(12) DEMANDE INTERNATIONALE PUBLIÉE EN VERTU DU TRAITÉ DE COOPÉRATION EN MATIÈRE DE BREVETS (PCT)

(19) Organisation Mondiale de la Propriété Intellectuelle
Bureau international

(43) Date de la publication internationale
8 novembre 2012 (08.11.2012)

WIPO | PCT

(10) Numéro de publication internationale
WO 2012/150396 A2

(51) Classification internationale des brevets :
G06F 7/53 (2006.01)

(21) Numéro de la demande internationale :
PCT/FR2012/050818

(22) Date de dépôt international :
13 avril 2012 (13.04.2012)

(25) Langue de dépôt : français

(26) Langue de publication : français

(30) Données relatives à la priorité :
1153870 5 mai 2011 (05.05.2011) FR
1161674 15 décembre 2011 (15.12.2011) FR

(71) Déposant *(pour tous les États désignés sauf US)* : **ALTIS SEMICONDUCTOR** [FR/FR]; 224, boulevard John Kennedy, F-91105 Corbeil Essonnes (FR).

(72) Inventeurs; et

(75) Inventeurs/Déposants *(pour US seulement)* : **NACCACHE, David** [FR/FR]; 52, rue Letort, F-75018 Paris (FR). **SABEG, Karim** [FR/FR]; 34, rue de la Faisanderie, F-75016 Paris (FR).

(74) Mandataires : **HERVOUET, Sylvie** et al.; Feray Lenne Conseil, Le Centralis, 63, avenue du Général Leclerc, F-92340 Bourg la Reine (FR).

(81) États désignés *(sauf indication contraire, pour tout titre de protection nationale disponible)* : AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

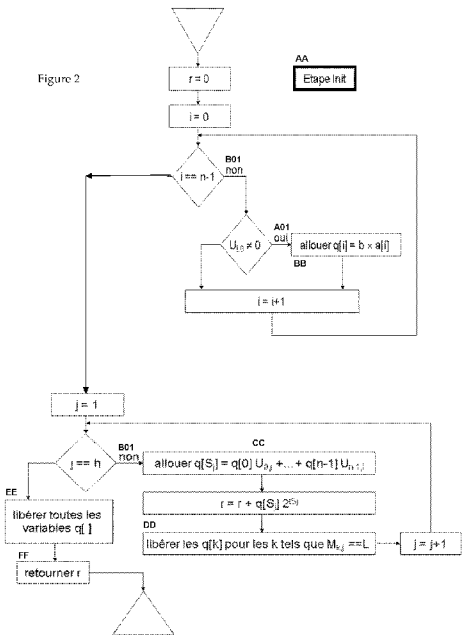
(84) États désignés *(sauf indication contraire, pour tout titre de protection régionale disponible)* : ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, SZ, TZ,

[Suite sur la page suivante]

(54) Title : DEVICE AND METHOD FOR FAST MULTIPLICATION

(54) Titre : DISPOSITIF ET PROCEDE DE MULTIPLICATION RAPIDE

Figure 2



AA Init step
B01 no
A01 yes
BB allocate
CC allocate
DD free the q[k] for the k such that M_{kj} ==L
EE free all the variables q[]
FF return r

(57) Abstract : The present invention relates to a method of encoding an integer number using an encoding function taking as input an integer number of n words of t bits or a multiple of t bits, and outputting an ordered array U of j rows and i columns containing integers Ui,,j. The words a[k] making up the number a are intercompared in order to organise them into a series of words having an increasing value (k=0, k=1, k=2,...). An array of indices makes it possible to conserve the rank k of the word a[k] in the ordered list. A first word of a is calculated by an equation using a group of words which are not expressed as a function of other words of a. Next, all the other words of a are calculated with the aid of an equation using already calculated words. In this way, it is possible to express a large number as an ordered series of terms having small discrepancies from one another. Such encoding makes it possible to limit the number of elementary multiplications when multiplying this number with another. The present invention also relates to the circuit implementing the encoding, as well as to a circuit implementing the multiplication of numbers thus encoded.

(57) Abrégé : La présente invention concerne un procédé d'encodage d'un nombre entier utilisant une fonction d'encodage prenant

[Suite sur la page suivante]

WO 2012/150396 A2

UG, ZM, ZW), eurasien (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), européen (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Publiée :

— *sans rapport de recherche internationale, sera republiée dès réception de ce rapport (règle 48.2.g))*

en entrée un nombre entier de n mot de t bits ou d'un multiple de t bits, et renvoyant en sortie un tableau ordonné U de j lignes et i colonnes contenant des entiers Ui,,j. Les mots a[k] composant le nombre a sont comparés entre eux afin de les organiser en une suite de mots ayant une valeur croissante (k=0, k=1, k=2,...). Un tableau d'indices permet de conserver le rang k du mot a[k] dans la liste ordonnée. Un premier mot de a est calculé par une équation utilisant un groupe de mots qui ne s'expriment pas en fonction d'autres mots de a. Puis tous les autres mots de a sont calculés à l'aide d'équation utilisant des mots déjà calculés. De cette façon, il est possible d'exprimer un grand nombre comme une suite ordonnée de termes ayant de faibles écarts les uns avec les autres. Un tel encodage permet de limiter le nombre de multiplications élémentaires lors de la multiplication de ce nombre avec un autre. La présente invention concerne aussi le circuit mettant en œuvre l'encodage, ainsi qu'un circuit mettant en œuvre la multiplication de nombres ainsi encodés.

Dispositif et procédé de multiplication rapide.

DOMAINE DE L'INVENTION

Le domaine de l'invention est celui des microprocesseurs et plus particulièrement celui des unités de cryptage.

Plus précisément, l'invention concerne la mise en œuvre d'opérations et de traitement numérique de type « multiplication d'entiers à multi-précision » dans un microprocesseur ou dans un bloc de logique configurable (FPGA) ou dans un coprocesseur arithmétique, et ce de façon simple et efficace.

L'invention s'applique notamment dans les systèmes embarqués et les composants de confiance tels que les modules de sécurité et les coprocesseurs à usage cryptographique basés sur une logique répartie ou configurable.

PRESENTATION DE L'ART ANTERIEUR

Selon des techniques bien connues, le traitement cryptographique de données numériques nécessite souvent d'effectuer des multiplications de grands entiers, de tels entiers ont typiquement pour taille 1024 ou 2048 bits.

La majorité des algorithmes cryptographiques à clé publique requière l'exécution répétée de l'opération $a \times b$ où a, b sont des grands entiers. La multiplication de grands entiers est particulièrement nécessaire à la réalisation d'algorithmes cryptographiques tels que le chiffrement ou la signature RSA décrits dans le brevet américain U.S. 4,405,829, de l'échange de clés Diffie-Hellman décrit dans le brevet américain U.S. 4,200,770, de la norme DSA décrite dans le brevet américain U.S. 5,231,668, des protocoles d'identification à apport nul de connaissance tels que les protocoles de Fiat-Shamir décrits dans le brevet européen EP 0252499 ou encore à la mise en œuvre de la cryptographie à courbes elliptiques, décrite dans la norme IEEE P1363, section A.12.1.

