Practical cellular dividers

Franco Preparata, J. Vuillemin

UNITÉ DE RECHERCHE
INRIA-ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

N° 807

PRACTICAL CELLULAR DIVIDERS

F. P. PREPARATA
J. E. VUILLEMIN

MARS 1988

DIVISION ARITHMETIQUE CELLULAIRE

F.P. Preparata* J.E. Vuillemin †

Résumé

Les algorithmes de division introduits ici utilisent des représentations redondantes des opérandes. Ils se réalisent sous forme de circuits, en taille et temps 0(n), où n est la longueur du dividende. Le quotient est obtenu par les poids forts. Ces diviseurs présentent des avantages significatifs pour le calcul d'exponentielles modulaires (cryptographie RSA), de plus grands communs diviseurs, d'unités de calcul flottant, et le traitement du signal.

* *Ecole Normale Supérieure, 45 rue d'Ulm, 75230, Paris. On leave from the University of Illinois at Urbana.* *This work was partially supported by NSF Grant ECS-84 10902.*

† *Institut National de Recherche en Informatique et Automatique, 78150, Rocquencourt, France.* *Part of this work was supported by Xerox Palo Alto Research Center.*

Practical Cellular Dividers

F.P. Preparata * J.E. Vuillemin †

Abstract

The parallel division algorithms discussed in this note may be classified among modified higher-radix nonrestoring division methods, where redundant representations are extensively utilized to speed up the operation. The network realizations of these algorithms are cellular, or even systolic with exclusively local control; they have size (area) and time both $O(n)$, where n is the length of the dividend representation. The quotient is produced most-significant digit first. This feature enables the use of the divider as a signed, digit-serial multiplier. Although only elementary techniques are involved, these divider/multipliers appear to be the first having such characteristics. When suitably equipped with some control and a few registers, the divider/multiplier brings remarkable performance to large modular arithmetic, RSA cryptography, and gcd computations. They are also of interest to the design of floating-point units, and signal processing applications.

1 Introduction

The design of efficient division algorithms is a central problem in digital arithmetic. Indeed, division (the formal inverse of multiplication over the rationals) is substantially different from multiplication when we operate over the integers, and algorithmically more complex.

Several approaches to digital division have been proposed in the last four decades [H]. Division methods can be broadly subdivided into two classes: those that iteratively update the dividend, and those that compute the quotient through multiplication of the dividend by the reciprocal of the divisor. Only the latter have the capability of achieving high speed, at the expense of great circuit complexity [Me],[BCH],[PM]. In particular, to this class belongs the only known method [BCH] that achieves minimal time $O(log(n))$, where n is the operand length.

*Ecole Normale Supèrieure, 45 rue d'Ulm, 75230, Paris. On leave from the University of Illinois at Urbana. This work was partially supported by NSF Grant ECS-84 10902.

†Institut National de Recherche en Informatique et Automatique, 78150, Rocquencourt, France. Part of this work was supported by Xerox Palo Alto Research Center.

The approach presented in this note belongs to the first class. The operation is very simple and only elementary techniques are involved, resulting in a small and practical design. By adopting, as in SRT division [R], a redundant representation for the quotient, each quotient digit is *estimated* on the basis of short prefixes of the divisor and current dividend (partial remainder). Due to the redundant representation, an inaccurate estimate of a quotient digit gets *corrected* by the next digit, ultimately resulting in the (redundantly represented) correct quotient.

The use of redundant representations not only permits the correct restoration of the quotient, but also allows constant parallel time computation of the dividend updates. As a result, the division process can be carried out in time $O(n)$ by a circuit whose size is also $O(n)$. Asymptotically analogous performance is exhibited by a recently proposed divider circuit [PP], which is based on an entirely different technique, and generates the quotient from the least significant end. The dividers proposed in this note have thus performance parameters analogous to those of well known multipliers, bit-parallel as in [L] or cellular as in [A],[Mu]. Likewise, our divider can be implemented in a digit parallel fashion, or as a cellular (systolic [Ku]) structure, where each module is amenable to exclusively local control and data operation.

An approach based on similar principles was presented in [TYY] for the base 2, yielding an implementation with area $O(n^2)$ and time $O(n)$. In this note, we introduce a related technique, based on euclidean division, which generalizes to arbitrary base $b = 2^l$. We also present an alternative method, based on the division by smallest remainder. Both techniques are ameanable to cellular or systolic realizations.

The presented schemes lend themselves to a number of applications.

- The divider can be operated, in a rather straightforward manner, as an integer multiplier, where the digits of the (redundantly represented) product are serially generated starting from the most significant end.

- The divider/multiplier can be used to implement modular multiplication, which in turn can compute exponentials modulo a given integer. This leads to an implementation of high-security ($\geq$ 512 bits) RSA-cryptography [RSA] more efficient than previously known ones.

- The approach can be adapted to *greatest common divisor* (gcd) computation, in *linear* $O(n)$ time and area. For this application, the use of division by smallest remainder appears to be crucial. Our direct implementation of Euclid's algorithm should be contrasted with the one described in [BK], based on the binary gcd.

This paper is organized as follows. In Section 2, we briefly review the positional irredundant and redundant number representations used in the algorithms. In Section 3, we present the two division algorithms. Section 4 describes two typical circuit implementations, digit parallel and cellular. Finally, in Section 5, we illustrate uses of the divider in the applications mentioned above.

2 Number Representations

Numbers are represented in base $b = 2^l$, $l \geq 1$ by a sequence $n_0 n_1 \cdots n_k$ of *digits*, with the usual positional convention:

$$n_0 n_1 \cdots n_k \equiv \sum_{0 \leq j \leq k} n_j b^{-j}.$$

By placing various constraints on the range of values allowed for digits n_j, $j > 1$, to the right of the fractional point, different number systems arise whose nomenclature is given below:

- Ordinary nonredundant base b unsigned-digit representation:

  $$n_j \in [0, b-1].$$

  Each digit is encoded with l bits. For example, $n = 1000/4^4$ is represented in base 4 by [3.3 2 2 0].

- Nonredundant base b signed-digit representation:

  $$n_j \in [-\frac{b}{2}, \frac{b}{2} - 1].$$

  Each digit is encoded with l bits, the most significant one having negative sign. For example, $n = 1000/4^4$ is represented in base 4 by [4.0 $\bar{1}$ $\bar{2}$ 0], with $\bar{1}, \bar{2}$ denoting -1,-2.

- one-value-redundant base b representation:

  $$n_j \in [-\frac{b}{2}, \frac{b}{2}].$$

  Each digit is encoded with $l + 1$ bits, the most significant having negative sign. For example, $n = 1000/4^4$ can be represented in base 4 by [4.$\bar{1}$ 2 2 0] as well as by [4.0 $\bar{1}$ $\bar{2}$ 0], redundant representations being no longer unique.

