# A Fast Modular-multiplication Algorithm based on a Higher Radix

Hikaru Morita
*NTT Communications and Information Processing Laboratories*
*3-9-11, Midori-cho, Musashino-shi, Tokyo, 180 Japan*

**Abstract**

This paper presents a new fast compact modular-multiplication algorithm, which will multiply modulo $N$ in $log(N)/log(r)$ clock pulses when the algorithm is based on radix $r$ ($r \geq 4$).

## 1 Introduction

Modular multiplication is essential for public-key cryptosystems such as the Rivest-Shamir-Adleman scheme (RSA) [1]. To guarantee security, the multiplication-word lengths have to be significantly greater than conventional computer word lengths. Therefore, techniques for speeding-up the modular multiplication are important.

Brickell [2] has shown how to design a compact operation based on radix 2 without frequent data transmission between processors performing modular multiplication and memories storing data in progress.

This paper proposes a new modular multiplication algorithm based on a radix higher than two that is faster than conventional algorithms based on radix 2. This new algorithm overcomes the problem of overflowing a limited computation range by using a partial product to adjust a partial remainder before a multiply-addition of the partial product. Furthermore, an approximate method is developed to reduce the comparator which determines the partial remainder. Consequently, the modular multiplier using this algorithm based on radix 4 can compute 512-bit modular exponentiation at a throughput of 80 Kbits/s at 30 MHz.

## 2 New Fast Compact Algorithm

If "$n$" is the bit length of the modulus $N$, then modular multiplication is represented as $A \times B$ *modulo* $N$ where $A$, $B$, and $N$ are $n$-bit binary integers related by $A$, $B$

G. Brassard (Ed.): Advances in Cryptology - CRYPTO '89, LNCS 435, pp. 387-399, 1990.

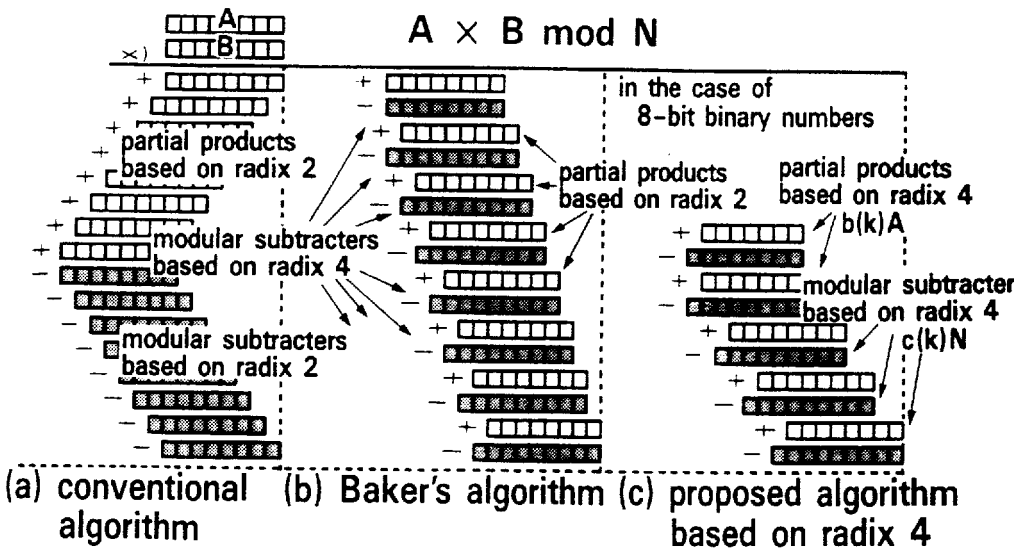


**Figure 1 Comparison of modular multiplication algorithms**

$\in [0, N-1]$. A conventional modular-multiplication algorithm shown in *Figure 1(a)* multiplies first and then divides the $2n$-bit product by the modulus $N$. This algorithm cannot accomplish high-speed computation and needs a massive amount of hardware because data transmission between processors and memories occurs frequently.

