システムＬＳＩ設計技術　94－3
（2000. 1. 11）

減算シフト型立方根計算回路

高木 直史　南 利明

名古屋大学大学院工学研究科情報工学専攻
〒464-8603 名古屋市千種区不老町
電話：052–789–3312, email: ntakagi@nuie.nagoya-u.ac.jp

あらまし 3次方程式や4次方程式の求解に用いられる立方根計算のための減算シフト型のハードウェアアルゴリズムを提案し、これに基づく立方根計算回路の構成を示す。立方根計算では、筆算による平方根計算と同様の剰余に関する漸化式を導くと、その中に立方根の中間結果の平方計算が含まれる。提案するアルゴリズムでは、中間結果の平方を保持し、剰余および中間結果の平方の更新を加減算とシフトのみで行う。立方根の基数や桁集合、剰余の表現法、桁選択関数等により、さまざまなバージョンを設計できる。これらはいずれも、順序回路あるいは組合せ回路として実現できる。回路は規則正しい配列構造をもち、ＶＬＳＩ実現に適している。本稿では、基数2のアルゴリズムの詳細を設計し、その順序回路実現について述べる。

キーワード 算術演算回路、立方根計算、ハードウェアアルゴリズム、減算シフト型アルゴリズム、ＶＬＳＩ

A subtract-and-shift cube rooting circuit

Naofumi Takagi and Toshiaki Minami

Department of Information Engineering, Nagoya University
Furo-cho, Chikusa-ku, Nagoya 464-8603, Japan
phone: +81–52–789–3312, email: ntakagi@nuie.nagoya-u.ac.jp

Abstract A hardware algorithm for cube rooting which is used for solving algebraic equations of third or fourth degree is proposed, and a design of a cube rooting circuit based on the algorithm is shown. In cube rooting, the digit recurrence equation on the residual includes the square of the partial result of the cube root. In the proposed algorithm, the square of the partial result is kept, and is updated by addition/subtractions and shifts as the residual. Different specific versions of the algorithm are possible, depending on the radix and the digit set of the cube root, the type of representation of the residual and the square of the partial result, and the digit selection function. Any version of the algorithm can be implemented as a sequential (folded) circuit or a combinational (unfolded) circuit. The implementations have a regular cellular array structure suitable for VLSI realization. In this report, details of a radix-2 version of the algorithm and its sequential implementation are shown.

key words Computer arithmetic, cube rooting, hardware algorithm, digit recurrence algorithm, VLSI.

1 Introduction

Advances of VLSI technologies make it attractive to accelerate important complex operations by special hardware. It is also nice to avoid intermediate rounding errors between atomic floating point operations, but only have a bounded error for the final result of a complex operation. In this report, we propose a digit recurrence algorithm for cube rooting which is used for solving algebraic equations of third or fourth degree.

For cube rooting, as square rooting, we can derive digit recurrence equations on the residual and on the partial result. However, the recurrence equation on the residual includes the square of the partial result of the cube root, and therefore, a naive calculation of the j-th residual requires a j-digit square calculation. In the proposed algorithm, we keep the square of the partial result, as well as the partial result itself, and derive a recurrence equation on it, so that we can calculate it, as well as the residual, by addition/subtractions and shifts. We select each cube root digit from a redundant digit set by estimating the residual and the partial result. We perform addition/subtractions appearing in the calculation of the recurrence equations on the residual and on the square of the partial result without carry/borrow propagation by representing them in a redundant representation such as carry-save form or signed-digit representation. We use the on-the-fly conversion algorithm [1] for calculating the recurrence equation on the partial result.

Different specific versions of the algorithm are possible, depending on the radix of the cube root, the redundancy factor of the cube root digit set, the type of representation of the residual and the square of the partial result, and the cube root digit selection function, as digit recurrence algorithms for division or square rooting [2]. Any version of the algorithm can be implemented as a sequential (folded) circuit or a combinational (unfolded) circuit. Pipelining can also be used. The implementations have a regular array structure suitable for VLSI realization.