- two-value-redundant base b representation:

  $$n_j \in [-\frac{b}{2} - 1, \frac{b}{2}].$$

  This is a minor, but interesting, variation of the one-value-redundant representation.

- one-bit-redundant base b representation:

  $$n_j \in [-b, b-1].$$

  Each digit is encoded with $l + 1$ bits, the most significant having negative sign. For example, $n = 1000/4^4$ can be represented in base 4 by [5.$\bar{4}$ $\bar{1}$ 2 0] as well as by [3.3 3 $\bar{1}$ 4], among others.

In all representations involving signed digits, a digit is represented in the conventional two's complement form.

It is easily seen that conversion (both ways) between n-digit nonredundant signed and unsigned representations can be achieved with an ordinary adder, with digit-serial time n, or bit-parallel time $log_2(n)$ as in [VG]. The same applies to conversion from redundant to nonredundant representation. By using classical *carry-save* adders, to be generalized later, we can convert between one-bit-redundant and one-value-redundant representations in *constant* parallel time. The following easily established facts are worth noting:

(i) In the conversion from the one-bit-redundant to the two-value-redundant representation (or to any representation of intermediate redundancy), carries propagate no more than *one* digit position;

(ii) In the conversion from one-bit-redundant to one-value-redundant, carries propagate no more than *two* digit positions.

3 Division Algorithms

3.1 N- and Z- Divisions

Let integers N, D, Q, and R respectively denote the dividend, divisor, quotient and remainder, related by:

$$N = D \times Q + R.$$

A further condition uniquely determines Q and R. The algorithms to be presented in this note are based on the following criteria.

(N) Euclidean Division (N-division):

$$0 \leq R < D;$$

(Z) Division by smallest remainder (Z-division):

$$-\frac{D}{2} \leq R < \frac{D}{2}.$$

To distinguish between the two, we use the notation

$$R = N \, mod \, D \quad , \quad Q = N \, quo \, D$$

for the (usual euclidean) N-division, and

$$R = N \cdot|\cdot D \quad , \quad Q = N \div D$$

for the (smallest remainder) Z-division. Both divisions are simply related as follows:

$$\begin{aligned} (N \, quo \, D) &= if \quad N \cdot|\cdot D < 0 \quad then \quad -1 + N \div D \quad else \quad N \div D; \\ (N \, mod \, D) &= if \quad N \cdot|\cdot D < 0 \quad then \quad D + N \cdot|\cdot D \quad else \quad N \cdot|\cdot D. \end{aligned}$$

3.2 Algorithm for Z-division

The operands are suitably normalized, by shifting the fractional point, and we adopt the following representations:

- Dividend $N = n_0 \cdots n_n$ is one-bit redundant:

$$j \geq 1,\ n_j \ \in \ [-b, b-1]. \tag{1}$$

- Divisor $D = d_0 \cdots d_d$ is signed-digit nonredundant:

$$j \geq 1,\ d_j \ \in [-\tfrac{b}{2}, \tfrac{b}{2} - 1]. \tag{2}$$

- The membership interval of n_0 and d_0 will be specified later.

- Quotient $Q = q_0 \cdots q_{n-d}$ is one-value-redundant:

$$t \geq 0,\ q_t \in [-\frac{b}{2}, \frac{b}{2}]. \tag{3}$$

In the description of the iterative algorithm, we use superscripts to denote the index t of the division step. Specifically, we initialize $N^{(0)} = N$, and assume throughout that $d_0 > 0$. At the completion of the t-th division step, we have computed a partial quotient

$$Q^{(t)} = q_0 \cdots q_{t-1} \ \equiv \ \sum_{0 \leq j < t} q_j b^{-j},$$

and a current dividend $N^{(t)}$, related to the initial dividend $N = N^{(0)}$ by:

$$N \ = \ D \times Q^{(t)} + N^{(t)}.$$

We determine the next quotient digit of q_t by dividing the first digit of the dividend $N^{(t)}$ by that of the divisor, according to the Z-division criterion:

$$q_t \ = \ n_0^{(t)} \div d_0. \tag{4}$$

We then compute the partial remainder

$$R^{(t+1)} \ = \ N^{(t)} - D \times q_t,$$

all digits being processed independently without carry propagation

$$j \geq 0,\ r_j^{(t-1)} \ = \ n_j^{(t)} - d_j \times q_t, \tag{5}$$

so as to obtain the current remainder in one parallel time unit. In order to obtain the updated dividend $N^{(t+1)}$ in the representation (1), we propagate carries through one position to the left in constant time. To do so, define

$$j \geq 1,\ r_j^{(t+1)} = s_j^{(t-1)} + b \times c_j^{(t+1)}, \tag{6}$$

with

$$\begin{aligned} j \geq 1,\ c_j^{(t+1)} &= r_j^{(t+1)} \div b, \\ j \geq 1,\ s_j^{(t+1)} &= r_j^{(t-1)} \cdot|\cdot b; \end{aligned}$$

compute the next dividend $N^{(t+1)} = b \times R^{(t-1)}$ by

$$n_0^{(t-1)} = (r_0^{(t+1)} b + s_1^{(t-1)}) + (c_1^{(t+1)} b + c_2^{(t-1)}) \tag{7}$$

$$j \geq 1,\ n_j^{(t+1)} = s_{j-1}^{(t-1)} + c_{j-2}^{(t+1)} \tag{8}$$

in constant time.

Invariance of the representations of operands From (1), (2), (3), and (5), we obtain

$$j > 0,\ r_j^{(t+1)} \in [-b - \frac{b^2}{4}, b + \frac{b^2}{4} - 1]; \tag{9}$$

from (6),

$$j > 0,\ s_j^{(t+1)} \in [-\frac{b}{2}, \frac{b}{2} - 1], \tag{10}$$

and

$$j > 0,\ c_j^{(t+1)} \in [-\frac{b}{4} - 1, \frac{b}{4} + 1], \tag{11}$$

for $b \geq 4$; base 2 has:

$$b = 2,\ c_j^{(t+1)} \in [-1, 1].$$

Relations (10) and (11) imply that

$$j > 0,\ n_j^{(t+1)} \in [-\frac{3b}{4} - 1, \frac{3b}{4}], \tag{12}$$

for $b \geq 4$ and

$$j > 0,\ n_j^{(t+1)} \in [-1, 1]$$

for $b = 2$. Since both membership intervals are contained in $[-b, b-1]$, we have established the invariance of (1) for $t = 0, 1, \ldots$.

