Vol.2013-DPS-154 No.8
Vol.2013-CSEC-60 No.8
2013/3/14

# 公開鍵暗号ハードウェアのための多ビット乗算器について

白勢 政明[1,a)]　木村 圭吾[1]　村山 広行[1]　加藤 翔[1]　小林 悠太[1]　畠山 遼平[1]

**概要**：多くの公開鍵暗号は多ビット整数乗算を必須とするため，乗算器の性能はそれらのためのハードウェアの性能に影響を与える．Wallace tree 乗算器は，ビット数を $n$ とし配線遅延を無視すると，処理時間は $\log n$ に比例する．従って例えば，正しく設計するならば 64 ビット乗算器と 128 ビット乗算器との処理時間の差は理論的にはわずかである．本稿は，配線遅延以外の性能が予定通りとなり，ハードウェアの記述が容易な，更にパイプライン化が容易な，任意のビット数の Wallace tree 乗算器の構成法を提案する．

**キーワード**：公開鍵暗号, 乗算器, Wallace tree, ハードウェア記述言語, VHDL

# On A Large-scale Multiplier for Public Key Cryptographic Hardware

Masaaki Shirase[1,a)]　Kimura Keigo[1]　Murayama Hiroyuki[1]　Kato Shou[1]　Kbayashi Yuta[1]
Hatakeyama Ryohei[1]

***Abstract:*** Performance of multipliers influences one of hardwares for many public key cryptographies because such cryptographies require many large-bit integer multiplications. It is known that processing time of $n$ bit Wallace tree multiplier is proportional to $\log n$ ignoring wiring delay. Therefore, the difference between processing time of 64 bit multiplier and one of 128 bit multiplier is a little in theory when multipliers are correctly designed. This paper proposes a design method of Wallace tree multiplier with arbitrary bit which has easy hardware description and correctness, and can be easily pipelined.

***Keywords:*** Public key cryptography, Multiplier, Wallace tree, Hardware description language, VHDL

## 1. はじめに

最も広く普及している公開鍵暗号は RSA 暗号であるが，近年楕円曲線暗号も普及し始めている．両者に共通していることは，暗号処理に多ビットの剰余乗算 $a \times b \bmod n$ が多く必要なことである．例えば，RSA 暗号では $a, b, n$ は 1024〜2048 ビット，楕円曲線暗号では $a, b, n$ は 160〜256 ビットのサイズが標準的である．Montgomery 乗算 [10] や擬 mersenne 素数剰余算法 [1] 等を用いると，剰余乗算のうちの剰余計算 mod$n$ は乗算やシフト処理に帰着できる．従って，普及している公開鍵暗号の処理の高速化には，多ビット非負整数乗算の高速化が重要である*1 多ビット整数乗算や剰余乗算の方法として，次のどれかが用いられることが多い．

(1) シフトアド法
(2) 小サイズに分割し複数回の乗算を実行
　(a) 複数回の乗算を直列的に実行
　(b) 複数回の乗算を並列的に実行
　(c) 複数の小さな素数での剰余乗算に帰着させる RNS(residue number system)[4] の使用

(2)–(a,b) の場合一般に Karatuba 法等を用いなければ，$kn$ ビット整数乗算には，$k^2$ 回の $n$ ビット整数乗算が必要であり，64 ビット乗算の回数は，例えば，256 ビット乗算では 16 回，1024 ビット乗算では 256 回となる．

公開鍵暗号の処理を高速化する手段の一つに，専用ハードウェアの使用が挙げられる．実際に RSA 暗号や楕円曲線暗号のハードウェアの研究が数多くなされてきた．いく

1　公立はこだて未来大学
116-2 Kamedanakano, Hakodate, Hokkaido 042–8655, Japan
a)　shirase@fun.ac.jp
*1　暗号処理には，このような多ビット非負整数乗算が数千回必要となる．

つかの既存の公開鍵暗号ハードウェアの剰余乗算や乗算の方法に着目すると，[12] ではシフトアド法による乗算，[3] ではシフトアド法による Montgomry 乗算を行っており，[2] では RNS を使用し，[18] では 18 ビット乗算器を設計しかつ RNS を使用し，[6] では 32 ビット乗算器の使用，[7] では 64 ビットまでの乗算器を使用した時の暗号ハードウェアの性能比較を行っている．また，[11] では様々な構成法による 32 ビットまでの剰余乗算をハードウェア実装し性能を比較している．多ビット整数乗算が必要な公開鍵暗号ハードウェアにおいても，搭載している乗算器は 64 ビットまでのようである[*2]．

著者等は，公開鍵暗号ハードウェアの高速化のために，既存の乗算器には頼らず多ビット高速乗算器から設計してみたいと考えた．高速乗算器である Wallace tree 乗算器 [17] の設計法は，[15] や [8] などでも扱われているが，本稿は，ハードウェア記述言語による記述とパイプライン化が容易な Wallace tree 乗算器の設計法を提案する．更に，提案手法に基づき記述した 128 ビット乗算器の VHDL の，Altera 社から市販されている FPGA 開発キットに搭載されている Cyclone IV GX EP4CGX150DF31C7 をターゲット FPGA とした時の，Quartus II ウェブ・エディションによる合成結果を提示する[*3]．

ここで注意したいことは，Wallace tree 乗算器の処理時間は，配線遅延を無視すると，ビット数の対数に比例することである．従って，$n$ ビット Wallece tree 乗算器と $2n$ ビット Wallace tree 乗算器の処理時間の増加はわずかである．(ただし，ゲート数はビット数の 2 乗に比例する．)