De nombreux algorithmes existent pour exécuter des multiplications d'entiers de grandes tailles. Soient a et b deux nombres entiers, a et b sont tous deux formés de n mots de t bits.

$$a = a[0]+a[1]\ 2^t + a[2]\ 2^{2t} + \ldots + a[n-1]\ 2^{(n-1)t}$$

$$b = b[0]+b[1]\ 2^t + b[2]\ 2^{2t} + \ldots + b[n-1]\ 2^{(n-1)t}$$

Ainsi le produit r = a×b sera un entier de 2n mots de t bits :

$$r = a \times b = r[0]+r[1]\ 2^t + r[2]\ 2^{2t} + \ldots + r[2n-1]\ 2^{(2n-1)t}$$

La présente invention est aussi applicable à la multiplication de grands nombres entiers n'ayant pas un nombre identique de bits. A cette fin, il y a lieu de rappeler une première méthode permettant de multiplier de tels nombres.

Une première méthode fait appel à une sous-routine de base notée MADA (acronyme signifiant « *Multiplier, Additionner, Diviser et Accumuler* ») et imite le procédé de multiplication manuel. La sous-routine MADA prend en entrée quatre mots x, y, c, r et retourne deux mots u, v tels que : $u\ 2^t + v = x\ y+c+r$. Comme x, y, c, r sont tous inférieurs ou égaux à l'entier maximal représentable dans le mot machine de t bits (à savoir 2^t-1) leur somme est inférieure ou égale à :

$$(2^t-1)(2^t-1)+(2^t-1)+(2^t-1) = (2^{2t}-2^{t+1}+1)+ 2^{t+1}-2 = 2^{2t}-1.$$

Il s'en suit que {u,v}=MADA(x,y,c,r) ne peut causer un débordement nécessitant plus que deux mots de t bits pour représenter le résultat {u,v}.

La multiplication classique est effectuée selon l'algorithme 1 suivant :

Soit deux entiers en entrée a et b, où:

$$a = a[0]+a[1]\ 2^t + a[2]\ 2^{2t} + \ldots + a[n-1]\ 2^{(n-1)t}$$

et

$$b = b[0]+b[1]\ 2^t + b[2]\ 2^{2t} + \ldots + b[n-1]\ 2^{(n-1)t}$$

1 : mettre à zéro le registre résultat r[0] = r[1] = … = r[2n-1] = 0

2 : pour i = 0 à n-1 {

2.1. : c = 0

2.2. : pour j = 0 à n-1 {

2.2.1. : {c,r[i+j]} = MADA(a[i],b[j],c,r[i+j]) }

2.3. : r[i+n]=c }

3 : retourner le résultat $r = r[0]+r[1]\ 2^t + r[2]\ 2^{2t} + \ldots + r[2n-1]\ 2^{(2n-1)t}$

Algorithme 1

Selon cette méthode, supposons un nombre de 64 octets multiplié à un nombre de 64 octets et, supposons que le processeur ne dispose que d'un multiplieur octet par octet, la multiplication de ces deux nombres nécessite d'utiliser séquentiellement 64 au carré : 4196 fois ce multiplieur. Ce grand nombre de multiplications élémentaires occupe le processeur pendant un temps non négligeable.

D'autres méthodes de multiplication applicables au grand nombre existent. On peut citer la méthode de multiplication de Karatsuba.

D'autres méthodes de multiplication applicables au grand nombre existent. On peut citer la méthode de multiplication de Karatsuba.

Cette méthode, publiée dans l'article de A. Karatsuba et Yu. Ofman (1962). "Multiplication of Many-Digital Numbers by Automatic Computers". Proceedings of the USSR Academy of Sciences 145: 293–294.) utilise comme sous-routine un programme de multiplication existant, par exemple un programme mettant en œuvre la multiplication classique que nous venons d'exposer.

La méthode de Karatsuba considère les nombres entiers a et b comme la concaténation de deux parties de tailles égales : $a = a_{haut}\ 2^L + a_{bas}$ et $b = b_{haut}\ 2^L + b_{bas}$. Typiquement $L = tn/2$. La multiplication de Karatsuba est effectuée selon l'algorithme 2 suivant :

Soit deux entiers en entrée a et b, où:

$a = a_{haut}\ 2^L + a_{bas}$ et $b = b_{haut}\ 2^L + b_{bas}$.

1 : $u = a_{haut} \times b_{haut}$

2 : $v = a_{bas} \times b_{bas}$

3 : $w = (a_{haut} + a_{bas}) \times (b_{haut} + b_{bas}) - u - v$

4 : retourner le résultat $r = u\ 2^{2L} + w\ 2^L + v$

Algorithme 2

Il est facile de vérifier, en substituant les valeurs de u, v et w que la quantité r retournée par l'algorithme de Karatsuba est :

$a_{haut}\, b_{haut}\, 2^{2L} + ((a_{haut} + a_{bas})\,(b_{haut} + b_{bas}) - a_{haut}\, b_{haut} - a_{bas}\, b_{bas})\, 2^{L} + a_{bas}\, b_{bas} =$

$a_{haut}\, b_{haut}\, 2^{2L} + ((a_{haut}\, a_{bas}) + (b_{haut}\, b_{bas}))\, 2^{L} + a_{bas}\, b_{bas} = a \times b$

L'avantage de la méthode de Karatsuba consiste en ce que le résultat r de la multiplication de a par b s'obtient en effectuant trois (et non quatre) multiplications de nombres de L bits. De ce fait, en utilisant une telle méthode, il est déjà possible de réduire de 25 % le nombre de multiplications. La méthode de Karatsuba se prête facilement à une généralisation récursive connue à l'homme de l'art permettant d'accroître le gain de vitesse au-delà de 25% sur un processeur standard.

L'utilisation de circuits spécifiques permet d'améliorer les performances en termes de temps de calcul. Pour cela, il est possible d'utiliser des coprocesseurs arithmétiques spécialisés dans les calculs cryptographiques utilisant des grands nombres. Ces circuits peuvent être prédiffusés ou programmables après la fabrication. Les circuits prédiffusés (en Anglais « Gate Array ») réalisent des fonctions établis lors de leur conception, ils ne sont pas modifiables ensuite. Les circuits programmables sont constitués d'un grand nombre de portes logiques reliées à une unité de contrôle et de programmation. Ces circuits effectuent certaines fonctions, selon leur programmation. Parmi les types de circuits programmables, on peut citer les plus courants :

- FPGA (« Field-Programmable Gate Array »),
- PLD (« Programmable Logic Device »),
- EPLD (« Erasable Programmable Logic Device »),
- CPLD (« Complex Programmable Logic Device »),
- PAL (« Programmable Array Logic »)

Le nombre de portes disponibles dans ces circuits, et de ce fait, la puissance de calcul utilisable, est limité. La mise en œuvre d'une multiplication simple, ou d'une multiplication modulaire dans de tels composants nécessite une programmation performante.

L'invention a pour objectif général de pallier, dans des contextes particuliers, à au moins certains inconvénients de ces techniques connues et ce

surtout lorsque l'on dispose de multiplieurs matériels de taille relativement modeste (par exemple 8 bits × 8 bits).

