Time Optimal Mixed Radix Conversion for Residue Number Applications

FERRUCCIO BARSI* AND M. CRISTINA PINOTTI†

*Dipartimento di Matematica, Universitá di Perugia, Via Vanvitelli 1, 06100 Perugia, Italy
†Istituto di Elaborazione dell'Informazione del CNR, Via S. Maria 46, 56100 Pisa, Italy

A new method is proposed for converting residue integers into a mixed radix notation. The method is based upon a modified formulation of the Chinese Remainder Theorem, and permits both conventional logic and look-up table implementations. Moreover, it represents the first method enabling optimal, residue-to-weighted system, asymptotic conversion time. To prove this, a constructive VLSI design has been devised, exhibiting time $O(\log s)$, where s is the total number of input bits. If compared with the existing mixed radix converting techniques, the method proposed considerably enhances the conversion time. To conclude, it is shown that, at the present state of the technology, practical ECL IC's implementations achieve 35–40 ns conversion times with RAMs and 60–70 ns with logic circuitry for dynamic ranges up to 300 bits.

Received March 15, 1994, revised October 12, 1994

1. INTRODUCTION

Residue Number Systems (RNSs) represent an integer X by means of a set of independent digits (the residues of X modulo a set of pairwise prime integers, *the moduli of the system*). The modular nature of the residue representation and the independence of the digits permit fast addition and multiplication but prevent any knowledge of the number magnitude. This is the major drawback of RNSs as the advantages of their natural ability to support fast and parallel arithmetics are greatly reduced whenever operations requiring a conversion to a weighted representation, such as sign determination, magnitude comparison, base extension or division, are involved. Therefore, due to the increasing interest in residue computing for Digital Signal Processing applications and for high speed arithmetics, a great research work has been spent to speed up conversion methods.

There are two basic approaches for converting integers from an RNS. The first one is based upon the Chinese Remainder Theorem (CRT) and requires, in its standard formulation, $(n-1)$ full range additions. An alternative approach, i.e. the Mixed Radix Conversion method (MRC) (Szabo and Tanaka, 1967), maintains the modular nature of the computations but requires $(n-1)$ additive/multiplicative sequential steps to obtain the result.

So far, attention has been devoted towards time saving implementations of CRT achieving asymptotic conversion times $O(\log m + \log n \log s)$ (Vu, 1985), $O(\log^2 s)$ (Alia and Martinelli, 1984) and $O(\log s \log n)$ (Capocelli and Giancarlo, 1988), where s is the number of input bits, and n and m represent the number and the average value of residue system moduli, respectively.

The Mixed Radix Conversion algorithm has also been reconsidered with the aim of obtaining, by means of table look-up techniques, weighted binary representations of integers through base extension (Shenoy and Kumaresan, 1988) or, more efficiently, the mixed radix digits in time $t_T + (n-1)(t_A + t_C)$ (Huang, 1983) and $t'_T + (n/2)t_A$ (Chakraborti *et al.*, 1986), where t_T and t'_T indicate the single digit or twin digit memory addressing time, respectively, t_A is the time required to perform a $\log m$ bits binary addition and t_C is the time taken by a comparator/subtractor logic. The solutions proposed offered modular designs and short execution times. However, because of the use of table look-up techniques, their application is strongly dependent on the state of the art of the technology and the time asymptotic performances are still far from the theoretical, time conversion lower bound $\Omega(\log s)$.

This paper starts with the approach presented in Huang (1983) and reconsiders the conversion to a mixed radix representation in order to obtain a method which is faster than the previous table look-up implementations and which also matches the time conversion asymptotic lower bound $\Omega(\log s)$. In addition, the method should be used for both table look-up and standard logic circuitry implementations.

The proposed method will be described in Section 2 and it will be shown that all conversion parameters have mathematical expressions and, consequently, their computation can be carried out with conventional logic. The mixed radix representation of a residue number X will be first obtained, by means of a fully modular (carry free), parallel computation in the form of a pair of mixed radix integers (R, Q) which may represent a sufficient result for certain applications. Adding R and Q will get the explicit mixed radix notation of X. In the same section an implementation of the new method based on table look-ups will be compared with Huang's and Chakraborti's results. As the proposed implementation has a conversion time $t_T + 2t_A + 3t_C + k \log n$, where k is a constant which depends on the time required to switch a gate, it will be concluded that the proposed method considerably speeds up the mixed radix conversion.

In Sections 3 and 4, a VLSI layout for an implementation of the new method using logic circuitry will be illustrated achieving an optimal conversion time $t_C = O(\log s)$.

Finally, in Section 5 both logic circuitry and look-up table implementations using current ECL ICs will be proposed with 35–40 ns conversion time for RAM based and 60–70 ns for full logic based converters with dynamic ranges up to 312 bits.

2. THE PROPOSED METHOD

Let an RNS be defined by a set of n pairwise prime moduli $\{m_1, m_2, \ldots, m_{n-1}, m_n\}$ and assume, without loss of generality, that $m_i < m_j$ for $i < j$ and $i, j = 1, \ldots, n$. Any integer X in the range $[0, M)$, $M = \Pi_{i=1}^{n} m_i$ will be represented as:

$$X \equiv \{x_1, x_2, \ldots, x_{n-1}, x_n\} \text{ where } x_i = |X|_{m_i}, \quad i = 1, 2, \ldots, n \tag{1}$$

Conversely, starting from the residue representation (1), the value of X will be reconstructed by means of the CRT as:

$$X = \left| \sum_{i=1}^{n} M_i \left| \frac{x_i}{M_i} \right|_{m_i} \right|_M \tag{2}$$

where $M_i = M/m_i$ and $|1/M_i|_{m_i}$ is the multiplicative inverse of $M_i \bmod m_i$. As the ith term of expression (2):

$$X_i = M_i \left| \frac{x_i}{M_i} \right|_{m_i} \tag{3}$$

has the residue representation:

$$\{0, \ldots, x_i, \ldots, 0\}$$

the same expression (2) can be given the form:

$$X = \{x_1, 0, \ldots, 0\} + \cdots + \{0, \ldots, x_i, \ldots, 0\} + \cdots + \{0, \ldots, 0, x_n\} \bmod M \tag{2'}$$

Now, consider the mixed radix number system (MRS) associated with the given RNS. Any integer Y will be represented as:

$$Y = \sum_{j=1}^{n} w_j M^{(j)} \tag{4}$$

where:

$$w_j = \left| \left\lfloor \frac{Y}{M^{(j)}} \right\rfloor \right|_{m_j} \tag{5}$$

and

$$M^{(j)} = \frac{M}{\Pi_{k=j}^{n} m_k} = \Pi_{k=1}^{j-1} m_k \text{ for } j = 2, \ldots, n \text{ and } M^{(1)} = 1 \tag{6}$$

