An On-Line Square Root Algorithm

VOJIN G. OKLOBDŽIJA AND MILOŠ D. ERCEGOVAC

Abstract—**In this correspondence a systematic derivation of an on-line square root algorithm is presented. The algorithm operates on variables represented in the normalized radix r floating-point system in a digit-by-digit fashion with an on-line delay of 1. The approach used in deriving the square root algorithm is described in detail. The basic characteristics of hardware-level implementation are also discussed.**

Index Terms—**Floating-point square root algorithm, modular LSI/VLSI implementation, on-line arithmetic, redundant number systems.**

I. Introduction

An on-line arithmetic operation is performed in a digit-by-digit manner beginning with the most significant digit. At each step only one additional digit of the input variables is required in order to generate the next digit of the output variable. In general, an on-line algorithm may require the first δ operand digits to be known in order to start the operation. This number δ, called the on-line delay, is a small positive integer between 1 and 4 for the basic arithmetic operations [15], [11].

On-line arithmetic has several advantageous features over conventional full-precision arithmetic. First, a sequence of arithmetic operations can be performed faster by overlapping the operations at the digit level. For example, n operations which must be performed in the sequential order require only $m + \sum_{i=1}^{n} \delta_i$ digit steps for an m digit precision instead of mn digit steps. Second, the on-line arithmetic unit has a minimal input/output bandwidth of one digit per variable. This feature enhances the modularity of implementation and reduces the interconnection complexity, which makes the algorithm convenient for LSI/VLSI implementation. The use of on-line units is attractive in array and other concurrent computer system organizations. The on-line algorithms provide also a simple approach for achieving variable precision computations. The significance monitoring [4] can also be directly incorporated in on-line arithmetic. Finally, some low-cost error checking codes [16], [17], defined for serial arithmetic, can also be efficiently applied in on-line algorithms.

There have been several algorithms developed which satisfy the on-line property with respect to the operands and the result. Possibly, the first such attempt was done in [1] for a conventional, right-to-left multiplication algorithm. Multiplication and division in the on-line, left-to-right manner were developed in [15] and their implementation together with addition and subtraction algorithms considered in [11].

Common to all on-line algorithms is the use of redundancy in the number representation. The redundant number representation has often proved useful for speeding up arithmetic operations [2], [3], [5], [8], [14] and its application in the on-line algorithms is essential. In a nonredundant system even a simple operation like addition would have a significant on-line delay due to the carry propagation. By allowing redundancy in the number representation it is possible to limit the carry propagation to one (or two) digit positions and thus the left-to-right computations are possible. A general evaluation method [8] is also characterized by on-line property. The results described here represent a generalization of a binary square rooting algorithm [9].

Manuscript received March 7, 1980; revised June 29, 1981. This work was supported in part by the Department of Energy under Contract DE-AS03-76SF0034 P.A. and the ONR under Contract N00014-79-C-0866.

The authors are with the Department of Computer Science, University of California, Los Angeles, CA 90024.

The square root algorithm is derived under the following requirements [8].

1) The algorithm is on-line with respect to the operand digits. At some step j of the computation only those operand digits up to and including jth are used for performing the computation and for producing the result digit.

2) The algorithm is on-line with respect to the result digits. The most significant digit of the result is generated first. The result digit produced at step j is not affected by the digits produced at the subsequent steps k $(k > j)$.

3) The time required by the computational step is invariant and the only primitive operations are carry-free addition and multiplication with single-digit multipliers.

4) The implementation requirements of the unit are compatible with the constraints of LSI/VLSI technology, i.e., it is modular with high gate/pin ratio.

II. Derivation of the Algorithm

The argument

$$X = \sum_{i=1}^{m} x_i r^{-i} \tag{1}$$

is a positive number in the range $[r^{-p}, r^{-p+1})$ and represented in the redundant number system with the digit set

$$D_\rho = \{-\rho, \cdots, -1, 0, 1, \cdots, \rho\}, \qquad r/2 \le \rho < r.$$

The scaling factor p is a positive integer which will be determined later.