RESUME DE L'INVENTION

Ainsi, la présente invention propose un procédé d'encodage d'un nombre entier utilisant une fonction d'encodage prenant en entrée un nombre entier de n mot de t bits ou d'un multiple de t bits, et renvoyant en sortie un tableau ordonné U de j lignes et i colonnes contenant des entiers Ui,j. Les mots composant le nombre a sont dans l'ordre du poids le plus faible au poids le plus grand : a[0], a[1], a[2], … a[n-1],

la fonction d'encodage renvoie également en sortie un sous ensemble A non vide de « p » mots du nombre a et une fonction de correspondance f associant l'indice « k » d'au moins un mot de a à l'indice « j » d'au moins une ligne de U,

les entiers Ui,j sont les coefficients d'une pluralité d'égalités permettant d'exprimer la valeur de mots a[k] de a en fonction des autres mots a[k] de a, le calcul de la valeur du mot a[k] s'effectuant par l'équation suivante :

$$a[k] = U0,j \times a[0] + U1,j \times a[1] + \ldots + U(n-1),j \times a[n-1], \text{ où } f(k) = j$$

la première ligne j=0 de U permettant de calculer une valeur de mot a[k] à l'aide de mots composant l'ensemble A de « p » mots qui ne s'expriment pas en fonction d'autres mots de a, chaque ligne suivante de U permettant de calculer une valeur de mot a[k] à l'aide de mots de l'ensemble A et de mots dont la valeur est calculée dans une ligne précédente, l'ensemble des (n-p) lignes de U permettant de calculer toutes les valeurs de mots a[k] qui ne sont pas dans l'ensemble A.

En listant de façon ordonnée les nombres composant le multiplicande, les multiplications peuvent être organisées comme une suite d'additions. Cette suite ordonnée de nombres permet de multiplier d'abord le multiplicateur avec le plus petit nombre, puis de considérer que les multiplications suivantes sont des additions de une ou de plusieurs unités. Chaque résultat est additionné au précédent en prenant en compte le rang du nombre dans la suite ordonnée. Soit n le nombre de nombres composant le multiplicande, le calcul ainsi réalisé utilise entre n et 2n multiplications. De ce fait, l'exécution d'une addition est

plus rapide que celle d'une multiplication classique, les temps de calcul sont ainsi diminués. L'invention permet de fournir une technique permettant d'effectuer efficacement des multiplications d'entiers variables par un entier constant.

L'invention permet également de fournir une technique permettant d'effectuer efficacement des multiplications d'entiers variables par d'autres entiers variables. L'invention permet également de fournir une technique efficace de gestion de la mémoire vive, permettant de réduire la sauvegarde nécessaire à l'exécution des méthodes de multiplication inventives. L'invention permet également de fournir une technique de multiplication pouvant être facilement parallélisée sur une architecture multiprocesseur SIMD ou sur une architecture à logique configurable. L'invention permet également de fournir une technique de génération de clés cryptographiques facilement multipliables par des entiers variables à l'aide des méthodes inventives.

La présente invention propose également un circuit apte à mettre en œuvre le procédé d'encodage d'un nombre entier décrit précédemment.

BREVE PRESENTATION DES DESSINS

D'autres caractéristiques et avantages de l'invention apparaîtront à travers la description d'un exemple de réalisation non limitatif de l'invention, explicité à l'aide des figures jointes, parmi lesquelles :

- la figure 1 est un diagramme bloc d'un circuit comportant des portes logiques aptes à exécuter une multiplication selon un exemple de réalisation.
- la figure 2 présente un ordinogramme pour la mise en œuvre de l'invention selon un exemple de réalisation.

DESCRIPTION D'UN MODE DE REALISATION

Pour faciliter la description, l'invention sera décrite ci-après dans le cadre de l'implémentation du procédé d'encodage dans un FPGA. L'implémentation du procédé dans tout circuit électronique doté de portes est parfaitement envisageable.

Un circuit logique programmable dit FPGA, ou réseau logique programmable, tel que représenté par la figure 1, est un circuit intégré logique qui peut être reprogrammé après sa fabrication. Il est composé de nombreuses cellules logiques élémentaires (élément 3 sur la figure 1) librement assemblables.

Un bloc logique est de manière générale constitué d'une table de correspondance (LUT ou Look-Up-Table) et d'une bascule (Flip-Flop en anglais). La LUT sert à implémenter des équations logiques ayant généralement 4 à 6 entrées et une sortie. Elle peut toutefois être considérée comme une petite mémoire, un multiplexeur ou un registre à décalage. Le registre permet de mémoriser un état (machine séquentielle) ou de synchroniser un signal (pipeline).

Les blocs logiques, présents en grand nombre sur la puce (de quelques milliers à quelques millions en 2007) sont connectés entre eux par une matrice de routage configurable (élément 1 sur la figure 1). Ceci permet la reconfiguration à volonté du composant, mais occupe une place importante sur le silicium et justifie le coût élevé des composants FPGA. La topologie est dite « Manhattan », en référence aux rues à angle droit de ce quartier de New York. Le FPGA communique avec le mode extérieur à l'aide de cellules d'entrée sortie (élément 2 sur la figure 1).

La structure d'un tel circuit rend malaisée la réalisation de grands multiplieurs parallèle-parallèle car dans de tels circuits, où chaque bit du résultat dépends de tous les bits d'entrée demandent la connexion de signaux par-dessus les blocs logiques. Ainsi, la programmation directe d'un algorithme de multiplication classique sur FPGA résultera en de piètres performances.

Ainsi, il devient important de mettre au point des circuits de multiplication adaptés aux FPGAs au cas où le FPGA servirait pour le calcul et non pour prototyper un algorithme.

Selon un aspect particulier de la présente invention, l'accroissement d'efficacité s'exprime par un nombre réduit d'appels à la sous-routine MADA et par une parallélisation possible de l'opération de multiplication. La multiplication d'un nombre variable par une constante fixe est très utile, surtout

en cryptographie où il est souvent nécessaire de calculer le résultat de l'opération g^x où g est fixe et x variable. Une telle opération est aussi très utile pour le traitement graphique où une multiplication par des constantes fixes est souvent nécessaire afin d'effectuer des calculs trigonométriques. Enfin, de nombreux algorithmes de traitement du signal (filtrage, FFT, convolution) nécessitent d'effectuer des multiplications de nombres variables par des constantes fixes utilisant le procédé d'encodage selon la présente invention.

L'invention peut également s'adapter aux cas dans lesquels il est nécessaire de multiplier deux nombres variables l'un par l'autre comme il sera expliqué dans les paragraphes qui suivent. Un exemple de réalisation de l'invention va maintenant être décrit.

Une représentation d'un entier a est maintenant décrite. On rappelle que l'invention permet de calculer la valeur $r=a\times b$.

Il est rappelé que le nombre a est divisé en n mots de taille t bits.

$a = a[0]+a[1]\ 2^t + a[2]\ 2^{2t} + \ldots + a[n-1]\ 2^{(n-1)t}$

Dans l'expression a[k], k représente l'indice du mot a[k], ce mot étant une partie du nombre a.

Nous allons maintenant décrire un algorithme permettant d'encoder a pour calculer la valeur $r=a\times b$ en minimisant le nombre de multiplication élémentaires.

L'algorithme génère, une fois pour toutes, un tableau U comportant au plus $h \leq n-2$ lignes, organisé comme suit : la cellule $U_{i,j}$ contient une valeur entière $U_{i,j}$ comprise entre $-c$ et $+c$ où c est une petite borne entière, typiquement 1, 2 ou 3. La colonne « cible » contient un entier S_j compris entre 0 et n-1.

