Vol.2020-EMB-54 No.2
2020/6/26

サイドチャネル攻撃耐性を持つ楕円曲線スカラー倍算アルゴリズムのハードウェア実装

田中 清史[1,a)]　宮地 充子[2,1,b)]

概要：楕円曲線暗号の主演算であるスカラー倍算は，実行時間がスカラー値に依存して変動することによりサイドチャネル攻撃に対して脆弱となる．本研究では実行時間が一定となるスカラー倍算アルゴリズムを FPGA に実装する．ハードウェア設計において，アルゴリズムにおける楕円曲線上の加算／倍算同士，および多倍長のフィールド演算同士のデータフローを解析することにより，並列性を可能な限り抽出して高速化を図る．評価において，ソフトウェアのみの実行および既存のハードウェア実装と比較することによりその性能向上を示す．

Hardware Implementation of Elliptic Curve Scaler Multiplication Algorithms with Side-Channel Protection

Kiyofumi Tanaka[1,a)]　Atsuko Miyaji[2,1,b)]

Abstract: Scalar multiplication, which is a main operation in elliptic curve cryptosystems, is vulnerable to side-channel attacks if the execution times vary depending on the scalar value. I this research, we try to design a scalar-multiplication algorithm the execution time of which is fixed and implement it on an FPGA. In the hardware design, parallelism between elliptic-curve-point addition and doubling and between multi-precision field operations is fully extracted by analyzing data flows in the algorithm, which leads to high-performance processing. In the evaluation, the performance is compared with software-only processing and another hardware implementation.

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 2048 ビットの鍵を使用する RSA 暗号に対し，楕円曲線暗号では 256 ビット程度の鍵で同等の安全性となる [2]）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 [3]．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，暗号処理ソフトウェアの全ての機能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

[1] 北陸先端科学技術大学院大学
JAIST, Asahidai 1-1, Nomi, Ishikawa 923–1292, Japan
[2] 大阪大学 大学院工学研究科
Graduate School of Engineering, Osaka Univ. Yamadaoka 2-1, Suita, Osaka, 565–0871, Japan
[a)] kiyofumi@jaist.ac.jp
[b)] miyaji@comm.eng.osaka-u.ac.jp

両方の実行がデバイス内で可能となっている．このような System-on-Chip（SoC）型の FPGA デバイス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．

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

この曲線上の点に無限遠点 $\mathcal{O}$ を加えた集合において幾何学的に加算が定義される．Affine 座標系において，点 $P = (x_p, y_p)$ と点 $Q = (x_q, y_q)$ $(P \neq Q)$ 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:

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

なお，楕円曲線暗号には affine 座標系以外の加算が存在するが，本稿ではメモリ量の観点で最も効率的な affine 座標系を採用する．

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 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としての加減算，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．（除算は計算量が大きいため，逆元を求めた後に乗算を行うことで代替する．）

2.2 スカラー倍算

楕円曲線暗号の暗号化と復号において楕円曲線上の点に対するスカラー倍演算の計算コストが支配的となる．2.1 節の加算と 2 倍算を繰り返し適用することにより，点 P に対する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 $\Theta(n)$ 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 進数で表現されたスカラー値を最下位桁から最上位桁（またはその逆）の方向で走査し，桁の値が 0 のときは Q を 2 倍算で更新し，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 Q を 2 倍算で更新した後に R の加算更新を行う．）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 $O(log_2 n)$ に抑えることができる．

上記のバイナリ法では，厳密には n の各桁の値によって計算量が異なる．すなわり，2 倍算のみの場合と加算と 2 倍算を行う場合がある．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，楕円曲線の加算として上記の affine 座標系を用いる場合には，途中計算の中で例外的な計算となる無限遠点（$\mathcal{O}$) との加算が必要となり，SPA 攻撃への脆弱性は完全には除去できない．

木藤らはこの無限遠点の問題を解決する 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 の方法における無限遠点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 4 倍算を導入してループアンローリングを行うことにより効率化を図っている [*1]．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) [11] 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 $4P$ が得られるため，アンローリングされたイテレーション内の計算量が削減される．）

本研究では文献 [9] で提案された Modified Joye's 2-ary 2-bit Right-to-Left を更に修正した New two-bit 2-ary Rright-to-Left Powering Ladder アルゴリズム [10] によるスカラー倍算を対象とし，データフロー解析によりアルゴ

