Opérateur matériel de tests de divisibilité par des petites constantes sur de très grands entiers

Karim Bigou, Thomas Chabrier, Arnaud Tisserand

ComPAS'2013 : RenPar'21/ SympA'15 / CFSE'9
Grenoble, France, du 16 au 18 janvier 2013

Opérateur matériel de tests de divisibilité par des petites constantes sur de très grands entiers

Karim Bigou[4,1], Thomas Chabrier[3,1] et Arnaud Tisserand[2,1]

[1]IRISA, UMR 6074 [2]CNRS – [3]Université Rennes 1, 6 rue de Kerampont, 22305 Lannion

[4]INRIA Rennes Bretagne Atlantique

Résumé
Dans ce papier, nous présentons un opérateur arithmétique matériel dédié aux tests de divisibilité par des petites constantes sur des grands entiers. Ces grands entiers, de plusieurs centaines de bits, sont représentés en multi-précision. La méthode proposée permet de n'effectuer qu'un très faible nombre de calculs pour chaque mot de la représentation multi-précision. Par exemple, elle permet de tester la divisibilité par $(2^a, 3, 5, 7, 9)$, où $1 \leq a \leq 12$, beaucoup plus efficacement qu'en testant la divisibilité par chacune des petites constantes séparément. La méthode proposée a été implantée et validée sur circuit FPGA.

Mots-clés : opérateur arithmétique, test divisibilité, nombre multi-précision, implantation matérielle, circuit FPGA.

---

1. Introduction

Dans certaines applications particulières, il est nécessaire de pouvoir tester rapidement si un entier est divisible par des constantes comme 2, 3, 5 ou d'autres petits premiers et des petites puissances de ces nombres comme 3^2. Ceci se fait assez facilement, en logiciel et en matériel, pour une seule constante et sur un nombre à tester de taille modérée (de la taille d'un mot machine en logiciel ou de quelques dizaines de bits en circuit). Mais lorsqu'il s'agit d'effectuer ces tests sur de grands entiers de plusieurs centaines de bits ou plus (p. ex. pour des tailles de nombres utilisés en cryptographie asymétrique) et pour plusieurs constantes à la fois, ceci nécessite des calculs élémentaires bien plus nombreux et ainsi des opérateurs coûteux à implanter en matériel. Ceci limite considérablement l'application de méthodes utilisant des tests de divisibilité de grands entiers en matériel.
Par exemple, en cryptographie sur les courbes elliptiques (ou ECC pour *elliptic curve cryptography*, cf. [5]), la multiplication scalaire, l'opération principale dans les protocoles, fait souvent appel à des algorithmes de recodage en bases multiples. Ces recodages permettent de réduire significativement le nombre total d'opérations à effectuer sur les points de la courbe elliptique considérée. En base double $(2, 3)$, on représente un nombre par une somme de termes de la forme $\pm 2^{e_1} 3^{e_2}$, voir [1] par exemple. Pour des bases multiples comme $(2, 3, 5, 7)$, la représentation se fait via la somme de termes de la forme $\pm 2^{e_1} 3^{e_2} 5^{e_3} 7^{e_4}$, voir [6] par exemple. Nous travaillons sur de tels recodages qui nécessitent des tests de divisibilité efficaces par les différents éléments des bases multiples (et si possible des petites puissances de ces premiers).

Dans cet article, nous présentons une méthode permettant d'effectuer simultanément et rapidement, et sur des petits circuits, les tests de divisibilité par plusieurs petites constantes comme $(2^a, 3, 5, 7, 9)$, avec a petit, sur de très grands entiers de plusieurs centaines de bits et représentés en multi-précision (c.-à-d. sous forme de vecteurs de mots de taille modérée). La méthode proposée repose sur l'adaptation d'une très ancienne méthode décrite par Blaise Pascal [8, 10]. Nous l'avons adaptée au cas de la divisibilité par plusieurs constantes et en matériel.
La section 2 présente les notations utilisées dans ce papier et quelques hypothèses qui seront utiles pour les implantations matérielles. L'état de l'art du domaine est présenté en section 3. La section 4 présente la version simple en base 2, directement issue de la méthode de Pascal, de l'opérateur de divisibilité et son implantation FPGA. La section 5 présente notre amélioration de la méthode en utilisant une base intermédiaire plus grande de type 2^v. Nous donnons aussi les résultats d'implantation FPGA pour cette amélioration. La section 6 décrit très brièvement des comparaisons avec d'autres travaux proches. Enfin, la section 7 présente la conclusion et quelques perspectives.

