Additive bit-serial algorithm for discrete logarithm modulo 2^k

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

A novel algorithm for computing the discrete logarithm modulo 2^k that is suitable for fast software or hardware implementation is described. The chosen preferred implementation is based on a linear-time multiplier-less method and has a critical path of less than k modulo 2^k shift-and-add operations.

Introduction and summary: Hardware capabilities for integer arithmetic generally include addition, multiplication, and division with precision k typically chosen as 16, 32 or 64. Multiplication and division are often implemented by recursive bit serial algorithms employing $O(k)$ serial additions to avoid the size and power requirements of a large multiplier. The integer addition and multiplication operations realised are effectively 'exact' residue arithmetic operations with modulo 2^k.

Hardware support for applications where fast residue arithmetic computation is desirable is typically limited to only residue addition and multiplication. There is a need to find efficiently implementable algorithms for other fundamental residue operations for the 'hardware friendly' modulus 2^k. Furthermore, for implementations where hardware support does not include a large multiplier, there is a particular need for additive bit-serial algorithms for these additional residue operations.

The fundamental residue arithmetic operations supplementing residue addition and multiplication of particular interest for feasibility of hardware implementation are: multiplicative inverse, powering (or exponentiation), and discrete logarithm. 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 $0 \le j \le 2^k - 1$. The *discrete logarithm* modulo 2^k with logarithmic base $3 dlg(j) = e$ of an odd residue j, $1 \le j \le 2^k - 1$, is the minimum exponent e, when it exists, such that $|3^e|_{2^k} = j$. Similarly, $e = dlg_{(\beta,M)}(j)$ represents the discrete logarithm modulo M with logarithmic base β of j: $|\beta^e|_M = j$.

From [2–4], $dlg(j)$ exists whenever $|j|_8 \in \{1, 3\}$, and also $0 \le dlg(j) \le 2^{k-2} - 1$. Furthermore, for any odd residue j with $1 \le j \le 2^k - 1$, there is a unique *sign*, *exponent pair* (s, e) with $s \in \{0, 1\}$, $0 \le e \le 2^{k-2} - 1$ such that

$$|(-1)^s \times 3^e|_{2^k} = j \tag{1}$$

In [3] we showed that the pair (s, e) of (1) can be determined from the odd input j employing $O(k)$ dependent modular multiplications. Our main result in this Letter is showing how to determine the pair (s, e) by a bit serial (shift-and-add) algorithm employing only $O(k)$ dependent additions and a lookup table of size roughly k^2 bits.

Discrete logarithm modulo 2^k—algebraic properties: Lemma 1 represents the core result for Algorithm 1. We omit a formal proof and instead proceed with pointing out the essential mathematical properties that lead to a constructive proof and its corresponding algorithm.

Lemma 1: For any $k \ge 2$, every odd integer j with $1 \le j \le 2^k - 1$ has a unique modular factorisation

$$j = \left| (-1)^s \prod_{i \in I_j} (2^i + 1) \right|_{2^k}$$

with factor selection specified by $s \in \{0, 1\}$ and $I_j \subseteq \{1\} \cup \{3, 4, \ldots, k-1\}$ for $k \ge 3$.

Notationally we use a_j as shorthand for the jth digit of A, and a_j^i for the jth digit of A_i. Also, we will call a residue $\tau_i = |2^i + 1|_{2^k}$, $3 \le i < k$ to be a *two-ones residue*. The key advantage of multiplying by two-ones residues τ_i is that a multiplication by τ_i can be performed simply as a less expensive shift-and-add operation:

$$P_i \times \tau_i = P_i + P_i \ll i$$

where $P_i \ll i$ represents an i bits left shift of P_i.