### 1.1 表記について

本稿では $(a_{n-1}a_{n-2}\cdots a_1a_0)_2$ を

$$a_{n-1}2^{n-1}+a_{n-2}2^{n-2}+\cdots+a_12+a_0,\ a_i\in\{0,1\}$$

の 2 進数表記とする．

各図において，論理積 $a$ and $b$ を $ab$ と略記している．

## 2. 乗算器

本節では，配列型乗算器と Wallace tree 乗算器の一般論を説明する．

### 2.1 乗算について

$123\times456$ を筆算で計算すると

$$\begin{array}{cccccc}
 & & & 1 & 2 & 3\\
\times & & & 4 & 5 & 6\\
\hline
 & & & 7 & 3 & 8\\
 & & 6 & 1 & 5 & \\
+ & 4 & 9 & 2 & & \\
\hline
 & 6 & 6 & 0 & 8 & 8
\end{array}$$

[*2] Intel の Itanium CPU は 64 ビット Wallace tree 乗算器を搭載している [13].

[*3] 本稿は，平成 24 年度公立はこだて未来大学プロジェクト学習「暗号ハードウェア–乗算器編」の成果をまとめたものである．

となるが，この計算は乗数と被乗数の各桁の値の九九と，加算の繰り返しから成っている．

計算機では整数は 2 進数で表現されるが，2 進数での乗算も，10 進数の場合と基本は同じである．例えば，$(10101)_2\times(11001)_2$ は，

$$\begin{array}{cccccccccc}
 & & & & & 1 & 0 & 1 & 0 & 1\\
\times & & & & & 1 & 1 & 0 & 0 & 1\\
\hline
 & & & & & 1 & 0 & 1 & 0 & 1\\
 & & & & 1 & 0 & 1 & 0 & 1 & \\
 & & & 1 & 0 & 1 & 0 & 1 & & \\
 & & 1 & 0 & 1 & 0 & 1 & & & \\
+ & 1 & 0 & 0 & 0 & 0 & 0 & & & \\
\hline
 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1
\end{array}$$

となり，乗数と被乗数の各桁の値の九九と，加算の繰り返しから成っている．但し，2 進数では九九に対応する計算は

| $a$ | $b$ | $c=a\times b$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

のみであり，これは論理積 $c=a$ and $b$ と一致する．$(a_{n-1}a_{n-2}\cdots a_1a_0)_2\times(b_{n-1}b_{n-2}\cdots b_1b_0)_2$ で必要となる $a_i$ and $b_j$ を部分積という．乗算器は，部分積を生成した後で加算の繰り返しを行う回路である．乗算器の高速化には，この繰り返される加算を如何に高速化するかが重要である．

**注意 1**

乗算 $(a_{n-1}a_{n-2}\cdots a_1a_0)_2\times(b_{n-1}b_{n-2}\cdots b_1b_0)_2$ における部分積 $a_i$ and $b_j$ は，$a_i2^i\times b_j2^j=(a_i \text{ and } b_j)2^{i+j}$ の結果であるため，$2^{i+j}$ の位を表している．

### 2.2 加算器

#### 2.2.1 半加算器と全加算器

2 つの $a,b\in\{0,1\}$ を入力とし，

$$2\times cout+s=a+b,\ cout,s\in\{0,1\}$$

を満たす $cout,s$ を出力する回路を半加算器 (HA, half adder) という．HA の入力と出力の関係を真理表で表すと，

| 入力 | | 出力 | |
|---|---|---|---|
| $a$ | $b$ | $cout$ | $s$ |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

となる．なお，出力の $s$ は $sum$ の頭文字であり，$cout$ は carry out の略で「桁あがりの出力」の意味である．

3 つの $a,b,cin\in\{0,1\}$ を入力とし，

$$2\times cout+s=a+b+cin,\ cout,s\in\{0,1\}$$

を満たす $cout,s$ を出力する回路を全加算器 (FA, full adder)

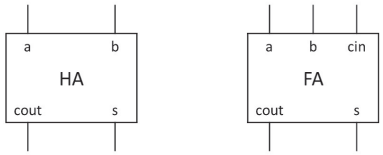


**図 1** HA(半加算器) と FA(全加算器) の概略図
**Fig. 1** Briefs of HA(half adder) and FA(full adder)

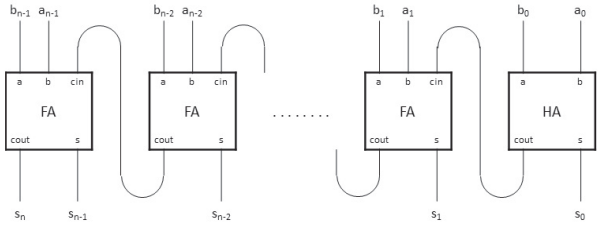


**図 2** n ビット順次桁上げ加算器
**Fig. 2** n bit ripple carry adder

という．FA の入力と出力の関係を真理表で表すと，

| 入力 | | | 出力 | |
|---|---|---|---|---|
| $a$ | $b$ | $cin$ | $cout$ | $s$ |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

となる．なお，入力の $cin$ は carry in の略で「桁あがりの入力」の意味である．

HA と FA の概略を図 1 のように描く．

### 2.2.2 n ビット加算器

$n$ ビットの 2 進数 $(a_{n-1}a_{n-2}\cdots a_1a_0)_2$ と $(b_{n-1}b_{n-2}\cdots b_1b_0)_2$ の和 $(s_ns_{n-1}\cdots s_1s_0)_2$ (和は桁あがりにより $n+1$ ビットとなる) を求める回路は，$n-1$ 個の FA と 1 つの HA を図 2 のように組み合わせて構成できる．このように構成された加算器を $n$ ビット順次桁上げ加算器といい，FA/HA を $n$ 回通るため，遅延時間は $n$ に比例する．