Assuming $Y = X_i$, it will be obtained:

$$X_i = \sum_{j=1}^{n} w_{i,j} M^{(j)} \tag{4'}$$

and, from equalities (3) and (5):

$$\begin{aligned} w_{i,j} &= \left| \left\lfloor \frac{\left| \frac{x_i}{M_i} \right|_{m_i} M_i}{M^{(j)}} \right\rfloor \right|_{m_j} = \left| \left\lfloor \frac{\left| \frac{x_i}{M_i} \right|_{m_i} \Pi_{k=j}^{n} m_k}{m_i} \right\rfloor \right|_{m_j} \\ &= \left| \frac{\left| \frac{x_i}{M_i} \right|_{m_i} \Pi_{k=j}^{n} m_k - \left| \left| \frac{x_i}{M_i} \right|_{m_i} \Pi_{k=j}^{n} m_k \right|_{m_i}}{m_i} \right|_{m_j} \end{aligned} \tag{7}$$

or, equivalently:

$$\begin{aligned} w_{i,j} &= 0 && \text{for } j < i \\ w_{i,j} &= \left| \left| \frac{x_i}{M_i} \right|_{m_i} \Pi_{k=i+1}^{n} m_k \right|_{m_i} = \left| \frac{x_i}{M^{(i)}} \right|_{m_i} && \text{for } j = i \\ w_{i,j} &= \left| - \left| \frac{1}{m_i} \right|_{m_j} \left| \left| \frac{x_i}{M_i} \right|_{m_i} \Pi_{k=j}^{n} m_k \right|_{m_i} \right|_{m_j} \\ &= \left| - \left| \frac{1}{m_i} \right|_{m_j} \left| \frac{x_i}{M^{(j)}/m_i} \right|_{m_i} \right|_{m_j} && \text{for } j > i \end{aligned} \tag{7'}$$

As shown in Appendix A, substituting for $w_{i,j}$ in (4′) and then for X_i in equality (2), the CRT takes the form:

$$X = \left| \sum_{j=2}^{n} M^{(j)} \left\lfloor \frac{\sum_{i=1}^{j-1} w_{i,j-1}}{m_{j-1}} \right\rfloor + \sum_{j=1}^{n} M^{(j)} \left| \sum_{i=1}^{j} w_{i,j} \right|_{m_j} \right|_M \quad \text{with } w_{i,0} = 0 \tag{8}$$

Letting:

$$Q = \sum_{j=2}^{n} M^{(j)} \left\lfloor \frac{\sum_{i=1}^{j-1} w_{i,j-1}}{m_{j-1}} \right\rfloor = \sum_{j=2}^{n} q_{j-1} M^{(j)}$$

$$R = \sum_{j=1}^{n} M^{(j)} \left| \sum_{i=1}^{j} w_{i,j} \right|_{m_j} = \sum_{j=1}^{n} r_j M^{(j)}$$

with

$$\begin{aligned} q_{j-1} &= \left\lfloor \frac{\sum_{i=1}^{j-1} w_{i,j-1}}{m_{j-1}} \right\rfloor \leqslant \left\lfloor \frac{(j-1)(m_{j-1}-1)}{m_{j-1}} \right\rfloor \\ &= (j-1) - 1 < m_j \quad \text{for } 2 \leqslant j \leqslant n \\ r_j &= \left| \sum_{i=1}^{j} w_{i,j} \right|_{m_j} < m_j \quad \text{for } 1 \leqslant j \leqslant n \end{aligned} \tag{7''}$$

the above formulation (8) indicates that X is obtained as the mod M sum of two mixed radix integers, namely:

$$X = |Q + R|_M \tag{8'}$$

where the mod M operator is trivially applied by discarding carries coming from the most significant digits.

As an example, consider the RNS of moduli $m_1 = 25$, $m_2 = 27$, $m_3 = 29$, $m_4 = 31$, $m_5 = 32$, and let $X \equiv \{0,7,21,12,13\}$ be a residue representation to be converted in the associated mixed radix system.

Computing parameters $w_{i,j}(i,j = 1,\dots,5)$ from (7′), it is obtained:

$$
\begin{array}{llll}
X_1 \equiv \{0,0,0,0,0\} & w_{1,1} = 0 & w_{1,2} = 0 & w_{1,3} = 0 \\
 & w_{1,4} = 0 & w_{1,5} = 0 & \\
X_2 \equiv \{0,7,0,0,0\} & w_{2,1} = 0 & w_{2,2} = 10 & w_{2,3} = 5 \\
 & w_{2,4} = 9 & w_{2,5} = 8 & \\
X_3 = \{0,0,21,0,0\} & w_{3,1} = 0 & w_{3,2} = 0 & w_{3,3} = 28 \\
 & w_{3,4} = 14 & w_{3,5} = 26 & \\
X_4 \equiv \{0,0,0,12,0\} & w_{4,1} = 0 & w_{4,2} = 0 & w_{4,3} = 0 \\
 & w_{4,4} = 23 & w_{4,5} = 23 & \\
X_5 \equiv \{0,0,0,0,13\} & w_{5,1} = 0 & w_{5,2} = 0 & w_{5,3} = 0 \\
 & w_{5,4} = 0 & w_{5,5} = 5 &
\end{array}
$$

and mixed radix digits (7″) are derived from summations $\sum_{i=1}^{j} w_{i,j}, j = 1,\dots,5$:

$$\sum_{i=1}^{1} w_{i,1} = w_{1,1} = 0 \rightarrow q_1 = 0 \quad r_1 = 0$$

$$\sum_{i=1}^{2} w_{i,2} = w_{1,2} + w_{2,2} = 10 \rightarrow q_2 = 0 \quad r_2 = 10$$

$$\sum_{i=1}^{3} w_{i,3} = w_{1,3} + w_{2,3} + w_{3,3} = 33 \rightarrow q_3 = 1 \quad r_3 = 4$$

$$\sum_{i=1}^{4} w_{i,4} = w_{1,4} + w_{2,4} + w_{3,4} + w_{4,4}$$
$$= 46 \rightarrow q_4 = 1 \quad r_4 = 15$$

$$\sum_{i=1}^{5} w_{i,5} = w_{1,5} + w_{2,5} + w_{3,5} + w_{4,5} + w_{5,5}$$
$$= 62 \rightarrow q_5 = 1 \quad r_5 = 30$$

i.e.

$$Q = 0M^{(1)} + 0M^{(2)} + 0M^{(3)} + 1M^{(4)} + 1M^{(5)}$$

$$R = 0M^{(1)} + 10M^{(2)} + 4M^{(3)} + 15M^{(4)} + 30M^{(5)}$$

Mixed radix conversion will be completed by adding Q and R:

$$X = Q + R = 0M^{(1)} + 10M^{(2)} + 4M^{(3)} + 16M^{(4)} + 31M^{(5)}$$

As a comparison with the previous work, let us refer to the following Figure 1, where an architecture for the above example of five system moduli is reported. In Figure 1, the blocks $w_{i,j}(i,j = 1,\dots,5)$ correspond to the circuitry which is necessary to obtain the starting parameters. As, to be fair, a comparison should consider implementations based on similar circuitry, it will be assumed here that these blocks are look-up tables addressed by a single residue digit with an access time t_T.

