# High-Radix Modular Multiplication for Cryptosystems *

Peter Kornerup
Dept. of Mathematics and Computer Science
Odense University
Odense, Denmark
email: kornerup@imada.ou.dk

## Abstract

*Two algorithms for modular multiplication with very large moduli are analyzed, in particular for their applicability when a high radix is used for the multiplier. Both algorithms perform modulo reductions interleaved with the addition of partial products, one algorithm is using the standard residue system, whereas the other utilizes a non-standard system employing reductions modulo a power of the base. The emphasis is on situations – like in cryptosystems – where modular exponentiation is to be realized by many repeated modular multiplications on very large operands, e.g. for cryptosystems with key lengths of 500-1000 bits.*

## 1 Introduction

Modular multiplication is a fundamental operation in the implementation of modular exponentiation as needed in many cryptosystems, e.g. the RSA two-key system [6] and in the recently proposed digital signature standard DSS [3]. In such applications very large moduli are needed to safeguard the information, which makes modular exponentiation a very compute-intensive task. Also for use in communications a high throughput is needed. Parallelism has to be exploited to the highest degree possible, and single-chip implementations are preferable in many applications.

There has been a number of publications in recent years reporting on modular exponentiation. A survey of actual hardware implementations was given in [1], and further designs and implementations have been reported since then. All reported implementations so far have been based on multiplications where accumulated partial products are formed by interpreting the multiplier in base 2. A fairly straightforward such VLSI 1024-bit single-chip implementation was reported in [9], where modulo reductions are based on quotient estimates computed from the most significant bits of the accumulator contents. An alternative approach, where modulo reductions are computed from the least significant bits based on an idea by Peter Montgomery [2], has been reported implemented on a gate-array in [7], also base 2.

*This work has been supported by the Danish Natural Science Research Council, grant no. 5.21.08.02

In this paper we will analyze these two approaches for modular reductions during multiplications, however applied to higher radix implementations to reduce the total number of cycles. Emphasis will be on designs which are suitable for single-chip VLSI implementations of modular exponentiations. A design for radix 4 has recently been presented in [8], and a radix 32 design, which was presented in [5], is presently being implemented as a single chip prototype for RSA encryption/decryption with 561-bit keys and a projected speed above 64Kbps. Both of these designs employ the standard method of modular reduction.

This paper is organized as follows: In Section 2 modular exponentiation is first discussed, followed by some general considerations on the internal organization of the modular multipliers. Two computational schemes are identified, differing in the way in which the partial products are accumulated, with implications on the choice of radix for the multiplier within given area constraints, and the possibilities for parallelism and internal pipelining.

Section 3 then analyzes the standard algorithm for modular multiplication, where reduction steps are interleaved with the accumulation of partial products. Bounds on the quotients used for the reductions are derived, based on some parameters of the algorithm, and implications of these bounds are discussed.

In Section 4 an equivalent algorithm is analyzed, however based on the alternative residue system proposed by Peter Montgomery [2]. In this system the quotient determination is based on the least significant digits of the accumulated partial products, and is thus simplified.

Finally, in Section 5, the two algorithms are compared and discussed in the context of the two computational schemes introduced in Section 2.

## 2 Modular Exponentiation

A standard way of performing the modular exponentiation $x^e \bmod m$ is by repeated modular multiplications and squarings, scanning the exponent $e$ from the least-significant end, as described below:

**Algorithm 2.1** *(Modular Exponentiation)*

**Stimulus:** *A modulus* $m \geq 2$, *and integers* $x$ *and* $e$, *where* $0 \leq x < m$ *and* $e \geq 0$, $e = \sum_{i=0}^{n-1} e_i 2^i$.

**Response:** *An integer* $y$, *such that* $y = x^e \bmod m$.

**Method:**
$i := 0$; $y := 1$; $z := x$;
**while** $i < n$ **do**
  **if** $e_i = 1$ **then** $y := (y * z) \bmod m$;
  $z := (z * z) \bmod m$;
  $i := i + 1$;
**end**

Each cycle of the loop potentially requires two multiplications (noting that one factor is in common), which may be executed in parallel or pipelined through a single multiplier. In this case the product $y * z$ may just be inhibited if it is not needed, thus the execution time will be constant, which is advantageous if used in cryptosystems for communication. Since the wordsize of the operands is very large (500 - 1000 bits) each multiplication must be composed of a number of smaller multiplications, realized by a rectangular aspect-ratio multiplier, forming a partial product of the multiplicand by a digit of the multiplier. After addition of each partial product the accumulator contents has to be reduced modulo $m$, to keep its value as small as possible. This requires that a suitable multiple of the modulus has to be determined and subtracted each time a partial product has been added.

