Consiglio Nazionale delle Ricerche

ISTITUTO DI ELABORAZIONE
DELLA INFORMAZIONE

PISA

THE VLSI RESIDUE MULTIPLICATION AND ITS IMPLICATION IN THE DIRECT AND REVERSE POSITIONAL-TO-RESIDUE CONVERSION

Giuseppe Alia and Enrico Martinelli

Progetto finalizzato "Materiali e Dispositivi per l'Elettronica a Stato Solido"

Nota interna B4-70

Dicembre 1986

THE VLSI RESIDUE MULTIPLICATION AND ITS IMPLICATION IN THE DIRECT AND REVERSE POSITIONAL-TO-RESIDUE CONVERSION

Giuseppe ALIA and Enrico MARTINELLI

Progetto realizzato unitamente e [illegible]
[illegible]

AFFILIATION OF AUTHORS

Istituto di Elaborazione dell'Informazione

of the Italian National Research Council

via S. Maria, 46

56100 PISA, ITALY

The work described in this paper has been supported by the National Program on Solid-State Electronics and Devices of the Italian National Research Council.

INDEX TERMS

- area-time complexity
- positional-residue converters
- RNS multipliers
- table look-up
- VLSI

ABSTRACT

Computing structures based on Residue Number Sistems are very interesting because addition and multiplication are fast and modular and are well suited for VLSI implementation. In this paper the problem of multiplying two integers in residue representation is faced with and a new modulo m multiplier is defined. Such a multiplier can be integrated on a single chip with present VLSI technology and exhibits, for example, an expected response time of about 150 nseconds to multiply integers ranging up to 10^{12} using five 8-bit moduli. It allows any choice of moduli values and has considerably low complexity figures, if compared with ROM-based structures, which are generally considered the most suited for RNS-based systems.

Finally, new direct and reverse converters between binary positional representation and residue representation, exploiting the multiplier structure defined, are presented. They are the best solutions known under a wide range of hypotheses about RNS's.

I. Introduction

Computing by means of residue number systems (RNS) has been interesting many authors for a long time [1, 2, 3, 4, 5, 6]. This interest has arisen because RNS-based arithmetic units are fast and simple, at least for addition and multiplication. In fact, in residue arithmetic, these operations are carry-free, i.e. each residue digit of the result is only determined by the corresponding digits of both operands.

Special RNS-based hardware structures have been designed for particular applications (e.g., discrete Fourier transform processors and digital filters [7, 8, 9, 10]).

In the past, the use of RNS-based arithmetic units was limited by the inherent difficulty of achieving the same level of performance for operations such as division, and sign and magnitude detection, which cannot be carried out separately for each residue digit, and also by the heavy hardware requirements generally assumed for all operations, mainly in the form of large stored tables [1, 7, 11]. In the last few years, however, VLSI technology, permitting the integration of complex circuits on a single chip, has contributed towards making such architectures increasingly viable [12, 3, 9].

In this paper, the problem of multiplying two integers in residue representation is examined, and a new modulo m multiplication structure is defined, with the following characteristics:

a) with the current VLSI technology, the whole structure can be integrated in a single chip;

b) a typical total response time to multiply integers ranging up to 10^{12} using five 8-bit moduli is expected to be within about 150 nanoseconds; in pipeline, an interval of 70-100 nanoseconds can be achieved;

c) with respect to table look-up based structures, when moduli larger than 256 must be chosen, the defined structure becomes a mandatory choice;

d) unlike some previously known modulo m_i multipliers, any choice is permitted for the moduli values;

e) complexity figures are exhibited which are considerably lower than those for ROM-based structures in terms of area, whereas the response time remains of the same complexity; this result is independent of the number and the magnitude of moduli.

This last feature sets our structure against solutions based on table look-up techniques which are generally considered the most suitable to implement RNS-based digital systems with a large number of moduli.

Finally, new direct and reverse converters between binary positional number representation and residue number representation will be designed exploiting the VLSI multiplier structure defined. Both VLSI converters represent the best solution known for the particular problem, either in terms of area complexity or in terms of time complexity: for the area this is true at least as long as the number s of moduli, considered as a function of the number n of bits in the positional representation, respectively ranges in $[\vartheta(\text{const}), \vartheta(\log^3 n))$ and $[\vartheta(\text{const}), \vartheta(\log^3 n/\log\log n))$; on the other hand, for the time this result holds for any choice of s.

II. Residue Number Systems (RNS)

Let X be a non-negative integer number, $0 \leqslant X \leqslant 2^n - 1$, represented in the weighted binary system as

$$X \triangleq \{b_{n-1}, b_{n-2}, \ldots, b_1, b_0\} = \sum_{j=0}^{n-1} b_j 2^j , \qquad b_j \in \{0,1\}$$

In a residue number system, X is represented by s residue digits α_i:

$$X \triangleq \{\alpha_1, \alpha_2, \ldots, \alpha_s\}$$

where

$$\alpha_i = |X|_{m_i} = X - \lfloor N/m_i \rfloor . m_i ,$$

$\lfloor X/m_i \rfloor$ denotes the largest integer not exceeding X/m_i, and $\{m_i\}$ is the set of moduli. If numbers m_i are pairwise relatively prime, it can be shown [13] that there is a unique representation for each number X in the range

$$0 \leqslant X \leqslant \prod_{i=1}^{s} m_i = M.$$

In this paper, we assume that

$$n = 2^r \quad \text{and} \quad 2^n \leqslant \prod_{i=1}^{s} m_i < 2^{n+1}.$$

This means that we consider an RNS which has a range of representation, at most, twice as redundant as the binary range. Moreover, for any pair of integers X,Y, the following relations hold:

$$|X \pm Y|_{m_i} = ||X|_{m_i} \pm |Y|_{m_i}|_{m_i}, \qquad |X.Y|_{m_i} = ||X|_{m_i} . |Y|_{m_i}|_{m_i}.$$

Consequently, arithmetic operations can be performed separately for each modulus, and hence fast parallel arithmetic units based on RNS's can be implemented.

It is easy to show that results obtained in this paper also hold for digital systems in which signed arithmetic is used. In fact, assuming M to be even for the sake of simplicity, an implicit sign representation can be taken, by associating to any number X, $-2^{n-1} \leqslant X \leqslant 2^{n-1}-1$, a number N in the range [0,M) defined by the relation:

$$\begin{cases} N = X, & \text{if } X \geqslant 0 \\ N = M-|X|, & \text{if } X < 0. \end{cases}$$

In this way, signed arithmetic operations can be carried out in the residue form without the necessity for sign knowledge, which is difficult to obtain rapidly in RNS's. Moreover, after the RNS-to-positional conversion, the resulting non-negative number N, $0 \leqslant N < M$, carries an implicit representation of the sign of the actual result X, which can be obtained in its range [-M/2, M/2) as follows:

$$\begin{cases} X = N, & \text{if } N < M/2 \\ X = N-M, & \text{if } N \geqslant M/2 \end{cases}$$

III. Modulo m_i multiplication

Let X, Y be two integers such that $0 \leqslant X,Y < M$ and let us denote their corresponding RNS representations by $\{\alpha_1, \alpha_2, \ldots, \alpha_s\}$ and $\{\beta_1, \beta_2, \ldots, \beta_s\}$, respectively. Let us also suppose M to be sufficiently large to permit the representation of P = X.Y. To obtain the RNS representation of product P, products $\pi_i = |\alpha_i . \beta_i|_{m_i}$, i=1,s, must be evaluated. Of course, α_i and β_i are represented by means of $b_i = \lceil \log m_i \rceil$ binary digits; consequently, $2b_i$ digits are needed to represent each product $p_i = \alpha_i . \beta_i$.