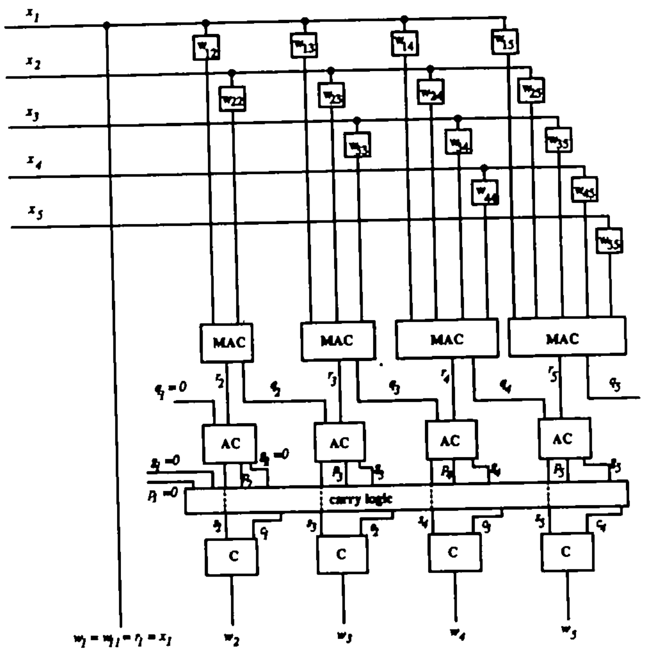


FIGURE 1

The MAC (Multioperand Adder/Comparator) blocks consist of a multioperand adder followed by a comparator/subtractor. A multioperand adder can be effectively implemented by means of a carry save adder tree (Hwang, 1979) and a conventional adder. Thus, the time required to perform a multiple addition of n items is:

$$t_{CSA} \log n + t_A$$

where t_{CSA} is the time required to perform a carry save addition and n is, in the general case, the number of system moduli. It is noteworthy that t_{CSA} is proportional to the time required to switch a gate.

The mixed radix digits r_i, q_i are then obtained by using a comparator/subtractor (with time t_C) for a total MAC time:

$$t_{CSA} \log n + t_A + t_C$$

Once the mixed radix digits of R and Q are computed, the MR expression of X is derived from (8′) at a low cost by using a carry look ahead approach with time proportional to the logarithm of the number of the inputs. The carry logic inputs are generated by an adder/comparator and the outputs enter a final comparator yielding the mixed radix digits of X. The total time taken by these last operators is:

$$t_A + t_C + h \log n + t_C$$

where h is a constant indicating the gate switching time.

It can be concluded that the conversion time is:

$$T_{\text{Conv}} = t_T + 2t_A + 3t_C + k \log n$$

where $k = t_{CSA} + h$ is a constant.

A comparison with the existing methods is abstracted in Table 1 where $b = \log m$ is the number of residue digit bits. It can be concluded that the present approach considerably enhances the time performances

TABLE 1

Reference	*Tables*	*Adders*	*Comp/subtractors*	*Conversion time*
Huang (1983)	$n(n+1)/2^*$	$n(n+1)/2$	$n-1$	$t_T+(n-1)(t_A+t_C)$
Chakraborti *et al.* (1986)	$n(n-2)/4+(n-1)^{\dagger}$	$n/2(n/2+1)-3$	$n-2$	$t'_T+n/2t_A$
New method	$n(n+1)/2^*$	$(n-1)(n+2)/2+2(n-1)$	$3(n-1)$	$t_T+2t_A+3t_C+k\log n$

* $2^b \times b$ bits; † $2^{2b} \times b$ bits. t_A, addition time; t_T, single digit addressing time; t'_T, twin digit addressing time; t_C, comparison/subtraction time.

(the number of moduli appears as a logarithm) whereas the hardware requirements are correspondingly increased up to $(n-1)(n+2)/2+2(n-1)$ adders (including CSA units) and $3(n-1)$ comparators.

3. A VLSI LAYOUT FOR MRC

The proposed VLSI layout derives from the computational scheme of Figure 1 and consists of n columns of logic elements performing additions and multiplications: the jth column is related to modulus, $m_j, j=1,\ldots,n$.

A pipelined scheme of computation will be adopted according to Brent and Kung's adder design (Brent and Kung, 1982), AT^2—optimal Mehlhorn—Preparata's multiplier (Mehlhorn and Preparata 1983) and their combinations leading to modular adders and multipliers (Alia and Martinelli, 1991; Barsi, 1991). We recall here that in a pipelined scheme of computation input data are divided into a number of strings which are processed in sequence.

For any given computational element, let μ indicate the total number of input bits and τ be the number of input strings (i.e. the number of segments into which operands are partitioned). The width of input strings will be $\omega = \mu/\tau$, and adders and multipliers will present the following time and area performances.

For adders, with the constraint $1 \leqslant \omega \leqslant \mu$ (Brent and Kung 1982), it is obtained:

$$\begin{aligned} T_A &= O(\mu/\omega + \log\omega) \\ A_A &= O(\omega\log\omega) \end{aligned} \tag{9}$$

Similarly, for multipliers, with $\sqrt{\mu} \leqslant \omega \leqslant \mu/\log\mu$ (Mehlhorn and Preparata, 1983):

$$\begin{aligned} T_M &= O(\mu/\omega) \\ A_M &= O(\omega^2) \end{aligned} \tag{9'}$$

As the elements of each column are requested to operate in sequence, the same width ω of input strings is to be maintained in each column element in order to avoid mismatches or undesired delays. In addition, observing from Figure 1 that residue digits feed several columns (digit x_j is an input for columns $j,\ldots,n$), the same width ω of input strings is to be maintained throughout all columns. Assuming, without loss of generality, that moduli are of the same order, i.e.

$$\log m_i = \theta(\log m) = \theta(\mu), \quad i=\{1,\ldots,n\}$$

ω will be requested to satisfy both conditions $1 \leqslant \omega \leqslant \mu$ and $\sqrt{\mu} \leqslant \omega \leqslant \mu/\log\mu$, i.e.

$$\sqrt{\mu} \leqslant \omega \leqslant \mu/\log\mu \tag{10}$$

and, observing from (9) and (9′) that addition and multiplication times decrease for increasing ω, it is concluded that:

$$\omega = \mu/\log\mu \tag{11}$$

guarantees the shortest execution time.

