Real/Complex Reconfigurable Arithmetic Using Redundant Complex Number Systems

Takafumi AOKI, Hiroaki AMADA, and Tatsuo HIGUCHI
Department of System Information Sciences
Graduate School of Information Sciences, Tohoku University
Aoba-ku, Sendai 980-77, Japan

Abstract

This paper presents a hardware algorithm for a real/complex reconfigurable arithmetic unit, which can change its structure for three different arithmetic modes in real time. The three modes realize (i) a single-precision complex-number multiplication, (ii) a double-precision real-number multiplication, and (iii) a pair of single-precision real-number four-operand multiply-add operations, respectively. We discuss the reconfiguration of hardware structure on the basis of the transformation of the number system used in each arithmetic mode. The designed arithmetic unit can perform high-speed real/complex arithmetic computations based on binary-tree addition scheme, and also exhibits highly regular structure suited for VLSI implementation.

1. Introduction

Complex arithmetic computations are of major importance for various signal processing and scientific computation algorithms including complex orthogonal transformations, convolutions, correlations and filtering [1]. These applications require efficient representation and manipulation of complex numbers. In the usual representation scheme, the real part and the imaginary part of the given number are treated separately in arithmetic operations. For example, the most natural way of complex multiplication requires four real multiplications and two real additions. At the recent conference [2], a high-speed complex-number multiplier has presented. The reported method involves three additions to generate pre-multiplication sums, three multiplications and two additions for final results.

To simplify the manipulation of complex numbers, several authors have proposed *complex number systems* [3]. Recently, a new complex-number representation called the *polygonal representation* has been proposed [4]. In spite of its theoretical elegance, however, high-speed circuit implementations that can be used in practical VLSI signal processors seem difficult due to the circuit complexity required for implementing elementary arithmetic operations.

In this paper, we present an efficient hardware algorithm for a real/complex reconfigurable arithmetic unit that can be installed into VLSI signal processors. The proposed algorithm employs *Redundant Complex Number Systems* (RCNSs), a class of complex number representations proposed by the authors [5]. The idea of this number representation is originally based on the concept of "complex radix" in Knuth's *quarter-imaginary number system* [6]. An RCNS is a radix-(rj) system with digits in $\{-\alpha, \cdots, 0, \cdots, \alpha\}$, where $r \geq 2$ and $\lceil r^2/2 \rceil \leq \alpha \leq r^2 - 1$. The complex radix rj allows unified complex number representation without treating real and imaginary parts separately. Also, redundancy in number representation allows the carry-free addition and the binary-tree multiple-operand addition, as in Avizienis' Signed-Digit (SD) number systems [7]. In the case of RCNS with $r = 2$ and $\alpha = 3$, conversion to and from standard binary number representation can be easily performed. The complex-number multiplier based on this system exhibits highly regular structure as is observed in real-number multipliers, which suggests an efficient design of real/complex reconfigurable arithmetic units.

In this context, we discuss the design of a reconfigurable arithmetic unit that can change its structure for three different arithmetic operation modes in real time. The three modes realize (i) a single-precision complex-number multiplication, (ii) a double-precision real-number multiplication, and (iii) a pair of single-precision real-number four-operand multiply-add operations, respectively. We propose a new design methodology for this type of reconfigurable units, in which the reconfiguration of arithmetic circuits is discussed in terms of algebraic transformation of the number system used in each operation modes. The proposed real/complex reconfigurable arithmetic unit will be useful in many applications which require both real-number and complex-number computations.

This paper is organized as follows: Section 2 briefly describes the basic properties of RCNSs and their elementary arithmetic algorithms. Section 3 introduces a unified notation for discussing hardware reconfiguration. In Section 4, we describe the algorithm and architecture of our real/complex reconfigurable arithmetic unit. The final section summarizes our design.

2. Redundant Complex Arithmetic

2.1. Redundant Complex Number Systems

A redundant complex number system (RCNS) is defined as a positional number system that has a complex radix rj, where r is an integer that is not less than 2 and j denotes the imaginary unit. Each digit of a redundant complex number can assume the following $2\alpha+1$ values

$$\{\overline{\alpha},\cdots,\overline{1},0,1,\cdots,\alpha\}, \tag{1}$$

where $\overline{\alpha}=-\alpha$, and the maximum digit magnitude α must be within the following range:

$$\left\lceil \frac{r^2}{2} \right\rceil \leq \alpha \leq r^2-1. \tag{2}$$

The notation $\lceil x \rceil$ refers to the least integer that is not less than the real number x. An RCNS with radix rj and digit set $\{\overline{\alpha},\cdots,0,\cdots,\alpha\}$ is denoted simply by RCNS rj,α. In the following, we focus on RCNS $2j,3$ because of its compatibility with binary number systems. In this case, each digit can take the seven values, $\{\overline{3},\overline{2},\overline{1},0,1,2,3\}$. The algebraic value of a number $X=(X_{k-1}\cdots X_0.X_{-1}\cdots X_{-l})$ in RCNS $2j,3$ notation can be evaluated as

$$X=\sum_{i=-l}^{k-1} X_i(2j)^i. \tag{3}$$

The right-hand side of (3) can be decomposed into real and imaginary parts as

$$X=\sum_{i=-\lfloor l/2 \rfloor}^{\lfloor (k-1)/2 \rfloor} X_{2i}(-4)^i+2j\left\{\sum_{i=-\lceil l/2 \rceil}^{\lceil (k-1)/2 \rceil -1} X_{2i+1}(-4)^i\right\}. \tag{4}$$

The notation $\lfloor x \rfloor$ refers to the largest integer that is not greater than the real number x. Thus, if we decompose the radix-$(2j)$ redundant complex number representation into the real vector and the imaginary vector, each representation can be regarded as a special Signed-Digit (SD) number representation that has the negative radix -4. This feature enables us to design high-speed complex arithmetic circuits with highly regular structure.