Precision required on the first digit Relation (4) indicates that digit q_t of the quotient has been *estimated* as $n_0^{(t)} \div d_0$ rather than being *computed* as $N^{(t)} \div D$. However, we now show that by appropriately controling the membership intervals of $n_0^{(t)}$ and d_0, the current quotient Q_t differs from the exact quotient $N \div D$ by less than $\frac{1}{b^t}\frac{b}{2(b-1)}$. This bound follows directly from (3). In order for (3) to remain invariant throughout the division process, the following relation between $n_0^{(t)}$ and d_0 must hold:

$$n_0^{(t)} \in [-\frac{b+1}{2}d_0, \frac{b+1}{2}d_0 - 1]. \tag{13}$$

Using (13) as an inductive hypothesis, we show that it holds for $t+1$ as well. Indeed, (7) can be rewritten as

$$n_0^{(t+1)} = br_0^{(t+1)} + r_1^{(t+1)} + c_2^{(t+1)}.$$

From (9) and (11), we have

$$\left|r_1^{(t+1)} + c_2^{(t+1)}\right| \leq \frac{b^2+5b+4}{4}.$$

Using the fact that $\left|r_0^{(t+1)}\right| \leq d_0/2$,

$$\left|n_0^{(t+1)}\right| \leq \frac{b}{2}d_0 + \frac{b^2+5b+4}{4}.$$

Combining this last inequality with (13), we obtain

$$\frac{b}{2}d_0 + \frac{b^2+5b+4}{4} \leq \frac{b+1}{2}d_0,$$

hence

$$d_0 \geq \frac{b^2+5b+4}{2}. \tag{14}$$

This certainly holds for:

$$\begin{aligned} b \geq 8, \quad d_0 &\geq b^2; \\ b = 4, \quad d_0 &\geq 32; \\ b = 2, \quad d_0 &\geq 8. \end{aligned}$$

Correspondingly, relation (13) establishes the membership interval of $n_0^{(t)}$.

Remark 1 *Relation (12) shows that for $b \geq 4$, the membership interval of $n_j^{(t-1)}$ for $j \geq 1$ is contained in $[-b, b-1]$, as specified by (1) . This suggests that the membership intervals of either d_j or q_t may be enlarged without significantly affecting any other detail of the algorithm. Indeed, it is easy to verify that we may adopt for d_j or q_t any interval I such that:*

$$[-\frac{b}{2}, \frac{b}{2}] \subseteq I \subseteq [-b+1, b-1].$$

Example of Z-division An example of Z-division is given in Figure 1. The operands, N and D, are represented in decimal by:

$$N = 156411, D = 5793,$$

and, in base $b = 4$, by:

$$N = 2\,1\,2\,0\,2\,3\,3\,2\,3, \; D = 1\,1\,2\,2\,2\,0\,1. \tag{15}$$

After normalization, the operands are initially presented as:

$$N = 38.0\,2\,3\,3\,2\,3, \; D = 23.\bar{1}\,\bar{2}\,0\,1.$$

The resulting quotient $Q = 27$ is computed, in base 4, by the algorithm in the form: $Q = 2\,\bar{1}\,\bar{1}$.

t	$q^{(t)}$	$R^{(t)}$	$N^{(t)}$	$-q^{(t)}D^{(t)}$
0	2		$38.0\,2\,3\,3\,2\,3$	$\overline{46}.2\,4\,0\,\bar{2}$
1	$\bar{1}$	$\bar{8}.2\,6\,3\,1\,2\,3$	$\overline{28}.\bar{1}\,\bar{1}\,2\,\bar{1}\,\bar{1}$	$23.\bar{1}\,\bar{2}\,0\,1$
2	$\bar{1}$	$\bar{5}.\bar{2}\,\bar{3}\,2\,0\,\bar{1}$	$\overline{23}.2\,\bar{2}\,0\,\bar{1}$	$23.\bar{1}\,\bar{2}\,0\,1$
3		$0.1\,\bar{4}\,0\,0$	$0.0\,0\,0$	

Figure 1: Example of Z-division

3.3 Algorithm for N-division

The algorithm described in this section is based on different selections of the intervals for the parameters $n_j^{(t)}$, d_j and q_t, and of the division criterion. In the notation of the preceding section, we have:

$$j > 0, \; d_j \in [0, b-1]; \tag{16}$$

$$j > 0, \; n_j^{(t)} \in [-b+1, b-1]; \tag{17}$$

$$q_t \in [-b+1, b-1]. \tag{18}$$

The quotient digit q_t is computed by N-division as follows:

$$q_t \;=\; sign(n_0^{(t)}) \times (\left|n_0^{(t)}\right| \; quo \, d_0). \tag{19}$$

As a consequence, the sign of $r_0^{(t+1)} = n_0^{(t)} - q_t \times d_0$ is identical to that of $n_0^{(t)}$, and $\left|r_0^{(t+1)}\right| \leq d_0 - 1$.

We now distinguish three cases, depending upon the sign ($> 0, = 0, < 0$) of q_t:

$\boxed{q_t > 0}$ In this case, we first compute

$$j \geq 1,\; r_j^{(t+1)} \;=\; n_j^{(t)} - d_j \times q_t,$$

and obtain the terms $c_j^{(t+1)}$ and $s_j^{(t+1)}$ according to the N-division:

$$c_j^{(t+1)} = r_j^{(t+1)} \, quo \, b \quad , \quad s_j^{(t+1)} = r_j^{(t+1)} \, mod \, b. \tag{20}$$

From (17),(16),(18) we immediately have

$$j > 0,\; r_j^{(t+1)} \;\in\; [-b(b-1), b-1],$$

and from (20):

$$c_j^{(t+1)} \in [-b+1, 0] \quad , \quad s_j^{(t+1)} \in [0, b-1].$$

From these, we obtain

$$j > 0,\; n_j^{(t+1)} \;\in\; [-b+1, b-1],$$

which shows that (17) is an invariant of the method.

Moreover, we repeat below the expression for $n_0^{(t+1)}$:

$$n_0^{(t+1)} \;=\; b r_0^{(t+1)} + r_1^{(t+1)} + c_2^{(t+1)}.$$

Bounding $n_0^{(t+1)}$ from above, we obtain

$$n_0^{(t+1)} \;\leq\; b(d_0 - 1) + b - 1 = bd_0 - 1;$$

bounding it from below, we have $n_0^{(t+1)} \geq -b(b-1) - (b-1) = -b^2 + 1$, so that

$$n_0^{(t+1)} \;\in\; [-b^2 + 1, bd_0 - 1].$$

$\boxed{q_t < 0}$ In this case, symmetric to the previous one, we compute the terms $c_j^{(t+1)}$ and $s_j^{(t+1)}$, $j > 0$ according to the rule:

$$-c_j^{(t+1)} = -r_j^{(t+1)} \, quo \, b \quad , \quad -s_j^{(t+1)} = -r_j^{(t+1)} \, mod \, b. \tag{21}$$