Consequently, methods using specialized hardware have been developed [2,3,4]. These methods use compact $n$-bit-length operators in which each subtraction step in binary division is embedded in the repeated multiply-addition. Brickell [2] has particularly proposed a modular multiplier that can perform one step of addition and subtraction simultaneously in each clock pulse. The modular multiplier needs about $n$ clock pulses.

This section describes a new algorithm based on a radix higher than two, which can reduce the amount of processing.

## 2.1 Approaches

A higher radix has been used to speed-up ordinary multiplication and division. For modular multiplication, Baker [5] has designed an algorithm which combines modular subtraction based on radix 4 and multiply-addition based on radix 2 to simplify operation as shown in *Figure 1(b)*. However, until now, no algorithm using a higher radix to speed-up modular multiplication has been developed because of the problem of overflowing a finite computation range.

The new algorithm presented here was designed to speed-up modular multiplication by using the same higher radix throughout. In practice, the addition and subtraction steps can be reduced to half or less those of the other algorithms as shown in *Figure 1 (c)*.

The proposed algorithm is carried out by using the following equation repeatedly.

$$R^{(k-1)} \leftarrow rR^{(k)} + b(k)A - c(k)N \tag{1}$$

where "$k$" is the step number of repeated processing, "$r$" is the radix number, $R^{(k)}$ and $R^{(k-1)}$ are partial remainders, $b(k)A$ is a partial product, and $c(k)N$ is a modulus subtracter. To overcome the problem of overflowing, this algorithm uses the following approaches:

**(1)** To prevent the absolute value of the next partial remainder $R^{(k-1)}$ from overflowing the upper limit $dN$ (the variable $d$ is derived in *Appendix A*), the modulus subtracter $c(k)N$ is determined by using the next partial product $b(k-1)A$ in advance.

**(2)** To reduce the absolute range of the partial product $b(k)A$, the multiplicand $A$ is modified from the range $[0, N-1]$ into the range $[-N/2, N/2]$.

## 2.2 Procedure based on a Higher Radix

The main procedure of the algorithm is explained by using the modified Robertson's diagram. *Figure 2* shows the diagram for radix 4 in which the boundary variable $d$ equals $7/12$. The horizontal axis shows the partial dividend $(rR^{(k)} + b(k)A)/N$ of the present remainder $R^{(k)}$ plus the present partial product $b(k)A$. The vertical axis shows the next remainder $R^{(k-1)}/N$. The graph $c$ (where $c \in \{-2, -1, 0, 1, 2\}$ for radix 4) in the diagram represents Eq. (1). After the subtracter $c(k)N$ is determined by the graph to have the value of $(rR^{(k)} + b(k)A)/N$ in the horizontal axis, the remainder $R^{(k-1)}$ is calculated.

The graph $c$ takes the form of several discrete lines, each with a fixed 45-degree slope. However, this modified Robertson's diagram is different from the ordinary Robertson's diagram [6]. For example, in *Figure 2*, the window enclosing the range $[-7/3, 7/3]$ and $[-7/12 - bA/(4N), +7/12 - bA/(4N)]$ moves up or down according to the value of the each coefficient $b(k-1)$. Then, the boundary indices, $\ell_2, \ell_1, \ell_{-1}$, and $\ell_{-2}$, which show the boundaries between the discrete lines, move left or right.

The algorithm procedure is as follows:

**Step 1 (initialize the numbers):**

$$\begin{aligned} &\text{If } N < 2A,\ A \leftarrow A - N. \\ &n \leftarrow \lfloor log_2(N) \rfloor + 1 \\ &k \leftarrow \lfloor n/r' \rfloor + 1 \\ &R^{(k)} \leftarrow 0 \end{aligned} \tag{2}$$

where $r' = log_2(r)$.

