A Systolic, Linear–Array Multiplier for a Class of Right-Shift Algorithms

Peter Kornerup*
Dept. of Mathematics and Computer Science
Odense University
DK–5230 Odense M, Denmark

Abstract

A very simple multiplier cell is developed for use in a linear, purely systolic array forming a digit-serial multiplier for unsigned or 2'complement operands. Each cell produces two digit-product terms and accumulates these into a previous sum of the same weight, developing the product least significant digit first. Grouping two terms per cell, the ratio of active elements to latches is low, and only $\lceil \frac{n}{2} \rceil$ cells are needed for a full n by n multiply. A modulo-multiplier is then developed by incorporating a Montgomery type of modulo-reduction. Two such multipliers interconnect to form a purely systolic modulo exponentiator, capable of performing RSA encryption at very high clock frequencies, but with a low gate count and small area. It is also shown how the multiplier, with some simple back-end connections, can compute modular inverses and perform modular division for a power of two as modulus.

Keywords: Systolic array, digit-serial multiplier, 2'complement multiplication, modulo-multiplication and exponentiation, RSA, modular inverse, modular division, exact division, Hensel codes.

1 Introduction

Motivated by the wish for a simple systolic structure to be used in a modulo multiplier for cryptosystems, a serial multiplier is first developed. Usually the digit product terms in multipliers are computed either along rows or along columns of the total array of terms, but here we have chosen to compute the terms along slanted lines, moving across the array. Computing two terms from each column in each cycle, only $\lceil \frac{n}{2} \rceil$ cells are needed. By gradually right-shifting, a linear array of cells produces the result in $2n$ cycles, each cell having only nearest neighbor communication, thus it is suitable for the very large multipliers (500-1000 bits) needed in cryptosystems. The array is developed as a parallel-by-serial multiplier; however it can be modified to a serial-by-serial structure, but then the multiplicand digits will have to be delivered at twice the speed of the multiplier digits.

Several other serial multipliers have been developed in the past, the one closest to the one developed here seems to be the classical design by Atrubin [Atr65] which is of the type serial-by-serial and also has a linear, systolic structure, but with a much higher cell complexity.

*This work has been supported by the Danish Research Councils, Grant No. 5.21.08.02.

Although the multiplier is developed for unsigned operands and in any radix, it turns out that it is very simple to modify the binary version to operate on 2'complement numbers. Hence it could be useful in standard applications for serial multipliers, e.g. in digital signal processing. However, the intended applications are in number-theoretic computation, primarily for cryptosystems, but other applications are possible and are demonstrated. The intended application of modulo multiplication is supported by adapting the cell to generate and accumulate two additional terms, corresponding to partial products of the modulus to be added (subtracted). By suitably selecting factors of the modulus (quotient digits), the steps of modular reduction are thus interleaved with the multiplication. The result is a linear array modular multiplier much simpler than the 2-dimensional systolic structure proposed in [Wal93].

This paper is organized as follows. In Section 2 the cell and the basic multiplier structure is developed from the dependency graph. After some comments on its operation, it is shown how it can be modified to accept 2'complement operands.

In Section 3 the cells are modified to allow the interleaving of multiplication and modulo reduction. The reduction is based on the idea by Montgomery [Mon85], where quotient digits can be determined from the least significant digits of the partial products. Since the reduction step is a shift-and-add step very similar to the basic steps of a multiplication, it is then simple to extend the cell to combine the two operations. To perform the modular exponentiations needed in cryptosystems, two linear arrays of such cells are then combined into one structure, capable of computing two modulo-multiplications in parallel, sharing a common modulus. By related computations of $yz2^{-n} \bmod m$ and $zz2^{-n} \bmod m$, the exponentiation $x^e \bmod m$ can be performed.

As an illustration of other applications of the basic shift-and-add structure of the multiplier array, in Section 4 two simple algorithms are shown to compute $a^{-1} \bmod 2^k$, respectively $da^{-1} \bmod 2^k$. Both algorithms can then be implemented in the array by some simple modifications to the right-most cell. For the particular case of the modulus being a power of 2, these simple algorithms can substitute for the Extended Euclidean Algorithm which would otherwise be needed in the case of a more general modulus. Very recently Jebelian [Jeb93] has described similar algorithms for these problems, but without specific systolic designs.

Finally, in Section 5 some general comments and conclusions are provided.

2 A Bit-Serial, Purely Systolic Multiplier Structure

Looking for a way in which a purely systolic, linear array of cells could be organized for a bit-serial multiplier, let us initially consider the array of partial product terms of the product AB in base β where $A = \sum_0^{n-1} a_i\beta^i$ and $B = \sum_0^{n-1} b_i\beta^i$, as illustrated in Figure 1 for the case of $n = 5$.

In usual linear-time, parallel multipliers, the terms a_ib_j are constructed row by row, and suitably accumulated in a linear array of adders, requiring the broadcast of b_j to n cells. Dadda types of serial multipliers (e.g. [Dad83]) operate columnwise, starting at the least significant end. Only one new digit of the multiplier and of the multiplicand is introduced in each step, but it seems difficult to organize the generation and accumulation of the terms in a purely systolic structure.

