Vol.2019-ARC-240 No.42
Vol.2019-SLDM-191 No.42
Vol.2019-EMB-53 No.42
2019/2/28

FPGA SoCにおける多倍長演算の実装

田中 清史[1,a)]

概要：公開鍵暗号方式において主流である RSA 暗号は，安全性確保のために長い鍵長を必要とするため組込み機器での利用が困難である．一方，楕円曲線暗号は短い鍵長で同等の安全性を提供可能であり，メモリ量が限られたシステムに適用可能である．楕円曲線暗号の処理では基本多倍長演算および対象座標系の加法，スカラー倍算が必須となるが，これらはソフトウェア実装では高オーバヘッドであり，ハードウェアの利用が望まれる．しかし複雑なアルゴリズムを全てハードウェアで実装することはハードウェアサイズの点で困難であり，かつ柔軟性に欠けるため，ソフト側からの基本ハードモジュールの利用が現実的である．本稿では FPGA SoC を対象としたハードウェア基本モジュールの実装について述べる．

Implementation of Multi-Precision Arithmetic Unit on FPGA SoC

Kiyofumi Tanaka[1,a)]

Abstract: RSA, one of major public key cryptography systems, is difficult to apply to embedded systems since it requires long key lengths for enough security levels. On the other hand, elliptic curve cryptosystems can be applied to embedded systems with small amounts of memory where relatively short keys provide enough security levels. In processing of elliptic curve cryptography, basic multiple-precision arithmetics and addition/scalar multiplication on the target coordinate system are indispensable. These calculations cause large overheads and hardware implementations are strong candidates. However, it is difficult to implement all the complicated algorithm in hardware due to large amounts of hardware and lack of flexibility, which makes it realistic to use basic hardware modules from software. In this paper, we describe the implementation of basic hardware modules for FPGA SoC.

1. はじめに

インターネットの普及とともに，情報交換におけるセキュリティ対策として公開鍵暗号方式の一つである RSA 暗号が長く利用されてきたが，今日の計算機器の高性能化にともない高いセキュリティレベルが要求されつつあり，セキュリティ強化のために長い鍵長が必要となる RSA 暗号の処理には多くの計算資源が不可欠となっている [1]．これに対し，同じく公開鍵暗号方式である楕円曲線暗号は，RSA 暗号よりも短い鍵長で同等の安全性を確保可能（例えば 4096 ビットの鍵を使用する RSA 暗号に対し，楕円曲線暗号では 300 ビット程度の鍵で同等の安全性となる [3]）であり，メモリ資源が限られた組込み機器における暗号システムとして主流になりつつある [2]．しかしながら，楕円曲線暗号を使用することによりアルゴリズムの複雑さが緩和されるわけではなく，計算資源／速度が限られた組込み機器におけるソフトウェアによる計算では相当の処理時間を要する．ソフトウェア実行のオーバヘッドを削減する方法として処理のハードウェア化が有力な選択肢となるが，暗号処理アルゴリズムの全てをハードウェア化することは使用ハードウエア量および処理の複雑さによる実行速度低下の観点から現実的ではない．

近年，プロセッサをハードマクロとして搭載する FPGA デバイスが普及してきており，ソフトウェアと，プログラマブルロジックで実現されたハードウエアモジュールの両方の実行がデバイス内で可能となっている．本稿ではこのような種類のデバイスを FPGA SoC と呼ぶ．FPGA SoC は従来のプログラマブルロジックのみの FPGA デバイスと比較し，その密結合性によりプロセッサとハードウェア

[1] 北陸先端科学技術大学院大学
JAIST, Asahidai 1-1, Nomi, Ishikawa 923–1292, Japan
[a)] kiyofumi@jaist.ac.jp

Vol.2019-ARC-240 No.42
Vol.2019-SLDM-191 No.42
Vol.2019-EMB-53 No.42
2019/2/28

間のデータ転送が高速であり，効率の良いハード・ソフト協調実行が可能となる．本研究では楕円曲線暗号処理で基本かつ主要となる演算をプログラマブルロジック部を利用してハードウェアモジュールとして提供し，ソフトウェアと協調動作することにより処理全体の高速化と柔軟性を提供する方式を採用する．