In this paper, we first define the computation of the cube root. Then, we show a general algorithm for cube rooting in Section 3. We show a radix-2 version of the algorithm in Section 4, and consider its sequential implementation in Section 5.

2 Cube rooting

We consider computation of the cube root of the mantissa part of a floating-point number $F_X = (-1)^{S_X} \cdot 2^{E_X} \cdot M_X$ where S_X ($\in \{0,1\}$), E_X, and M_X ($\frac{1}{2} \leq M_X < 1$) are the sign bit, the exponent part, and the mantissa part of F_X, respectively. The cube root of F_X is $F_C = (-1)^{S_X} \cdot 2^{\frac{E_X+k}{3}} \cdot (M_X \cdot 2^{-k})^{\frac{1}{3}}$ where $k = E_X \bmod 3 \in \{0,1,2\}$.

Therefore, in this report, we consider the computation of $C = X^{\frac{1}{3}}$, where $2^{-3} \leq X < 1$. $\frac{1}{2} \leq C < 1$ holds. We assume X is represented as an n-digit r-ary fraction where $r = 2^b$. Namely, $X = \sum_{i=1}^{n} x_i r^{-i}$. We intend to compute the result C in n-digit precision, i.e.,

$$-r^{-n} \leq X^{\frac{1}{3}} - C < r^{-n}. \tag{1}$$

3 General algorithm

3.1 Recurrence

Here, we derive a general digit-recurrence equations for radix r, where $r = 2^b$.

As digit-recurrence algorithms for division or square rooting [2], the cube root digit q_j is obtained step by step. Let $C[j]$ be the partial result after j iterations. Then, $C[j] = C[0] + \sum_{i=1}^{j} q_i r^{-i}$, where $C[0]$ is the initial value of the partial result. The recurrence equation on the partial result is

$$C[j+1] \quad := \quad C[j] + q_{j+1} r^{-j-1}. \tag{2}$$

We select the cube root digit q_{j+1} from a redundant digit set $\{-a, \cdots, -1, 0, 1, \cdots, a\}$, where $\frac{r}{2} \leq a < r$. The final result is $C = C[n] = \sum_{i=1}^{n} q_i r^{-i}$.

We define a residual (or scaled partial remainder) $W[j]$ as

$$W[j] = r^j (X - C[j]^3). \tag{3}$$

Subtracting r times (3) from the equation for $W[j+1]$, we get the recurrence equation on the residual as

$$W[j+1] := rW[j] - 3C[j]^2 q_{j+1} - 3C[j] q_{j+1}^2 r^{-j-1} - q_{j+1}^3 r^{-2j-2}.$$

Since this equation includes the term $-3C[j]^2 q_{j+1}$, we need squaring of j digit number $C[j]$ for the calculation. To avoid the squaring, we keep $C[j]^2$ and update it by addition/subtraction and shift.

Let $C[j]^2$ be $S[j]$. Then, the recurrence equation on $W[j]$ is rewritten as

$$W[j+1] :=$$

$$rW[j] - 3S[j]q_{j+1} - 3C[j]q_{j+1}^2 r^{-j-1} - q_{j+1}^3 r^{-2j-2}. \quad (4)$$

The recurrence equation on $S[j]$ is

$$S[j+1] := S[j] + q_{j+1} r^{-j-1}(2C[j] + q_{j+1} r^{-j-1}). \quad (5)$$

From (1), $(C - r^{-n})^3 \leq X < (C + r^{-n})^3$ must hold. Since $C = C[j] + \sum_{i=j+1}^{n} q_i r^{-i}$ and the minimum and the maximum cube root digit values are $-a$ and a, respectively, from (3), we get the bounds for $W[j]$ as

$$\begin{aligned} -3C[j]^2\rho + 3C[j]\rho^2 r^{-j} - \rho^3 r^{-2j} &\leq W[j] \\ &< 3C[j]^2\rho + 3C[j]\rho^2 r^{-j} + \rho^3 r^{-2j}, \end{aligned}$$

where $\rho = \frac{a}{r-1}$ is the redundancy factor of the cube root digit set.

