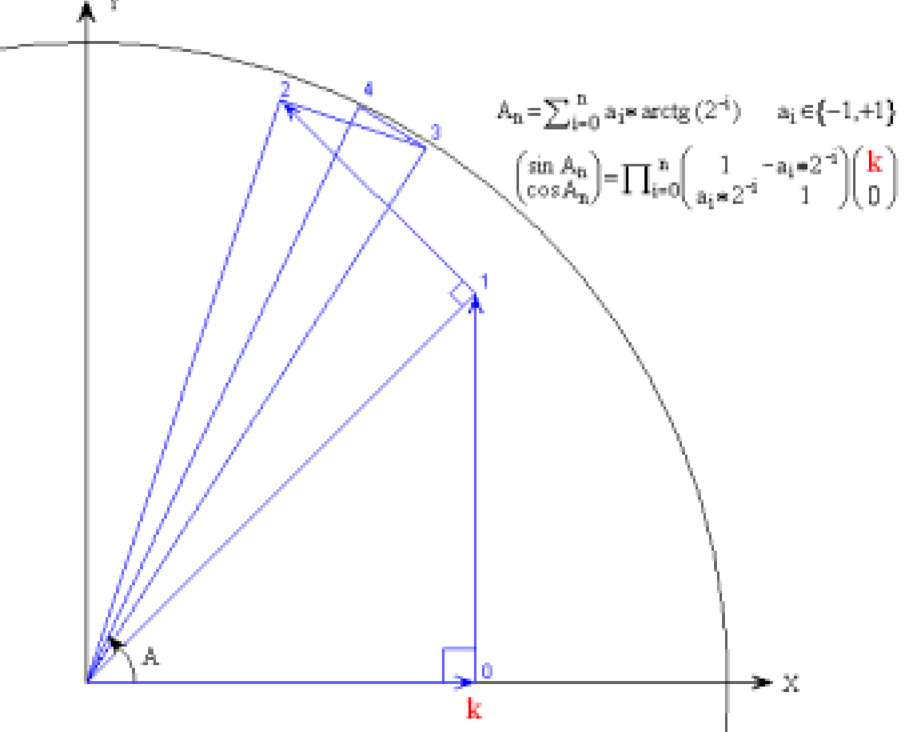
De la pesée du pain au sinus et au cosinus

On veut calculer sin (A) et/ou cos (A). On dispose d'une balance, d'un pain dont le poids est justement A et enfin d'une série de poids de valeur ArcTangente(2^{-i}). Les poids ne peuvent être que sur l'un ou l'autre des deux plateaux.



Calcul de Sinus et Cosinus

Soit un vecteur V_i d'extrémité (X_i, Y_i). Une "pseudoRotation" de V_i d'un angle 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'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 chaque "pseudoRotations" on n'a effectué une addition et une soustraction.



La constante k

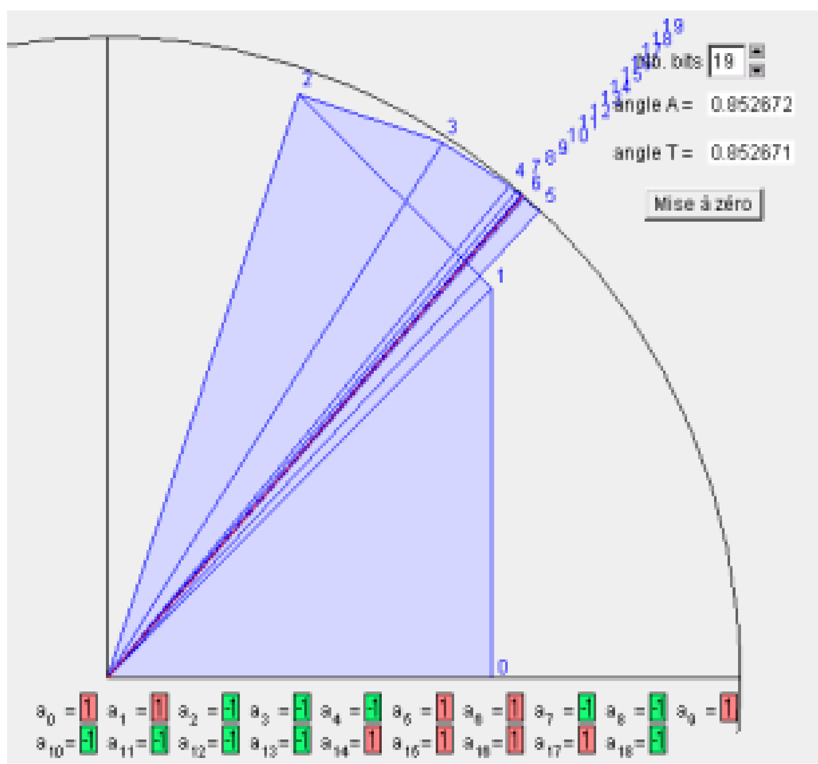
Chaque "pseudoRotation" de arctg(2^{-i}) entraîne un allongement du vecteur de $\sqrt{1+2^{-2i}}$, car ce n'est pas exactement une rotation mais un déplacement de l'extrémité du vecteur

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 arctg(2^{-i}) utilise des chiffres ∈ { '$\overline{1}$', '1' } .

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

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 arctg (2^{-i}).



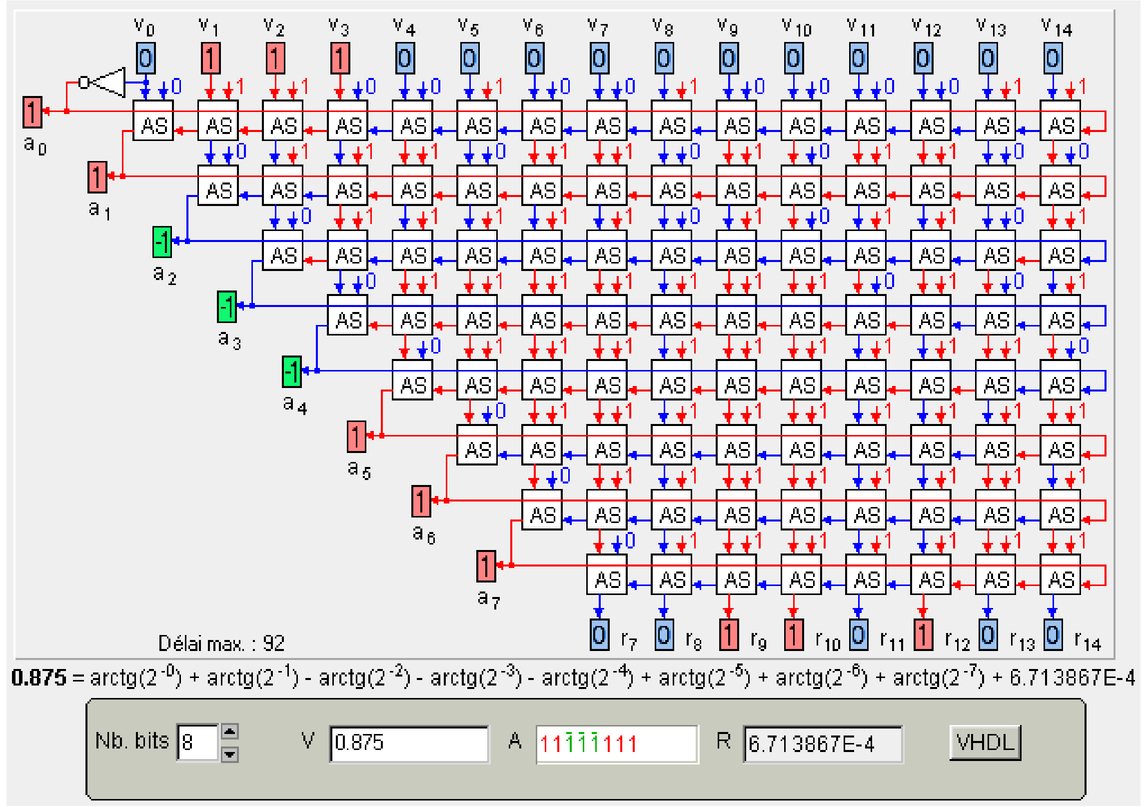
"Diagramme de Robertson" pour sinus et cosinus