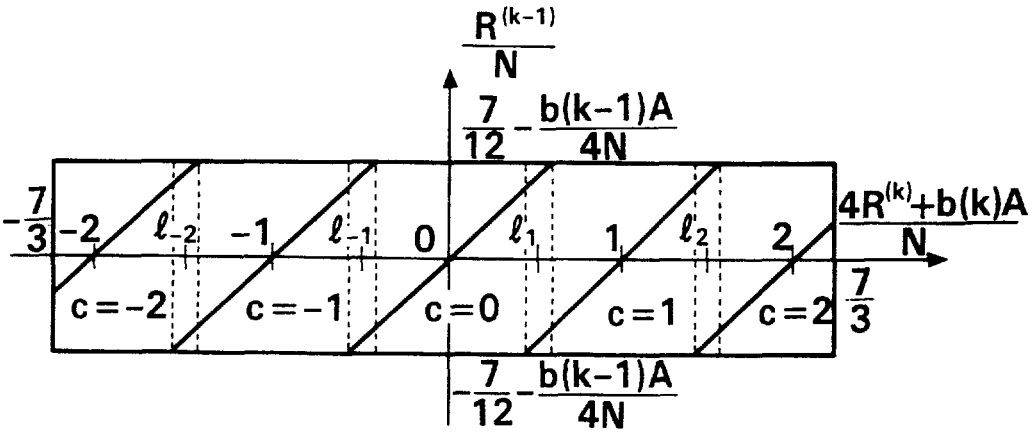


**Figure 2 Modified Robertson's diagram for the new algorithm (in a case of radix 4)**

**Step 2 (repeat $\lfloor n/r' \rfloor + 1$ rounds):**

$$c(k) \leftarrow f_c((rR^{(k)} + b(k)A), b(k-1)) \tag{3}$$

$$R^{(k-1)} \leftarrow rR^{(k)} + b(k)A - c(k)N \tag{4}$$

$$k \leftarrow k - 1$$

**Step 3 (last routine):**

$$A \leftarrow \begin{cases} R^{(0)} & \text{if } R^{(0)} \geq 0 \\ R^{(0)} + N & \text{otherwise} \end{cases}$$

In this procedure, we let variables $R$ and $A$ have sign bits, respectively.

*Notes:*

• function $f_c(R, b)$:

$$f_c \leftarrow \begin{cases} i & \text{if } \ell_i N < R \leq \ell_{i+1} N \\ 0 & \text{if } \ell_{-1} N \leq R \leq \ell_1 N \\ -i & \text{if } \ell_{-i-1} N \leq R < \ell_{-i} N \end{cases} \tag{5}$$

where

$\ell_i \equiv i - 1/2 - bA/(rN)$,

$\ell_{-i} \equiv -i + 1/2 - bA/(rN)$,

and the positive integer $i$ is in $\{1, 2, \cdots, r/2\}$.

The boundary indices $(\ell_i, \ell_{-i})$ are prepared beforehand for all $b \in \{-r/2, \cdots, -1, 0, 1, \cdots, r/2\}$.

• function $b(k)$ for the multiplier $B$:

$$b(k) \leftarrow -rB[r'k] + \sum_{m=0}^{r'-1} rB[r'k - m]/2^{m+1} + B[r'k - r'] \tag{6}$$

where by choosing bit-level representation, the variable $B$ is represented by the vectors $B[n]$, $B[n-1]$, $\cdots$, and $B[1]$, and $B[0] = 0$. Eq. (6) can satisfy $b(k) \in \{-r/2, \cdots, -1, 0, 1, \cdots, r/2\}$.

We can let $R$ represent the usual binary remainder in a carry-save redundant form. $R$ has two components: the sum $R_s$ and the carry $R_c$, where $R \equiv R_s + 2R_c$. By using carry-save adders, we can reduce carry propagation until every addition and subtraction pair takes only one clock.

## 2.3 Approximate method

To simplify the $n$-bit-length comparison in Eq. (5) for determining the modulus subtracter $cN$, the comparison is replaced with a short-precision comparison of $\ell_i N$ and $R$. Consequently, the previous function $f_c(R,b)$ in *section 2.2* is replaced the following function.

• function $f_c(R,b)$:

$$f_c \leftarrow \begin{cases} i & \text{if } L_i N_{top} < R_{top} \le L_{i+1} N_{top} \\ 0 & \text{if } L_{-1} N_{top} \le R_{top} \le L_1 N_{top} \\ -i & \text{if } L_{-i-1} N_{top} \le R_{top} < L_{-i} N_{top} \end{cases} \tag{7}$$

where

$L_i \equiv i - 1/2 - bA_{top}/(rN_{top})$,
$L_{-i} \equiv -i + 1/2 - bA_{top}/(rN_{top})$,
$R_{top} \equiv \{\text{top } (x + log_2(r)) \text{ bits of } R\}$,
$N_{top} \equiv \{\text{top } x \text{ bits of } N\}$,
$A_{top} \equiv \{\text{top } x \text{ bits of } A\}$.