2. Notations et hypothèses d'implantation matérielle

L'*entier naturel* x est l'opérande dont la divisibilité doit être testée. Il est représenté en numération simple de position de base 2 (la notation binaire standard, cf. [7, p. 27] par exemple) sur n bits. On a donc :

$$x \quad = \quad (x_{n-1}x_{n-2}\dots x_1x_0)_2 \quad = \quad \sum_{i=0}^{n-1} x_i 2^i$$

La notation $()_2$ signifie que les éléments entre parenthèses sont les bits de la représentation de base 2 avec les poids faibles à droite. Dans nos applications cibles — l'implantation matérielle de crypto-processeurs ECC — les nombres ont des tailles dans l'intervalle $n \in [160, 600]$ bits, mais notre méthode fonctionne directement pour des tailles plus importantes. L'extension au cas des entiers relatifs est triviale et ne sera pas décrite dans cet article (car peu utile dans nos applications en cryptographie).
En pratique dans le circuit, un tel grand entier est représenté — et donc stocké — en multi-précision par un *vecteur* de t *mots* de taille w bits. Le nombre t de mots nécessaires pour représenter les entiers de n bits est donné par la relation : $w \cdot (t-1) < n \leq w \cdot t$. Si besoin, le mot de poids le plus fort est complété par des zéros (0-*padding*). On note $x^{(j)}$, avec $0 \leq j < t$, le j-ième mot de la représentation multi-précision de x en partant des poids faibles. Le stockage de x en multi-précision est illustré au niveau architecture en figure 1. L'adresse du j-ième mot de x est entrée dans le bloc mémoire et le contenu de ce mot $x^{(j)}$ sort sur le port de lecture. Dans nos implantations matérielles en FPGA, ce bloc mémoire sera implanté en LUT (*look-up table*) afin de pouvoir mesurer l'impact de n et w sur la surface de circuit utilisée. Bien évidement, notre méthode s'applique directement aux implantations avec des blocs dédiés de mémoire câblée des FPGA modernes (p. ex. les BRAM des FPGA Xilinx).
Nous noterons $\mathcal{D}$ l'ensemble des l diviseurs par lesquels on souhaite tester la divisibilité de x, on a ainsi $\mathcal{D} = (d_1, d_2, \dots, d_l)$. Nous désignons par d un des éléments de $\mathcal{D}$ (quand sa position dans $\mathcal{D}$ n'a pas d'importance afin d'alléger les notations). Les diviseurs de $\mathcal{D}$ envisagés pour les tests sont des petits nombres premiers comme 2, 3, 5 ou 7 ainsi que quelques puissances limitées de ces premiers comme 2^a (avec $a \leq w$ en pratique dans l'architecture) ou 3^2.
Les réalisations matérielles de ce papier ont été décrites en VHDL puis implantées sur un FPGA XC5VLX50T en utilisant l'environnement ISE 12.4, tous deux de la société Xilinx, avec des paramètres d'effort standard pour la synthèse et le placement/routage. Dans la suite, nous

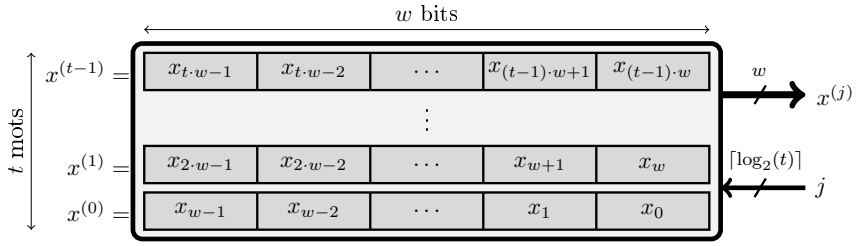

FIGURE 1 – Stockage de l'argument x sur t mots de w bits.

résumerons les résultats d'implantation, obtenus par les outils, en termes de nombre de cycles d'horloge, de surfaces exprimées en *slices* et de fréquences d'horloge. Les surfaces seront aussi indiquées en nombre de LUT (à 6 entrées dans Virtex 5) et nombre de bascules (FF pour *flip-flop*). Pour information, un XC5VLX50T contient 7 200 *slices* de 4 LUT-6 et 4 FF par *slice*.