Arguing as before, we find that:

$$j > 0,\; n_j^{(t+1)} \;\in\; [-b+1, b-1],$$

and

$$n_0^{(t+1)} \;\in\; [-bd_0 + 1, b^2 - 1].$$

$\boxed{q_t = 0}$ In this case, we let

$$n_j^{(t+1)} = n_{j-1}^{(t)},$$

for $j \geq 1$, and

$$n_0^{(t+1)} = bn_0^{(t)} + n_1^{(t)}.$$

We note first that invariant (17) trivially holds. Moreover, $q_t = 0$ implies that

$$n_0^{(t)} \in [-d_0 + 1, d_0 - 1].$$

Combining with the above definition of $n_0^{(t+1)}$, we readily obtain:

$$n_0^{(t+1)} \in [-bd_0 + 1, bd_0 - 1].$$

Merging the analyses of the three cases yields:

$$n_0^{(t+1)} \in [-max(b^2 - 1, bd_0 - 1), max(b^2 - 1, bd_0 - 1)];$$

condition (18) thus holds as long as $bd_0 - 1 \geq b^2 - 1$, that is

$$d_0 \geq b. \tag{22}$$

Remark 2 *For $b = 2$, the N-division scheme offers a simple algorithm which is an alternative to the one described in [TYY]. The characterizing features of this technique are relations (20), (21) and (22).*

Remark 3 *The Z- and N-divisions can be contrasted as follows. The Z-division has a smaller interval for the quotient digits ($[-\frac{b}{2}, \frac{b}{2} - 1]$ versus $[-b + 1, b - 1]$), while it requires one more digit for d_0 ($d_0 \geq b^2$ versus $d_0 \geq b$). This means that the Z-algorithm simplifies the coefficients update, at the expense of a costlier computation of each quotient digit q_t.*

Remark 4 *The N-division appears to require that the divisor be represented in nonredundant form. This should be contrasted with Remark 1, and will prove to be a handicap in the applications.*

Example of N-division An example of N-division is given in Figure 2. The operands, N and D, are those of the previous example (15). After normalization, they are initially presented as:

$$N = 9.2023323, \; D = 5.22201.$$

The resulting quotient $Q = 27$ is computed, in base 4, by the algorithm in the form: $Q = 1\,3\,\overline{1}$.

t	$q^{(t)}$	$R^{(t)}$	$N^{(t)}$	$-q^{(t)}D^{(t)}$
0	1		9.2023323	$\bar{5}.\bar{2}\bar{2}\bar{2}0\bar{1}$
1	3	$4.0\bar{2}03223$	15.203223	$\bar{1}\bar{5}.\bar{6}\bar{6}\bar{6}0\bar{3}$
2	$\bar{1}$	$0.\bar{4}\bar{6}\bar{3}2\bar{1}3$	$\bar{5}.\bar{2}\bar{2}\bar{2}0\bar{1}$	5.22201
3		0.00000	0.0000	

Figure 2: Example of N-division

4 Implementation

In this section, we consider some network implementations, and limit our considerations to the algorithm for Z-division (Section 3.2). The approach is readily extensible to the N-division (Section 3.3).

For convenience, we repeat below the operations to be executed, for $t = 0, 1, 2, \ldots$.

- For $j = 0$:

$$q_t = n_0^{(t)} \div d_0; \tag{23}$$
$$r_0^{(t+1)} = n_0^{(t)} \cdot|\cdot d_0; \tag{24}$$
$$n_0^{(t+1)} = (r_0^{(t+1)} b + s_1^{(t+1)}) + (c_1^{(t+1)} b + c_2^{(t+1)}). \tag{25}$$

- For $j > 0$:

$$r_j^{(t+1)} = n_j^{(t)} - d_j q_t; \tag{26}$$
$$s_j^{(t+1)} = r_j^{(t+1)} \cdot|\cdot b; \tag{27}$$
$$c_j^{(t+1)} = r_j^{(t+1)} \div b; \tag{28}$$
$$n_j^{(t+1)} = s_{j-1}^{(t+1)} + s_{j+2}^{(t+1)}. \tag{29}$$

We also recall the representation adopted for the operand digits.

- The dividend is one-bit redundant: $j \geq 1$, $n_j \in [-b, b-1]$.

- The divisor is signed-digit irredundant: $j \geq 1$, $d_j \in [-\frac{b}{2}, \frac{b}{2} - 1]$.
- The quotient is one-value redundant: $j \geq 0$, $q_j \in [-\frac{b}{2}, \frac{b}{2}]$.

4.1 Initialization of the operands

If we assume that the operands are stored in memory in conventional binary form, i.e. unsigned-digit irredundant form, it is necessary to perform a preliminary conversion of the dividend and of the divisor. The dividend is initialized by inserting a 0 bit to the left of each b-ary digit represented in the usual form; the inserted digit becomes the sign bit in two's complement representation. The initialization of the divisor requires more work. Indeed, we must change each digit $d_j \in [\frac{b}{2}, b-1]$ to $d_j - b$, and propagate a carry. This conversion is equivalent to an addition; it can be performed, digit after digit starting from the least significant, in time proportional to the divisor length. For both dividend and divisor, the initialization can be simply merged with the normalization phase, all in linear time. An alternative, based on remark 1, is to simply convert d to a redundant representation, in constant time.

4.2 Network operations

The above operations (23)-(29) immediately identify two types of modules; the *front-end* module ($j = 0$), which computes the next quotient digit, and the *standard* multiplier modules ($j \geq 1$).

4.2.1 Broadcast implementation

The most straightforward implementation of the algorithm is one where q_i is broadcast by the front-end to all the standard modules, which all execute (26) simultaneously. It has the advantage of simplicity, but it falls short of balancing the workload between front-end and standard modules. Indeed, operations must be performed in the order

$$(23),(24);(25),(26);(27),(28);(29)$$

where ";" and "," respectively separate parallel and sequential steps. Since n_0 has 3 and d_0 2 digits, the most time consuming operations are (23) and (25), both to be performed sequentially by the front-end. Furthermore, the front-end must be able to broadcast q_i, which physically requires time asymptotically proportional to length of the global communication link, hence to the divisor length d [MC]. This realization is thus not modular electrically; in view of its conceptual simplicity, it is however likely to be implemented in practice for *moderate* values of n.

4.2.2 Systolic implementation

An alternative implementation has a *systolic* nature. In this solution, the quotient digits migrate from the front-end along the chain of neighborly connected standard modules. The connection is modular and the control exclusively local.