Since

$$\pi_i = p_i - k_i . m_i \ , \qquad (1)$$

where $k_i = \lfloor p_i / m_i \rfloor$, the problem of performing modulo m_i multiplication is reduced to evaluating constant k_i and carrying out the operations required by relation (1). In our approach, the integer constant k_i is replaced by an integer $\bar{k}_i$ which can be obtained in a more straightforward way and which can assume two values: k_i or $k_i - 1$.

Let us approximate the fractional value $1/m_i$ using a number obtained by truncating its representation at the first r_i fractional bits; let t_i be such a number. Then

$$t_i \leqslant 1/m_i < t_i + 2^{-r_i} \qquad (2)$$

Multiplying inequalities (2) by p_i, we obtain:

$$p_i t_i \leqslant p_i / m_i < p_i t_i + p_i 2^{-r_i}$$

As a consequence, the maximum error E_i generated by approximating p_i / m_i by $p_i t_i$ is less than $p_i 2^{-r_i}$. If error E_i is to be below one, a number r_i must be chosen which satisfies the relation

$$r_i \geqslant 2b_i \ .$$

In fact, under this assumption and considering that $p_i < 2^{2b_i}$, we have

$$E_i = p_i 2^{-r_i} < 2^{2b_i} . 2^{-r_i} \leqslant 1 \ .$$

Denoting the value $\lfloor p_i t_i \rfloor$ by $\bar{k}_i$, we can write

$$k_i = \lfloor p_i / m_i \rfloor = \lfloor p_i t_i + E_i \rfloor = \lfloor \bar{k}_i + \mathrm{Fract}(p_i t_i) + E_i \rfloor$$
$$= \bar{k}_i + \lfloor \mathrm{Fract}(p_i t_i) + E_i \rfloor = \bar{k}_i + E_i' \ ,$$

where Fract(z) denotes the fractional part of the real number z. As $0 \leqslant \mathrm{Fract}(p_i t_i) + E_i < 2$, quantity E_i' belongs to

{0,1}.

Finally, product π_i can be obtained from eq.(1) as

$$\pi_i = p_i - \bar{k}_i m_i - E_i^? m_i \qquad (3)$$

Considered as an algorithm, the evaluation of π_i corresponds to computing expression $p_i - \bar{k}_i m_i$ and testing the result: if it is less than m_i, it will be correct; otherwise, a subtraction of m_i must be carried out to obtain π_i.

IV. An example

Consider the RNS based on moduli $m_1=7$, $m_2=11$, $m_3=13$. The range of the one-to-one representation of any positive integer X is $[0,\ 7\times11\times13 = 1001)$.

Let X=37, Y=12 be two numbers to be multiplied; in decimal notation their RNS expressions are:

$$37 = \{2,4,11\} \quad \text{and} \quad 12 = \{5,1,12\}$$

Conversely, the base 2 representations of the residue digits of X and Y are:

$37 = \{010,0100,1011\}$ and $12 = \{101,0001,1100\}$

Each residue digit has been expressed in a field of b_i bits, i.e. of the length required by the corresponding modulus value, i.e. $b_1 = 3$, $b_2 = 4$ and $b_3 = 4$. Multiplying the proper residue digits we obtain

$$\begin{aligned} p_1 &= \alpha_1\beta_1 = 010\times101 = 001010 \\ p_2 &= \alpha_2\beta_2 = 0100\times0001 = 00000100 \\ p_3 &= \alpha_3\beta_3 = 1011\times1100 = 10000100 \end{aligned}$$

Choosing $r_1 = 6$, $r_2 = 8$, $r_3 = 8$, the values of fractions $1/m_i$, truncated over r_i digits, can be taken as constants t_i, according to equation (2), so that:

$$\begin{aligned} t_1 &= .001001 \\ t_2 &= .00010111 \\ t_3 &= .00010011 \end{aligned}$$

Consequently, values $\bar{k}_i$ are:

$$\begin{aligned} \bar{k}_1 &= \lfloor p_1 t_1 \rfloor = \lfloor 001010\times.001001 \rfloor = 0001 \\ \bar{k}_2 &= \lfloor p_2 t_2 \rfloor = \lfloor 00000100\times.00010111 \rfloor = 00000 \\ \bar{k}_3 &= \lfloor p_3 t_3 \rfloor = \lfloor 10000100\times.00010011 \rfloor = 01001 \end{aligned}$$

and then:

$$p_1 - \bar{k}_1 m_1 = 001010-(0001\times111) = 001010-0000111 = 011 ;$$
$$p_2 - \bar{k}_2 m_2 = 00000100-000000000 = 0100 ;$$

since these results are less than $m_1 = 7$ and $m_2 = 11$, they coincide with π_1 and π_2, respectively.

For π_3, we have:

$$p_3 - \bar{k}_3 m_3 = 10000100-(01001\times1101) = $$
$$= 10000100-001110101 = 1111 ;$$

as this result is greater than $m_3 = 1101$, in this case a further subtraction of m_3 is needed to obtain the correct value of π_3 ($E'_3 = 1$):

$$\pi_3 = 1111-1101 = 0010.$$

The triple {3,4,2} so evaluated is the correct RNS representation of the product XY = 37x12 = 444.

It is worthwhile observing that the values resulting from the subtraction $p_i - \bar{k}_i m_i$ must always be positive and below $2m_i$, i.e. b_i+1 bits are sufficient for their representation. Consequently, only the b_i+1 less significant bits of p_i and of $-\bar{k}_i m_i$ can be considered to evaluate π_i.

V. **Description and evaluation of a modulo m_i multiplier architecture**

The algorithm for the modulo m_i multiplication presented in Section III can be implemented using the structure shown in Fig. 1.

At first, residue digits α_i and β_i, represented by b_i bits, are multiplied and value p_i is obtained at the $2b_i$-bit long output of MULTIPLIER1; p_i is then multiplied by t_i which is the approximated value of $1/m_i$, and the result is a product number R_i of $4b_i$ bits. Bits $\{2b_i, \dots, 3b_i\}$ of R_i contain the total representation of $\bar{k}_i$, which can be finally multiplied by $-m_i$ to obtain the correction term C_i, $2b_i$ -bit long. ADDER1 generates the difference $p_i - \bar{k}_i m_i$, which requires b_i bits for its representation. Depending on the sign bit value of the difference $D_i - m_i$, performed by ADDER2, the value D_i or $D_i - m_i$ is selected as the correct value of π_i.

It should be recalled that, as D_i differs from the correct result either by zero or by m_i, b_i+1 bits are sufficient for its representation and consequently bits $\{b_i+1, \dots, 2b_i-1\}$ of input data of ADDER1 can be ignored.

To evaluate the design performance in terms of response time, let us refer to a reasonable choice of an RNS to represent integers in a range such as $0 \leqslant X \leqslant 10^{12}$. For example, the RNS based on the set of moduli {255,254,253,251,247} may be an acceptable choice. In this case, $b_i=8$, $i=1,\ldots,5$, and three parallel multipliers 8x8, 16x16, and 8x8, as MULTIPLIER1, MULTIPLIER2 and MULTIPLIER3 respectively, are required. Total response times of about 150 nseconds can be achieved using commercially available multipliers with a multiplication time below 50 nseconds (e.g., the recent Weitek 16x16 integer multiplier WTL 2516C has a response time of 38 nseconds). In addition to the three multipliers, other LSI units are needed, and they can be accomodated in a single custom chip. In fact, the total number of connection lines, excluding the supply and control wires, is 50, which is compatible with current dual-in-line packages. On the other hand, the whole structure could also be integrated on a single VLSI chip.

