Pipelined Bipartite Modular Multiplication

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

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

Abstract We propose a fast hardware algorithm for calculating modular multiplication. This algorithm is based on the recently proposed bipartite modular multiplication method. The calculation of the modular multiplication is performed in the KT-residue system which enables the splitting of the multiplier into two parts which can then be processed in parallel. In the hardware algorithm that we propose, the addition of the partial products to the intermediate accumulated product is pipelined in order to reduce the critical path delay. A radix-4 version of the hardware algorithm is given and its hardware implementation is discussed.

Key words modular arithmetic, modular multiplication, Montgomery algorithm, interleaved modular multiplication, bipartite modular multiplication.

1. Introduction

Most of the public-key cryptosystems such as RSA [8], ElGamal [2], and others, are based on operations in modular arithmetic. One of the operations that are heavily used is the modular multiplication of large integers. Because of its computational intensity, many efforts have been directed towards developing fast algorithms and hardware implementations to speed up this operation.

There are two well known approaches for calculating the modular multiplication. One is based on the interleaved modular multiplication algorithm where the multiplier is processed from the most significant position [1], [9]. The other one is based on the Montgomery algorithm where the multiplier is processed from the least significant position [6], [7], [11].

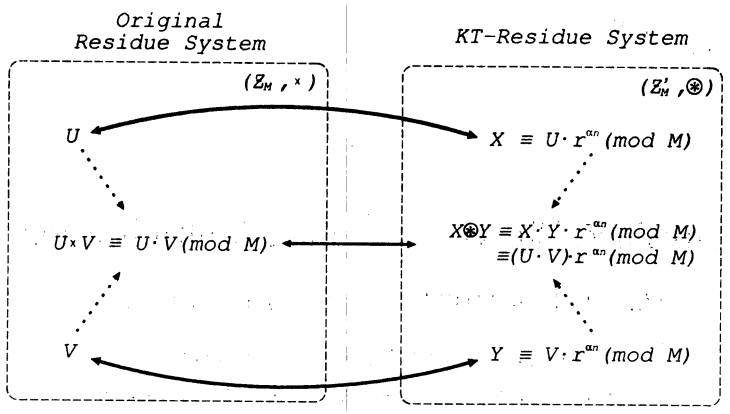
We recently proposed a method called bipartite modular multiplication (BMM) which combines the two previous methods to further speed up modular multiplication [4]. The calculation is performed in the KT-residue system which enables the splitting of the multiplier into two parts. These two parts can then be processed in parallel. One part of the split multiplier can be processed using the interleaved modular multiplication algorithm while the other part can be processed using the Montgomery algorithm. When the performance of these two algorithms are similar and the multiplier is split into two parts of equal size, this method theoretically doubles the speed compared to either the interleaved modular multiplication algorithm or the Montgomery algorithm when applied to unsplit operands.

This paper proposes a new hardware algorithm based on the BMM method which enables the pipelining of the intermediate calculations in a natural way. This is accomplished by applying the shift operations, the modular reduction and the Montgomery reduction to the multiplicand instead of applying them to the intermediate accumulated product. A radix-4 version of the algorithm that can be implemented with similar performance and with less hardware requirements to that of a pipelined radix-16 Montgomery multiplier [7] is presented and discussed.

The remainder of this paper is organized as follows. Section 2 reviews the BMM method. Section 3 introduces the new hardware algorithm and explains its hardware implementation. A radix-4 version of the hardware algorithm is given as an example and its hardware implementation is discussed in Section 4. Section 5 contains our concluding remarks.

