Space/Time Trade-offs for Higher Radix Modular Multiplication using Repeated Addition

by

Colin D. Walter

Computation Department, U.M.I.S.T.,

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, higher radix, optimal speed.*

Abstract

The value of using a higher radix for modular multiplication in the context of RSA is investigated. The main conclusion is that for algorithms which perform the multiplication via repeated addition, there is, broadly speaking, a direct trade-off between space and time provided by change of radix. Thus chip area utilised is roughly proportional to speed. However, initially as the radix is increased from 2, there is a short-lived increase in speed greater than the extra area used.

IEEE Transactions on Computers, vol. **46** no. 2, (Feb 1997), pp. 139-141.

1. Introduction.

Increasingly widespread use of highly secure cryptography methods have made necessary dedicated hardware for performing the intensive associated arithmetic. Many algorithms, such as the RSA scheme [6], use modular exponentiation, which is achieved through repeated multiplication. There are two standard algorithms for reducing modular multiplication to repeated addition (see [2], [5]) besides more complex methods such as the Discrete Fourier Transform [1]. In two recent papers, [7] and [8], Eldridge and the author have shown that, without compromising on chip area, the speed of the addition should be the limiting factor on the overall time of the standard algorithms. This contrasts with the simplest approaches whose time efficiency is bound by the intermediate calculation of the multiple of the modulus to subtract in the modular addition step. Both these papers concentrated on number representations with radix 2. This work applies these methods with other radices in order to quantify any benefits that might accrue from the use of a higher radix.

It is known that multiplication (and hence also modular multiplication) of k bit numbers can be performed in $log\ k$ time using $k^2/log\ k$ area [4]. As well as being complex, such time-optimal multipliers are currently too area-hungry for practical implementations of RSA cryptography. However, they can be used for the digit by digit products required in higher radix versions of the standard methods mentioned above. This then yields a very fast yet easily verifiable circuit in reasonable design time, although it is at the cost of the sub-optimal multiplication time taken by the standard algorithm. The algorithm provides products most quickly by tailoring the choice of radix to the maximum area available on the chip. However, we will see that increasing the radix is probably only worthwhile for low radices; large ones eventually make this style of modular multiplication too expensive in terms of area and design complexity. The only really tangible benefit over a number of base 2 or base 4 multipliers in parallel with the same area is probably the convenience of sequential output.

2. The Modular Addition Cycle.

At the heart of most implementations of the RSA cryptosystem [6] is an adder for performing the modular multiplication $(A \times B)\ mod\ M$ by the repeated operation:

$$R \leftarrow r^{\delta}R + aB - \delta qM$$

where $\delta = \pm 1$ depends on the choice of algorithm ($\delta = +1$ for the standard algorithm, $\delta = -1$ for Montgomery's algorithm [5]), r is the radix for the number representations, a is a digit of A, and q is a redundant digit chosen to keep R between 0 and $2M$. The multiplication or division by r is performed by a shift and, when $\delta = 1$, q is a digit of the integer quotient $Q = (A \times B)\ div\ M$. (A similar, though more complex, relation holds when $\delta = -1$.) The whole operation can be performed in one clock cycle if a redundant number system is used for R, so that carry propagation is bounded. We need not be precise about the digit range used, but will assume the extra redundancy is equivalent to that given by using one extra bit. Hence typical ranges might be $-r+1$ to $r-1$ or 0 to $2r-1$. For convenience, we will assume also that the radix is a power of 2, say $r = 2^k$, and we will use n for the number of base r digits in the hardware registers we need. Thus $n \approx \lceil log_r\ rM \rceil$, $k+1$ bits are required for a digit (although in places $2k$ are convenient), and the number of bits in the inputs, namely around nk, is constant for the circuits of interest.

In the context of RSA, various assumptions are possible. First, the numbers involved have many digits, being typically up to 2^{1000} in size. Secondly, since A and B are usually residues of previous modular multiplications, we assume that they are fully known initially and that they have the same redundant form as R. Thirdly, we assume that the modulus M is fixed for many consecutive multiplications in which outputs are used as the next inputs. Hence time can be spent in pre- and post- processing at each end of the sequence with very little affect on the overall cost. In particular, M may be scaled beforehand and have a non-redundant form, whilst the final output R may differ from the least non-negative residue by a small multiple of M.

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 $S = aB - \delta qM$ can be generated without delaying the main addition. With these assumptions, the addition cycle reduces to just

$$R \leftarrow r^{\delta}R + S$$

in which both R and S have redundant representations. An adder for this could propagate carries in the binary representation of the individual digits of the output to give R as a redundant number of radix r with digits in the required form. As this might require time but little extra area if any, we will just use $2k$ bits for each digit instead of k+1, so that both R and S basically consist of two binary numbers each. Then we require an adder which reduces four binary numbers to two. By slightly reconfiguring two full adders, this can be done with a critical path length of about 4 gates, leading to a very short clock cycle indeed. A and B will have the same form also. With the adder being so fully utilised throughout the multiplication, it is not possible to pipeline several calculations through this part of the hardware at once. Hence the timing results obtained will cover both latency and throughput.

