拡張ユークリッド法に基づく剰余系乗除算回路

Marcelo E. KAIHARA[†]　　高木　直史[†]

[†] 名古屋大学大学院工学研究科情報工学専攻　〒466-8603 名古屋市千種区不老町
E-mail: †{mkaihara,takagi}@takagi.nuie.nagoya-u.ac.jp

あらまし　VLSI 実現向きの剰余系乗除算回路を提案する．除算は拡張ユークリッド法，乗算は通常の乗算アルゴリズムとモンゴメリ乗算に基づく．通常の乗算は二倍の操作と加減算の繰り返しで行う．モンゴメリ乗算は新しく提案した左シフト型の手法に基づく．これらの演算はビットシフトや加減算などの単純な操作の繰り返しで行う．基数 2 の符号付きディジット表現を用いることにより全ての加減算を桁上げ伝播なしに行う．提案する剰余系乗除算回路は，規則正しいビットスライス構造を持ち，n ビットの剰余系乗算，モンゴメリ乗算及び剰余系除算を $O(n)$ クロックサイクルで実行する．クロックサイクルの長さは n に依存しない．提案する剰余系乗除算回路は，モンゴメリ法に基づく乗算回路，通常の剰余系乗算回路と剰余系除算回路を別々に設計したものより少ないハードウェア量で実装できる．

キーワード　剰余系演算、剰余乗算、剰余除算、モンゴメリアルゴリズム、拡張ユークリッドアルゴリズム、ハードウェアアルゴリズム、VLSI アルゴリズム

A Multiplier/Divider for Modular Arithmetic Based on the Extended Euclidean Algorithm

Marcelo E. KAIHARA[†] and Naofumi TAKAGI[†]

[†] Department of Information Engineering,
Nagoya University, Nagoya-shi, 464–8603 Japan
E-mail: †{mkaihara,takagi}@takagi.nuie.nagoya-u.ac.jp

Abstract　We propose a multiplier/divider for modular arithmetic suitable for VLSI realization. It is based on our newly proposed combined algorithm for modular multiplication and division. It performs modular division, Montgomery's modular multiplication and ordinary modular multiplication. Modular division is based on the extended Euclidean algorithm. Montgomery's modular multiplication is based on our newly proposed method consisting of processing the multiplier from the most significant digit first. The ordinary modular multiplication is based on conventional doubling and adding procedures. The three operations are carried out through iteration of simple operations such as shifts and addition/subtractions. The radix-2 signed-digit representation is employed so that all additions and subtractions are performed without carry propagation. The multiplier/divider for modular arithmetic has a linear array structure with a bit-slice feature and carries out n-bit modular multiplication/division in $O(n)$ clock cycles, where the length of the clock cycle is constant and independent of n. The multiplier/divider can be implemented with much smaller hardware than the necessary to implement ordinary modular multiplier, Montgomery's modular multiplier and divider separately.

Key words　modular arithmetic, modular multiplication, modular division, Montgomery's algorithm, extended Euclidean algorithm, hardware algorithm, VLSI algorithm

1. Introduction

Modular multiplication and modular division are basic operations in abstract algebra and play important roles in processing many public-key cryptosystems. For example they are used in RSA [1] and ElGamal [2] cyrptosystems.

Considering the need of PCs and mobile devices to manage several security protocols, and the great demand in technology to shrink hardware to reduce fabrication costs, it is important to develop a modular multiplier/divider that can be implemented with reduced hardware requirements.

In this paper, we propose a multiplier/divider for modular arithmetic suitable for VLSI realization. It is based on our newly proposed combined algorithm for modular arithmetic which is efficient in hardware requirements. It performs modular division, Montgomery's modular multiplication and ordinary modular multiplication. Modular division is based on the extended Euclidean algorithm. The original division algorithm [3] has been improved to reduce hardware requirements. Montgomery's modular multiplication is based on our newly proposed method consisting of processing the multiplier from the most significant digit first. This method enables to use the same hardware to perform Montgomery's modular multiplication. The ordinary modular multiplication is based on conventional doubling and adding procedures and it can also be computed with the same hardware. Hence, the three operations share almost all the same hardware components reducing considerably hardware requirements. The three operations are carried out through iterations of simple operations such as shifts and addition/subtractions.