3. État de l'art

Il existe de nombreuses références qui présentent des tests de divisibilité. Par exemple, les tests élémentaires comme la divisibilité par 3, 5, 9 ou 10 en base 10 se trouvent dans les livres de mathématiques pour le collège. Des listes plus étoffées de tests de divisibilité en bases 10 et 2 se trouvent sur des sites comme Wikipédia. Enfin, des livres bien plus techniques comme [11] proposent des astuces logicielles pour effectuer certains tests très rapidement en utilisant de courtes séquences d'instructions arithmétiques et logiques. Mais il existe très peu de référence sur l'implantation matérielle de ces tests, en particulier pour des grands nombres (cf. section 6). Toutes ces méthodes utilisent, plus ou moins directement et avec des adaptations, l'idée de base présentée ci-dessous.

Blaise Pascal (1623–1662) a proposé une méthode générale permettant de tester la divisibilité de x par d dans une représentation de base b quelconque (avec x en numération simple de position, c.-à-d. $x = \sum_{i=0}^{n-1} x_i b^i$). Cette méthode est décrite dans une publication posthume, en latin, de 1819 [8]. Une analyse moderne et en anglais de ce texte se trouve dans [10] (on trouve sur Internet une version équivalente de ce texte et en français sous le titre « La machine à diviser de Monsieur Pascal »). Cette méthode est souvent appelée *ruban de Pascal*. La méthode décrite par Pascal est peut-être encore plus ancienne, mais nous ne connaissons pas de texte antérieur présentant cette méthode en base b quelconque.

Pascal a remarqué que les valeurs des restes $b^i \bmod d$, avec $i \in \mathbb{N}$ et d premier avec la base b, forment une séquence périodique très simple dans certains cas. Dans la suite de ce papier, nous nous limiterons au cas de la base $b = 2$ ou $b = 2^v$ avec v petit. La table 1 présente les restes $2^i \bmod d$ pour $d \in \{3, 5, 7\}$ et $i \leq 16$. Chaque ligne de la table 1 présente le ruban de Pascal correspondant à d.

À partir de la table 1, on peut utiliser le ruban de Pascal pour tester facilement la divisibilité par $d = 3$, pour lequel la séquence périodique est $(2\,1)^*$. Ce ruban indique que :

$$\begin{aligned} x \bmod 3 &= (\ldots + 2^5 x_5 + 2^4 x_4 + 2^3 x_3 + 2^2 x_2 + 2^1 x_1 + x_0) \bmod 3 \\ &= (\ldots + 2\ x_5 + \quad x_4 + 2\ x_3 + \quad x_2 + 2\ x_1 + x_0) \bmod 3 \\ &= \left(\underbrace{\sum (2x_{2i+1} + x_{2i})}_{\alpha} \right) \bmod 3 \end{aligned}$$

	i																
d	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
3	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	**2**	**1**
5	1	3	4	2	1	3	4	2	1	3	4	2	1	**3**	**4**	**2**	**1**
7	2	1	4	2	1	4	2	1	4	2	1	4	2	1	**4**	**2**	**1**

TABLE 1 – Rubans de Pascal pour la divisibilité par $d \in \{3, 5, 7\}$ (les valeurs sont $2^i \bmod d$).

Cela signifie qu'il faut commencer par sommer les sous-mots de taille 2 bits de type $(x_{2i+1}\, x_{2i})_2$ sur toute la largeur de l'opérande. La somme obtenue est alors congrue à 3 dans le cas où x est divisible par 3. En appliquant récursivement cette méthode sur l'écriture de la somme obtenue pour α (la taille de α dépend de t), comme illustré à la figure 2, on obtient une valeur pour laquelle il suffit de tester si elle est égale à 0 ou à 3. Si la valeur finale obtenue est différente de 0 et de 3 alors x n'est pas divisible par $d = 3$.