The proposed structure can be easily adapted for pipeline operations, simply adding buffer registers at the output lines of each multiplier. In this way a pipeline interval in the range of 70-100 nseconds can be achieved.

Alternatively, modulo m_i multiplication can be implemented exploiting table look-up techniques. Referring to the previous example, the 16 bits of α_i and β_i form the address of a 64-kword ROM, each word containing a possible 8-bit result for π_i. Using commercial devices, the total response time can be brought close to 150 nseconds, comparable with the total response time determined above, and only one chip is needed to implement the multiplier .

It is easy to see that, when RNS's with moduli smaller than 2^8 are chosen, the ROM based solution is favoured, both with respect to the total response time and the number of chips needed. It should be remembered, in fact, that the total number of moduli increases as the mean modulus value decreases in order to keep M as large as necessary. Conversely, as the RNS moduli magnitude exceeds 2^{10} , the memory dimensions become so large that table look-up techniques are impractical, whereas the advantages of the structure in Fig. 1 are maintained. Indeed, doubling b_i in the above example, which would require a memory as large as 4096 M-words, 16-bit long, simply means substituting MULTIPLIER2 by a structure consisting of two adders and four multipliers equal to the one replaced, according to the logic organization shown in Fig. 2. The four multipliers can work in parallel and thus either the total response time or the pipeline interval will only be increased by two addition times.

In [12] a VLSI residue arithmetic multiplier, which is suitable for residue systems with large moduli values, and which also exploits commercial multipliers, is described. This multiplier consists of four pipeline stages, and, as in our structure in Fig. 1, the pipeline interval is mainly determined by the multiplication time. The number of circuit elements is also either of approximately the same order as ours, or slightly higher. The difference between the VLSI multiplier presented in [12] and our proposal is that the former was conceived for RNS's based on moduli $2^n-1, 2^n, 2^n+1$, which were chosen to ensure efficiency in scaling operations, whereas the latter has no design constraints on the number and values of the RNS moduli. Of course, unless these same moduli as in [12] are used in our design, the methods suggested for a generic RNS must be used for overflow detection and scaling, when this is required by the data dynamic range [13].

To complete the evaluation of the proposed architecture, we now intend to consider its possible implementation in large VLSI residue arithmetic units, by computing its order of complexity in terms of silicon area and execution time.

We refer to a VLSI model of computation which is by now generally accepted [14,15,16] and is based on the following assumptions:

a) wires have minimal width $\lambda = \vartheta(\text{const})$;

b) one bit of stored information requires area $\vartheta(\lambda^2)$;

c) the distance between parallel wires is $\vartheta(\lambda)$;

d) only two wires may cross at any point;

e) wires run horizontally and vertically;

f) any transistor needs a minimal transit time $\tau = \vartheta(\text{const})$ to change its state;

g) to propagate a binary change along a wire takes $\vartheta(\tau)$; to meet this assumption, wires of length $\vartheta(L)$ can be driven by drivers with area $A=\vartheta(\lambda)\times\vartheta(L)$.

Furthermore, we assume that

$$m_i = \vartheta(m), \qquad 1 \leqslant i \leqslant s$$

i.e., that all moduli chosen are of the same order of magnitude. Recalling that

$$2^n \leqslant \prod_{i=1}^{s} m_i < 2^{n+1}$$

it follows that, for large n,

$$2^n = \vartheta(m^s), \qquad n \approx s \log m.$$

The latter relation shows that the sum of the field lengths of the weighted representations of α_i, for large n, is about equal to the length of the positional representation of X. Moreover, this relation states a functional dependence among n, s, and m, and thus allows a complexity evaluation in terms of the two parameters n and s only.

To evaluate the total area required to implement the structure of Fig. 1, it is convenient to refer to Fig. 3, where the dimensions of each element are shown as functions of the size of m. As a VLSI integer multiplier, the network described in [17] has been chosen, which is (area).(time)^2 - optimal within the range of computation times $T_M = [\vartheta(\log n), \vartheta(\sqrt{n})]$ and exhibits an area $\vartheta((n/T_M)\times(n/T_M))$, where n is the length of the operands. In this network, each operand is subdivided into T_M strings of n/T_M bits which are viewed as binary numbers sequentially fed to the inputs. In our case, the multiplier layout needs an area $A_M = \vartheta((\log m/T_M)\times(\log m/T_M))$ and T_M ranges in $[\vartheta(\log\log m), \vartheta(\sqrt{\log m})]$.

Adders can be implemented by means of the structure proposed in [18], which, in our case, has an area $A_A = \vartheta((\log m/T_M)\times\log(\log m/T_M))$ and a time $T_A = \vartheta(T_M+\log(\log m/T_M))$. Considering the allowed range of T_M, T_A reduces to $\vartheta(T_M)$.

Of course, the complexity of MULTIPLEXER, in terms of area or of time, can be ignored.

Because data are processed in strings, the structure of Fig. 1 was slightly modified: delays were inserted in paths which connect not adjacent successive devices, in order to operate everywhere with relatively consistent strings.

Finally, delays were also inserted at both inputs of multiplexer, because the decision about what input is correct depends on the sign bit, which will be ready with the last string.

Observing Fig. 3, two remarks are relevant: first, adders are as wide as multipliers, but their depth is always less. Consequently, they can also be ignored in evaluating the overall complexity. The second remark is that the whole connection area has the same complexity figure in both dimensions as the multipliers in Fig. 3, because input data to adders must be permuted in order to satisfy constraints imposed by the chosen VLSI adder [18].

As a conclusion, the modulo m_i multiplier described has the same complexity figures as the integer multiplier in [17] and exhibits the same behaviour: in fact it requires input data and yields output data subdivided in T_M strings; input strings are sequentially processed.

We now intend to relate the previously obtained complexity figures to the number n of bits necessary to represent integers in the overall range [0,M) through the RNS parameters s and m, in order to evaluate the complexity of the overall VLSI RNS multiplier, which consists of s modulo m_i multipliers. It is worthwhile considering that the choice of s and m is essential for the performance of any computing system based on RNS arithmetic. Consequently, this choice cannot depend only on optimization constraints of the multiplier design. In Table I A(n) and T(n) are given for the overall VLSI RNS multiplier, according to several hypotheses on s and logm. Note that the hypothesis in the last entry of Table I must be considered as an asymptotic hypothesis because it is impossible to keep logm constant as n increases arbitrarily; in fact, for any constant k>0, the numbers below k form a finite set.

It can also be observed that the total width of the VLSI RNS multiplier has the same complexity figure for all hypotheses, i.e. $\vartheta(n/T_M)$; this is a consequence of the balancing between the string length for each module and the number of moduli. On the other hand, both RNS multiplier depth and execution time monotonically decrease with the average length logm of the moduli.

