Still Faster Modular Multiplication

by

Colin D. Walter

Computation Department, UMIST,

PO Box 88, Sackville Street, Manchester M60 1QD, U.K.

e-mail: cdw@sna.co.umist.ac.uk

Index Terms: *Computer arithmetic, cryptography, RSA, modular multiplication, redundant number systems.*

Abstract

By an appropriate choice of the modulus used in RSA cryptography, it is possible simplify the hardware for performing the required modular multiplication step, and thereby increase the speed of encryption and decryption. Here we consider this when P. L. Montgomery's algorithm is used.

1. Introduction

In earlier work by the author and Eldridge [5,6], it was shown how certain moduli M in RSA cryptography [1] enable simplification of the hardware for performing the modular multiplication step, and this in turn leads to an increase in the speed of both encryption and decryption. It is possible to make further improvements of the type seen in [5], using Montgomery's algorithm [3] for modular multiplication instead. Here we extend [6] using such techniques, and so familiarity with the detailed discussion in both might be helpful.

Modular multiplication is normally performed, as in ordinary multiplication, by keeping a running total, initially zero, which is repeatedly shifted and added to the product of the multiplicand by the next digit of the multiplier. This is reduced modulo M during each iteration in order to keep the partial product down to the size of M. The multiples of *M* subtracted on successive iterations are actually the digits of the integer quotient of the product by *M*. This is Brickell's approach [2].

The main bottle-neck in this computation is the calculation and broadcasting of the quotient digits to every digit position for every iteration. This can be solved using a number of techniques to simplify the hardware in order to reduce its critical path length. The methods in [6] were sufficient for moduli up to about 1000 bits in length. Here we perform operand scaling, or its equivalent, to extend the results to moduli of any size. As a consequence, almost all of the hardware is fully utilised almost all of the time, even for moduli of arbitrarily many digits. This means that no significant further improvement is likely without resorting to entirely different algorithms.

2. Montgomery's Algorithm

In Montgomery's algorithm for modular multiplication, the expected order of processing is reversed, with the digits of the multiplier being considered in the opposite order from usual, namely starting with the *lowest* digit. The shift is then *downwards* on each iteration, and a multiple of *M* is *added* in order to make this shift possible without a fractional part developing. This reversal of processing introduces an extra factor of 2^i where i is the number of iterations performed. In brief, to calculate $(A\times B)$ *mod M* up to this factor, the repeated operation required is:

$$R \leftarrow 2^{-1}R + aB + qM$$

where a is a digit of A, and q is a digit of the appropriately scaled integer quotient $(A \times B)$ *div M*.

Some pre- and post- processing is necessary to remove the factor 2^i and extract the true modular product (see [6]). However, one advantage of this method is that carries propagate *away* from the digits of *R* which are used to determine the quotient digits *q*, i.e. the multiples of the modulus to be added or subtracted. Furthermore, at the cost of a few more iterations, the multiplicand *B* can be shifted away from these digits in order to simplify the process of calculating quotient digits still further.

3. The Simplifications

As usual, a redundant representation of the numbers is employed to enable the addition cycle to be performed with digit parallel operations. *A* and *B*, which are normally results from previous modular multiplications, and the partial product *R*, are assumed to be of this form. We choose the digit range {0,1,2,3} for them, but note that, as *A* is consumed from the bottom up, it can be converted on the fly to non-redundant form so that the digit *a* used to form *aB* is just a single bit. The modulus *M* is known beforehand, and so can reasonably be assumed to be in the usual non-redundant binary form.

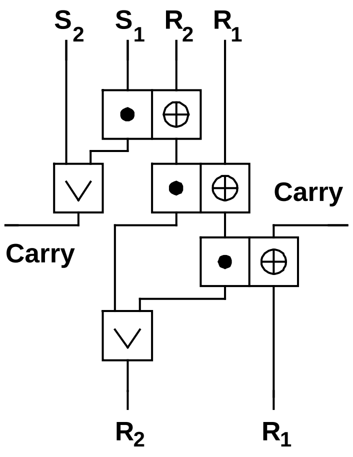

Figure 1 Bit Slice for Adder of depth 4.

As *A* and *B* are known initially, *aB* can be calculated as far in advance as necessary to speed the computation, although space may be required to buffer partial calculations before they are used. We will show below that *q* can be computed as long before it is needed as necessary. Thus *qM* can also be computed in advance, which means that *S* = *aB* + *qM* can be generated without delaying the main addition. Indeed, *S* can be given a representation with a digit range of just {0,1,2}. Now the addition cycle has been reduced to just

$$R \leftarrow 2^{-1}R + S$$

in which both *R* and *S* have (different) redundant representations. This can be done as in Figure 1 by an adder with a critical path length of only 4 gates, leading to a very short clock cycle. Moreover, if 3 bits are used for each digit of *R*, then the adder can be shortened to a mere 3 gates, as in Figure 2 – just half the depth of Brickell's adder [2].