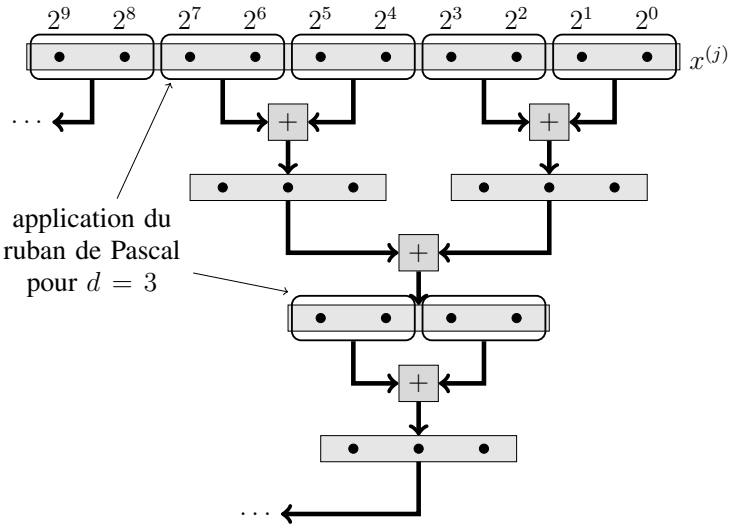

FIGURE 2 – Application récursive du ruban de Pascal pour $d = 3$.

Cette méthode s'applique directement au cas $d = 7$ pour lequel la séquence périodique est $(4\,2\,1)^*$. Ce qui signifie qu'il faut faire la somme α (spécifique à cette valeur de d) des sous-mots de taille 3 bits et de la forme $(x_{3i+2}\, x_{3i+1}\, x_{3i})_2$ le plus possible (en appliquant la décomposition récursivement), puis enfin tester si la somme réduite finale est égale à 0 ou 7.
Mais dans le cas $d = 5$, la séquence périodique est $(3\,4\,2\,1)^*$. On ne peut plus faire la somme par sous-mots de taille 4 bits du fait du facteur 3 (3 n'étant pas une puissance de 2). Pour résoudre ce problème, deux solutions s'offrent à nous : considérer $3 = 1 + 2$ ou bien considérer $3 \equiv -2 \bmod 5$. La première solution consiste à utiliser le bit de rang $4i + 3$ comme entrée de l'addition aux rangs $4i+1$ et $4i$ simultanément. La seconde nécessite d'utiliser la représentation en complément à deux pour effectuer toutes les sommes intermédiaires pour obtenir α.
Le test de divisibilité par 2^a avec $a \in \mathbb{N}$ est trivial pour x représenté en base 2. Il suffit en effet de regarder si les a bits de poids faibles sont nuls ou non.

4. Méthode utilisant directement les rubans de Pascal en base 2

Dans un premier temps, nous avons utilisé directement la méthode des rubans de Pascal en base 2, c.-à-d. en considérant les sommes de sous-mots de 2 bits pour $d = 3$ et 3 bits pour $d = 7$ par exemple. L'architecture correspondante est présentée en figure 3. Nous avons implanté une version pour $\mathcal{D} = (2^{1\cdots a}, 3, 5, 7)$. Les blocs de somme « $\sum$ » calculent la valeur de α propre à chaque diviseur d de $\mathcal{D}$ différent de $2^{1\cdots a}$. En plus, dans ces blocs, nous appliquons le ruban de Pascal pour chaque d afin de réduire la taille du registre d'accumulation. Donc, nous n'accumulons pas réellement la somme α, mais la somme α après application du ruban de Pascal. Lors de cette accumulation/réduction, il faut traiter les t mots de l'opérande x. Les blocs « $\mathcal{R}$ », quant à eux, effectuent les toutes dernières applications du ruban de Pascal ainsi que le test final (p. ex. tester si on a une valeur égale à 0 ou à d). Les signaux d'horloge et de contrôle ne sont pas représentés sur la figure 3.

