Une implémentation matérielle de CS-Cipher sur carte PCI Pamette

Pierre Corbineau, Antoine Kremer et Sébastien Villemot
Ecole Normale Supérieure (Paris)
e-mail: {Pierre.Corbineau, Antoine.Kremer, Sebastien.Villemot}@ens.fr

14 juin 1999

Abstract

In this article we present a hardware implementation of CS-Cipher, a secret-key block ciphering algorithm mostly developped at the Ecole Normale Supérieure [1]. We use a Xilinx XC4020E based PCI Pamette board [2], together with the PamDC library as software development environment [3]. We reached the performances foreseen by Jacques Stern and Serge Vaudenay. The M-box takes 20 combinatory logic blocks, which leads to an encryption rate of 528Mbps on a single XC4020E (the design uses the PCI clock, that is to say 33Mhz). The 2Gbps rate should be reachable on a four times bigger chip, with 64-bit wide busses.

1 Présentation générale

Cet article tente de décrire de façon synthétique l'implémentation de CS-Cipher réalisée dans le cadre du cours de Mark Shand et Jean Vuillemin. Une connaissance minimale de cet algorithme est supposée par la suite ; pour une description complète, voir [1].

Cette réalisation occupe deux des quatre FPGA de la carte PCI Pamette [2]; LCA0 est utilisé pour le chiffrement, LCA1 pour l'interface PCI et l'expansion de clefs.

La seule horloge utilisée pour l'ensemble de l'implémentation est celle du bus PCI, cadencé à 33Mhz.

2 Le module de chiffrement

2.1 Les boîtes P

La boîte P constitue le coeur de l'algorithme CS-Cipher, et a donc demandé une attention particulière pour réduire au maximum son coût en place et en délai de propagation.

C'est une fonction $8b \to 8b$, qui fait appel à deux fonctions f,g : $4b \to 4b$. Comme g n'a pas d'expression simple, des blocs de ROM internes du FPGA sont utilisés (`g1r/g1l`) [1]. Au contraire, un bit de sortie de f ne dépend que de deux bits d'entrée ($f(x) = not(not(x)\&rol(x))$, où rol est la rotation d'un bit à gauche). Le premier appel à f est donc groupé avec le xor en aval, pour donner des LUT $3b \to 1b$ (`g0r/g0l`). De même, le deuxième appel à f et le xor en amont sont regroupés pour obtenir des LUT $4b \to 1b$ (`f2r/f2l`). Ce même xor est, de plus, recalculé dans une boîte H (`outrr/outrl`) ; c'est une perte de place compensée par un gain réel en profondeur combinatoire. Les xor finaux après le deuxième appel à f sont également placés dans des boîtes H (`outlr/outll`).

1. Les symboles entre parenthèses désignent les variables dans le fichier source PamDC.

2.2 Les boîtes M

La solution retenue occupe un espace total de 20 CLB, répartis en deux blocs 4×2 et un bloc 4×1, pour une latence de 2 cycles entre l'entrée et la sortie des données. Elle prend en entrée deux mots de 16 bits - la donnée et une constante (clef ou constante réelle) - et renvoie une sortie de 16 bits. Chacune des deux boîtes P est placée dans un bloc 4×2, accompagnée d'une partie des *xor* d'entrée; le reste de ces *xor* est placé dans le bloc 4×1. Une barre de registres est disposée avant et après les boîtes P.

2.3 Une ronde de chiffrement

Elle consiste essentiellement en l'agencement de 3 rangées de 4 boîtes M ; de plus, 2 des 3 constantes sont indépendantes de la clef de chiffrement, ce qui permet de les intégrer dans les fonctions combinatoires des boîtes M correspondantes.

En imbriquant les boîtes M, on arrive à une surface de 10×24 CLBs, pour une profondeur de 6 registres.

2.4 Architecture globale de la puce de chiffrement

L'espace disponible dans les FPGA XC4020E de la carte utilisée ne permet de mettre que deux rondes de chiffrement ; elles sont placées en "série", et une rangée de registres est rajoutée à la fin de la deuxième ronde. Les données rentrant dans ce pipeline ressortent donc avec une latence de 13 cycles. Un bloc de message en clair doit parcourir 4 fois ce pipeline, et ensuite traverser le *xor* avec le 9ème bloc de clef, pour donner le résultat chiffré, soit un délai de 56 cycles.

Cette architecture permet de chiffrer un bloc de 64 tous les 4 cycles, soit 16 bits par cycles. En effet, en présentant une nouvelle donnée à l'entrée du pipeline tous les 4 cycles, il n'y aura pas de collision avec d'autres données dans le pipeline, ce qui peut s'expliquer simplement par le fait que 4 est inversible modulo 13 (la profondeur en registres du pipeline), donc générateur du groupe des entiers modulo 13.

L'ensemble des registres du module de chiffrement est placé sous un *enable* qui est directement donné par le FPGA d'interface ; ce dernier active ce signal tous les cycles où 16 bits de données valides sont présent sur le bus reliant les deux FPGA. Le retour des données chiffrées se fait sur 16 autres bits du bus.

De plus, un dispositif de téléchargement des clefs est prévu ; ce transfert se fait également sur 16 bits du même bus, avec un signal indiquant la présence de données valides. Pour des raisons pratiques, on n'envoie pas 9 mais 10 blocs de clef de 64 bits, le dernier étant le *not* de l'avant dernier ; le transfert dure donc 40 cycles. En effet, une RAM interne dont on n'utilise que deux bits d'adresses est employée pour effectuer le *xor* final.

3 L'expansion de clefs

Elle est implémentée dans le même chip que l'interface, c'est-à-dire LCA1.