However, if in each step terms along a slanted line as indicated in Figure 1 are constructed, it is straightforward to accumulate the terms in an array of cells containing previous terms of the same weight. And again here only one new digit from multiplier and multiplicand is introduced in each step. If one term a_ib_j from each column is computed and accumulated per step, then $\left\lceil \frac{n}{2} \right\rceil$ cells and $3n-2$ steps are needed. If two terms from each column are formed

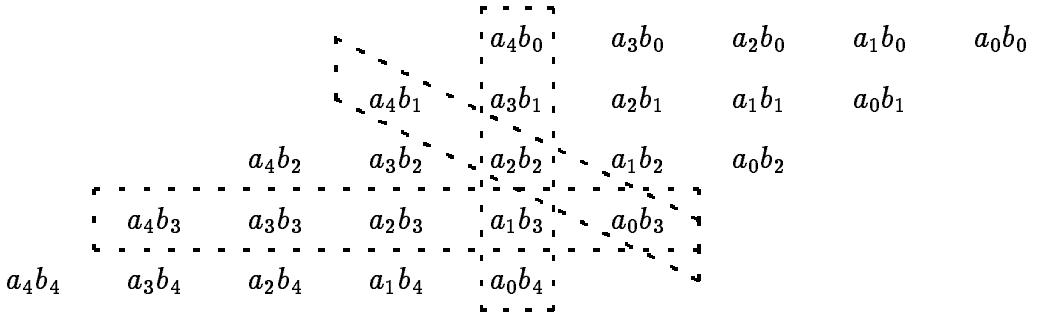


Figure 1: Array of partial product terms.

and accumulated per step, then also $\left\lceil \frac{n}{2} \right\rceil$ cells are needed, but now only $\left\lceil \frac{3n-1}{2} \right\rceil$ steps are necessary to generate all digit product terms. However, $2n$ steps are needed to generate all result digits serially, so to further decrease the number of cells and steps, it then seems more advantageous to combine a low number of terms with a higher radix, rather than generating three or more terms from each column.

We will thus concentrate on the case of using two terms from each column, and restrict the discussion to radix 2 since modification to a higher radix is straightforward. In Figure 2 a part of an array of partial products is shown, with pairs of terms enclosed in dashed boxes. The terms in boxes connected by the slanted lines are the terms to be computed and accumulated in a particular time step.

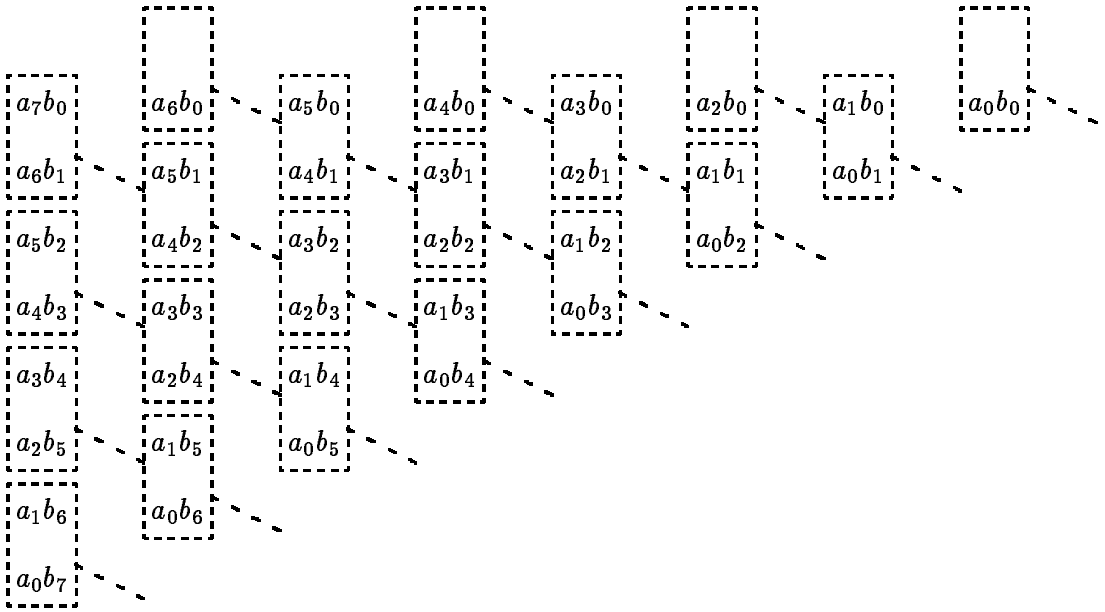


Figure 2: Grouping of terms by two from each column.

Using a suitable redundant representation for the accumulation it is possible to start generating and accumulating terms from the most significant end (e.g. for an on-line multiplier), but here we will choose to start at the least significant end. Note that in Figure 2, in each cycle one new multiplier digit plus two new multiplicand digits are needed, and one digit of the result can be produced.