[*1] 文献 [9] では更に，2-ary 3bit Right-to-Left，4-ary Left-to-Right，8-ary Left-to-Right が提案されている．

リズム内における並列性を最大限に抽出し，かつ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 New tow-bit 2-ary Right-to-Left Powering Ladder [10] 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，ステップ 5〜10 のループに注目すると，各イテレーションで 2 回の加算（ステップ 6 と 7），および 1 回の 2 倍算，4 倍算（ステップ 8）を実行する．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が可能である [11]．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．

Affine Add および Affine DQ は基本多倍長演算群（加減算，乗算，剰余算，逆元計算）から構成される．多倍長加算のハードウェア実装に関しては，一般的に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およびフィールドサイズが数百ビットに抑えられることから，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，単純な加算（ハードウェア記述言語における加算演算子）を使用する．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ため，本研究では単純な筆算方式の乗算を使用する．筆算方式の場合，ハードウェア化の際に部分並列化の適用（全ての部分積は並列に生成可能）が可能である．

剰余算は加（減）算後に行う場合は 1 回の減（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 [14] に置き換える．（前述の多倍長乗算器をモンゴメリ乗算内の基本演算器として使用する．）逆元計算については，ハードウェアの肥大化を回避するために除算を必要としないバイナリ拡張 GCD アルゴリズム [15] を使用する．

以上の基本多倍長演算器を使用し，アルゴリズム内の各ステップに対応してハードウェアモジュールを設計・実装する．アルゴリズムのステップ 4，6，7 および 11 が Affine Add モジュール，ステップ 3 および 8 が Affine DQ モジュール，ステップ 12 が Affine Double Change モジュールと Extended Affine モジュールの組み合わせで実行される．**表 1** に各ハードウェアモジュールを実行するための基本演算数を示す．

本研究では，多種のアルゴリズムに対応するためにプログラマビリティを確保する方針を採る．すなわち，基本演算器群および演算モジュール群をハードウェアで提供し，それらの起動の制御はソフトウェアで行うソフト／ハード協調実行方式を実現する．1 節で述べた通り，今日の SoC 型の FPGA デバイスはその密結合性によりソフト・ハード間のデータ転送や同期が高速である．プログラマビリティの提供と SoC 型 FPGA の利用により，前節で述べた他のアルゴリズムに対して微小の変更で高速なハードウェアア

Algorithm 1 New two-bit 2-ary Rright-to-Left Powering Ladder (Algorithm 8 in [10])

Input: $P \in E/\mathbb{F}_q$
$k = \sum_{i=0}^{l-1} k_i 2^i,\ k \in [0, N]$
Output: $Q = kP$

Initialization
1: $R[0] \leftarrow -P$
2: $R[1] \leftarrow P$
3: $\{A, A1\} \leftarrow \{2P, 4P\}$
4: $R[k_0] \leftarrow R[k_0] + A$

Main Loop
5: **for** $i = 1$ **to** $l - 1$ **do**
6: $R[k_i] \leftarrow R[k_i] + A$
7: $R[k_{i+1}] \leftarrow R[k_{i+1}] + A1$
8: $\{A, A1\} \leftarrow \{2A1, 4A1\}$
9: $i \leftarrow i + 2$
10: **end for**

Final Correction
11: $R[k_0] \leftarrow R[k_0] - P$
12: $A \leftarrow -A + R[0] + 2R[1]$
13: **return** A

表 1 各ハードウェアモジュール内の基本多倍長演算数.

楕円曲線上の演算	基本多倍長演算	回数
Affine Add	加算	2
	減算	6
	モンゴメリ乗算	3
	逆元	1
Affine DQ	加算	9
	減算	10
	モンゴメリ乗算	20
	逆元	1
Affine Double Change	加算	3
	減算	3
	モンゴメリ乗算	4
	逆元	1
Extended Affine	加算	1
	減算	5
	モンゴメリ乗算	7
	逆元	1

クセラレーションが可能となる．

4. 設計と実装

4.1 演算器の設計

前節の方針に基づき，以下の基本多倍長演算器と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）
- 多倍長減算器（SUB）
- 多倍長乗算器（MUL）
- 多倍長モンゴメリ乗算器（MONT_MUL）
- 逆元演算器（INVERSE）
- Affine Add モジュール（AFFINE_ADD）
- Affine Double-Quadruple モジュール（AFFINE_DQ）
- Affine Double Change モジュール（AFFINE_DBLCH）
- Extended Affine モジュール（EXTENDED_AFFINE）