At the jth step of the algorithm the argument is represented as

$$X_j = X_{j-1} + x_{\delta+j} r^{-\delta-j} \tag{2}$$

where $\delta > 0$ is the on-line delay. Initially,

$$X_0 = \sum_{i=1}^{\delta} x_i r^{-i} \tag{3}$$

and at the end $X_m = X$.

The result

$$Y = \sum_{i=1}^{m} y_i r^{-i}$$

is in the range $[r^{-p/2}, r^{(1-p)/2})$. The on-line representation of the result is

$$Y_j = Y_{j-1} + y_j r^{-j}, \qquad j = 1, 2, \cdots, m \tag{4}$$

where $Y_0 = 0$.

Recursion

The recursion is obtained in the usual way from the scaled partial remainder

$$R_j = r^j [X_j - Y_j^2] \tag{5}$$

so that

$$\begin{aligned} R_j &= rR_{j-1} + x_{\delta+j} r^{-\delta} - y_j [2Y_{j-1} + y_j r^{-j}] \\ &= rR_{j-1} + x_{\delta+j} r^{-\delta} - y_j [Y_{j-1} + Y_j] \qquad j = 1, 2, \cdots, m. \end{aligned} \tag{6}$$

Initially, $R_0 = X_0$.

Since the partial remainder is bounded

$$|R_j| \leq c < 1 \quad (7)$$

the convergence of the algorithm is assured. The error is defined as

$$e = |X_m^{1/2} - Y_m| \quad (8)$$

and it satisfies the following condition:

$$|e| \leq \frac{c}{2} r^{-m+p/2}. \quad (9)$$

As will be seen later, the scaling factor $p = 2$ and

$$|e| \leq \frac{c}{2} r^{-m+1}. \quad (10)$$

Selection Procedure

The algorithm requires a suitable procedure for selecting the result digits in on-line manner. In this section we discuss an approach in defining selection rules. The selection of the result digit y_j is based on the comparison of the partial remainder R_{j-1} with some suitable comparison constants. In particular, we show that a low-precision estimate of the partial remainder can be used [2], [3], [5], [8], [14] and that the required constants are relatively simple to obtain.

In order to define the selection function we consider the intervals in which the partial remainder R_{j-1} can be found for different values of the output digit y_j. The intervals are defined as

$$I_k = [a_k, b_k] \qquad -\rho \leq k \leq \rho.$$

These intervals must satisfy the following conditions [9].

1) The continuity condition:

$$a_k \leq b_{k-1} \qquad -\rho < k \leq \rho. \quad (11)$$

2) The containment condition:

$$a_{-\rho} \leq R_j \leq b_\rho. \quad (12)$$

The selection intervals are easily obtained from the recursion formula (6) by substituting the boundary values of $|R_j| = c$ and $y_j = k$ for $-\rho \leq k \leq \rho$:

$$a_k = -\frac{c}{r} - u + \frac{k}{r}[2Y_{j-1} + kr^{-j}] \quad (13)$$

and

$$b_k = \frac{c}{r} - u + \frac{k}{r}[2Y_{j-1} + kr^{-j}] \quad (14)$$

where

$$u = x_{\delta+j} r^{-\delta-1}.$$

In order to satisfy the continuity condition (11), there must exist a nonzero overlap Δ between the adjacent intervals I_k and I_{k+1}. In this overlap region the choice of the result digits k and $k+1$ is equally valid.

Specifically, the overlap $\Delta_{k-1,k}$ between the intervals I_k and I_{k-1} is

$$\Delta_{k-1,k} = b_{k-1} - a_k = \frac{1}{r}[2c - 2Y_j^{(k)} + r^{-j}] \quad (16)$$

for $1 - \rho \leq k \leq \rho$, where $Y_j^{(k)}$ denotes the value of Y_j with k selected as the digit in the jth position.