本稿では詳しくは説明しないが，順次桁上げ加算器を構成する各 FA の $cin$ の入力を予め生成することで，遅延時間が $\log_2 n$ に比例するような加算器 (桁上げ先見加算器) が知られている．(順次桁上げ加算器と桁上げ先見加算器の入出力の関係は同じである．)

なお，構造に依らず $n$ ビット加算器の概略を図 3 のように描くことにする．

## 2.3 乗算器

### 2.3.1 配列型乗算器

設計が容易な配列型乗算器を説明する．6 ビットの整数乗算

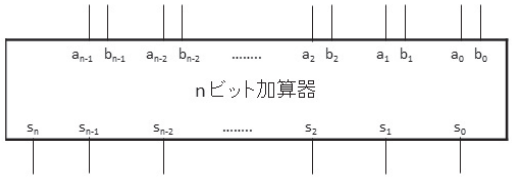


**図 3** n ビット加算器の概略
**Fig. 3** Brief of n bit adder

$$(s_{11}s_{10}s_9s_8s_7s_6s_5s_4s_3s_2s_1s_0)_2 = (a_5a_4a_3a_2a_1a_0)_2 \times (b_5b_4b_3b_2b_1b_0)_2$$

を例にすると，この計算のための配列型乗算器は図 4 のようになり，19 個の FA，5 個の HA，6 ビット加算器から構成される．

図 4 の FA や HA が平行四辺形に並んでいる部分では，列毎に注目すると部分積 $a_ib_j$ の $i+j$ の値が等しくなっている．例えば，図 4 の右から 3 列目に入力されている部分積は，$a_3b_0, a_2b_1, a_1b_2, a_0b_3$ となっている．つまり，同じ列には同じ位の部分積が入力される．これは，**注意 1** により各列とも $2^3$ 位の値が入力されていることを意味する．

乗算器において，FA や HA が平行四辺形に並んでいる部分は，桁上げ (carry) の結果をそのままセーブして 1 つ上の位へ渡しているため，CSA(carry save adder) と呼ばれる．なお，CSA の行数をステージ数と呼ぶことにする．図 4 の 6 ビット乗算器の CSA はステージ数は 4 である．また，各ステージを上からステージ 0,1,2,3 と呼ぶことにする．CSA の終わりでは，各位とも 2 ビット以下になっている．

これに対して，図 4 の 6 ビット加算器のような桁上げ伝播を行う加算器を CPA(carry propagation adder) という．CPA の出力において各位のビットが 1 つとなり，これが乗算結果となる．

**注意 2**
どの CSA ステージも処理時間は FA の処理時間に等しい．

1 つのビットを • で表すと，配列型乗算器の処理は以下のようになる．

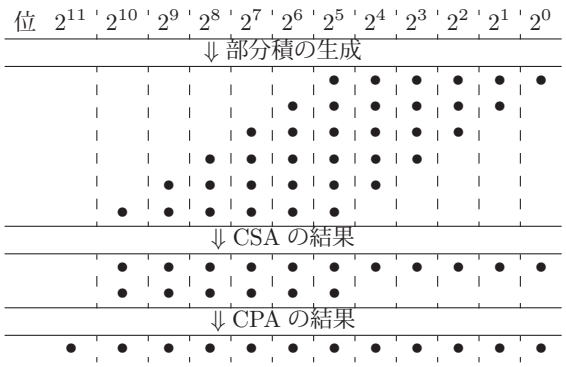


図 4 は 6 ビット乗算器を描いているが，任意の $n$ ビット乗算器も同様に構成できる．$n$ ビットの場合では，CSA の

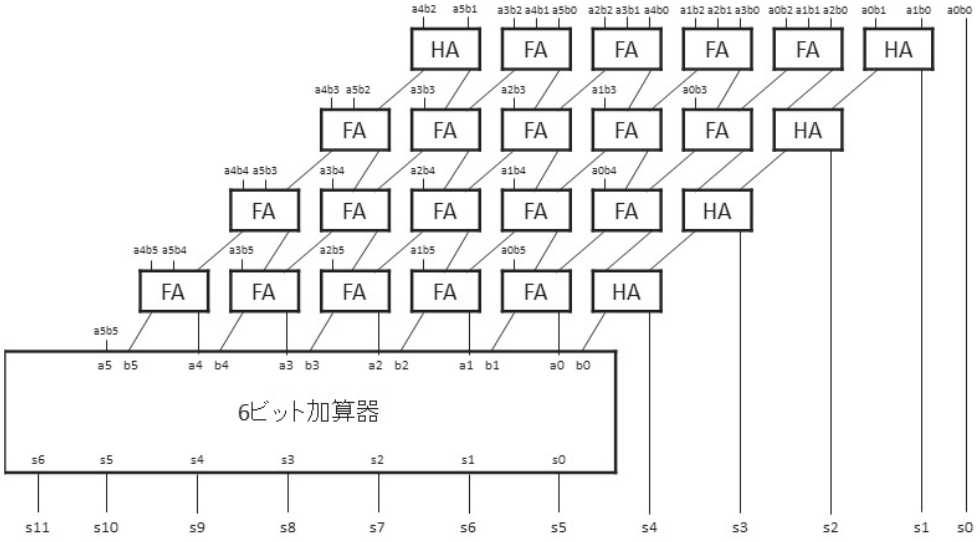


**図 4** 6 ビット乗算器
**Fig. 4** 6 bit multiplier

ステージ数は $n-2$，CPA のビット数は $n$ となる．

配線遅延を無視すると，部分積の生成時間は定数となり，配列型乗算器において $n$ ビット乗算器の CSA の処理時間は $n-2$ に比例する．また，CPA の処理時間は，桁上げ先見加算器を使用するならば，2.2.2 節で説明したように $\log_2 n$ に比例する．従って $n$ が大きい場合，配列型乗算器の処理時間は CSA が支配的となる．