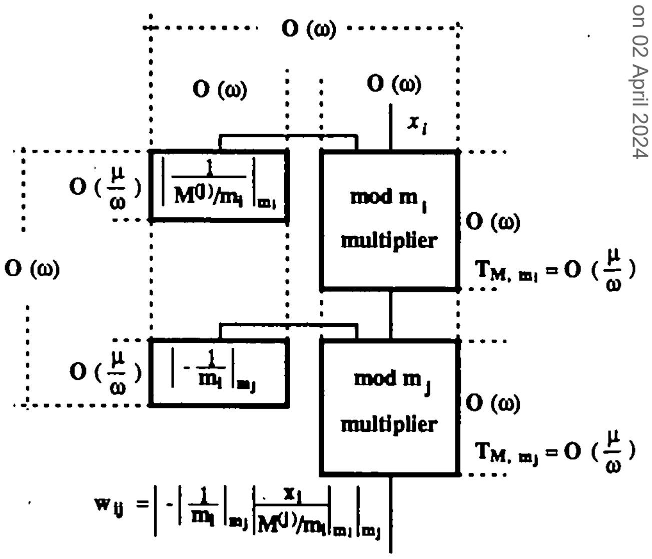
Now, let us see how the computational scheme of Figure 1 can be implemented.

3.1. Computing $w_{i,j}$

Figure 2 shows how each term $w_{i,j}$ is calculated from equations (7′) by means of two modular multipliers performing multiplications by constants. It is easy to verify that area and time figures coincide with those of a multiplier.

3.2. Computing MR digits of Q and R

Mixed radix digits q_j and r_j are computed (see equations 7″) from summations $\sum_{i=1}^{j} w_{i,j}$. These summations are obtained, as reported in Figure 3, by means of a pipelined tree of carry save adders (Wallace, 1964; Hwang, 1979) and the result has a total length of $(\mu+\log j)$ bits. As the input string length tends to increase in the tree, some modifications are to be introduced to the basic scheme (see Appendix B for detail).



O (ω)
O (ω)
O (ω)
x_i
O (μ/ω)
$\left|\frac{1}{M^{(i)}/m_i}\right|_{m_i}$
mod m_i multiplier
O (ω)
O (ω)
$T_{M,m_i} = O(\mu/\omega)$
O (μ/ω)
$\left|-\frac{1}{m_i}\right|_{m_j}$
mod m_j multiplier
O (ω)
$T_{M,m_j} = O(\mu/\omega)$
$w_{ij} = \left|-\left|\frac{1}{m_i}\right|_{m_j}\left|\frac{x_i}{M^{(i)}/m_i}\right|_{m_i}\right|_{m_j}$

FIGURE 2

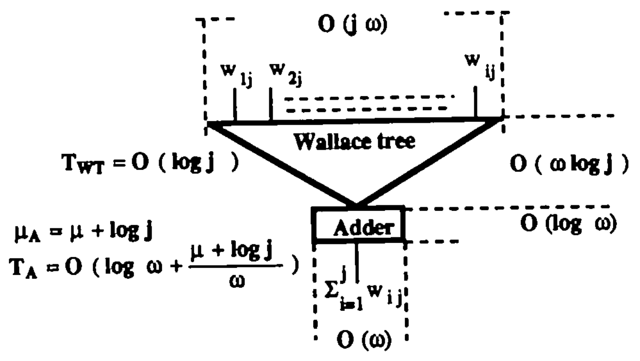


FIGURE 3

Starting from summations $\sum_{i=1}^{j} w_{i,j}$, Figure 4 shows a layout for computing both r_j and q_j digits. Digits r_j are easily obtained by means of a mod m_j converter (Barsi, 1991) whereas computing q_j's is a bit more difficult. However, recalling (7″) and taking an integer j^* such that:

$$j^* \geqslant j \text{ and } (j^*, m_j) = 1$$

q_j can be obtained as:

$$q_j = \left\lfloor \frac{\sum_{i=1}^{j} w_{i,j}}{m_j} \right\rfloor = \left| \left\lfloor \frac{\sum_{i=1}^{j} w_{i,j}}{m_j} \right\rfloor \right|_{j^*}$$

$$= \left| \frac{\sum_{i=1}^{j} w_{i,j} - \left|\sum_{i=1}^{j} w_{i,j}\right|_{m_j}}{m_j} \right|_{j^*}$$

$$= \left| \left|\frac{1}{m_j}\right|_{j^*} \left| \sum_{i=1}^{j} w_{i,j} - \left|\sum_{i=1}^{j} w_{i,j}\right|_{m_j} \right|_{j^*} \right|_{j^*}$$

$$= \left| \left|\frac{1}{m_j}\right|_{j^*} \left| \sum_{i=1}^{j} w_{i,j} - r_j \right|_{j^*} \right|_{j^*}$$

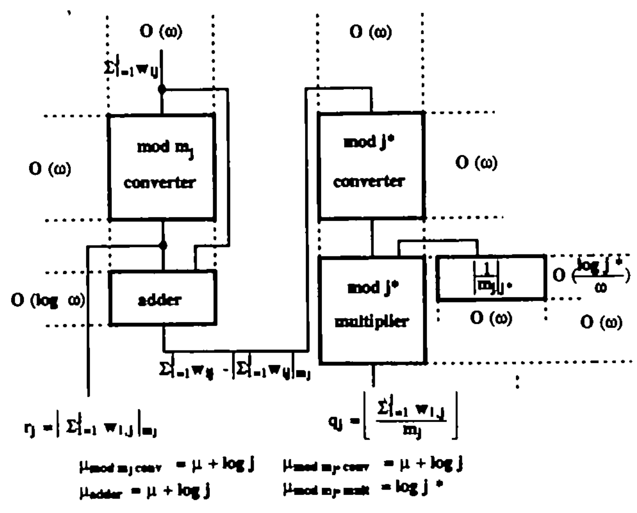


FIGURE 4

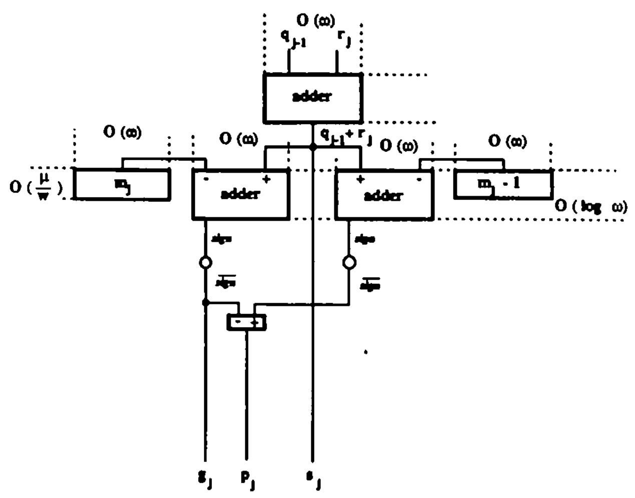


FIGURE 5

3.3. Computing $Q + R$ mod M

A carry generator logic can be provided to add integers Q and R by using a carry look ahead approach. In fact, carry from the jth position is given the same expression as in binary addition:

$$c_j = g_j + p_j c_{j-1}, c_0 = 0$$

where, recalling limitations (7″):

$$g_j = \left\lfloor \frac{r_j + q_{j-1}}{m_j} \right\rfloor, \quad g_j = \{0, 1\}$$

i.e.

