Iterative modular multiplication algorithm without magnitude comparison

C.W. Chiou and T.C. Yang

Indexing terms: Digital arithmetic, Number theory

A fast iterative modular multiplication algorithm is proposed for modular exponentiation with a large modulus, such as the RSA cryptosystem. The limit on partial products is raised to become less than 2^n instead of modulus N with n-bit length. The naturally generated carry signifies when the computed partial product is greater than or equal to 2^n, and a subtraction is subsequently performed. No magnitude comparisons are required.

Introduction: With the increasing importance of computer security, computer cryptography is becoming more and more important. Most number-theoretic cryptosystems, such as the RSA cryptosystem [1], are constructed based on modular multiplication. The design of a fast algorithm for modular multiplication with a large modulus (longer than 500 bit) is the key to developing high-speed encryption/decryption algorithms. One of the most attractive methods is one in which each subtraction step for division is embedded in the repeated addition-multiplication algorithm [2–4]. This technique computes the product in n steps for n-bit modulus N, where at each step one left shift, one addition, and at most two magnitude comparisons (i.e. subtractions) are performed. However, the magnitude comparison slows down for longer operands (for example, 644 bit) because its computation time is linearly proportional to the word length of operands. The key point to increasing the computation speed is to keep the number of required magnitude comparisons as small as possible. Based on the Blakley algorithm [2], an iterative modular multiplication algorithm without magnitude comparison is proposed.

Algorithm: We consider the computation of $A*B$ modulo N. Let A, B, and N be three n-bit positive binary integers, where n is very large and A, $B < N$. Let B at the bit level be represented by $b_{n-1}b_{n-2} \ldots b_1b_0$.

A traditional algorithm based on the Blakley algorithm for the computation of $A*B$ modulo N is as follows:

Algorithm A: (Traditional algorithm for computing $R = A*B \bmod N$)

```
P = 0
for j = n–1 down to 0 do
  begin
   P = 2*P              (* shift-left the partial product *)
   if (P ≥ N) then P = P–N    (* magnitude comparison is
                                 performed. *)
   if (b_j = 1) then
     begin
       P = P+A
       if (P ≥ N) then P = P–N (* magnitude comparison is
       performed *)
     end
  end
```

where the partial sum P is $(n+1)$ bit long. Each iteration in the loop performs a left shift and an addition of a n-bit number. This requires up to two subtractions of the modulus. On average, the algorithm takes a total of n left-shift operations, $3n/2$ subtractions, and $n/2$ additions. We assume that both addition and subtraction requires similar time complexity and the shift operation is negligible as compared with additions. Thus, the traditional algorithm requires $2n$ additions on average.

If P is greater than or equal to N, subtraction of N is performed. In this algorithm, an accurate magnitude comparison for possible subtraction of N is required. Owing to the carry-propagation problem, it takes $O(n)$ gate delays and limits the operation speed as n becomes large. To avoid the time-consuming magnitude comparison operations, we propose an algorithm to limit the value of the partial sum P to be less than 2^n instead of N. A carry generated in normal addition signifies the condition that the computed number is greater than or equal to 2^n. If this is the case, the generated carry is discarded and an addition of $(2^n \bmod N)$ to P is then performed. The new algorithm is described in the following paragraphs.

Algorithm B: (* compute $R = A*B \bmod N$ *)

```
(* let P be the partial product *)
(* let carry(P) represent a carry generated by computation of P *)
P = 0; S = 2^n mod N (* first, compute 2^n mod N *)
for i = n–1 down to 0 do
   begin
    P = 2*P              (* shift-left the partial product *)
    if carry(P) then P = P+S–2^n (* a carry-out indicates the case
    P ≥ 2^n *)
    if (b_j = 1) then
     begin
       P = P+A
       if carry(P) then P = P+S–2^n (* a carry-out indicates the
       case P ≥ 2^n *)
     end
   end
if (P ≥ N) then R = P–N else R = P
```

When a carry, i.e. carry(P), is generated, a precomputed value, $2^n \bmod N$, is added to the partial sum P. Therefore, no magnitude comparison with the modulus N is required in this algorithm. On average, $(5n/4 + 1/2)$ additions are performed for the computation of $A*B$ modulo N. In fact, the number of additions can be further reduced, as shown in the following algorithm.