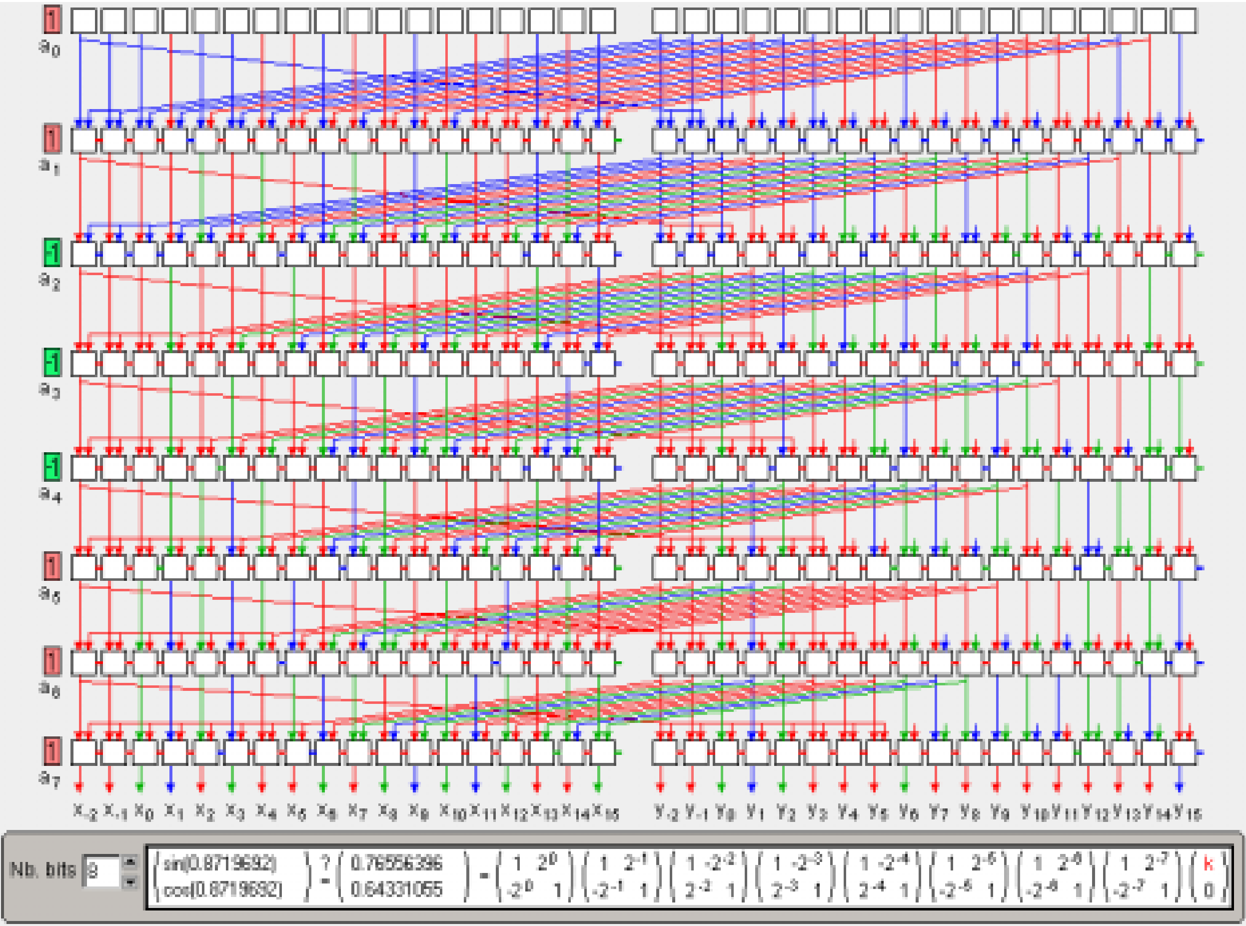
Le "diagramme de Robertson" montre que l'itération de conversion de l'angle dans la base arctg(2^{-i}) peut être choisie comme suit:

si R ≥ 0 **alors** { S = R – arctg(2^{-i}) ; a_i = '1' ; } **sinon** { S = R + arctg(2^{-i}) ; a_i = '$\overline{1}$' ; }.

Diviseur "sans restauration" pour sinus et cosinus

L'angle A (en haut du diviseur) est dans l'intervalle [-1,743.. +1,743...]. Les bits constants entrant dans les cellules "AS" sont câblés. Les opérations sont choisies par le signe des restes partiels R précédent et par y_0 pour la première opération .





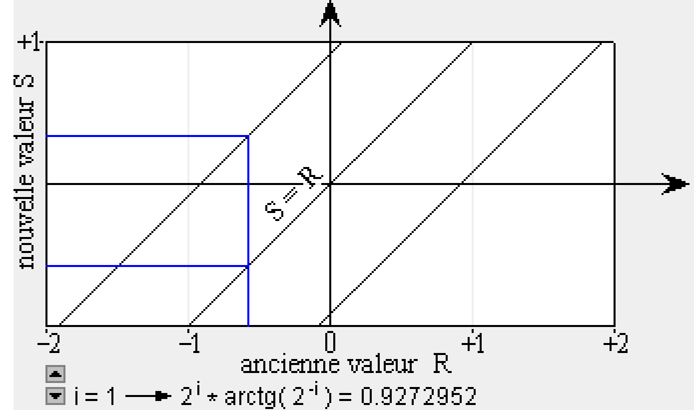
Opérateur de "PseudoRotation" pour sinus et cosinus

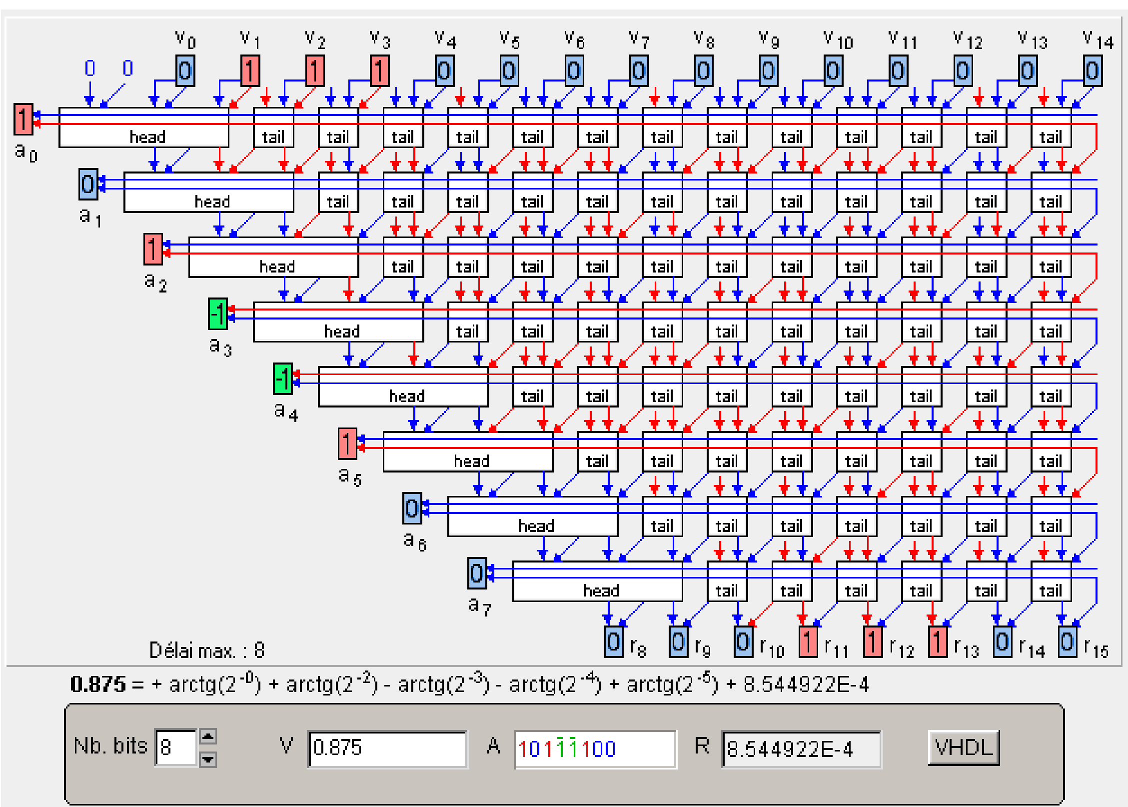
Si les "PseudoRotations" utilisent des additions/soustractions sans retenue (ici en "CS") alors le délai est linéaire avec le nombre d'étages. Le premier étage n'effectue pas vraiment une addition, puisque $X_0 = k$, $Y_0 = 0$.
si a_0 = '1' **alors** { $X_1 = k$; $Y_1 = -k$ } **sinon** { $X_1 = k$; $Y_1 = k$ }.
Pour les étages suivants
si a_i = '1' **alors** { $X_{i+1} = X_i - Y_i * 2^{-i}$; $Y_{i+1} = Y_i + X_i * 2^{-i}$ }