étape j	a[0]	a[1]	…	a[n-2]	a[n-1]	cible
j=0	$U_{0,0}$	$U_{1,0}$	…	$U_{n-2,0}$	$U_{n-1,0}$	S_0
j=1	$U_{0,1}$	$U_{1,1}$	…	$U_{n-2,1}$	$U_{n-1,1}$	S_1
…	…	…	…	…	…	…
j=h-1	$U_{0,h-1}$	$U_{1,h-1}$	…	$U_{n-2,h-1}$	$U_{n-1,h-1}$	S_{h-1}
j=h	$U_{0,h}$	$U_{1,h}$	…	$U_{n-2,h}$	$U_{n-1,h}$	S_h

Le tableau $U_{i,j}$ est une représentation alternative du nombre a, cette représentation alternative exprime une manière dont les mots a[i], constituant l'entier a, peuvent être obtenus les uns des autres par des combinaisons linéaires successives à petits coefficients, selon la formule suivante :

$$a[S_j] = a[0]\ U_{0,j} + \ldots + a[n-1]\ U_{n-1,j}$$

Voici un exemple de tableau :

Etape j	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	cible
j=0	$U_{0,0}=0$	$U_{1,0}=1$	$U_{2,0}=0$	$U_{3,0}=0$	$U_{4,0}=-1$	$U_{5,0}=0$	$U_{6,0}=1$	$S_0=2$
j=1	$U_{0,1}=0$	$U_{1,1}=0$	$U_{2,1}=0$	$U_{3,1}=0$	$U_{4,1}=1$	$U_{5,1}=0$	$U_{6,1}=0$	$S_1=5$
j=2	$U_{0,2}=0$	$U_{1,2}=0$	$U_{2,2}=-1$	$U_{3,2}=0$	$U_{4,2}=0$	$U_{5,2}=1$	$U_{6,2}=0$	$S_2=0$
j=3	$U_{0,3}=1$	$U_{1,3}=0$	$U_{2,3}=0$	$U_{3,3}=0$	$U_{4,3}=0$	$U_{5,3}=-1$	$U_{6,3}=0$	$S_3=3$

Un tel tableau U exprime le fait que les chiffres du nombre a peuvent s'obtenir successivement ainsi : On débute le processus avec trois mots : a[1], a[4] et a[6]. Cette première ligne j=0 de U permettant de déterminer une valeur de mot a[Si] à l'aide de mots de a qui ne s'expriment pas à partir d'autres mots de a mais qui s'expriment uniquement par leur valeur. A partir de cette ligne, il est possible de calculer une autre valeur a[Si'] à partir de mot a[si] précédemment déterminé.

Former a[2]=a[1]-a[4]+a[6] (étape j=0) puis a[5]=a[4] (étape j=1) puis a[0]=a[5]-a[2] (étape j=2) et enfin a[3]=a[0]-a[5] (étape j=3).

Ainsi, on peut constater que le mot a[2] est formé à partir des mots a[1] a[4] et a[6]. Les trois mots a[1] a[4] et a[6] sont représentés uniquement par leur valeur et ne sont pas le résultat d'une précédente équation. La valeur du mot a[2] de a est calculé à partir d'autres mots de a. Les mots suivants en descendant chaque ligne du tableau permettent de calculer une autre valeur a[Si'] à partir de mot a[si] précédemment calculé.

Il importe maintenant de décrite la façon de remplir le tableau. Etant donné un nombre entier a, la construction du tableau U qui lui est associé fait appel à la technique de « retour sur trace » (appelée aussi « *backtracking* » en anglais). Cette technique est décrite dans de très nombreuses références

bibliographiques, dont par exemple « Gurari Eitan, 1999 Backtracking Algorithms « CIS 680 : Data Structures : Chapter 19 : Backtracking Algorithms ».

La technique du retour sur trace, connue de l'homme de l'art, consiste à revenir légèrement en arrière sur des décisions prises afin de sortir d'un blocage.

L'allocation et la libération de la mémoire en cours de l'algorithme est maintenant décrite. Il est possible de définir une table auxiliaire M indiquant les variables pouvant être libérées à la fin de chaque étape du calcul.

Le tableau M est dérivé du tableau U ainsi :

Entrée :

un tableau $U_{i,j}$

1 : mettre à la valeur 1 toutes les cellules telles que $U_{i,j}\neq 0$.

2 : supprimer la colonne « cible »

3 : pour i = 1 à n-1 {

3.1. : soit T_i le plus grand indice j tel que $U_{i,j}=1$

3.2. : mettre le symbole L dans toutes les cellules $U_{i,Ti}$ }

4 : mettre à la valeur 0 toutes les cellules telles que $U_{i,j}=1$

5 : enlever la ligne j = h.

6 : retourner le tableau $M_{i,j}$ résultant de l'exécution des étapes 1 à 5

Algorithme 3

Nous noterons par M^k le tableau généré après l'étape k (k=1,2,3,4,5) du processus que nous venons de décrire.

Le symbole « L » lisible dans le tableau signifie « Libérer après usage ». Cette notation indique que lors de la prochaine boucle, cette case mémoire peut être affectée à un autre calcul. De cette façon, lors de la programmation du FPGA, des mémoires peuvent servir plusieurs fois pour des résultats temporaires. De cette façon, il est possible de diminuer le nombre de cases mémoires utiles à l'algorithme.

Il est évident que le symbole « L » n'a rien d'obligatoire ou de fondamental et toute autre façon de marquer l'information peut être substituée au symbole « L ».

Par exemple : après les étapes 1 et 2 le tableau donné à titre d'exemple de réalisation non limitatif deviendra M^2 :

étape j	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
j=0	0	1	0	0	1	0	1
j=1	0	0	0	0	1	0	0
j=2	0	0	1	0	0	1	0
j=3	1	0	0	0	0	1	0

après l'étape 3 le tableau donné à titre d'exemple de réalisation non limitatif deviendra M^3 :

étape j	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
j=0	0	L	0	0	1	0	L
j=1	0	0	0	0	L	0	0
j=2	0	0	L	0	0	1	0
j=3	L	0	0	0	0	L	0

après l'étape 4 le tableau donné à titre d'exemple de réalisation non limitatif deviendra M^4 :

étape j	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
j=0	0	L	0	0	0	0	L
j=1	0	0	0	0	L	0	0
j=2	0	0	L	0	0	0	0
j=3	L	0	0	0	0	L	0

après l'étape 5 le tableau donné à titre d'exemple de réalisation non limitatif deviendra M^5=M :

étape j	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
j=0	0	L	0	0	0	0	L
j=1	0	0	0	0	L	0	0
j=2	0	0	L	0	0	0	0

La reconstruction des mots de a, particulièrement économique en mémoire, selon l'exemple de réalisation illustré par le tableau M ci-dessus, s'exécute ainsi :

Reprenons l'exemple précédent et appliquons la partie de l'algorithme dédié à l'allocation des mémoires du tableau M. L'algorithme débute le processus avec trois mots : a[1], a[4] et a[6]

comme indiqué par le tableau U, générer a[2]=a[1]-a[4]+a[6],

à cette étape le tableau M indique que l'on peut libérer les registres contenant a[1] et a[6]. En effet, la présence du symbole L indique que les valeurs mémorisées dans ces mémoires ne seront plus utilisées dans aucune étape ultérieure, il est donc possible de les affecter à d'autres valeurs ou d'autres calculs effectués en parallèle par exemple. De ce fait, le nombre de mémoire utilisées par l'algorithme est diminué,

comme indiqué par le tableau U, générer a[5]=a[4],

à cette étape le tableau M indique que l'on peut libérer le registre contenant a[4],

comme indiqué par le tableau U, générer a[0]=a[5]-a[2],

à cette étape, le symbole L présent dans le tableau M indique que l'on peut libérer le registre contenant a[2],

générer a[3]=a[0]-a[5],

à cette étape on peut libérer tous les registres restant alloués, en effaçant les symboles L du tableau.