2. Bipartite Modular Multiplication

The recently proposed BMM method [4] enables the splitting of the multiplier into two parts which can then be processed in parallel reducing the time complexity. We consider the modulus M to be an n-digit integer in radix-r so that $r^{n-1} < M < r^n$ and $\gcd(M,r) = 1$. If U is an integer in



Original Residue System
$(\mathcal{Z}_M, \times)$
U
$U \times V \equiv U \cdot V \pmod M$
V
KT-Residue System
$(\mathcal{Z}'_M, \circledast)$
$X \equiv U \cdot r^{\alpha n} \pmod M$
$X \circledast Y \equiv X \cdot Y \cdot r^{-\alpha n} \pmod M$
$\equiv (U \cdot V) \cdot r^{\alpha n} \pmod M$
$Y \equiv V \cdot r^{\alpha n} \pmod M$

Fig. 1 Mapping between the original residue system and the KT-residue system

the range $[0, M-1]$, the image, or the KT-residue of U is defined as $X = U \cdot r^{\alpha n} \mod M$ where α is a rational number so that $0 < \alpha < 1$ and αn is an integer. The set of these images forms a complete residue system called KT-residue system. Given two images X and Y, multiplication modulo M in the KT-residue system is defined as:

$$X \circledast Y \triangleq X \cdot Y \cdot r^{-\alpha n} \mod M$$

The existence of $r^{-\alpha n} \mod M$ is assured by the relative primalty condition between $r^{\alpha n}$ and M. Since M is odd for cryptographic applications, by utilising $r = 2^k$, the primalty condition is satisfied.

The transformation from the original representation to the new residue system can be performed by calculating the ordinary modular multiplication between the integer value and the constant $r^{\alpha n}$. The inverse transformation from the new residue system back to the original representation can be calculated by performing a modular multiplication between an image and the constant $r^{-\alpha n}$ which can be calculated using the Montgomery algorithm.

Isomorphism between the original integer set $\mathcal{Z}_{\mathcal{M}}$ with the ordinary modular multiplication denoted with the symbol $\times$, and the KT-residue system $\mathcal{Z}'_{\mathcal{M}}$ with the operation $\circledast$, holds as illustrated in Fig. 1. In the figure, X and Y are the images of U and V respectively.

The advantage of using the described representation becomes evident if we consider the multiplier Y split into two parts Y_H and Y_L so that $Y = Y_H \cdot r^{\alpha n} + Y_L$. Then, the multiplication modulo M in the KT-residue system of the images X and Y can be efficiently computed using the following identity:

$$X \circledast Y = (X \cdot Y_H \mod M + X \cdot Y_L \cdot r^{-\alpha n} \mod M) \mod M$$

The term $X \cdot Y_H \mod M$ can be calculated using the interleaved modular multiplication algorithm while the term $X \cdot Y_L \cdot r^{-\alpha n} \mod M$ can be calculated using the Montgomery algorithm. Since the split operands Y_H and Y_L are shorter in length than Y, the calculations $X \cdot Y_H \mod M$ and $X \cdot Y_L \cdot r^{-\alpha n} \mod M$ are performed faster than either calculating the ordinary modular multiplication or the Montgomery multiplication with the unsplit operands. When these two calculations can be performed with similar performance and the multiplier is split into two parts of equal size, the above equation shows that it is theoretically possible to achieve a maximum speed of twice that of either of these two algorithms when applied to unsplit operands. Furthermore, compared to the Montgomery method, conversion speed between the original integer set and the KT-residue system is potentially doubled, and precomputation of constants is no longer necessary. It is also worth noticing that any technique to speeds up the calculation of $X \cdot Y_H \mod M$ and $X \cdot Y_L \cdot r^{-\alpha n} \mod M$ can be used independently in this method.

3. Pipelined Bipartite Modular Multiplication

In this section we present a new hardware algorithm for modular multiplication based on the BMM method. The feature of this hardware algorithm is that it enables the pipelining of the intermediate calculations reducing the clock cycle time. This is accomplished by initially storing copies of the multiplicand into two variables and performing a shift operation and a modular reduction on one variable and a shift operation and a Montgomery reduction on the other variable instead of applying a shift operation and either of the reductions to the intermediate accumulated product. For simplicity, we describe the algorithm for the parameter $\alpha = 1/2$. We assume hereafter that n is an even number. When n is odd, the algorithm that we will describe works by concatenating a 0 digit on the most significant position of the multiplier.