2.2. RCNS $2j$,3 Addition

The addition of two numbers, $X=(X_{k-1}\cdots X_i\cdots X_{-l})$ and $Y=(Y_{k-1}\cdots Y_i\cdots Y_{-l})$ in RCNS $2j,3$, where $X_i,Y_i\in\{\overline{3},\cdots,0,\cdots,3\}$, is performed by the following three steps for each digit:

$$\begin{aligned} &\text{Step 1}: && Z_i = X_i+Y_i, && (5)\\ &\text{Step 2}: && -4C_i+W_i=Z_i, && (6)\\ &\text{Step 3}: && S_i=W_i+C_{i-2}, && (7)\end{aligned}$$

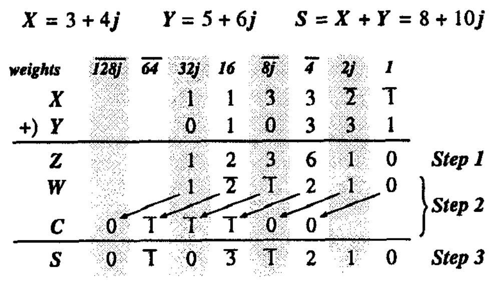

$X = 3+4j \quad Y = 5+6j \quad S = X+Y = 8+10j$
weights $\overline{128j}$ $\overline{64}$ $32j$ 16 $\overline{8j}$ $\overline{4}$ $2j$ 1
X 1 1 3 3 $\overline{2}$ $\overline{1}$
+) Y 0 1 0 3 3 1
Z 1 2 3 6 1 0 Step 1
W 1 $\overline{2}$ $\overline{1}$ 2 1 0
C 0 $\overline{1}$ $\overline{1}$ $\overline{1}$ 0 0
Step 2
S 0 $\overline{1}$ 0 $\overline{3}$ $\overline{1}$ 2 1 0 Step 3

Figure 1. RCNS $2j$,3 addition.

where Z_i is the *linear sum*, W_i is called the *intermediate sum*, and C_i is the *carry*. The ranges of these values are $Z_i\in\{\overline{6},\cdots,0,\cdots,6\}$, $W_i\in\{\overline{2},\overline{1},0,1,2\}$, and $C_i\in\{\overline{1},0,1\}$, respectively. Obviously, the *final sum* S_i is in the range of $\{\overline{3},\cdots,0,\cdots,3\}$, which is the same set as the input digit set. The carry output C_i is determined from Z_i in Step 2 independently of the other carries. Thus totally parallel addition can be achieved without carry propagation. Fig. 1 shows an example of RCNS $2j,3$ addition. The carry must be complemented and added two columns to the left.

When RCNS $2j,3$ arithmetic circuits are constructed with binary logic elements, each digit X_i ($\in\{\overline{3},\cdots,0,\cdots,3\}$) must be coded into a vector of binary bits. We introduce here redundant coding of the RCNS $2j,3$ digit X_i using 2-digit radix-2 SD code $x_i^1x_i^0$ as follows:

$$X_i \;=\; 2x_i^1+x_i^0, \quad (x_i^1,x_i^0\in\{\overline{1},0,1\}). \tag{8}$$

On the basis of this coding scheme, we can construct an RCNS $2j,3$ parallel adder with radix-2 SD adders (or "redundant binary adders" [8],[9]) as shown in Fig. 2. In this circuit, every digit x_i^δ ($\delta\in\{0,1\}$) of the radix-2 SD number is represented by 2-bit binary code.

Assuming the above coding scheme, RCNS $2j,3$ addition process (5) $\sim$ (7) can be decomposed as

$$\begin{aligned} &\text{Step 1}': && z_i^1=x_i^1+y_i^1,\; z_i^0=x_i^0+y_i^0, && (9)\\ &\text{Step 2}': && -2c_i^1+w_i^1=z_i^1,\; 2c_i^0+w_i^0=z_i^0, && (10)\\ &\text{Step 3}': && s_i^1=w_i^1+c_i^0,\; s_i^0=w_i^0+c_{i-2}^1. && (11)\end{aligned}$$

The ranges of z_i^δ, w_i^δ, and c_i^δ are $z_i^\delta\in\{\overline{2},\overline{1},0,1,2\}$, $w_i^\delta\in\{\overline{1},0,1\}$, and $c_i^\delta\in\{\overline{1},0,1\}$, respectively. The values of w_i^δ and c_i^δ are selected in order that the final sum s_i^δ is kept within the range of $\{\overline{1},0,1\}$. This can be done by using well-known carry rules of the radix-2 SD number system (redundant binary number system) [8],[9]. Fig. 3 shows an example of RCNS $2j,3$ addition using this coding.