4.1.1 多倍長加減算器

対象暗号システムでは素体 $\mathbb{F}_q$ における素数 q として 256 ビットのサイズを使用するため，データサイズは 256 ビットを基本とするが，内部的な一時データとして 260 ビットデータが出現するため，260 ビット加算器を設計した．また，モンゴメリ乗算において乗算を扱うことから，一時データとしての乗算結果は最大 520 ビットとなり，これをオペランドとする 520 ビット加算器を設計した．同じくモンゴメリ乗算において 264 ビットの一時データを扱う減算が行われるため，264 ビット減算器を設計した．これらの加減算器は 1 クロックサイクルで結果を生成する．

4.1.2 多倍長乗算器

288 ビットオペランド間の乗算を行う乗算器を設計した．各 288 ビットオペランドを 2 つの 144 ビットデータに分割し，144 ビット× 144 ビットの部分積を並列に計算することにより，部分積の総和演算を含めて 6 クロックサイクルで乗算結果を生成する．部分積生成はハードウェア記述言語の乗算演算子によるが，論理合成により対象デバイスの埋込み乗算器（DSP48E2：27 ビット× 18 ビット乗算器）が使用される．

4.1.3 多倍長モンゴメリ乗算器

モンゴメリ乗算は上記の 520 ビット加算を 2 回，264 ビット減算を 2 回，288 ビット乗算を 6 回を実行することにより結果を得る．本演算器は内部の演算間で並列性が無く，各演算を逐次実行するため，計算時間は 40 クロックサイクルとなる．

4.1.4 逆元演算器

Affine Add，Affine DQ 内の一時計算値（260 ビット）を入力とし，256 ビットの逆元値を生成する演算器を設計した．バイナリ拡張 GCD アルゴリズムに従ってループイテレーション内で減算を行うが，オペランドが 260 ビットであることと，モンゴメリ乗算器内の 520 ビット加算が 1 クロックサイクル実行の設計であることから，本演算器では遅延のバランスを考慮して 2 回のイテレーションをアンローリングし，2 回の減算を逐次的に 1 クロックサイクルで実行する設計とした．これにより，アンローリングを行わない実行に対して実行時間は半分となる．すなわち，オリジナルのバイナリ拡張 GCD では実際の実行時間はデータに依存し，256 ビットデータを対象とする場合は平均 570 サイクル程度であるが，一方本実装では約 285 サイクルに短縮することになる．

4.1.5 ハードウェアモジュール

Affine Add，Affine DQ，Affine Double Change，および Extended Affine ハードウェアモジュールは前述の加減算器，モンゴメリ乗算器，逆元演算器を制御シーケンスにしたがって使用することにより実現される．各演算の結果は一時データを保持するための 260 ビットレジスタに格納される．後述するモジュール内並列実行を実現するために Affine Add が 3 個，Affine DQ が 5 個，Affine Double Change が 3 個，および Extended Affine が 5 個の一時レジスタを使用する．

モジュール内並列計算について Affine DQ の場合を述べる．（その他のハードウェアモジュールも同様の方法が適用される．）**図 1** に Affine DQ 内のデータフローを示す．データフローの解析と各演算の実行時間情報から，加減算器を 1 個，モンゴメリ乗算器を 2 個，逆元演算器を 1 個使用することにより最大限の並列性が抽出できることがわかる．これを基に演算スケジューリングとレジスタ割当てを行った結果を**図 2** に示す．この結果から，必要な一時レジスタは 5 個となった．

Algorithm 1 において，ループ内で 2 回の Affine Add（ステップ 6，7），1 回の Affine DQ（ステップ 8）を実行する．ステップ 6 と 7 の Affine Add は変数 R[0]，R[1] に関してデータ依存の可能性があるため逐次的に実行する．一方，ステップ 8 の Affine DQ は，先行する Affine Add との間で read-after-write 依存が無いため，ステップ 6，7 と並行して実行可能である．ただし，変数 A，A1 に関して先行計算に対する write-after-read の関係があるため，Affine DQ の結果を変数 A および A1 に保存するタイミングは，先行計算の終了後とする必要がある．このモジュール間並列実行および同期を実現するために，Affine Add，Affine DQ に対する以下の API を C 言語で実装した．