To derive recursive equations for the computation we start by noting that at time step

t, for $i = 0, 1, \ldots, \min\left(\left\lceil \frac{n}{2} \right\rceil - 1, t\right)$ we compute the term

$$d_{it} = a_{2i} b_{t-i} + a_{2i+1} b_{t-i-1} \quad \text{of weight } 2^{i+t}$$

which is then accumulated in:

$$S_i^{(t)} = S_{i+1}^{(t-1)} + d_{it}.$$

To derive a systolic structure we rewrite this into the recursive equations:

$$\begin{aligned} S(i,t) &= S(i+1,t-1) + A(2i,t-1)B(i-1,t-1) + A(2i+1,t-1)B(i,t-1) \\ A(2i,t) &= A(2i,t-1), \quad A(2i+1,t) = A(2i+1,t-1) \\ B(i,t) &= B(i-1,t-1) \end{aligned}$$

for $0 \le i \le \left\lceil \frac{n}{2} \right\rceil - 1$ and $t \ge 0$, with boundary conditions:

$$\begin{aligned} S(i,-1) &= 0 \quad \text{for all } i \\ S(\left\lceil \tfrac{n}{2} \right\rceil, t) &= 0 \quad \text{for all } t \\ A(i,-1) &= a_i \quad \text{for } 0 \le i \le n-1 \\ B(-1,t-1) &= b_t \quad \text{for } 0 \le t \le n-1 \\ B(i,-1) &= 0 \quad \text{for } i \ge 0 \end{aligned}$$

and results in $S(0,t)$ of weight 2^t. The final result is then $\sum_{t=0}^{2n-1} S(0,t)2^t$, but note that we still have to consider how to handle the problem of carries. It could be handled by modification of the recursion, but we shall see that we can easily incorporate the carries later.

We can then draw the dependency graph as shown in Figure 3 for the case $n = 4$.

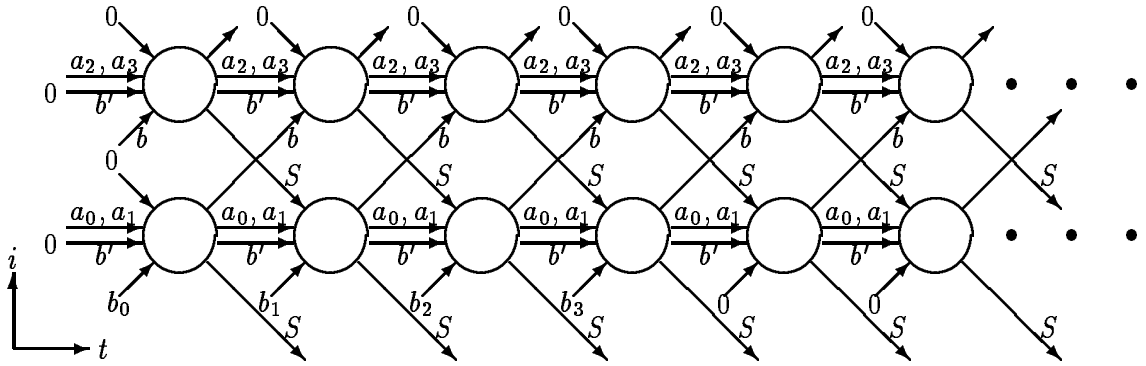
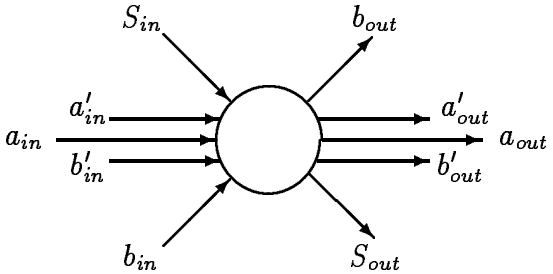


Figure 3: Dependency Graph, $n = 4$.

The graph is based on cells of the form:



with the following in-out relations:

$$\begin{aligned} S_{out} &:= S_{in} + a_{in}b_{in} + a'_{in}b'_{in} \\ a_{out} &:= a_{in} \\ a'_{out} &:= a'_{in} \\ b'_{out} &:= b_{out} := b_{in}. \end{aligned}$$

Projecting the graph in the direction of t yields the systolic structure of Figure 4, when furthermore the accumulation in S is expanded, using two full adders and two latches to retain the values of carries from one step to the next:

$$s_i^{(t)} + 2(c_{1,i}^{(t)} + c_{2,i}^{(t)}) := s_{i+1}^{(t-1)} + c_{1,i}^{(t-1)} + c_{2,i}^{(t-1)} + a_{2i}b_{t-i} + a_{2i+1}b_{t-i-1}.$$

i.e. $S_i^{(t)}$ is in an "extended" carry-save representation with two carries, $S_i^{(t)} = s_i^{(t)} + 2(c_{1,i}^{(t)} + c_{2,i}^{(t)})$.