A modular multiplier/divider based on our proposed algorithm has a linear array structure with a bit-slice feature and is suitable for VLSI . The amount of hardware of an n-bit modular multiplier/divider is proportional to n. It performs an n-bit modular multiplication/division in $O(n)$ clock cycles where the length of clock cycle is constant independent of n. Almost all the hardware components in VLSI algorithm are shared reducing considerably hardware requirements.

In order to compare hardware requirements between the proposed multiplier/divider, and a modular divider based on the algorithm proposed in [3], a Montgomery's modular multiplier based on our new method described above and an ordinary modular multiplier based on conventional doubling and adding procedures, we designed them and estimated the circuit areas. The area of the modular multiplier/divider resulted practically the same to the area of the modular divider based on the original division algorithm [3] and much smaller that the sum of areas of the ordinary modular multiplier, the Montgomery's modular multiplier and the modular divider implemented separately with delays remaining practically to the same value.

2. Preliminaries

2.1 Extended Euclidean Algorithm for Modular Division

Extended Euclidean Algorithm is an efficient way of calculating modular division. Consider the residue class field of integers with an odd prime modulus M. Let X and $Y(\neq 0)$ be elements of the field. The algorithm calculates $Z(< M)$ such that $Z \equiv X/Y \pmod{M}$ (the algorithm also works with an odd number M and Y relatively prime to M).

It performs modular division by intertwining a procedure for finding the modular quotient with that for calculating $\gcd(Y, M)$.

[Algorithm 1]
(Extended Euclidean Algorithm)

```
Function: Modular Division
Inputs: $M$: $2^{n-1} < M < 2^n$
        $X, Y$: $0 \leqq X < M$,  $0 < Y < M$
Output: $Z = X/Y \bmod M$
Algorithm:
  $\mathcal{A} := M$; $\mathcal{B} := Y$; $\mathcal{U} := 0$; $\mathcal{V} := X$;
  while $\mathcal{B} \neq 1$ do
    Choose $Q$ so that $|\mathcal{A} - \mathcal{B} \cdot Q| < |\mathcal{B}|$;
    $\mathcal{A}' := \mathcal{A} - \mathcal{B} \cdot Q$;
    Calculate $\mathcal{U}'$ which satisfies
      $\mathcal{U}' \equiv \mathcal{U} - \mathcal{V} \cdot Q \pmod{M}$ and $|\mathcal{U}'| < M$;
    $\mathcal{A} := \mathcal{B}$; $\mathcal{B} := \mathcal{A}'$;
    $\mathcal{U} := \mathcal{V}$; $\mathcal{V} := \mathcal{U}'$;
  endwhile
  if $\mathcal{B} = -1$ then $Z' := -\mathcal{V}$; else $Z' := \mathcal{V}$; endif
  if $Z' < 0$ then $Z := Z' + M$ else $Z := Z'$; endif
  output $Z$ as the result;
```

$\mathcal{A}$ ($\mathcal{A}'$) and $\mathcal{B}$ are involved in the calculation of GCD and are allowed to be negative. $\mathcal{U}$ ($\mathcal{U}'$) and $\mathcal{V}$ are used in the algorithm for calculating the quotient and are also allowed to be negative. $\mathcal{V} \times Y \equiv \mathcal{B} \times X \pmod{M}$ always holds. Since the final $\mathcal{B}$ satisfies $|\mathcal{B}| = 1$, $Z' \times Y \equiv X \pmod{M}$. Since $|\mathcal{V}| < M$ always holds, $-M < Z' < M$. Therefore, $Z \times Y \equiv X \pmod{M}$ and $0 < Z < M$ hold. Namely, Z is the quotient of X/Y modulo M.

2.2 Modular Multiplication

2.2.1 Ordinary Modular Multiplication

Consider the residue class ring of integers with an odd modulus M. Let X and Y be elements of the ring. Modular multiplication is defined as finding Z such that $0 \leqq Z < M$ and $Z \equiv XY \pmod{M}$. In ordinary modular multiplication operation, the digits of the multiplier are scanned from the most significant position. For each digit that is processed, the partial product is doubled. If the scanned digit has the value of one, the multiplicand is then added to the partial product, otherwise, none is done. The multiplier is then shifted one position to the left to allow the next digit to be scanned. The multiplication algorithm is described below. Note that A is n-digits long and the most significant one is represented as a_{n-1}.