si a_i = '$\bar{1}$' **alors** { $X_{i+1} = X_i + Y_i * 2^{-i}$; $Y_{i+1} = Y_i - X_i * 2^{-i}$ }.
Pour simplifier le dessin, la nappe $X * 2^{-i}$ ne figure pas.

"Diagramme de Robertson" pour CORDIC à "double rotation"

On utilise une approximation $\hat{R}$ du reste R pour déterminer la rotation:
si $\hat{R} > 0$ **alors** a_i = '1' ; **si** $\hat{R} = 0$ **alors** a_i = '0' ; **si** $\hat{R} < 0$ **alors** a_i = '$\bar{1}$' ;
On constate sur le diagramme que l'approximation peut être grossière.





CORDIC à "double rotation"

Ce diviseur a les mêmes cellules "head" et "tail" que le diviseur "SRT". Pour s'accommoder du '0', l'angle A est préalablement divisé par 2 et les "pseudoRotation" sont doublées. Grâce à cela l'allongement reste le même ($2*\sqrt{1+4^{-i}}$) quelque soit a_i.

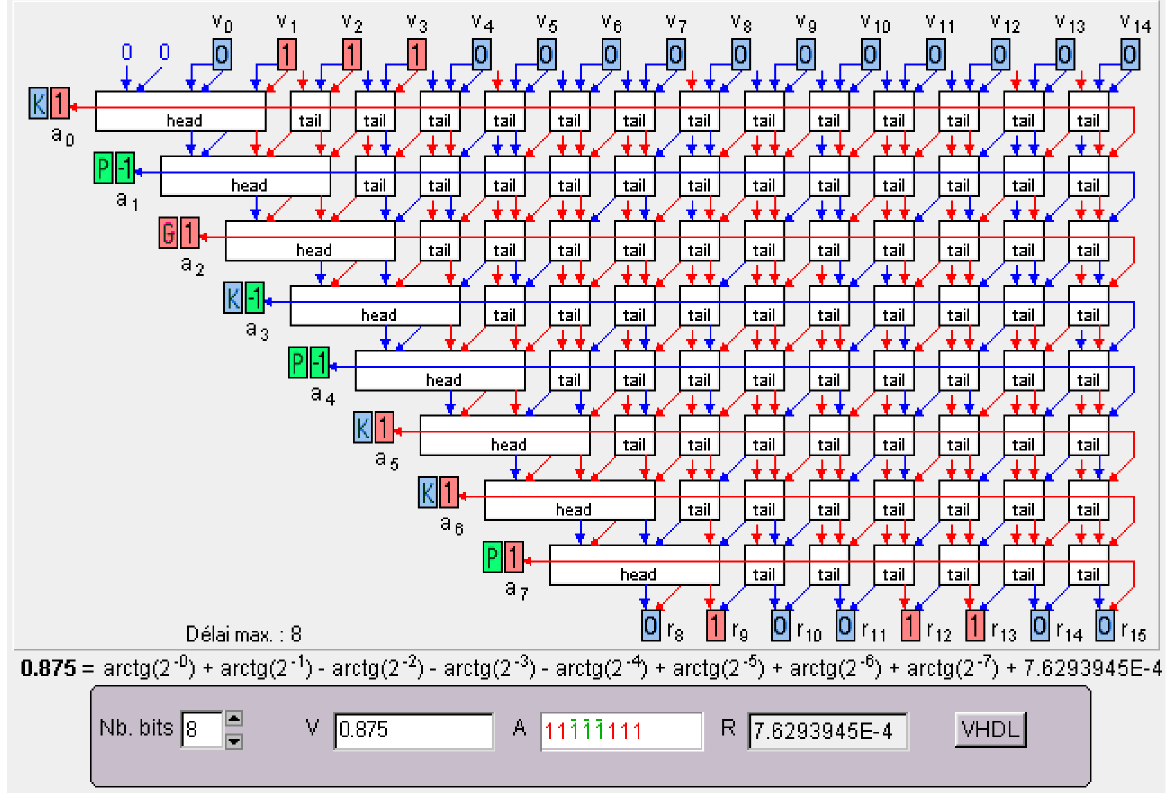
- **si** a_i = '$\overline{1}$' **alors** { rotation de ($arctg(2^{-i})$) puis rotation de ($arctg(2^{-i})$) } ;
- **si** a_i = '0' **alors** { rotation de ($arctg(2^{-i})$) puis rotation de ($-arctg(2^{-i})$) } ;
- **si** a_i = '1' **alors** { rotation de ($-arctg(2^{-i})$) puis rotation de ($-arctg(2^{-i})$) } ;

CORDIC à "double division"