2. 楕円曲線暗号と基本演算

2.1 楕円曲線上の加法公式

楕円曲線暗号は，楕円曲線離散対数問題の困難性に基づく公開鍵暗号であり，1980 年代半ばに提案された [4], [5]．本節では，本稿で対象とする楕円曲線と加算公式および倍算公式について述べる．

素体 $\mathbb{F}_q(q > 3)$ 上の以下の Short Weierstrass 標準形で与えられる曲線 $E/\mathbb{F}_q$ を対象とする．

$$y^2 = x^3 + ax + b \ \ (a, b \in \mathbb{F}_q, 4a^3 + 27b^2 \neq 0)$$

この曲線上の点に零元 $\mathcal{O}$ を加えた集合において幾何学的に加算が定義できる．Affine 座標系において，点 $P = (x_p, y_p)$，$Q = (x_q, y_q)$ $(P \neq Q)$ に対する加算公式（Affine Add: $R = (x_r, y_r) = P + Q$）は以下の通りである．

Affine Add:

$$x_r = (\frac{y_q - y_p}{x_q - x_p})^2 - x_p - x_q$$
$$y_r = (\frac{y_q - y_p}{x_q - x_p})(x_p - x_r) - y_p$$

同様に，2 倍算 (Affine Double: $R = 2P$) は以下のように定義される．

Affine Double:

$$x_r = (\frac{3x_p^2 + a}{2y_p})^2 - 2x_p$$
$$y_r = (\frac{3x_p^2 + a}{2y_p})(x_p - x_r) - y_p$$

実際の暗号計算では，上記の式における各要素（x_P，y_Q など）は多倍長データであり，かつ結果の R は $\mathbb{F}_q$ の元である必要があるため，Affine Add，Affine Double を計算するためには多倍長演算（加算，乗算，除算のための逆元計算，および剰余計算）が必要となる．

2.2 スカラー倍算

2.1 節の加算，2 倍算を繰り返し適用することにより，点 P のスカラー倍が計算できる．最も単純な方法は，スカラー値を n とした場合，n 回の加算を実行することにより nP を得るものである．この場合，計算量は n のオーダーとなる．これに対し，より効率の良い方法の一つがバイナリ法 [6] である．バイナリ法では変数 R と Q（それぞれ $\mathcal{O}$ と P で初期化したもの）を用意し，2 進数のスカラー値を最下位桁から最上位桁（またはその逆）に向かって走査し，桁の値が 0 のときは Q を 2 倍算で更新，1 のときは R と Q の加算値で R を更新した後に Q を 2 倍算で更新する．（最上位桁から最下位桁への走査の場合は 2 倍算更新後に加算更新を行う．）最終的に得られた R がスカラー倍算の結果（nP）となる．これにより，スカラー倍算の計算量をスカラー値のビット数のオーダー，すなわち $log_2 n$ に抑えることができる．

バイナリ法では n の各桁の値によって計算量が異なる（2 倍算のみ，あるいは加算と 2 倍算）．このことは，消費電力を計測してその波形から入力データを推測する Simple Power Analysis (SPA) 攻撃 [7] に対して脆弱であることを意味する．そこでバイナリ法を改良し，各桁で計算量を一定とする Joye's m-ary Ladder が提案された [8]．これはスカラー値の m 進数表現を各桁に 0 が含まれない形に変形することにより，必ず加算と m 倍算を行う方法である．しかしながら，加算のオペランドに零元（$\mathcal{O}$）が含まれる場合は例外的な計算が必要であり，SPA 攻撃への脆弱性は完全に除去できていない．

木藤らはこの零元の問題を解決する Modified Joye's 2-ary 2-bit Right-to-Left（および Left-to-Right）を提案した [9]．これは前述の Joye の方法における零元での初期化を回避し，例外処理が発生しない方法である．さらに 2 倍算の繰り返しを 4 倍算に置き換えることで効率化を図っている．この 4 倍算には Le らによって提案された 4 倍算公式である Affine Double-Quadruple(Affine DQ) [10] を利用することが可能である．（Affine DQ では一回の逆元計算の結果を利用して $2P$ と $4P$ が得られる．）