[Algorithm 2]
(Modular Multiplication Algorithm)

```
Function: Modular Multiplication
Inputs: $M : 2^{n-1} < M < 2^n$
        $X, Y : 0 \leqq X, Y < M$
Output: $Z = XY \bmod M$
Algorithm:
  $A := Y$; $U := 0$; $V := X$;
  for $i := 1$ to $n$ do
    $U := 2 \cdot U \bmod M$;
    $q := a_{n-1}$;
    $A := A << 1$; $U := (U + qV) \bmod M$;
  end for
  $Z := U$;
  output $Z$ as the result;
```

2.2.2 Montgomery's Modular Multiplication

Montgomery introduced an efficient algorithm for calculating modular multiplication [4]. Consider the residue class ring of integers with an odd modulus M. Let X and Y be elements of the ring. Montgomery's modular multiplication algorithm calculates $Z(< M)$ such that $Z \equiv XYW^{-1} \pmod{M}$ where W is an arbitrary constant relatively prime to M. The value of W is usually set to 2^n when the calculations are performed in radix-2 with an n-bit modulus M.

The radix-2 Montgomery's multiplication algorithm is described below.

[Algorithm 3]
(Montgomery's Multiplication Algorithm)

```
Function: Montgomery's Modular Multiplication
Inputs: $M : 2^{n-1} < M < 2^n$
        $X, Y : 0 \leqq X, Y < M$
```

```
Output: $Z = XY2^{-n} \bmod M$
Algorithm:
  $A := Y$; $U := 0$; $V := X$;
  for $i := 1$ to $n$ do
    if $A \bmod 2 = 0$ then $q := 0$;
    else $q := 1$; end if
    $A := (A - q)/2$; $U := (U + qV)/2 \bmod M$;
  end for
  if $U \geqq M$ then $Z := U - M$;
  else $Z := U$; end if
  output $Z$ as the result;
```

Note that U is always bounded by $2M$ throughout all iterations. Therefore, the last correction step assures that the output is correctly expressed in modulo M.

3. Basic Modular Operations in RB System

In the hardware algorithm on which we base for constructing the modular multiplier/divider, the inputs operands X and Y, as well as the output result Z are represented in radix-2 signed-digit (SD2) integers [5]. X and Y are assumed to be n-digit. The output result Z is assumed to be $(n+1)$-digit integer. Both the inputs and the output are in the range $[-M+1, M-1]$. Intermediate results are also represented in the SD2 representation.

The algorithm requires a doubling procedure for a SD2 integer without overflow. Let A and B be n-digit SD2 integers. The doubling procedure is performed only when A satisfies $a_{n-1} = 0$ or $a_{n-2} = -a_{n-1}$. A doubling $B := DBL(A)$, i.e., the calculation of B such that $B = 2 \cdot A$ is performed as follows. When $a_{n-1} = 0$, $B = [a_{n-2}a_{n-3}a_{n-4} \cdots a_1a_0]$ and otherwise $(a_{n-2} = -a_{n-1})$, $B = [a_{n-1}a_{n-3}a_{n-4} \cdots a_1a_0]$.

Procedures for addition, doubling and halving modulo M in the SD2 system are also required and are described next.

Let the modulus M $(= [1m_{n-2} \cdots m_1 1])$ be an n-bit binary odd integer satisfying $2^{n-1} < M < 2^n$. Let U, V and T be $(n+1)$-digit SD2 integers satisfying $-M < U, V, T < M$. A modular addition $T := MADD\ (U, V, M)$, i.e., the calculation of T such that $T \equiv U + V \pmod{M}$, is performed through two steps. In the first step, we calculate $S := U + V$ in the SD2 system. S is an $(n+2)$-digit SD2 number. In the second step, we add M or 0 or M' to S, accordingly as the value of the number formed by the three most significant digits of S, i.e. the value of $[s_{n+1}s_ns_{n-1}]$, is negative or zero or positive. $M' = [\bar{1}0m'_{n-2}...m'_1 1]$ is a $(n+1)$-digit SD2 number where m'_i is 1 or 0 accordingly as m_i is 0 or 1, and has the value $-M$. This addition is also performed in the SD2 system. Since all the digits of the addend are non-negative except the most significant one, the addition in this step is simpler. For the details of the modular addition procedure, see [6].