The number of top bits $x$ ($x = 7$ for radix 4) is derived in *Appendix B*. $L_i$ and $L_{-i}$ which are approximate numbers corresponding to the boundary indices $\ell_i$ and $\ell_{-i}$ are prepared beforehand for $b \in \{-r/2, \cdots, -1, 0, 1, \cdots, r/2\}$.

## 2.4 Remarks

This algorithm is fast (as shown in *Table 1*) because steps for addition and subtraction are reduced, and it can use a compact operator with an $n$-bit word. Other features of the algorithm related to hardware are as follows:

**(1)** Small amount of hardware:

The partial product $bA$ and the modular subtracter $cN$ are produced by using only selectors and exclusive ORs. The coefficient $c(k)$ is determined by the short-precision comparator.

**(2)** Easy to speed-up:

The algorithm can use carry-save adders which have no carry propagation.

# Table 1 Comparison of modular multiplication algorithms

| | Steps of Addition and Subtraction | | ratio |
|---|---|---|---|
| | proposed algorithm | Baker's algorithm | |
| average case | 2n(r−1)/(rr′)<br>(multiply:n(r−1)/(rr′))<br>(modulo:n(r−1)/(rr′)) | 5n/4<br>(multiply:n/2)<br>(modulo:3n/4) | 5rr′/8(r−1)<br>(1.7 if r=4)<br>(2.1 if r=8) |
| worst case | 2n/r′<br>(multiply:n/r′)<br>(modulo:n/r′) | 2n<br>(multiply:n)<br>(modulo:n) | r′<br>(2 if r=4)<br>(3 if r=8) |

**(3)** Easy to expand:

The algorithm has a bit-slice structure.

## 3 Applications for Radix 4

A radix-4 modular multiplier is suitable for implementing the proposed algorithm in the latest C-MOS technology, so applications for radix 4 are described in this section.

### 3.1 Procedure for Radix 4

In the procedure in *section 2.2*, the new algorithm has to perform Eqs. (3) and (4) simultaneously. However, it is hard to speed-up this procedure, because the total delay time is the delay time of the function $f_c$ of Eq. (3) plus the addition operation in Eq. (4).

Therefore, the procedure is modified to the following procedure:

**Step 1 (initialize the numbers):**

$$\text{If } N < 2A,\ A \leftarrow A - N.$$

$$\begin{aligned} n &\leftarrow \lfloor log_2(N) \rfloor + 1 \\ k &\leftarrow \lfloor n/2 \rfloor + 1 \\ R^{(k)} &\leftarrow 0 \\ c(k+1) &\leftarrow 0 \end{aligned} \tag{8}$$

**Step 2 (repeat $\lfloor n/2 \rfloor + 1$ rounds):**

$$R^{(k-1)} \leftarrow 4(R^{(k)} - c(k+1)N) + b(k)A \tag{9}$$

$$c(k) \leftarrow f_c(R^{(k-1)}, b(k-1)) \tag{10}$$

$$k \leftarrow k - 1$$

**Step 3 (last routine):**

$$R^{(0)} \leftarrow R^{(0)} - c(1)N \tag{11}$$

$$A \leftarrow \begin{cases} R^{(0)} & \text{if } R^{(0)} \geq 0 \\ R^{(0)} + N & \text{otherwise} \end{cases}$$

The modified procedure gives the function $f_c$ a delay time of one clock pulse to produce the coefficient $c(k)$.