In order to enable the pipelining of the intermediate operations, we need two variables of n digits of length, L and R, which are initialized to the value of the multiplicand X. Then, at each iteration, L is shifted to the left by one digit position and reduced modulo M. Similarly, R is shifted to the right by one digit position and reduced modulo M using the Montgomery reduction. We also need two variables A and B for storing the split two parts of the multiplier. A and B initially store Y_H and Y_L respectively. Then, L and A are used for generating the partial products of $X \cdot Y_H \bmod M$, whereas R and B are used for generating the partial products of $X \cdot Y_L \cdot r^{-\frac{n}{2}} \bmod M$. A and B have $\frac{n}{2}+1$ digits of length. The i-th digit ($i = 0, 1, \cdots, \frac{n}{2}$) of A is denoted by a_i. Namely, $A = \sum_{i=0}^{\frac{n}{2}} a_i \cdot r^i$. Similarly, $B = \sum_{i=0}^{\frac{n}{2}} b_i \cdot r^i$. The digits of A are scanned from the least significant position while in B, the digits are scanned from the most significant position. The scanning procedures can be implemented using shift operations. We use a variable C of n digits of length to store the result of the addition between the generated partial products. A variable D of n digits of length is used to store the intermediate accumulated product.

Below is the radix-r pipelined hardware algorithm based on the BMM method. In the algorithm, $\{0, Y_H\}$ and $\{0, Y_L\}$ mean that a 0 digit has been concatenated to the most significant position of Y_H and Y_L respectively.

[Hardware Algorithm PBMM]
(Pipelined Bipartite Modular Multiplication)

```
Inputs: $M : r^{n-1} < M < r^n$, $\gcd(M, r) = 1$ and $r = 2^k$
        $X, Y : 0 \leq X, Y < M$
Output: $Z = X \cdot Y \cdot r^{-\frac{n}{2}} \bmod M$
Algorithm:
  Step 1: $L_0 := X$; $R_0 := X$; $M := M$;
          $A := \{0, Y_H\}$; $B := \{0, Y_L\}$;
          /* $Y = Y_H \cdot r^{\frac{n}{2}} + Y_L$ */
  Step 2:
  Step 2-1: $D_0 := 0$;
            $T_1 : L_1 := L_0 \cdot r \bmod M$;
                   $R_1 := R_0 / r \bmod M$;
                   $C_0 := (a_0 \cdot L_0 + b_{\frac{n}{2}} \cdot R_0) \bmod M$
  Step 2-2: for $j := 1$ to $\frac{n}{2}$ do
            $T_1 : L_{j+1} := L_j \cdot r \bmod M$;
                   $R_{j+1} := R_j / r \bmod M$;
                   $C_j := (a_j \cdot L_j +$
                           $b_{\frac{n}{2}-j} \cdot R_j) \bmod M$;
            $T_2 : D_j := (C_{j-1} + D_{j-1}) \bmod M$;
            endfor
  Step 2-3: $T_2 : D_{\frac{n}{2}+1} := (C_{\frac{n}{2}} + D_{\frac{n}{2}}) \bmod M$;
  Step 3: $Z := D_{\frac{n}{2}+1}$;
```

In Step 1, initialization of variables takes place. In Step 2, we process the upper part of the multiplier, i.e. A, from the least significant position and the lower part of the multiplier, i.e. B, from the most significant position. In contrast to the interleaved modular multiplication algorithm and the digit-serial Montgomery multiplication algorithm where the intermediate accumulated product is shifted and reduced modulo M at each iteration, we shift the variables that initially stores the multiplicand instead. In Step 2, two atomic tasks are performed in parallel: Task T_1 executes the shifting of the variable L by one digit position towards the most significant position and the modular reduction on this shifted value. It also executes the shifting of the variable R by one digit position towards the least significant position and the Montgomery reduction on this shifted value. This task also generates the partial products, adds them and applies the modular reduction to this added value. The result is then placed into variable C. Task T_2 executes the addition of the previous result of C to the value of the intermediate accumulated product stored in variable D. Then, modular reduction is applied to the result of this addition and the result is placed into D. One step, i.e. Step 2-1, is required until the two tasks are executed fully in parallel. One extra step, i.e. Step 2-3, is necessary for obtaining the final result placed into D. Step 2-2 is the core of the algorithm. In this step, the two tasks are executed in parallel. The dependency graph for the tasks in Step 2 is depicted in Fig. 2.