楕円曲線暗号の暗号化と復号化においてスカラー倍演算の計算コストが支配的となる．本研究では Modified Joye's 2-ary 2-bit Right-to-Left によるスカラー倍算をハードウェア化により高速化することを目標とする．

3. 対象アルゴリズム

Algorithm 1 に本稿で対象とするスカラー倍算アルゴリズムである Modified Joye's 2-ary 2-bit Right-to-Left [9] を示す．このアルゴリズムは楕円曲線上の点 P とスカラー値 k を入力とし，それらの積 $Q = kP$ を出力するものである．

アルゴリズムにおいて，3〜7 行目のループに注目すると，各イテレーションで 2 回の加算（4 行目と 5 行目），1 回の 2 倍算，4 倍算（6 行目）を実行することになる．加算には 2.1 節の Affine 座標系の加算（Affine Add）を使用し，2 倍算および 4 倍算は 2.2 節で述べた Affine DQ を使用する．Affine DQ は逆元の計算を 1 回のみ含むため，逆元を求める計算量が比較的大きい場合に，2 回の 2 倍算を行う方法と比較し，小さい計算量で 2 倍算と 4 倍算の結果を同時に得ることになる [10]．また，Joye's m-ary Ladder が k の各桁に対応したイテレーションを実行することに対

Vol.2019-ARC-240 No.42
Vol.2019-SLDM-191 No.42
Vol.2019-EMB-53 No.42
2019/2/28

Algorithm 1 Modified Joye's 2-ary 2-bit Right-to-Left[9]

Input: $P \in E/\mathbb{F}_q$ and $k = \sum_{i=0}^{n-1} k_i 2^i$ (n is even)
Output: $Q = kP$
1: $A[0] \leftarrow -P; A[1] \leftarrow P$
2: $\{R[0], R[1]\} \leftarrow \{2P, 4P\}; R[2] \leftarrow 2P$
3: **for** i=1 **to** n-5 **step** 2 **do**
4: $\quad A[k_i] \leftarrow A[k_i] + R[0]$
5: $\quad A[k_{i+1}] \leftarrow A[k_{k+1}] + R[1]$
6: $\quad \{R[1], R[0]\} \leftarrow \{4R[1], 2R[1]\}$
7: **end for**
8: $A[k_{n-3}] \leftarrow A[k_{n-3}] + R[0]; A[k_{n-2}] \leftarrow A[k_{n-2}] + R[1]$
9: $A[0] \leftarrow A[0] + 2A[1]; R[1] \leftarrow P$
10: $A[0] \leftarrow A[0] + R[k_0 + 1]$
11: **return** $A[0]$

し，本ループは 2 桁毎に走査することにより計算量を削減している．

表 1 に対象ループのイテレーションを実行するための総演算数を示す．Affine Add および Affine DQ は基本多倍長演算（加算，乗算，剰余算，逆元計算）から構成される．多倍長加算に関しては，プロセッサの演算データサイズを超える carry-look-ahead 方式や carry-skip 方式 [12] などの適用が候補となるが，楕円曲線暗号においては鍵長が大きくないことから，本研究では多倍長データのサイズは 1024 ビット以下を想定しており，プロセッサの演算データサイズに基づく単純な逐次加算を使用する．同様に，多倍長乗算に関しては Karatsuba 法 [13] 等の高速化方式があるが，多倍長データサイズが 1024 ビット以下の場合は，増加する基本加算が相対的にオーバヘッドとなることと，ハードウェア化の際の部分並列化の適用可能性を考慮し，本研究では単純な筆算方式の乗算を使用する．（全ての部分積は並列に生成可能である．）

剰余算は加（減）算後に行う場合は 1 回の加算に相当する．乗算後に行う場合は乗算と剰余算の組をモンゴメリリダクションを使用するモンゴメリ乗算 [11] に置き換える．逆元計算はハードウェア実装が未完であるため，本稿では逆元計算の結果からなるテーブルをあらかじめ用意し，それを参照する．