Modular doubling $T := MDBL\ (U, M)$, i.e., the calculation of T such that $T \equiv 2 \cdot U \pmod{M}$, can be performed by applying the second step of the modular addition to $2 \cdot U$, which is obtained by shifting U by one position to the left.

Modular halving $T := MHLV(V, M)$, i.e., the calculation of T such that $T \equiv V/2 \pmod{M}$, is performed through two steps. In the first step, we add M to V when V is odd, i.e. when $v_0 \neq 0$. Nothing is performed when V is even. In the second step, we shift the result of the first step by one position to the right throwing away the least significant digit, which is 0. (Recall that M is odd.)

Procedures $MADD$, $MDBL$ and $MHLV$ can be performed in a constant time independent of n by means of combinational circuits.

4. A VLSI Algorithm for Modular Multiplication/Division

The hardware algorithm is presented here. It is divided in 4 steps. Initialization of variables takes place in Step 1. The core of the algorithm is described in Step 2. A correction is performed in Step 3, and in Step 4 the output result is selected. The algorithm has three modes of operation. In mode=1, the algorithm performs modular division. In mode=2, it performs Montgomery's modular multiplication. Finally, in mode=3, it performs ordinary modular multiplication.

[Algorithm 4]
(A VLSI Algorithm for modular multiplication and modular division)

```
Function: Modular Multiplication and
          Modular Division
Inputs: $M : 2^{n-1} < M < 2^n$
        $X, Y : -M < X, Y < M$
Output: $mode = 0 : Z \equiv X/Y \bmod M$
        $mode = 1 : Z \equiv XY2^{-n} \bmod M$
        $mode = 2 : Z \equiv XY \bmod M$
Algorithm:
  Step 1:
    $A := Y$; $B := M$; $P := 1$; $M := M$;
    if $mode = 0$ then
      $D := 1$; $U := X$; $V := 0$;
    else
      $D := 2^n$; $U := 0$; $V := X$;
      goto Step 2-4;
    endif
  Step 2:
    Step 2-0:
      if $[a_{n-1}a_{n-2}] = [11]$ or $[\bar{1}\bar{1}]$ then
        $q := a_{n-1} \cdot b_{n-1}$;
        $A := A - q \cdot B$;
        $U := MADD(U, -q \cdot V, M)$;
      endif
    Step 2-1:
      while $p_{n-1} = 0$ and
        $[a_{n-1}a_{n-2}] \neq [10]$ and $[a_{n-1}a_{n-2}] \neq [\bar{1}0]$;
        if $[a_{n-1}a_{n-2}a_{n-3}] = [1\bar{1}1]$ or
          $[a_{n-1}a_{n-2}a_{n-3}] = [011]$ then
          $[a_{n-1}a_{n-2}a_{n-3}] := [10\bar{1}]$;
        elseif $[a_{n-1}a_{n-2}a_{n-3}] = [\bar{1}1\bar{1}]$ or
          $[a_{n-1}a_{n-2}a_{n-3}] = [0\bar{1}\bar{1}]$ then
          $[a_{n-1}a_{n-2}a_{n-3}] := [\bar{1}01]$;
        else
          $A := DBL(A)$; $P := 2 \cdot P$; $D := 2 \cdot D$;
          $U := MDBL(U, M)$;
        endif
      endwhile
```

```
Step 2-2:
  $T := A$; $A := B$; $B := T$;
  $T := U$; $U := V$; $V := T$;
Step 2-3:
  if $p_{n-1} = 1$ then
    while $d_0 = 0$ do
      $D := D/2$; $V := MHLV(V, M)$;
    endwhile
    goto Step 3;
  endif
Step 2-4:
  / * Main Stage (MUL/DIV) * /
  while $d_0 = 0$ do
    if $a_{n-1} = 0$ or $a_{n-2} = -a_{n-1}$ then
      $S := A$;
    else
      if mode $= 0$ then
        $q := a_{n-1} \cdot b_{n-1}$;
        $S := A - q \cdot B$;
      else
        $q := -a_{n-1}$;
        $S := A$;
        if mode $= 1$ then $s_{n-1} := 0$; endif
      endif
      $U := MADD(U, -q \cdot V, M)$;
    endif
    if ($s_{n-1} = 0$ or $s_{n-2} = -s_{n-1}$) then
      $A := DBL(S)$; $D := D/2$;
      if mode $= 2$ and $d_0 = 0$ then
        $U := MDBL(U, M)$;
      else
        $V := MHLV(V, M)$;
      endif
    else
      if mode $= 2$ then $s_{n-1} := 0$; endif
      $A := S$;
    endif
  endwhile
  if mode $\neq 0$ then goto Step 4;
  / * Termination Stage (DIV) * /
  $r := sgn([a_{n-1}a_{n-2}])$;
  while $sgn([a_{n-1}a_{n-2}] = r$ and
    ($abs([a_{n-1}a_{n-2}a_{n-3}]) \geq 3$ or
      ($b_{n-3} = -b_{n-1}$ and
      $abs([a_{n-1}a_{n-2}a_{n-3}] = 2)$))
  do
    $q := r \cdot b_{n-1}$;
    $A := A - q \cdot B$;
    $U := MADD(U, -q \cdot V, M)$;
  endwhile
  goto Step 2-1;
Step 3:
  if $b_{n-1} = \bar{1}$ then $V := -V$; endif
Step 4:
  if mode $= 0$ then $Z := V$;
  else $Z := U$; endif
  output $Z$ as the result;
```