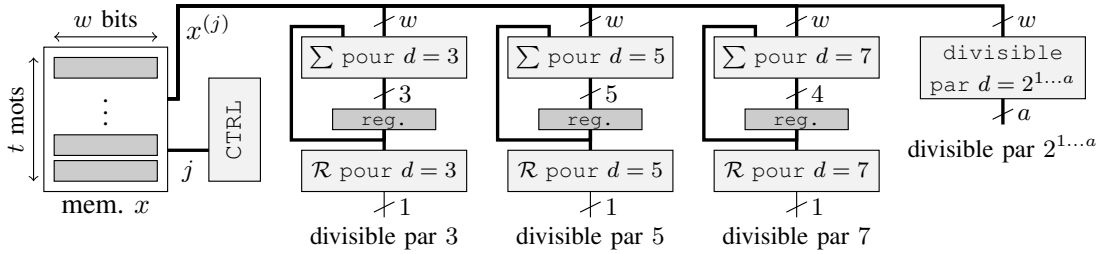

FIGURE 3 – Architecture des tests de divisibilité par $\mathcal{D} = (2^{1\cdots a}, 3, 5, 7)$ en utilisant directement les rubans de Pascal en base 2.

Nous avons testé les deux solutions pour le test de divisibilité par $d = 5$: considérer $3 = 1 + 2$ ou bien considérer $3 \equiv -2 \bmod 5$. C'est la version non signée, $3 = 1 + 2$, qui s'avère être la plus efficace en vitesse et en surface. Ceci est très probablement lié au surcoût de l'extension de signe pour les additions/soustractions en complément à deux.

L'architecture de la figure 3 permet de tester la divisibilité par chacun des éléments de $\mathcal{D}$ de façon simultanée et en un seul parcours des mots de la représentation de x. Le nombre de cycles d'horloge nécessaire est $t + O(1)$ où la constante dépend de la taille w des mots. Le paramètre w influence grandement les performances et la taille de l'opérateur. Les longueurs des séquences périodiques de restes $2^i \bmod d$ pour $d = 3, 5, 7$ sont respectivement $2, 4, 3$ (voir table 1). Afin d'éviter un décodage complexe, nous utilisons le plus petit commun multiple de ces longueurs pour la valeur de w. Dans notre cas particulier, on a donc $w = \mathrm{ppcm}(2, 4, 3) = 12$. En pratique, on peut utiliser des multiples du ppcm. Nous avons testé $w = 12$ et $w = 24$ (de plus grandes valeurs réduisent beaucoup la fréquence d'horloge et nécessitent l'introduction d'étages de pipeline supplémentaires).

Les résultats d'implantation en FPGA sont résumés dans la table 2 pour des opérandes de $n = 160$ bits (la plus petite taille utile pour nos applications ECC) et pour des tests de divisibilité par $\mathcal{D} = (2^{1\cdots a}, 3, 5, 7)$. Nous avons utilisé $a = 12$ quelque soit la valeur w. Ce choix du paramètre a limité à 12 est justifié par des évaluations statistiques sur nos applications ECC.

Dans la table 2, nous exprimons le nombre de cycles d'horloge nécessaire pour effectuer les tests de divisibilité simultanés en fonction de t (le nombre de mots de x). En effet, il faut lire chaque mot pour accumuler sa contribution à α. La valeur de t utilisée est déterminée par les paramètres n et w (n pour l'application et w pour l'architecture). On rappelle que $w \cdot (t - 1) <$

w	t	surface *slices* (FF/LUT)	fréquence MHz	nombre de cycles
12	14	37 (90/100)	418	$t+3$
24	7	42 (100/105)	408	$t+4$

TABLE 2 – Résultats d'implantation sur FPGA des tests de divisibilité par $\mathcal{D} = (2^{1\cdots a}, 3, 5, 7)$ utilisant directement les rubans de Pascal en base 2 et pour $n = 160$ bits.

$n \leq w \cdot t$. La fréquence de l'opérateur, quant à elle, dépend fortement des paramètres t et w (et donc indirectement aussi de n).
Notre méthode fonctionne sans aucun problème pour des tailles plus importantes que $n = 160$. Le surcoût en surface de circuit est alors celui du stockage d'arguments de plus grande taille. Ces plus grands arguments nécessitent un compteur de boucle un peu plus large (la taille du compteur étant en $\lceil \log_2(t) \rceil$). Enfin, nous avons validé fonctionnellement nos opérateurs par des simulations aléatoires intensives.