- Start_Affine_Add(): Affine Add 演算器を起動する．
- End_Affine_Add(): Affine_Add 演算の終了をポーリングで待つ．
- Start_Affine_DQ(): Affine DQ 演算器を起動する．
- End_Affine_DQ(): Affine_DQ 演算の終了をポーリングで待つ．
- Sync_Affine_DQ(): Affine DQ の演算結果を指定するバッファ（後述）に格納する．

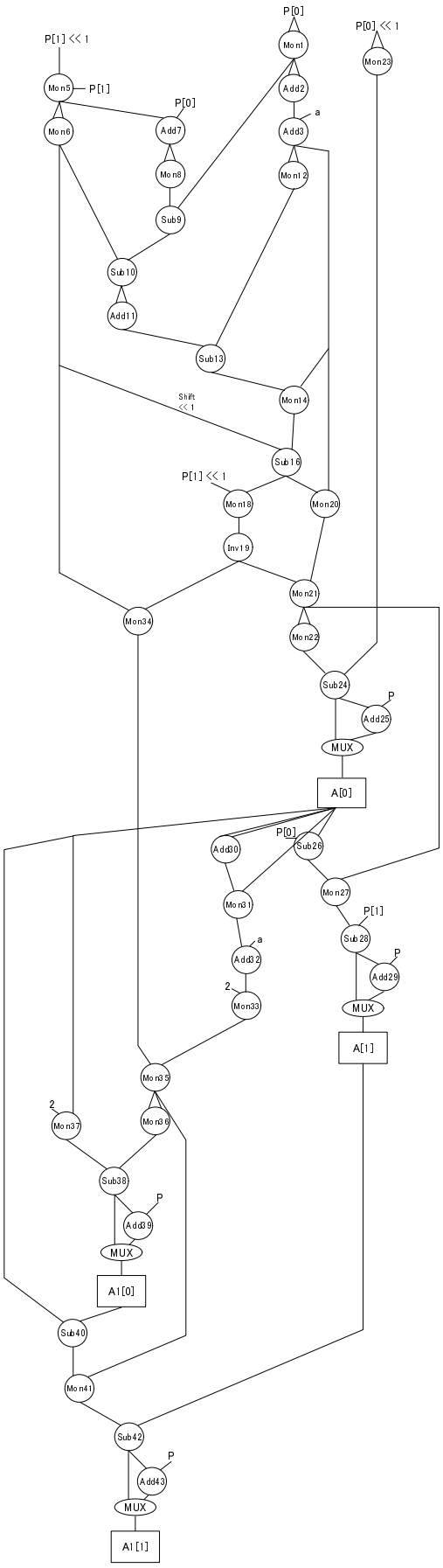

図 1　Affine DQ 内のデータフロー.