The minimum overlap occurs for $k = \rho$. Let the minimum positive value for the overlap be

$$\Delta = r^{-t}. \quad (17)$$

From the continuity condition (16) and (17) we obtain that in the worst case the following relation must hold:

$$\frac{1}{r}[2c - 2Y_j^{(\rho)} + r^{-j}] \geq r^{-t}. \quad (18)$$

In order to guarantee a nonnegative overlap Δ, the maximum value of the result Y should not exceed the partial remainder bound c, i.e.,

$$Y_{\max} \leq c. \quad (19)$$

Since $Y_{\max} = [X_{\max}]^{1/2}$ and $c < 1$, it follows that the argument X should be shifted initially for at least one digit position to the right in order to satisfy (19). This implies that the range scaling factor p is greater than one. Since in general the initial scaling of the argument increases the on-line delay, we select $p = 2$. As will be seen later, there is no time penalty for this preshifting of the argument.

The minimum on-line delay δ and the minimum overlap r^{-t} are determined as follows. The containment condition (12) implies that the selection intervals should contain completely the range of the partial remainders. Therefore, the maximum value of R_j should satisfy the following condition:

$$R_j = rc - \rho[Y_{j-1} + Y_j^{(\rho)}] + ru \leq c. \quad (20)$$

In the worst case $x_{\delta+j} = \rho$ and

$$c \leq K[Y_{j-1} + Y_j^{(\rho)} - r^{-\delta}] \quad (21)$$

where $K = \rho/(r-1)$ is the redundancy factor. Equation (21) shows that in order to minimize the on-line delay δ the minimum value of the result and, hence the argument should be as large as possible. This reasoning is consistent with our choice of $p = 2$. The other expected conclusion from (21) is that more redundancy leads to a faster algorithm, i.e., a smaller on-line delay.

Since $c > 0$, and $p = 2$ it follows that

$$2Y_{\min} = 2r^{-1} > r^{-\delta} \quad (22)$$

which indicates that the minimal on-line delay is $\delta_{\min} = 1$.

From (18) and (21) we obtain

$$r^{-t+1} + r^{-\delta} \leq c\left(2 - \frac{1}{K}\right) - (\rho - 1)r^{-j}. \quad (23)$$

Again, maximal redundancy in the number representation would allow the choice of a smaller value for t. This means that we could truncate the remainder to fewer digits in the comparison process and obtain a faster algorithm. For this reason the algorithm is designed to operate with the maximally redundant digit set which, in addition, has an advantage in the fact that no input conversion is necessary when the argument is in the conventional form. In the limit ($j = \infty$), for $K = 1$ and $\delta = 1$

$$r^{-t} \leq \frac{Y_{\max} - r^{-\delta}}{r} = \frac{r^{1/2} - 1}{r^2}. \quad (24)$$

The smallest t we can choose is $t = 2$. For $t = 2$, the equation (24) reduces to $r^{1/2} \geq 2$ and it will be satisfied for any radix $r \geq 4$. For radix $r = 2$, a somewhat different algorithm is proposed in [9].

We are now able to define the selection procedure to be used in the algorithm. In each overlap region we choose a value B_k as close to the overlap midpoint as possible. The selection intervals and these comparison values are indicated in Fig. 1.

$$B_k = A_k Y_{j-1} + D_k r^{-(j+1)} - u \quad (25)$$

and

$$B_{-k} = -A_k Y_{j-1} + D_k r^{-(j+1)} - u \qquad \text{for } k = 1, \cdots \rho$$

where $A_k = (2k-1)/r$ and $D_k = k^2 - k + 1/2$. These coefficients require a small number of digits. A most important consequence of the use of redundancy is that only estimates of B_k's need to be known. We utilize one half of the overlap $\Delta_{k-1,k}$ for these estimates and another half for approximation of the partial remainders. For $t = 2$ we define these estimates to satisfy the following conditions:

$$|B_k - \hat{B}_k| < r^{-3} \quad (26)$$

and

$$|R_j - \hat{R}_j| < r^{-3}. \quad (27)$$

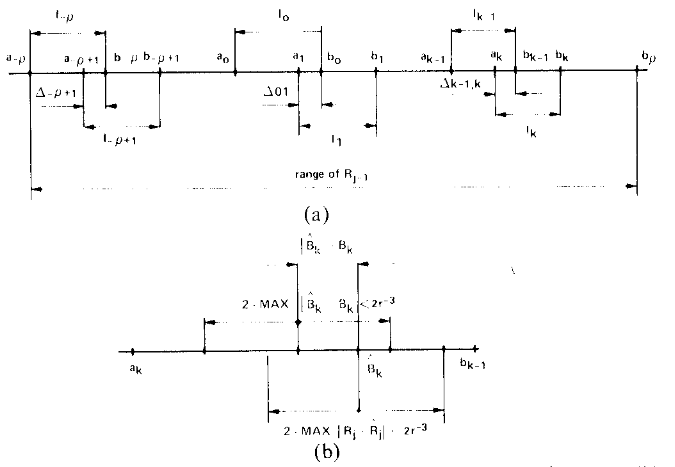

Fig. 1. (a) Selection intervals Ik and the overlap regions $\Delta_{k-1,k}$. (b) Comparison process between truncated selection constants $\hat{B}_k$ and truncated partial remainder $\hat{R}_j$.

The selection function is now defined as

$$y_j = S(\hat{R}_{j-1}) = \begin{cases} k & \hat{B}_k < \hat{R}_{j-1} \le \hat{B}_{k+1} \\ 0 & \hat{B}_{-1} < \hat{R}_{j-1} \le \hat{B}_1 \\ -k & \hat{B}_{-k-1} < \hat{R}_{j-1} \le \hat{B}_{-k} \end{cases} \tag{28}$$

where

$$\hat{B}_k = A_k \hat{Y}_{j-1} + \hat{D}_k - u \tag{29}$$

and

$$\hat{B}_{-k} = -A_k \hat{Y}_{j-1} + \hat{D}_k - u$$

and

$$\hat{Y}_j = \lfloor r^3 Y_j \rfloor r^{-3}; \qquad u = x_{j+1} r^{-2}, \tag{30}$$

i.e., the partial result is truncated to three most significant digits. The term

$$\hat{D}k = \lfloor r^3 (D_k r^{-j-1}) \rfloor r^{-3} \tag{31}$$

can be neglected for $j > 4$.

III. Algorithm

As mentioned in Section II, in order to assure the nonzero positive overlap between the selection intervals, the input X is shifted for one digit position to the right. Therefore, X is in the range

$$X \leftarrow [r^{-2}, r^{-1}) \tag{32}$$

and the output Y is in the range

$$Y \leftarrow [r^{-1}, r^{-1/2}). \tag{33}$$

In the case of a normalized floating-point representation system the following adjustments are performed. Let the argument whose square root we seek be

$$Z = zr^{E_z} \tag{34}$$

and the result

$$W = wr^{E_w} = (z^{1/2}) r^{E_z/2}. \tag{35}$$

Since the exponent E_w should be an integer, the following correspondence between the floating-point argument (result) and the input X (the output Y) is defined.

i) E_z **even**: The argument fraction z is shifted two positions to the right in order to satisfy (32) and obtain an integer exponent.

$$Z = (zr^{-2}) r^{E_z+2} \tag{36}$$

and

$$W = [r(zr^{-2})^{1/2}] r^{E_z/2}. \tag{37}$$

Therefore,

$$x_1 = x_2 = 0 \tag{38}$$
$$x_i = z_{i-2} \qquad \text{for } i = 3, 4, \cdots, m+2$$
$$x_{m+3} = 0$$

and

$$w_i = y_{i+1} \qquad \text{for } i = 1, 2, \cdots, m; \quad E_w = E_z/2.$$

ii) E_z **odd**: In this case

$$Z = (zr^{-1}) r^{E_z+1} \tag{39}$$

and

$$W = [(zr^{-1})^{1/2}] r^{(E_z+1)/2}.$$

Therefore,

$$x_1 = 0 \tag{40}$$
$$x_i = z_{i-1} \qquad \text{for } i = 2, 3, \cdots, m+1$$
$$x_{m+2} = x_{m+3} = 0$$

and

$$w_i = y_i \qquad \text{for } i = 1, 2, \cdots, m; \quad E_w = (E_{z+1})/2.$$

We now state the on-line square root floating-point algorithm.