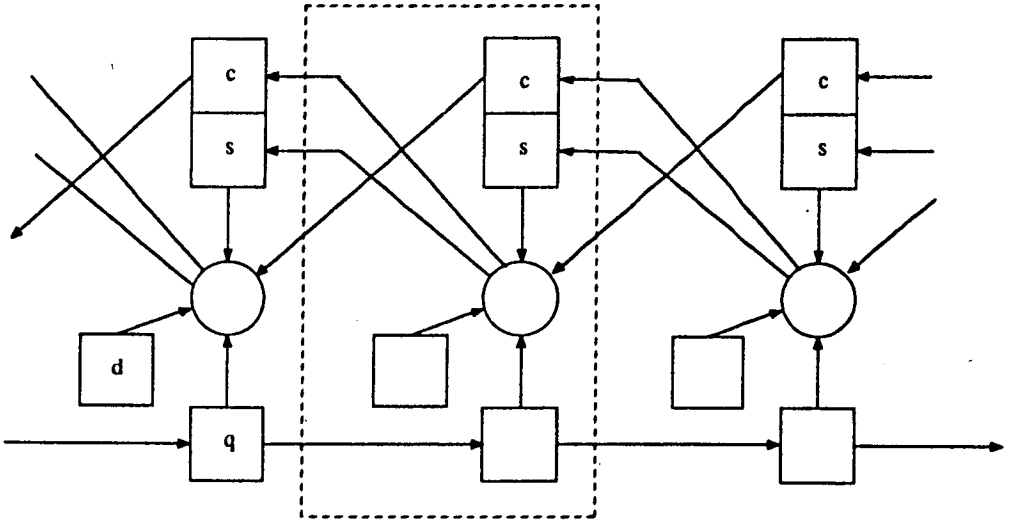

Figure 3: Physical interconnection of the standard module chain.

Standard Modules We begin by considering the standard modules. Using (26),(27) and (28), we first rewrite (29) as

$$n_j^{(t-1)} = (n_{j-1}^{(t)} - d_{j+1}q_t) \cdot |\cdot b + (n_{j-2}^{(t)} - d_{j-2}q_t) \div b$$

This expression reveals that, in order to compute $n_j^{(t+1)}$, the correct values of $n_{j-1}^{(t)}$ and $n_{j-2}^{(t)}$ must be available, i.e., the digit q_{t-1} of the quotient must have proceeded beyond the position of weight b^{-j-2}. This means that q_{t-1} must be at least *three cells ahead* of q_t. This spacing of the quotient digits is due to the carry save feature, independently of the base.

For simplicity, terms $n_j^{(t)}$ $(j \geq 1, t \geq 0)$ are represented implicitly by the pair $c_j^{(t)}$ and $d_j^{(t)}$, related to $n_j^{(t)}$ by (29), with the initializing conventions $c_j^{(0)} = 0$ and $s_j^{(0)} = n_j^{(0)}$ for $j \geq 1$. As indicated above, quotient digits are present in a module only one time unit out of three; this time unit is said to be *active* for that module. Thus, during its active time unit for q_t, module M_j performs the following action:

$$c_{j-1}^{(t+1)} = (s_j^{(t)} + c_{j-1}^{(t)} - d_j q_t) \div b \tag{30}$$

$$s_{j-1}^{(t+1)} = (s_j^{(t)} + c_{j-1}^{(t)} - d_j q_t) \cdot |\cdot b \tag{31}$$

To prove correctness, we assume that q_t is available at module $M_k, k \geq 2$ at time $3t+k+1$, for all $t \geq 0$. Next, we assume inductively that the pair $(s_k^{(p)}, c_k^{(p)})$ is available at time $3p+k$, for all $0 \leq p < t$. Then, by (30),(31), $s_j^{(t)}$ is available at time $3t+j$ and $c_{j-1}^{(t)}$ at time $3t+j+1$. This and the assured availability of q_t imply that, one time unit later, i.e. at time $3t+j+2 = 3(t+1)+j-1$, the values of $c_{j-1}^{(t+1)}$ and $s_{j-1}^{(t+1)}$ can be computed, thereby extending the inductive hypothesis.

It follows that each standard module contains four registers, respectively for digits c, s, q and d. A fragment of the standard modules chain is shown in Figure 3. The following is a timing diagram of the operations of modules M_j, M_{j-1} and M_{j-2}, where we display values only at the times of their updates in the module registers:

time	M_j	M_{j-1}	M_{j-2}
$3t+j-1$		q_{t-1}	
$3t+j$	$c_j^{(t)}\ s_j^{(t)}$		q_{t-1}
$3t+j+1$	q_t	$c_{j-1}^{(t)}\ s_{j-1}^{(t)}$	
$3t+j+2$		q_t	$c_{j-2}^{(t)}\ s_{j-2}^{(t)}$
$3t+j$	$c_j^{(t+1)}\ s_j^{(t+1)}$		q_t

Front end Modules We now consider the structure of the front-end section of the network. Refering to (25), we note that $c_2^{(t-1)}$ is available two time units after the generation of q_t. In order to feed the standard modules at the stipulated rate, q_{t-1} must be produced three time units after q_t. Therefore, we only have one time unit to carry out the computations:

$$q_{t-1} = n_0^{(t+1)} \div d_0$$

$$n_0^{(t+1)} = (r_0^{(t-1)} b + s_1^{(t+1)}) + (c_1^{(t+1)} b + c_2^{(t+1)})$$

Considering the large size of the operands $n_0^{(t+1)}, r_0^{(t-1)}$ and d_0, this objective can only be achieved by adequately lengthening the duration of the time unit, thereby resulting in a global slowdown.

To circumvent this difficulty, we modify the computation embodied by (23)-(25) in such a way that the update of $n_0^{(t)}$ and the calculation of of q_t can be conveniently overlapped, while preserving the essential features of the technique. Specifically, let q_t now denote the values of the quotient digits obtained by the modified technique. Rather than performing the calculation of $s_1^{(t+1)}$ and $c_1^{(t+1)}$ as specified by (26)-(29), we directly add the sum $c_1^{(t)}b + s_1^{(t)} + c_2^{(t)}$, available at time $3t+2$, to the (shifted) current remainder $r_0^{(t)}$. The resulting sum has a discrepancy $-d_1 q_t$ from the correct value; however, since q_t is available only at time $3t+3$, it is necessary to wait for the next digit cycle to add the appropriate correction, which must then be multiplied by b. Thus we have the modified algorithm

$$n_0^{(t+1)} = (r_0^{(t)}b + c_1^{(t)}b + s_1^{(t)} + c_2^{(t)}) - q_{t-1}d_1 b \tag{32}$$

$$q_{t-1} = n_0^{(t+1)} \div d_0 \tag{33}$$

$$r_0^{(t-1)} = n_0^{(t+1)} - d_0 q_{t-1} \tag{34}$$

with the convention that $c_1^{(-1)} = s_1^{(-1)} = c_2^{(-1)} = q_{-2} = q_{-1} = 0$ and $r_0^{(-1)} = n_0/b$.