## 3.2 Generation of Partial Product Coefficient $b$ related by Redundant Multiplier $B$

To speed-up modular exponentiation using the modular multiplier, the modular multiplier has to be able to deal directly with the multiplier $B$ in a carry-save redundant form because the final remainder $R^{(0)}$ may be put in the place of the multiplier $B$. Brickell's algorithm can successfully deal with redundant numbers $B_s$ and $B_c$ of the multiplier $B$ ($B = B_s + 2B_c$) by using the characteristic that

$$s \wedge \sigma = 0, \tag{12}$$

where $s \leftarrow u \oplus v$, $\sigma \leftarrow u \wedge v$, and $u$ and $v$ are independently 0 or 1. However, our algorithm cannot use Brickell's technique which is based on radix 2. Therefore, we have extended Booth's multiplication [6] for the redundant multiplier $B$. Eq. (6) is modified to the following function $b(k)$. This new method produces a radix-4 partial product $bA$ by using 4 neighborhood bits of $B_s$ and $B_c$ (shown in *Appendix C*).

• function $b(k)$ for the redundant multiplier $B$:

$$\begin{aligned} b(k) \leftarrow\ & -2(B_s[2k] \oplus B_c[2k-1]) + B_s[2k-1] + B_c[2k-2] \\ & -4\overline{(B_s[2k] \oplus B_c[2k-1])} \wedge B_s[2k-1] \wedge B_c[2k-2] \\ & +B_s[2k-2] \vee B_c[2k-3] \vee (B_s[2k-3] \wedge B_c[2k-4]) \end{aligned} \tag{13}$$

where

$B_s[i] = B_s[n+1]$ and $B_c[i] = B_c[n+1]$ for $i \geq n+1$,
$B_s[i] = B_c[i] = 0$ for $i \leq 0$,
and $B_s[i] \wedge B_c[i] = 0$ for all $i \in [1, n]$.

Eq. (13) is derived in *Appendix C*. It satisfies $b(k) \in \{-2, -1, 0, 1, 2\}$.
If the multiplier $B$ is a single number, then $B_c = 0$.

## 3.3 Hardware Implementation

The modular multiplier using this algorithm, which can perform one step of addition and subtraction simultaneously in each clock pulse, is constructed by arranging $n$ cells into an array. Each cell, which has 5 registers, 3 full adders, and some logic gates as shown in *Figure 3*, contains about 100 gates.

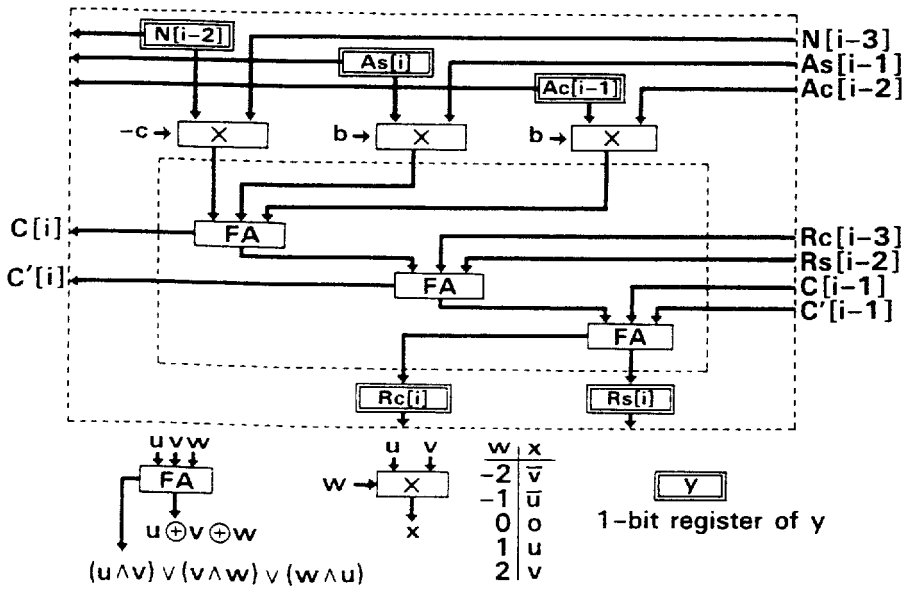


**Figure 3 Cell of "$R^{(k-1)} \leftarrow 4(R^{(k)} - c(k+1)N) + b(k)A$"**

Using the latest C-MOS technology, the total delay time, most of which is the delay time of adders and the writing time for registers, is about 20 to 30 ns. Therefore, the 512-bit modular multiplier has about 50 Kgates and a delay time of about 8 $\mu$s. The delay time for 512-bit modular exponentiation using it is about 6 ms.