It is interesting to compare the complexity figures of our RNS multiplier with the simple RNS multiplication structure which can be obtained using a table look-up technique for each module. The whole input field must be considered in this case and one ROM of $\vartheta(m)$ words of $\vartheta(\log m)$ bits is necessary for each module. This ROM is addressed by the $\vartheta(\log m)$-bit operands. We can assume that the access time is $T = \vartheta(\log\log m)$. Table II gives complexity figures under the same hypotheses as in Table I. A comparison with Table I is straightforward. While choosing the lower extreme

TABLE I

s	$\log m$	$A(n,T_M)$	range of $T(n) = \vartheta(T_M)$
$\vartheta(\text{const})$	$\vartheta(n)$	$\vartheta((n/T_M)\times(n/T_M))$	$[\vartheta(\log n), \vartheta(\sqrt{n})]$
$\vartheta(\log\log n)$	$\vartheta(n/\log\log n)$	$\vartheta((n/T_M)\times(n/T_M \log\log n))$	$[\vartheta(\log n), \vartheta(\sqrt{n/\log\log n})]$
$\vartheta(\log n)$	$\vartheta(n/\log n)$	$\vartheta((n/T_M)\times(n/T_M \log n))$	$[\vartheta(\log n), \vartheta(\sqrt{n/\log n})]$
$\vartheta(n/\log n)$	$\vartheta(\log n)$	$\vartheta((n/T_M)\times(\log n/T_M))$	$[\vartheta(\log\log n), \vartheta(\sqrt{\log n})]$
$\vartheta(n/\log\log n)$	$\vartheta(\log\log n)$	$\vartheta((n/T_M)\times(\log\log n/T_M))$	$[\vartheta(\log\log\log n), \vartheta(\sqrt{\log\log n})]$
$\vartheta(n)$	$\vartheta(\text{const})$	$\vartheta((n/T_M)\times(1/T_M)) = \vartheta(n)$	$[\vartheta(\text{const}), \vartheta(\text{const})]$

TABLE II

s	log m	A(n)	T(n)
$\vartheta(\text{const})$	$\vartheta(n)$	$\vartheta(n\,2^{n})$	$\vartheta(\log n)$
$\vartheta(\log\log n)$	$\vartheta(n/\log\log n)$	$\vartheta(n2^{n/\log\log n})$	$\vartheta(\log n)$
$\vartheta(\log n)$	$\vartheta(n/\log n)$	$\vartheta(n\,2^{n/\log n})$	$\vartheta(\log n)$
$\vartheta(n/\log n)$	$\vartheta(\log n)$	$\vartheta(n^{2})$	$\vartheta(\log\log n)$
$\vartheta(n/\log\log n)$	$\vartheta(\log\log n)$	$\vartheta(n \log n)$	$\vartheta(\log\log\log n)$
$\vartheta(n)$	$\vartheta(\text{const})$	$\vartheta(n)$	$\vartheta(\text{const})$

complexity in the range of T(n) in Table I leads to the same time behaviour for both structures (one using multipliers and the other ROM's), the former structure exhibits an area complexity considerably lower than the latter for any value of T(n). However, the gap reduces as the number of moduli grows. The two structures require areas of the same complexity only in the last hypothesis, which, on the other hand, is not a practicable design for large values of n. This comparison evidences that the described RNS multiplier has better complexity figures than ROM-based RNS multipliers, even though table look-up techniques are generally considered the most suited to implement RNS's with a large number of small moduli. Note that the similar comparison previously carried out, which stated $m = 2^8$ as a bound below which ROM-based multipliers are more convenient, refers to structures designed under the last hypothesis in Tables I and II.

VI. Conversion from positional to residue number system representation and viceversa

Any VLSI implementation of RNS multiplication, as well as of other residue arithmetic operations, would be generally embedded in a larger VLSI system, devoted to particular applications, which exploits the characteristics of RNS's to achieve high levels of efficiency or reliability [9].

To interact with an environment in which data are represented by means of a binary positional system, direct and reverse conversions must be performed. A number of VLSI converters have been recently proposed in the literature [4,5,6]. The modulo m_i multiplication algorithm described in Section III suggests straightforward ways to perform the direct and the reverse conversion and to design their VLSI implementation. These implementations will be proved to be the fastest known residue converters.

In fact, let X be an integer such that $0 \leqslant X < M$; with assumptions similar to those in Section III, it is

$$X = k_i m_i + \alpha_i$$

and

$$\alpha_i = \breve{X} - \breve{k}_i m_i - E_i' m_i$$

where $\breve{k}_i = \lfloor X . t_i \rfloor$ and E_i' takes values 0 or 1 if $1/m_i$ is approximated by means of the sum of a proper subset of the first $n = \lceil \log M \rceil$ negative powers of two.

Since $\alpha_i = |\alpha_i|_{2^{b_i}}$, the relation

$$\alpha_i = ||X|_{2^{b_i}} - ||\bar{k}_i|_{2^{b_i}} \cdot m_i|_{2^{b_i}} - E_i' \cdot m_i|_{2^{b_i}}$$

holds; however, as the value of E_i' is unknown, a possible procedure to determine α_i consists in evaluating the expression

$$|X|_{2^{b_i+1}} - ||\bar{k}_i|_{2^{b_i+1}} \cdot m_i|_{2^{b_i+1}}$$

and in testing the result: if this is greater than or equal to m_i, m_i must be subtracted from it, otherwise it is the correct residue digit.

In Fig. 4 the VLSI structure to generate a residue digit is shown, together with the dimension of each element as a function of m_i and $n=\lceil \log M \rceil$. As in the case of Fig. 3, the largest element in the structure is the integer multiplier, which in this case occupies $\vartheta((n/T_M)x(n/T_M))$, where T_M ranges between $\vartheta(\log n)$ and $\vartheta(\sqrt{n})$. Since s structures are required to calculate all residue digits of X, the total conversion area is $A=\vartheta(sn^2/T_M^2)$, and the total time is T_M, see Table III.

This VLSI structure represents the best solution known for the positional to residue conversion problem, both in

TABLE III

s	$\log m$	$A(n,T_M)$	range of $T=\vartheta(T_M)$
$\vartheta(\mathrm{const})$	$\vartheta(n)$	$\vartheta(n^2/T_M^2)$	$[\vartheta(\log n), \vartheta(\sqrt{n})]$
$\vartheta(\log\log n)$	$\vartheta(n/\log\log n)$	$\vartheta(\log\log n\ n^2/T_M^2)$	$[\vartheta(\log n), \vartheta(\sqrt{n})]$
$\vartheta(\log n)$	$\vartheta(n/\log n)$	$\vartheta(\log n\ n^2/T_M^2)$	$[\vartheta(\log n), \vartheta(\sqrt{n})]$
$\vartheta(n/\log n)$	$\vartheta(\log n)$	$\vartheta(n^3/T_M^2 \log n)$	$[\vartheta(\log n), \vartheta(\sqrt{n})]$
$\vartheta(n/\log\log n)$	$\vartheta(\log\log n)$	$\vartheta(n^3/T_M^2 \log\log n)$	$[\vartheta(\log n), \vartheta(\sqrt{n})]$
$\vartheta(n)$	$\vartheta(\mathrm{const})$	$\vartheta(n^3/T_M^2)$	$[\vartheta(\log n), \vartheta(\sqrt{n})]$

terms of area complexity and in terms of time complexity: for the time aspect this assertion holds for any choice of s; for the area it is true under the hypothesis that the number s of moduli is in the range [ϑ(const), ϑ($\log^3 n$)). In fact, compare the area complexity expression A= $\vartheta(sn^2/\log^2 n)$ with the corresponding expression for the structure proposed in [5]:

$$A' = \vartheta(sn(\log(n/s))((n/s)+\log\log n)) .$$

It is easy to see that, when s ranges in [ϑ(const), ϑ($\log^3 n$)), A is a function of n with a lower order of complexity than A'. This comparison has been carried out for $T_M = \vartheta(\log n)$, but it can be verified that for T_M increasing towards $\sqrt{n}$ the area complexity evaluated for $s=\vartheta(\log^3 n)$ decreases. Consequently, the range of s for which the new converter will be effective extends beyond $\log^3 n$, possibly keeping also the time complexity low.

On the other hand, a simple residue to positional VLSI converter can be designed observing that [19]

$$X = \left| \sum_{i=1}^{s} p_i \alpha_i \right|_M$$

where p_i are constants which depend on the values of the selected moduli. In this case, s modulo M multiplications followed by s additions must be performed; s multipliers such as those proposed in Section 3 can be adopted for the former computations, whereas the latter would require modulo M adders. The additions can be carried out by means of a tree of binary adders arranged in logs levels followed by a structure which converts the resulting sum, expressed in a field of n+logs bits, to a modulo M integer. This structure is the same as that used to generate a single residue modulo m_i digit from any integer X, except for its size.