5. Amélioration de la méthode en utilisant les rubans de Pascal en grande base 2^v

Appliquer directement la méthode des rubans de Pascal à des tests de divisibilité plus nombreux, comme $\mathcal{D} = (2^{1\cdots a}, 3, 5, 7, 9, 11, 13)$, complique sensiblement le contrôle et augmente le nombre de registres internes utilisés pendant la boucle d'accumulation. La table 3 fournit les rubans de Pascal pour $d \in \{9, 11, 13\}$ à partir des bits de x (c.-à-d. les restes sont les valeurs $2^i \bmod d$). Elle montre clairement qu'avec des valeurs de d plus grandes, la longueur des séquences périodiques et leur « complexité » (forme des coefficients) croît de façon importante. Devoir organiser le décodage de ce qu'il convient de faire de chaque bit en fonction du ppcm de ces longueurs va engendrer un contrôle bien plus complexe. De plus, il faut accumuler les valeurs α propres à chaque diviseur d dans des registres distincts.

	i																			
d	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
9	2	1	5	7	8	4	2	1	5	7	8	4	2	1	**5**	**7**	**8**	**4**	**2**	**1**
11	6	3	7	9	10	5	8	4	2	1	**6**	**3**	**7**	**9**	**10**	**5**	**8**	**4**	**2**	**1**
13	11	12	6	3	8	4	2	1	**7**	**10**	**5**	**9**	**11**	**12**	**6**	**3**	**8**	**4**	**2**	**1**

TABLE 3 – Rubans de Pascal en base 2 pour $d \in \{9, 11, 13\}$ (les valeurs sont $2^i \bmod d$).

Afin d'étudier les liens entre les longueurs et la forme des séquences périodiques dans les rubans de Pascal d'une part, et les performances et le coût des opérateurs nécessaires pour des tests plus complexes d'autre part, nous avons modifié différents paramètres arithmétiques et de l'architecture.
En faisant varier ces paramètres, nous avons remarqué une propriété très intéressante si l'on considère les restes de sous-mots au lieu des restes des bits (au sens arithmétique) directement. Supposons que l'on découpe les mots de w bits en sous-mots de v bits avec $v \leq w$, c.-à-d. que l'on lit les chiffres de x en base $b = 2^v$. En considérant les restes $\left(2^v\right)^i \bmod d$ au lieu de

$2^i \bmod d$, on obtient des séquences périodiques très intéressantes pour certaines valeurs de v et de d. En particulier pour $v = 12$, on obtient les rubans de Pascal présentés à la table 4 pour $d \in \{3, 5, 7, 9, 11, 13, 16, 25\}$. Dans cette table, les valeurs sont les restes $\left(2^{12}\right)^i \bmod d$ pour $0 \leq i \leq 9$.

	i									
b	9	8	7	6	5	4	3	2	1	0
3	1	1	1	1	1	1	1	1	1	**1**
5	1	1	1	1	1	1	1	1	1	**1**
7	1	1	1	1	1	1	1	1	1	**1**
9	1	1	1	1	1	1	1	1	1	**1**
11	3	9	5	4	1	**3**	**9**	**5**	**4**	**1**
13	1	1	1	1	1	1	1	1	1	**1**
17	16	1	16	1	16	1	16	1	**16**	**1**
25	6	11	16	21	1	**6**	**11**	**16**	**21**	**1**

TABLE 4 – Rubans de Pascal en base 2^{12} pour $d \in \{3, 5, 7, 9, 11, 13, 16, 25\}$ (les valeurs sont $\left(2^{12}\right)^i \bmod d$).

Les lignes qui correspondent aux diviseurs 3, 5, 7, 9 et 13 de la table 4 présentent la même séquence périodique $(1)^*$. Ces lignes signifient que les tests correspondants peuvent être effectués en utilisant un unique registre d'accumulation. Le même accumulateur est alors partagé pendant les t cycles de l'accumulation, puis il faut effectuer une réduction finale propre à chaque diviseur dans notre nouvelle architecture illustrée en figure 4. L'optimisation provient de l'accumulation d'une seule valeur α partagée par tous les diviseurs ayant $(1)^*$ comme séquence périodique. Afin de tirer parti de cette propriété, il convient de choisir pour w un multiple de v (c.-à-d. $w = v$, $w = 2 \cdot v$, $w = 3 \cdot v$, ...). Dans notre cas, nous avons implanté la version améliorée pour $v = 12$ et $w \in \{12, 24\}$. Dans cette version, la sortie du registre d'accumulation est plus grande ($w + \lceil \log_2(t) \rceil$ bits) que les sorties des registres de la figure 3. Les blocs de réduction « $\mathcal{R}'$ » propres à chaque diviseur utilisent la technique spécifique de base $b = 2$ présentée en section 3 (figure 2).