$$g_j = 1 \text{ if } r_j + q_{j-1} \geqslant m_j \text{ else } g_j = 0$$

$$p_j = \left\lfloor \frac{r_j + q_{j-1}}{m_j - 1} \right\rfloor - \left\lfloor \frac{r_j + q_{j-1}}{m_j} \right\rfloor, \quad p_j = \{0, 1\} \qquad (12)$$

i.e.

$$p_j = 1 \text{ if } r_j + q_{j-1} = m_j - 1 \text{ else } p_j = 0$$

It is immediately verified that g_j and p_j cannot simultaneously assume the value '1'. They can be computed by a tree structure (carry chain computation logic) (Brent and Kung, 1982) having a n-bit input and a $(\log n)$ depth. The required mixed radix digits are then obtained as:

$$w_j = |r_j + q_{j-1} + c_{j-1}|_{m_j}$$

Figure 5 shows a possible design for computing $s_j = r_j + q_{j-1}$, g_j and p_j.

4. AREA AND TIME ASYMPTOTIC COMPLEXITIES

To evaluate the asymptotic complexity of the proposed implementation, area and time will be derived according to the VLSI model first introduced by Thompson (Thompson, 1980; Mehlhorn and Preparata, 1983). We abstract the model by recalling the major assumptions:

1. All wires have minimum width λ.
2. Transistors have minimum area $k_T\lambda^2$ and I/O pads have minimum area $k_{I/O}\lambda^2$.

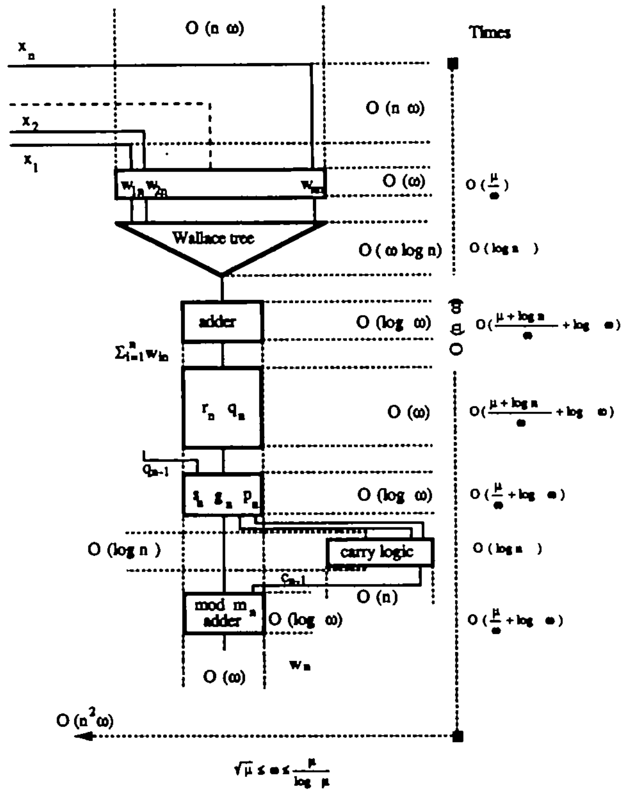


FIGURE 6

3. The time of an elementary action is constant regardless of wire length.
4. The computation time is the sum of the times of all elementary actions.
5. The I/O protocol is semellective and unilocal (i.e. each input is read exactly once and at exactly one input site)

Asymptotic area and time complexities could be easily derived from Figure 1 and recalling the proposed implementations of each computational step. However, for the sake of simplicity, Figure 6 shows a detailed description of the nth column together with the area occupancies and the execution times of all elementary blocks.

The total area is immediately derived observing that the proposed structure has width $O(\omega + 2\omega + \cdots + n\omega) = O(n^2\omega)$ and depth $O(n\omega)$, i.e.

$$A = O(n^3\omega^2) \tag{13}$$

To derive the conversion time, it can be observed that it is determined by the time required by the slowest column to process data and by the time $O(\log n)$ which is necessary for computing carries. The conversion time is then obtained as:

$$T = O\left(\frac{\mu + \log n}{\omega} + \log \omega + \log n\right)$$

and, taking $\omega = \mu/\log \mu$ to achieve the best computation time (equation 11), it follows:

$$T_{\min} = O\left(\log \mu + \frac{\log n \log \mu}{\mu} + \log n\right)$$

Recalling the assumption $\log m_j = \theta(\mu) = \theta(\log m)$, $j = \{1, \ldots, n\}$ and observing that $n < m$, it is concluded:

$$T_{\min} = O(\log \log m + \log n)$$

or, equivalently:

$$\begin{aligned} T_{\text{Conv}} &= O(\log \log m + \log n) = O(\log(n \log m)) \\ &= O(\log s) \end{aligned} \tag{14}$$

where $s = \log m^n = n \log m$ represents the total number of input bits.

As $\Omega(\log s)$ is a trivial lower bound for T, it is proved that the proposed layout represents a time optimal solution for the mixed radix conversion.

Similarly, with the same assumptions, the area occupancy (13) becomes:

$$A = O\left(n^3 \frac{\log^2 m}{\log^2 \log m}\right) = O\left(s^2 \frac{n}{\log^2 \log m}\right) \tag{13'}$$

Preceding results can be compared with the complexity of the conversion structures proposed in the literature by Alia and Martinelli (1984) yielding $O(\log^2 s)$ time with area $O(s^2 \log s)$ and by Capocelli and Giancarlo (1988) exhibiting $O(\log s \log n)$ time with

TABLE 2

		Time			Area		
n	log m	BP	AM	CG	BP	AM	CG
θ(const)	θ(s)	O(log s)	O(log² s)	O(log s)	$O(\frac{s^2}{\log^2 s})$	O(s² log s)	O(s²/log s)
θ(log s)	θ(s/log s)	O(log s)	O(log² s)	O(log s log log s)	$O(\frac{s^2 \log s}{\log^2 (s/\log s)})$	O(s² log s)	O(s²/log log s)
θ(s/log s)	θ(log s)	O(log s)	O(log² s)	$O(\log s \log \frac{s}{\log s})$	$O(\frac{s^3}{\log s \log^2 \log s})$	O(s² log s)	$O(s^3 \log \frac{s}{\log s})$
θ(s)	θ(const)	O(log s)	O(log² s)	O(log² s)	O(s³)	O(s² log s)	O(s³ log² s)

area $A = O(s^2n/\log n \log s)$. To this purpose, Table 2 reports complexity figures as a function of the total number of bits for some typical $(n, \log m)$ pairs.