Due to the large wordsize and the large number of multiplications needed, it is essential that accumulation takes place in a redundant representation, and that the resulting products in redundant representation can be used directly as factors in subsequent multiplications. It is also essential to notice that it is not possible to perform "perfect" modulo reductions when the operands are in redundant representations. But as we shall show in the following two sections, it is possible to compute products $S \equiv AB \pmod m$ such that $-m < S < m$, even when $-m < A, B < m$, while $A$, $B$ and $S$ are in redundant representations. Thus only at the very end of the exponentiation is it necessary to convert into a non-redundant representation, and possibly to add $m$ to obtain the correct result. To save space on chip, a final conversion into non-redundant representation, and into the interval $0 \leq S < m$, can be realized by shifting $S$ and $S + m$ through two serial adders, least significant digit first, directly transmitting the two results off-chip, while serially inputting another operand. The decision which result to use can then rest on the environment.

We have chosen here and in the subsequent analysis of two different methods for modular multiplication to assume that a signed digit base 2 representation (e.g. "borrow-save") is used for the accumulation of partial products, which is the reason for the symmetric intervals of operands and results mentioned above. An equivalent analysis could be performed, assuming "carry-save" representations of $A$, $B$ and $S$, satisfying $0 \leq A, B, S < 2m$; but there are advantages in using a balanced digit set, due to the smaller absolute values of digits.

The basic structure of the modular multiplications $S := AB \bmod m$ to be analyzed in the following is

**loop**
  *{determine a reduction factor* $q_i$*}*
  $S := 2^f(S - q_i m) + b_i A$
  $next(i)$
**end**

where $f$ is either $k$ or $-k$ and $b_i$ is a radix $2^k$ digit of $B$.

As we shall see later the reduction factors $q_i$ will belong to the same range as the digits $b_i$, hence the updating of $S$ requires three multiplications $q_i m$, $b_i A_1$ and $b_i A_2$ where $A = A_1 \pm A_2$ is a redundant encoding of $A$. One possible way of implementing the updating of $S$ is then sequentially to perform three fused multiply-add operations, using a single rectangular aspect-ratio multiplier, whose shorter dimension corresponds to the radix chosen for the digits $b_i$ and $q_i$. This is the method used in [5], where the three sequential multiplications (in radix 32) are overlapped with the determination of $b_i$ and $q_i$, and furthermore the multiplications and squarings of Algorithm 2.1 are pipelined utilizing the common factor in the two products computed. Let us call this implementation scheme the *S-Scheme* for the sequential execution of the multiplications.

An alternative as often used (e.g. in [7, 8]) with a small radix (e.g. 2 or 4) is to perform the multiplications in parallel, in the sense that $q_i m$, $b_i A_1$ and $b_i A_2$ are all accumulated in one single adder tree. In this *P-Scheme* the determination of $q_i$ (and $b_i$) cannot be overlapped with the multiplications, but it is still possible to pipeline the multiplications and squarings of Algorithm 2.1.

In the following sections we shall analyze two methods for modular multiplications, to provide a background for further architectural considerations.

## 3 Interleaved Modular Multiplication

The basic idea of this and the following algorithm is to interleave the accumulation steps of the multiplication with steps of a division operation. Based on an estimate of a quotient digit, a multiple of the modulus is subtracted from the value in the accumulator, and a new partial product is added in. To avoid the gradually increasing number of digits needed when adding a shifted version of the multiplicand if (as usual) starting with the least significant digit of the multiplier, it is essential here to start with the most significant digit.

To reduce the number of cycles we will use techniques known from high-radix multiplication and division, e.g. we will assume the multiplier is given in a radix $\beta = 2^k$ and quotient digits are determined in that base also. The system modulus $m$ satisfies $2^{k(n-1)} < m < 2^{kn}$ (cases where $m$ is a power of 2 are much simpler), and the algorithm will proceed through $n + 2$ cycles.

**Algorithm 3.1** *(Interleaved Modular Multiplication)*

**Stimulus:** *A modulus* $m > 2$, *and integers* $n \geq 1, k \geq 1$ *such that* $2^{k(n-1)} < m < 2^{kn}$. *Integers* $A$ *and* $B$, $-m < A < m$, $-m < B < m$ *where* $B = \sum_{i=-2}^{n-1} b_i(2^k)^i$, $b_{-2} = b_{-1} = 0$ *and* $b_i \in D = \{-\sigma, \cdots, \sigma\}$ *for* $0 \leq i \leq n-1$, $2^{k-1} \leq \sigma < 2^k$. *A real parameter* $\alpha$, $\frac{1}{2} \leq \alpha < 1$. *An integer parameter* $r$, $r \in \{0, k\}$.