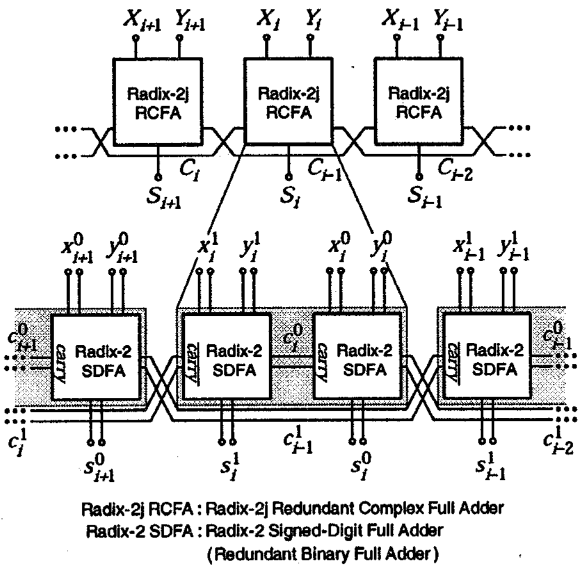
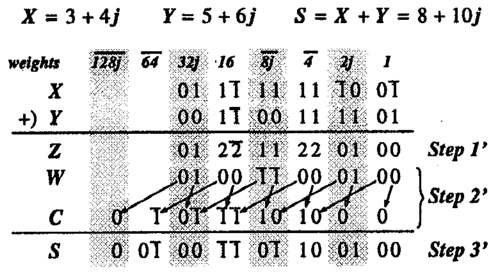

Radix-2j RCFA
Radix-2 SDFA
carry
Radix-2j RCFA : Radix-2j Redundant Complex Full Adder
Radix-2 SDFA : Radix-2 Signed-Digit Full Adder
(Redundant Binary Full Adder)

Figure 2. Realization of the RCNS parallel adder using radix-2 SDFAs.



$X = 3 + 4j \quad Y = 5 + 6j \quad S = X + Y = 8 + 10j$

weights	$\overline{128j}$	$\overline{64}$	$32j$	16	$\overline{8j}$	$\overline{4}$	$2j$	1	
X			01	$1\bar{1}$	11	11	$\bar{1}0$	$0\bar{1}$	
+) Y			00	$1\bar{1}$	00	11	11	01	
Z			01	$2\bar{2}$	11	22	01	00	Step 1'
W			01	00	$\bar{1}\bar{1}$	00	01	00	Step 2'
C	0	$\bar{1}$	$0\bar{1}$	$\bar{1}\bar{1}$	10	10	0	0	
S	0	$0\bar{1}$	00	$\bar{1}\bar{1}$	$0\bar{1}$	10	01	00	Step 3'

Figure 3. RCNS $2j$,3 addition using radix-2 SD number representation.

3. Formulation of the Hardware Reconfiguration

In this section, we show how the reconfiguration of a specific hardware structure can be described by a simple arithmetic notation. The proposed reconfigurable arithmetic unit changes its function by switching "number system" according to its operation mode. We use the triple $\langle\ D, \boldsymbol{W}, \boldsymbol{\Lambda}\ \rangle$ for representing a positional number system of n digits, where D is the *digit set*, $\boldsymbol{W} = (w_{n-1}, \cdots, w_i, \cdots, w_0)$ is the *positive weight vector*, and $\boldsymbol{\Lambda} = (\lambda_{n-1}, \cdots, \lambda_i, \cdots, \lambda_0)$ is the *sign vector* for the n weights, respectively. The multiplying factor λ_i ($\in \{\bar{1}, 1\}$) allows us to select between two possible weights w_i and $-w_i$, individually, for every digit position i. We assume the value w_i to be a power of 2, since we are interested in binary logic implementations. For example, 4-bit 2's complement binary number system is denoted by

$$\begin{aligned} &\langle\ D, \boldsymbol{W}, \boldsymbol{\Lambda}\ \rangle \\ &= \langle\ \{0,1\}, (2^3, 2^2, 2^1, 2^0), (\bar{1}, 1, 1, 1)\ \rangle. \end{aligned} \quad (12)$$

In the following, we extend the concept of "sign" into complex domain such that $\lambda_i \in \{1, \bar{1}, j, \bar{j}\}$, in order to deal with complex number systems.

A real/complex reconfigurable arithmetic unit proposed here can execute the following three arithmetic operations by changing its structure:

(i) A complex-number multiplication (with single precision).
(ii) A real-number multiplication (with double precision).
(iii) A pair of real-number four-operand multiply-add operations (with single precision).

(We assume typical signal processing applications using FFT-based pattern matching in selecting these operation modes.)

The operation (i) is executed using RCNS $2j$,3 representation. Using the triple notation, we can represent RCNS $2j$,3 of 4 digits as

$$\begin{aligned} \langle\ D_{RCNS} &= \{\bar{3}, \bar{2}, \bar{1}, 0, 1, 2, 3\}, \\ \boldsymbol{W}_{RCNS} &= (2^2, 2^1, 2^0, 2^{-1}), \\ \boldsymbol{\Lambda}_{RCNS} &= (\bar{1}, j, 1, \bar{j})\ \rangle \end{aligned} \quad (13)$$

As mentioned in the last section, the circuit implementations of RCNS $2j$,3 is based on the decomposition of RCNS $2j$,3 into radix-2 SD number representations. By decomposing every digit of the above triple notation into 2-digit radix-2 SD code, we have

$$\begin{aligned} \langle\ D_1 &= \{\bar{1}, 0, 1\}, \\ \boldsymbol{W}_1 &= (2^3, 2^2, 2^2, 2^1, 2^1, 2^0, 2^0, 2^{-1}), \\ \boldsymbol{\Lambda}_1 &= (\bar{1}, \bar{1}, j, j, 1, 1, \bar{j}, \bar{j})\ \rangle \end{aligned} \quad (14)$$

Fig. 4 (a) shows the corresponding parallel adder structure for 8 digits (4 digits for each of the real and imaginary parts). A carry generated at a specific digit position is added to the adjacent digit position or is complemented and added to the 3-digit-upper position. The single-precision complex-number multiplier for the operation (i) can be constructed with this parallel adder.