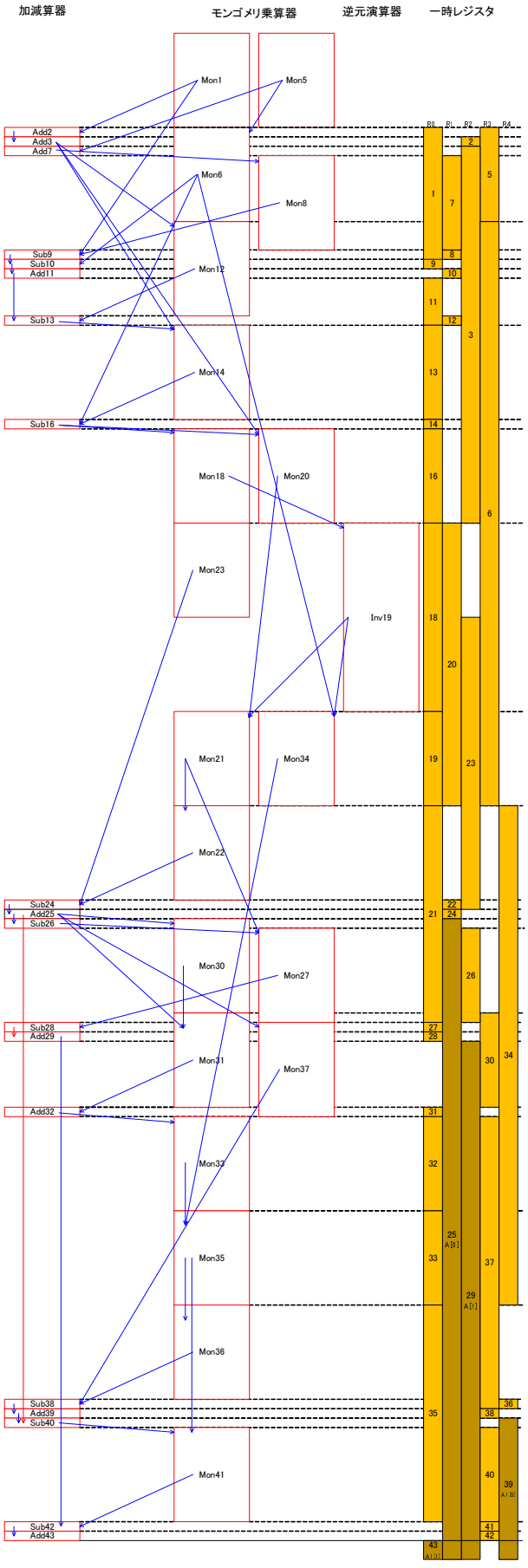

図 2　Affine DQ 内の演算スケジューリングとレジスタ割当て.

ループ実行に対応する以下の C 言語プログラムコードのようにこれらの API を使用することにより，モジュール間の並列実行が可能となる．

```
for (  i = 1; i < l; i += 2 ) {
   Start_Affine_DQ(A1,A,A1); /* A=2A1; A1=4A1;*/

   m = (k >> i) & 0x1;
   Start_Affine_Add(A,R[m]); /* R[m]=A+R[m];*/
   End_Affine_Add();

   m = (k >> i+1) & 0x1;
   Start_Affine_Add(A1,R[m]); /* R[m]=A1+R[m];*/
   End_Affine_Add();

   End_Affine_DQ()
   Sync_Affine_DQ();
}
```

モジュール間並列実行により，高速化に加えて，SPA 攻撃への耐性が強化される利点がある．アルゴリズムの解析により，モジュール間並列実行は，Affine Add と Affine DQ，および Affine Add と Affine Double Change の組み合わせが可能である．この組み合わせから，前述の一時レジスタに関して，Affine Add が 3 個のレジスタを占有し，Affine DQ，Affine Double Change，および Extended Affine が 3 個のレジスタを共有し，Affine DQ，Affine と Extended Affine が 2 個のレジスタを共有する．以上から，全体では 8 個の一時レジスタが実装される．

同じくモジュール間並列実行を考慮し，モジュール間で基本演算器の共有が可能となる．各モジュールは 1 個の加減算器，1 個のモンゴメリ乗算器（Affine DQ のみ 2 個），1 個の逆元演算器を使用するが，これらの組は Affine Add と Extended Affine，Affine DQ と Affine Double Change の間で共有可能である（**図 3**）．この共有方針によりハードウェア量の削減を図る．

4.2 システム構成

Xilinx 社の Zynq UltraScale+ MPSoC ZU7EV[17] を対象デバイスとし，評価ボードは同社の ZCU104 を使用した．図 3 に各ハードウェアモジュール（AFFINE_ADD, AFFINE_DQ, AFFINE_DBLCH, EXTENDED_AFFINE）および基本演算器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と演算器は Programmable Logic（PL）部に実装した．PL 部の動作周波数は 214MHz 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 上のソフトウェアから前節で述べた START_AFFINE_ADD() などによりコントロール値の書き込みを行うことでハードウェア実行を起動し，END_AFFINE_ADD() などで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．コントロールレジスタおよびステータスレジスタはメモリマップ I/O として PS 上のソフトウェアからアクセス可能としている [*2]．

演算パラメータ値を PL 部に転送する方法として，PS がソフトウェア実行により AXI 接続を介して PL 部のバッファに対して 64 ビット単位で書き込み／読み出しを行う方法を採る．バッファは 4 個（図 3 内の G-Buffer 0〜3）あり，メモリマップ I/O でソフトウェアからアクセス可能である．各バッファは楕円曲線上の点データ（260 ビット × 2）が格納される．各モジュールはコントロールレジスタに書き込まれる番号のバッファを使用して計算を行う．計算の過程で各基本演算器は結果を一時レジスタ（図中の TMP-Reg 0〜7）に格納し，基本演算器間のデータの授受を行う．