**Response:** *An integer* $S$ *such that* $-m < S < m$ *and* $S \equiv AB \pmod{m}$.

**Method:**
$S := 0$; $i := n-1$;
**while** $i \geq -(1 + r/k)$ **do**
$L$: {*determine integer* $q_i$ *such that* $|S - q_i 2^r m| \leq \alpha 2^r m$};
$S := 2^k(S - q_i 2^r m) + b_i A$;
$i := i - 1$;
**end**;
$S := S \text{ div } 2^{k+r}$;

Note that we allow $A$ and $B$ to be negative, $-m < A, B < m$, and assume $B$ is represented in a redundant digit set. Since we shall prove that the resulting $S$ is in the same interval, and may also be in redundant representation, the algorithm is suitable for modular exponentiation.

**Theorem 3.2** *Algorithm 3.1 correctly computes* $S$ *such that* $S \equiv AB \pmod{m}$ *and* $-m < S < m$. *Furthermore the quotients* $q_i$ *satisfy the bound*

$$|q_i| \leq \lceil (\alpha(2^k + 1) + \sigma 2^{-r} \rceil - 1 \qquad (1)$$

*in the parameters* $\alpha, r$ *and* $\sigma$, $\frac{1}{2} \leq \alpha < 1$, $r \in \{0, k\}$ *and* $2^{k-1} \leq \sigma < 2^k$.

**Proof:** Since the multiplier digits $b_i$ are being added into successively shifted values of $S$, it is easy to see that $S$ satisfies the following invariant at the label $L$ during the loop:

$$I: \; S \equiv A \cdot \sum_{j=i+1}^{n-1} b_j 2^{k(j-i-1)} \pmod{m}.$$

Hence when $i = -1$ we have $S \equiv AB \pmod{m}$ at label $L$ and at termination of the loop $S \equiv AB2^{k+r} \pmod{m}$ since $b_{-2} = b_{-1} = 0$.

From the updating of $S$ we obtain

$$\begin{aligned} |S| &< 2^k |S - q_i 2^r m| + \sigma m \\ &\leq (\alpha 2^{k+r} + \sigma) m, \end{aligned} \qquad (2)$$

but for $i = -1$ and $-2$ a slightly sharper bound is

$$|S| \leq 2^{k+r} \alpha m < 2^{k+r} m. \qquad (3)$$

Equation (3) assures $-m < S < m$ after the final division by $2^{k+r}$. Note that for $r = 0$ the algorithm stops at $i = -1$, and by (3) the proper reduction has then been obtained when a final division by $2^k$ is performed.

The bound (1) follows from (2) and the way $q_i$ is determined: $\left|\frac{S}{2^r m} - q_i\right| \leq \alpha$, implying

$$|q_i| < \alpha(2^k + 1) + \sigma 2^{-r}. \qquad (4)$$

□

The parameter $\alpha$ specifies the "quality" of the quotient determination, $\alpha = \frac{1}{2}$ requires an exact division of $S$ by $2^r m$, whereas larger values of $\alpha$ allow more imprecise values of $q_i$, e.g. as determined by table look-up. The value $r = 0$ allows the algorithm to stop at $i = -1$, but the value of the bound (1) for $q_i$ is lower for $r = k$. Hence we shall choose $r = k$ in the following.

The range of values of $q_i$ is crucial to an implementation of the algorithm, since $q_i m$ must be computed or otherwise made available in each cycle. It would then be convenient if $q_i$ belong to the same digit set as $b_i$, i.e. $|q_i| \leq \sigma$. By (4) $|q_i| < \sigma + 1$ can be achieved if:

$$\alpha < \frac{(1 - 2^{-k})\sigma + 1}{2^k + 1}. \qquad (5)$$

As in high radix multiplications with recoded multipliers and in SRT division, we here face the problem of making multiples $q_i 2^k m$ and $b_i A$ available. One way of doing this is to represent the digits $b_i$ and $q_i$ themselves in a smaller radix, e.g. base 4 using the digit set $\{-2, -1, 0, 1, 2\}$. A base 64 digit $d$, $-42 \leq d \leq 42$, can thus be represented as:

$$d = a_2 4^2 + a_1 4 + a_0, \;\; a_i \in \{-2, -1, 0, 1, 2\}, \qquad (6)$$

implying that all multiples $b_i$ and $q_i$ can be generated by shifts and two adds. For base 256 a tree-structure of three adders is similarly sufficient to generate multiples in the range $[-170; 170]$. Note that such hardware corresponds to a "rectangular aspect-ratio multiplier" producing the product of $nk$ by $k$-bit operands, using base 4 for the multiplier.