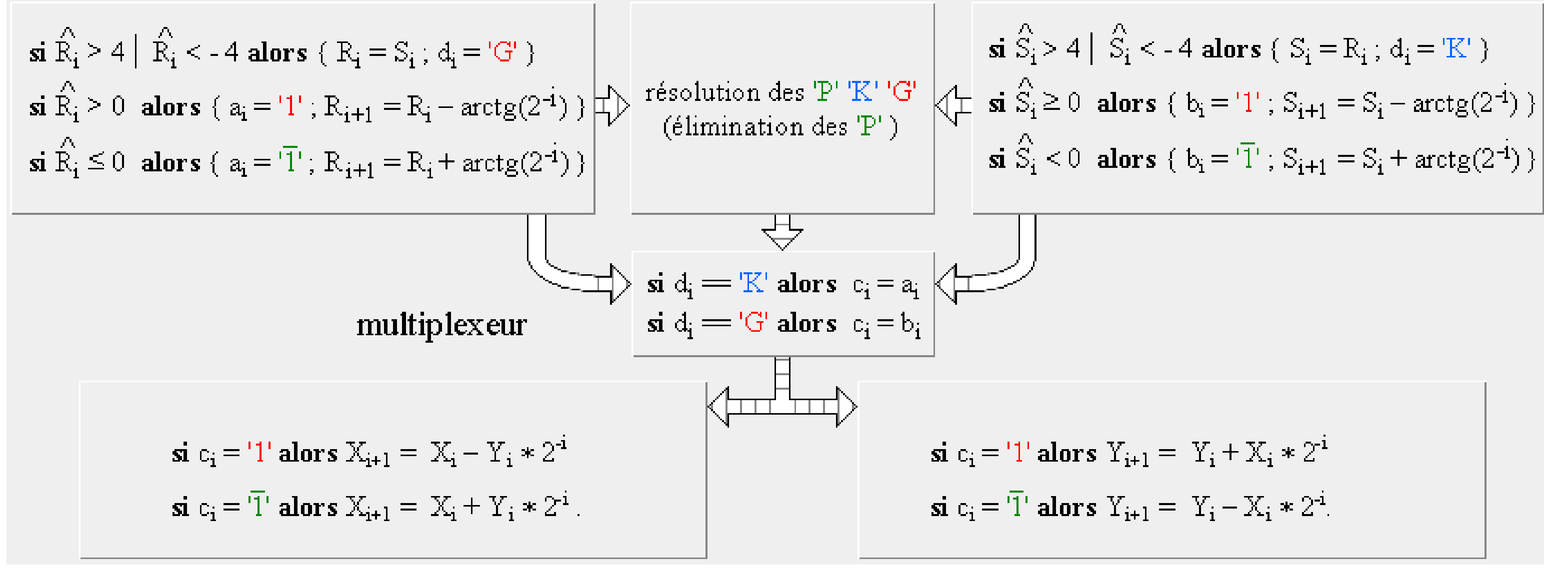
La "double rotation" est coûteuse: elle double le matériel de la rotation et probablement le délai. La "double division" est plus astucieuse. Elle utilise deux diviseurs fonctionnant simultanément avec des cellules "head" légèrement modifiées.

"head" de diviseur1	"head" de diviseur2
si $\hat{R} > 0$ **alors** { $\hat{S} = \hat{R} - 2$; a_i = '1' ; }	**si** $\hat{R} \geq 0$ **alors** { $\hat{S} = \hat{R} - 2$; a_i = '1' ; }
si $\hat{R} \leq 0$ **alors** { $\hat{S} = \hat{R} + 1$; a_i = '$\overline{1}$' ; }	**si** $\hat{R} < 0$ **alors** { $\hat{S} = \hat{R} + 1$; a_i = '$\overline{1}$' ; }

Il est clair que lorsque $\hat{R} = 0$, le diviseur1 parie que $R \leq 0$ et diviseur2 que $R \geq 0$. L'un des deux au plus se trompe et en conséquence peut déborder, avant ou en même temps que l'apparition du $\hat{R} = 0$ suivant. Au débordement on saura que le seul résultat correct est celui de l'autre diviseur.Seul le diviseur1 est affiché par l'applet ci-dessous.



Les "heads" des 2 diviseur détectent chacune le débordement pour afficher ensemble un indicateur à 3 valeurs:
'K' le chiffre de diviseur1 est correct (divisieur2 déborde), 'G' le chiffre de diviseur2 est correct (diviseur1 déborde), 'P' propager l'indicateur suivant (pas de débordement). Chaque fois qu'un diviseur déborde, il importe le reste partiel R de l'autre diviseur. L'indication "Overflow" sur une cellule "head" signale un débordement fatal de diviseur1 et de diviseur2 simultanément.
La propagation est semblable à la propagation de retenue de l'addition.



Sinus, Cosinus et Arc Tangente

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.

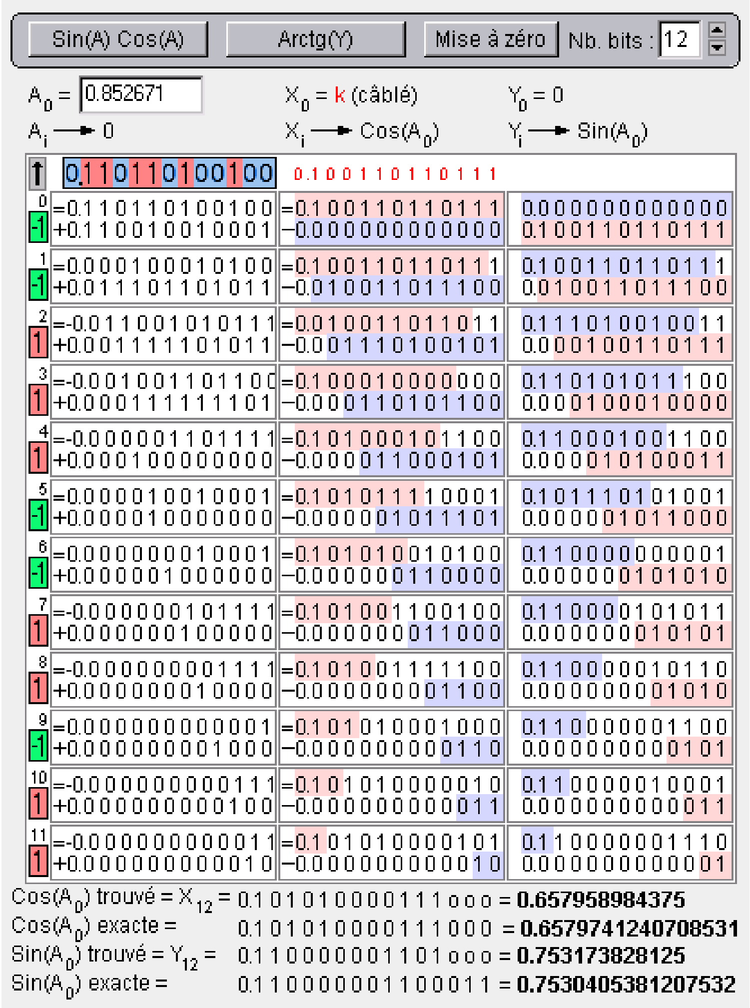
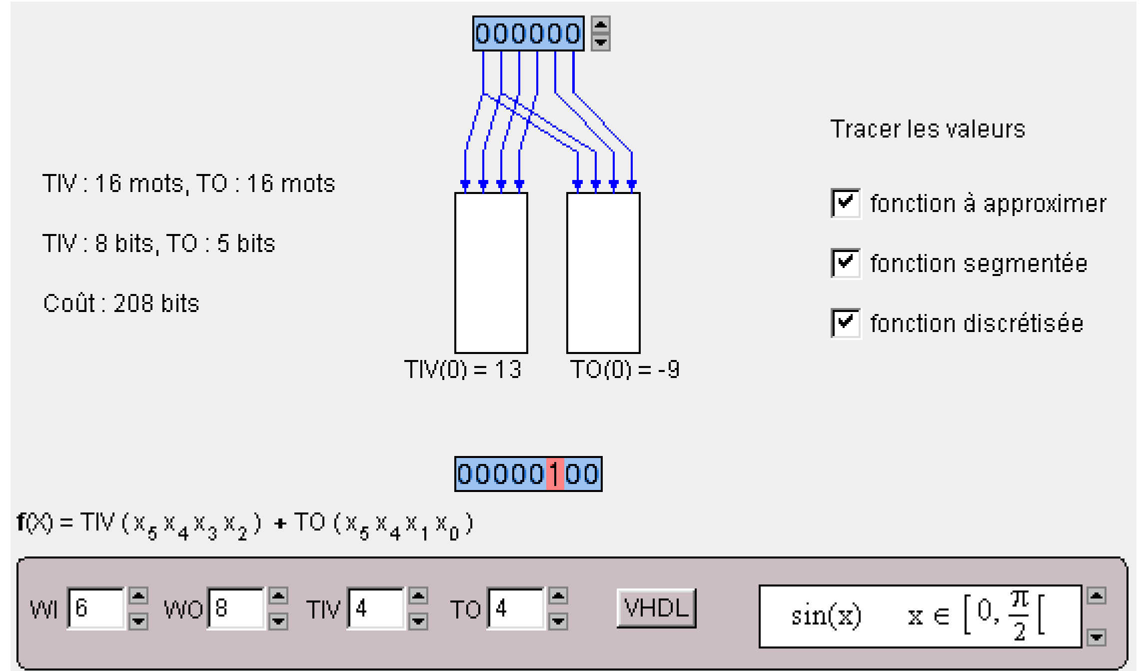