4.3 各演算器およびモジュールの実行サイクル数

表 2 に各基本演算器およびハードウェアモジュールの実行サイクル数を示す．逆元演算器（INVERSE）の実行サイクル数が入力データ値に依存して変動するため，これを使用する各ハードウェアモジュールの実行サイクル数も変動する．今後の課題の一つとして，SPA 攻撃に対する完全な耐性を確保するために，逆元演算器の実行サイクルを固定化するための改良が必要である．

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使用する実行の実行時間を比較評価する．

表 2 各演算器およびモジュールの実行サイクル数.

演算／モジュール	実行サイクル数
ADD	1
MUL	6
MONT_MUL	40
INVERSE	258 – 290
AFFINE_ADD	385 – 417
AFFINE_DQ	788 – 820
AFFINE DBCH	382 – 414
EXTENDED_AFFINE	422 – 454

[*2] ARM プロセッサは緩和されたメモリコンシステンシモデルを採用しているため，プログラム内で I/O レジスタアクセスを行う場合は適宜メモリバリア（DMB）を挿入してアクセス順序を保証する必要がある．

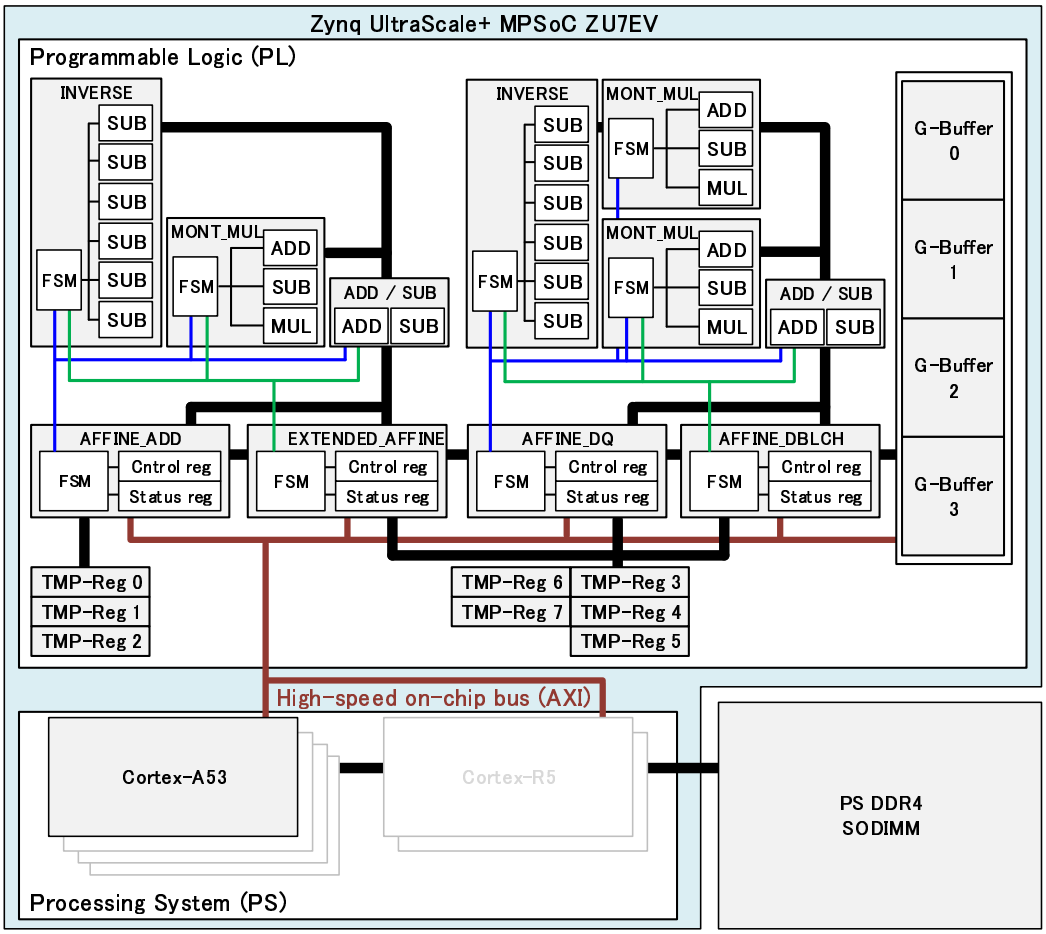

図 3　システム構成.