It is concluded that the proposed VLSI design represents the only time optimal solution to the problem of converting a residue representation into a weighted notation. It is noteworthy that the area occupancies as well compare favourably for $n = O(\log s)$.

5. IC IMPLEMENTATIONS

To evaluate in practice the proposed method, a sketch of design will be presented using off-the-shelf ICs for implementing an MR converter. To emphasize time performances, our choice will be limited to ECL components. The proposed layout, which is reported in Figure 7, has been derived directly from the algorithm of Section 2 without any pipelined scheme of computation. It can be observed that:

1. Modular multiplications by constants for computing terms w_{ij} (equation 7′) are performed by means of CSA trees (see Appendix C for details).
2. Mixed radix digits q_j, r_j (equation 7″) and w_i are computed by using a comparator/subtractor logic whereas one adder and a comparator yield g_j and p_j (equation 12).

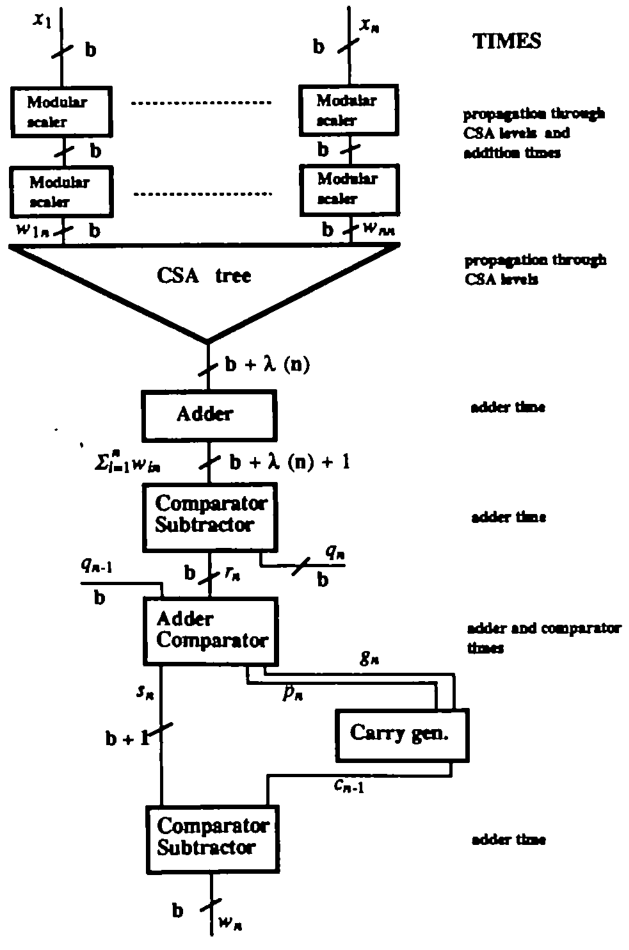


FIGURE 7

The time required to perform conversion can be derived following Figure 7. It can be expressed as:

$$T_{\mathrm{MRC}} = 2[\lambda_b t_p + t_{A(b+\lambda_b)} + t_{CS(b+\lambda_b+1)}] + \lambda_n t_p + t_{A(b+\lambda_n)} + t_{CS(b+\lambda_n+1)} + t_{A(b)} + t_{C(b)} + t_{\mathrm{carry}(n)} + t_{CS(b+1)} \tag{15}$$

where λ_h indicates, in general, the minimum number of CSA levels which is necessary to process h input data (see Appendix C); t_p is the propagation time through one CSA level; $t_{A(h)}$, $t_{C(h)}$, $t_{CS(h)}$ are the times required to add two h-bit numbers and to compare or compare/subtract an h-bit input number, respectively; and $t_{\mathrm{carry}(n)}$ is the time required to generate n binary carries from $n + n$ bit input.

Observing that a comparator/subtractor can be designed as a row of adders followed by a trivial test logic, it will be assumed that $t_{CS(h)} = t_{A(h)}$ and equation (15) will become:

$$T_{\mathrm{MRC}} = 2\lambda_b t_p + \lambda_n t_p + 2t_{A(b+\lambda_b)} + 2t_{A(b+\lambda_b+1)} + t_{A(b+\lambda_n)} + t_{A(b+\lambda_n+1)} + t_{A(b+1)} + t_{A(b)} + t_{C(b)} + t_{\mathrm{carry}(n)} \tag{15′}$$

As an example, let $b = \lceil \log m \rceil = 12$ be the number of residue digits bits and $n = 13$ be the number of moduli, with a dynamic range of 156 bits. Then, $l_b = \lambda_n = 5$ (Hwang, 1979).

Using 6-bit 100180 type elements with 2 ns execution time for constructing adders and 9 bit 10E166 type comparators with 1.4 ns, we obtained:

$$t_{A(b+\lambda_b)} = t_{A(b+\lambda_n)} = t_{A(17)} = 6\,\mathrm{ns}$$

$$t_{A(b+\lambda_n+1)} = t_{A(b+\lambda_b+1)} = t_{A(18)} = 6\,\mathrm{ns}$$

$$t_{A(b+1)} = t_{A(13)} = 6\,\mathrm{ns}$$

$$t_{A(b)} = t_{A(12)} = 4\,\mathrm{ns}$$

$$t_{C(12)} = 2.8\,\mathrm{ns}.$$

Taking a 2 ns carry generator logic 100179 type and assuming 1 ns/(CSA-tree level) propagation time, the conversion time is derived from equation (15′) as:

$$T_{\mathrm{MRC}} = 66\,\mathrm{ns}$$

In a look-up table approach, modular scalers can be replaced by $2^b \times b$ tables and the conversion time is obtained from equation (15) by substituting the memory access time t_T for the term $2[\lambda_b t_p + t_{A(b+\lambda_b)} + t_{CS(b+\lambda_b)}]$, i.e.

$$T_{\mathrm{MRC}}(\text{lookups}) = t_T + \lambda_n t_p + t_{A(b+\lambda_n)} + t_{A(b+\lambda_n+1)} + t_{A(b+1)} + t_{A(b)} + t_{C(b)} + t_{\mathrm{carry}(n)} \tag{17}$$

In the residue system of the preceding example, using a fast 4 k × 4 ECL RAM module 10A484 type with 5 ns access time yields:

$$T_{\mathrm{MRC}}(\text{lookups}) = 37\,\mathrm{ns}$$

TABLE 3

b	n	s	T_{MRC} (ns)	T_{MRC} (RAM) (ns)
8	19	152	58	35
10	15	150	64	35
10	28	280	65	36
12	13	156	66	37
12	26	312	70	41
28	9	252	114	not allowed
40	3	120	145	not allowed
40	6	240	148	not allowed

A more complete evaluation of the conversion times is shown in Table 3 for some typical RNSs with 100–300 bit ranges. Data have been derived by using similar ECL components and fast RAM modules. The results reported in Table 3 stress once again that a memory approach is not viable whenever, as in applications using a limited number of large moduli, the value of b exceeds a given threshold. At the present state of technology, $b = 22$ (i.e. 4 M memory modules) is to be considered as an upper bound.