**配列型乗算器の特徴**

配線遅延を無視すると

(1) 部分積生成時間は定数.
(2) ステージ 0 を除く CSA の各ステージで，各位のビットが 1 つずつ削減される.
(3) CSA の処理時間は $n-2$ に比例.
(4) CPA の処理時間は $\log_2 n$ に比例.
(5) 従って，配列型乗算器全体の処理時間は CSA の処理時間が支配的となり $n-2$ に比例.

### 2.3.2 リデューサー

乗算器の CSA を構成する FA は，同じ位に対応する 3 ビット $(a, b, cin)$ を入力とし，同じ位の 1 ビット $(s)$ と 1 つ上の位の 1 ビット $(cout)$ の計 2 ビットを出力する．つまり，FA は入力 3 ビットを出力 2 ビットに削減している．このため，CSA を構成する FA をリデューサー (reducer) と呼ぶことがある．

## 2.4 Wallace tree 乗算器

Wallace tree 乗算器は高速乗算器の一つである．Wallace tree 乗算器では，初めに CSA の前に全ての部分積を生成する．部分積の生成は入力の各ビット同志の and を計算するだけであるが，各ビットを $n$ 個コピーする必要があり，そのためには $\log_2 n$ に比例する時間が一般に必要となる．CSA の各ステージでは，各位のビットを 3 ビットずつにまとめ FA に通して 2 ビットを生成，2 ビットが余る時はその 2 ビットを HA に通し[*4]，1 ビットが余る時はその 1 ビットに対しては何もしない．各位とも，CSA の各ステージの出力のビット数は，入力のビット数の約 2/3 倍となる．全ての位のビット数が 2 以下になるまで，CSA のステージを繰り返す．最後に CPA に通す．従って，$n$ ビット Wallace tree 乗算器では，位毎の CSA への入力ビット数は $2^{n-1}$ の位での $n$ が最大であるため，CSA のステージ数は約 $\log_{3/2} n$ となる．(2.3.1 節で説明した配列型乗算器では，CSA のステージ数は $n-2$ である．)

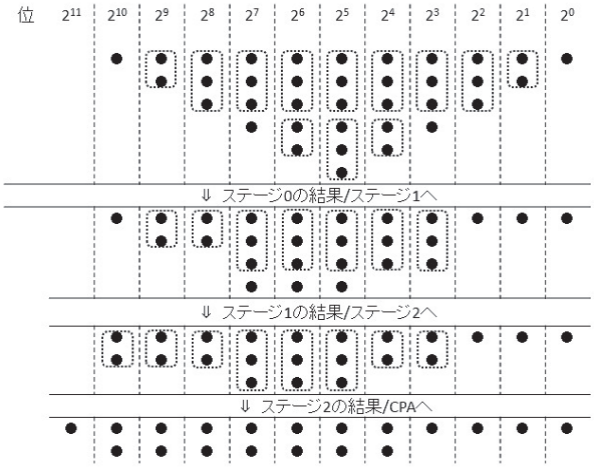


**図 5** 6 ビット Wallace tree 乗算器の処理
**Fig. 5** Process of 6 bit Wallace tree multiplier

例えば 6 ビットの場合，Wallace tree 乗算器の CSA は図 5 からステージ数が 3 であることが分かり，6 ビット配列型乗算器場合と比較して 1 ステージ削減できる．また，

*4 HA はリデューサーとしての機能がないので，余る 2 ビットを HA に通す必要はないかもしれない.

$n$ が大きいほど CSA のステージ数の削減率は大きくなる．

**Wallace tree 乗算器の特徴**

配線遅延を無視すると

(1) 部分積生成時間は $\log_2 n$ に比例．
(2) CSA の各ステージで，各位のビット数が約 2/3 倍となる．
(3) CSA の処理時間は $\log_{3/2} n$ に比例．
(4) CPA の処理時間は $\log_2 n$ に比例．
(5) 従って，Wallace tree 乗算器全体の処理時間は $\log n$ に比例．

**注意 3**

Wallace tree 乗算器ではビット数を 2 倍にしても，増加する処理時間はわずかである．

**注意 4**

配列型乗算器と Wallace tree 乗算器の CSA の違いは，基本的には物理的な配線の違いだけである．そのため両者のゲート数の差は，冗長なリデューサーのためゲートによるものであり，わずかである．

## 3. 本稿の寄与

### 3.1 本稿の動機

2.4 節で説明したように，$n$ ビット Wallace tree 乗算器の処理時間は配線遅延を無視するとおおよそ $\log n$ に比例する．この特性により，著者等は，乗算器を自ら設計することで，公開鍵暗号ハードウェアを以下のように改良できると期待している．

(1) 乗算器の制約を受けない設計
(2) 処理時間の短縮
　クロック周波数を上げない範囲で可能な限り乗算器のビット数を大きくすると，暗号処理で必要とする乗算器の使用回数が削減できる．
(3) プログラムの簡単化
　高速乗算のための乗算の並列化や Karatsuba 法等の使用が削減または不要になる．