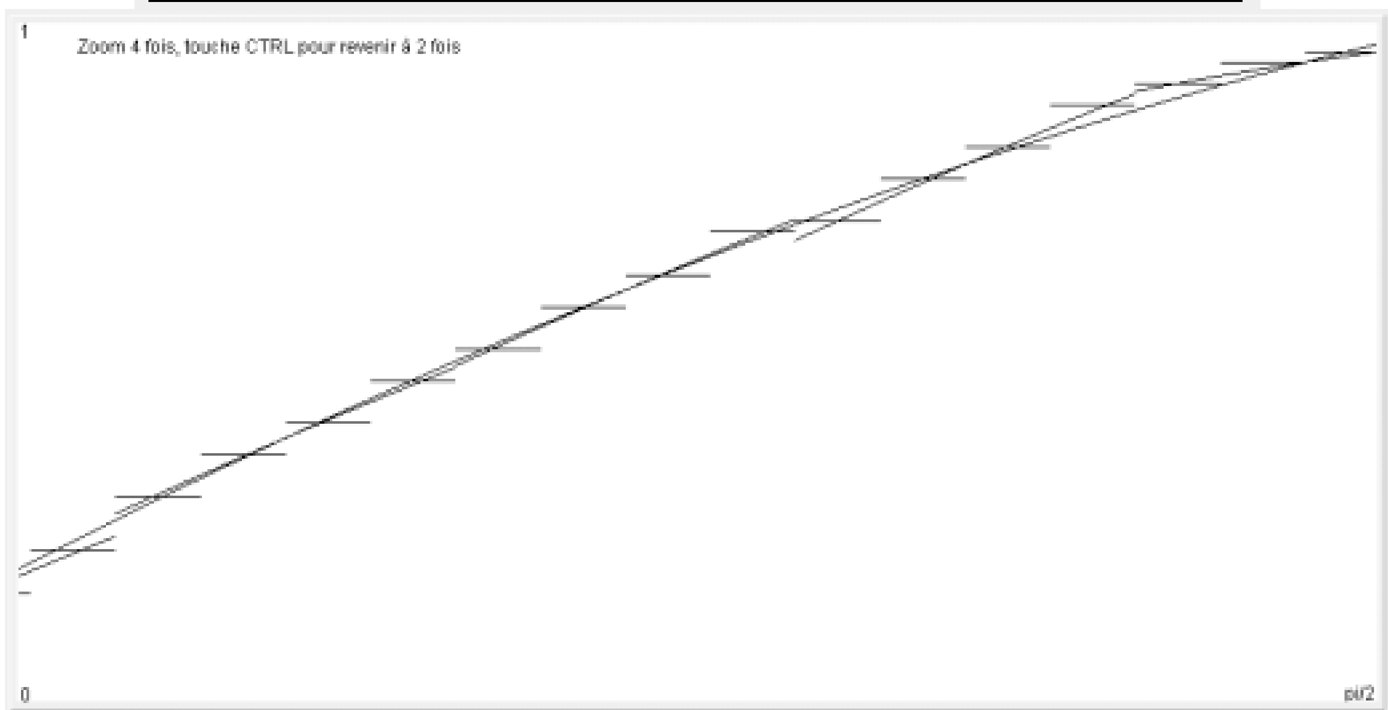
Table Bipartite

- $\sin(x) \quad x \in \left[0, \frac{\pi}{2}\right[$
- $\sin(x) \quad x \in \left[0, \frac{\pi}{4}\right[$
- $2^x - 1 \quad x \in [0, 1[$
- $\log_2(x) \quad x \in [1, 2[$
- $\frac{1}{x} - \frac{1}{2} \quad x \in [1, 2[$
- $\sqrt{x} - 1 \quad x \in [1, 4[$
- $\frac{1}{\sqrt{x}} - \frac{1}{2} \quad x \in [1, 4[$
- $\sqrt[3]{x} - 1 \quad x \in [1, 8[$

On peut précalculer puis stocker les valeurs d'une fonction en table, cependant la taille de cette table croît très vite avec la précision requise, ce qui limite en pratique cette approche. Pour des fonctions continues avec une faible variation de pente (cas de nombreuses fonctions), on peut ne stocker que la position de segments dans une première table "TIV", et le ou les segment(s) dans une autre table "TO".

L'applet ci-dessous rempli les tables pour les fonctions et les intervalles listés à gauche et permet d'observer graphiquement le résultat.
Pour ajouter des fonctions il faudrait modifier le source Java.
La valeur des tables est imprimable en VHDL.