It is easy to reconfigure this structure to perform real-number additions in double-length precision. Fig. 4 (b) shows an example of possible configurations. This adder structure implies the number system:

$$\begin{aligned} \langle\ D_2 &= \{\bar{1}, 0, 1\}, \\ \boldsymbol{W}_2 &= (2^5, 2^4, 2^3, 2^2, 2^1, 2^0, 2^{-1}, 2^{-2}), \\ \boldsymbol{\Lambda}_2 &= (\bar{1}, \bar{1}, 1, 1, 1, 1, \bar{1}, \bar{1})\ \rangle \end{aligned} \quad (15)$$

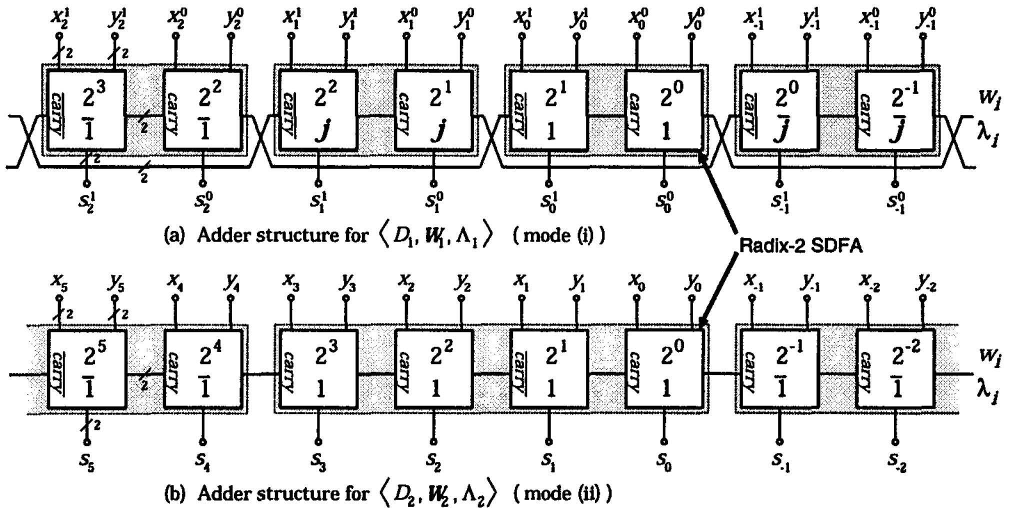

(a) Adder structure for $\langle D_1, W_1, \Lambda_1 \rangle$ (mode (i))
Radix-2 SDFA
(b) Adder structure for $\langle D_2, W_2, \Lambda_2 \rangle$ (mode (ii))
carry
w_i
λ_i

Figure 4. Reconfiguration of parallel adder structures.

Although this system employs positive/negative mixed weights, it is functionally complete for number representation. We select this representation for the operation mode (ii): double-precision real-number multiplication, so as to minimize the hardware overhead due to the mode change between (i) and (ii). Special attention is paid to the vector Λ_2 not to change the signs of its elements after reconfiguration, which allows us to use a common partial product generator for the operation modes (i) and (ii). The difference between W_1 and W_2 clearly defines the way of reconfiguring the carry interconnections in the adder tree.

For the operation mode (iii), the imaginary digits in $\langle D_1, W_1, \Lambda_1 \rangle$ are used for representing an extra real number of single precision (4 digits). Therefore, the representation thus obtained contains two distinct real numbers "overlapped" with each other, and may be written as

$$\begin{aligned} \langle\, D_3 &= \{\bar{1},0,1\}, \\ W_3 &= (2^3,2^2,\mathit{2}^{\mathit{2}},\mathit{2}^{\mathit{1}},2^1,2^0,\mathit{2}^{\mathit{0}},\mathit{2}^{-\mathit{1}}), \\ \Lambda_3 &= (\bar{1},\bar{1},\mathit{1},\mathit{1},1,1,\bar{\mathit{1}},\bar{\mathit{1}})\,\rangle \end{aligned} \quad (16)$$

where the roman part and the italic part are regarded as two distinct 4-digit real numbers. Clearly, addition in $\langle D_3, W_3, \Lambda_3 \rangle$ can be achieved by using the same parallel adder structure as shown in Fig. 4 (a), where two 4-digit real-number additions are executed in parallel. If we eliminate real-imaginary dependence within the complex-number multiplier (for the mode (i)) using this number representation, we can obtain a pair of real-number four-operand multiply-adders required for the operation mode (iii).

4. Algorithms for the Three Operation Modes

In this section, details of the arithmetic modes (i) ~ (iii) are presented. For compatibility with the binary system, input and output of the reconfigurable arithmetic unit are assumed to be 2's complement binary numbers.

4.1. Single-Precision Complex-Number Multiplication

Let the multiplicand and multiplier be denoted by X and Y, respectively. The multiplicand X is first given as an alternating sequence of the real and imaginary parts each are represented in 2's complement form, and is converted into $\langle D_1, W_1, \Lambda_1 \rangle$ representation by sign complementation (see the upper portion of Fig. 5). On the other hand, the multiplier Y is converted into RCNS $2j,2$ ($=$ $\langle \{\bar{2},\bar{1},0,1,2\}, W_{RCNS}, \Lambda_{RCNS} \rangle$) by using the Booth's recording algorithm [10]. This reduces by half the number of partial products. Assuming that X and Y are first given with the precision of $2n$ bits (n bits for each of the real and imaginary parts), the above recording process produces the expressions:

$$X = (x^0_{n-1} \cdots x^1_s x^0_s \cdots x^1_0 x^0_0 x^1_{-1}), \quad (17)$$

$$Y = (Y_{n-2} \cdots Y_t \cdots Y_0 . Y_{-1}), \quad (18)$$

where $x^\delta_s \in \{\bar{1},0,1\}$, and $Y_t \in \{\bar{2},\bar{1},0,1,2\}$.