S-Algorithm

```
1) [Initialization]
   $w \leftarrow 0;\ R \leftarrow z_1 r^{-2};\quad E_w = \begin{cases}(E_z+1)/2: E_z \text{ odd} \\ E_z/2: E_z \text{ even.}\end{cases}$
2) [Recursion]
   For $j = 1, 2, \cdots, m$ do:
   2.1) for $k = 1, \cdots \rho$
        $\hat{B}_{\pm k} = \pm A_k \hat{w} + \hat{D}'_k - u$
              where: $\hat{D}'_k = \lfloor r^3 (D_k r^{-i-1}) \rfloor r^{-3}\ u \leftarrow z_{j+1} r^{-3}$
   2.2) $w_j \leftarrow S(\hat{R}, \hat{B}_i)$
   2.3) $w \leftarrow w + w_j r^{-i};$
        $R \leftarrow rR + z_{j+1} r^{-2} - 2Ww_j - w_j^2 r^{-i}.$
3) [End]
   $W = wr^{E_w}$
```

where $\hat{R}$ is a scaled remainder truncated to 3 most significant digits, B_k is the selection constant as defined in (29), and

$$i = \begin{cases} j & \text{if } E_z \text{ is even} \\ j+1 & \text{if } E_z \text{ is odd.} \end{cases}$$

Note that the next w is generated by concatenating the jth digit w_j to the right of the present w. The semicolon (;) separates operations that can be performed concurrently. Two examples are shown in Figs. 2 and 3.

IV. Implementation Considerations

The proposed on-line algorithm has two characteristics important for a modular implementation in LSI/VLSI technology. First, a small number of inputs and outputs is needed because of the digit-by-digit mode of operation. Second, the primitive operations (limited carry/borrow propagation addition/subtraction and concatenation) can be efficiently implemented in a modular fashion. Thus, an imple-

$(r=10, \ \rho=9)$

$$Z = 0.980\bar{7}3\bar{8}946*10^{-13}$$

$$\text{VALUE OF } Z = 0.979322946*10^{-13}$$

$$\sqrt{Z} = 0.3129413596826*10^{-6}$$

j	Z_j	$\hat{R}$	$\hat{B}_{\pm k}$	$\hat{B}_{\pm (k+1)}$	w_j	w^*	$E = \sqrt{Z} - W$
0	-	0.09	-	-	-	0	-
1	9	0.08	0.057	0.117	3	0.3	$1.294\ 10^{-8}$
2	8	0.19	0.030	0.092	1	0.31	$2.941\ 10^{-9}$
3	0	-0.039	0.162	0.225	3	0.313	$-5.864\ 10^{-11}$
4	-7	0.265	-0.034	-0.969	-1	0.3129	$4.135\ 10^{-11}$
5	3	0.075	0.226	0.228	4	0.31294	$1.359\ 10^{-12}$
6	-8	0.220	0.022	0.084	1	0.312941	$3.596\ 10^{-13}$
7	9	-0.259	0.214	0.276	4	0.3129414	$-4.031\ 10^{-14}$
8	4	-0.022	-0.024	-0.286	-4	0.31294136	$-3.173\ 10^{-16}$
9	6	-0.216	-0.031	-0.031	0	0.312931360	$-3.173\ 10^{-16}$
10	0	-	-0.156	-0.218	-3	0.3129413597	$-1.739\ 10^{-17}$

w^* - partially converted on-line representation of the result.

Result:

$$W = 0.313\bar{1}414\bar{4}0\bar{3}\ *(10)^{-6}$$

which corresponds to the value $0.3129413597\ *(10)^{-6}$

Fig. 2. Square root example for radix $r = 10$.