If in general we assume that $k = 2p$ and represent digits $d$ as

$$d = \sum_{i=0}^{p-1} 4^i a_i, \;\; a_i \in \{-2, -1, 0, 1, 2\}$$

then $-d_{max} \leq d \leq d_{max}$ where $d_{max} = \frac{2}{3}(4^p - 1) = \frac{2}{3}(2^k - 1)$.

We thus choose $\sigma = d_{max}$ and find from (5) that it is sufficient that $\alpha$ satisfies

$$\alpha < \frac{\frac{2}{3}(2^k - 1)^2 + 2^k}{2^k(2^k + 1)} < \frac{2}{3}, \qquad (7)$$

so some sample values of the bound for $\alpha$ are then:

| $k$ | $\alpha <$ |
|---|---|
| 4 | 0.6102 |
| 6 | 0.6514 |
| 8 | 0.6627 |

(8)

which will assure that $|q_i| \leq \sigma$ when also $|b_i| \leq \sigma$. Note that for $k = 2$ the bound (7) requires $\alpha < \frac{1}{2}$, which is obviously not possible.

Based on the value of $\alpha$ it is now possible to determine how many leading digits of $S$ are needed to find an estimate of $q_i$ in Algorithm 3.1. Assume that $q_i$ is determined at label $L$ in the algorithm as

$$q_i = \text{round}\left(\frac{S+\Delta}{2^k m}\right)$$

where $\Delta$ is the truncation error by considering only some leading digits of $S$. Then from

$$\begin{aligned}\left|\frac{S}{2^k m} - q_i\right| &\leq \left|\frac{S}{2^k m} - \frac{S+\Delta}{2^k m}\right| + \left|\frac{S+\Delta}{2^k m} - q_i\right| \\ &\leq \left|\frac{\Delta}{2^k m}\right| + \frac{1}{2}\end{aligned}$$

we find that $\left|\frac{S}{2^k m} - q_i\right| \leq \alpha$ if $|\Delta| \leq (\alpha - \frac{1}{2})2^k m$.

Since by (2) and (7) we have $|S| < \frac{2}{3}(2^{2k} + 2^k)m < 2^{n+2k}$ it is sufficient to use an accumulator of width $(n+2)k$ radix 2 signed digits, where all operands are initially positioned such that $m$ has precisely $2k$ leading zeros. If $p$ leading digits are extracted from the accumulator containing $S$, then the truncation error $\Delta$ satisfies $\log_2|\Delta| \leq 2k - p + \lceil \log_2 m \rceil$, hence $p = k + 1 - \lfloor \log_2(\alpha - \frac{1}{2}) \rfloor$ is sufficient to assure $|\Delta| \leq (\alpha - \frac{1}{2})2^k m$. Since $S$ is in signed-digit binary a subsequent conversion into sign-magnitude [4] is needed for the table look-up. Note that only the magnitude part is needed for the look-up since $sign(q_i) = sign(S)$, thus using the values of $\alpha$ from (8) we obtain the following sufficient table sizes:

| $k$ | Table size |
|---|---|
| 4 | 512 |
| 6 | 1024 |
| 8 | 4196 |

(9)

The size of the table entries depends on the encoding of $q_i$ needed subsequently. An alternative is to compute the estimate of $q_i$ as the product of the leading digits of $S$ by a pre-computed approximate reciprocal of $m$. This reciprocal then has to be available to at least the same number of bits as used for table look-up. By a detailed analysis for the case $k = r = 2$ Takagi in [8] found 8 leading digits of $S$ sufficient, equivalent to a table size of 256. But the approach there was to compare digits of $S$ against bounds derived from Robertson diagrams, a similar method was also used in [5] for k=32.

# 4 Montgomery Modular Multiplication

By a clever change of residue system it is possible to simplify the determination of $q_i$. The trick is to trade reduction modulo $m$ with multiplications and reduction modulo $r$, where $r$ is chosen such that the latter operation is simple, e.g. where $r$ is a power of the base of the arithmetic. This method is based on [2], for completeness we shall here repeat the background on the residue system.