These multiplicand and multiplier generate n partial products $P_{n-2}, \cdots, P_{-1}$, which can be represented as

$$\begin{aligned} P_t &= X \cdot Y_t \times 2^{2t} \\ &= (p^0_{nt} \cdots p^1_{st} p^0_{st} \cdots p^1_{0t} p^0_{0t} p^1_{-1t}) \times 4^t, \end{aligned} \quad (19)$$

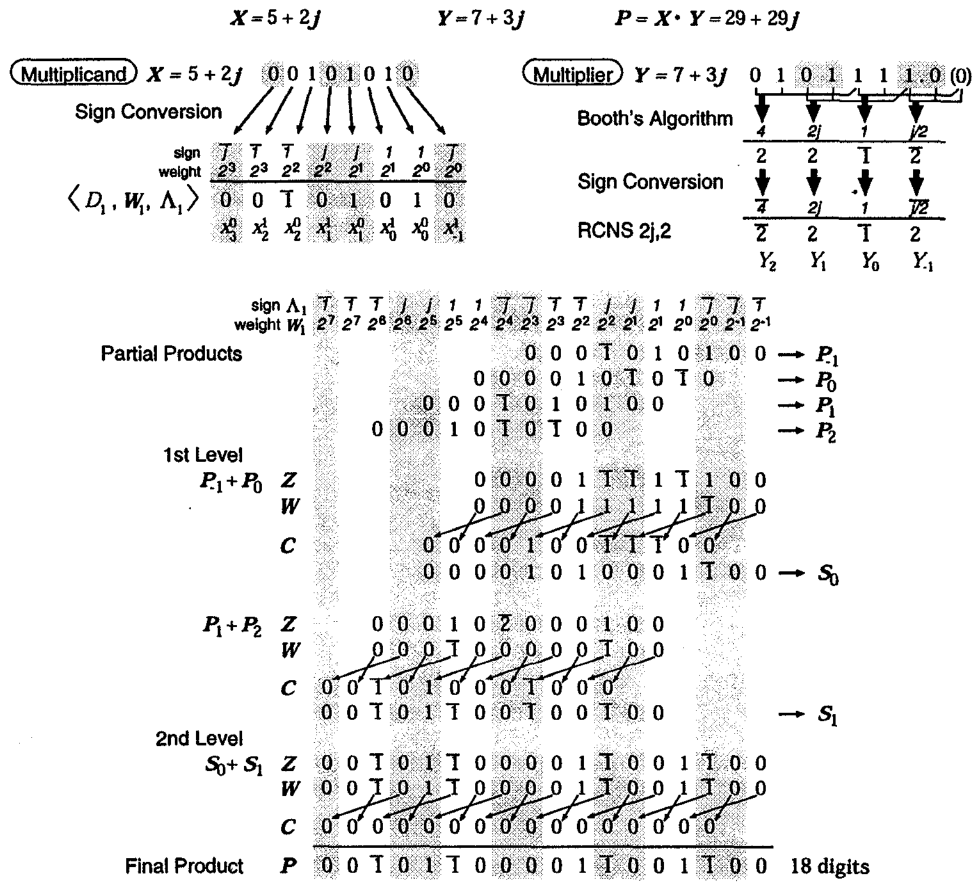


Figure 5. An example of complex multiplication (4-bit precision for each of the real and imaginary parts).

where $p^{\delta}_{st} \in \{\bar{1},0,1\}$ and $(t = -1,\cdots,n-2)$. Note that these partial products are generated by shift and complement operations as follows:

$$
\begin{array}{lll}
\text{if } Y_t = 2, & \text{then } p^1_{st} = x^0_s, & p^0_{st} = \overline{x^1_{s-2}}, \\
\text{if } Y_t = 1, & \text{then } p^1_{st} = x^1_s, & p^0_{st} = x^0_s, \\
\text{if } Y_t = 0, & \text{then } p^1_{st} = p^0_{st} = 0, & \\
\text{if } Y_t = \bar{1}, & \text{then } p^1_{st} = \overline{x^1_s}, & p^0_{st} = \overline{x^0_s}, \\
\text{if } Y_t = \bar{2}, & \text{then } p^1_{st} = \overline{x^0_s}, & p^0_{st} = x^1_{s-2}.
\end{array}
$$

The complex-number final product $\boldsymbol{P}$ is obtained by adding $\boldsymbol{P}_{n-2},\cdots,\boldsymbol{P}_{-1}$. Each partial product is in the form of $\langle D_1, \boldsymbol{W}_1, \boldsymbol{\Lambda}_1 \rangle$, and therefore the parallel adder shown in Fig. 4 (a) can be employed to achieve fast binary-tree multi-operand addition. Fig. 5 shows an example of single-precision complex-number multiplication in the mode (i).