Algorithm C: (* compute $R = A*B \bmod N$ *)

```
P = 0; c = 0; S1 = 1*2^n mod N; S2 = 2*2^n mod N; S3 = 3*2^n mod N
T1 = (2^n+A) mod N; T2 = (2*2^n+A) mod N; T3 = (3*2^n+A) mod N
for i = n–1 down to 0 do
  begin
   P = 2*P
   if (carry(P) = 1) then c = c+1
            (* a carry out indicates that 2^n mod N is added to P *)
   if (b_j = 1) then (* b_j = 1 indicates that A is added to P *)
    begin
     case c of
       0: P = P+A; 1: P = P+T1; 2: P= P+T2; 3: P= P+T3
     end case
    end
   else
    begin
     case c of
       1: P = P+S1; 2: P = P+S2; 3: P = P+S3
     end case
    end
   if carry(P) then c = 2 else c = 0
      (* a carry out indicates that 2*2^n mod N is added to P *)
  end
if P ≥ N then R = P–N else R = P
```

In algorithm C, the expected number of additions is $(n+10)$. The addition of the 2 bit counter c is negligible as compared with the addition of the partial sum P. Compared with the traditional algorithm, algorithm C provides a speedup ratio of 2.

Conclusion: Based on the Blakley algorithm, an iterative algorithm for the computation of $A*B$ modulo a large modulus N (larger than 500 bit) has been proposed. No magnitude comparisons are required in our new algorithm. It is also shown that the new algorithm has a speedup ratio of 2 compared with the conventional algorithm.

6 September 1994
Electronics Letters Online No: 19941383

G. W. Chiou (*Chung Shan Institute of Science and Technology, Taiwan, Republic of China*)

T. C. Yang (*Department of Information Engineering, Feng Chia University, Taiwan, Republic of China*)

T. C. Yang: Corresponding author

References

1 RIVEST, R.L., SHAMIR, A., and ADLEMAN, L.: 'A method for obtaining digital signatures and public-key cryptosystems', *Commun. ACM*, Feb. 1978, **21**, (2), pp. 120–126
2 BLAKLEY, G.R.: 'A computer algorithm for calculating the product *AB* modulo *M*', *IEEE Trans.*, May 1983, **C-32**, (5), pp. 497–500
3 SLOAN, K.R.: 'Comments on: 'A computer algorithm for calculating the product *AB* modulo *M*'', *IEEE Trans.*, 1985, **C-34**, pp. 290–292
4 BAKER, P.W.: 'Fast computation of *A*B* modulo *N*', *Electron. Lett.*, 1987, **23**, (15), pp. 794–795

Efficient IQ filter structure for use in adaptive equalisation

D.R. Bull

Indexing terms: Adaptive filters, Equalisers

A method is presented for reducing the implementation cost of a complex digital filter structure such as that used for IQ processing in adaptive equalisation. It is shown that the number of real filter sections required can be reduced from four to three at the expense of an increase in external addition stages from two to three (or five if coefficient additions are included). The approach is applicable to both LTE and DFE structures and results in savings which approach 25% for most practical cases.

Introduction: Adaptive equalisation techniques are frequently employed in digital communication systems to combat the inter-symbol interference caused by multipath effects in the channel [1]. Numerous equalisation structures have been proposed ranging from the simplest linear transversal structure (LTE) to decision feedback equalisation (DFE) [2], Viterbi and neural network based solutions [3]. DFE is generally preferred to LTE because it is capable of reducing ISI without significant noise enhancement in a frequency selective fading channel. Also the complexity of the Viterbi and neural network approaches coupled with the convergence properties of certain neural network paradigms often preclude their practical adoption.

In practice, most modulation schemes employ both in-phase and quadrature components (e.g. QAM or multiphase PSK) and therefore require an equaliser structure capable of processing complex signal values. In its basic form the complex equaliser is equivalent to a parallel structure comprising four real filter structures as shown in Fig. 1.

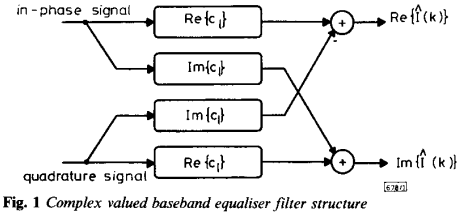

Fig. 1 *Complex valued baseband equaliser filter structure*

The computational complexity of the receiver architecture is of paramount importance especially in mobile handsets where good quality communications must be provided at low cost and power. To this end it is advantageous to exploit any hardware complexity savings which can be obtained, for example in areas such such as the equaliser. This Letter presents an approach which enables the complex filtering operation in an equaliser to be realised using three real signal convolvers rather than the four shown in Fig. 1. This results in a hardware saving for VLSI, with associated power and cost reductions, or alternatively a throughput improvement for programmable DSP micoprocessor solutions.