4.1 Division Mode

In division mode, $\mathcal{A}$ and $\mathcal{B}$ are represented by two n-digit RB integers, $A(=[a_{n-1}a_{n-2}\cdots a_0]))$ and $B(=[b_{n-1}b_{n-2}\cdots b_0]))$, and two n-bit binary integers of the form 2^i, $P(=[p_{n-1}p_{n-2}\cdots p_0])$ and $D(=[d_n d_{n-1} d_{n-2}\cdots d_0])$, so that $\mathcal{A} = A/(P/D)$ and $\mathcal{B} = B/P$. A, B, U and V are initialized to the values of Y, M, X and 0 respectively. At the beginning of each iteration of Step 2, $D = 1$ and $b_{n-1} \neq 0$. $A = \mathcal{A} \cdot 2^{n-k}$, $B = \mathcal{B} \cdot 2^{n-k}$ and $P = 2^{n-k}$. $U = \mathcal{U}$ and $V = \mathcal{V}$ when $\mathcal{B}$ is effectively k-bit long.

At each iteration of Step 2, we first strongly normalize A which stores the divisor during Step 2-1. A is strongly normalized if $a_{n-1} \neq 0$ and $a_{n-2} = 0$, i.e., $[a_{n-1}a_{n-2}] = [10]$ or $[\bar{1}0]$. During the normalization, A is doubled several times by means of DBL shown in Section 3. At the same time that A is doubled, D and P are also doubled, and U is doubled modulo M by means of $MDBL$ shown in Section 3. In Step 2-2, A and B, and U and V are swapped respectively. At this time, $A = \mathcal{A} \cdot (P/D)$, $B = \mathcal{B} \cdot P$ and $V \equiv \mathcal{V} \cdot D \pmod{M}$.

Then, we perform an integer division. We produce Q as a sequence of q's where $q \in \{-1, 1\}$ and $Q = \sum q \cdot D$. Note that D is updated during the integer division. As the production of q, we calculate $A - q \cdot B$ in the SD2 system without carry propagation ($A - q \cdot B$ corresponds to $\mathcal{A} - (q \cdot D) \cdot \mathcal{B}$). Concurrently, we calculate $U - q \cdot V$ modulo M in the SD2 system by means of $MADD$ shown in Section 3. During the integer division, we double A several times by means of DBL. When we double A, we halve D, and halve V modulo M by means of $MHLV$ shown in Section 3. After the integer division, $D = 1$, $A = \mathcal{A}' \cdot P$, $B = \mathcal{B} \cdot P$ ($b_{n-1} \neq 0$), $U = \mathcal{U}'$ and $V = \mathcal{V}$.

$sgn([a_{n-1}a_{n-2}])$ is -1 or 0 or 1, accordingly as the value of $[a_{n-1}a_{n-2}]$ is negative or zero or positive. $abs([a_{n-1}a_{n-2}a_{n-3}])$ means $|4a_{n-1} + 2a_{n-2} + a_{n-3}|$.