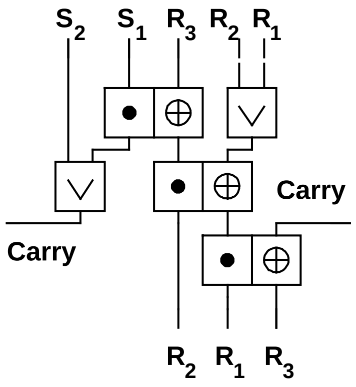

Figure 2 Bit Slice for Adder of depth 3.

Assume that the registers containing *A*, *B*, *M* and *R* all contain *n* digit positions. The computing of $S = aB - qM$ requires the broadcasting of digits *q* and *a* to each of the *n* digit positions using a tree of multiplexers and then the digit-parallel addition. This requires O(*log n*) time and O(*n*) area. So the digits *q* and *a* need to be available about *t* = O(*log n*) cycles, say, before the addition cycle in which the next value of *R* is calculated from $S = aB - qM$. This is achieved by modifying *B* and *M* so that their lowest *t* digits are always fixed, and known in advance.

First, we replace *B* by 2^tB so that its lowest *t* digits are all zero. The registers must thereby be increased in size by *t* digit positions over and above the number of digits in *M*. So the number of iterations in the modular multiplication must also be increased by *t*. This is only a marginal change in the space and time usage, and does not increase the post-processing.

Secondly, as *M* is odd, we could replace *M* by a multiple $M' = mM$ with the property $mM \equiv 1\ (mod\ 2^t)$. However, the result using the scaled modulus *M´* will need some further minor post-processing to reduce it modulo the original *M*. Alternatively, the modulus can be chosen to satisfy $M \equiv 1\ (mod\ 2^t)$ directly. The modulus used in the

RSA crypto-system is a product $p_1\, p_2$ of two primes, each typically of around 100 decimal digits in length. These primes are obtained by considering a sequence of numbers until one is found that, using an algorithm such as that of Solovay and Strassen [4], is likely to be prime with a given, very high probability. Any suitable choice for the first prime p_1 will determine the congruence which p_2 must satisfy, namely $p_2 \equiv p_1^{-1}$ ($mod\ 2^t$). The same algorithm can then be used to search for an appropriate value for the second prime.

A consequence of these simplifications is that the bottom t digits of R are determined merely by shifting down the bottom t+1 digits of its previous value. Since these digits do not change when S is added, they will be the digits q. Thus the digits q are indeed produced when required. Moreover, they can be obtained in non-redundant binary form: since the lowest t digits of R are initially 0, they have non-redundant form initially, and thereafter, the digit at position t+1 can be converted to non-redundant form, with its carry moving upwards, so that the non-redundancy property of the lowest digits is maintained.

4. Conclusion

To sum up, for *any* number n of binary digits in the modulus of the modular multiplication, we have described how to generate the digits required for the modular reduction steps without delaying the formation of the product. Thus modular multiplication suited for RSA may be implemented in $n + O(log\ n)$ clock cycles using P. L. Montgomery's algorithm and a clock cycle determined by an adder with a critical path length of only 3 gates. This adder has just half the depth of those used previously in the literature, and so leads to significantly faster performance.

References

[1] R.L. Rivest, A. Shamir and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” *Comm. ACM*, vol. **21**, 1978, pp. 120-126.

[2] E. F. Brickell, “A fast modular multiplication algorithm with application to two-key cryptography,” in *Advances in Cryptology - CRYPTO* ’82, Chaum et al., Eds., New York, Plenum, 1983, pp. 51-60.

[3] P. L. Montgomery, “Modular multiplication without trial division,” Math. Computation , vol. **44**, 1985, pp. 519-521.

[4] R. Solovay & V. Strassen, “A fast Monte-Carlo test for primality” SIAM J. Comput., vol. **6**, 1977, pp. 84-85.

[5] C. D. Walter, “Faster Modular Multiplication by Operand Scaling,” *Advances in Cryptology - CRYPTO ’9*1, Lecture Notes in Comp. Sci. vol. **576**, 1992, pp. 313-323, Springer-Verlag.

[6] C. D. Walter & S. E. Eldridge, “Hardware Implementation of Montgomery's Modular Multiplication Algorithm,” IEEE Trans. Comp. vol. **42**, 1993, pp. 693-699.