Nous allons maintenant détailler l'algorithme dans le cas d'une multiplication par une constante fixe. Afin d'effectuer l'opération a×b où a est fixe et b variable, l'algorithme procède de manière très analogue au procédé de reconstruction des mots de a que nous venons de décrire. L'algorithme calcule par une multiplication « entier × mot » les quantités b×a[1], b×a[4] et b×a[6], et on initialise le registre de résultat à la valeur $r = b \times a[1] \times 2^{t} + b \times a[4] \times 2^{4t} + b \times a[6] \times 2^{6t}$. Il est maintenant possible de calculer par une addition et une soustraction la quantité $b \times a[2] = b \times a[1] - b \times a[4] + b \times a[6]$ et de mettre à jour le registre de résultat $r = r + b \times a[2] \times 2^{2t}$. De ce fait, un certain nombre de multiplications sont remplacés par des additions et des soustractions successives, réduisant ainsi le temps de calcul de la multiplication.

L'algorithme libère alors les registres contenant $b \times a[1]$ et $b \times a[6]$ comme l'indique le tableau M. En consultant le tableau U on voit que a[5]=a[4] et donc que l'opération suivante à effectuer est $b \times a[5] = b \times a[4]$.

Ainsi, l'algorithme met à jour le registre de résultat $r = r + b \times a[4] \times 2^{5t}$ et on libère le registre contenant $b \times a[4]$ comme l'indique le tableau M.

Ensuite, l'algorithme calcule $b \times a[0] = b \times a[5] - b \times a[2]$, on met à jour $r = r + b \times a[0]$ puis on libère le registre contenant $b \times a[2]$.

Enfin, le tableau U indique que l'algorithme doit calculer $b \times a[3] = b \times a[0] - b \times a[5]$.

L'algorithme calcule donc le résultat final $r = r + b \times a[3] \times 2^{3t}$ et libère tous les registres restant alloués, en supprimant tous les symboles « L » restant dans le tableau.

Un exemple d'algorithme est présenté ci-dessous et illustré par la figure 2:

Entrée :

une liste de mots a[i] pour les indices i tels que $U_{i,0} \neq 0$

un entier b

un tableau $U_{i,j}$

un tableau $M_{i,j}$

1 : initialiser r = 0

2 : pour i = 0 à n-1 {

2.1. : si $U_{i,0} \neq 0$ alors allouer la variable q[i] = b × a[i] }

3 : pour j = 1 à h {

3.1. : allouer la variable $q[S_j] = q[0]\ U_{0,j} + \ldots + q[n-1]\ U_{n-1,j}$

3.2. : mettre à jour $r = r + q[S_j]\ 2^{t\,Sj}$

3.3. : libérer les variables q[k] pour les k est tel que $M_{k,j} == L$ }

4 : libérer toutes les variables q[ ] restantes allouées.

5 : retourner le résultat r.

Algorithme 4

A noter que, dans le calcul de la somme $q[S_j] = q[0]\ U_{0,j} + \ldots + q[n-1]\ U_{n-1,j}$ le résultat de la multiplication d'une variable q[ ] non allouée par un $U_{i,j}$ est défini comme étant zéro.

A noter que, le nombre de registres nécessaires à l'exécution de l'algorithme à l'étape j est égale à $v_j = n - z_j + 1$ où z_j note le nombre de cellules à la ligne j dans le tableau M^4 contenant un zéro. Ainsi, la consommation totale de mémoire de l'algorithme multiplication est égale à $\max_j (v_j)$.

Nous allons maintenant détailler l'algorithme dans le cas d'une multiplication avec un entier variable.

Dans le cas où a et b sont tous deux des nombres entiers variables, il devient trop coûteux de construire un tableau U en temps réel par la technique de « retour sur trace ». La méthode objet de la présente invention peut s'adapter au cas où a et b sont tous deux des nombres entiers variables.

Pour ce faire l'algorithme trie les mots a[i] par ordre croissant afin d'obtenir une suite d[0],…,d[n-1]. Ensuite, l'algorithme calcule les différences Δ[i] = d[i+1]-d[i] dont l'espérance est de l'ordre de $2^t/n$. Ainsi, en calculant les quantités b × j pour j = 2,…,B pour une borne B de l'ordre de $2^t/n$ il devient possible de construire, par additions successives la grande majorité des valeurs b × a[i].

Lorsque l'écart entre d[i+1] et d[i] est tel que Δ[i]>B, on traite un tel Δ[i] comme une exception :

- soit en ajoutant des termes de la forme b × j dont nous disposons pour un sous-ensemble des valeurs de j dont la somme égale Δ[i]

- soit en effectuant directement la multiplication $b \times \Delta[i]$ permettant ensuite de passer de $b \times d[i]$ à $b \times d[i+1]$ par une simple addition.

Pour plus de clarté, nous illustrons ici le procédé en base 100 et non en une base 2^t.

Par exemple, supposons que l'on souhaite multiplier le nombre a :

03141592653589793238462643383279502884197169399375105820974944

59230781640628620899862803482534211706 80

par un certain nombre entier b également.

L'algorithme découpe a en groupes de deux chiffres décimaux :

a[00]= 03	a[01]= 14	a[02]= 15	a[03]= 92	a[04]= 65
a[05]= 35	a[06]= 89	a[07]= 79	a[08]= 32	a[09]= 38
a[10]= 46	a[11]= 26	a[12]= 43	a[13]= 38	a[14]= 32
a[15]= 79	a[16]= 50	a[17]= 28	a[18]= 84	a[19]= 19
a[20]= 71	a[21]= 69	a[22]= 39	a[23]= 93	a[24]= 75
a[25]= 10	a[26]= 58	a[27]= 20	a[28]= 97	a[29]= 49
a[30]= 44	a[31]= 59	a[32]= 23	a[33]= 07	a[34]= 81
a[35]= 64	a[36]= 06	a[37]= 28	a[38]= 62	a[39]= 08
a[40]= 99	a[41]= 86	a[42]= 28	a[43]= 03	a[44]= 48
a[45]= 25	a[46]= 34	a[47]= 21	a[48]= 17	a[49]= 06
a[50]= 80.				