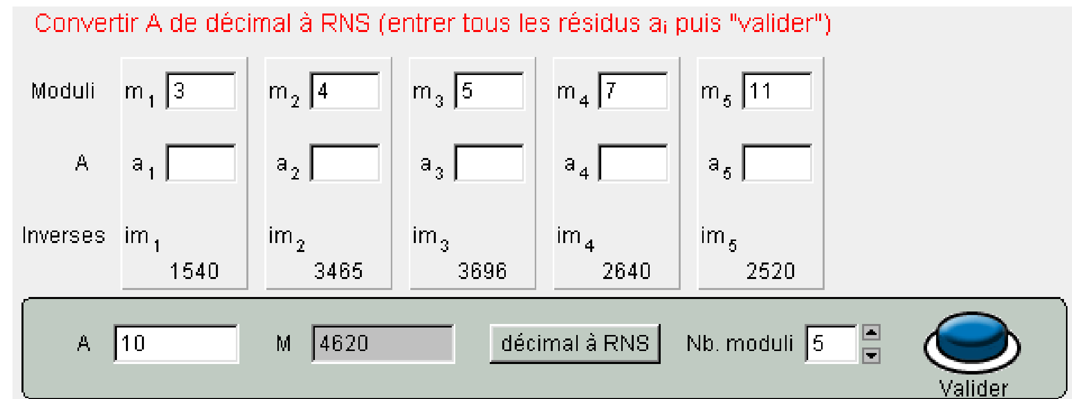




Opérateurs Modulaire

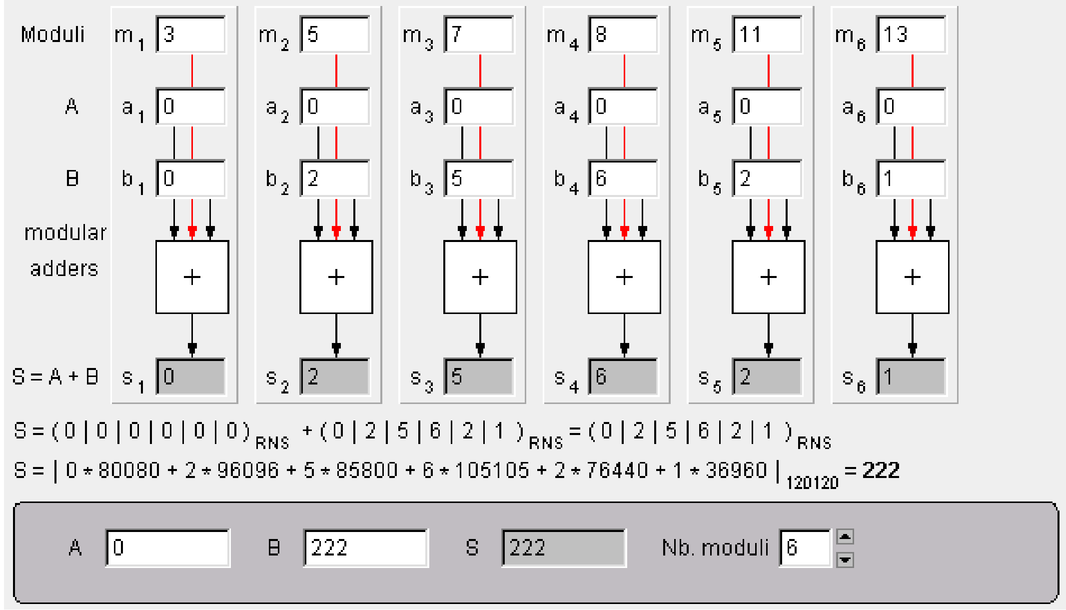
Représentation modulaire

Soit un ensemble { m_1, m_2, m_3, ... m_n} de n constantes entières premières entre elles appelées moduli et soit M le produit de ces constantes, $M = m_1 * m_2 * m_3 * ... * m_n$.
Soit A une variable entière inférieure à M.
A peut s'écrire ($a_1 | a_2 | a_3 | .... | a_n)_{RNS}$ où a_i = A modulo m_i (résidu).
Cette définition dit comment calculer les a_i à partir de A.
Il est également possible de calculer A à partir des a_i en utilisant une autre série de constantes { im_1, im_2, im_3, ... im_n} précalculées appelées inverses modulo M des premières. $A = | a_1 * im_1 + a_2 * im_2 + a_3 * im_3 + ... a_n * im_n |_{\text{modulo } M}$
Ce calcul est démontré par le "théorème du reste chinois".
Vérifiez que vous vous maîtrisez cette représentation en en convertissant A de "décimal à RNS" ou de "RNS à décimal".



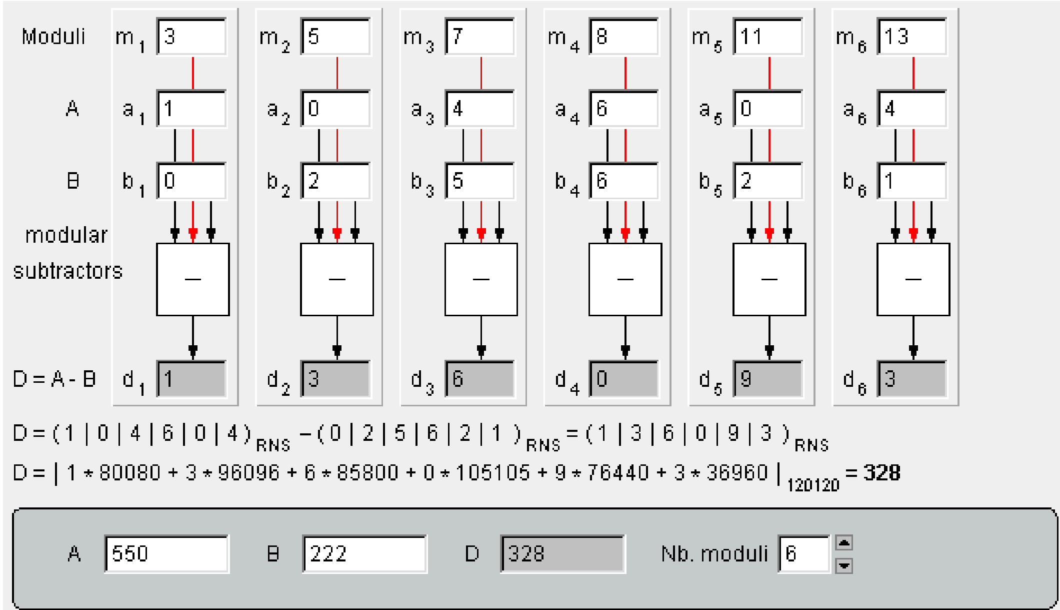
Addition modulaire

L'addition modulaire utilise n <u>petits</u> additionneurs calculant <u>simultanément</u> toutes les sommes $s_i = | a_i + b_i |_{\text{modulo } m_i}$.

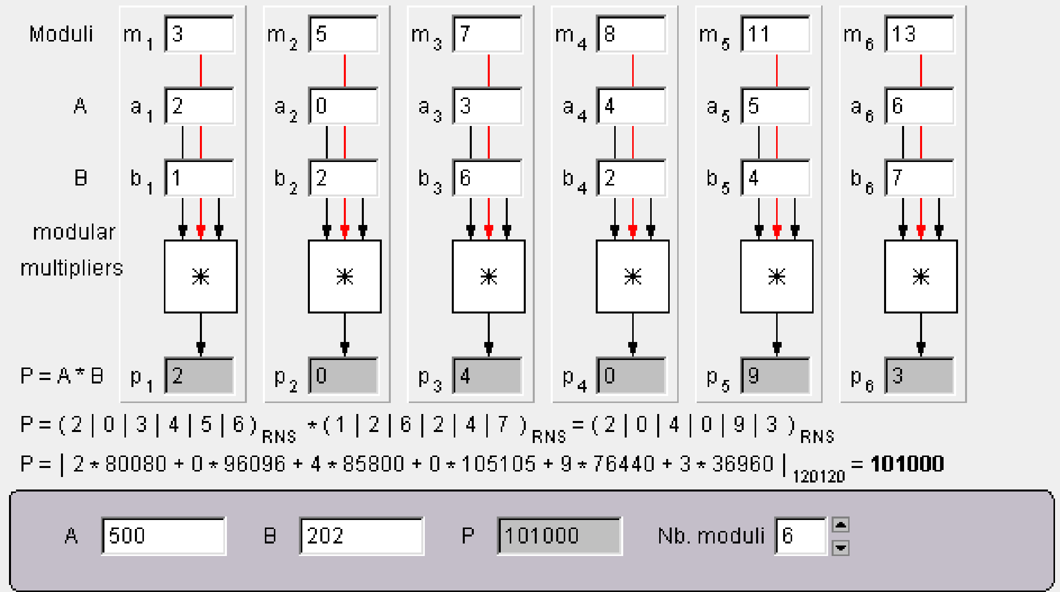


Soustraction modulaire

La soustraction modulaire utilise n <u>petits</u> soustracteurs calculant <u>simultanément</u> toutes les différences $d_i = | a_i + m_i - b_i |_{\text{modulo } m_i}$.



Multiplication modulaire La multiplication modulaire utilise n petits multiplieurs calculant simultanément tous les produits $p_i = | a_i * b_i |_{\text{modulo}} \ m_i$.