表 1 イテレーション内の演算数．

演算	回数	基本多倍長演算	回数
Affine 加算	2	加算	6
(Affine Add)		乗算	3
		剰余算	4
		逆元	1
Affine 2,4 倍算	1	加算	14
(Affine DQ)		乗算	25
		剰余算	16
		逆元	1

4. 実装

4.1 構成

前節の方針に基づき，以下の演算ハードウェアモジュールをハードウェア記述言語（Verilog HDL および VHDL）で設計および実装した．

- 多倍長加算器（ADD）
- 多倍長乗算器（MUL）
- 多倍長モンゴメリリダクション演算器（REDUCTION）
- 多倍長モンゴメリ乗算器（MONTGOMERY_MUL）
- Affine 加算器（AFFINE_ADD）
- Affine 2 倍&4 倍算器（AFFINE_DQ）

各モジュールのオペランドは 704（=64×11）ビットのサイズで統一した．対象デバイスは Xilinx 社の Zynq UltraScale+ MPSoC ZU7EV[14]，評価ボードは同社の ZCU104 を使用した．

図 1 にハードウェアモジュールを含むシステムの構成図を示す．Processing System（PS）部の Cortex-A53 コアを 1 つ使用し，楕円曲線暗号のソフトウェアを実行する．PS 部の動作周波数は 500MHz に設定した．各ハードウェアモジュールは Programmable Logic（PL）部に実装した．PL 部の動作周波数は 100MHz に設定した．各ハードウェアモジュールはコントロールレジスタおよびステータスレジスタを持ち，PS からのコントロール値の書き込みでハードウェア実行を起動し，ステータスレジスタの値を PS から監視することで終了を判断する．

4.2 データ転送と PL 内オペランド伝搬

演算パラメータ値を PL 部に転送する方法には二つある．一つは PS がソフトウェア実行により AXI 接続を介して PL 部のバッファに 128 ビット単位で書き込む方法

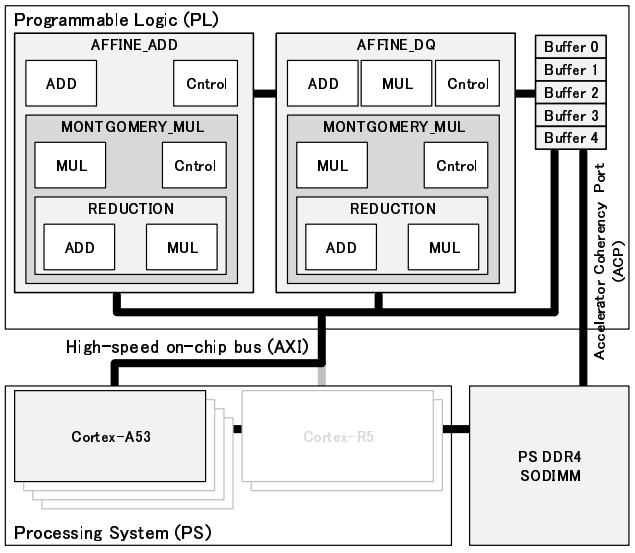

図 1 ハードウェア構成．

Vol.2019-ARC-240 No.42
Vol.2019-SLDM-191 No.42
Vol.2019-EMB-53 No.42
2019/2/28

であり，他方は PL 部の DMA を起動し，PS 部のメモリ領域をキャッシュコヒーレントで読み出し，バッファにセットする方法である．後者の DMA 機構は Accelerator Coherency Port（ACP）と呼ばれるもので，転送サイズが選択できるが，対象デバイスで 704 ビットのサイズの転送を行う場合，512 ビットの転送サイズを選択して 2 回分の転送を行う必要がある．したがって使用されないダミーのデータを転送することになり効率が悪い．このことから本実装では前者の PS による 128 ビット単位での書き込みを採用する．（実際に 512 ビット分のデータ転送を前者と後者で行った場合，所用時間にほとんど差が無いことが確認された．）同様に演算結果について，PS が PL 内のバッファのデータを 128 ビット単位で読み出す方法を採る．（ACP を使用した PS メモリ領域への書き込みにより，結果受け渡しの時間を削減できる可能性があるが，これは未実装であり，今後の課題の一つである．）