3. Forming $aB - \delta qM$.

The hardware model which we use assumes bounded fan-in and fan-out, with all gates having $O(1)$ area and $O(1)$ time delay, and all wires having $O(1)$ width but conveying signals in zero time. We assume power, earth and timing signals are available at any point of the chip (e.g. through additional layers in the chip), so that no wiring overhead need be counted for any such connections which a gate requires. This is a good approximation to reality for circuits of a size appropriate to RSA cryptography in a technology where gate delays exceed signal propagation time along a chip-length wire. Hence, for arbitrary t, a 1:t bit multiplexer formed from a tree of multiplexers with bounded fan-out will have $O(t)$ gates, a critical path “length” of $O(log\ t)$, which is the time it takes to operate, and a wire length and area of $O(t\ log\ t)$. Thus the wires dominate the area, and it is hardly worth trying to recover any area from the obvious rectangular $t \times log\ t$ circuit layout. Such multiplexers account for the longest wires in the circuits we describe, but we will show how to compute their inputs far enough in advance to avoid any delay caused by wire length. Thus critical paths will involve only local connections, and the wire delay may then be ignored.

The computing of qM and aB requires the broadcasting of digits q and a to each digit position of M and B and then either the calculation of the digit multiple or its selection from a set of pre-calculated values. In either case, we need a tree of multiplexers to distribute q and a to the n digits. Since a digit has $O(k)$ bits, this will have critical path time $O(log\ n)$, $O(nk)$ area of multiplexers, and a wire length or area of $O(nk\ log\ n)$. This and the rest of the hardware for calculating $aB-\delta qM$ needs to be pipelined as the results from successive computations must be available at each clock cycle of the main adder.

If all digit multiples of M and B are pre-calculated, then $O(nkr)$ register area is required, and the selection of the right value takes $O(k)$ critical path depth, a further $O(nkr)$ gate area and a wire length, or area, of $O(nrk^2)$. This is the more expensive choice, as we see next.

On the other hand, suppose the multiples of M and B are calculated on the spot. Then only $O(nk)$ register area is required, and the calculation of the multiples takes the same additional time and area as $2n$ parallel products of two k-bit integers. It suffices to choose a pipelineable digit by digit multiplier with minimal area which generates one product per cycle as throughput. Thus we take a multiplier of area $O(k^2)$ per digit with a latency of $O(log\ k)$, pipelined to give a product every cycle. It can be defined simply by means of a divide-and-conquer method (see 3M in [3]) and yet be optimal with respect to $Time^2 \times Area$ without having to use the Discrete Fourier Transform. The overall digit multiplier area is just $O(nk^2)$, which is much less than that needed for pre-computed multiples. This then is the method to choose for computing the multiples of M and B.

Finally, the addition (or subtraction) in $aB-\delta qM$ takes a single clock tick and area $O(nk)$ by virtue of the redundant representation. Overall, for all digits together, the time taken from the generation of q and a to the completion of computing $aB-\delta qM$ is $O(log\ nk)$ clock cycles and it uses $O(\ nk^2 + nk\ log\ n\ )$ area.

4. Operand Scaling.

The calculations of the last section mean that the digits q and a need to be available t cycles, say, before $aB-\delta qM$ is needed, where t is $O(\log nk)$. Unexpectedly, but not surprising, t is essentially dependent only on the number of bits in the inputs, not on the choice of radix. In order to provide the quotient digit q t digits in advance, we need to scale and shift M and A. We describe the appropriate techniques in the case of the classical algorithm (i.e. $\delta = +1$). For Montgomery's reformulation [5] (i.e. $\delta = -1$), corresponding suitable techniques have been described by the author and Eldridge in [8].

First of all, following [7] §3, we choose m so that mM has a most significant digit 1 followed by at least t digits 0. This is achieved by choosing m to be the first $t+1$ (or so) significant digits of M^{-1}, rounded up. With the scaled modulus mM, the residue R will now be bounded by $2mM$. Hence about t cycles of additional post-processing will be necessary after all the modular multiplications are done in order to obtain the least non-negative residue $mod\ M$. Before such post-processing, we can expect to use R for the inputs A and B of the next modular multiplication, so that they too will be up to $2mM$ in size.