In Fig. 5 the global VLSI converter is depicted. Adders are designed as proposed in [18]; they have been also used in the multiplier layout. The area of the residue to positional converter in Fig. 5 is easily evaluated as the sum of three contributes, i.e. the areas of the s modulo M multipliers, the area devoted to the adder tree, which is mainly determined by the communication paths, and the area of the modulo (M+s)-to-modulo M converter:

$$A = \vartheta(s(n/T_H)^2 + s \log s(n/T_H)^2 + (n/T_H)^2) = \\ = \vartheta(s \log s\ (n/T_H)^2)$$

and the time is:

$$T = \vartheta(T_M + \log s \; \log(n/T_M) + T_M)$$

We recall that input residue digits are processed by subdividing them into T_M strings, each n/T_M -bit wide, and making the whole structure operate in pipeline. Hence, the way that the time complexity expression has been derived takes into account the contributes of the multiplication phase, the addition phase and the conversion phase, respectively (see Table IV).

In this case too, the proposed solution is the best solution known to solve the residue to positional conversion, either referring to the time complexity, or to the area complexity: for the time it is true for any choice of s except for the asymptotic hypothesis $s = \vartheta(n)$, $\log m = \vartheta(\text{const})$, when both exhibit the same time performance; as far as the area is concerned, the comparison between expression $A = \vartheta(s \log s(n^2/\log^2 n))$, which holds for $T_M = \vartheta(\log n)$, and $A' = \vartheta(n^2 \log n)$ [5], shows that, when s ranges in $[\vartheta(\text{cost}), \vartheta(\log n/\log\log n))$, A is a function of n with a lower order of complexity than A'. Moreover, considerations similar to those expressed for the direct conversion about the possible increasing of T_M towards $\sqrt{n}$ can be generalized to this case.

Finally, it can be inferred that the VLSI structures

TABLE IV

s	$\log m$	$A(n,T_M)$	$T(n,T_M)$	range of T_M
$\vartheta(\mathrm{const})$	$\vartheta(n)$	$\vartheta(n^2/T_M^2)$	$\vartheta(T_M)$	$[\log n, \sqrt{n}]$
$\vartheta(\log\log n)$	$\vartheta(n/\log\log n)$	$\vartheta(\log\log n \log\log\log n\ n^2/T_M^2)$	$\vartheta(T_M + \log\log\log n \log(n/T_M))$	$[\log n, \sqrt{n/\log\log n}]$
$\vartheta(\log n)$	$\vartheta(n/\log n)$	$\vartheta(\log n \log\log n\ n^2/T_M^2)$	$\vartheta(T_M + \log\log n \log(n/T_M))$	$[\log n, \sqrt{n/\log n}]$
$\vartheta(n/\log n)$	$\vartheta(\log n)$	$\vartheta(n^3/T_M^2)$	$\vartheta(T_M + \log(n/\log n)\log(n/T_M))$	$[\log\log n, \sqrt{\log n}]$
$\vartheta(n/\log\log n)$	$\vartheta(\log\log n)$	$\vartheta(\log n\ n^3/T_M^2 \log\log n)$	$\vartheta(T_M + \log(n/\log\log n)\log(n/T_M))$	$[\log\log\log n, \sqrt{\log\log n}]$
$\vartheta(n)$	$\vartheta(\mathrm{const})$	$\vartheta(\log n\ n^3/T_M^2)$	$\vartheta(T_M + \log n \log(n/T_M))$	$[\mathrm{const}, \mathrm{const}]$

proposed for direct and reverse conversion and for modulo m_i multiplication are very good designs for any permitted value of T_M, when $s=\vartheta(\text{const})$. In fact, an AT^2-optimal integer multiplier could be constructed by putting in a cascade a binary to residue converter, s modulo m multipliers and a residue to binary converter, all with the same value for T_M. The overall area is $A=\vartheta((n/T_M)\times(n/T_M))$ and the total time range is $[\log n, \sqrt{n}]$. This integer multiplier is an optimal design since it reaches the lower bound on complexity $AT^2=\Omega(n^2)$ [15].

VII. Conclusions

We have presented a method to compute modulo m_i multiplication, which requires only a few binary multiplications on data which are at most $2\lceil\log m_i\rceil$ -bit long.

Using this method, an arithmetic structure has been designed, tailored for a possible RNS choice allowing a single chip integration; this structure has been compared with solutions based on table look-up techniques and with Taylor's residue multiplier which uses the triple of moduli 2^n-1, 2^n, 2^n+1.

The proposed multiplier has also been evaluated according to the complexity theory of VLSI algorithms, in order to evidence its advantages, when large RNS's are used.

The method has also suggested new solutions to the problem of direct and reverse conversion from positional to residue representation of integers.The VLSI complexity of such solutions has been evaluated and they have been proved to be the best known, under a wide range of assumptions.

Acknoledgements

The authors wish to thank F. Barsi for the many helpful discussions during the preparation of the work.

References

[1] G. A. Jullien, "Residue number scaling and other operations using ROM arrays", IEEE Trans. Comput., Vol. C-27, pp. 325-336, Apr. 1978.

[2] C. H. Huang, D. G. Peterson, H. E. Rauch, J. W. Teague and D. F. Fraser, "Implementation of a fast digital processor using the residue number system", IEEE Trans. Circuits Syst., Vol. CAS-28, pp.32-38, Jan. 1981.

[3] M. A. Bayoumi, G. A. Jullien and W. C. Miller, "A VLSI model for residue number system architecture", INTEGRATION, the VLSI journal, Vol. 2, pp. 191-211, 1984.

[4] G. Alia, F. Barsi and E. Martinelli, "A fast VLSI conversion between binary and residue systems", Inform. Process. Lett., Vol. 18, pp. 141-145, Mar. 1984.

[5] G. Alia and E. Martinelli, "A VLSI algorithm for direct and reverse conversion from weighted binary number system to residue number system", IEEE Trans. Circuits Syst., Vol. CAS-31, pp. 1033-1039, Dec. 1984.

[6] C. D. Thompson, "VLSI design with multiple active layers", Inform. Process. Lett., Vol. 21, pp. 109-111, Sep. 1985.

[7] W. K. Jenkins and B. J. Leon, "The use of residue number systems in the design of finite impulse response

digital filters", IEEE Trans. Circuits Syst., Vol. CAS-24, pp. 191-201, Apr. 1977.

[8] M. A. Soderstrand, "A high-speed low cost recursive digital filter using residue number arithmetic", Proc. IEEE, Vol. 65, pp. 1065-1067, Jul. 1977.

[9] G. Alia, F. Barsi and E. Martinelli, "A fast near optimum VLSI implementation of FFT using residue number systems", INTEGRATION, the VLSI journal, Vol. 2, pp. 133-147, 1984.

[10] W. K. Jenkins, "Techniques for residue-to-analog conversion for residue-encoded digital filters", IEEE Trans Circuits Syst., Vol. CAS-25, pp. 555-562, Jul. 1978.

[11] W. K. Jenkins, "A highly efficient residue-combinatorial architecture for digital filters", Proc. IEEE, Vol. 66, pp. 700-702, Jun. 1978.

[12] F. J. Taylor, "A VLSI residue arithmetic multiplier", IEEE Trans. Comput., Vol. C-31, pp. 540-546, Jun. 1982.

[13] N. S. Szabo and R. I. Tanaka, Residue arithmetic and its applications to computer technology. New York: McGraw-Hill, 1967.