Given $m$ choose $r > m$ such that $\gcd(m, r) = 1$ and determine $r^{-1}$ as the multiplicative inverse of $r$ modulo $m$. Also determine $m'$, $0 < m' < r$, such that:

$$rr^{-1} - mm' = 1 \tag{10}$$

which is equivalent to requiring that[1] $|-mm'|_r = 1$, so $m'$ is the multiplicative inverse of $(-m)$ modulo $r$.

We then define a new residue operator, mapping integers into $\mathbb{Z}_m = \{i \in \mathbb{Z} | 0 \leq i < m\}$ as follows:

**Definition 4.1** *The mapping* $[\cdot]_m : \mathbb{Z} \to \mathbb{Z}_m$ *is defined by:*

$$[a]_m = i \in \mathbb{Z}_m \text{ for } a \in \mathbb{Z}$$

*if and only if* $i = (ar) \bmod m$, *or equivalently* $a \equiv ir^{-1} \pmod m$.

We will denote $[a]_m$ the *M-residue* after Peter Montgomery who suggested this residue system in [2].

At a first glance it looks like multiplication now has become more complicated since both multiplicand and multiplier contain the factor $r$, whereas the product is only supposed to contain a single factor $r$. But the product of the two residues also has to be reduced, and it turns out that this process is simplified in the sense that now only reductions modulo $r$ are needed (and not modulo $m$).

**Algorithm 4.2** *M-reduce(t)*

**Stimulus:** *An integer* $t$ *such that* $0 \leq t < rm$.
*Integer constants* $m, r, m', r^{-1}$ *such that* $\gcd(m, r) = 1, r > m > 2$ *and* $rr^{-1} - mm' = 1$.

**Response:** *An integer* $u$, $u = (tr^{-1}) \bmod m$.

**Method:**
$q := ((t \bmod r)m') \bmod r$;
$u := (t + qm) \operatorname{div} r$;
*if* $u \geq m$ *then* $u := u - m$;

The simplification occurs when $r$ is chosen as a power of the radix of the arithmetic used, so that the operations mod $r$ and div $r$ become trivial.

**Theorem 4.3** *Algorithm 4.2 correctly computes* $u = (tr^{-1}) \bmod m$ *given* $t$, $0 \leq t \leq rm$.

[1] Define $|a|_r = a \bmod r$.

**Proof:** Observing that $q \equiv tm' \pmod{r}$ we have $qm \equiv tm'm \equiv -t \pmod{r}$ and hence that $r$ divides $t + qm$. Also $ur \equiv t \pmod{m}$ so $u \equiv tr^{-1} \pmod{m}$, and finally $0 \leq t + qm < rm + rm$, so $0 \leq u < 2m$ before the final adjustment. □

Now let $x = [a]_m$ and $y = [b]_m$ and compute $z = M$-reduce$(xy)$, then $z = xyr^{-1} \bmod m$ and

$$ab \equiv (xr^{-1})(yr^{-1}) \equiv zr^{-1} \pmod{m},$$

hence $z = [ab]_m$ since $0 \leq z < m$.

The function $M$-reduce$(\cdot)$ can also be used to compute M-residues $[\cdot]_m$ since

$$[a]_m = M\text{-reduce}((a \bmod m)(r^2 \bmod m)),$$

where the constant $r^2 \bmod m$ can be pre-computed for the system. By Definition 4.1 the function $M$-reduce$(\cdot)$ directly maps M-residues back into the integer domain.

If many modular multiplications are to be performed then the cost of mapping back and forth can be amortized, and the multiplications then become simpler, since only reductions modulo $r$ will be needed.

We shall now modify Algorithm 3.1 for interleaved modular multiplication into an algorithm operating on M-residues. As before we shall reduce the number of cycles by assuming that the multiplier is given in radix $2^k$.

**Algorithm 4.4** *(Montgomery Modular Multiplication)*

**Stimulus:** *A modulus $m > 2$ with $\gcd(m, 2) = 1$ and integers $k \geq 2, n \geq 1$ such that $m < 2^{kn-2}$.*
*With $r = 2^{kn}$, integers $r^{-1}$ and $m'$ are given such that $rr^{-1} - mm' = 1$.*
*Also integers $A$ and $B$ are given where $-m < A < m$ and $-m < B < m$ with $B = \sum_{i=0}^{n}(2^k)^i b_i$, $b_n = 0$ and $b_i \in D = \{-\sigma, \cdots, \sigma\}$ for $0 \leq i \leq n-1$, with $2^{k-1} \leq \sigma < 2^k$.*