Complex multiplication: The dominant component in an equalisation filter for I and Q signals is the complex multiplier. In the standard form of eqn. 1, a complex multiplication operation requires four real multiplications and two real addition/subtractions.

$$\begin{aligned} v+jw &= (g+jh)(x+jy) \\ &= (gx-hy)+j(hx+gy) \end{aligned} \quad (1)$$

The complexity of performing multiplication is generally considered to be $O(B^2)$, where B represents the internal wordlength of the process whereas that for additions or subtractions is $O(B)$. This observation has led a number of researchers to investigate methods for reducing the number of real multipliers required to compute eqn. 1, possibly at the expense of extra addition/subtraction operations. Several three-multiplier forms of the complex multiplier have been reported in the literature. For example Preuss in [4] gives

$$v+jw = \{gx-hy\}+j\{(g+h)(x+y)-gx-hy\} \quad (2)$$

whereas Winograd [5] gives

$$v+jw = \{(g+h)x-h(x+y)\}+j\{(g+h)x-g(x-y)\} \quad (3)$$

Horrocks and Bull [6] use a generic structure to synthesise all possible three multiplier versions (12 in total) of the complex multiplier and classify these according to coefficient combinations. Three examples of these are given in eqns. 4–6. It can be seen that all structures require five addition/subtractions, in comparison with the two required for the four multiplier case.

$$v+jw = \{g(x-y)+(g-h)y\}+j\{(g-h)y+h(x+y)\} \quad (4)$$

$$v+jw = \{g(x+y)-(g+h)y\}+j\{(g+h)y+h(x-y)\} \quad (5)$$

$$v+jw = \{g(x-y)+(g-h)y\}+j\{(g+h)x-g(x-y)\} \quad (6)$$

Fam [7] has extended this approach to the case of complex matrix multiplication where further savings are possible.

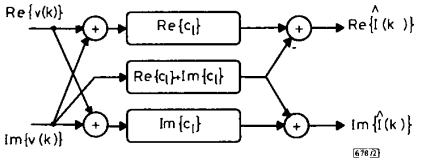

Fig. 2 *Example three filter version of complex equaliser*

Complex convolution: Although in the case of the equalisation filter we are concerned with convolution rather than multiplication, because convolution is a linear operator, the principles outlined above for complex multiplication still apply.

In the simplest case of an Nth order linear transversal equaliser, the symbol estimate $\hat{I}(k)$ at the kth signalling interval is a weighted linear combination of previous outputs $\{v(k)\}$ as given in eqn. 7.

$$\hat{I}(k) = \sum_{l=-N/2}^{N/2} c_l v(k-l) \quad (7)$$

where $\hat{I}(k)$, c_k and $v(k)$ in eqn. 7 are all complex quantities. Conventionally eqn. 7 will be realised either with each complex multiplier implemented using four real multipliers or equivalently, as a parallel combination of four independent filter sections as given by eqn. 8 and previously shown in Fig. 1. The computational complexity of both of these solutions is identical.

$$\begin{aligned} &\mathrm{Re}\left\{\hat{I}(k)\right\}+j\mathrm{Im}\left\{\hat{I}(k)\right\} \\ &= (\mathrm{Re}\{v(k)\}+j\mathrm{Im}\{v(k)\}) * (\mathrm{Re}\{c_k\}+j\mathrm{Im}\{c_k\}) \\ &= (\mathrm{Re}\{v(k)\} * \mathrm{Re}\{c_k\}-\mathrm{Im}\{v(k)\} * \mathrm{Im}\{c_k\}) \\ &\quad +j\,(\mathrm{Re}\{v(k)\} * \mathrm{Im}\{c_k\}+\mathrm{Im}\{v(k)\} * \mathrm{Re}\{c_k\}) \end{aligned} \quad (8)$$

This latter approach can be modified, in the same manner as shown previously, to yield three-filter realisations of eqn. 7. An example of this which corresponds to eqn. 5 above is given in eqn. 9 and is shown diagrammatically in Fig. 2.

$$\begin{aligned} &\mathrm{Re}\left\{\hat{I}(k)\right\}+j\mathrm{Im}\left\{\hat{I}(k)\right\} \\ &\quad = (\mathrm{Re}\{c_k\} * (\mathrm{Re}\{v(k)\}+\mathrm{Im}\{v(k)\}) \end{aligned}$$