[14] C. D. Thompson, "A complexity theory for VLSI", Ph.D. Thesis, Carnegie-Mellon University, Computer Science Dept., Aug. 1980.

[15] R. P. Brent and H. T. Kung, "The area-time complexity of binary multiplication", JACM, Vol. 28, pp. 521-534, Jul. 1981.

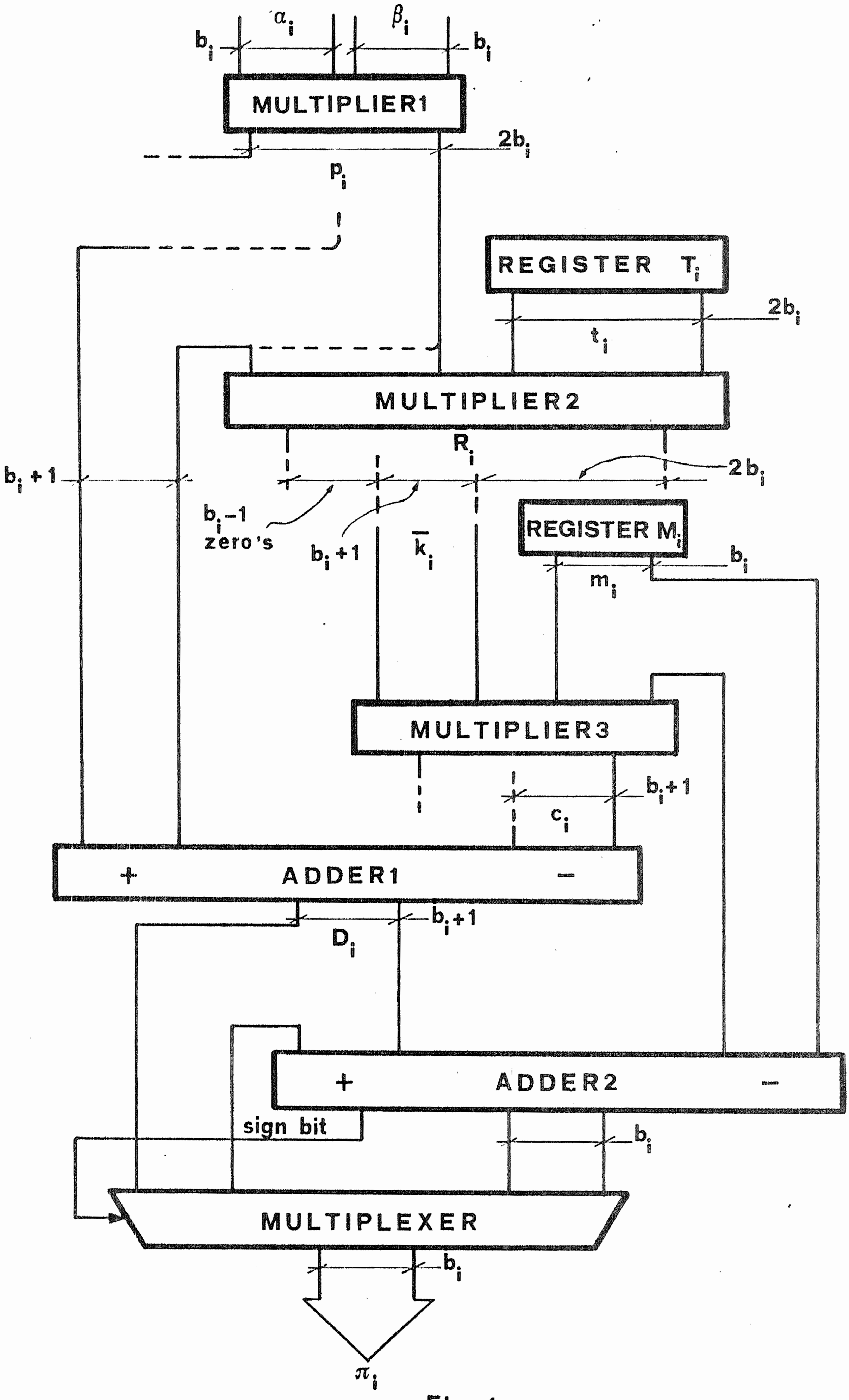
[16] C. A. Mead and L. A. Conway, Introduction to VLSI Systems. Reading, Mass.: Addison-Wesley, 1980.

[17] K. Mehlhorn and F. P. Preparata, "Area-time optimal VLSI Integer Multiplier with minimum computation time", Information and Control, Vol. 58, pp. 137-156, 1983.

[18] R. P. Brent and H. T. Kung, "A regular layout for parallel adders", IEEE Trans. Comput., Vol. C-31, pp. 260-264, Mar. 1982.

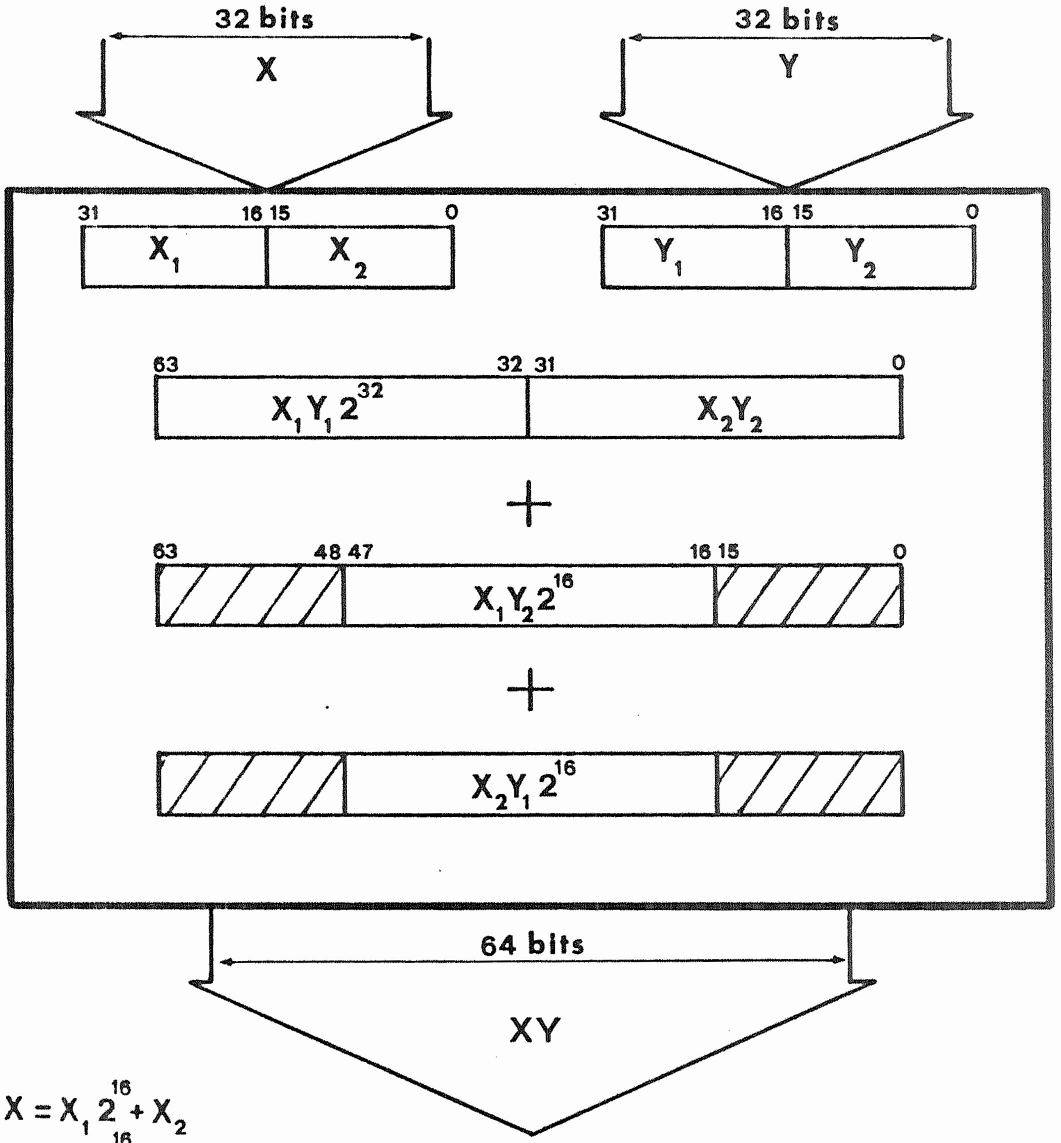
[19] A. V. Aho, J. E. Hopcroft and J. D. Ullman, The design and analysis of computer algorithms. Reading, Mass.: Addison-Wesley, 1974.