$(r=256, \ \rho=255)$

$$Z = .124\ 245\ 023\ 146\ 087\ 235\ 189\ 000\ *(256)^{-3}$$

$$\text{VALUE OF } Z = 2.909390975835\ *10^{-8}$$

$$\sqrt{Z} = 1.705693693438304\ *10^{-4}$$

j	Z_j	$\hat{R}$	$\hat{B}_{\pm k}$	$\hat{B}_{\pm(k+1)}$	w_j	$\epsilon = \sqrt{Z} - W$
0	-	$1.892089\ 10^{-3}$	-	-	-	-
1	124	$1.545715\ 10^{-2}$	$1.671493\ 10^{-3}$	$2.007186\ 10^{-3}$	11	$2.7226\ 10^{-6}$
2	245	$-2.803039\ 10^{-2}$	$1.539605\ 10^{-2}$	$1.573723\ 10^{-2}$	46	$-1.9124\ 10^{-6}$
3	23	$-1.196694\ 10^{-2}$	$-2.781313\ 10^{-2}$	$-2.815425\ 10^{-2}$	-82	$-3.1890\ 10^{-11}$
4	146	$-5.606698\ 10^{-2}$	$-1.177447\ 10^{-2}$	$-1.211561\ 10^{-2}$	-35	$-5.8243\ 10^{-14}$
5	87	$-3.442462\ 10^{-2}$	$-5.301658\ 10^{-3}$	$-5.642797\ 10^{-3}$	-16	$-1.3999\ 10^{-15}$
6	235	$1.066394\ 10^{-2}$	$-3.429571\ 10^{-2}$	$-3.463685\ 10^{-2}$	-101	$1.6805\ 10^{-18}$
7	189	$2.269325\ 10^{-2}$	$1.040473\ 10^{-2}$	$1.074587\ 10^{-2}$	31	0.0000
8	0	-	$2.268435\ 10^{-2}$	$2.302549\ 10^{-2}$	67	$-1.3552\ 10^{-20}$

Result: $W = 0.\ 011\ 046\ \overline{082}\ \overline{035}\ \overline{016}\ \overline{101}\ 031\ 067\ *(256)^{-1}$ which corresponds to the value $0.705693693438304\ *10^{-4}$

Fig. 3. Square root example for radix $r = 256$.

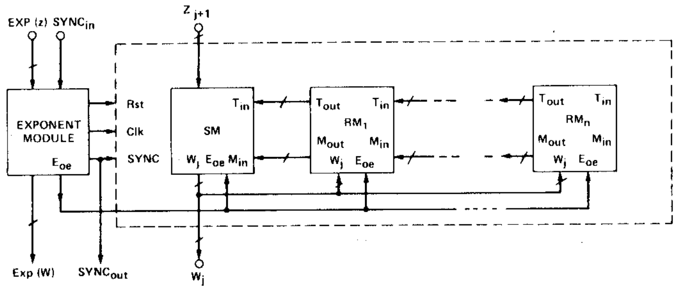

Fig. 4. Organization of the on-line square root unit.

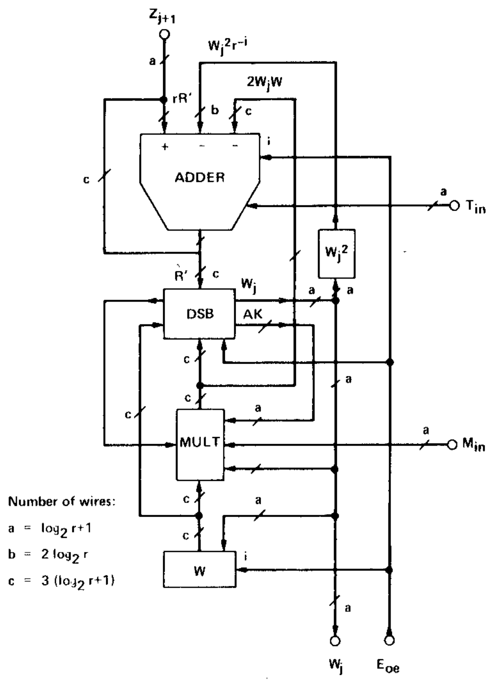

Fig. 5. Organization of the selection module (SM).

mentation can satisfy easily the main LSI constraints: high gate to pin ratio and regularity in the layout. A complete unit is obtainable by cascading the desired number of basic modules. The recursion time is not affected by the number of modules due to the use of a redundant number representation system. In particular, the low input/output count reduces drastically the number of off-chip drivers that usually require large silicon area.