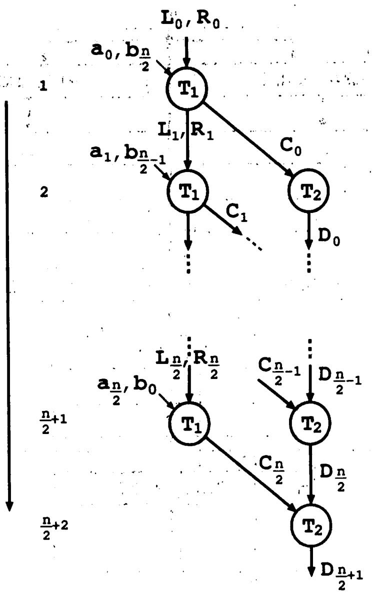


Fig. 2 Dependency graph of the tasks in Step 2 of Hardware Algorithm PBMM

In Step 3, the result is obtained from the variable D. If we implement the hardware algorithm so that all the operations contained in one single iteration of each step are executed in one clock cycle, a modular multiplication is calculated in $\frac{n}{2}+4$ clock cycles.

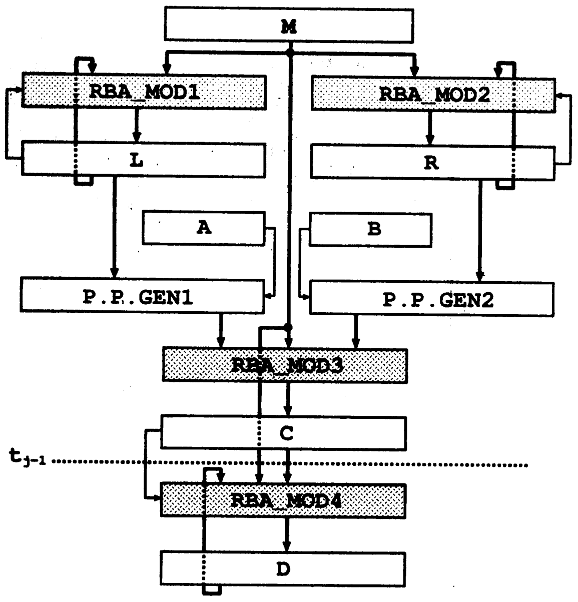


Fig. 3 Block diagram of a PBMM multiplier

A modular multiplier based on the new hardware algorithm consists of seven registers and four adders. The rest of the components are multiplexers. A block diagram of a radix-r digit multiplier based on this hardware algorithm is given in Fig. 3.

Compared to the conventional interleaved modular multiplication algorithm and the digit-serial Montgomery multiplication, where the modular reduction is performed after shifting the result of the addition between the generated partial product and intermediate accumulated product, the proposed hardware algorithm does not require a shift operation neither on the addition of the generated partial products nor on the accumulation of the partial products. Thus, modular reduction can be applied to the results of these operations independently. This reduces the number of candidates for selecting the multiples of the modulus M required for performing the modular reduction, thus, reducing the critical path delay.

4. A Radix-4 Implementation Example

In this section we present a radix-4 implementation example of the proposed hardware algorithm. In this example, the radix $r = 4$. We denote with n' the number of bits required to represent the modulus M. Then, $2^{n'-1} < M < 2^{n'}$ and $n = \lceil n'/2 \rceil$. The radix-2 signed-digit representation (SD2) [10] is employed to all additions and subtractions so that they can be performed without carry propagation. The SD2 representation uses the digit set $\{\bar{1}, 0, 1\}$ where $\bar{1}$ denotes -1. An $(n'+1)$-digit SD2 integer $L = [l_{n'}, l_{n'-1}, \cdots, l_0]$ ($l_i \in \{\bar{1}, 0, 1\}$) has the value $\sum_{i=0}^{n'} l_i \cdot 2^i$. Addition of two SD2 numbers can be performed without carry propagation [10]. In cryptographic applications, modular multiplication are usually required sequentially. In order to enable the direct feedback of the output into the inputs and avoid the conversion from the SD2 representation into the binary representation in each multiplication, we represent the inputs and the output using the same SD2 representation. Then, X, Y and Z are SD2 numbers of $n' + 1$ digits in the range $[-M + 1, M - 1]$.