Now, we consider the initial values of the residual and the partial result. Initially,

$$\begin{aligned} -3C[0]^2\rho + 3C[0]\rho^2 - \rho^3 &\leq W[j] \\ < \quad 3C[0]^2\rho + 3C[0]\rho^2 + \rho^3 & \end{aligned}$$

must hold. Since $2^{-3} \leq X < 1$ and $\frac{1}{2} < \rho \leq 1$, we can satisfy these bounds by letting

$$\begin{aligned} C[0] &= 1, \\ W[0] &= X - 1. \end{aligned}$$

When $\rho = 1$, we can also satisfy the bounds by letting

$$\begin{aligned} C[0] &= 0, \\ W[0] &= X. \end{aligned}$$

The algorithm for cube rooting consists in performing n iterations of calculation of the recurrence equations (2), (4) and (5). In each iteration, we first produce the shifted residual $rW[j]$, and then, select the cube root digit q_{j+1} by examining the shifted residual and the partial result $C[j]$. We will discuss on this later. Finally, we perform the calculation of the recurrence equations.

The general algorithm is summarized as follows:

Algorithm [CBRT]

Step 1:

$C[0] := 1;$

$S[0] := 1;$

$W[0] := X - 1;$

Step 2:

for $j := 0$ to $n - 1$ do

{

Select q_{j+1} from $\{-a, \cdots, -1, 0, 1, \cdots, a\}$;

$C[j+1] := C[j] + q_{j+1} r^{-j-1};$

$S[j+1] := S[j] + q_{j+1} r^{-j-1}(2C[j] + q_{j+1} r^{-j-1});$

$W[j+1] := rW[j] - 3S[j]q_{j+1}$

$\quad - 3C[j]q_{j+1}^2 r^{-j-1} - q_{j+1}^3 r^{-2j-2};$

} □

When $\rho = 1$, we can replace Step 1 by $C[0] := 0$, $S[0] := 0$, and $W[0] := X$. We will discuss on selection of the cube root digit q_{j+1} in the next subsection.

We can increase the speed of the implementation with a small increase in hardware complexity by performing the addition/subtractions in the recurrence equations without carry/borrow propagation by the use of a redundant representation. Therefore, in this report, we concentrate on this type of implementations. Namely, we represent the residual $W[j]$ and the square of the partial result $S[j]$ in a redundant representation, such as the carry-save form or the (binary) signed-digit representation, and perform the addition/subtractions without carry/borrow propagation.

Since $-3 < W[j] < 3$, we can represent $W[j]$ by either a carry-save form with 3-bit integer part (including the sign bit) or a binary signed-digit representation with 3-bit integer part. Since $0 \leq S[j] \leq 1$, we can represent $S[j]$ by either a carry-save form with 1-bit integer part (which is the sign bit) or a binary signed-digit representation with 1-bit integer part.

Although we may represent the partial result $C[j]$ in a redundant representation as well, we keep the non-redundant representation of it by the on-the-fly conversion [1].

3.2 Cube root digit selection

We have to select cube root digit q_{j+1} from $\{-a, \cdots, -1, 0, 1, \cdots, a\}$ so that the bounds for $W[j+1]$, i.e.,

$$\begin{aligned} -3C[j+1]^2\rho + 3C[j+1]\rho^2 r^{-j-1} - \rho^3 r^{-2j-2} & \\ \leq \quad W[j+1] & \\ < \quad 3C[j+1]^2\rho + 3C[j+1]\rho^2 r^{-j-1} + \rho^3 r^{-2j-2}, & \end{aligned}$$

are satisfied.