FIGURE CAPTIONS



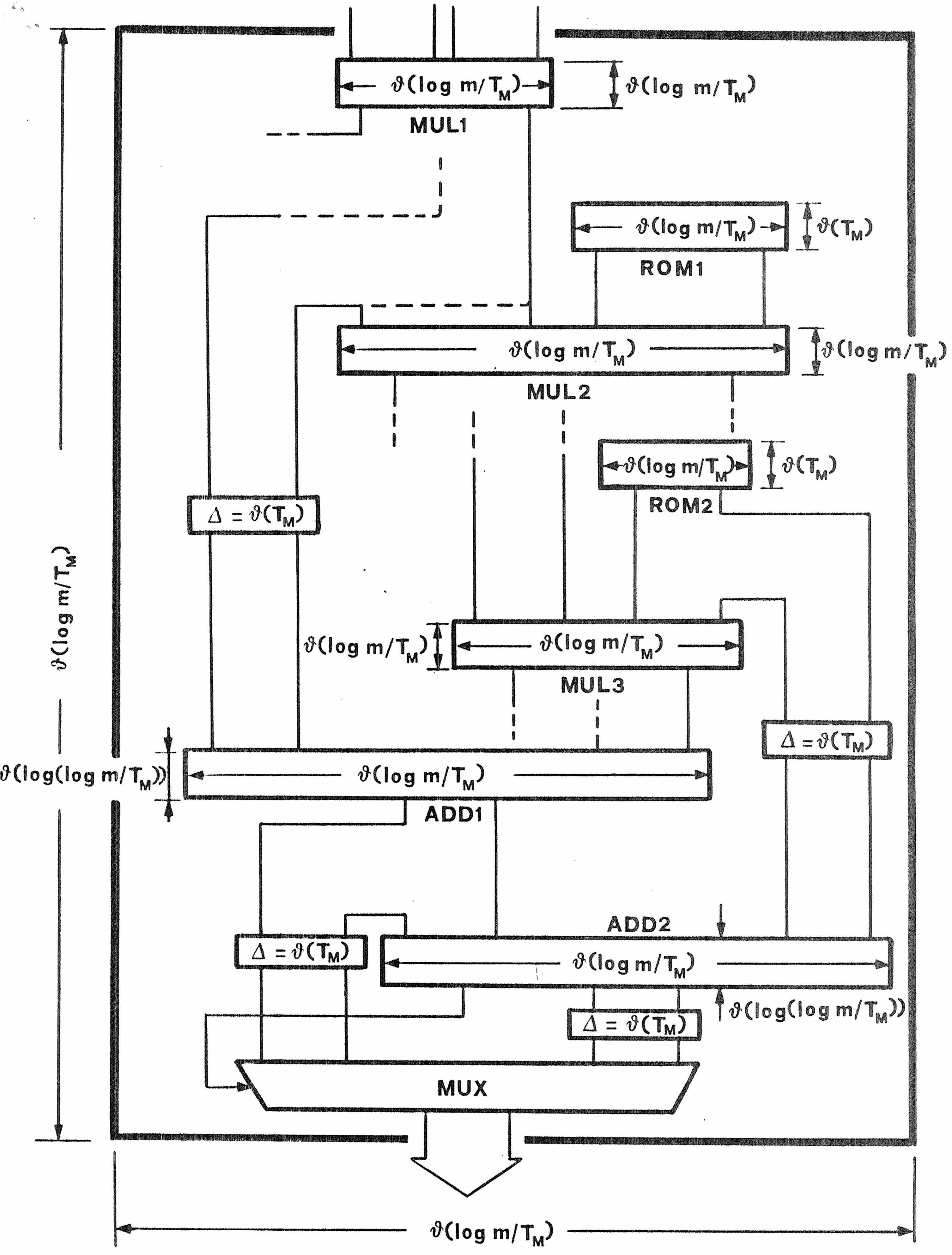
α_i
β_i
b_i
b_i
MULTIPLIER1
$2b_i$
p_i
REGISTER T_i
$2b_i$
t_i
MULTIPLIER2
R_i
b_i+1
$2b_i$
b_i-1 zero's
b_i+1
$\overline{k}_i$
REGISTER M_i
b_i
m_i
MULTIPLIER3
b_i+1
c_i
+
ADDER1
−
b_i+1
D_i
+
ADDER2
−
sign bit
b_i
MULTIPLEXER
b_i
π_i

Fig. 1



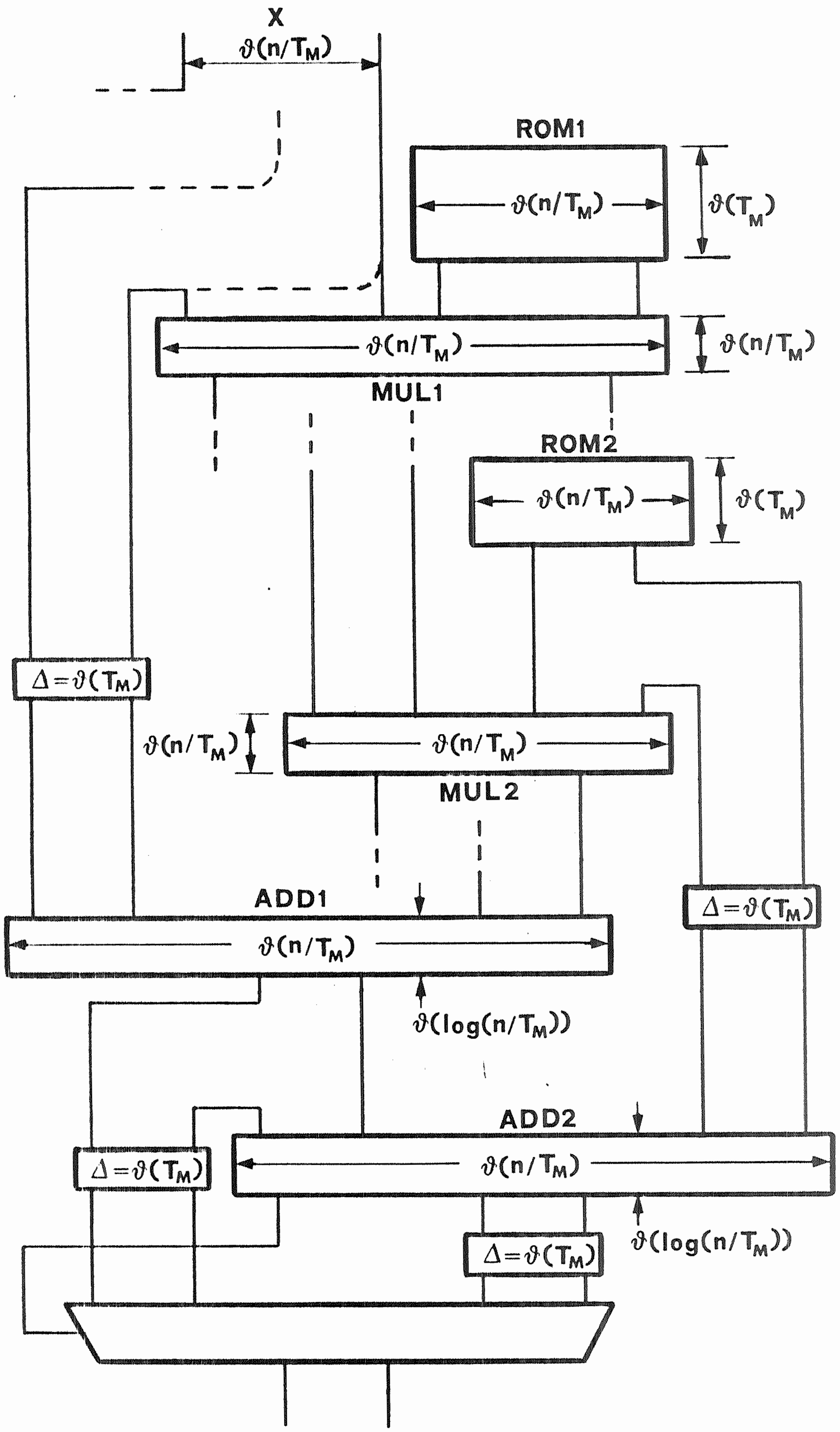
$X = X_1 2^{16} + X_2$
$Y = Y_1 2^{16} + Y_2$