しかしながら，Wallace tree 乗算器の構成は理論的には困難ではないが，多ビットの Wallace tree 乗算器をハードウェア記述言語で記述することは容易な作業ではないと考えられる．また，記述したとしても配線が複雑になるため記述に誤りがないことを確認したり，CSA のステージ数を予定通り $\log_{3/2} n$ にさせることも困難そうである．

本稿は，Wallace tree 乗算器の CSA の分かりやすい構成法を提案し，その手法により記述した 128 ビット Wallace tree 乗算器の合成結果を提示する．「128 ビット」を選んだ理由は，以下の通りである．

(1) 64 ビット Wallace tree 乗算器は既に存在している．
(2) 公開暗号では，128 の倍数ビット整数乗算が必要となることが多い．

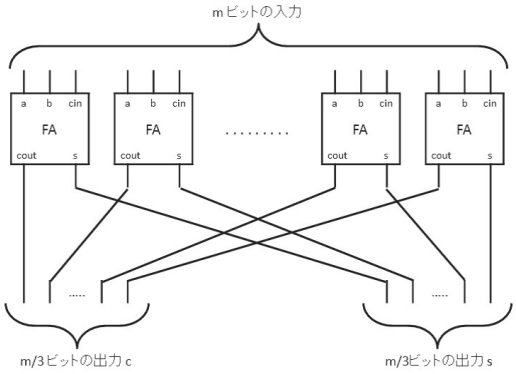


**図 6** Red $m$ ($m \bmod 3 = 0$) の図
**Fig. 6** Figure of Red $m$ ($m \bmod 3 = 0$)

(3) 256 ビット Wallace tree 乗算器の VHDL 記述は困難であることが予想され，記述できたとしても FPGA に実装できるか判断できなかった．

なお，副産物的であるが，提案手法での Wallace tree 乗算器は，パイプライン化が容易である．

### 3.2 回路 Red $m$

本稿が提案する Wallace tree 乗算器の構成法で最も特徴的な点は，回路 Red $m$ の導入である (Red は reducer の意味)．回路 Red $m$ は，正整数 $m$ に対して $m$ ビット $(a_{m-1}a_{m-2}\cdots a_1a_0)_2$ を入力とする以下のような回路である．

$m = 1$ の場合
回路と呼べるか分からないが，入力 1 ビット $a_0$ をそのまま出力する ($s = a_0$) 回路とする．

$m \geq 2$ の場合
$m$ mod 3 の値によって更に 3 つの場合分けを行う．

$m \bmod 3 = 0$ の場合
$k = m/3$ とする．図 6 のように，FA を $k$ 個並べ，Red $m$ の $m$ ビットの入力を 3 ビットずつ各 FA へ入力する．各 FA の出力 $cout$ を纏めたものを Red $m$ の出力 $c$($c$ は $k = m/3$ ビット) とし，各 FA の出力 $s$ を纏めたものを Red $m$ の $s$($s$ は $k = m/3$ ビット) とする．Red $m$ は入力の $m$ ビットを $2m/3$ ビットの出力に削減する．

$m \bmod 3 = 1$ の場合
$k = (m-1)/3$ とする．図 7 のように，FA を $k$ 個並べ，Red $m$ の $m$ ビットの入力のうち，$m-1$ ビットを 3 ビットずつ各 FA へ入力する．どの FA にも入力されない残りの 1 ビットを $a_0$ と仮定する．各 FA の出力 $cout$ を纏めたものを Red $m$ の出力 $c$($c$ は $k = (m-1)/3$ ビット) とし，各 FA の出力 $s$ と $a_0$ を纏めたものを Red $m$ の $s$($s$ は $k+1 = (m+2)/3$ ビット) とする．Red $m$ は入力の $m$ ビットを $(2m+1)/3$ ビットの出力に削減する．

$m \bmod 2 = 2$ の場合

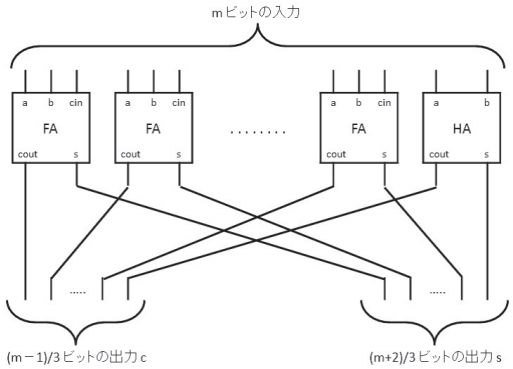


**図 7** Red $m$ ($m \bmod 3 = 1$) の図
**Fig. 7** Figure of Red $m$ ($m \bmod 3 = 1$)

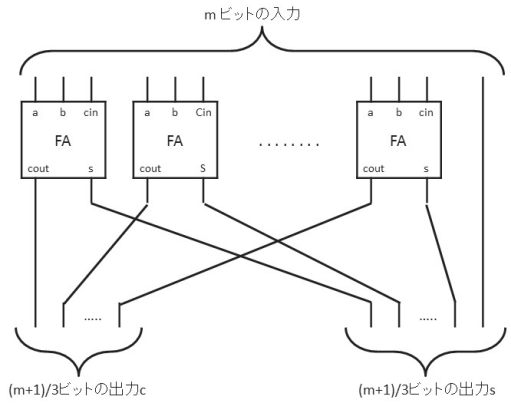


**図 8** Red $m$ ($m \bmod 3 = 2$) の図
**Fig. 8** Figure of Red $m$ ($m \bmod 3 = 2$)

$k = (m-2)/3$ とする．図 8 のように，FA を $k$ 個，HA を 1 つ並べ，Red $m$ の $m$ ビットの入力のうち，$m-2$ ビットを 3 ビットずつ各 FA へ入力し，残りの 2 ビットを HA へ入力する．各 FA の出力 *cout* と HA の出力 *cout* を纏めたものを Red $m$ の出力 $c$($c$ は $k+1 = (m+1)/3$ ビット) とし，各 FA の出力 $s$ と HA の出力 $s$ を纏めたものを Red $m$ の出力 $c$($c$ は $k+1 = (m+1)/3$ ビット) とする．Red $m$ は入力の $m$ ビットを $(2m+2)/3$ ビットの出力に削減する．