**Response:** *An integer $S$ such that $-m < S < m$ and $S \equiv ABr^{-1} \pmod{m}$*

**Method:**
$S := 0$; $i := 0$;
**while** $i \leq n$ **do**
$L$: $q_i := ((S \bmod 2^k)m') \bmod^* 2^k$;
$S := (S + q_i m) \operatorname{div} 2^k + b_i A$;
$i := i + 1$;
**end**

**Where:** *For all $a$, $a \bmod^* 2^k \equiv a \pmod{2^k}$ and $-2^{k-1} \leq a \bmod^* 2^k < 2^{k-1}$.*

Observe that here the values of $q_i$ belong to a subset of the digits of $B$, and that $q_i$ is determined directly from the least significant $k$ digits of $S$. For an implementation also note that only the least significant $k$ bits of $m'$ are needed (neither is $r^{-1}$ ever used), and since $m'$ is a constant for the system, a table look-up is feasible for moderate values of $k$. Also note that $m$ is always odd for cryptosystems, so here $\gcd(m, 2) = 1$ is fulfilled.

**Theorem 4.5** *Given two M-residues $A = [a]_m$ and $B = [b]_m$, Algorithm 4.4 correctly computes $S$, such that $[ab]_m \equiv S \equiv ABr^{-1} \pmod{m}$ with $-m < S < m$ for $k \geq 2$.*

**Proof:** As in the proof of Theorem 4.3 we note that $q_i \equiv Sm' \pmod{2^k}$ so $q_i m \equiv -S \pmod{2^k}$ and hence that $2^k$ divides $S + q_i m$ in the updating of $S$. Next we want to establish that the invariant

$$I : 2^{k(i-1)}S = A \cdot \sum_{j=0}^{i-1} b_j 2^{kj} + m \cdot \sum_{j=0}^{i-2} q_{j+1} 2^{kj}$$

holds at label $L$ during the loop. The invariant holds trivially for $i = 0$ and for $i = 1$, defining the value of a sum to be zero when the upper bound is smaller than the lower, and noting that $q_0 = 0$. Assuming it holds for $i = \ell$, from the updating of $S$, $2^k S' = S + q_\ell m + 2^k b_\ell A$, where $S'$ is the new value, we then obtain:

$$\begin{aligned} 2^{k\ell}S' &= A \cdot \sum_{j=0}^{\ell-1} b_j 2^{kj} + m \cdot \sum_{j=0}^{\ell-2} q_{j+1} 2^{kj} \\ &\quad + 2^{k(\ell-1)} q_\ell m + 2^{k\ell} b_\ell A \\ &= A \cdot \sum_{j=0}^{\ell} b_j 2^{kj} + m \cdot \sum_{j=0}^{\ell-1} q_{j+1} 2^{kj}, \end{aligned}$$

hence the invariant holds for $i = \ell + 1$. We then immediately find that upon exit of the loop

$$2^{kn}S = AB + m \cdot \sum_{j=0}^{n-1} q_{j+1} 2^{kj} \qquad (11)$$

so $S \equiv ABr^{-1} \pmod{m}$.

Equation (11) also allows us to find a bound for $S$ since $|q_i| \leq 2^{k-1}$

$$\begin{aligned} |S| &< 2^{-kn} m \left( 2^{kn-2} + 2^{k-1} \frac{2^{kn} - 1}{2^k - 1} \right) \\ &< m \left( \frac{1}{4} + \frac{2^{k-1}}{2^k - 1} \right) \end{aligned}$$

so that $|S| < m$ for $k \geq 2$.

□

Algorithm 4.4 has been formulated such that the multiplier $B$ is assumed to be in redundant form, and such that the computation of $S$ can be performed in redundant arithmetic. For $k \geq 2$ the result $S$ can thus

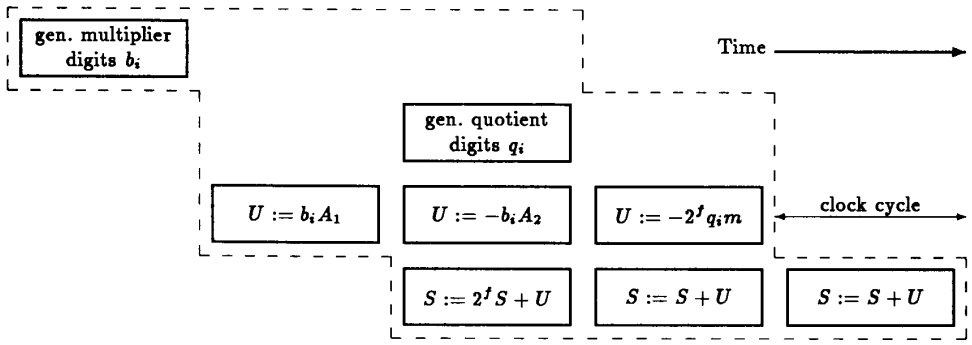

Figure 1: S-Scheme, internal pipelining of modular multiplication.

be used as a new multiplier, as needed in modular exponentiation, but it may have to be recoded into another digit set.