6. CONCLUSIONS

A new parallel method to perform mixed radix conversion has been presented which is based upon an innovative formulation of the CRT. This method can be used for both conventional logic and look-up table implementations and produces, in a fully modular and parallel fashion, the result in the form of a pair (R, Q) of mixed radix integers whose sum is the explicit mixed radix representation of the number to be converted.

An implementation of the method has been compared with previous results (Huang, 1983; Chakraborti *et al.*, 1986) by adopting similar complexity criteria and it has been verified that significant enhancements are obtained for the conversion time, whereas the hardware requirements are slightly increased.

The method has also been evaluated from an asymptotic point of view by using a general VLSI model of computation and an appropriate layout, and it has been found that the proposed algorithm enables time optimal VLSI implementations.

Practical ECL IC implementations have been indicated exhibiting a conversion time of 60–70 ns for dynamic ranges up to 300 bit whereas RAM-based implementations attain 35–40 ns.

ACKNOWLEDGMENTS

This research was partially supported by the Italian National Research Council (CNR) in the Project 'Sistemi Informatici e Calcolo Parallelo' and by MURST in the Project 'Algoritmi, Modelli di Calcolo e Strutture Informative'.

REFERENCES

Alia, G. and Martinelli, E. (1984) A VLSI algorithm for direct and reverse conversion from weighted binary number system to residue number system. *IEEE Trans. Circuits Syst.*, **CAS-31**, 1033–1039.

Alia, G. and Martinelli, E. (1991) A VLSI modulo *m* multiplier. *IEEE Trans. Comput.*, **C-40**, 873–878.

Barsi, F. (1991) Mod *m* arithmetic in binary systems. *Inform. Process. Lett.*, **40**, 303–309.

Barsi, F. and Pinotti, M. C. (1992) *Optimal Time Mixed Radix Conversion VLSI Algorithms*. IEI Internal Report B4-09.

Brent, R. P. and Kung, H. T. (1982) A regular layout for parallel adders. *IEEE Trans. Comput.*, C-31, 260–264.

Capocelli, R. M. and Giancarlo, R. (1988) Efficient VLSI networks for converting an integer from binary system to residue number system and vice versa. *IEEE Trans. Circuits Syst.*, **CAS-35**, 1425–1430.

Chakraborti, N. B., Soundararajan, J. S. and Reddy, A. L. N. (1986) An implementation of mixed-radix conversion for residue number applications. *IEEE Trans. Comput.*, **C-35**, 762–764.

Elleithy, K. M. and Bayoumi, M. A. (1992) Fast and flexible architectures for RNS arithmetic decoding. *IEEE Trans. Circuits Syst. II*, **CAS-II 39**, 226–235.

Huang, C. H. (1983) A fully parallel mixed-radix conversion algorithm for residue number applications. *IEEE Trans. Comput.*, **C-32**, 398–402.

Hwang, K. (1979) *Computer Arithmetic: Principles, Architecture, and Design*. John Wiley, New York.

Mehlhorn, K. and Preparata, F. P. (1983) Area-time optimal VLSI integer multiplier with minimum computation time. *Inf. Control*, **58**, 137–156.

Shenoy, A. P. and Kumaresan, R. (1988) Residue to binary conversion for RNS arithmetic using only modular look-up tables. *IEEE Trans. Circuits Syst.*, **CAS-35**, 1158–1162.

Szabo, N. S. and Tanaka, R. I. (1967) *Residue Arithmetic and Its Applications to Computer Technology*. McGraw Hill, New York.

Thompson, C. D. (1980) *A Complexity Theory for VLSI*. PhD Thesis, Computer Science, Carnegie-Mellon University.

Vu, T. V. (1985) Efficient implementations of the Chinese remainder theorem for sign detection and residue decoding. *IEEE Trans. Comput.*, **C-34**, 646–651.

Wallace, C. S. (1964) A suggestion for a fast multiplier. *IEEE Trans. Electronic Comput.*, **EC-13**, 14–17.

APPENDIX A

$$X = \left|\sum_{i=1}^{n} M_i \left|\frac{x_i}{M_i}\right|_{m_i}\right|_M = \left|\sum_{i=1}^{n} X_i\right|_M$$

$$= \left|\sum_{i=1}^{n}\sum_{j=1}^{n} w_{i,j} M^{(j)}\right|_M = \left|\sum_{j=1}^{n} M^{(j)} \sum_{i=1}^{n} w_{i,j}\right|_M$$

$$= \left|\sum_{j=1}^{n} M^{(j)} \sum_{i=1}^{j} w_{i,j}\right|_M$$

$$= \left|\sum_{j=1}^{n} M^{(j)} \left\{ m_j \left\lfloor \frac{\sum_{i=1}^{j} w_{i,j}}{m_j} \right\rfloor + \left|\sum_{i=1}^{j} w_{i,j}\right|_{m_j} \right\}\right|_M$$

$$= \left|\sum_{j=1}^{n} M^{(j+1)} \left\lfloor \frac{\sum_{i=1}^{j} w_{i,j}}{m_j} \right\rfloor + \sum_{j=1}^{n} M^{(j)} \left|\sum_{i=1}^{j} w_{i,j}\right|_{m_j}\right|_M$$

$$= \left| M^{(n+1)} \left\lfloor \frac{\sum_{i=1}^{n} w_{i,n}}{m_n} \right\rfloor + \sum_{j=1}^{n-1} M^{(j+1)} \left\lfloor \frac{\sum_{i=1}^{j} w_{i,j}}{m_j} \right\rfloor \right.$$

$$\left. + \sum_{j=1}^{n} M^{(j)} \left| \sum_{i=1}^{j} w_{i,j} \right|_{m_j} \right|_M$$

$$= \left| \sum_{j=1}^{n-1} M^{(j+1)} \left\lfloor \frac{\sum_{i=1}^{j} w_{i,j}}{m_j} \right\rfloor + \sum_{j=1}^{n} M^{(j)} \left| \sum_{i=1}^{j} w_{i,j} \right|_{m_j} \right|_M$$

$$= \left| \sum_{j=2}^{n} M^{(j)} \left\lfloor \frac{\sum_{i=1}^{j-1} w_{i,j-1}}{m_{j-1}} \right\rfloor + \sum_{j=1}^{n} M^{(j)} \left| \sum_{i=1}^{j} w_{i,j} \right|_{m_j} \right|_M$$

APPENDIX B

A *pseudo* (Wallace, 1964) or *carry save* adder (CSA) is a three input–two output device consisting of a set of disconnected full adder elements whose sum and carry outputs form the output of CSA. When a multioperand adder is to be designed, a CSA-tree connection [Wallace Tree (Wallace, 1964)] represents the best known solution to reduce the addition time.