First, we prove the correctness of the method, and later illustrate its implementation. Indeed, by (33) and (34) q_t and $r_0^{(t)}$ are such that $q_t \in [-b/2, b/2]$ and $r_0^{(t)} \in [-d_0/2, d_0/2-1]$ whence by (2),(27),(28), and (32):

$$\left|n_0^{(t+1)}\right| \leq b\frac{d_0}{2} + \frac{(b+1)(b+4)+b(b^2+2)}{4}.$$

The method is correct if $n_0^{(t-1)}$ satisfies (13), or, equivalently, if

$$d_0 \geq \frac{(b+1)(b+4)+b(b^2+2)}{2}$$

for base $b \geq 4$. In the special case $b = 2$, we have $d_0 \geq 16$.

In this modified algorithm, the term $r_1^{(t+1)}$ is not computed according to the general relation (23), but rather we set $r_1^{(t+1)} = n_1^{(t+1)}$. This shows that module M_1 is somewhat different from the other standard modules; thus the front-end section consists of the pair M_0, M_1, whose structure we now consider in detail. We allot two time units to carry out operations (32), (33) and (34), concurently with the calculation of

$$\sigma^{(t)} = c_1^{(t)}b + s_1^{(t)} + c_2^{(t)} - q_{t-1}d_1 b$$

A third time unit is used to form $n_0^{(t+1)} = r_0^{(t)}b + \sigma^{(t)}$. A network diagram is given in Figure 4 and the following provides a timing diagram of the front-end modules M_0 and M_1:

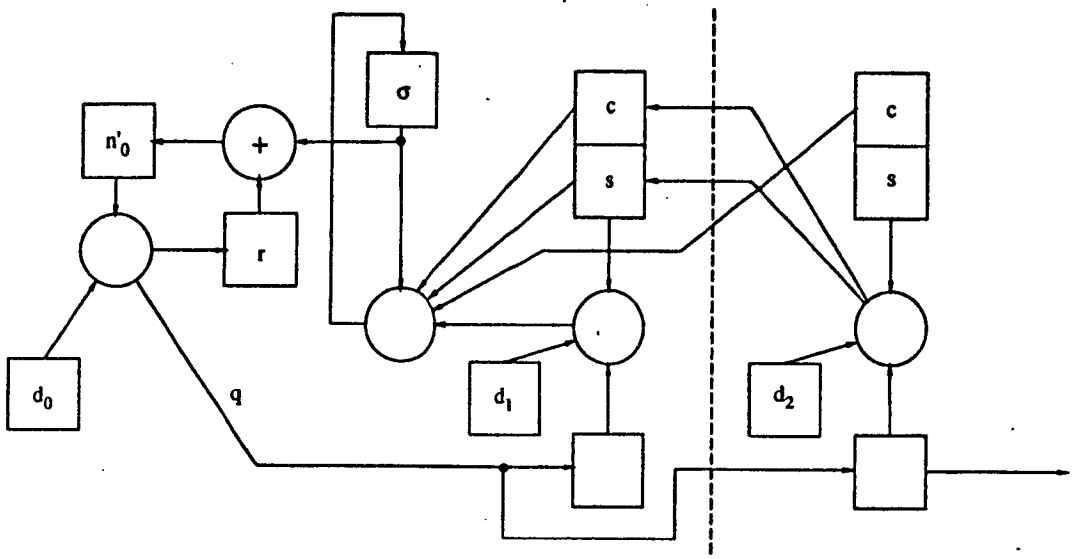

Figure 4: Circuit diagram of the front-end modules M_0 and M_1.

time	M_0	M_1	M_2	M_3
0	$r_0^{(-1)} = n_0^{(0)}/b$	$c_1^{(0)} = 0, s_1^{(0)} = n_1^{(0)}$	$c_2^{(0)} = 0, s_2^{(0)} = n_2^{(0)}$	
1	$n_0^{(0)}$			
2	$bc_1^{(0)} + s_1^{(0)}$	q_0		
3	$r_0^{(0)}, \sigma_0^{(0)}$		q_0	
4	$n_0^{(1)}, -bd_1q_0$	$c_1^{(1)}, s_1^{(1)}$		q_0
5	$bc_1^{(1)} + s_1^{(1)} - bd_1q_0$	q_1	$c_2^{(1)}, s_2^{(1)}$	
6	$r_0^{(1)}, \sigma_0^{(1)}$		q_1	
7	$bc_1^{(2)} + s_1^{(2)} - bd_1q_0$		$c_2^{(2)}, s_2^{(2)}$	q_1

Note that this diagram establishes that q_t is present at time $3t+k+1$ at module M_k.

We conclude this section by considering the bases $b=2$ and $b=4$, the most attractive ones from a hardware designer viewpoint.

Base b=2 In this case, the range values of operand digits are

$$s_j \in [\bar{1},0], c_j \in [\bar{1},1], d_j \in [\bar{1},0], q_j \in [\bar{1},1], d_0 \in [16,31], n_0 \in [\bar{77},77]$$

Thus n_0 and d_0 are respectively represented with 5 and 8 bits, which makes the implementation of (33) ameanable to standard random logic design, such as PLA. The standard module operation is also very simple, since $q_j \in [\bar{1},1]$.

Base b=4 In this case, the range values of operand digits are

$$s_j \in [\bar{2},1], c_j \in [\bar{2},2], d_j \in [\bar{2},1], q_j \in [\bar{2},2], d_0 \in [64,127], n_0 \in [\bar{3}\bar{1}\bar{7},317]$$

While the operation of the standard module is still simple since $q_j \in [\bar{2},2]$, operation (33) is not likely to be PLA-implemented since the operands d_0 and n_0 have 7 and 10 bits. We propose to implement (33) by the method of repeated substractions. With this choice, module M_0 operates as shown below, where at time $(3t+1)$, ρ is inductively assumed to be equal to $n_0^{(t)}$.

τ	ϵ	ρ	σ	q
$3t+2$	$\epsilon := if\ \ \rho<0$ $then\ 1$ $else\ -1$	$\rho' = \rho - \epsilon d_0/2$ $if\ \ \epsilon\rho' \geq 0$ $then\ \rho := \rho'$	$\sigma := \sigma + 4c_1 + s_1$	$if\ \ \epsilon\rho' \geq 0$ $then\ q := q+\epsilon$
$3t+3$		$\rho' = \rho - \epsilon d_0$ $if\ \ \epsilon\rho' \geq 0$ $then\ \rho := \rho'$	$\sigma := \sigma + c_2 - 2\epsilon d_0$	$if\ \ \epsilon\rho' \geq 0$ $then\ q := q+\epsilon$
$3t+4$		$\rho = 4\rho + \sigma$	$\sigma := -4d_1 q$	$q := 0$

Illustration of the operation of module M_0.