ソフトウェア／ハードウェア協調実行によるスカラー倍算は Algorithm 1 および表 1 が示す通り，2 回の AFFINE_ADD と 1 回の AFFINE_DQ を起動することになる．AFFINE_ADD 同士および AFFINE_ADD と AFFINE_DQ 間で結果の受け渡しが必要となるが，このためにアドレッサブルなバッファを用意する．バッファは 704×2 ビットのデータ毎に，計 5 個実装した．これらは Algorithm 1 のループイテレーション内の $A[0]$，$A[1]$，$R[0]$，$R[1]$ に対応する 4 個（それぞれ x 座標，y 座標の 2 つの 704 ビットデータを格納）と k のための 1 個である．ソフトウェアがコントロールレジスタに入出力オペランドのためのバッファ番号を書き込むことにより，ハードウェアモジュールは指定されたバッファを使用して計算を行う．

4.3 各モジュールの実行サイクル数

表 2 に各演算モジュールの実行サイクル数を示す．表 1 が示す通り，Affine_Add と比較し Affine_DQ は多数の乗算を含むため，実行サイクル数が大きくなっている．なお，前節で述べた通り，逆元計算を行う INVERT は未実装であるため，テーブル参照の 2 サイクルのみ示している．このことから，AFFINE_ADD と AFFINE_DQ の実行サイクル数は逆元計算の実際の実行時間を含んでいない値である．

表 2 各演算モジュールの実行サイクル数.

演算	実行サイクル数
ADD	13
MUL	47
REDUCTION	120
MONTGOMERY_MUL	334
AFFINE ADD	1,108
AFFINE DQ	7,138
(INVERT)	(2)

5. 評価

本節では，実装した PL 部のハードウェアサイズの評価，およびソフトウェアのみによる実行とハードウェアモジュールを利用する実行の実行時間を評価する．

5.1 ハードウェアサイズ

設計したハードウェアモジュールに対して Xilinx 社の Vivado v2019.2 を使用して論理合成および配置・配線を行った．FPGA リソースの使用数を**表 3** に示す．本実装はモジュールの共有化等の最適化（例えば AFFINE_ADD と AFFINE_DQ 間の下位モジュールの共有化）を行っていないため，特にルックアップテーブル（CLB LUTs）の使用量が大きくなっている．同様に，下位モジュールの実行結果を，モジュール毎に用意したレジスタに格納しているため，レジスタ（フリップフロップ）数（CLB Registers）の使用率もある程度高くなっている．今後は下位モジュールおよびレジスタの共有化を行い，ハードウェアサイズを削減する予定である．

5.2 実行時間

スカラー倍算の計算時間を評価ボード（ZCU104）上で計測した．PS 部のプロセッサでは Linux 4.14.0 上でソフトウェアが動作する．ソフトウェアは C 言語で記述されたものであり，コンパイルは gcc 6.3.0（-O4 オプション）を使用した．

ソフトウェア／ハードウェア協調実行方式において，ソフトウェアから呼び出すハードウェアモジュールの粒度として，下位モジュールの ADD，MUL を直接呼び出す（起動する）方法では呼出し毎の PS によるパラメータセットのオーバヘッドにより実行時間が大きくなった．したがって，ここではより粒度の大きい（上位階層の）MONTGOMERY_MUL を直接呼び出す方法，および AFFINE_ADD と AFFINE_DQ を呼び出す方法での実行時間を示す．スカラー倍算を 100 回計算した場合の，一回当たりの平均の実行時間を**表 4** に示す．

MONTOGOMERY_MUL を直接呼び出す方法では，多数の呼び出しに対して毎回パラメータをセットするオーバヘッドがなお大きく，ソフトのみの実行よりも低速になった．一方，AFFINE_ADD と AFFINE_DQ の粒度で呼び出す方法は高速化が確認できる．結果としては，ソフトウェ

表 3 FPGA リソース使用数.

Type	Used	Available	Util.(%)
CLB LUTs	77,997	230,400	33.85
CLB Registers	50,973	460,800	11.06
Block RAM	1	312	0.32
DSPs	180	1,728	10.42

表 4　スカラー倍算の実行時間.