4.2. Double-Precision Real-Number Multiplication

The multiplicand $\boldsymbol{X}$ is first given as a $2n$-bit 2's complement binary form, which is converted into $\langle D_2, \boldsymbol{W}_2, \boldsymbol{\Lambda}_2 \rangle$ representation by sign complementation (see the upper portion of Fig. 6). On the other hand, the multiplier $\boldsymbol{Y}$ is converted into the representation $\langle \{\bar{2},\bar{1},0,1,2\},(4^2,4^1,4^0,4^{-1}),(\bar{1},1,1,\bar{1}) \rangle$ (i.e., the minimally redundant radix-4 SD number system whose weights are complemented every 2-digit positions) by using Booth's algorithm. Using this representation, partial product generation is achieved only by shift and complement operations. Consequently, the $2n$-bit multiplicand $\boldsymbol{X}$ and the $2n$-bit multiplier $\boldsymbol{Y}$ are converted into the form:

$$\boldsymbol{X} = (x_{2n-2}\cdots x_s \cdots x_0.x_{-1}), \quad (20)$$
$$\boldsymbol{Y} = (Y_{n-2}\cdots Y_t \cdots Y_0.Y_{-1}), \quad (21)$$

where $x_s \in \{\bar{1},0,1\}$, and $Y_t \in \{\bar{2},\bar{1},0,1,2\}$.

These multiplicand and multiplier generate n partial products $\boldsymbol{P}_{n-2},\cdots,\boldsymbol{P}_{-1}$, which are denoted by

$$
\begin{aligned}
\boldsymbol{P}_t &= \boldsymbol{X} \cdot Y_t \times 2^{2t} \\
&= (p_{(2n-1)t} \cdots p_{st} \cdots p_{-1t}) \times 4^t, \quad (22)
\end{aligned}
$$

where $p_{st} \in \{\bar{1},0,1\}$ and $(t = -1,\cdots,n-2)$. These

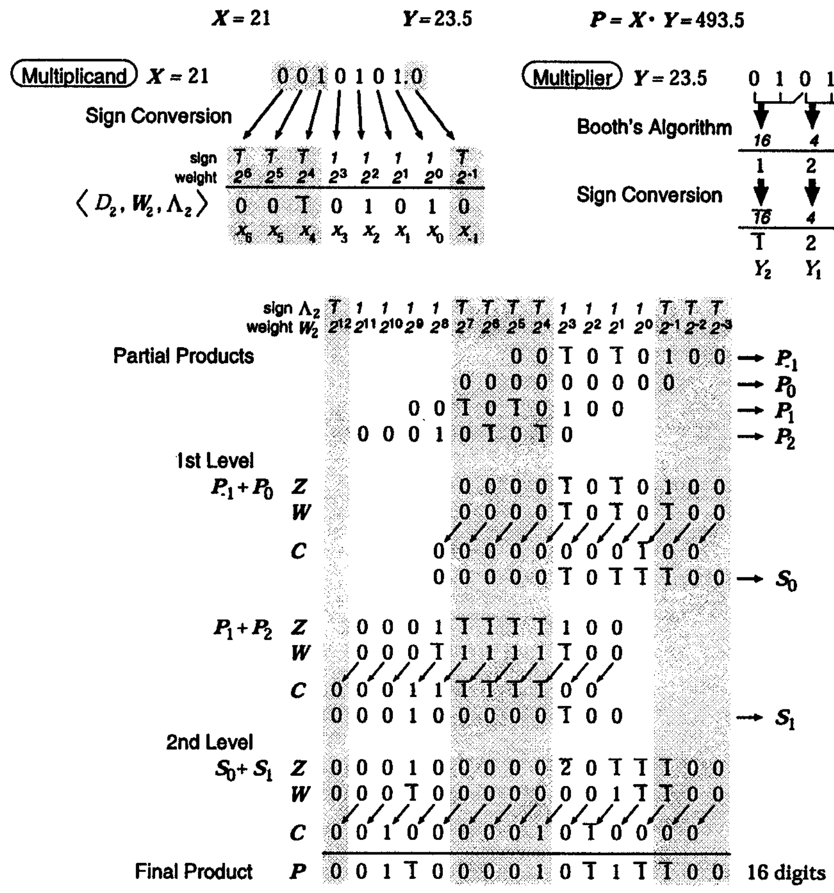


Figure 6. An example of real multiplication (8-bit precision).

partial products are generated by shift and complement operations as follows:

(a) t : even

$$\text{if } Y_t = 2, \quad \text{then } p_{st} = \begin{cases} \overline{x_{s-1}} & (\text{for } s = 4m) \\ x_{s-1} & (\text{for } s \neq 4m) \end{cases}$$
$$\text{if } Y_t = 1, \quad \text{then } p_{st} = x_s$$
$$\text{if } Y_t = 0, \quad \text{then } p_{st} = 0$$
$$\text{if } Y_t = \bar{1}, \quad \text{then } p_{st} = \overline{x_s}$$
$$\text{if } Y_t = \bar{2}, \quad \text{then } p_{st} = \begin{cases} x_{s-1} & (\text{for } s = 4m) \\ \overline{x_{s-1}} & (\text{for } s \neq 4m) \end{cases}$$

(b) t : odd

$$\text{if } Y_t = 2, \quad \text{then } p_{st} = \begin{cases} x_{s-1} & (\text{for } s = 4m) \\ \overline{x_{s-1}} & (\text{for } s \neq 4m) \end{cases}$$
$$\text{if } Y_t = 1, \quad \text{then } p_{st} = \begin{cases} x_s & (\text{for } s = 4m, 4m{+}1) \\ \overline{x_s} & (\text{for } s \neq 4m, 4m{+}1) \end{cases}$$
$$\text{if } Y_t = 0, \quad \text{then } p_{st} = 0$$
$$\text{if } Y_t = \bar{1}, \quad \text{then } p_{st} = \begin{cases} \overline{x_s} & (\text{for } s = 4m, 4m{+}1) \\ x_s & (\text{for } s \neq 4m, 4m{+}1) \end{cases}$$
$$\text{if } Y_t = \bar{2}, \quad \text{then } p_{st} = \begin{cases} \overline{x_{s-1}} & (\text{for } s = 4m) \\ x_{s-1} & (\text{for } s \neq 4m) \end{cases}$$

where m is an integer. The real-number final product $\boldsymbol{P}$ is obtained by adding $\boldsymbol{P}_{n-2}, \cdots, \boldsymbol{P}_{-1}$. Each partial product is in the form of $\langle D_2, W_2, \Lambda_2 \rangle$, and hence the parallel adder shown in Fig. 4 (b) can be employed to achieve fast binary-tree multi-operand addition. Fig. 6 shows an example of double-precision real-number multiplication in the mode (ii).