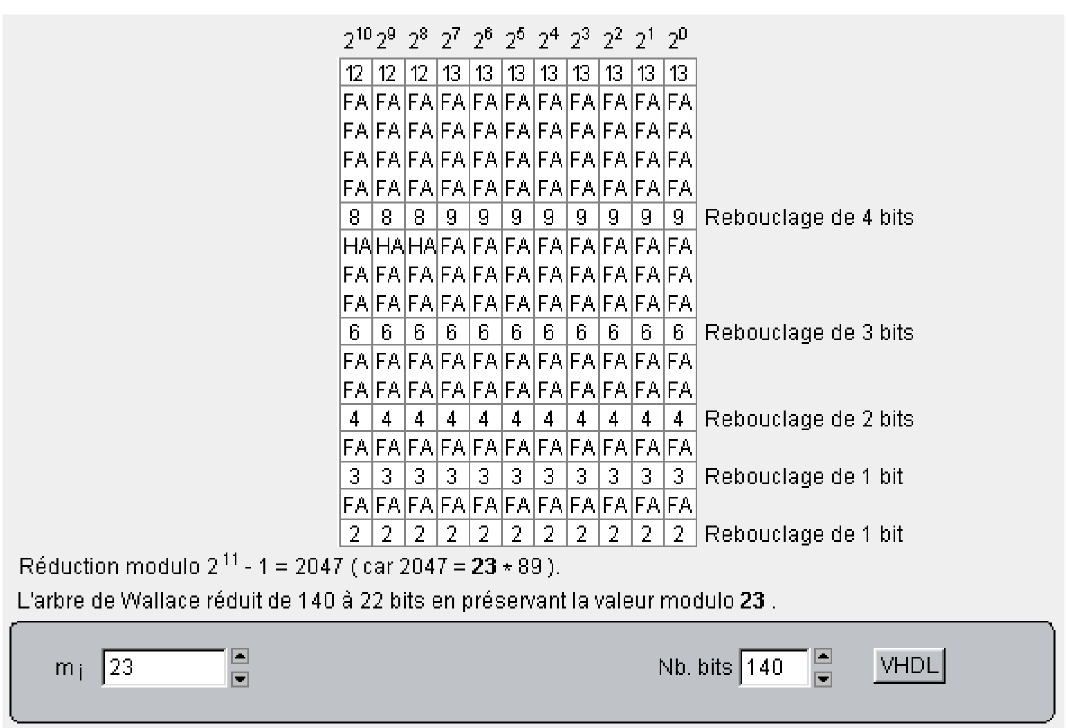


Conversion en RNS La conversion d'une variable A binaire en RNS consiste à trouver les a_i = A modulo m_i qui sont les restes de la division de A par m_i. Tous ces calculs peuvent se faire simultanément, cependant utiliser des diviseurs n'est pas la meilleure méthode.

- le reste modulo 2^n est immédiat,
- le reste modulo $2^n - 1$ ne demande que des additions,
- le reste modulo $2^n + 1$ demande des additions et soustractions.

autrement on se ramène à celui des deux deniers qui a le n le plus petit. Des arbres d'additionneurs (arbres de Wallace) réduisent A à la somme de 2 entiers de n bits sans changer le reste modulo m_i.

Les conventions graphiques sont celles de la réduction des produits partiels.



Réducteur modulo $2^n - 1$

L'applet ci-dessous réduit 64 bits en 6 bits en en conservant la valeur modulo 63 ($63 = 2^6 - 1$) . En sortie zéro a deux notations: "000 000" ou encore "111 111"

Délai : 28, activité : 45 1416215070652579123 mod 63 = 13

Additionneur modulo $2^n - 1$

L'additionneur "à rebouclage de retenue" ci-dessus offre deux avantages : il fonctionne correctement et il est simple, et aussi deux inconvénients: il est lent et difficilement testable, et ce pour la même raison : il y a pour la valeur zéro deux états stables.

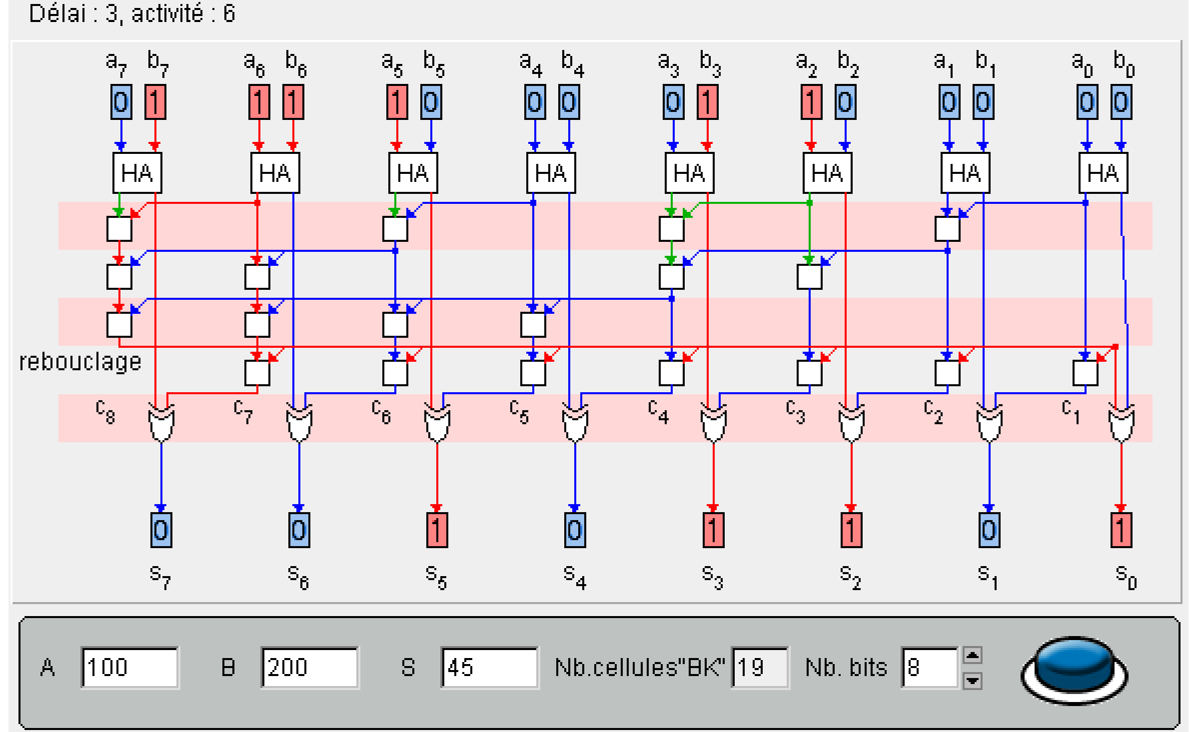
Un additionneur rend spontanément une somme modulo 2^n. Avec une légère modification l'additionneur de Sklanski rend une somme modulo $2^n - 1$.

- **si** $A + B < 2^n - 1$ **alors** $S = A + B$;
- **si** $A + B \geq 2^n - 1$ **alors** $S = | A + B + 1 |_{\text{modulo } 2^n}$;

Dans les deux cas on a $S = | A + B |_{\text{modulo } (2^n - 1)}$.

