Addition-based exponentiation modulo 2^k

A. Fit-Florea, D.W. Matula and M.A. Thornton

A novel method for performing exponentiation modulo 2^k is described. The algorithm has a critical path consisting of k dependent shift-and-add modulo 2^k operations. Although 3 is the preferred exponent base, the algorithm can be extended easily in order to perform the general binary powering operation.

Introduction and background: The basic integer arithmetic operations of addition/subtraction, multiplication and division are implemented typically in hardware using k bits of precision with k usually 16, 32, or 64, and up to 1024 in the case of cryptography. Having a precision limited to k bits makes the arithmetic operations equivalent to their corresponding residue arithmetic modulo 2^k operations along with appropriate overflow handling. When the hardware support does not include a large multiplier, there is a particular need for additive bit-serial algorithms for these and additional residue operations. In this Letter we present a bit-serial algorithm for the fundamental residue arithmetic operation of powering (or exponentiation). Following [1] we herein employ $|n|_{2^k}=j$ to denote the congruence relation $n\equiv j \pmod{2^k}$ with the residue j satisfying $0\le j\le 2^k-1$.

When computing the exponentiation operation $\beta^e \pmod{2^k}$ of a basis β (our preferred case is $\beta=3$), usually some variation of the square-and-multiply algorithm is being employed. In this method the squaring operation is performed sequentially obtaining $|3^{2^1}|_{2^k}$, $|3^{2^2}|_{2^k}$, $|3^{2^3}|_{2^k}$, $\ldots$, $|3^{2^{k-1}}|_{2^k}$. From these residues a subset is selected to be part of the product corresponding to $|3^e|_{2^k}$:

$$|3^e|_{2^k} = \left|3^{\sum_{i\in B_e}(2^i)}\right|_{2^k} = \left|\prod_{i\in B_e} 3^{2^i}\right|_{2^k} = \left|\prod_{i\in B_e} |3^{2^i}|_{2^k}\right|_{2^k} \tag{1}$$

The exponent e is expressed as a sum of powers of 2 reflecting its binary representation, and B_e is the set of weights for the 1 digits in the binary representation of e. For example $B_{19}=\{0,\ 1,\ 4\}$ since $19=2^0+2^1+2^4$.

Using a square-and-multiply method, $O(k)$ squaring and $O(k/2)$ multiplications modulo 2^k are to be performed in the worst case [2]. Storing in a k-entries lookup table the results of the squaring operations $|3^{2^i}|_{2^k}$ reduces the computations needed to $O(k/2)$ multiplication modulo 2^k. In the following we present a method that virtually replaces each multiplication with one shift and two concurrent add modulo 2^k operations, thus having the potential to improve a hardware implementation in both area and time over a square-and-multiply method implementation.

Relevant algebraic properties: We note the fact that the exponentiation modulo 2^k is cyclic with period 2^{k-2} [3], hence we consider w.l.g. the exponents e to be in the range $0, 1, \ldots, (2^{k-2}-1)$. The algebraic property that makes possible expressing any exponent e as a sum of powers of 2 is the fact that $\mathcal{B}=\{2^i\colon 0\le i<(k-2)\}$ is a basis for the additive group of residues e modulo 2^{k-2}. Decomposing e as a sum of elements of another basis still produces a correct result. In the following we present such a basis and show that using it has the advantage of eliminating the need for multiplications when computing the exponentiation modulo 2^k.

We denote the discrete logarithm modulo 2^k with logarithmic base 3 of A (in case it exists) by $dlg(A)$. This simply represents the exponent e such that 3^e is congruent with ($A \bmod 2^k$). That is: $|A|_{2^k}=|3^{dlg(A)}|_{2^k}$. For more details the reader is referred to [3]. Also from [3], we mention the following result:

Lemma 1: Let ρ be a residue modulo 2^k of the form

$$\rho = 1+2^i+2^{i+1}\varrho, \quad 2<i<k, \quad 0\le\varrho<2^{k-i-1} \tag{2}$$

Its corresponding discrete logarithm $dlg(\rho)$ is then of the form

$$dlg(\rho) = 2^{i-2}+\delta_\rho\times 2^{i-1}, \quad \text{for some } \delta_\rho, \quad 0\le\delta_\rho<2^{k-i-1} \tag{3}$$

We use τ_i to denote what we call the two-ones residues modulo 2^k: $\tau_i=|2^i+1|_{2^k}$. The following observation comes as a direct consequence of *Lemma 1*.

Observation 1: The discrete logarithm of two-ones residues τ_i is of the form:

$$dlg(\tau_i) = 2^{i-2}+2^{i-1}\times\theta_i, \quad 2<i<k, \quad \text{for some } \theta_i, \quad 0\le\theta_i<2^{k-i-1}$$