Our method consists of three stages. The first stage is initialising $P_2 = A$. The second stage and main iteration step is updating $P_{i+1} = |P_i \times B_i|_{2^k}$. Values B_i are to be selected in such a way that after $(k-2)$ steps $P_k \equiv |1|_{2^k}$ is obtained. Finally, in the third stage, the discrete logarithm modulo 2^k of A is readily available. This because $P_k = |A \times \prod_{i=2}^{i \le k} B_i|_{2^k} = 1$, and we have:

$$\left| dlg(A) + \sum_{i=2}^{i \le k} dlg(B_i) \right|_{2^{k-2}} = 0, \quad \text{hence: } dlg(A) = \left| -\sum_{i=2}^{i \le k} dlg(B_i) \right|_{2^{k-2}}$$

can be directly computed if $dlg(B_i)$ are all known and $P_k \equiv |1|_{2^k}$. For more mathematical details the reader is referred to [3]. We choose $B_i = \tau_i$ and update P_i such that its last i digits become $00\ldots01$ (i.e. $|P_i|_{2^i} = 1$):

Observation 1: Whenever the binary digit p_i^i of P_i equals 1 (i.e. $|P_i|_{2^i} = 1$), multiplying P_i with the two-ones residue $\tau_i = (2^i + 1)$ results in a product $P_{i+1} = P_i \times \tau_i$ that is congruent with 1 modulo 2^{i+1}. That is:

$$P_I \equiv |2^i + 1|_{2^{i+1}} \Rightarrow P_{i+1} = P_i \times \tau_i \equiv |1|_{2^{i+1}}$$

When the binary digit p_i^i of P_i equals 0, P_{i+1} can be set to $P_{i+1} = (P_i \times 1)$ and it is still congruent with $|1|_{2^{i+1}}$.

We show in Table 1 the (valid) 8-bit *dlgs* associated with the corresponding two-ones residues (i.e. $\tau_i \equiv 3^{dlg(\tau_i)}$). Also, in the last column we suggest how the updating of the partial products P_{i+1} works when $p_i^i = 1$ and values τ_i are to be used. The values $dlg(\tau_i)$ can be precomputed using any dlg method, e.g. the one we presented in [3]. Storing these values in a table requires a lookup table of uncompressed size smaller than k^2 bits.

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

i	τ_i	$dlg(\tau_i)$	$P_i \times \tau_i \rightarrow P_{i+1}$
3	0000 1001	00 0010	$p_7^3 p_6^3 p_5^3 p_4^3$ 1001 × 1001 → $p_7^4 p_6^4 p_5^4 p_4^4$ 0001
4	0001 0001	11 0100	$p_7^4 p_6^4 p_5^4$1 0001 × 1 0001 → $p_7^5 p_6^5 p_5^5$0 0001
5	0010 0001	10 1000	$p_7^5 p_6^5$10 0001 × 10 0001 → $p_7^6 p_6^6$00 0001
6	0100 0001	01 0000	$p_7^6$100 0001 × 100 0001 → $p_7^7$000 0001
7	1000 0001	10 0000	1000 0001 × 1000 0001 → 0000 0001

Shift-and-add DLG algorithm:

Stimulus: A modulus 2^k with $k \ge 3$ and an odd valued residue $A = a_{k-1} a_{k-2} \ldots a_0$.
Response: $dlg(A)$, expressed as an (s, e) pair where $|(-1)^s \times 3^e|_{2^k} = A$.
Method: L1: $P := A$; $|e|_{2^k} := 0$; $s := 0$;
L2: **if** $(p_2 = 1)$ **then** $s := 1$; $P := |-P|_{2^k}$; **fi**
L3: **if** $|P|_{2^3} = 011$ **then** $|e|_{2^1} := 1$; $P := |P + P \ll 1|_{2^k}$; **fi**
L4: **for** i from 3 to $(k-1)$ **do**
L5: **if** $(p_i = 1)$ **then** $e := e + dlg(\tau_i)$; $P := |P + |P \ll i|_{2^k}|_{2^k}$; **fi**
L6: Result: $(s, |-e|_{2^{k-2}})$.