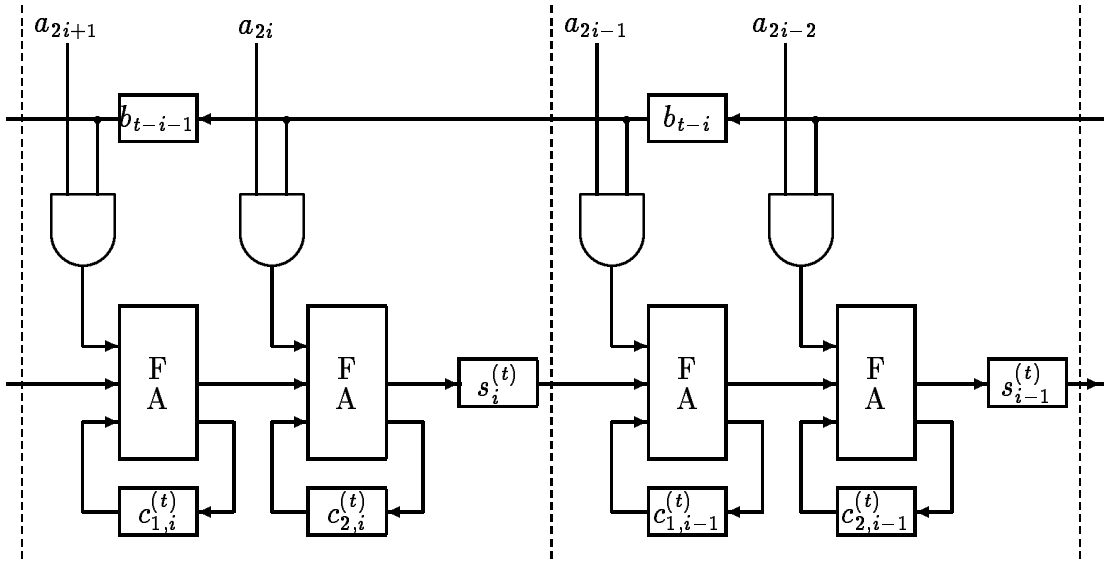


Figure 4: Systolic Multiplier Cells

A few remarks are due here. What has been obtained is a purely systolic serial-parallel multiplier consisting of $\lceil \frac{n}{2} \rceil$ cells. However, a serial-serial multiplier can be obtained by delivering the multiplicand bits a_i sequentially at twice the speed of the multiplier bits b_j. Initially all latches are assumed reset. During the first n cycles $b_0, b_1, \ldots, b_{n-1}$ are delivered, and the first n (least significant) bits $s_0, s_1, \ldots, s_{n-1}$ are produced. During the next n cycles for an unsigned multiply, zeroes have to be supplied for $b_n, b_{n+1}, \ldots, b_{2n-1}$, while the remaining results $s_n, s_{n+1}, \ldots, s_{2n-1}$ are output. In the case of a signed 2'complement multiplier, the value of b_{n-1} just has to be supplied as the sign-extension during the last n cycles. A 2'complement signed multiplicand can be accepted by a modification of the leftmost cell, as shown in Figure 5 for the case of n even, $n = 2k$.

The leftmost carry-latch is to be initialized by the value of the most significant bit of the multiplicand, a_{2k-1}. Since $b_i = 0$ for $i < 0$, the latch effectively remembers the value of a_{2k-1} until the first non-zero b_i arrives, when it is then added in. As the terms added into this position are $a_{2k-1}\overline{b}_{t-k}$ a Baugh and Wooley Scheme is thus realized.

It is easy to see that the same structure can be used at a higher radix also. Assuming a digit set $\{d \mid 0 \leq d \leq \beta - 1\}$ is used for radix β, then the carry that has to be retained is at most $2(\beta - 1)$.

The time for a complete n by n multiplication is $2n$ cycles, but due to the very local communication it can operate at a high frequency. Note that $2n$ cycles are needed although

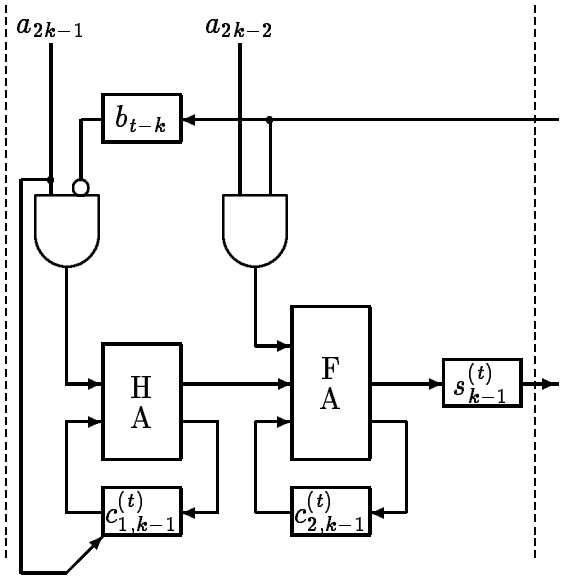


Figure 5: Leftmost cell for 2'complement multiplicand.

only $\left\lceil \frac{3n-1}{2} \right\rceil$ cycles are necessary to generate terms, the extra cycles needed to empty the array and complete the carry ripple.

This multiplier shows some similarity with the Atrubin serial multiplier [Atr65] in forming two partial product terms per cell. However, the Atrubin multiplier cell number i stores two pairs of operand bits $(a_{2i}, b_{2i}), (a_{2i+1}, b_{2i+1})$ plus a transient pair (a_{t-i}, b_{t-i}) together with sum and carry information. Thus the amount of state information in each cell is much larger than in this design (about 11 or 13 bits vs. 6).

3 Systolic Modular Exponentiation

In many cryptographic applications, like RSA and other two-key systems, modulo-multiplication is a fundamental operation for the implementation of the compute intensive modular exponentiations needed. Due to the extremely long operands (500-1000 bits), systolic structures with pure nearest-neighbor communication will be advantageous if very high clock frequencies are wanted to achieve speed.

The modular reductions needed for modulo-multiplication can be arranged to fit the basic shift-and-add structure of the above systolic multiplier. Based on the idea of P. Montgomery [Mon85], the decision on what multiple of the modulus to subtract can be based on least significant digits of the product. Hence his method can easily be adopted to a serial multiplier as described above.

The idea is to change to a different residue system, using residues of the form $r = u\beta^k \bmod m$, where m is the very large odd integer modulus, β is the base of the arithmetic with $gcd(m, \beta) = 1$ and $\beta^k > m$. We will here restrict the discussion to the case $\beta = 2$ which is particularly simple to describe and implement. Choosing higher values of β is possible, corresponding to the choice of radix for the multiplication. In particular, choosing $\beta = 4$ is also sufficiently simple to implement, allowing a selection of the reduction factor to be made at speed matching the speed of the cells.