La liste précédente est ensuite triée par ordre croissant afin d'obtenir :

d[00]= 03	d[01]= 03	d[02]= 06	d[03]= 06	d[04]= 07
d[05]= 08	d[06]= 10	d[07]= 14	d[08]= 15	d[09]= 17
d[10]= 19	d[11]= 20	d[12]= 21	d[13]= 23	d[14]= 25
d[15]= 26	d[16]= 28	d[17]= 28	d[18]= 28	d[19]= 32
d[20]= 32	d[21]= 34	d[22]= 35	d[23]= 38	d[24]= 38
d[25]= 39	d[26]= 43	d[27]= 44	d[28]= 46	d[29]= 48
d[30]= 49	d[31]= 50	d[32]= 58	d[33]= 59	d[34]= 62
d[35]= 64	d[36]= 65	d[37]= 69	d[38]= 71	d[39]= 75
d[40]= 79	d[41]= 79	d[42]= 80	d[43]= 81	d[44]= 84
d[45]= 86	d[46]= 89	d[47]= 92	d[48]= 93	d[49]= 97
d[50]= 99				

Ce qui donne la liste des différences suivante :

Δ[00]= 0	Δ[01]= 3	Δ[02]= 0	Δ[03]= 1	Δ[04]= 1
Δ[05]= 2	Δ[06]= 4	Δ[07]= 1	Δ[08]= 2	Δ[09]= 2
Δ[10]= 1	Δ[11]= 1	Δ[12]= 2	Δ[13]= 2	Δ[14]= 1
Δ[15]= 2	Δ[16]= 0	Δ[17]= 0	Δ[18]= 4	Δ[19]= 0
Δ[20]= 2	Δ[21]= 1	Δ[22]= 3	Δ[23]= 0	Δ[24]= 1
Δ[25]= 4	Δ[26]= 1	Δ[27]= 2	Δ[28]= 2	Δ[29]= 1
Δ[30]= 1	Δ[31]= 8	Δ[32]= 1	Δ[33]= 3	Δ[34]= 2
Δ[35]= 1	Δ[36]= 4	Δ[37]= 2	Δ[38]= 4	Δ[39]= 4
Δ[40]= 0	Δ[41]= 1	Δ[42]= 1	Δ[43]= 3	Δ[44]= 2
Δ[45]= 3	Δ[46]= 3	Δ[47]= 1	Δ[48]= 4	Δ[49]= 2

Ainsi, les écarts possibles sont 0, 1, 2, 3, 4 et 8.

En prenant la borne B=4 il s'avère nécessaire de calculer une fois pour toutes les écarts $2 \times b$, $3 \times b$ et $4 \times b$ afin de couvrir toute la liste à l'exception de Δ[31]= 8.

Afin de générer $8 \times b$, deux méthodes sont donc possibles :

- soit en ajoutant des termes de la forme $b \times j$ dont nous disposons pour un sous-ensemble des valeurs de j dont la somme égale Δ[i]. Dans le cas présent $4 \times b + 4 \times b$.

- soit en effectuant directement la multiplication $b \times \Delta[i]$ permettant de passer de $b \times d[i]$ à $b \times d[i+1]$ par une simple addition. Dans le cas présent il s'agit d'effectuer la multiplication $b \times 8$.

Un perfectionnement consistant à exécuter en parallèle des multiplications élémentaires est maintenant détaillé.

Dans le cas où l'on dispose de plusieurs cœurs de multiplication, il est possible d'adapter le procédé afin de faire fonctionner un nombre « e » de cœurs de multiplication en parallèle. Selon un exemple préféré, les multiplications s'exécutent sur 16 bits, mais ce nombre peut évoluer selon les performances des cœurs de multiplication.

Pour l'exécution en parallèle, il est nécessaire d'adapter le procédé de retour sur trace afin de construire e tableaux $U^1,\ldots,U^e$ minimisant la dépendance entre les a[i] de sorte que les e processus lancés sur e cœurs puissent être soit indépendants soit synchronisés. Lorsque les e processus se terminent, le résultat

final r s'obtient en faisant la somme des résultats $r^1,\ldots,r^e$ retournés par les e processus.

Les cœurs de multiplication peuvent être indépendants ou synchronisés entre eux. Deux processus de multiplication sont dits indépendants si aucun des deux processus n'utilise les a[i] produits par l'autre processus. En d'autres termes, les différents processus indépendants partitionnent l'ensemble des a[i] en e classes mutuellement disjointes dans le temps et dans l'espace mémoire et dont le calculs indépendants sont codés par les $U^1,\ldots,U^e$.

Deux processus sont dits synchrones si aucun des deux processus n'utilise un a[i] avant que ce dernier n'ait été produit par l'autre processus. En d'autres termes, les deux processus partitionnent l'ensemble des a[i] en e classes indépendantes seulement dans l'espace mémoire mais pas dans le temps, et dont les calculs sont codés par les $U^1,\ldots,U^e$.

Quoique technique, la manière d'obtenir à partir d'un nombre entier a, un ensemble de e tableaux $U^1,\ldots,U^e$ indépendants ou synchrones est bien connue de l'homme de l'art. Il n'est donc pas utile de la détailler d'avantage.

Selon un perfectionnement, il est possible de rendre le procédé de recherche d'un tableau U plus rapide en offrant à l'algorithme de retour sur trace plus de solutions potentielles.

Afin d'illustrer ce perfectionnement, considérons un exemple de nombre entier a, défini de manière décimale ainsi :

a = [chiffres] 01 66 [chiffres] 81 [chiffres] 85 [chiffres]

Comme 166 = 81 + 85 il est clair que la quantité 166 × b peut s'obtenir par l'addition de 81 × b + 85 × b. Et ce même si le nombre 166 s'étale sur deux mots consécutifs. De cette manière, il est possible de minimiser le nombre de multiplication en utilisant des résultats précédemment calculés.

De même considérons un autre nombre entier a, défini de manière décimale ainsi :

a = [chiffres] 41 66 [chiffres] 81 [chiffres] 85 [chiffres] 40 [chiffres]

Comme 166 = 81 + 85 il est clair que la quantité 41 66 × b peut s'obtenir par l'addition de 81 × b + 85 × b + 100 × (40 × b). Ce perfectionnement s'adapte également aux nombres négatifs comme illustré dans l'exemple suivant où l'entier a, est toujours défini de manière décimale ainsi :

a = [chiffres] 41 66 [chiffres] 34 [chiffres] 42 [chiffres]

Puisque 41 66 = 4200 - 34 la quantité 41 66 × b peut s'obtenir par l'opération 100 × (42 × b) - 34 × b.

Selon un perfectionnement, le procédé objet de l'invention permet de générer des clefs cryptographiques. Dans un certain nombre de scénarios cryptographiques, il s'avère possible de choisir des clés ayant une forme particulière.

Par exemple, dans le protocole d'identification de Fiat-Shamir, il est nécessaire d'effectuer la multiplication de clés privées s_i dont les clés publiques v_i associées sont définies par $v_i = 1/s_i^2 \bmod n$. Ainsi, il est possible de choisir les clés s_i de sorte à ce qu'une multiplication d'un s_i par un s_j soit rapide.

Pour ce faire, l'algorithme peut générer d'abord le tableau U et ensuite dériver le nombre a qu'il représente (plutôt qu'effectuer la démarche inverse). Cette méthode présente au moins deux avantages :

- Il n'est plus nécessaire d'avoir recours à l'algorithme de « retour sur trace » afin de générer U.
- Le nombre de registres q[ ] nécessaires à l'algorithme 4 peut être contrôlé et fixé d'avance lors de la génération de la clé.

Deux éléments doivent être évalués avant toute mise en œuvre du présent procédé de génération de clés multipliables efficacement:

- L'entropie des clés ainsi générées se trouve réduite.
- Il n'est parfois plus possible de fixer les autres paramètres du système à des valeurs arbitraires si ces autres paramètres sont générés à partir de clés multipliables efficacement.

Ce second point se comprendra facilement en considérant l'exemple du protocole de Fiat-Shamir :

Si le générateur de la clé génère d'abord s_i et calcule ensuite $v_i = 1/s_i^2 \bmod n$, le nombre v_i n'aura aucune forme particulière.