To accelerate the computations, 2-bit signed digit Booth's recoding algorithm [9] is applied to the variables A and B. Note that an $(\frac{n'}{2})$-bit binary multiplier is recoded into a radix-4 number of $\frac{n}{2} + 1$ digits. The i-th recoded digit of A is denoted as $\hat{a}_i$. The i-th recoded digit of B is denoted as $\hat{b}_i$. The recoding rule for the variable B is shown in Table 1. The same recoding rule can be applied to variable A. However, this rule can be simplified if we take into account the direction that carry propagates.

Table 1 The recoding rule for B in Step 2

(a) Phase 1

$b_{2i+1}b_{2i}$	b_{2i-1}	bc_{i+1}	bt_i
$\bar{1}1$	–	$\bar{1}$	1
$\bar{1}0$	non-negative	0	$\bar{2}$
	negative	$\bar{1}$	2
$\bar{1}\bar{1}$	–	$\bar{1}$	1
$0\bar{1}/\bar{1}1$	–	0	$\bar{1}$
00	–	0	0
$01/1\bar{1}$	–	0	1
10	non-negative	1	$\bar{2}$
	negative	0	2
11	–	1	$\bar{1}$

(b) Phase 2

$bt_i bc_i$	$\hat{b}_i$
$\bar{2}\bar{1}/21$	×
$\bar{1}\bar{1}/\bar{2}0$	$\bar{2}$
$0\bar{1}/\bar{1}0/\bar{2}1$	$\bar{1}$
$1\bar{1}/00/\bar{1}1$	0
$2\bar{1}/10/01$	1
$20/11$	2

Following the notation of the previous section, the variable L involved in the calculation of $X \cdot Y_H \bmod M$, represents a number $\gamma \in Z_M$ as an SD2 number of $(n' + 1)$ digits which satisfies $-d \cdot M < L < d \cdot M$ and $L \equiv \gamma \pmod{M}$. d is a parameter and can be any number that satisfies $\frac{5}{8} \leq d \leq \frac{2}{3}$. We explain how these values are derived. The shifting procedure and the modular reduction are based on the following recurrence.

$$L_{j+1} := 4 \cdot L_j - \hat{p}_j \cdot M$$

We select $\hat{p}_j$ from $\{\bar{2}, \bar{1}, 0, 1, 2\}$ so that modular reduction can be performed by adding 0 or $\pm M$ or $\pm 2M$ which can be obtained from shifts and complements of the modulus. In order that L_j stays in the range $-d \cdot M < L < d \cdot M$, the following inequality must hold:

$$4 \cdot d \cdot M - 2 \cdot M \leq d \cdot M$$

From this inequality, $d \leqq \frac{2}{3}$. Now, we calculate how many digits we need to check for determining $\hat{p}_j$. Assume that we compare $4 \cdot L_j$ with $\pm\frac{M}{2}$ and $\pm\frac{3}{2}M$ down to the k-th position. We need to find the largest k that satisfies the following inequality:

$$2 \cdot 2^k \leqq (d - \tfrac{1}{2}) \cdot M$$

Since M can be $2^{n'-1}+1$, the inequality $k \leqq n' - 2 + \log_2(d - \frac{1}{2})$ must hold. Since $d \leqq \frac{2}{3}$, k is at most $n' - 5$. Conversely, k is $n' - 5$ when $d \geqq \frac{5}{8}$. The Robertson's diagram for the modular reduction procedure can be found overleaf in Fig. 4.