Let the interval of $rW[j]$ where k can be selected as q_{j+1} be $[L_k[j], U_k[j])$. Then,

$$\begin{aligned} L_k[j] = 3C[j]^2(k-\rho) + 3C[j](k-\rho)^2 r^{-j-1} & \\ +(k-\rho)^3 r^{-2j-2}, & \\ U_k[j] = 3C[j]^2(k+\rho) + 3C[j](k+\rho)^2 r^{-j-1} & \\ +(k+\rho)^3 r^{-2j-2}. & \end{aligned}$$

Note that the lower bound of the interval for $k = -a$ and the upper bound of the interval for $k = a$ are equal to the lower bound and the upper bound of $rW[j]$, respectively.

The continuity condition of adjacent intervals $U_{k-1}[j] \geq L_k[j]$ yields

$$(2\rho - 1)(3C[j]^2 + 3C[j](2k-1)r^{-j-1} + (3k^2 - 3k + \rho^2 - \rho + 1)r^{-2j-2}) \geq 0. \quad (6)$$

The left-hand side of (6), i.e., $(2\rho-1)\,(3C[j]^2+3C[j](2k-1)r^{-j-1}+(3k^2-3k+\rho^2-\rho+1)r^{-2j-2})$, indicates the overlap between consecutive selection intervals. This overlap is used to simplify the selection function.

q_{j+1} depends on $rW[j]$ and $C[j]$. Using the overlap, we can select q_{j+1} by estimates of them. Let the digit selection function be $Select(r\hat{W}[j], \hat{C}[j])$ where $r\hat{W}[j]$ and $\hat{C}[j]$ are estimates of $rW[j]$ and $C[j]$, respectively. Then, the function is described by a set of selection constants, $\{m_k(\hat{C}[j]) | k \in \{-a+1, \cdots, -1, 0, 1, \cdots, a\}\}$, where k is selected as q_{j+1} if $m_k(\hat{C}[j]) \leq r\hat{W}[j] < m_{k+1}(\hat{C}[j])$.

We obtain $r\hat{W}[j]$ by truncating $rW[j]$, which is in a redundant representation, to t fractional bits. (Note that not r-ary digits but bits.) We obtain $\hat{C}[j]$ by truncating $C[j]$ to d fractional bits. (Note again that not r-ary digits but bits.) Since $C[j]$ is in the non-redundant representation, $\hat{C}[j] = C[j]$ for the small j's such that $r^{-j} \geq 2^{-d}$ and $\hat{C}[j] \leq C[j] \leq \hat{C}[j] + 2^{-d} - r^{-j}$ for the other j's.

When $W[j]$ is in the carry-save form, $r\hat{W}[j] \leq rW[j] < r\hat{W}[j] + 2^{-t+1}$. Therefore,

$$m_k(\hat{C}[j]) \geq \max_{\hat{C}[j]}(L_k[j]), \quad (7)$$

$$(m_k(\hat{C}[j]) - 2^{-t}) + 2^{-t+1} \leq \min_{\hat{C}[j]}(U_{k-1}[j]) \quad (8)$$

must be satisfied. Namely, $m_k(\hat{C}[j])$ must be a multiple of 2^{-t} that satisfies (7) and (8). Here, $\max_{\hat{C}[j]}(L_k[j])$ denotes the maximum value of the lower bound of the interval of $rW[j]$ where k can be selected as q_{j+1} when the estimate of $C[j]$ is $\hat{C}[j]$. $\min_{\hat{C}[j]}(U_{k-1}[j])$ denotes the minimum value of the upper bound of the interval of $rW[j]$ where $k-1$ can be selected as q_{j+1} when the estimate of $C[j]$ is $\hat{C}[j]$. Note that the maximum value of $r\hat{W}[j]$ for which $k-1$ is selected as q_{j+1} is $m_k(\hat{C}[j]) - 2^{-t}$.

When $W[j]$ is in the binary signed-digit representation, $r\hat{W}[j] - 2^{-t} < rW[j] < r\hat{W}[j] + 2^{-t}$, and hence,

$$m_k(\hat{C}[j]) - 2^{-t} \geq \max_{\hat{C}[j]}(L_k[j]),$$

$$(m_k(\hat{C}[j]) - 2^{-t}) + 2^{-t} \leq \min_{\hat{C}[j]}(U_{k-1}[j])$$

must be satisfied.