Red $m$ の概略図を図 9 ように描くことにする．

**注意 5**
Red $m$ の入力の全ての $m$ ビットが同じ位とする．すると，出力 $s$ の各ビットは入力と同じ位，出力 $c$ の各ビットは入力より 1 つ上の位となる．

**注意 6**
配線遅延を無視すると，Red 1 の処理時間は 0，Red 2 の処理時間は HA の処理時間に等しく，Red $m$ ($m \geq 2$) の処理時間は FA の処理時間に等しい．

**注意 7**
各 Red $m$ は規則的にハードウェア記述言語で記述することが容易である．

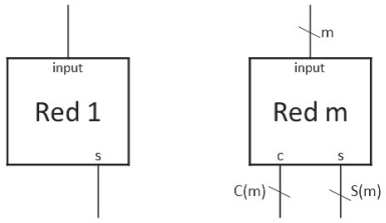


**図 9** Red 1 と Red $m$ ($m \geq 2$) の概略図
**Fig. 9** Brief of Red 1 and Red $m$ ($m \geq 2$)

### 3.3 $n$ ビット Wallace tree 乗算器の構成法

この節の目的は，回路 Red $m$ を使った Wallace tree 乗算器の構成法を提案することであり，この構成法は 3 つの工程，(3.3.1 節で説明する) リデューサー表の作成，Red $m$ の配置と接続，最終工程，から成る．

#### 3.3.1 リデューサー表の作成

$n$ を設計したい Wallace tree 乗算器のビット数とする．まず，自然数 $x$ を定義域とする関数 $C(x)$ と $S(x)$ を次のように定義する．

$$C(x) = \text{Red } x \text{ の } c \text{ のビット数} = \begin{cases} m/3 & x \bmod 3 = 0 \text{ の時} \\ (m-1)/3 & x \bmod 3 = 1 \text{ の時} \\ (m+1)/3 & x \bmod 3 = 2 \text{ の時} \end{cases}$$

$$S(x) = \text{Red } x \text{ の } s \text{ のビット数} = \begin{cases} m/3 & x \bmod 3 = 0 \text{ の時} \\ (m+2)/3 & x \bmod 3 = 1 \text{ の時} \\ (m+1)/3 & x \bmod 3 = 2 \text{ の時} \end{cases}$$

次に，$0 \leq j \leq 2n-2$ に対して，自然数 $n_{0,j}$ を以下のように定義する．

$$n_{0,j} = \begin{cases} j+1 & 0 \leq j \leq n-1 \\ 2n-j-1 & n \leq j \leq 2n-2 \end{cases}$$

$n_{0,j}$ は，$n$ ビット乗算器の $2^j$ の位の部分積の個数である．$i \geq 1$ に対して，$n_{i,j}$ を $\{n_{i-1,j}\}$ と $C(x)$，$S(x)$ 使って以下のように定義する．

$$n_{i,j} = \begin{cases} S(n_{i-1,j}) & j = 0 \text{ の時} \\ S(n_{i-1,j}) + C(n_{i-1,j-1}) & j \geq 1 \text{ の時} \end{cases}$$

$i$ をインクリメントをしながら

$$\max_{0 \leq j \leq 2n-2} \{n_{i,j}\} = 3 \qquad (1)$$

となるまで，$n_{i,j}$ の生成を続ける．式 (1) が成り立つ最小の $i$ の値を $i_{MAX}$ とおく．このように生成される $n_{i,j}$ を $i$ 行，右から $j$ 列に置いた表をリデューサー表と呼ぶことにする．