Example of Systolic Z-division An example of systolic Z-division is given in Figure 5. The operands, N and D, are those of the previous example (15). After normalization, they are initially presented as:

$$\begin{aligned} N &= 152.233\,23 \\ D &= 90.201 \end{aligned}$$

The quotient $Q=27$ is computed by the systolic Z-division algorithm in the form $2-1-1$.

time	M_0				M_1			M_2			M_3			M_4			M_5		
	ϵ	ρ	σ	q	c	s	q	c	s	q	c	s	q	c	s	q	c	s	q
1		152	0	0	0	2		0	3		0	3		0	2		0	3	
2	1	107	2	1															
3		17	$\overline{178}$	2			2			2									
4		$\overline{110}$	$\overline{16}$	0	1	$\overline{1}$							2						
5	$\overline{1}$	$\overline{65}$	$\overline{13}$	$\overline{1}$				0	1							2			
6		$\overline{65}$	167	$\overline{1}$			$\overline{1}$			$\overline{1}$	1	$\overline{2}$							2
7		$\overline{93}$	8	0	1	$\overline{2}$							$\overline{1}$	1	$\overline{1}$				
8	$\overline{1}$	$\overline{48}$	10	$\overline{1}$				0	0							$\overline{1}$			
9		$\overline{48}$	190	$\overline{1}$			$\overline{1}$			$\overline{1}$	0	1							
10		$\overline{2}$	8	0	0	0							$\overline{1}$						
11	$\overline{1}$	$\overline{2}$	8	0				0	0										
12		$\overline{2}$	8	0				0	0										

Figure 5: An example of systolic Z-division

5 Applications

The division techniques of Section 3 can be adapted to several significant arithmetic computations. We shall consider integer multiplication, modular integer multiplication and exponentiation, and greatest-common-divisor (gcd) calculation.

5.1 Integer multiplication

Let integers A and B be the multiplier and the multiplicand, respectively represented in base b by

$$A \equiv a_{n-1}a_{n-2}\cdots a_0 \quad \text{and} \quad B \equiv b_{n-1}b_{n-2}\cdots b_0.$$

We want to compute the product

$$P = A \times B \equiv p_{2n-1}p_{2n-2}\cdots p_0.$$

As in Section 3, we assume that both operands A and B are represented either in the signed-digit-nonredundant form, or in the one-value-redundant form. We initially set $n_j^{(0)} = 0$ for $j \geq -2$, and $d_j = b_{n-j}$ for $1 \leq j \leq n$, with $d_{-2} = d_{-1} = d_0 = 0$.

Subsequently, we carry out $2n + 2$ steps of the Z-division algorithm, replacing the determination of the quotient digits (4) by:

$$q_t = \begin{cases} -a_{n-t} & \text{if } 1 \leq t \leq n, \\ 0 & \text{if } n+1 \leq t \leq 2n+2. \end{cases}$$

Relations (5) and (6) are extended to $j \geq -1$, while (8) is extended to $j \geq -2$. Moreover, digit n_{-3} is updated as follows:

$$n_{-3}^{(t+1)} = n_{-2}^{(t)} + c_{-1}^{(t+1)}. \tag{35}$$

We now establish the correctness of the method. For $b \geq 4$, we prove by induction on t that $n_j^{(t)} \in [-\frac{3}{4}b - 1, \frac{3}{4}b]$ for all $t \geq 0$ and $j \geq -1$:

- The base of induction is trivially established for $t = 0$.
- For $t + 1$, we have $r_j^{(t+1)} \in [-\frac{b^2}{4} - \frac{3}{4}b - 1, \frac{b^2}{4} + \frac{3}{4}b]$ for $j \geq 1$, while $r_i^{(t+1)} = n_i^{(t-1)} \in [-\frac{3}{4}b - 1, \frac{3}{4}b]$ for $i = -1, 0$. It follows that $s_j^{(t+1)} \in [-\frac{b}{2}, \frac{b}{2} - 1]$ for $j \geq -1$, and $c_j^{(t+1)} \in [-\frac{b}{4} - 1, \frac{b}{4} + 1]$ for $j \geq 0$. This and relation (8) establish the claim.

Since, for $i \leq 0$, $r_i^{(t+1)} = n_i^{(t-1)}$, relation (6) yields $c_i \in [-1, 1]$. It follows that $n_{-2}^{(t-1)} \in [-\frac{b}{2} - 1, \frac{b}{2}]$, whence by (35), $n_{-2} \in [-\frac{b}{2} - 2, \frac{b}{2} + 1] \subseteq [-b, b - 1]$, for $b \geq 4$. An analogous, more specialized, argument can be developed for the base $b = 2$. This shows that carries never propagate beyond the position of index -2. Since the t-th step of the computation yields the partial product

$$P^{(t)} = bP^{(t-1)} + a_{n-t} \times B$$

for $t > 0$, with $P^{(0)} = 0$, clearly $n_{-3}^{(t-3)}$ is the t-th digit of the product for $0 < t \leq 2n$, counting from the most significant end. The product is therefore available in one-bit-redundant form, starting from the most significant digit.

This multiplier has both area and time $O(n)$, as for the well-known Muller-Atrubin multiplier [A],[Mu].

5.2 Modular integer multiplication and exponentiation

Let A, B, and M be three integers. We want to compute the modular product

$$P = A \times B \bmod M.$$

Initially, the operands are represented with n digits in base-b signed-irredundant form. We assume, without loss of generality, that $M \geq (b/2)b^{n-1}$. Operands A and B are used as multiplier and multiplicand, respectively. Integers B and M, the multiplicand and the modulus, must be normalized so that the most significant digit of M is sufficiently large, while the corresponding digit of B is zero. Specifically, we have:

$$A = \sum_{0 \leq j < n+\sigma-1} a_j b^{-j},\ B = \sum_{1 \leq j \leq n} b_j b^{-j},\ M = \sum_{0 \leq j \leq n} m_j b^{-j},$$

with $m_0 > 0$. The choice of the interval membership of m_0, to be made shortly, defines a unique integer $\sigma > 0$ such that b^σ gives the relative shift of M versus B. Correspondingly,

A is also multiplied by b^σ, i.e. σ zero digits are appended to the right of its representation. In conclusion, we are effectively carrying out the calculation:

$$Pb^\sigma = (Ab^\sigma) \times B \pmod{Mb^\sigma}.$$

The computation is performed in $n+\sigma$ iterations, designated by index $t \geq 1$, and using variables $P^{(t)}, U^{(t)}, u_0^{(t)}$ and q_t as follows:

$$\begin{aligned}
U^{(t-1)} &= P^{(t-1)} + a_{n+\sigma-t} \times B; && (36)\\
u_0^{(t-1)} &= \text{coefficient of } b^0 \text{ in the representation of } U^{(t)}; && (37)\\
q_{t-1} &= u_0^{(t-1)} \div m_0; && (38)\\
P^{(t)} &= b(U^{(t)} - q_{t-1} \times M); && (39)
\end{aligned}$$

with $P^{(0)} = 0$. We thus alternate between one multiplication (36) and one division (38,39) step. However, the left-shift step is performed only once, after each division step. Obviously, the divisor register is now replaced by two registers, one for the multiplicand, the other for the modulus. Total area and time remain $O(n)$.