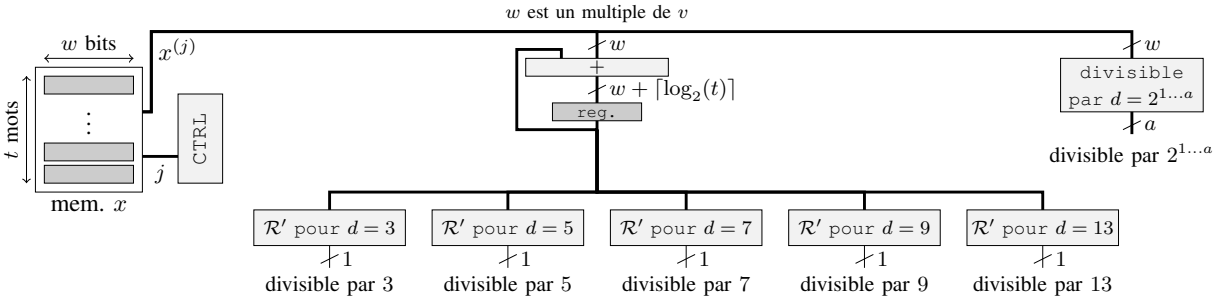

FIGURE 4 – Architecture des tests de divisibilité par $\mathcal{D} = (2^{1...a}, 3, 5, 7, 9, 13)$ en utilisant l'amélioration en base 2^v des rubans de Pascal.

Les résultats d'implantation en FPGA pour notre méthode améliorée sont donnés dans la table 5 pour des opérandes de $n = 160$, 256 et 521 bits (qui correspondent à des tailles cryptographiques assez classiques pour ECC) et pour des divisibilités par $\mathcal{D} = (2^{1\cdots a}, 3, 5, 7, 9, 13)$. Ici aussi, nous avons utilisé $a = 12$ pour les différentes valeurs de w.

n	w	t	surface *slices* (FF/LUT)	fréquence MHz	nombre de cycles
160	12	14	49 (112/135)	454	$t+3$
	24	7	72 (176/198)	490	$t+4$
256	12	22	54 (127/149)	460	$t+3$
	24	11	74 (188/205)	495	$t+4$
521	12	44	56 (129/155)	424	$t+3$
	24	22	74 (192/208)	485	$t+4$

TABLE 5 – Résultats d'implantation sur FPGA des tests de divisibilité par $\mathcal{D} = (2^{1\cdots a}, 3, 5, 7, 9, 13)$ utilisant l'amélioration des rubans de Pascal en base 2^{12} et pour $n \in \{160, 256, 521\}$ bits.

6. Comparaisons

On trouve dans [2] une application de la représentation modulaire des nombres (ou RNS pour *residue number system*) au test de divisibilité en matériel mais sans aucun résultat d'implantation. La méthode décrite dans [2] suppose que l'opérande x soit représenté en RNS ce qui n'a pas vraiment de sens pour les applications envisagées en cryptographie ECC. On peut représenter les coordonnées des points d'une courbe elliptique (éléments d'un corps fini) avantageusement en RNS (voir p. ex. [3, 4]). Mais nous ne connaissons pas de proposition de cryptosystème dans lequel le scalaire k est représenté en RNS pour l'opération de multiplication scalaire $[k]P$.
On trouve sur Internet le texte correspondant à un poster [9] prétendument publié à la conférence FCCM (*Field-Programmable Custom Computing Machines*) 2002 mais ce papier n'est pas dans la liste des posters ou papiers présentés à la conférence (ni pour les autres années à FCCM, ni sur DBLP et IEEEXplore qui stockent pourtant toutes les publications à FCCM, ni dans une autre conférence). Ce texte présente, sans aucun détail, des résultats d'implantation en FPGA de tests de divisibilité pour des nombres en multi-précision avec un temps de calcul qui semble quadratique (en tous cas non-linéaire). Notre méthode a un temps de calcul seulement linéaire en $t + O(1)$ cycles et où le terme constant est tout petit (3 ou 4 selon les versions).
Nous ne connaissons pas d'autre référence qui traite d'implantation matérielle de tests de divisibilité pour des grands entiers et qui en détaille les performances. Ceci est probablement lié au fait que l'utilisation de tels tests de divisibilité sur des grands nombres en matériel est rarissime. Nous espérons que ce travail permettra d'utiliser ces tests de divisibilité du fait de leur assez bonne efficacité.