We will now explain the procedure of calculating the modular reduction. The selection of $\hat{p}_j$ from $\{\bar{2}, \bar{1}, 0, 1, 2\}$ is performed by comparing $4 \cdot L_j$ with $\pm\frac{M}{2}$ and $\pm\frac{3}{2}M$ down to the $(n-5)$-th position. If we define $top(W)$ as a function that evaluates the most significant digits down to the $(n-5)$-th position of a number W, then the rule for selecting the values for $\hat{p}_j$ is as follows.

$$\hat{p}_j = \begin{cases} \bar{2} & \text{if} \quad top(L_j) < -top(\frac{3}{2}M) \\ \bar{1} & \text{if} \quad -top(\frac{3}{2}M) \leqq top(L_j) < -top(\frac{M}{2}) \\ 0 & \text{if} \quad -top(\frac{M}{2}) \leqq top(L_j) < top(\frac{M}{2}) \\ 1 & \text{if} \quad top(\frac{M}{2}) \leqq top(L_j) < top(\frac{3}{2}M) \\ 2 & \text{if} \quad top(L_j) \geqq top(\frac{3}{2}M) \end{cases}$$

$top(L_j)$ can be calculated from the most significant 6 digits of L_j. We can calculate $top(\frac{M}{2})$ from the most significant 4 bits of M. We can calculate $top(\frac{3}{2}M)$ from the most significant 5 bits of M so that $|\frac{3}{2}M - top(\frac{3}{2}M)| \leqq 2^{n'-4}$. Since M is a binary number, the addition of $\pm M$ or $\pm 2M$ for obtaining L_{j+1} is simpler than the conventional SD2 addition. For details, see [9].

The variable R involved in the calculation of $X \cdot Y_L \cdot r^{-\frac{n}{2}} \bmod M$ represents an SD2 number of $n'+2$ digits which satisfies $-2M < R < 2M$. The shifting operation and the Montgomery reduction, i.e. the calculation of $R/4 \bmod M$, is performed by using the function $MQRTR(R, M)$. For the details of how this function is implemented, see [3].

For the addition of the partial products. i.e. $C_j := (a_j \cdot L_j + b_{\frac{n}{2}-j} \cdot R_j) \bmod M$, we use a more redundant representation to make the modular reduction procedure simple. The variable C that stores the result of the addition after performing the modular reduction represents an SD2 number of $n'+2$ digits which satisfies $-2M < C < 2M$. After adding the partial products, we add $2M$ or 0 or $2M'$, accordingly as the value of the number formed by the three most significant digits of $\hat{a}_j \cdot L_j + \hat{b}_j \cdot R_j$ 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 $-M$. For the details of this modular addition procedure, see [10].

The result of the previous addition is added to the intermediate accumulated product stored in variable D. D also represents an SD2 number of $(n'+2)$ digits which satisfies $-2M < D < 2M$. Modular reduction is performed in the same way as described for the addition of the partial products. The result obtained in Step 3 can be reduced into the range $-M < D < M$ by executing a shift operation and a modular reduction using the same rule. Then, the final result can be obtained from the most significant $n'+1$ digits of D.

The radix-4 version of the hardware algorithm described here can be implemented using four redundant modular adder, where two of them are simpler, three registers for storing SD2 numbers, one SD2 shift register for storing the intermediate accumulated product, one register for storing the modulus M and two SD2 shift registers for storing the split two parts of the multiplier. Taking into account that the SD2 adder can be implemented with carry save adders [5], and that the modular reduction procedure used in our implementation requires an extra delay of one AND gate, a modular multiplier based on the proposed hardware algorithm can be implemented with similar performance to a radix-16 pipelined Montgomery multiplier [7].

In our implementation, the modular reduction of the variables that initially store the multiplicand can be performed in parallel to the addition of the generated partial products without increasing the clock cycle time. This enables the reduction of the number of stages for pipelining, and thus, the number of registers to latch the intermediate results between the stages. For obtaining similar performance, the proposed modular multiplier can be implemented with two less stages for pipelining and requires one less redundant adder, three less n-digit registers for storing redundant binary numbers and less complex multiplexers compared to the radix-16 modular multiplier of [7].