Par contre, si le générateur de la clé souhaite que v_i soit une chaîne particulière (par exemple l'identité d'un utilisateur), alors le nombre s_i calculé par $s_i = 1/\sqrt{v_i} \bmod n$ n'aura aucune forme particulière et la génération du tableau U qui lui est associé nécessitera un recours à un algorithme de retour sur trace.

Des personnes versées dans l'art pourront adapter la présente invention sous de nombreuses autres formes spécifiques sans l'éloigner du domaine d'application de l'invention comme revendiqué. En particulier, le circuit apte à mettre en œuvre le procédé d'encodage peut être tout circuit intégré doté de portes logiques contrôlables par une programmation appropriée. Par conséquent, les présents modes de réalisation doivent être considérés à titre d'illustration mais peuvent être modifiés dans le domaine défini par la portée des revendications jointes.

REVENDICATIONS

1. Procédé d'encodage d'un nombre entier utilisant une fonction d'encodage prenant en entrée un nombre entier de n mot de t bits ou d'un multiple de t bits, et renvoyant en sortie un tableau ordonné U de j lignes et i colonnes contenant des entiers Ui,j, les mots composant le nombre a étant dans l'ordre du poids le plus faible au poids le plus grand : a[0], a[1], a[2], … a[n-1],

caractérisé en ce que,

la fonction d'encodage renvoie également en sortie un sous ensemble A non vide de « p » mots du nombre a et une fonction de correspondance f associant l'indice « k » d'au moins un mot de a à l'indice « j » d'au moins une ligne de U,

les entiers Ui,j étant les coefficients d'une pluralité d'égalités permettant d'exprimer la valeur de mots a[k] de a en fonction des autres mots a[k] de a, le calcul de la valeur du mot a[k] s'effectuant par l'équation suivante :

$$a[k] = U0,j \times a[0] + U1,j \times a[1] + \ldots + U(n-1),j \times a[n-1], \text{ où } f(k) = j$$

la première ligne j=0 de U permettant de calculer une valeur de mot a[k] à l'aide de mots composant l'ensemble A de « p » mots qui ne s'expriment pas en fonction d'autres mots de a, chaque ligne suivante de U permettant de calculer une valeur de mot a[k] à l'aide de mots de l'ensemble A et de mots dont la valeur est calculée dans une ligne précédente, l'ensemble des (n-p) lignes de U permettant de calculer toutes les valeurs de mots a[k] qui ne sont pas dans l'ensemble A.

2. Procédé d'encodage d'un nombre entier selon la revendication 1 caractérisé en ce que les entiers $U_{i,j}$ sont tels que $-c \leq U_{i,j} \leq c$ et, c est une constante entière.

3. Procédé d'encodage d'un nombre entier selon la revendication 2, caractérisé en ce que l'entier c a pour valeur 1 ou 2 ou 3.

4. Procédé d'encodage d'un nombre entier selon l'une quelconque des revendications précédentes, caractérisé en ce qua mise à jour du tableau Ui,j s'effectue ligne par ligne en utilisant la technique de retour de trace.

5. Procédé d'encodage d'un nombre entier selon l'une quelconque des revendications précédentes, caractérisé en ce qu'il comporte des étapes de libération des mémoires contenant les entiers Ui,j, un tableau Mi,j définissant par un symbole (L) l'état de libération des mémoires du tableau Ui,j, consistant à effectuer les étapes suivantes,

A. mettre à la valeur 1 toutes les cellules du tableau U telles que $U_{i,j} \neq 0$.

B. pour i = 1 à n-1 {

B.1. soit T_i le plus grand indice j tel que $U_{i,j}=1$

B.2. mettre le symbole L dans toutes les cellules $U_{i,Ti}$ }

C. mettre à la valeur 0 toutes les cellules telles que $U_{i,j}=1$

D. supprimer la dernière ligne j=h.

E. retourner le tableau $M_{i,j}$ résultant de l'exécution des étapes A à D.

6. Procédé de multiplication d'un entier a par un entier b, avec un résultat r, caractérisé en ce que l'entier a est encodé selon l'une quelconque des revendications précédentes.

7. Procédé de multiplication d'un entier a par un entier b, selon la revendication 6, caractérisé en ce qu'il comporte des étapes de libération des mémoires contenant les entiers Ui,j, un tableau Mi,j définissant par un symbole l'état de libération des mémoires du tableau Ui,j, consistant à effectuer les étapes suivantes :

A. initialiser r = 0

B. pour i = 0 à n-1 {

B.1. si $U_{i,0} \neq 0$ alors allouer la variable q[i] = b × a[i] }

C. pour j = 1 à h {

C.1. allouer la variable $q[S_j] = q[0]\ U_{0,j} + \ldots + q[n-1]\ U_{n-1,j}$

C.2. mettre à jour $r = r + q[S_j]\ 2^{t\,Sj}$

C.3. libérer les variables q[k] pour les k est tel que $M_{k,j} == L$ }

D. libérer toutes les variables q[ ] restantes allouées.

E. retourner le résultat r.

8. Circuit électronique apte à encoder un nombre entier selon l'une quelconque des revendications 1 à 5, ou à multiplier un entier a par un entier b selon l'une des revendications 6 et 7.

9. Circuit électronique selon la revendication 8 caractérisé en ce qu'il comporte une architecture à e multiplieurs disposés de façon parallèle, lesdits e multiplieurs exécutant chacun un calcul indépendant de e instances de la méthode d'encodage,

ledit calcul indépendant étant obtenu en adaptant le procédé de retour sur trace de la revendication 5 afin de déterminer e tableaux $U_{i,j^1},\ldots,U_{i,j^e}$ minimisant la dépendance entre les a[i] de sorte que les e processus lancés sur e multiplieurs sont mutuellement indépendants.

10. Circuit électronique selon la revendication 8 caractérisé en ce qu'il comporte une architecture à e multiplieurs disposés de façon parallèle, lesdits e multiplieurs exécutant chacun un calcul indépendant ou par un calcul synchronisé de e instances du procédé décrit dans la revendication 8.

ledit calcul synchronisé étant obtenu en adaptant le procédé de retour sur trace de la revendication 5 afin de déterminer e tableaux $U_{i,j^1},\ldots,U_{i,j^e}$ minimisant la dépendance entre les a[i] de sorte que les e processus lancés sur e multiplieurs sont mutuellement indépendants.

11. Circuit électronique selon la revendication 9 ou 10 caractérisé en ce qu'aucun des e processus exécutés sur les multiplieurs en parallèle n'utilise les entiers a[i] générés par un autre processus, les différents processus indépendants exécutés en parallèle partitionnant l'ensemble des entiers a[i] en e classes mutuellement disjointes dans le temps et dans l'espace mémoire, les calculs correspondant à chaque classe étant codés par les $U_{i,j^1},\ldots,U_{i,j^e}$.

12. Circuit électronique selon l'une quelconque des revendications 8 à 11, caractérisé en que qu'il possède un moyen de génération de clés cryptographiques issues de la multiplication de deux entiers dont l'un au moins est encodé selon la revendication 1.

13. Circuit électronique selon la revendication 12, caractérisé en que ledit moyen de génération de clés cryptographiques utilisant l'un des algorithmes pris dans l'ensemble suivant : Fiat-Shamir, ou Diffie-Hellman, ou RSA ou DSA ou tout algorithme opérant sur une courbe elliptique.

14. Composant de confiance sécurisé détachable ou fixe caractérisé en ce qu'il comporte un circuit électronique selon l'une quelconque des revendications 8 à 13.