7. Conclusion

Nous avons proposé une architecture d'opérateur arithmétique matériel dédié aux tests de divisibilité par des petites constantes pour de grands entiers. La méthode proposée permet d'effectuer simultanément différents tests comme les divisibilités par $(2^{1\cdots a}, 3, 5, 7, 9, 13)$ avec $a \leq 12$ en une seule lecture des chiffres de l'opérande à tester. Les résultats d'implantation FPGA montrent que ces tests s'effectuent très rapidement (tant au niveau de la fréquence de l'opérateur que du nombre de cycles d'horloge nécessaire) et enfin sur de petits circuits.
Dans l'avenir, nous souhaitons pouvoir évaluer l'utilisation d'autres propriétés arithmétiques pour permettre les tests de divisibilité par encore plus de diviseurs.

Remerciements

Nous remercions chaleureusement Christiane Frougny pour nous avoir indiqué les références historiques [8] et [10]. Ce travail a été financé en partie par une bourse de thèse de la Région Bretagne et du Conseil Générale des Côtes d'Armor (projet Robusta) et en partie par une thèse DGA–INRIA. Il a aussi été soutenu en partie par le financement du projet ANR Blanc 2012 PAVOIS (`http://pavois.irisa.fr/`, réf : ANR-12-BS02-002-01) et du projet ANR INS 2011 ARDyT (`http://ardyt.irisa.fr/`), réf : ANR-11-INSE-15). Nous remercions les relecteurs anonymes pour leurs remarques et corrections.

Bibliographie

1. Dimitrov (V.), Imbert (L.) et Mishra (P. K.). – The double-base number system and its application to elliptic curve cryptography. *Mathematics of Computation*, vol. 77, n262, avril 2008, pp. 1075–1104.
2. Gamberger (D.). – Incompletely specified numbers in the residue number system-definition and applications. *In : Proc. 9th IEEE Symposium on Computer Arithmetic*. pp. 210–215. – Santa Monica, CA, USA, septembre 1999.
3. Guillermin (N.). – A high speed coprocessor for elliptic curve scalar multiplications over fp. *In : Proc. Cryptographic Hardware and Embedded Systems (CHES)*. pp. 48–64. – Santa Barbara, USA, août 2010.
4. Guillermin (N.). – *Implémentation matérielle de coprocesseurs haute performance pour la cryptographie asymétrique*. – Phd thesis, Université Rennes 1, janvier 2012.
5. Hankerson (D.), Menezes (A.) et Vanstone (S.). – *Guide to Elliptic Curve Cryptography*. – Springer, 2004.
6. Longa (P.) et Gebotys (C.). – Fast multibase methods and other several optimizations for elliptic curve scalar multiplication. *In : Proc. Public Key Cryptography (PKC)*, pp. 443–462.
7. Muller (J.-M.). – *Arithmétique des ordinateurs*. – Masson, 1989.
8. Pascal (B.). – *Œuvres complètes*, chap. De Numeribus Multiplicibus, vol. 5, pp. 117–128. – Librarie Lefèvre, 1819.
9. Raman (E.), Chakrapani (L. N.), Sankaranarayanan (K.) et Parthasarathi (R.). – A scalable reconfigurable architecture for divisibility testing of variable long precision numbers. – Personnal webiste from one author.
10. Sakarovitch (J.). – *Elements of Automata Theory*, chap. Prologue : M. Pascal's Division Machine, pp. 1–6. – Cambridge, 2009.
11. Warren (H. S.). – *Hacker's Delight*. – Addison-Wesley, 2003.