3.1 Une itération

Une itération de l'expansion de clef consiste - outre des *xor* initiaux et finaux - en un agencement de 8 boîtes P, dont le placement suit les mêmes règles que celles décrites précédemment. Un découpage de l'itération a été fait de telle sorte que 8 bits soient traités sur une bande horizontale de 2 CLBs de hauteur, permettant de rendre le routage le plus régulier possible. Le résultat avant la "transposition" est placé à l'extrémité droite de ces bandes pour diminuer la complexité du routage (ce placement est possible car on reboucle l'itération sur elle-même, avec la mécanique de registres adéquate).

3.2 Expansion complète des 9 sous-blocs de clef

Elle est déclenchée par un signal contrôlé par l'interface. Comme expliqué précédemment, elle dure 40 cycles (et non pas 36) en utilisant un compteur modulo 10 et une seule instance physique d'une itération.

4 L'interface PCI

4.1 Description

Dans LCA1 se trouve également l'interface qui permet les échanges avec l'hôte. Dans l'état actuel du projet, seul les transferts de données depuis l'hôte vers la carte sont réellement gérés. L'hôte peut envoyer des données à chiffrer jusqu'à concurrence de 128ko ; le résultat du chiffrement sera écrit dans la RAM près de LCA1. Pour lire ce résultat, on peut par exemple utiliser l'application standard `loadram` de la distribution Pamette. Une mécanique de retour des données permettrait de rendre la carte complètement fonctionnelle ; le recours à des accès DMA semble la solution la plus adaptée.

4.2 Echanges avec l'hôte

La carte PCI Pamette est utilisée en mode *transaction* pour échanger des données avec l'hôte. Voici l'utilisation faite de l'espace d'adresses utilisateur:

```
0x20000 ... 0x2003F: zone pour écrire les données à chiffrer
0x21000 ... 0x2100F: les 128 bits pour écrire la clef
0x22000: pour déclencher l'expansion de clef
```

Une transaction normale avec la carte se fait comme suit :

- L'hôte écrit tout d'abord la clef de chiffrement dans l'espace de 128 bits prévu à cet effet.
- Une écriture (de valeur quelconque) est ensuite faite en `0x22000` pour déclencher l'expansion de la clef et son téléchargement dans le module de chiffrement.
- Les données à chiffrer sont ensuite écrites dans l'espace de 64 octets, soit 8 blocs de 64 bits, prévu à cet effet. On doit tout d'abord commencer par écrire le premier bloc en `0x20000` et `0x20004`, les autres blocs aux adresses suivantes, en parcourant ces dernières circulairement de `0x20000` à `0x2003F`. Il faut également veiller à ne pas effectuer ces écritures trop rapidement ; voir la section suivante à ce sujet.

Remarque 1: Tous les échanges de données se font en *little endian*.

Remarque 2: Les 14 derniers blocs de 64 bits écrits seront perdus dans le pipeline de chiffrement ; ce comportement pourrait être modifié en réalisant la mécanique de retour des données.

4.3 Architecture de l'interface

La gestion du buffer d'entrée de 8 blocs de 64 bits se fait avec les RAMs internes double-port ; deux compteurs, qui servent de pointeurs, sont utilisés. Un pour l'adresse dans ce buffer où se fera la prochaine écriture de donnée provenant de l'hôte, et un autre pour l'adresse où se fera la prochaine lecture pour envoyer les données à LCA0.

Comme - en théorie - l'hôte peut envoyer 32 bits à chiffrer tous les cycles, tandis que la carte ne peut en gérer que 16, il importe de faire attention à ne pas remplir trop rapidement le buffer de 8 blocs. On peut imaginer un raffinement de l'interface avec un protocole pour gérer plus élégamment ce problème (ou même utiliser des accès DMA).

Les données reçues de LCA0 sont écrites dans le bloc de RAM statique situé sous LCA1 ; la nécessité d'écrire 16 bits tous les cycles oblige ici à sortir légèrement du modèle de logique synchrone pour générer le *write enable* de la RAM.

5 Conclusion

Cette réalisation vient donc confirmer les estimations données par Jacques Stern et Serge Vaudenay sur les performances d'une implémentation matérielle. En régime maximal, un taux de 528Mbps est réalisable avec le FPGA dédié au chiffrement (16 bits par cycle à une cadence de 33Mhz). On n'atteint pas ici les 2Gbps espérés, mais les facteurs limitants sont uniquement l'espace disponible dans les XC4020 et la largeur des bus pour communiquer sur la carte ou avec l'hôte. Pour obtenir un tel taux à même cadence, il faudrait quatre fois plus d'espace dans le FPGA et des bus d'une largeur de 64 bits au lieu de 16 et 32.

On peut imaginer différentes extensions à cette réalisation : des accès DMA pour récupérer les données chiffrées, ou encore un deuxième module de chiffrement dans LCA2 qui permettrait de certifier l'exactitude des résultats donnés par LCA0.

Remerciements

Nous tenons à remercier Mark Shand, Thomas Pornin, Pierre-Alain Fouque et Jean Vuillemin pour leur disponibilité et les nombreux conseils qu'ils nous ont prodigués.

Références

[1] Jacques Stern and Serge Vaudenay. CS-Cipher. In *Fast Software Encryption, Paris, France, Lecture Notes in Computer Science No. 1372*, pp. 189-205, Springer-Verlag, 1998.

[2] PCI Pamette v1, *http://www.research.digital.com/SRC/pamette/*

[3] PCI Pamette v1 - Software, *http://www.research.digital.com/SRC/pamette/Software.html*