例えば，$n = 6$ のリデューサー表は次のようになる ($i_{MAX} = 2$)．

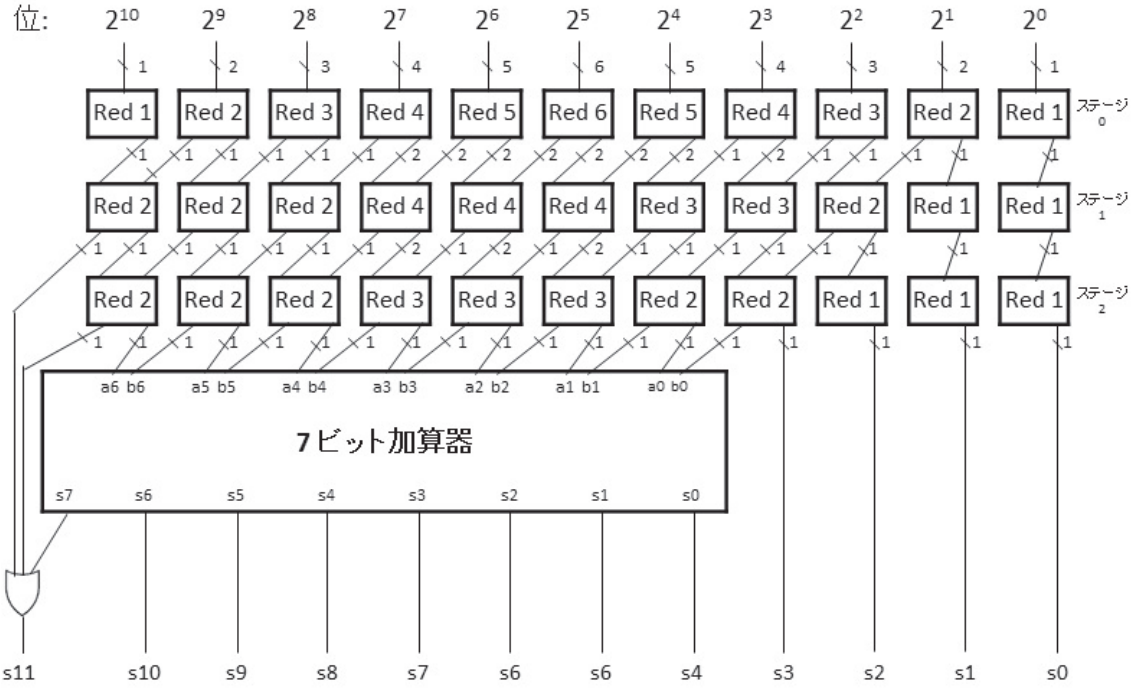


**図 10** 提案手法による 6 ビット Wallace tree 乗算器
**Fig. 10** Proposed 6 bit Wallace tree multiplier

| $i \setminus j$ | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 5 | 4 | 3 | 2 | 1 |
| 1 | 2 | 2 | 2 | 4 | 4 | 4 | 3 | 3 | 2 | 1 | 1 |
| 2 | 2 | 2 | 2 | 3 | 3 | 3 | 2 | 2 | 1 | 1 | 1 |

#### 3.3.2 Red $m$ の配置と接続

まず，リデューサー表に従って，Red $n_{i,j}$ を配置する．各 Red $n_{i,j}$ の入力を以下のようにする．

(1) 各 Red $n_{0,j}$ の入力を $2^j$ の位の部分積の連結

$$a_0b_j||a_1b_{j-1}||a_2b_{j-2}||\cdots||a_jb_0$$

とする．

(2) $i \geq 1,\ j = 0$ に対して，Red $n_{i,0}$ の入力は Red $n_{i-1,0}$ の $s$ とする．

(3) $i \geq 1,\ j \geq 1$ に対して，Red $n_{i,j}$ の入力は，Red $n_{i-1,j}$ の $s$ と Red $n_{i-1,j-1}$ の $c$ の連結とする．

各 Red $n_{i_{MAX},j}$ の出力はどの $j$ に対しても 2 ビット以下である．

これで提案手法による Wallace tree 乗算器の CSA は完成である．

#### 3.3.3 最終工程

CPA のための加算器を用意し，CSA の出力が 2 ビットとなっている位の信号を入力とする．

最後に，各 Red $n_{i,2n-2}$ の *cout* (これらは $2^{2n-1}$ の位である) と CPA の出力の最上位 (これも $2^{2n-1}$ の位である) の OR をとり，乗算結果の最上位の位とする．

**注意 8**

各 Red $n_{i,2n-2}$ の *cout* 及び CPA の最上位の出力において，1 となるビットは高々 1 つである．もし 2 つ以上ならば，$n$ ビットの乗算結果が $2^{2n}$ 以上となるが，これはあり得ない．従って，$2^{2n-1}$ の位のビット処理は単に OR をとれば十分である．

以上の方法で 6 ビット乗算器を構成すると図 10 のようになる．この図から分かるように，$i_{MAX}+1$ が CSA のステージ数となる．

各 Red $n_{i,j}$ は面積の差が大きいため，実際に配置配線を行うと図 10 のようなきれいな配置は不可能と思われる．しかしながら，この図から各 Red $n_{i,j}$ を規則的に接続できる．

### 3.4 乗算器のハードウェア記述

3.3 節で述べた手法の $n$ ビット乗算器をハードウェア記述言語で記述する方法の概要を説明する．なお，以下で使用している用語は VHDL での記述を念頭に置いている．

(1) リデューサー表 $\{n_{i,j}\}_{0\leq i\leq i_{MAX},0\leq j\leq 2n-2}$ を作成．

(2) コンポーネントを記述．
- (a) FA と HA を記述．
- (b) (既存の手法により，)CPA のとなる加算器を記述．
- (c) FA と HA をコンポーネントとして，$1 \leq m \leq 2n-2$ に対して Red $m$ を記述．これらは規則的に記述することが可能．

(3) 乗算器本体の記述．

(a) 各 Red $m$ 間の接続のための中間変数を宣言.
(b) コンポーネントの呼び出し (各 Red $n_{i,j}$ と CPA).
(c) 部分積を記述.
(d) リデューサー表に基づき，各コンポーネント間の接続を記述.
(e) 3.3.3 で説明した結果の最上位ビットのための処理を記述.

### 3.5 128 ビット乗算器の VHDL のシミュレーション/合成結果

提案手法により 128 ビット Wallace tree 乗算器を VHDL で記述し[*5]，論理シミュレータ ModelSim によるテストベンチの確認したところ，誤りは見つからなかった.

Altera 社から市販されている Cyclone IV GX FPGA 開発キット [5] に搭載されている FPGA，Cyclone IV GX EP4CGX150DF31C7 [*6]をターゲットとし，合成ツール Quartus II ウェブ・エディション [16] によって，上記 VHDL の合成を行った．但し，FPGA の I/O の制約のため，1 つの 128 ビットデータをコピーして，128 ビット乗算器に入力する回路に修正した．その結果，FPGA の LE の使用率は 28%となった．従って著者等は，該当 FPGA に対して 128 ビット Wallace tree 乗算器を自ら設計しても，残りの 72%の LE を使用して暗号ハードウェア全体の設計が可能でないかと期待している.