In both cases, the minimum overlap required for a feasible digit selection is

$$\min_{\hat{C}[j]}(U_{k-1}[j]) - \max_{\hat{C}[j]}(L_k[j]) \geq 2^{-t}.$$

Since $\max_{\hat{C}[j]}(L_k[j])$ and $\min_{\hat{C}[j]}(U_{k-1}[j])$ depend on j, a different selection function might result for different j. When $r = 2$, $a = 1$ and $\rho = 1$, a single selection function for all j exists. When $r \geq 4$, no single selection function for all j exists, and therefore, we should find J so that a single selection function can be used for $j \geq J$ and consider the cases for $j < J$ separately.

3.3 Representation of the partial result

We intend to have the ordinary non-redundant r-ary representation (with digit set $\{0, 1, \cdots, r-1\}$) of $C[j]$ by the on-the-fly conversion [1]. Since q_{j+1} is selected from $\{-a, \cdots, -1, 0, 1, \cdots, a\}$, there are two possible carry values, i.e., 0 and -1, into the j-th place of $C[j+1]$. We keep the non-redundant representations of $C[j]$ and $C[j]-r^{-j}$. Let the non-redundant representations of $C[j]$ and $C[j]-r^{-j}$ be $C[j]^+$ and $C[j]^-$, respectively.

Initially, when $C[0] = 1$, $C[0]^+ := 1$ and $C[0]^- := 0$. When $C[0] = 0$, $C[0]^+ := 0$ and $C[0]^- := -1$.

We can obtain $C[j+1]^+$ and $C[j+1]^-$ by selecting $C[j]^+$ or $C[j]^-$ and concatenating the $j+1$-th digit to the selected one, respectively. When $q_{j+1} < 0$, $C[j+1]^+ := C[j]^- + (q_{j+1} + r)r^{-j-1}$ and $C[j+1]^- := C[j]^- + (q_{j+1} + r - 1)r^{-j-1}$. When $q_{j+1} = 0$, $C[j+1]^+ := C[j]^+$ and $C[j+1]^- := C[j]^- + (r-1)r^{-j-1}$. When $q_{j+1} > 0$, $C[j+1]^+ := C[j]^+ + q_{j+1}r^{-j-1}$ and $C[j+1]^- := C[j]^+ + (q_{j+1} - 1)r^{-j-1}$.

For the calculation of the square of the partial result (recall (5)), we have to produce $q_{j+1}r^{-j-1}(2C[j] + q_{j+1}r^{-j-1})$ as an adder input. We can make the calculation simpler by producing the adder input as a non-redundant number from $C[j]^+$ and $C[j]^-$.

4 A radix-2 version

Different specific versions of the algorithm are possible, depending on the radix and the digit set of the cube root, the type of representation of the residual and the square of the partial result (carry-save or signed-digit), and the digit selection function. In this section, we show details of a radix-2 version of the algorithm.

Here, we consider the case that the radix of the cube root r is 2, the cube root digit set is $\{-1, 0, 1\}$, i.e., $a = 1$

and the redundancy factor $\rho = 1$, and the residual $W[j]$ and the square of the partial result are represented in the carry-save form. We represent $W[j]$ as a carry-save form with 3-bit integer part (including the sign bit), and $S[j]$ by one with 1-bit integer part (which is the sign bit).

The recurrence equations are

$$\begin{aligned} C[j+1] &:= C[j] + 2^{-j-1}q_{j+1}, \\ S[j+1] &:= S[j] + q_{j+1}2^{-j-1}(2C[j] + q_{j+1}2^{-j-1}), \end{aligned} \tag{9}$$

$$W[j+1] := 2W[j] - 3S[j]q_{j+1} - 3C[j]q_{j+1}^2 2^{-j-1} - q_{j+1}^3 2^{-2j-2}. \tag{10}$$