Step 2-0 is executed only when the most significant two bits of A, i.e., the divisor Y, have both the value of 1. Note that when its most significant two bits have both value of 1, A cannot be strongly normalized. We perform the subtraction of $A - B$ in the SD2 system using an ordinary SD2 addition rule, e.g., [6]. Since the most significant two bits of B, i.e., the modulus M, have also the value of 1 in this case, the most significant digit of the updated A has as result the value of 0.

In Step 2-1, we strongly normalize A. At the beginning of this stage, $[a_{n-1}a_{n-2}] \neq [11]$ nor $[\bar{1}\bar{1}]$, from Step 2-0 and the SD2 addition rule in the termination stage shown below. It never become [11] nor $[\bar{1}\bar{1}]$ during this stage, because we rewrite $[1\bar{1}1]$ and $[011]$ to $[10\bar{1}]$ and $[\bar{1}1\bar{1}]$ and $[0\bar{1}\bar{1}]$ to $[\bar{1}01]$. In Step 2-2, we swap A and B and also U and V. In Step 2-3, when p_{n-1} becomes 1, i.e., P becomes 2^{n-1}, it means that $|\mathcal{B}|$ has become 1. Then, we divide V by D modulo M by means of $MHLV$, because V has been multiplied by D modulo M. Then we terminate Step 2.

In Step 2-2, we perform an integer division, a Montgomery's modular multiplication or an ordinary modular multiplication. During the integer division, we perform the subtraction of $A - q \cdot B$ in the SD2 system. In this subtraction, we use the special addition rule shown in Table 1 at the most significant two positions. Note that when an addition is performed, $[a_{n-1}a_{n-2}]$ is not [00]. Furthermore, in the main stage, when an addition is performed, $[a_{n-1}a_{n-2}]$ is not $[\bar{1}1]$

Table 1 A special SD2 addition rule

$s_{n-1}s_{n-2}$

	$a_{n-1}a_{n-2}$					
c_{n-3}	$\bar{1}\bar{1}$	$\bar{1}0$	$\bar{1}1,0\bar{1}$	$01,1\bar{1}$	10	11
$\bar{1}$	$\bar{1}0$	$0\bar{1}$	00	$\bar{1}0$	$0\bar{1}$	00
0	$0\bar{1}$	00	01	$0\bar{1}$	00	01
1	00	01	10	00	01	10

nor $[0\bar{1}]$ nor $[01]$ nor $[1\bar{1}]$. (The rule for these cases in the table is for the addition in the termination stage.) Note also that $[b_{n-1}b_{n-2}]$ is $[10]$ or $[\bar{1}0]$ and that $q = sgn([a_{n-1}a_{n-2}]) \cdot b_{n-1}$ ($= an - 1 \cdot b_{n-1}$ in the main stage). In the table, c_{n-3} is the intermediate carry from the third significant position. We use an ordinary SD2 addition rule, e.g., that in [6], at the other positions. Note that $[s_{n-1}s_{n-2}]$ never become $[11]$ nor $[\bar{1}\bar{1}]$.

Because of the strongly normalization of A, the computation in this step is simple. We can show that in the main stage, no two successive SD2 additions are performed without doubling A ($DBL(S)$).

In the termination stage, we use a bit complicated condition for termination, in order to avoid the situation that the final $|A|$ ($|A'|$) is very near to $|B|$ ($|B|$). Note that this situation makes the convergence of the whole computation very slow. By the use of the complicated condition, we can make the final $|A|$ significantly smaller than $|B|$. Hence, we can guarantee that when A is not doubled in an execution of Step 2-1, the updated A must be doubled in the next execution of Step 2-1. We can show that no more than three SD2 additions are performed in the termination stage. Note that the final $[a_{n-1}a_{n-2}]$ is not $[11]$ nor $[\bar{1}\bar{1}]$, from the SD2 addition rule.

In Step 3, when $b_{n-1} = \bar{1}$, we negate V in the SD2 system.

In Step 4, we select the output depending on the mode of operation.

In applications where the output needs to be fed back into the inputs, the output can be transformed into an n-digit SD2 number subtracting M to the result when z_n has the value of 1, and adding M when z_n has the value of $\bar{1}$.