With our implementation, transformation back and forth between the ordinary integer set and KT-residue class representation can be performed with the same hardware. Furthermore, precomputation of the modulus is no longer necessary and postprocessing is simplified.

5. Concluding Remarks

We have proposed a fast hardware algorithm for calculating modular multiplication based on the recently proposed bipartite modular multiplication method. The addition of the partial products to the intermediate accumulated product is pipelined in order to reduce the critical path delay. Transformations back and forth between the ordinary representation and the KT-residue system can be performed using

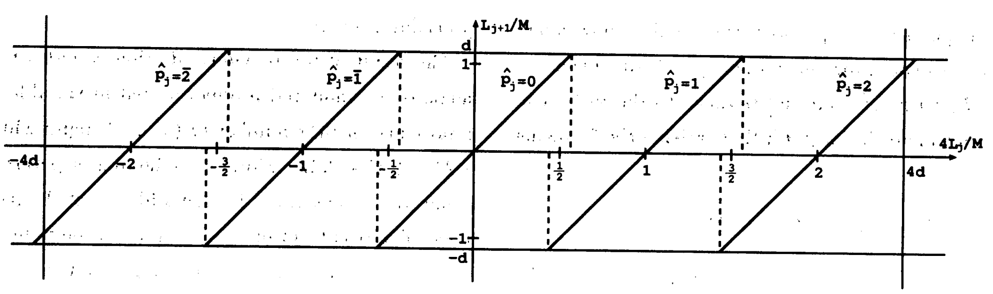


Fig. 4 Robertson's diagram for the modular reduction on L

the same hardware. We have presented a radix-4 implementation example of the hardware algorithm as an efficient alternative for calculating modular multiplication. Although we have described the proposed algorithm for $\alpha = 1/2$ (which means that the multiplier is split into two parts of equal size), a different value can be assigned for this parameter when the modular reduction and the Montgomery reduction are implemented with different critical path delays. In such a case, α can be calculated from the difference of the performance and one of the reduction operation is calculated using higher radices.

Acknowledgment

This work has been partially supported by "The Hori Information Science Promotion Foundation".

References

[1] E. F. Brickell, "A fast modular multiplication algorithm with application to two key cryptography," in *Advances in Cryptology, Proc. CRYPTO'82*, D. Chaum *et al.*, Eds. New York: Plenum, 1983, pp. 51–60.

[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.

[3] M. E. Kaihara and N. Takagi, "A Hardware Algorithm for Modular Multiplication/Division," *IEEE Trans. Computers*, vol. 54, no. 1, pp 12–, Jan. 2005.

[4] M. E. Kaihara and N. Takagi, "Bipartite Modular Multiplication," *Proc. Cryptographic Hardware and Embedded Systems - CHES 2005*, LNCS 3659, pp. 201-210, Aug./Sept. 2005.

[5] P. Kornerup, "Reviewing 4-to-2 Adders for Multi-Operand Addition," *Proc. The IEEE Int. Conf. Application-Specific Systems, Architectures, and Processors, ASAP02*, San Jose, California,pp. 218-229, July 17-19, 2002.

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

[7] H. Orup, "Simplifying quotient determination in high-radix modular multiplication," *Proc. 12th IEEE Symp. Computer Arithmetic*, S. Knowles and W. H. McAllister, eds., pp. 193-199, Bath, England, 1995.

[8] 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.

[9] N. Takagi, "A radix-4 modular multiplication hardware algorithm for modular exponentiation," *IEEE Trans. Comput.*, vol. 41, no. 8, pp. 949–956, Aug. 1990.

[10] 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.

[11] A. F. Tenca, G. Todorov, and Ç. K. Koç, "High-Radix Design of a Scalable Modular Multiplier," *Cryptographic Hardware and Embedded Systems - CHES 2001*, Ç. K. Koç, D. Naccache, C. Paar, eds., pp. 185–201, Paris, France, 2001.