In this method operands initially have to be converted into the special residue system. This can be achieved by Algorithm 4.4 by multiplication with the pre-computed constant $r^2 \bmod m$. The algorithm can also be used for the final conversion back, since multiplication by the constant 1 maps $S$ back into an ordinary residue $s$, however such that $-m < s < m$, where $s$ still is in redundant representation.

## 5 Comparison and Implementation Considerations

The hardware requirements for implementations of the two algorithms are basically the same, the major component being a multiplier structure and a redundant adder for the updating of $S$. Assuming that the partial products $q_i m$ and $b_i A$ are to be computed by a $nk$-by-$k$ rectangular aspect-ratio multiplier as discussed in Section 3, the radix $2^k$ digits $q_i$ and $b_i$ have to be available in radix 4 over the digit set $\{-2, -1, 0, 1, 2\}$. If $q_i$ is determined by table look-up, it is easy to provide $q_i$ in the appropriate representation and encoding.

In general, the multiplier $B$ is the result $S$ of a previous modular multiplication, hence $B$ is assumed to be in borrow-save (signed-digit, base 2) and needs to be recoded. Grouping pairs of signed digits, radix 2, directly provides a representation in maximally redundant radix 4, digit set $\{-3, -2, -1, 0, 1, 2, 3\}$, which can easily be converted into minimally redundant radix 4, digit set $\{-2, -1, 0, 1, 2\}$ with limited carry propagation (e.g. see [4]).

For the modular multiplication of Algorithm 3.1, the determination of $q_i$ severely limits the size of $k$, due to the number of leading digits of $S$ needed either for a table look-up (cf. (9)), or for multiplication by a pre-computed reciprocal of $m$. However, in the Montgomery method of Algorithm 4.4, $k$ digits of $S$ are sufficient, and thus by exploiting symmetry a table of size $2^k$ entries will suffice. Hence this method is feasible for larger values of $k$, but otherwise the two algorithms are very comparable in complexity as summarized in Table 1, where it is assumed that in the exponentiation of Algorithm 2.1 both products $y \times z$ and $z \times z$ are computed, although $y \times z$ may not be needed. Note that for $RSA$ encryption $e$ may be much smaller than $m$, but for decryption $e$ will then be of the same magnitude as $m$.

| | System Parameter | Cycles / mpy | # of mpy's in $x^e \bmod m$ | Register width |
|---|---|---|---|---|
| Alg. 3.1 | $n = \lceil \frac{\log_2 m}{k} \rceil$ | $n+2$ | $2\lfloor \log_2 e \rfloor + 2$ | $(n+2)k$ |
| Alg. 4.4 | $n = \lceil \frac{\log_2 m+2}{k} \rceil$ | $n+1$ | $2\lfloor \log_2 e \rfloor + 4$ | $(n+1)k$ |

Table 1: Complexity of $x^e \bmod m$ in radix $2^k$.

Let us now return to the organization of the computations in the individual cycles of the algorithms, in particular the possibilities of pipelining. Since $A$ is assumed to be represented in redundant binary (here assumed to be "borrow-save" encoding), $A$ can be represented as the difference of two words, $A = A_1 - A_2$. The updating of $S$ either requires three sequential multiply-adds (the S-Scheme, from [5]), or the three parallel multiplies in the form of a single larger adder tree (the P-Scheme), but the value of $k$ could be chosen about three times as large in the former case within the same area constraints.

In the S-Scheme each "rectangular aspect-ratio" multiplier must – within one clock cycle – be able to compute and accumulate the product of an $nk$-bit multiplicand by a $k$-digit (base 2, redundant) multiplier, where $nk$ is of the order 500–1000 and $k$ is of the order 2–10. Due to the need for distributing control over a very long word-size, the clock rate should not be too high, which implies that $k$ should be chosen as large as possible to reduce the number of cycles. Note that the determination of $q_i$ here can be overlapped with one of the multiplies. Given a fixed word-size $nk$, the area for the multiplier tree structure is essentially proportional to $k$, hence the choice of $k$ very much depends on what is affordable and feasible in the technology available. A radix 32 ($k = 5$) RSA implementation similar to the one described in [5], however using the P-Scheme, has been realized on a single chip, albeit fairly large (212 $mm^2$ in 1.2 micron).