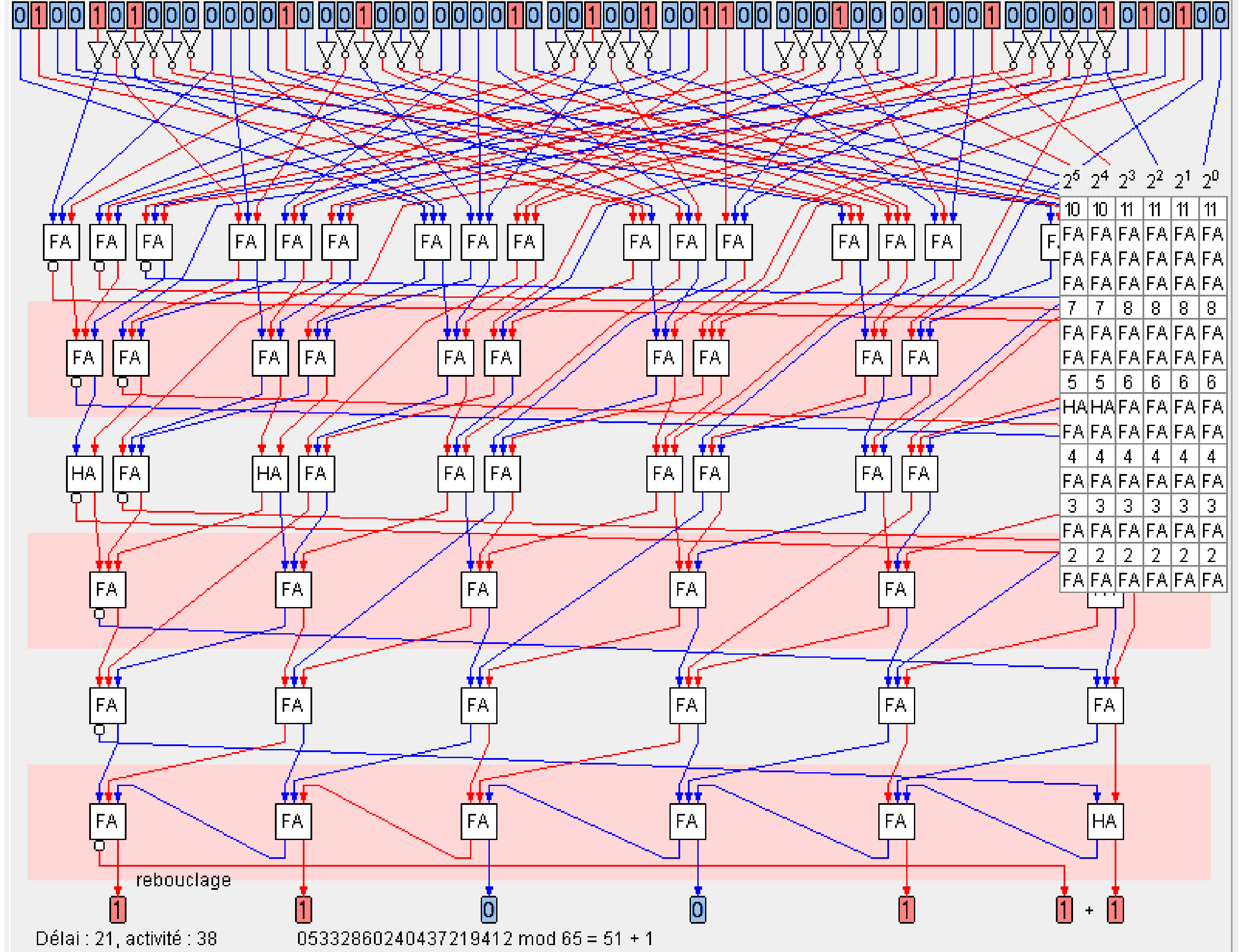
La condition est donnée par la retenue c_n: si c_n = 'K' alors $A + B < 2^n - 1$, si c_n = 'P' alors $A + B = 2^n - 1$, si c_n = 'G' alors $A + B > 2^n - 1$.

Donc le signal "rebouclage" qui contrôle " +1" vaut 'K' si c_n = 'K' et vaut 'G' autrement.



Réducteur modulo $2^n + 1$

L'applet suivant réduit 64 bits en 7 bits en conservant la valeur modulo 65 ($65 = 2^6 + 1$) . Il dérive du réducteur précédent en complémentant logiquement tous les bits de rang entre $6(2k + 1)$ et $6(2k + 1) + 5$. On ne peut pas reboucler la retenue de l'additionneur terminal, il faut au contraire l'ajouter au résultat de cet additionneur ce qui donne un nombre de 7 bits.



Additionneur modulo $2^n + 1$

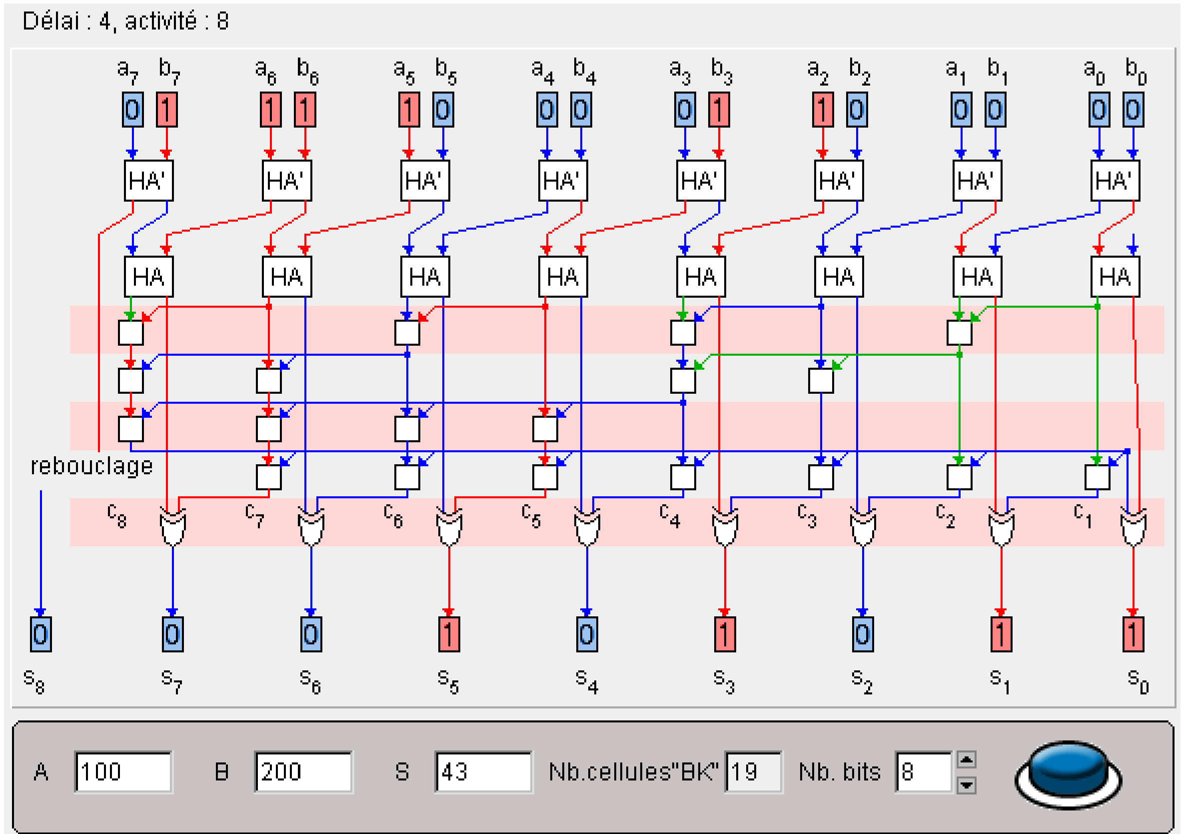
On cherche maintenant un additionneur qui fait :

- **si** $A + B < 2^n + 1$ **alors** $S = A + B$;
- **si** $A + B \geq 2^n + 1$ **alors** $S = |\, A + B - 1 \,|_{\text{modulo } 2^n}$;

On se ramène à l'additionneur précédent en calculant préalablement <u>sans propagation de retenue</u> deux nombres X et Y tels que $X + Y = A + B + 2^n - 1$ avec les cellules HA' , duales de HA.

- **si** $X + Y < 2^{n+1}$ **alors** $S = |\, X + Y + 1 \,|_{\text{modulo } 2^n}$;
- **si** $X + Y \geq 2^{n+1}$ **alors** $S = |\, X + Y \,|_{\text{modulo } 2^n}$;

Donc le signal "rebouclage" qui contrôle "+1" est le "nand" de x_n et (c_n = 'K'). Le bit poids fort s_n est le "and" de x_n et (c_n = 'P') .



Conversion de "RNS" en base mixe "MRS"

La notation "base mixe" ("MRS") est une notation de position avec les poids (1) (m_1) (m_1m_2) $(m_1m_2m_3)$ $(m_1m_2m_3.....m_{n-1})$.
Dans cette notation X s'écrit $(z_1 | z_2 | z_3 | .... | z_n)_{MRS}$ où $0 < z_i < m_i$. Remarquer que le domaine des chiffres "MRS" est le même qu'avec le "RNS" , mais les chiffres eux-mêmes sont différents.
La valeur $X = z_1 + m_1 * (z_2 + m_2 * (z_3 + m_3 * (.....)))$.