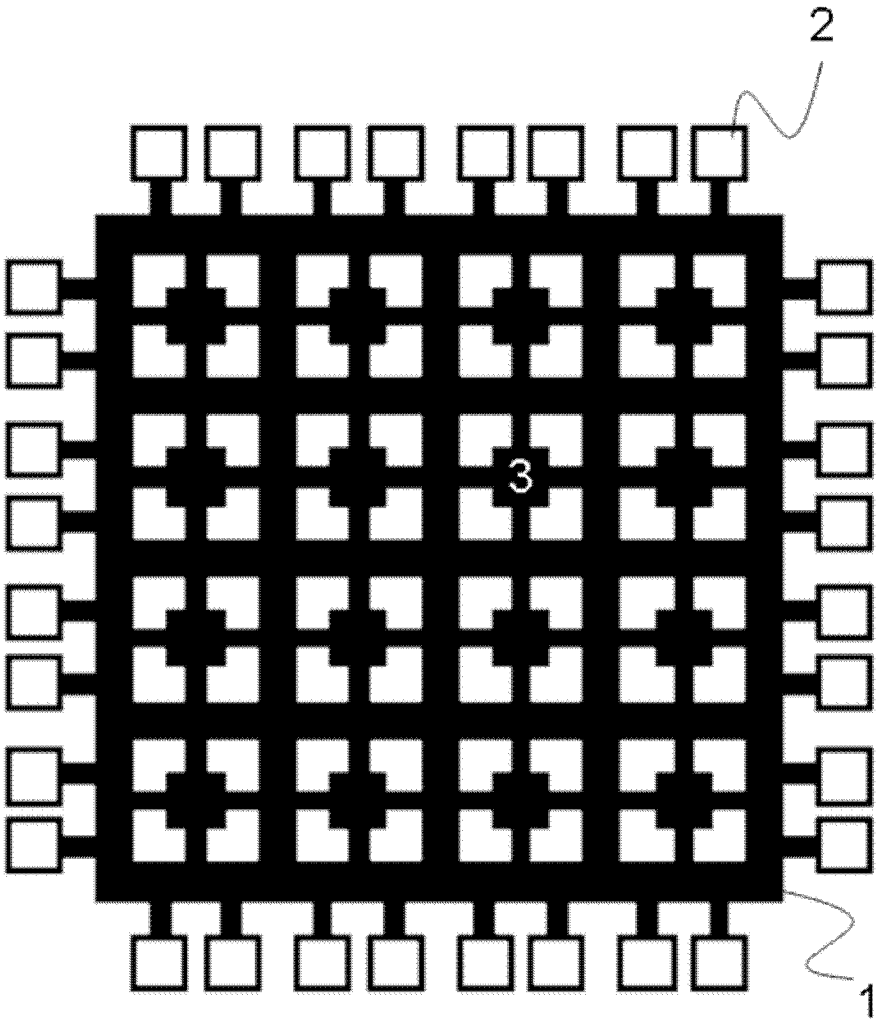
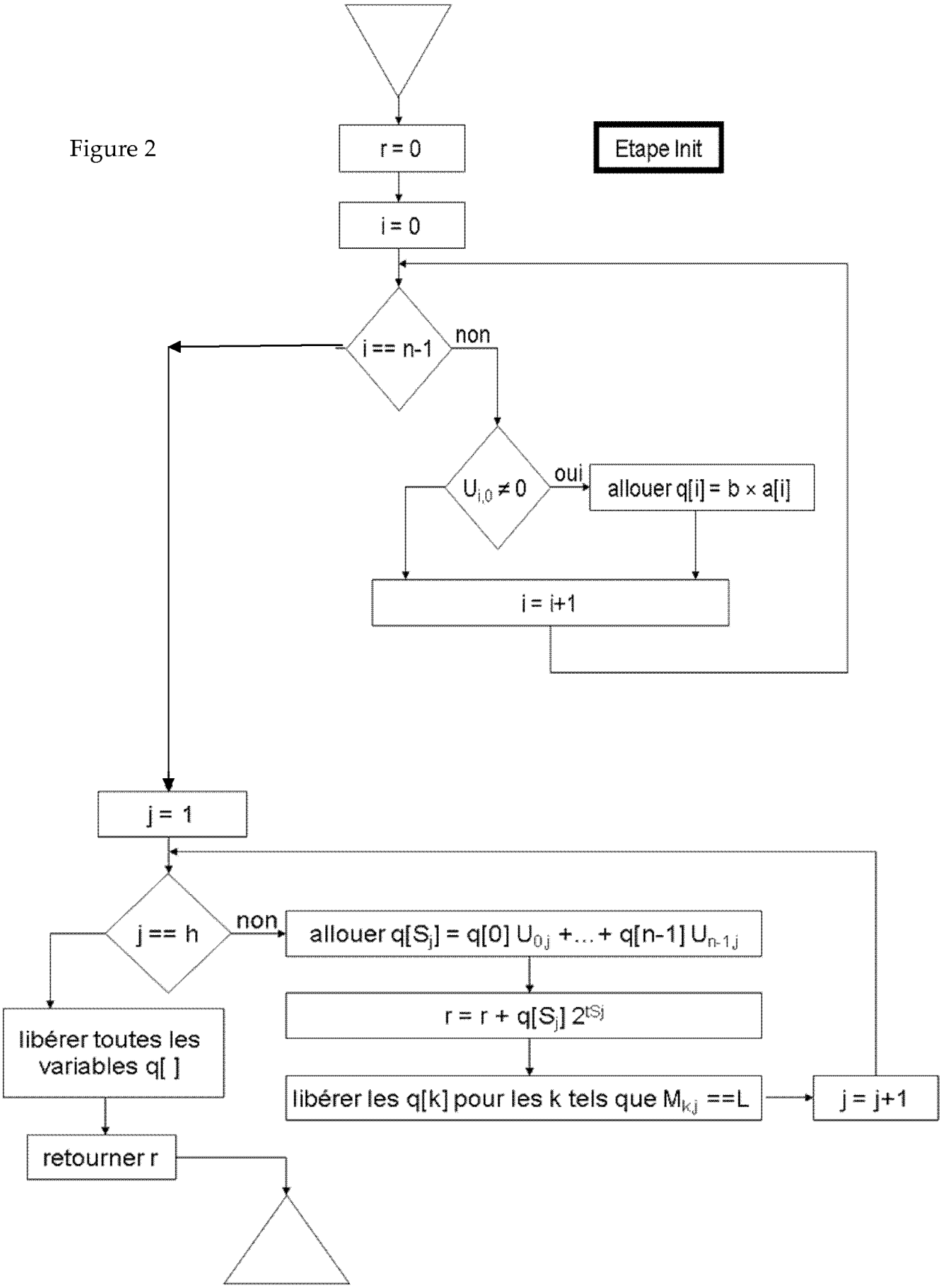


Figure 1

Figure 2


r = 0
Etape Init
i = 0
i == n-1
non
$U_{i,0} \neq 0$
oui
allouer q[i] = b × a[i]
i = i+1
j = 1
j == h
non
allouer $q[S_j] = q[0]\ U_{0j} + \ldots + q[n-1]\ U_{n-1,j}$
$r = r + q[S_j]\ 2^{tSj}$
libérer les q[k] pour les k tels que M_{kj} ==L
j = j+1
libérer toutes les variables q[ ]
retourner r