We shall not go into details and formal properties of the algorithm here, but refer the reader to other descriptions, e.g. [Mon85], [SV93], [Kor93]. For the case $\beta = 2$ the algorithm for computing $S = [ab]_m$ given $A = [a]_m$ and $B = [b]_m$ ($[\cdot]_m$ is the special Montgomery residue) is:

Algorithm MM

$S := 0$;
for $i := 0$ **to** n **do**
$L1$: $q_i := S \bmod 2$;
$L2$: $S := (S + q_i m) \text{ div } 2 + b_i A$;
end;
return S

where n is chosen such that $m < 2^{n-2}$ and $B = \sum_0^n b_i 2^i$. Note that by the choice of q_i, $S + q_i m$ is divisible by 2, so the algorithm basically is a multiplication $A \cdot B$ with interleaved steps of reduction (subtraction of a multiple of m). The loop delivers a value of S such that $0 \leq S < 2m$, but it also accepts operands A and B in the same interval, hence for the repeated multiplications in exponentiation there is no need to make a final reduction of S until at the very end. Observe that $b_{n-1} = b_n = 0$ are required for two extra steps of the algorithm to assure $0 \leq S < 2m$.

Noting that $S - q_i$ and $m + 1$ are both even, we can rewrite the assignment at $L2$ into:

$$S := (S - q_i) \text{ div } 2 + q_i((m + 1) \text{ div } 2) + b_i A.$$

We can now build a systolic, modular multiplier out of $\left\lceil \frac{n+1}{2} \right\rceil$ cells of the form shown in Figure 6, with $A = \sum_0^n a_i 2^i$, $B = \sum_0^n b_i 2^i$ and $\frac{m+1}{2} = \sum_0^n m_i 2^i$. The cell is obtained by merging two multiplier cells (Figure 4) into one, combining their adder structures. The maximal carry value to be retained here is 4, distributed in two latches of weight 1 and one of weight 2. The output of full adders has been marked with the relative weight of the signals.

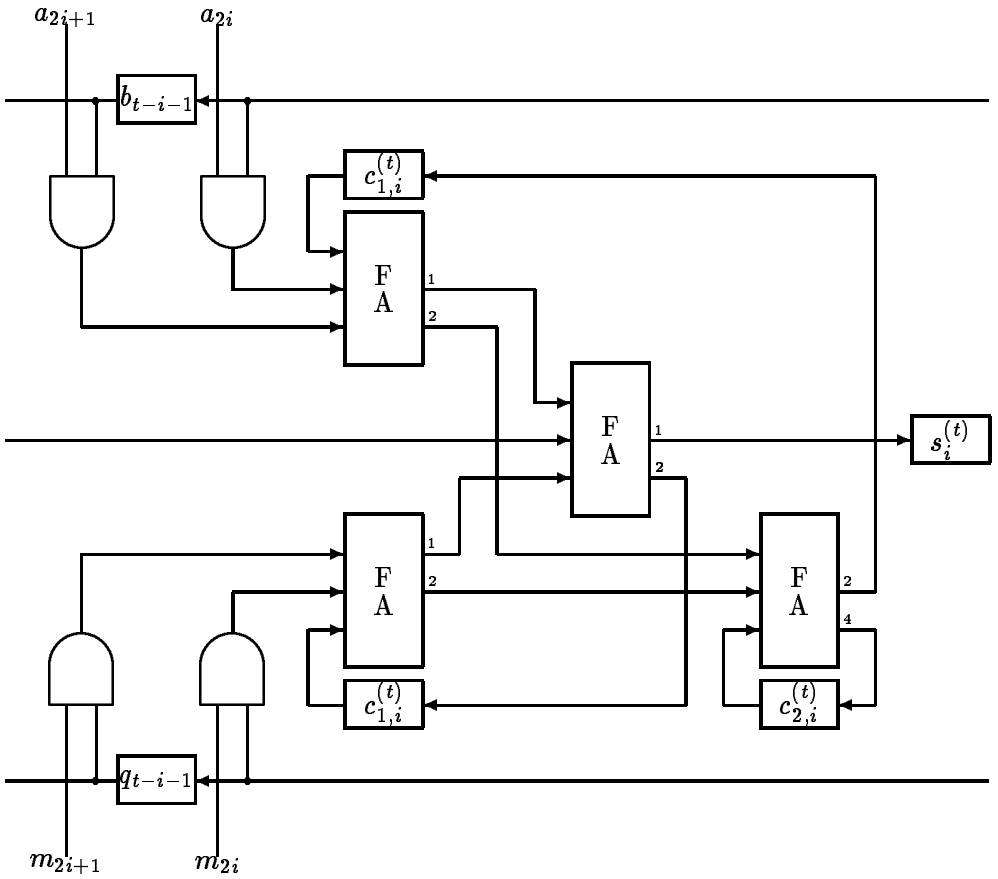


Figure 6: Systolic cell for Modular Multiplication.