We next scale A and mM by shifting both up $t' \approx t+2$ places, so that we are now computing $(Ar^{t'} \times B)\ mod\ (r^{t'}mM)$ and then shifting the result down by t' places to compensate. Let $M' = r^{t'}mM$ be the new modulus. The consequence of the scaling and shifting is that the term aB makes no contribution to the computation of R in the region of the topmost $t+1$ digits of M'. Moreover, the form of M' means that the term qM' only affects the topmost digit of R in this region. The digits q are chosen to keep R with fewer digits than M', though it may sometimes become slightly negative as a result. Initially, $R = 0$ so that such a property holds. Thereafter, q is picked equal to the digit of rR in the same position as the top digit of M'. So $rR + aB - \delta qM'$ still has fewer digits than M' as carry propagation is limited. Thus the new value for R will have the same property again. Indeed, if we are careful enough with carry propagation, then the top $t-1$ digits of R are now the ones which, with q, were the top t digits of the previous value of R. Hence the top t digits of R are just the next t values for q, and so the values of q are generated as early as required.

Scaling and shifting therefore solve the main problem which slows up most implementations. The cost of this is an increase in the time by an additional $O(\log nk)$ clock cycles per modular multiplication and an increase in area given by having $O(\log nk)$ extra digits, together with some minor extra post-processing after all the modular arithmetic. These increases are small compared to the total time and area requirements as long as $O(\log nk) \le O(n)$, i.e. if n is not too small. In such cases the overall order of area and time are as calculated in the previous section.

5. Time × Area.

When n is not too small, i.e. $n \ge O(\log nk)$, §3 gives the whole area of the adder to be $O(\ nk^2 + nk \log n\ )$ and the time required for a modular multiplication is $O(n)$. If n is not too large either, specifically if $\log n$ is at most $O(k)$, then this area simplifies to $O(nk^2)$, and so $Time \times Area$ is $O(n^2k^2)$. Now suppose the inputs are of fixed size so that nk, essentially the number of bits in M, is fixed. Then, except for extreme values of n, $Time \times Area$ is constant to a first approximation, no matter what choice of radix is made.

For large radix, i.e. small n, the design is not of any value because the time penalty for producing the digits q in time is too great. However, for small radix, i.e. k less than $O(\log n)$, the $\log n$ term in the area dominates the expression, so that $Time \times Area$ is roughly $O(n \log n)$, which increases with n. Thus, for such k, increasing the radix provides a more efficient algorithm at least initially. Now this dominant term corresponds to the part of the circuitry for distributing the digits q and a as opposed to that for computing the digit product. So the constant implied by the O notation is probably relatively small for the $nk \log n$ term compared with that for the nk^2 term. This means that the penalty for using a very small value for k (i.e. a small radix) is likely to be short-lived.

We have not considered here the relative advantages of different redundant representations. However, by observing that less extra area is required to represent minimal redundancy as the radix is increased, it is likely that for such representations, $Time \times Area$ would decrease slowly at least initially as the radix is increased. This provides an added, though diminishing, incentive to increase the radix away from 2.

6. Conclusion.

Overall our main conclusion is that, to a first approximation, $Time \times Area$ is inherently constant for this algorithm; that is, $Time \times Area$ is independent of the choice of radix for the best implementations of the standard method of modular multiplication by means of repeated addition. This result holds assuming the most efficient implementation of digit by digit products. Since the design described here is parametrised by the radix, the best speed is obtained easily just by choosing the largest radix for which sufficient chip area is available.

Our other conclusion is that, notwithstanding that $Time \times Area$ is approximately constant, there is an initial, though short-lived improvement in efficiency when the radix is moved away from 2. Thereafter, placing a number of modular multiplication processors in parallel on a chip would provide similar throughput for the same cost in terms of area, but without the advantage of the above scheme which produces its output sequentially.

References.

[1] R. P. Brent & H. T. Kung, "The Area-Time Complexity of Binary Multiplication," J. ACM, vol. **28**, 1981, pp. 521-534.

[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] W. K. Luk & J. E. Vuillemin, "Recursive Implementation of Optimal Time VLSI Integer Multipliers", VLSI '83, F. Anceau & E.J. Aas (eds.), Elsevier Science, 1983.

[4] K. Mehlhorn & F. P. Preparata, "Area-Time Optimal VLSI Integer Multiplier with Minimum Computation Time", Information & Control, vol. **58**, 1983, pp. 137-156.

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

[6] 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.

[7] C. D. Walter, “Faster Modular Multiplication by Operand Scaling,” *Advances in Cryptology* - *CRYPTO ’91*, Lecture Notes in Comp. Sci. vol. **576** (1992), pp. 313-323, Springer-Verlag.

[8] C. D. Walter & S. E. Eldridge, “Hardware Implementation of Montgomery's Modular Multiplication Algorithm,” IEEE Trans. Comp. vol. **42**, 1993, pp. 693-9.

4th May 96