The unit that implements the algorithm (Fig. 4) consists of three basic modules as follows.

1) Exponent module, which calculates the exponent of the result Exp(w), and provides exponent odd/even signal (E_{oe}) to the other modules indicating the proper selection of i. This module also provides clock (Clk), synchronization (SYNC), and reset (Rst) signals to the other modules.

2) Selection module (SM) (Fig. 5) is $t + 1$ digit wide and it performs the algorithm on the $t + 1$ most significant digits. This module selects the result digit w_j which is passed to the subsequent modules.

3) The recursion module RM_i is shown in Fig. 6. Each one of these modules is identical in its structure and is d digits wide. It consists of the d-digits slice of the multi-input limited carry/borrow propagation adder [6] and multiplier which generates the slice of the product w_jw. It has four input/output one digit wide lines for transfer digit from/to the adder and multiplier T and M, respectively. The inputs to the module are w_j and E_{oe}.

All registers in the RM modules are of $d(\log_2 r + 1)$ bits. The multiplier block in RM module consists of $d \times 1$ digit multiplier slice with M_{in} and M_{out} digits transferred from the last significant module to the most significant module, respectively. The implementation of

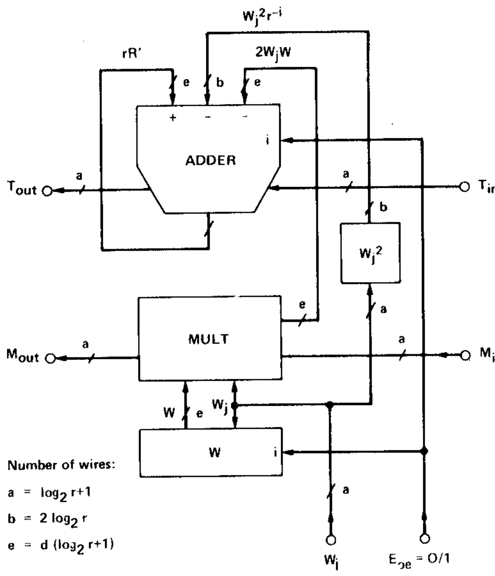

Fig. 6. Organization of the recursion module (RM).

the multiplier depends also on the time/complexity tradeoffs and could consist of the table look-up for the radix r of the moderate value. Concatenation is performed by selecting the appropriate input for the digit w_j to the redundant adder block. Multiplexing to the appropriate digit position is performed according to the exponent odd/even signal E_{oe} and step j, $i = i(E_{oe}, j)$.

The precision of the whole unit depends on the number of RM_i modules involved in computation. With n modules $2(n + t + 1)$ digits of the result could be accurately computed.

The difference between SM and RM modules is in digit select block (DSB), which does not exist in RM modules. Also, z_{j+1} input exists only in the SM module which does not have T_{out} and M_{out} lines. RM module has approximately $5\log_2 r + 9$, and SM module has $4\log_2 r + 8$ input/output lines.

The computation is performed according to the S algorithm. The multi-input redundant adder computes the partial reminder $R' = R + u$ which is compared to the selection constants $B'_k = B_k + u$ in the DSB block. The input digit z_{j+1} is fed to the select module only in the r^{-2} position of the redundant adder. Partial reminder R' is formed in two addition steps using CSA adder tree.

Comparison of the most significant $t + 1 = 3$ digits of the partial reminder $\hat{R}'$ to the 3 digits of the selection constants $\hat{B}'$ is performed in the DSB block of the select module SM. Selection and generation of selection constants $\hat{B}'$ is performed by table look-up and/or sucessive approximation depending on the value of the radix and time/hardware complexity tradeoffs.