Since $\rho = 1$, initially, we can let

$$\begin{aligned} C[0] &:= 0, \\ S[0] &:= 0, \\ W[0] &:= X. \end{aligned}$$

Now, we determine the cube root digit selection function. From (7) and (8), $m_k[j]$ must satisfy

$$\max_{\hat{C}[j]}(L_k[j]) \le m_k[j] \le \min_{\hat{C}[j]}(U_{k-1}[j]) - 2^{-t}.$$

To obtain the function, we get the values of $\max_{\hat{C}[j]}(L_k[j])$ and $\min_{\hat{C}[j]}(U_{k-1}[j])$.

$$\begin{aligned} \max_{\hat{C}[j]}(L_0[j]) &= -3\hat{C}[j]^2 + 3\hat{C}[j]2^{-j-1} - 2^{-2j-2}, \\ \min_{\hat{C}[j]}(U_{-1}[j]) &= 0, \\ \max_{\hat{C}[j]}(L_1[j]) &= 0, \\ \min_{\hat{C}[j]}(U_0[j]) &= 3\hat{C}[j]^2 + 3\hat{C}[j]2^{-j-1} + 2^{-2j-2}. \end{aligned}$$

$\min(U_{-1}[j]) = \max(L_1[j]) = 0$ independent of j and $C[j]$.

When $j = 0$, since $\hat{C}[0] = C[0] = 0$, $\max(L_0[0]) = -2^{-2}$ and $\min(U_0[0]) = 2^{-2}$. Since $2W[0] = 2X \ge 2^{-2}$, q_1 must be 1. Therefore, $m_0[0]$ can be any number $\le -2^{-t}$ and $m_1[0]$ must satisfy $0 \le m_1[0] \le \max(0, 2^{-2} - 2^{-t})$. Since $q_1 = 1$, $W[1] = 2X - 2^{-2} \ge 0$.

When $j = 1$, since $\hat{C}[0] = C[0] = 2^{-1}$, $\max(L_0[1]) = -7 \cdot 2^{-4}$ and $\min(U_0[1]) = 19 \cdot 2^{-4}$. Since $2W[1] \ge 0$, q_2 must be 0 or 1. Therefore, $m_0[1]$ can be any number $\le -2^{-t}$ and $m_1[1]$ must satisfy $0 \le m_1[1] \le 19 \cdot 2^{-4} - 2^{-t}$.

From the fact that $W[j+1] = 2W[j]$ when $q_{j+1} = 0$, for $j \ge 1$, $C[j] \ge 2^{-1}$. Hence, for $j \ge 2$, $\max(L_0[j]) < 9 \cdot -2^{-4}$ and $\min(U_0[j]) > 3 \cdot 2^{-2}$. Therefore, for $j \ge 2$, $m_0[j]$ must satisfy $9 \cdot -2^{-4} \le m_0[j] \le -2^{-t}$ and $m_1[j]$ must satisfy $0 \le m_1[j] \le 3 \cdot 2^{-2} - 2^{-t}$.

Hence, by letting $t = 1$, we can obtain a single digit selection function which is independent of j and $C[j]$ as $\{m_0 = -2^{-1}, m_1 = 0\}$.

This radix-2 version of the algorithm is summarized as follows:

Algorithm [CBRT_R2]

Step 1:

$C[0]^+ := 0$;

$C[0]^- := -1$;

$S[0] := 0$;

$W[0] := X$;

Step 2:

for $j := 0$ to $n-1$ do

{

$$q_{j+1} := \begin{cases} -1 & if \quad 2\hat{W}[j] \le -1 \\ 0 & if \quad 2\hat{W}[j] = -\frac{1}{2} \\ 1 & if \quad 2\hat{W}[j] \ge 0 \end{cases};$$

($2\hat{W}[j]$: truncation of $2W[j]$ to 1 fractional bits.)

Obtain $C[j+1]^+$ and $C[j+1]^-$;

(By on-the-fly conversion.)

$S[j+1] := S[j] + q_{j+1}2^{-j-1}(2C[j] + q_{j+1}2^{-j-1})$;

(Carry-save addition.)