The number system $\langle D_2, W_2, \Lambda_2 \rangle$ used here is carefully selected in order to minimize the hardware overhead required for the mode change between (i) and (ii).

4.3. Single-Precision Real-Number Four-Operand Multiply-Add Operation

This operation mode employs $\langle D_3, W_3, \Lambda_3 \rangle$ representation to execute a pair of four-operand multiply-add operations in parallel. Since there is no space for an extended discussion, we shall explain only the principle of this operation mode without describing its algorithm in detail.

The complex-number multiplication realized in the

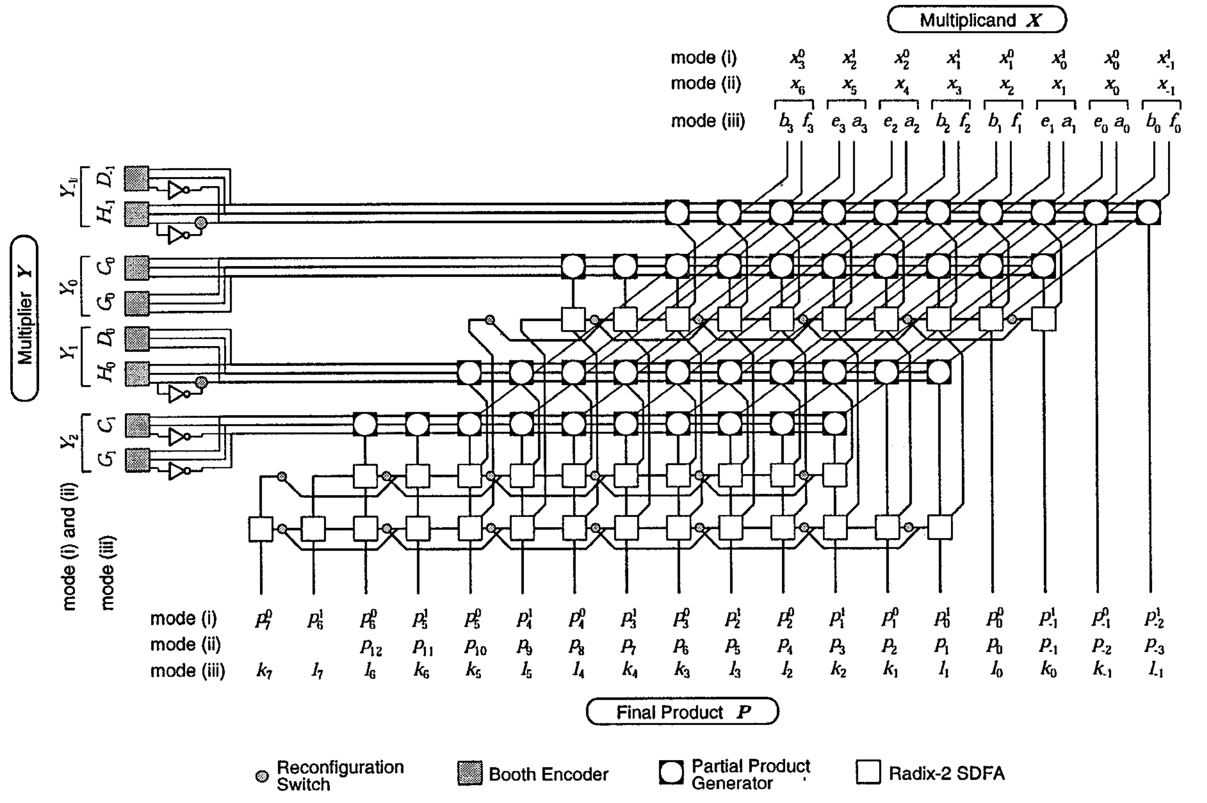


Figure 7. Real/complex reconfigurable arithmetic unit (8 bits for double precision).

operation mode (i) can be expressed simply by

$$(A+Bj)(C+Dj)=(AC-BD)+(AD+BC)j, \tag{23}$$

where A and C denote the explicit values of the real parts, and B and D are those of the imaginary parts. If we consider A, B, C, and D as four distinct real-number inputs of single precision, then the imaginary part $AD+BC$ directly corresponds to the result of four-operand multiply-add operation on these numbers. Thus, ~~a~~ complex-number multiplier can perform four-operand multiply-add operation.

However, it should be noted that the circuit elements to be operated in the calculation of $AD+BC$ are half of the elements of the whole complex multiplier. It is possible to achieve 100% utilization of hardware elements by introducing an extra set of operands E, F, G, and $-H$ to the remaining portion of the complex multiplier (i.e., the portion that calculates the real part of the result) as

$$(E+Fj)(G-Hj)=(EG+FH)+(FG-EH)j. \tag{24}$$

Thus, the real part $EG+FH$ corresponds to the result of multiply-add operation of the four operands. As a result, a pair of four-operand multiply-adds: $K = AD+BC$ and $L = EG+FH$, are carried out in parallel with the modified complex multiplier. Only some adjustments of the input data format are needed to achieve this operation.