V. Conclusion

An on-line floating-point algorithm for the computation of the square root is presented. The algorithm requires the use of a redundant number system in representing the result fraction, while for the exponent a conventional representation suffices. The algorithm has a minimal on-line delay of 1 and it is developed for a general radix r. The basic modules (RM and SM) are suitable for implementation in LSI/VLSI technology since they use only simple operators and very few external interconnections. For small radices, the selection becomes very simple and it can be incorporated in the RM module. The fractional unit is a one-dimensional cascade of RM modules with a few intermodule connections. For example, given d-digit wide RM modules and the fraction precision of n digits, the cascade consists of no more than $n/2$ RM modules. Due to the on-line nature of operation, higher even degree roots can be computed efficiently by cascading on-line square root units.

REFERENCES

[1] A. J. Atrubin, "A one-dimensional real-time iterative multiplier," *IEEE Trans. Electron. Comput.*, vol. EC-14, pp. 394–399, 1965.

[2] D. E. Atkins, "Introduction to the role of redundancy in computer arithmetic," *Computer*, vol. 8, pp. 74–76, June 1975.

[3] ——, "Design of the arithmetic units of Illiac III: Use of redundancy and higher radix methods," *IEEE Trans. Comput.*, vol. C-19, Aug. 1970.

[4] A. Avizienis, "On a flexible implementation of digital computer arithmetic," in *Proc. IFIP 1962*, pp. 664–668.

[5] ——, "Signed digit number representation for fast parallel arithmetic," *IRE Trans. Electron. Comput.*, vol. EC-10, pp. 389–400, 1961.

[6] R. T. Borovec, "The logical design of a class of limited carry-borrow propagation adders," M.S. thesis, Dep. Comput. Sci., Univ. of Illinois, Urbana, Rep. 275, Aug. 1968.

[7] M. D. Ercegovac, "A general hardware-oriented method for evaluation of functions and computations in a digital computer," *IEEE Trans. Comput.*, vol. C-26, pp. 667–680, July 1977.

[8] ——, "A general method for evaluation of functions and computations in a digital computer," Ph.D. dissertation, Dep. Comput. Sci., Univ. of Illinois, Urbana, Rep. 750, June 1975.

[9] ——, "An on-line square rooting algorithm," presented at 4th IEEE Symp. Comput. Arithmetic, Santa Monica, CA, Oct. 1978.

[10] I. T. Ho and T. C. Chen, "Multiple addition by residue threshold functions and their representation by array logic," *IEEE Trans. Comput.*, vol. C-22, pp. 762–764, Aug. 1973.

[11] M. J. Irwin, "An arithmetic unit for on-line computation," Ph.D. dissertation, Dep. Comput. Sci., Univ. of Illinois, Urbana, Rep. 873, 1977.

[12] G. Metze, "Minimal square rooting," *IEEE Trans. Electron. Comput.*, vol. EC-14, Apr. 1965.

[13] V. G. Oklobdzija, "An on-line higher radix square rooting algorithm," M.Sc. thesis, Univ. of California, Los Angeles, June 1978.

[14] J. E. Robertson, "A new class of digital division methods," *IRE Trans. Electron. Comput.*, vol. EC-7, pp. 218–222, Sept. 1958.

[15] K. S. Trivedi and M. D. Ercegovac, "On-line algorithms for division and multiplication," *IEEE Trans. Comput.*, vol. C-26, pp. 681–687, July 1977.

[16] A. Avizienis, "Arithmetic error codes: Cost and effectiveness studies for application in digital system design," *IEEE Trans. Comput.*, vol. C-20, pp. 1322–1331, 1971.

[17] A. Gorji-Sinaki, "Error-coded algorithms for on-line arithmetic," Ph.D. dissertation, Dep. Comput. Sci., Univ. of California, Los Angeles, Rep. CSD-810303, Feb. 1981.