For modular exponentiation we need repeated modular multiplications and squarings according to the following algorithm to compute $y = x^e \bmod m$:

Algorithm ME:

$y := 1$; $z := x$;
for $i := 0$ **to** $n-1$ **do**
    if $e_i = 1$ **then** $y := (y \times z) \bmod m$;
    $z := (z \times z) \bmod m$;
end;
return y

where $e = \sum_0^{n-1} e_i 2^i$ and the modulo reductions are substituted by Montgomery reductions assuming that operands initially are converted into Montgomery residues, and the result at the end is converted back into an ordinary residue. These conversions are then amortized over the total exponentiation cost, and can actually be performed as similar modular multiplications by constants.

Since potentially two multiplications are needed in each step of Algorithm ME, they may as well both be performed in parallel and the new value for y only conditionally substituted for the old value. By combining a systolic modular multiplier for yz with one for zz, operating in parallel, we obtain a systolic exponentiator as shown in Figure 7.

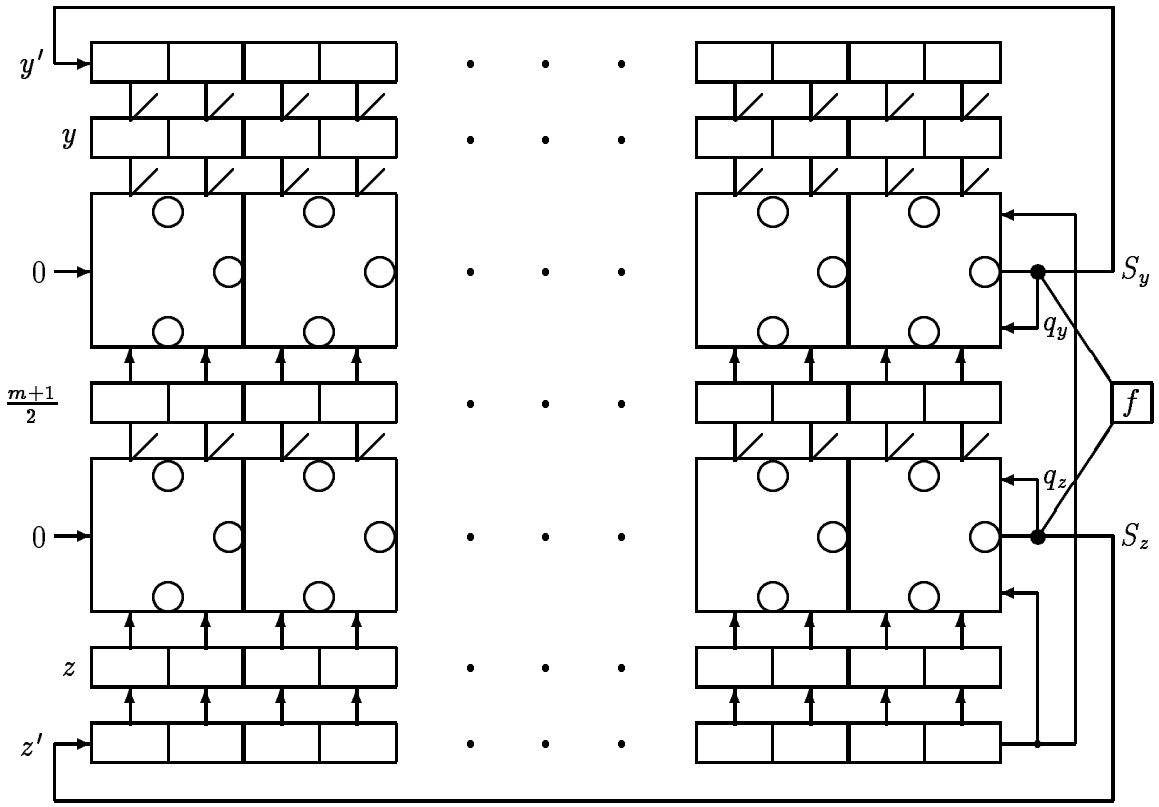


Figure 7: A Systolic System for Modular Exponentiation.

The f box at the right is a mode flip-flop, of value 0 throughout the first $n+1$ cycles of a multiplication and value 1 during the last $n+1$ cycles. It is used to control the determination of q_y, S_y, respectively q_z, S_z:

$$\begin{aligned} q_y &= S_y^{in} \wedge \overline{f}, \qquad S_y^{out} = S_y^{in} \wedge f \\ q_z &= S_z^{in} \wedge \overline{f}, \qquad S_z^{out} = S_z^{in} \wedge f, \end{aligned}$$

taking place in the black circles.

The y' and z' registers are right-shift registers. During the first $n+1$ cycles, zeroes are entered and during the next $n+1$ cycles new values of y' and z' are shifted in. Depending on exponent bit e_i (not shown), y' is then conditionally loaded into y, z' unconditionally into z. The value of z' is furthermore gradually shifted out the right end to be used as the multiplier for both multiplications. Note that the zeroes in z' are used during the second phase, to be used as extension of the multiplier.

Due to the cyclic nature of the structure, the number of cells has to match the number of steps of the algorithm. But since $a_{n-1} = a_n = 0$ and $m_{n-1} = m_n = 0$ the leftmost cell is actually not needed, since it can never contribute to the computation. It is possible to eliminate one of the two additional steps by noting that in Algorithm MM, the reduction is one step behind the introduction of the new term $b_i A$. By changing $L1$ into $q_i := (S + b_i a_0) \bmod 2$ and $L2$ into $S := (S + q_i m + b_i A) \operatorname{div} 2$, only n cycles of the loop are needed. However, this will require some modification in the right-most cell and thus destroy the regularity of the systolic structure.