4.2 Montgomery's Modular Multiplication Mode

In Montgomery's modular multiplication mode, A and V are initialized to the values of the multiplier Y and the multiplicand X respectively. U is used to store the partial product of the multiplication.

In Section 2.2.2, we described Algorithm 3 which performs Montgomery's modular multiplication. It examines the least significant bit of A to determine whether to add or not V to U. The result is then divided by 2 using modular arithmetic and A is shifted one position to the left.

In order to implement Montgomery's multiplication operation using the same hardware components required by the division mode, we introduce SD2 representation in operands, internal calculation and the output result and we examine the multiplier from the most significant position first, i.e. a_{n-1}. For a nonzero value of the digit, we add or subtract the multiplicand X stored in V to U depending on the sign of it. If it is positive, we add. Otherwise we subtract. Then, instead of dividing the partial product U by 2 modulo M, we divide the multiplicand V by 2 modulo M. For this operation we use $MHLV(V, M)$ described in Section 3. If the value of the digit a_{n-1} is 0, we perform a $DBL(A)$ to shift to the left the multiplier and $MHLV(V, M)$ to divide the multiplicand by 2 modulo M. The same operations are performed for the cases that $[a_{n-1}a_{n-2}] = 1\bar{1}$ or $\bar{1}1$ since they represent a 0 at the most significant digit of A. The 'for' loop is implemented using variable D which is initialized with 2^n and it is shifted to the right at each iteration step until it is equal to 1. Montomery's constant results in this case equal to the value of 2^{n-1}.

4.3 Ordinary Modular Multiplication Mode

We also implement the ordinary modular multiplication described in Section 2.2.1. We also perform the operations in SD2 representation. Since variable U, which stores the partial product, is initialized with 0, instead of doubling U and then adding the multiplicand, we proceed in the reverse order. For the case that $a_{n-1} = 0$ or $[a_{n-1}a_{n-2}] = 1\bar{1}$ or $1\bar{1}$, we just double A by means of $DBL(A)$ and U by $MDBL(U, M)$. For the cases that $[a_{n-1}a_{n-2}] = 11$ or 10 or $\bar{1}\bar{1}$ or $\bar{1}0$, in order to share hardware components without increasing the calculation delay, we split the calculation of $MDBL(U, M)$ and $MADD(U, -q \cdot V, M)$ into two different iteration steps. We first perform $U := MADD(U, -q \cdot V, M)$ depending on the value of a_{n-1} and we leave the most significant position of A in 0. By doing so, A is shifted to the left by $DBL(A)$ and U is doubled modulo M by $MDBL(U, M)$ in the next iteration. Since we reverse the order of doubling the partial product and adding the multiplicand, $MDBL(U, M)$ is not performed in the last iteration.

5. A Multiplier/Divider for Modular Arithmetic

An n-bit modular multiplier/divider based on Algorithm 4 consists of seven registers for storing A, B, P, D, U, V and M, and a combinational circuit part.

We assume that we perform each of Step 2-0, or one iteration of Step 2-1, or one iteration of Step 2-2, or one iteration of Step 2-3, or one iteration of main or termination stage of Step 2-4, or Step 3 in one clock cycle. Then, in one clock cycle, the modular divider mainly performs a SD2 addition of $A - B$ and $MADD(U, -V, M)$ in Step 2-0, or $DBL(A)$, one-bit-shifts of P and D and $MDBL(U, M)$, or swaps of A and B, and U and V, in Step 2-2, or a one-bit-shift of D and $MHLV(V, M)$ in Step 2-3, or a $DBL(A)$ and $MDBL(U, M)$ or $MHLV(V, M)$ and a one-bit-shift of D or a SD2 addition of $A - q \cdot B$ or a reset of the most significant digit of A and $MADD(U, -q \cdot V, M)$, and a possible $DBL(S)$, a one-bit-shift of D and $MHLV(V, M)$, in Step 2-4, or a negation of V in the SD2 system, in Step 3.