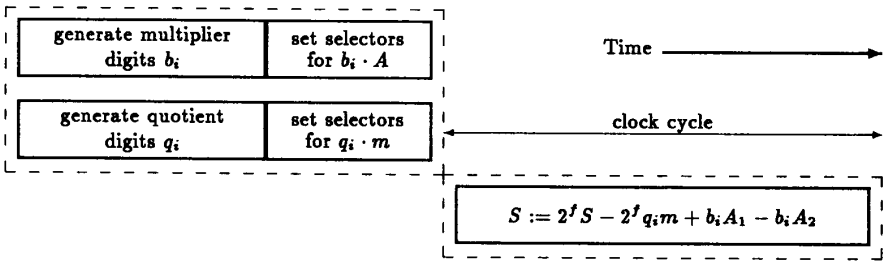


Figure 2: P-Scheme, permitting pipelining of two modular multiplications

For the P-Scheme, the three parallel multiplies in a single adder tree limits the value of $k$ substantially compared to the other scheme, which also implies that the determination of $q_i$ becomes simpler. Here it is not possible to overlap the q-determination with the multiplies of a single modular multiplication, but if the two multiplies of a modular exponentiation are interleaved in a pipeline, then overlap is possible as indicated in Figure 2.

For a final comparison, compare an S-Scheme implementation for $k = 6$ with a P-Scheme implementation with $k = 2$. The computation of $U$ in Figure 1 requires a 3-to-2 redundant adder with latched output, and the accumulation of $S$ requires a 4-to-2 adder operating in parallel with the other. The P-Scheme requires for $k = 2$ a single 5-to-2 redundant adder without internal latching necessary, hence requiring the same amount of 3-to-2 adders as the S-Scheme does for $k = 6$. The S-Scheme needs three clock cycles to complete one cycle of the algorithm, on the other hand it can potentially run at a higher clock rate. The most likely bottlenecks are the $q_i$-determination in the S-Scheme, and the adder tree in the P-Scheme.

## 6 Conclusions

This paper has presented an analysis of two algorithms, providing results needed in a careful study of the implementation of modular exponentiation. There is no doubt that the method of Montgomery, as described in Algorithm 4.4, leads to a much simplified selection of the reduction quotients $q_i$. It also seems reasonable to conclude that the S-Scheme for pipelining with a high radix is preferable to the P-Scheme using the comparably lower radix realizable within the same area, but final decisions for an actual VLSI implementation requires a more detailed design, together with timing simulations based on the available technology.

### Acknowledgements

The author wants to thank Holger Orup for fruitful discussions and the referees for very careful reviews and constructive comments.

## References

[1] Ernest F. Brickell. A Survey of Hardware Implementations of RSA. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89*, pages 368–370. Springer-Verlag, 1990.

[2] Peter L. Montgomery. Modular Multiplication Without Trial Division. *Mathematics of Computation*, 44(170):519–521, April 1985.

[3] A Proposed Federal Information Processing Standard for Digital Signature Standard (DSS). Tech. Rep. FIPS PUB XX, National Institute for Standards and Technology, August 1991. Draft.

[4] Peter Kornerup. Digit-Set Conversions: Generalizations and Applications. Technical report, Institut for Matematik og Datalogi, Odense Universitet, April 1992. Submitted.

[5] Holger Orup and Peter Kornerup. A High-Radix Hardware Algorithm for Calculating the Exponential $M^E$ Modulo $N$. In *Proc. 10th IEEE Symposium on Computer Arithmetic*, 1991, pages 51–66.

[6] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. A Method for Obtaining Digital Signatures and Public-key Cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[7] M. Shand, P. Bertin, and J. Vuillemin. Hardware speedups in long integer multiplication. In *Proc. 2nd Annu. ACM Symp. Parallel Algorithms and Architectures–SPAA '90*, pages 138–145, July 1990. Also in *Computer Architecture News*, Vol. 10, No. 1, pages 106-114, March, 1991.

[8] Naofumi Takagi. A Radix-4 Modular Multiplication Hardware Algorithm for Modular Exponentiation. *IEEE Transactions on Computers*, C-41(8):949–956, August 1992.

[9] André Vandemeulebroecke, Etienne Vanzieleghem, Tony Denayer, and Paul G. A. Jespers. A New Carry-Free Division Algorithm and its Application to a Single-Chip 1024-b RSA Processor. *IEEE Journal of Solid-State Circuits*, 25(3):748–755, June 1990.