The total system requires n systolic modular multiplier cells and $5n$ latches/flip-flops. The total execution time for a n-bit modular exponentiation is $2n^2$ cycles when the exponent e also has n bits, which amounts to $2n$ cycles per bit of the result. With a 100 MHz clock and $n = 500$ this corresponds to a processing speed of 100.000 bits per second.

For RSA encryption it is possible to use an exponent with much fewer bits, and for decryption the factorization of m is known, allowing this process to take place as two parallel modular exponentiations of half the size, and later combine the result by the Chinese Remainder Theorem. A speed-up of close to a factor of 4 is thus possible, using about the same amount of hardware [SV93].

By using radix 4 for the multiplications and reductions, the number of cycles can be cut in half. What is needed is then suitable logic which based on the least significant two bits of S determines a reduction factor $q_i \in \{0, 1, 2, 3\}$ such that $(S + q_i m) \bmod 4 = 0$, together with appropriate modifications of the cells to multiply in radix 4. For any radix β a similar adder structure can be obtained, since the value of the carry to be retained is limited to $4(\beta - 1)$ when digits are in the set $\{0, 1, \cdots, \beta - 1\}$. Since the complexity of the cell does not increase that much, it is feasible to use radix 4 and gain almost a factor of two in speed.

4 Other Shift-and-Add Algorithms

The multiplier structure developed in Section 2 can be used for other types of algorithms where the "multiplier" digits b_i are determined "on-the-fly", based on the least significant digit s_i produced during the algorithm. As a first example, let us consider the determination of the reduction factors q_i in the Montgomery multiplication above. In the general case of operating in radix 2^k, q_i can be determined as $q_i = ((S \bmod 2^k)m') \bmod 2^k$ where m' is the (precomputed) modular inverse of $-m$ modulo 2^k. A modular inverse can in general be determined by the extended Euclidean algorithm, but in the particular case of the modulus being a power of 2 there is a simpler algorithm, which is a variation of a classical algorithm [Hen08] by Hensel for p-adic inversion.

Modular Inverse, $a^{-1} \bmod 2^k$

For $b = a^{-1} \bmod 2^k$ to exist, a has to be odd. The algorithm will determine $b = \sum_0^{k-1} b_i 2^i$, least significant digit first, where obviously $b_0 = 1$ for $ba^{-1} \equiv 1 \pmod{2^j}$.

Algorithm MI

$S := a$; $b_0 := 1$;
for $i := 1$ **to** $k - 1$ **do**
    $S := S$ div 2; $b_i := S \bmod 2$;
    if $b_i = 1$ **then** $S := S + a$;
end;

The correctness of the algorithm follows from the fact that it actually performs the multiplication of a by b, where b_0 is chosen such that $(ab) \bmod 2 = 1$, and the subsequent b_i are chosen precisely such that all bits of S of higher weight become zeroes (the odd a is added when the previous S is odd, producing a zero which is shifted out).

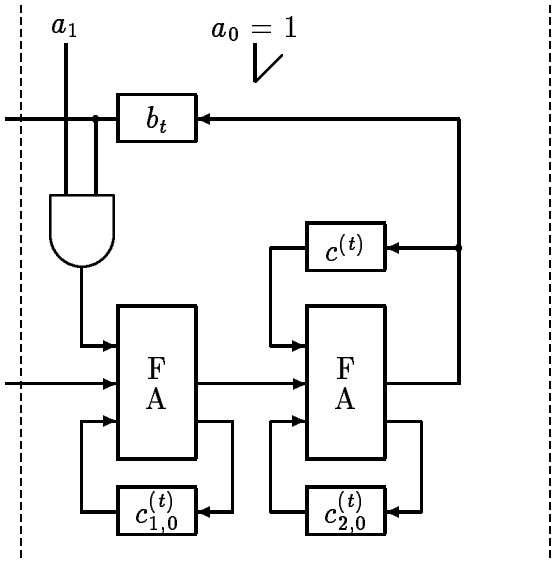


Figure 8: Rightmost Cell for Modular Inverse: $a^{-1} \bmod 2^k$.

Adapting the rightmost cell of the bit-serial systolic multiplier to Algorithm MI results in the cell in Figure 8. The rightmost full-adder here is directly connected to the line feeding the multiplier bits b_i, but also to an extra latch c which serves to obtain the effect of adding the term $a_0 b_t = b_t$. Initially all latches of the array are reset, except for the b-latch in the rightmost cell which is set to one, corresponding to $b_0 = 1$, and a is loaded as the multiplicand.

Modular Division, $d\,a^{-1} \bmod 2^m$

Since the previous algorithm is derived from "inverting" a multiplication $ab \equiv 1 \pmod{2^k}$, we can generalize the algorithm to the case of $ab \equiv d \pmod{2^k}$ corresponding to 2-adic division [Hen08] by determining bits of b successively to match the bits of d, again assuming that a is odd:

Algorithm MD

$S := 0$;
for $i := 0$ **to** $k-1$ **do**
  if $S \bmod 2 \neq d_i$ **then**
    $S := S + a$; $b_i := 1$
  else
    $b_i := 0$
  end;
  $S := S$ div 2
end;

where $d = \sum_0^{k-1} d_i 2^i$ and b is determined as $b = \sum_0^{k-1} b_i 2^i$. By a simple modification of Figure 8 we then obtain the cell in Figure 9, to be used as the rightmost cell in the systolic array.