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群および一時レジスタ，バッファを含むインターフェース部に対して 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．デバイスに対するリソース使用率は高くはなく，本稿の設計・実装は組込みシステムとして使用可能な水準であることが確認できる．

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．スカラー倍算ソフトウェアは 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．ソフトウェアのみの実行として 2 種類を用意した．一つは逆元計算を含めてハードウェア実装と同じ演算アルゴリズムに対して，多倍長加減算および乗算をプロセッサの基本データサイズに分割して逐次的に計算するものである．これを「ソフトウェアのみの実行（ベース）」とする．もう一つは逆元計算を含む多倍長基本演算に GNU Multiple Precision Arithmetic Library（GMP） Version 6.1.2 [16] を使用するものである．これを「ソフトウェアのみの実行（GMP）」とする．ハードウェアモジュールを使用する実行を「ソフト／ハード協調実行」とする．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．

ソフトウェアのみの実行（ベース）は 500MHz のプロセッサ上で実行されるが，これに対し 214MHz で動作するハードウェアモジュールを使用する実行が約 142 倍の高速化を達成している．多倍長計算ライブラリとして定評のある GMP を使用するソフトウェア実行と比較した場合，約

表 3　FPGA リソース使用数.

Type	Used	Available	Util.(%)
CLB LUTs	55,878	230,400	24.25
CLB Registers	16,859	460,800	3.66
Block RAM	1	312	0.32
DSPs	390	1,728	22.57

表 4　スカラー倍算の実行時間.

実行方式	実行時間（ms）
ソフトウェアのみの実行（ベース）	90.611
ソフトウェアのみの実行（GMP）	7.778
ソフト／ハード協調実行	0.638

12.2 倍の高速化が確認できる．

本研究と同じ楕円曲線およびフィールドサイズに対するスカラー倍算ハードウェアの研究として Javeed らによる研究が挙げられる [18]．この研究では bit-serial interleaved 乗算を使用して高速化を図っているが，十分な並列実行が行われていない．70MHz の動作周波数で実行時間が 2.8ms と報告されており，本稿における実行が 4.4 倍高速である．使用する FPGA デバイスに性能差があるものの，同一動作周波数として比較を行った場合も 1.4 倍高速である．

以上の結果から，本稿におけるソフト／ハード協調実行は十分な高速化を達成していることが確認された．

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．評価において提案・実装したハードウェアはソフトウェア実行に対し 12.2 倍，および同一楕円曲線を対象とする既存のハードウェア実装と比較し 4.4 倍の高速化が確認された．今後は逆元演算器の実行時間の固定化を行い，さらに他の関連する楕円曲線暗号ハードウェアとの比較評価を行う予定である．

謝辞 本研究の一部は文部科学省の平成 30 年度「Society 5.0 実現化研究拠点支援事業」及び JSPS 科研費 19K11873 の助成を受けています．

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] de I. Blake，G. Seroussi，N. Smart: Elliptic Curves in Cryptography，Cambridge University Press(1999).
[3] 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).
[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] Y. Jin, A. Miyaji: Secure and Compact Elliptic Curve Cryptosystems, Proc. of the 24th Australasian Conference on Information Security and Privacy (ACISP 2019), Lecture Notes in Computer Science, Springer-Verlag, pp.339–650 (2019).
[11] D-P. Le, B.P.Nguyen, Fast point quadrupling on elliptic curves, Proc. of SoICT, pp.218–222 (2012).
[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] P.L.Montgomery: Modular multiplication without trial division, Mathematics of Computation, 44(170), pp.519–521 (1985).
[15] A.J.Menezes, et al.: Handbook of Applied Cryptography, CRC Press (1996).
[16] `https://gmplib.org/`
[17] Xilinx, Inc.: Zynq UltraScale+ MPSoC Data Sheet: Overview, `https://www.xilinx.com/support/documentation/data_sheets/ds891-zynq-ultrascale-plus-overview.pdf`
[18] K. Javeed, X. Wnag: FPGA Based High Speed SPA Resistant Elliptic Curve Scalar Multiplier Architecture, International Journal of Reconfigurable Computing, Vol.2016, No.5, pp.1–10 (2016).