# 4 Conclusion

A new compact modular-multiplication algorithm has been developed based on a higher radix using an $n$-bit-length operator. The throughput of the algorithm is more than twice as fast as conventional ones. This algorithm allows a 512-bit modular multiplier based on radix 4 to be made with about 50 Kgates. The throughput of 512-bit modular exponentiation using it will be about 80 Kbits/s at 30 MHz.

# Acknowledgment

The author wishes to express his thanks to Dr Kunio Murakami who provided him with the opportunity to pursue this research. He would also like to thank Tsutomu Ishikawa and the other members of NTT LABS who discussed the research.

# References

[1] Rivest, R.L., Shamir, A., and Adleman, L., "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," Comm.ACM, Vol.21(2), pp.120-126, Feb.1978.

[2] Brickell, E.F., "A Fast Modular Multiplication Algorithm with Application to Two Key Cryptography," Advances in Cryptology- CRYPTO'82, pp.51-60, Plenum NY, 1983.

[3] Blakley, G.R., "A Computer Algorithm for Calculating the Product AB Modulo M," IEEE Trans.Comput., Vol.C-32, pp.497-500, May 1983.

[4] Hoornaert, F., Decroos, M., Vandewalle, J., and Govaerts, R., "Fast RSA-hardware: Dream or Reality ?," Advances in Cryptology- EUROCRYPT'88, pp.257-264, Springer-Verlag, 1988.

[5] Baker, P.W., "Fast Computation of A * B Modulo N," Electron.Lett., Vol.23, No.15, pp.794-795, July 1987.

[6] Hwang, K.,"Computer Arithmetic: Principles, Architecture, and Design," John Wiley & Sons, 1979.

# Appendix A: Derivation of the boundary variable $d$

*(For radix 4)*
The vertical boundary variable 7/12 in *Figure 2* is assumed to be the boundary variable $d$.
First, the following equation is the necessary condition for the graph to continue.

$$d > \frac{1}{2}. \tag{14}$$

The condition that $(4R + bA)/N$ exists in the horizontal range $[-4d, 4d]$ is

$$4d \leq 2 + d - \frac{|bA|}{4N}. \tag{15}$$

The conditions of $|b| \leq 2$ and $|A| \leq N/2$ allow Eqs. (14) and (15) to be transformed into

$$\frac{1}{2} < d \leq \frac{7}{12}. \tag{16}$$

Consequently,

$$d = \frac{7}{12} \tag{17}$$

is obtained.

*(General case for radix r)*
By using the same method as in the previous case and the conditions of $|b| \leq r/2$, $|c| \leq r/2$, and $|A| \leq N/2$,

$$d = \frac{1}{2} + \frac{1}{4(r-1)} \tag{18}$$

is obtained.

# Appendix B: Derivation of a top-bit number

*(Definitions)*

$$\hat{R} \equiv rR + bA \tag{19}$$

$$N_{top} \equiv \{ \text{ top } x \text{ bits of } N \} \tag{20}$$

$$A_{top} \equiv \{ \text{ top } x \text{ bits of } A \} \tag{21}$$

$$R_{top} \equiv \{ \text{ top } (x + log_2(r)) \text{ bits of } \hat{R} \} \tag{22}$$

where "$x$" is a positive integer, and $b \equiv b(k)$.