## 4. まとめと今後の課題

本稿は，公開鍵暗号ハードウェアにとって有用と思われる Wallace tree 乗算器の設計の一手法を提案した.

提案手法の特徴は，記述が容易な回路 Red $n_{i,j}$ を導入したことである．すると，図 10 のように，各 Red $n_{i,j}$ を配列型に配置でき，各 Red $n_{i,j}$ 間の接続は単純なものとなる．また，提案手法により生成した 128 ビット乗算器を，市販されている FPGA をターゲットとし合成を行った[*7].

提案手法による乗算器のハードウェア記述は規則的あり配線が単純である．従って，ビット数 $n$ を与えると $n$ ビット Wallace tree 乗算器のハードウェア記述言語のコードを生成するプログラムを比較的簡単に作成できそうであり，このプログラムの完成を今後の課題としたい．そして，何ビットまでの乗算器が FPGA に実装できるか実証を行いたい．また，今回は配線遅延を全く考慮していないので，配線遅延を考慮するとどうなるか検証してみたい．最終的には，多ビット (128 ビット以上) 乗算器を搭載した公開鍵暗号ハードウェアを設計し，多ビット乗算器の使用にどのような効果があるか検証を行いたい.

[*5] CPA には 243 ビット桁上げ先見加算器を記述した.
[*6] Cyclone IV GX EP4CGX150DF31C7 の仕様は，LE 数が 149,760，ユーザ I/O 数が 508，コアの電圧が 1.2V である.
[*7] FPGA へのコンフィギュレーションは行っていない.

### 参考文献

[1] Bailey, D.V. and Paar. C.: Optimal Extension Fields for Fast Arithmetic in Public-Key Algorithms, *CRYPTO'98*, LNCS 1462, pp. 472–485 (1998).
[2] Bajard, J.-C. and Imbert, L.: A Full RNS Implementation of RSA Laurent Imbert Jean-Claude Bajard *IEEE Trans. on Computers* Vol. 53, No. 6, pp. 769–774. (2004).
[3] Batina, L., Preneel, B., and Vandewalle, J.: Hardware Implementation of An Elliptic Curve Processor Over $GF(p)$, *Application-Specific Systems, Architectures, and Processors, 2003*, Proceedings, pp. 433–443. (2003)
[4] Blake, I., Seroussi, G., and Smart, N.: *Elliptic Curves in Cryptography,* Cambridge University Press. (1999)
[5] Cyclone IV GX FPGA 開発キット，Altera 社，入手先 ⟨http://www.altera.co.jp/products/devkits/altera/kit-cyclone-iv-gx.html⟩
[6] Mesquita, D.G., Perin, G., Herrmann, F.L., and Martins, J.B.: An Efficient Implementation of Montgomery Powering Ladder in Reconfigurable Hardware *SBCCI'10*, Proceedings, pp. 121-126.
[7] Miyamoto, Q., Homma, N, and Aoki, T.: Systematic Design of RSA Processors Based on High-Radix Montgomery Multipliers, *IEICE Trans. on VLSI*, Vol. 19, No. 7, pp. 1136–1146 (2011).
[8] 水口貴之，味元伸太郎，橘昌良：木構造部分加算器回路をもつ乗算器の高速化に関する研究，情報処理学会研究報告 2006-SLDM-127.
[9] Li, W. and Wanhammar, L.: VHDL Code Generator for a Complex Multiplier, 入手先 ⟨http://www.tde.isy.liu.se/publications/papers_and_reports/1999/weidongl_RVK99.pdf⟩ . (1999)
[10] Montgomery, L.P.: Modular Multiplication Without Trial Division, *Math Comp.*, Vol. 44, No. 170, pp. 519–521 (1985).
[11] Nedjah, N. and Mourelle, L.: A Review of Modular Multiplication Methods And Respective Hardware Implementations, *Informatica,* No. 30, pp. 111–129. (2006)
[12] Orlando, G. and Paar, C.: A Scalable $GF(p)$ Elliptic Curve Processor Architecture for Programmable Hardware, *CHES 2001*, Cryptographic Hardware and Embedded Systems, Springer. 入手先 ⟨https://www.ei.rub.de/media/crypto/veroeffentlichungen/2011/01/21/ecpp_ches.pdf⟩ . (2001)
[13] Riedlinger, R. et al.: A 32 nm, 3.1 Billion Transistor, 12 Wide Issue Itanium Processor for Mission-Critical Servers, *IEEE Journal of Solid-state Circuit,* Vol. 47, No. 1, pp. 177–193 (2012).
[14] 白勢政明：楕円曲線暗号ハードウェアの設計法と性能評価，北陸先端科学技術大学院大学修士論文，入手先 ⟨https://dspace.jaist.ac.jp/dspace/bitstream/10119/1684/2/1970paper.pdf⟩ (2003).
[15] 高木直史：算術演算の VLSI アルゴリズム，コロナ社 (1991).
[16] Quartus II ウェブ・エディション，Altera 社，入手先 ⟨http://www.altera.co.jp/products/software/quartus-ii/web-edition/qts-we-index.html⟩
[17] Wallace, C.S.: A Suggestion for A Fast Multiplier, *IEEE Trans. on Electron. Computers* Vol. EC-13, No. 1, pp. 14–17. (1964).
[18] Wu, T. and Liu, L.-T.: Elliptic Curve Point Multiplication by Generalized Mersenne Numbers, *Journal of electronic science and technology*, Vol. 10, No. 3, pp. 199–208. (2012).