Fig. 2



$\vartheta(\log m/T_M)$
MUL1
$\vartheta(\log m/T_M)$
$\vartheta(\log m/T_M)$
ROM1
$\vartheta(T_M)$
$\vartheta(\log m/T_M)$
MUL2
$\vartheta(\log m/T_M)$
$\vartheta(\log m/T_M)$
ROM2
$\vartheta(T_M)$
$\Delta = \vartheta(T_M)$
$\vartheta(\log m/T_M)$
$\vartheta(\log m/T_M)$
MUL3
$\vartheta(\log m/T_M)$
$\Delta = \vartheta(T_M)$
$\vartheta(\log(\log m/T_M))$
$\vartheta(\log m/T_M)$
ADD1
ADD2
$\Delta = \vartheta(T_M)$
$\vartheta(\log m/T_M)$
$\Delta = \vartheta(T_M)$
$\vartheta(\log(\log m/T_M))$
MUX
$\vartheta(\log m/T_M)$

Fig. 3



X
$\vartheta(n/T_M)$
ROM1
$\vartheta(n/T_M)$
$\vartheta(T_M)$
$\vartheta(n/T_M)$
$\vartheta(n/T_M)$
MUL1
ROM2
$\vartheta(n/T_M)$
$\vartheta(T_M)$
$\Delta=\vartheta(T_M)$
$\vartheta(n/T_M)$
$\vartheta(n/T_M)$
MUL2
ADD1
$\vartheta(n/T_M)$
$\Delta=\vartheta(T_M)$
$\vartheta(\log(n/T_M))$
ADD2
$\Delta=\vartheta(T_M)$
$\vartheta(n/T_M)$
$\Delta=\vartheta(T_M)$
$\vartheta(\log(n/T_M))$

Fig. 4

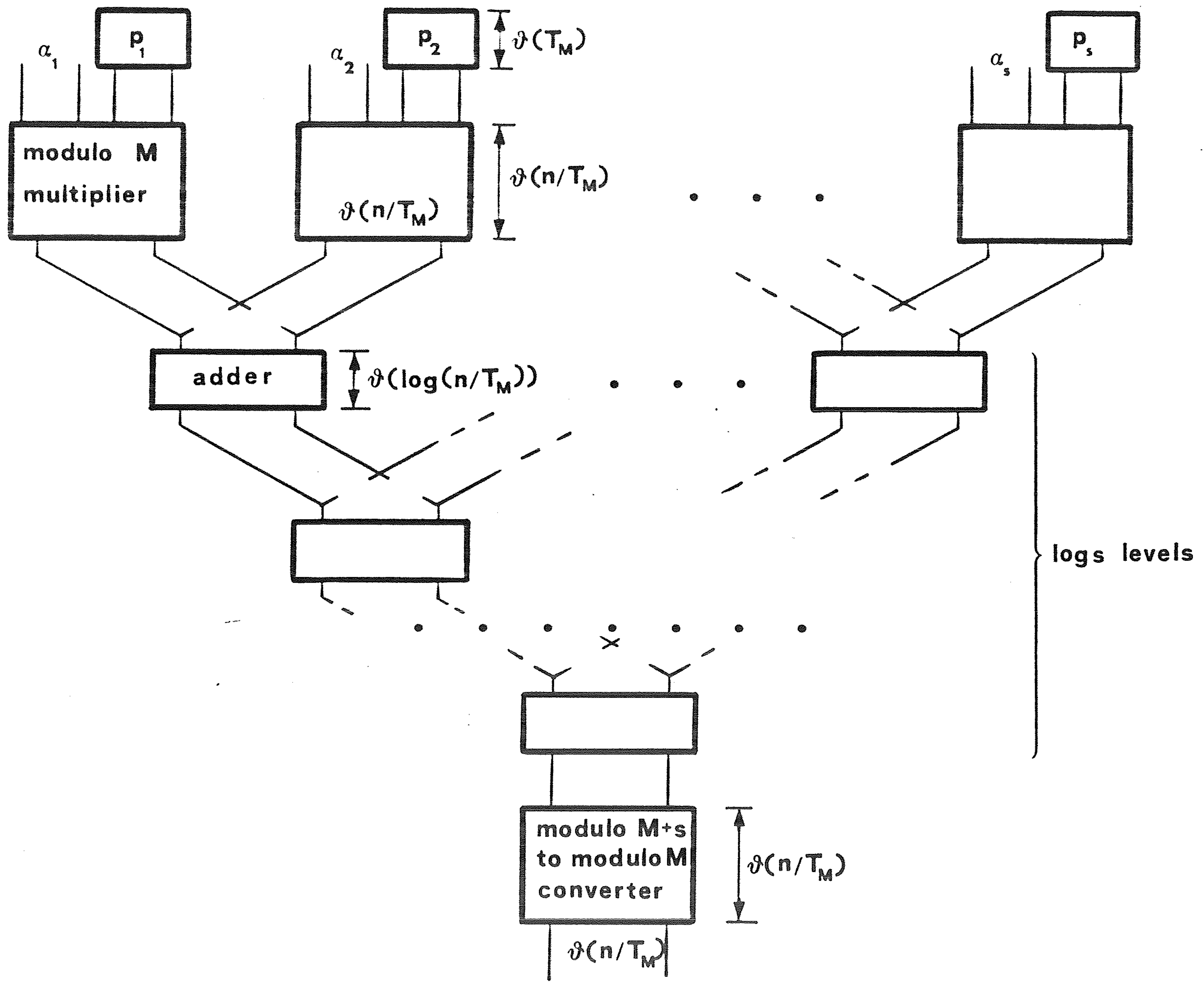

p_1
a_1
modulo M multiplier
p_2
a_2
$\vartheta(T_M)$
$\vartheta(n/T_M)$
$\vartheta(n/T_M)$
p_s
a_s
adder
$\vartheta(\log(n/T_M))$
logs levels
modulo M+s to modulo M converter
$\vartheta(n/T_M)$
$\vartheta(n/T_M)$

Fig. 5