In this paper, some minor modifications to the original WT structure have been introduced in order to maintain a pipelined scheme of computation with input and output strings of the same width ω.

Let, in general, l_{WT} be the number of a Wallace Tree levels. With ω-bit inputs, the WT outputs S (*sum*) and C (*carry*) will produce $(\omega + l_{WT} - 1)$ and $(\omega + l_{WT})$ bits, respectively, as shown in Figure B1.

To remove this obstacle and to infer the solution, let us consider the WT outputs starting from the beginnings. As shown in Figure B2, the ω least significant bits of $s(1)$ and $c(1)$ can be fed into final adder, whereas a four input WT is necessary to derive the second (and, in general, the hth) output pair $s''(2)$ and $c''(2)$ of width ω to be considered as the second output string of weight 2^{ω}. Figures B3 and B4 shows in detail how this four input WT can be constructed by using two CSAs.

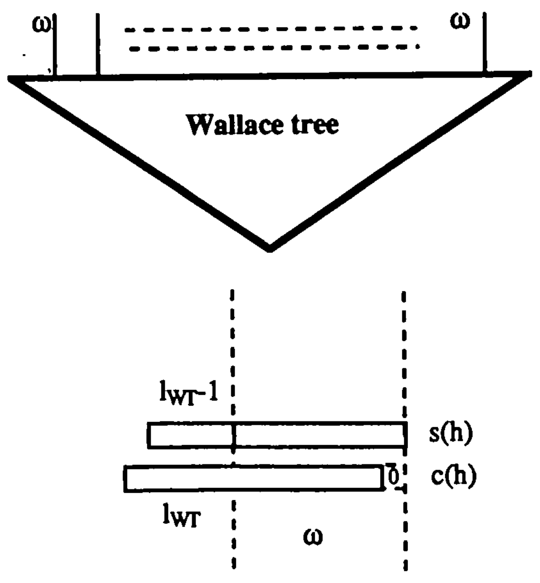


FIGURE B1

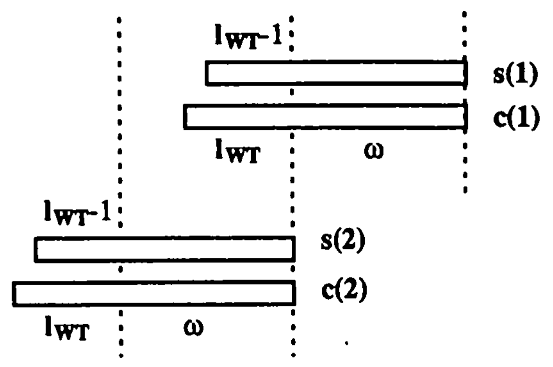


FIGURE B2

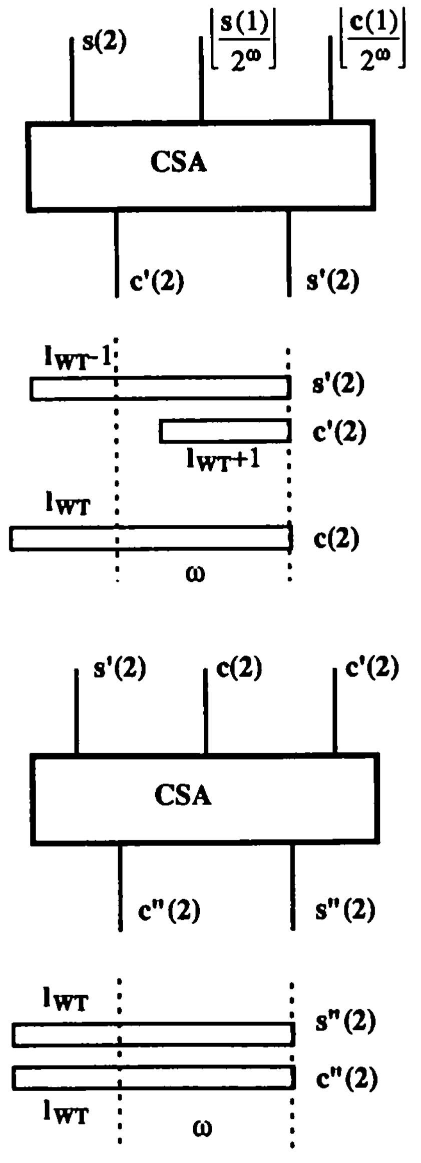
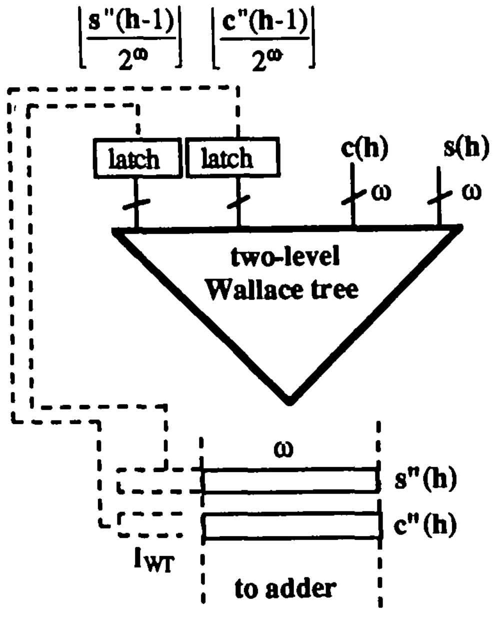


FIGURE B3



$\left\lfloor \frac{s''(h-1)}{2^\omega} \right\rfloor$
$\left\lfloor \frac{c''(h-1)}{2^\omega} \right\rfloor$
latch
latch
c(h)
s(h)
ω
ω
two-level Wallace tree
ω
s"(h)
c"(h)
lwt
to adder

FIGURE B4

APPENDIX C

Modular multiplication by a constant (modular scaling) can be performed without using multipliers by adding at most b values stored in a set of registers, where b is the number of bits which are necessary to represent residue digits.

Let k be the scaling constant and suppose that:

$$|xk|_{m_i}$$

is to be computed. Observing that:

$$x = \sum_{u=0}^{b-1} x_u 2^u, \quad x_u \in \{0, 1\}$$

it is obtained:

$$|xk|_{m_i} = \left| \sum_{u=0}^{b-1} k x_u 2^u \right|_{m_i} = \left| \sum_{u=0}^{b-1} x_u |k 2^u|_{m_i} \right|_{m_i}$$

Assuming that a set of registers are storing constants $|k2^u|_{m_i}$ and observing that $x_u|k2^u|_{m_i} = 0$ or $|k2^u|_{m_i}$ the scaling operation can be performed by means of a CSA-tree followed by a two operand adder and a comparator/subtractor logic. The number of CSA-tree levels will be derived directly or according to tables reported in Hwang (1979).