実行方式	実行時間（ms）
ソフトウェアのみ	20.2
ソフト／ハード協調（MONTGOMERY_MUL）	27.2
ソフト／ハード協調（AFFINE_ADD/DQ）	12.8

アのみ場合は 500MHz のプロセッサ上で実行されているが，100MHz で動作するハードウェアモジュールを利用する実行が約 1.58 倍高速である．今回の実装は初期段階の設計に基づくものであり，ハードウェアモジュールの実行効率を改善する余地は多数あるため，設計を改良することにより更なる高速化が期待できる．

6. おわりに

本稿では公開鍵方式の一つである楕円曲線暗号において主要な演算となるスカラー倍算を行うための各種ハードウェアモジュールの設計と FPGA SoC 上での実装，およびハードウェア量と実行時間に関する評価について述べた．本実装は初期の段階にあり，未実装の逆元計算モジュールの設計を完了し，各基本多倍長演算器を効率化することが今後の第一の課題である．更に，Affine 加算および Affine 2 倍算&4 倍算の計算に内在する並列性を抽出して実装を改良し，高速化を図る予定である．

楕円曲線上のスカラー倍算の FPGA を利用したハードウェア化については，Koblitz 曲線上のもの [15] を始め多数研究されている．効率化を含めた実装が完了次第，既存研究との比較評価を行うことを予定する．

謝辞　本研究の一部は JSPS 科研費 19K11873 の助成を受けて行われた．楕円曲線暗号における高速スカラー倍算のソフトウェアを提供していただいた大阪大学の宮地充子先生に感謝いたします．

参考文献

[1] 独立行政法人情報処理推進機構: SSL/TLS 暗号設定ガイドライン Ver. 2.0 (2018).
[2] 清藤 武暢, 四方 順司: 公開鍵暗号を巡る新しい動き : RSA から楕円曲線暗号へ，金融研究 32(3), pp.17–49 (2013).
[3] de I. Blake，G. Seroussi，N. Smart: Elliptic Curves in Cryptography，Cambridge University Press(1999).
[4] V.S.Miller: Use of Elliptic Curves in Cryptography, Proc. of Crypto 85, LNCS 219, Springer, pp.417–426 (1986).
[5] N.Koblitz: Elliptic Curve Cryptosystems, Mathematics of Computation, Vol.48, pp.203–209 (1987).
[6] D.Hankerson, et all.: Guide to Elliptic Curve Cryptography, Springer (2003).
[7] P.C.Kocher: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems, Proc. of CRYPTO '96, pp.104–113 (1996).
[8] M.Joye: Highly Regular m-Ary Powering Ladders, Selected Areas in Cryptography – SAC 2009, Vol.5867 of LNCS, pp.350–363, Springer (2009).
[9] 木藤 圭亮，宮地 充子: サイドチャネル攻撃耐性を持つスカラー倍算アルゴリズムの改良と実装，電子情報通信学会技術研究報告，115(488), pp.147–152 (2016).
[10] D-P. Le, B.P.Nguyen, Fast point quadrupling on elliptic curves, Proc. of SoICT, pp.218–222 (2012).
[11] P.L.Montgomery: Modular multiplication without trial division, Mathematics of Computation, 44(170), pp.519–521 (1985).
[12] M.Lehman, N.Burla: Skip Techniques for High-Speed Carry-Propagation in Binary Arithmetic Units, IRE Transactions on Electronic Computers, Vol.EC-10, No.4, pp.691–698 (1961).
[13] A.A.Karatsuba, Y.Ofman: Multiplication of Multidigit Numbers on Automata, Vol.7, No.7, pp.595–596 (1962).
[14] Xilinx, Inc.: Zynq UltraScale+ MPSoC Data Sheet: Overview, https://www.xilinx.com/support/documentation/data_sheets/ds891-zynq-ultrascale-plus-overview.pdf
[15] S.S.Roy, J.Fan, I.Verbauwhede: Accelerating Scalar Conversion for Koblitz Curve Cryptoprocessors on Hardware Platforms, IEEE Trans. on VLSI Systems, Vol.23, No.5, pp.810–818 (2015).