The combinational circuit part of the divider (for Steps 2 and 3) mainly consists of an SD2 adder (with an operand negator), a modular adder (with an operand negator), a modular doubling, a modular halving circuit, an SD2 negator and selectors. The modular adder consists of two SD2 adders one of which is simpler. The modular doubling and the modular halving circuit consist of simpler SD2 adders. These circuits

Table 2 The number of cells, area, and delay of a multiplier/divider, a divider, a Montgomery's multiplier and an ordinary multiplier

n	circuit	#cells	delay[ns]	area[mm^2]
128	MON/MUL/DIV	17462	6.69	2.753220
	DIV	17748	6.69	2.740202
	MON	9515	6.67	1.484740
	MUL	7892	6.69	1.242536
256	MON/MUL/DIV	34130	6.69	5.598141
	DIV	33816	6.69	5.539039
	MON	19573	6.69	3.093940
	MUL	15717	6.67	2.552651
512	MON/MUL/DIV	69117	6.69	11.140568
	DIV	70966	6.69	11.353540
	MON	42263	6.69	6.801672
	MUL	33485	6.69	5.485547

have bit-slice structure.

The depth of the combinational circuit part is a constant independent of n, and therefore, the length of the clock cycle is constant independent of n.

The modular divider has a bit-slice structure and is suitable for VLSI implementation. The amount of hardware of the modular multiplier/divider is proportional to n.

In division mode, from the discussion in the previous section, we can show that in Step 2-4, no two successive clock cycles are executed without doubling A ($DBL(S)$) in the main stage, and no more than three cycles are executed in the termination stage. We can also show that if $DBL(A)$ is not performed in an execution of Step 2-1 (normalization of A), $DBL(A)$ must be performed in the next execution of Step 2-1. Hence, we can show that the number of clock cycles executed in Step 2 is at least $2n$ and at most about $3n$. It varies with the operands.

Montgomery's multiplication is performed in Step 2-4 in exactly n clock cycles. Ordinary modular multiplication is performed in at least n and at most $2n$ clock cycles. It varies with the multiplier.

6. Experimental Results

We have designed a modular multiplier/divider, as well as a modular divider based on the algorithm proposed in [3], an ordinary modular multiplier, a Montgomery's modular multiplier separately. We used Verilog HDL and synthesized them by Synopsys Design Compiler using 0.35μm CMOS 3-metal technology provided by VLSI Design and Education Center(VDEC), the University of Tokyo, with the collaboration by Rohm Corporation. Table 2 shows the number of cells, area, and delay of the described circuits for $n = 128$, 256 and 512. As show on the table, area of the multiplier/divider is similar to the area of the modular divider based on [3] and it is much less than the sum of areas of the ordinary modular multiplier, Montgomery's modular multiplier and the modular divider with the delays remaining practically to the same value.

7. Conclusion

We have proposed a modulo M multiplier/divider that has large part of the circuit components shared by the three operations. The circuit has a linear array structure with a bit-slice feature, and is suitable for VLSI implementation. The amount of hardware of an n-bit modular multiplier/divider is proportional to n. It performs an n-bit modular multiplication/division in $O(n)$ clock cycles, where the length of clock cycle is constant independent of n. The area of the modular multiplier/divider resulted much smaller then the sum of areas of the ordinary modular multiplier, Montgomery's modular multiplier and the modular divider implemented separately. Delays remained practically to the same value.

References

[1] R. L. Rivest, A. Shamir, and L. Adleman, 'A method for obtaining digital signatures and public-key cryptosystems,' *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.

[2] T. ElGamal, 'A public key cryptosystem and a signature scheme based on discrete logarithms,' *IEEE Trans. Information Theory*, vol. IT-31, no. 4, pp. 469–472, July 1985. Nov. 1976.

[3] N. Takagi, 'A hardware algorithm for modular division based on the extended Euclidean algorithm,' *IEICE Trans. Information and Systems*, vol. E79-D, no. 11, pp. 1518–1522, Nov. 1996.

[4] P. L. Montgomery, 'Modular Multiplication without Trial Division' *Mathematics of Computation*, vol. 44, no. 170, pp. 519–521, Apr. 1985.

[5] N. Takagi, H. Yasuura and S. Yajima, 'High-speed VLSI multiplication algorithm with a redundant binary addition tree,' *IEEE Trans. Computers*, vol. C-34, no. 9, pp. 789–796, Sep. 1985.

[6] N. Takagi and S. Yajima, 'Modular multiplication hardware algorithms with a redundant representation and their application to RSA cryptosystem,' *IEEE Trans. Computers*, vol. 41, no. 7, pp. 887–891, July 1992.