*(Case of radix 4)*
In *Figure 2*, the boundary index $\ell_1$ between $c = 0$ and $c = +1$ is

$$\ell_1 \equiv \frac{1}{2} - \frac{\hat{b}A}{4N} \tag{23}$$

where $\hat{b} \equiv b(k-1)$.
Let $L_1$ be the approximate value of $\ell_1$,

$$L_1 \equiv \frac{1}{2} - \frac{\hat{b}A_{top}}{4N_{top}}. \tag{24}$$

If the following equation is satisfied, then the $n$-bit-length comparison between $\ell_1$ and $\hat{R}$ can be replaced by the short-precision comparison between $L_1$ and $R_{top}$.

$$\epsilon \leq \delta - \delta' \tag{25}$$

where

$$\epsilon \equiv |\hat{R}/N - R_{top}/N_{top}| \tag{26}$$
$$\delta \equiv d - 1/2 = 1/12 \tag{27}$$
$$\delta' \equiv |L_1 - \ell_1| \tag{28}$$

By using $\hat{R}/N \leq rd = 7/3$, Eq. (26) is transformed into

$$\epsilon \leq |\frac{R_{top}+1}{N_{top}} - \frac{R_{top}}{N_{top}+1}| \leq \frac{10}{3N_{top}}. \tag{29}$$

By using $|b| \leq 2$,

$$\delta' < \frac{|b|}{4}\frac{2}{N_{top}} \leq \frac{1}{N_{top}}. \tag{30}$$

From Eqs. (25), (27), (29), and (30),

$$N_{top} \geq 52. \tag{31}$$

Because the top bit of $N_{top}$ is always 1,

$$N_{top} \geq 2^{x-1}. \tag{32}$$

Consequently, selecting the minimum value of $x$ under the conditions of Eqs. (31) and (32) gives

$$x = 7. \tag{33}$$

In the other cases for the indices, $\ell_2$, $\ell_{-1}$, and $\ell_{-2}$, Eq. (33) is also satisfied.

*(General case for radix r)*
By using the same method as in the previous case, we obtain

$$N_{top} \geq 4(r-1)(rd+2). \tag{34}$$

Therefore, the minimum value of $x$ is

$$x = \lceil log_2\{(r-1)(rd+2)\} \rceil + 3. \tag{35}$$

# Appendix C: Derivation of Eq. (13)

Let us modify the digits in the upper region "k" to radix-4-multiply coefficient $b(k)$ ($b(k) \in \{-2,-1,0,1,2\}$) by using the digits in the lower region "k-1" where the condition of Eq. (12) is satisfied (these regions are shown in *Figure C.1*).

First, assume the auxiliary variables $(c_{k+1}, s_k)$ in the upper region "k" are the carry $c_{k+1}$ and the sum $s_k$ shown in *Table C.1*. These satisfy the conditions $c_{k+1} \in \{0,1\}$ and $s_k \in \{-2,-1,0,1\}$.

If the next equation is performed:

$$b(k) \leftarrow s_k + c_k, \tag{36}$$

then Eq. (36) satisfies $b(k) \in \{-2,-1,0,1,2\}$. Consequently, Eq. (13) is derived from Eq. (36) and the auxiliary variables shown in *Table C.1*.

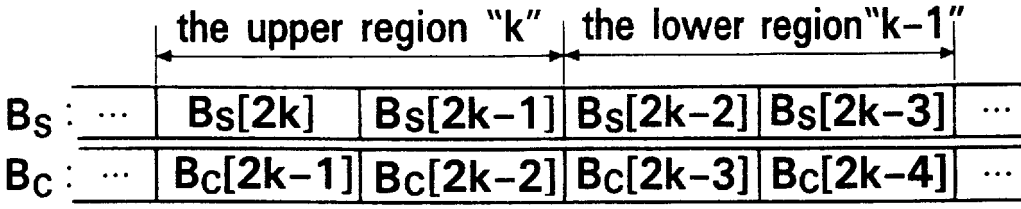


**Figure C.1 Alignment of $B_S$ and $B_C$**

**Table C.1 Definition of $(c_{k+1}, s_k)$**

| $B_S[2k]\ B_C[2k-1]$ | $B_S[2k-1]\ B_C[2k-2]$ | | | |
|---|---|---|---|---|
| | 00 | 01 | 11 | 10 |
| 00 | 0,0 | 0,1 | 1,-2 | 0,1 |
| 01 | 1,-2 | 1,-1 | × | × |
| 11 | 1,0 | 1,1 | × | × |
| 10 | 1,-2 | 1,-1 | 1,0 | 1,-1 |

Note: × means that $(c_{k+1}, s_k)$ doesn't have any value.