To ensure correctness, the interval for m_0 must be chosen so that:

$$u_0^{(t)} \in [-\frac{b+1}{2}m_0, \frac{b+1}{2}m_0 - 1].$$

Let $p_0^{(t)}$ denote the coefficient of b^0 in the representation of $P^{(t)}$. Using the notations of Section 3.2, operations (38,39) yield

$$p_0^{(t)} = br_0^{(t)} + r_1^{(t)} + c_2^{(t)};$$

since $b_0 = 0$ by normalization, we obtain from (36)

$$u_0^{(t)} = br_0^{(t)} + r_1^{(t)} + c_2^{(t)} + g_1^{(t)},$$

where $g_1^{(t)}$ is the carry in position b^0 resulting from addition (36). Knowing that $|r_0^{(t)}| \leq m_0/2$, and that $c_2^{(t)}, g_1^{(t)}$ both belong to $[-\frac{b}{4} - 1, \frac{b}{4} + 1]$, we obtain the condition

$$m_0 \geq \frac{b^2 + 6b + 5}{2},$$

which is analogous to (14).

The above technique for modular integer multiplication can be used to implement *modular exponentiation*, as required by [RSA] cryptography. Specifically, denoting by A, M, and E three integers, we wish to compute

$$C = A^E \bmod M.$$

If E is represented in *binary* as

$$E = e_{n-1}e_{n-2}\cdots e_0,$$

with $e_{n-1} = 1$, the computation of C is carried out by the following routine:

```
1  R    := A;
2  for  j = n - 1  downto 1 do
3       begin      R := R × R mod M;
4                  if e_{j-1} = 1 then R := R × A mod M;
       end.
```

Since both Steps 3 and 4 take time $O(n')$, where n' is the number of digits of A and M, the RSA-residue C is computed in time $O(nn')$.

5.3 Computation of the Greatest common divisor

Our last application will be to compute the greatest common divisor $gcd(N, D)$ of integers N and D, using Euclid's algorithm, with each step performed by Z-division. Let

$$N \equiv n_{n-1}\cdots n_0, D \equiv d_{d-1}\cdots d_0$$

denote the base-b two-value-redundant form (see Remark 1) of the two operands, i.e.:

$$j \geq 0,\ d_j, n_j \in [-\frac{b}{2} - 1, \frac{b}{2}].$$

Without loss of generality, we assume $|N| \geq |D|$ with $n_{n-1} \neq 0$. One iteration of the process consists of the following steps:

1. While the d-th most significant digit of D is zero, left shift D, and decrease its length by 1:
$$D := bD, d := d - 1.$$

2. If $d = 0$ (i.e. the dividend D is null) terminate the computation with $gcd = N$.

3. Otherwise (when $d > 0$) perform $n - d$ steps of Z-division, leaving the remainder $R = N\ mod\ D$ in place of N. By (1), R is one-bit redundant:
$$j \geq 1,\ r_j \in [-b, b - 1].$$

4. We perform one step of carry propagation; this can be achieved by computing one division step with partial quotient $q = 0$, but, without shifting the operand. As a result of (6), R is now in two-value-redundant form:
$$j \geq 1,\ r_j \in [-\frac{b}{2} - 1, \frac{b}{2}].$$

At this point, we exchange the respective positions and roles of both operands:

$$N := D;\ D := R;\ n := d;$$

and proceed to the next iteration.

Each shift in Step 1, and each division in Step 2 reduces the total operand length $n+d$ by one unit. Step 3 is performed only once per stage in Euclid's algorithms, for a total which is well-known [Kn] to be of at most $\log_\varphi D < 2d$. It follows that the above algorithm for computing *gcd* as both *time and space linear in the length of the operands.*

6 Bibliography

[**A**] A. J. Atrubin *An iterative one-dimensional real-time multiplier* , IEEE Trans. on Computers , 14, 3, 394-399 ; 1965.

[**BK**] R. P. Brent, H. T. Kung *A systolic algorithm for integer GCD computation* , VLSI '83 (edited by F. Anceau and E. J. Aas) , North-Holland, 145-154 ; 1983.

[**BCH**] P. W. Beame, S. A. Cook and H. J. Hoover *Log depth circuit for division and related problems* , 25-th FOCS Symposium , 1-6, 1984.

[**CH**] M. Coppa and V. C. Hamacher *An augmented iterative array for high-speed binary division,* IEEE trans. on Computers , C-22, 172-175 ; February 1973.

[**H**] K. Hwang, Computer Arithmetic, J.Wiley, N.Y. 1979.

[**Kn**] D. E. Knuth *The Art of Computer Programming, v.2, Seminumerical Algorithms:* Addison Wesley, 1981.

[**Ku**] H. T. Kung *Why systolic architectures?* IEEE Computer, vol. 15 , 37-46; February 1982.

[**L**] R. F. Lyon *Two's complement pipeline multipliers* , IEEE Trans. Comm. , COM-24: 418-425, 1976.

[**CM**] C. A. Mead and L. A. Conway *Introduction to VLSI Systems* , Addison-Wesley , Reading, Mass., 1980.

[**M**] K. Mehlhorn *AT^2-optimal VLSI integer division and integer square routing,* Integration , 2, 163-167, 1984.

[**MP**] K. Mehlhorn and F. P. Preparata *Area-time optimal division for $T = \Omega(log^{1-\epsilon})$* , Information and Computation , 1987.

[Mu] D. E. Muller *Asynchronous logic and applications to information processing* , Switching theory and Space Technology, (Hiken and Marin, eds.), Stanford University Press, 1963.

[PP] C. N. Purdy, G. B. Purdy *Integer division in linear time with bounded fan-in* , IEEE Trans. on Computers, C-36,5,640-644, 1987.

[RSA] R. L. Rivest, A. Shamir, L. Adleman *Public key cryptography* , CACM 21, 120-126, 1979.

[R] J. E. Robertson *A new class of digital division methods,* IEEE Trans. on Comput. , C-7, 218-222 ; September 1958.

[TYY] N. Takagi, H. Yasuura and S. Yajima *A VLSI-oriented high-speed divider using redundant binary representations* , Trans. of IECE of Japan , J67D, 4, 450-457, April 1984.

[VG] J.E. Vuillemin and L. Guibas *A fast n-MOS implementation of addition* , Proc. ICCC82 Conf. on Circuits and Computers , 147-150, 1982.

Contents