$W[j+1] := 2W[j] - 3S[j]q_{j+1} - 3C[j]q_{j+1}^2 2^{-j-1} - q_{j+1}^3 2^{-2j-2}$;

(Carry-save additions.)

} □

$C[n]^+$ is the final result C.

Fig. 1 shows an example of cube rooting according to [CBRT_R2]. In the figure, $\bar{1}$ denotes -1.

5 A radix-2 cube rooting circuit

Any version of the algorithm can be implemented as a sequential (folded) circuit or a combinational (unfolded) circuit. Here, we consider implementation of the radix-2 version of the algorithm shown in the previous section as a sequential circuit which performs one iteration of Step 2 in each clock cycle.

The circuit consists of a combinational circuit part and registers. We need registers REG-WC and REG-WS, REG-CP, REG-CM, and REG-SC and REG-SS, for storing $W[j]$, $C[j]^+$, $C[j]^-$, and $S[j]$, respectively. Since $W[j]$ and $S[j]$ are in the carry-save form, we need two registers for storing each of them.

To remove j-bit shift of $C[j]$ from the calculation of the residual $W[j+1]$ (equation (10)) and also from the

calculation of the square of the partial result $S[j+1]$ (equation (9)), we keep $2^{-j}C[j]^+$ and $2^{-j}C[j]^-$ instead of $C[j]^+$ and $C[j]^-$. To indicate the position of $2j$-th place, we prepare a shift register SREG-J whose content is 2^{-j} and is shifted one place to the right in each cycle.

The combinational circuit part consists of the following blocks.

1. A cube root digit selector DS for selecting the cube root digit q_{j+1} by examining significant bits of $rW[j]$.
2. An on-the-fly converter OTFC for the on-the-fly conversion to obtain $C[j+1]^+$ and $C[j+1]^-$, which mainly consists of selectors.
3. An adder input generator AIG-S for generating the adder input $q_{j+1}2^{-j-1}(2C[j] + q_{j+1}2^{-j-1})$ in the calculation of the square of the partial result from $C[j]^+$ and $C[j]^-$, which mainly consists of a selector.
4. A carry save adder CSA-S for adding $S[j]$ and the adder input produced by AIG-S and producing $S[j+1]$.
5. An adder input generator AIG-W for generating the adder input $-q_{j+1}(3(S[j] + C[j]q_{j+1}2^{-j-1}) + q_{j+1}^2 2^{-2j-2})$ in the calculation of the residual, which mainly consists of carry-save adders.
6. Carry save adders CSA-W for adding $2W[j]$ and the adder input produced by AIG-W and producing $W[j+1]$.

Fig. 2 shows a block diagram of the circuit. REG-CP, REG-CM and OTFC should be of $2n$-bit length. The other registers and blocks should be of $n + \log n + m$-bit length where m is a small constant.

The length of the clock cycle (the logical depth of the combinational circuit part) is a small constant independent of n. The amount of hardware is proportional to n. The circuit has a regular linear cellular array structure with a bit-slice feature suitable for VLSI realization. We can perform n-bit cube rooting in $n+1$ clock cycles.

Of course, we can construct a sequential circuit which performs more than one iterations of Step 2 per clock cycle, or further, a combinational circuit.

6 Concluding remarks

We have proposed a hardware algorithm for cube rooting and shown details of a radix-2 version of the algorithm and its sequential implementation.

In the algorithm, in order to remove a squaring from the residue calculation, we keep the square of the partial result and update it by addition/subtractions and shifts as the residual. Different specific versions of the algorithm are possible, depending on the radix and the digit set of the cube root, the type of representation of the residual and the square of the partial result, and the digit selection function. Any version of the algorithm can be implemented as a sequential (folded) circuit or a combinational (unfolded) circuit. The implementations have a regular cellular array structure suitable for VLSI realization.

We are now designing a radix-2 cube rooting circuit LSI.

References

[1] M. D. Ercegovac and T. Lang: 'On-the-fly conversion of redundant into conventional representations,' IEEE Trans. Comput., vol. C-36, no. 7, pp. 895-897, July 1987.