In Table 1 we show the two-ones residues and their corresponding discrete logarithms for $k=8$. As it can be inferred directly from *Observation 1*, the set $\mathcal{BT}=\{dlg(\tau_i)\colon i=1, 3, 4, \ldots, (k-1)\}$ represents a basis for residues e, $0\le e<2^{k-2}$, in the sense that, again, any exponent e can be represented as a sum of elements from $\mathcal{BT}$. Consequently, $|3^e|_{2^k}$ can be expressed as a product:

$$\begin{aligned}|3^e|_{2^k} &= \left|3^{\sum_{i\in\beta_e} dlg(\tau_i)}\right|_{2^k} = \left|\prod_{i\in\beta_e}|3^{dlg(\tau_i)}|_{2^k}\right|_{2^k}\\ &= \left|\prod_{i\in\beta_e}(2^i+1)\right|_{2^k} = \left|\prod_{i\in\beta_e}\tau_i\right|_{2^k}\end{aligned} \tag{4}$$

for a set β_e of indices unique to any e. Once the set β_e is known, $|3^e|_{2^k}$ can be computed as a product of two-ones residues. Multiplying by $\tau_i=(2^i+1)$ has the advantage that it can be performed as a modulo 2^k shift-and-add operation: $A\times\tau_i := A+A\ll(i)$, thus eliminating the need for a multiplier. In the following we show an algorithm for selecting the elements of sets β_e in a serial fashion.

Table 1: Two-ones discrete log table for $k=8$

i	τ_i	$dlg(\tau_i)$
1	0000 0011	00 0001
3	0000 1001	00 0010
4	0001 0001	11 0100
5	0010 0001	10 1000
6	0100 0001	01 0000
7	1000 0001	10 0000

Exponentiation modulo 2^k algorithm:

Stimulus: An exponent e (modulo 2^{k-2}).
Response: $|3^e|_{2^k}$.
Method: L1: $P := 1$; $|e'| := e$;
L2: **if** $(e'_0=1)$ **then** $P := 11$; $|e'| := e'-1$;
L3: **for** i from 1 to $(k-3)$ **do**
L4: **if** $(e'_i=1)$ **then**
L5: $e' := |e'-dlg(\tau_{i+2})|_{2^{k-2}}$;
L6: $P := |P+|P\ll(i+2)|_{2^k}|_{2^k}$;
L7: Result: P.

The initialisation is performed in lines L1 and L2. The product P is set to either 1 or 11 (corresponding to $e=0$ or $e=1$). The working variable exponent e' is always set in such a way that P corresponds to 3 raised at exponent $(e-e')$ and the least significant i digits of e' are all 0s. The algorithmic step of lines L3 – L6 is updating e' by subtracting $dlg(\tau_i+2)$, the exponent of $\tau_i=(2^i+1)$, and the product P to reflect the changes in exponent, $P := P\times(2^{i+2}+1)$. Eventually, after $(k-2)$ steps, e' becomes 0 and the 'product' P corresponds to $|3^{e-0}|_{2^k}=|3^e|_{2^k}$. The values $dlg(\tau_{i+2})$ can be computed beforehand (e.g. using the algorithm described in [3]), and stored in a lookup table of uncompressed size $(k-2)^2$ bits.

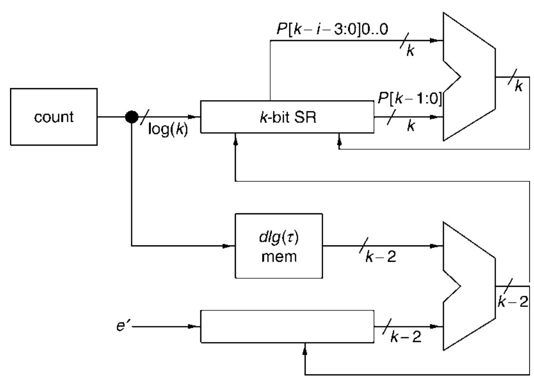

Fig. 1 *Iterative loop for L3–L6 of algorithm 1*

The algorithm has a critical path determined by $(k-2)$ dependent shift-and-add modulo 2^k operations. This is because the subtractions of lines L5 and L6 can be performed concurrently. An extension of the algorithm for computing exponentiation of a base β different than 3 is suggested in the section entitled 'Base exchange for discrete logarithm modulo 2^k' of [3]. The same formula that works for regular logarithms can be employed:

$$\beta^e = 3^{e \times dlg(\beta)} \tag{5}$$

Using it comes at the cost of computing an extra $dlg(\beta)$ while keeping the same tables.

Fig. 1 is a schematic diagram of an implementation of the datapath portion of the algorithm. It implements the iterative portion of the algorithm described in lines L3 – L6. This circuit consists of a counter, a small lookup table that may be in compressed form, and two add/accumulate units. The value of P is stored in shift-registers that shift content to the left. These values are replaced by multiples of 2^{i+2} depending on the value of each e_i' bit.

22 October 2004

Electronics Letters online no: 20057538

doi: 10.1049/el:20057538

A. Fit-Florea, D.W. Matula and M.A. Thornton (*Southern Methodist University, PO Box 750122, Dallas, TX 75275, USA*)

E-mail: alex@engr.smu.edu

References

1 Szabo, N.S., and Tanaka, R.I.: 'Residue arithmetic and its applications to computer technology' (McGraw-Hill, New York, 1967)

2 Lam, K.-Y., and Hui, L.C.K.: 'Efficiency of SS(l) square-and-multiply exponentiation algorithms', *Electron. Lett.*, 1994, **30**, (25), pp. 2115–2116

3 Fit-Florea, A., and Matula, D.W.: 'A digit-serial algorithm for the discrete logarithm modulo 2^k'. IEEE 15th Int. Conf. on Application-specific Systems, Architectures and Processors, ASAP, 2004