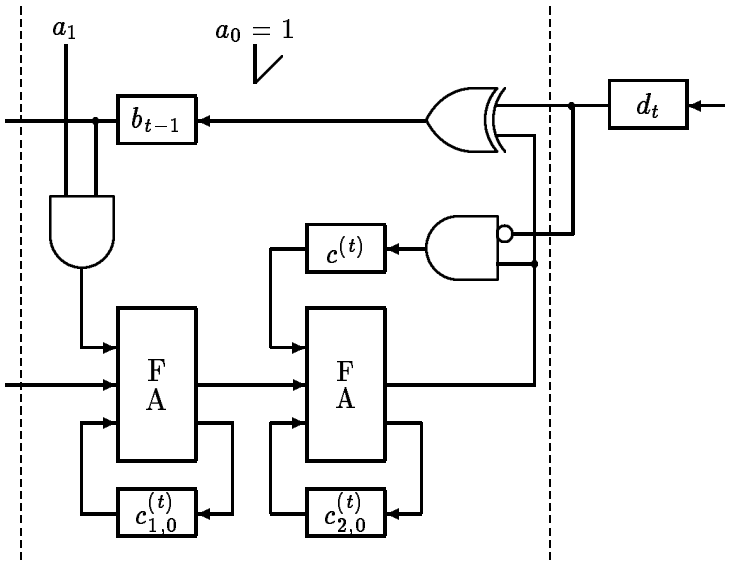


Figure 9: Rightmost Cell for Exact Division: $d \times a^{-1} \bmod 2^k$

Observe that if it is known that a divides d then Algorithm MD and the systolic array in Figure 9 performs the *exact division* of d by a, where the digits of the quotient are determined least significant digit first.

Note also that for fixed k, what is computed is the base 2, k'th order *Hensel code* [KRS75], or for unbounded k the infinite, but periodic, *2-adic representation* of the rational $\frac{d}{a}$.

For use in symbolic computations, Jebelean recently [Jeb93] presented algorithms similar to Algorithm MI and MD, however his algorithms are subtractive, based on division, whereas MI and MD are additive and based on multiplication. Jebeleans algorithms are presented for higher radix and to be implemented in a systolic fashion on a multiprocessor array for use in unbounded precision arithmetic. Algorithms MI and MD can easily be extended to a higher radix also and are very simple to use in a pipelined, LSB-first digit serial computation.

One particular application is to use the systolic multipliers and dividers in Figures 4 and 8 together with standard serial adders and subtracters, as a set of building blocks for the kind of "exact arithmetic" based on Hensel codes which is described in [GK84].

5 Conclusions

A serial-by-parallel multiplier array based on a simple and purely systolic cell has been developed. The linear array of cells performs the multiplication by a sweep across the total array of product terms at a slanted angle, such that each cell forms and accumulates two terms from each column. The motivation has been to find a simple and area efficient way of implementing modulo exponentiation as needed in cryptosystems. The cell and the array is easily extended to accommodate the interleaving of multiplication and modulo reduction. Two such arrays of cells then allow two multiplications to be performed in parallel, as needed for modulo exponentiation, thus providing a simple systolic system which can operate at a high clock frequency due to the purely nearest-neighbor communication.

By some simple modifications at the front-end or back-end cells, the basic multiplier structure has been shown to implement other algorithms, including 2'complement multiplication, modular inversion and exact division. Although the example designs are all shown in radix 2, it is simple to modify the cells to a higher radix of the form $\beta = 2^k$.

References

[Atr65] A.J. Atrubin. A One-Dimensional Real-Time Iterative Multiplier. *IEEE Transactions on Electronic Computers*, pages 394–399, 1965.

[Dad83] L. Dadda. Some Schemes for Fast Serial Input Multipliers. In *Proc. 6th IEEE Symposium on Computer Arithmetic*, pages 52–59. IEEE Computer Society, June 1983.

[GK84] R.T. Gregory and E.V. Krishnamurthy. *Methods and Applications of Error-Free Computations.* Springer-Verlag, New York, Berlin, Heidelberg, Tokyo, 1984.

[Hen08] K. Hensel. *Theorie der Algebraischen Zahlen.* B.G. Teubner, Leipzig and Berlin, 1908.

[Jeb93] T. Jebelean. An Algorithm for Exact Division. *J. Symbolic Computation*, 15:169–180, 1993.

[Kor93] P. Kornerup. High-Radix Modular Multiplication for Cryptosystems. In *Proc. 11th IEEE Symposium on Computer Arithmetic*, pages 277–283. IEEE Computer Society, 1993.

[KRS75] E. V. Krishnamurthy, T. M. Rao, and K. Subramanian. Finite Segment p-adic Number Systems with Applications to Exact Computation. *Proc. Indian Acad. Sci.*, 81:58–79, 1975. Vol. 82, pp. 165–175.

[Mon85] P. L. Montgomery. Modular Multiplication Without Trial Division. *Mathematics of Computation*, 44(170):519–521, April 1985.

[SV93] M. Shand and J. Vuillemin. Fast Implementations of RSA Cryptography. In *Proc. 11th IEEE Symposium on Computer Arithmetic*, pages 252–259. IEEE Computer Society, 1993.

[Wal93] C.D. Walker. Systolic Modular Multiplication. *IEEE Transactions on Computers*, C-42(3):376–378, March 1993.