[2] M. D. Ercegovac and T. Lang: *Division and Square Root – Digit-Recurrence Algorithms and Implementations*, Kluwer Academic Publishers, 1994.

$X = [.010110]$

j	$C[j]$		$S[j]$	q_{j+1}		$W[j]$	
0	0.		0.000000000000			000.010110000000	
	$\bar{1}$.		0.000000000000			000.000000000000	
						0000.10110000000	$2W$
				1		0000.00000000000	
		+	0.010000000000		+	1111.101111111111(+1)	
					+	1111.111111111111(+1)	
1	0.1		0.010000000000			001.100100000010	
	0.0		0.000000000000			110.110111111110	
						0011.00100000010	$2W$
				1		1101.10111111110	
		+	0.010100000000		+	1111.110011111111(+1)	
					+	1110.111111111111(+1)	
2	0.11		0.000100000000			000.111100000010	
	0.10		0.100000000000			110.101111111110	
						0001.11100000010	$2W$
				$\bar{1}$		1101.01111111110	
		+	1.110100111111(+1)		+	1100.010001111000	
					+	0101.001100001000	
3	0.101		0.010000111110			111.001001110000	
	0.100		0.001000000010			001.101100010000	
						1110.01001110000	$2W$
				1		0011.01100010000	
		+	0.000101010000		+	1111.110010111111(+1)	
					+	1110.111010001111(+1)	
4	0.1011		0.011101101100			010.100110110010	
	0.1010		0.000000100100			101.110010011110	
						0101.00110110010	$2W$
				1		1011.10010011110	
		+	0.000010110100		+	1111.101110000011(+1)	
					+	1110.110010111111(+1)	
5	0.10111		0.011111111100			000.101100101110	
	0.10110		0.000001001000			110.100110110110	
						0001.01100101110	$2W$
				$\bar{1}$		1101.00110110110	
6	0.101101						

$C = [.101101]$

Figure 1: An example of cube rooting according to [CBRT_R2]

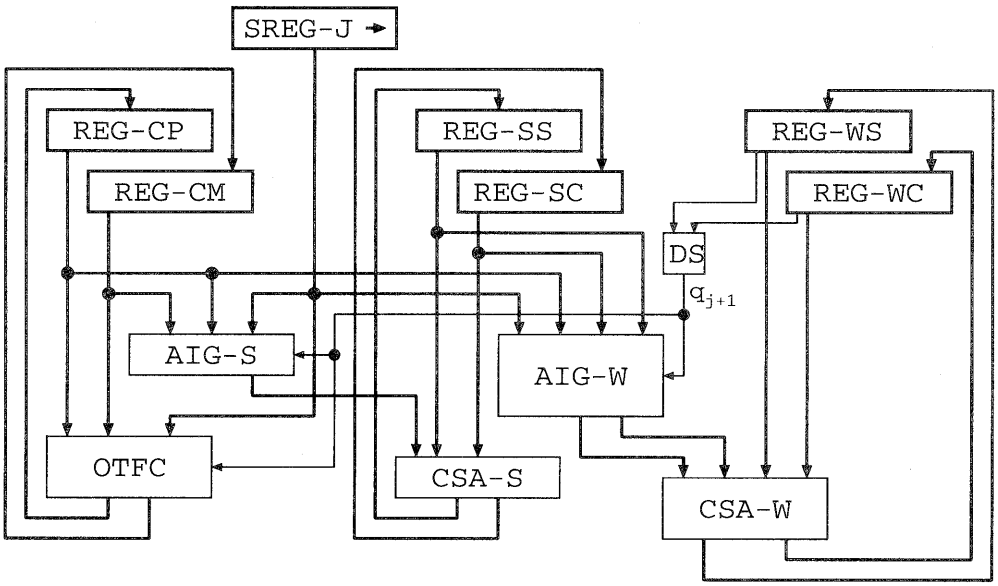

Figure 2: Block diagram of a sequential circuit based on [CBRT_R2]