The first-initialisation-stage is performed in lines L1–L3. If A is not congruent with 1 or 3 modulo 8, then $|-A|_{2^k}$ is, and the algorithm determines the *dlg* of $|-A|_{2^k}$ (i.e. $P = |-P|_{2^k}$ in L2). The variable e represents the exponent of 3 that gives $|P^{-1}|_{2^i} = |3^e|_{2^i}$. It is set to 0 in L1 corresponding to $|P|_{2^3} = 1$. In the case $|P|_{2^3} = 011$, e is adjusted in line L3 to be 1, along with the corresponding update of P (which is equivalent to $P = |3 \times P|_{2^k}$). In the second stage P is iteratively updated (conceptually) as a series of multiplications $P_{i+1} = P_i \times \tau_i$, while e is updated with the corresponding values $dlg(\tau_i)$ looked up from a table. The updating of e and P in line L5 can be performed concurrently. The final result is computed in line L6 as the sign s and the exponent $|-e|_{2^{k-2}}$. This is because e really represents $e = dlg((-1)^s \times A^{-1})$, hence $dlg((-1)^s \times A) = |-e|_{2^{k-2}}$. As can be seen after a quick look at algorithm 1, its time complexity is essentially k dependent shift-and-add modulo 2^k operations.

Figs. 1 and 2 are schematic diagrams of an implementation of the datapath portion of algorithm 1. Fig. 1 implements lines L1–L3 and sets up the appropriate values in the P and B registers based on the sign of A and the values of the least significant bits. Note that no error-checking circuitry is included and it is assumed that only odd values of A are used. Fig. 2 implements the iterative portion of the algorithm described in lines L4–L5. This circuit consists of a counter, a small lookup table that may be in compressed form, and three add/accumulate units. The values of P and B are stored in shift-registers that shift content to the left. These values are replaced by multiples of 2^k depending on the value of each p_i bit.

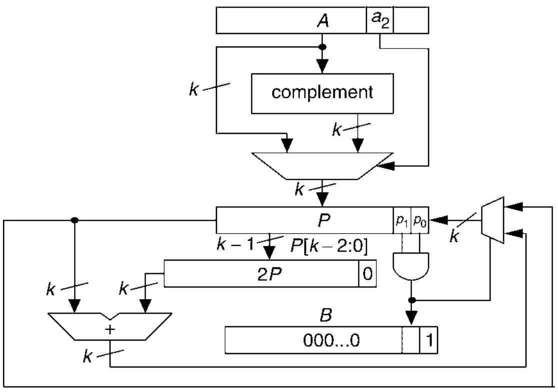

Fig. 1 *Register setup for L1–L3 of algorithm 1*

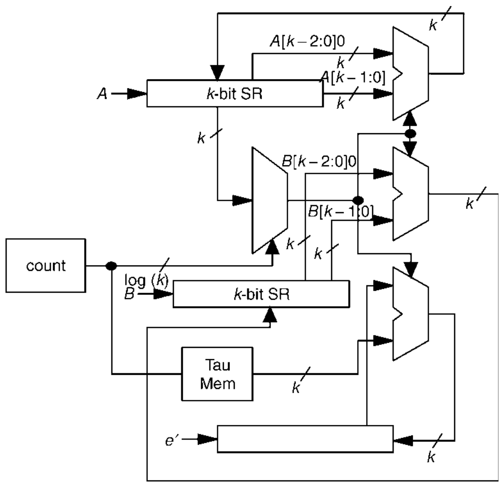

Fig. 2 *Iterative loop for L4–L5 of algorithm 1*

 13 September 2004
Electronics Letters online no: 20056993
doi: 10.1049/el:20056993

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 Benschop, N.F.: 'Multiplier for the multiplication of at least two figures in an original format', US Patent No. 5,923,888, July 13, 1999
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
4 Niven, I., and Zuckerman, H.S.: 'An introduction to the theory of numbers' (John Wiley & Sons, New York, 1966, 2nd edn.)