Fig. 7 shows the real/complex dynamically-reconfigurable arithmetic unit realizing the above mentioned three operation modes. The circuit includes switches to reconfigure the circuit structure. Transistor count of the real/complex reconfigurable arithmetic unit for various word lengths are shown in Table 1. The hardware overhead required for the reconfiguration is very small, that corresponds to 16% increase of gate count in comparison with the dedicated complex-number multiplier. Thus, mathematical treatment of hardware design makes possible the construction of highly regular reconfigurable arithmetic circuits including only homogeneous interconnection patterns.

5. Discussion and Conclusion

In this paper, we have proposed the algorithm for the real/complex reconfigurable arithmetic unit that executes (i) a single-precision complex-number multiplication, (ii) a double-precision real-number multiplication, or (iii) a pair of single-precision real-number

Word length (double precision)	8	16	32	64
SDFAs	2176	9088	35840	139776
PPGs	1120	4032	15232	59136
BEs	224	448	896	1792
Switches	232	768	2344	9592
Total	3752	14336	54312	210296

Table 1. Transistor count of the reconfigurable arithmetic unit.

four-operand multiply-add operations, according to external control instructions. We have shown that the reconfiguration of hardware can be formulated in terms of the transformation of the number system used in each arithmetic mode.

The proposed arithmetic unit can be installed into typical programmable DSP architectures as shown in Fig. 8 to achieve high-speed real/complex mixed computations. In this case, the data path includes the reconfigurable arithmetic unit and the ALU in tandem, and hence the ALU must also be reconfigured its carry interconnections according to the operation mode. Also, dynamic switches are required between the data bus and the reconfigurable arithmetic unit in order to adjust the data format to the specified mode. In practice, a few tag bits representing the operand type $\langle D_i, W_i, A_i \rangle$ $(i \in \{1,2,3\})$ may be attached to the data itself. The arithmetic unit will recognize this information and change its structure automatically, which makes possible the considerable simplification of reconfiguration control logic. Such highly flexible arithmetic unit will be useful in many digital signal processing applications, which require both real-number and complex-number arithmetic capabilities.

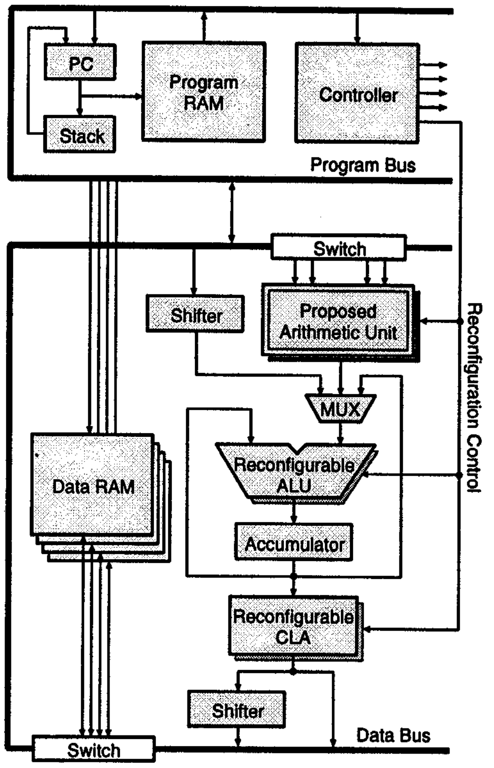


Figure 8. Installation of the reconfigurable arithmetic unit into typical DSP architecture.

References

[1] B. Barazesh, J. Michalina, and A. Picco,"A VLSI signal processor with complex arithmetic capability," *IEEE Trans. Circuits and Systems*, Vol. 35, No. 5, pp. 495–505, May 1988.

[2] B. W. Y. Wei, H. Du, and H. Chen,"A complex-number multiplier using radix-4 digits," *Proc. 12th IEEE Int'l Symp. Computer Arithmetic*, pp. 84–90, July 1995.

[3] T. T. Dao,"Knuth's complex arithmetic with quaternary hardware," *Proc. 12th IEEE Int'l Symp. Multiple-Valued Logic*, pp. 94–98, May 1982.

[4] J. Duprat, Y. Herreros, and S. Kla,"New redundant representations of complex numbers and vectors," *IEEE Trans. Computers*, Vol. C-42, No. 7, pp. 817–824, July 1993.

[5] T. Aoki, Y. Ohi, and T. Higuchi,"Redundant complex number arithmetic for high-speed signal processing," *VLSI SIGNAL PROCESSING VIII (1995 IEEE Workshop on VLSI Signal Processing)*, pp. 523–532, October 1995.

[6] D. E. Knuth,*The Art of Computer Programming*, Vol. 2,Reading, MA: Addison Wesley, 1973.

[7] A. Avizienis,"Signed-digit number representations for fast parallel arithmetic," *IRE Trans. Electronic Computers*, Vol. EC-10, pp. 389–400, September 1961.

[8] N. Takagi, H. Yasuura, and S. Yajima,"High-speed VLSI multiplication algorithm with a redundant binary addition tree," *IEEE Trans. Computer*, Vol. C-34, No. 9, pp. 789–796, September 1985.

[9] H. Makino, Y. Nakase, H. Suzuki, H. Morinaka, H. Shinohara, and K. Mashiko,"An 8.8-ns 54×54-bit multiplier with high speed redundant binary architecture," *IEEE J. Solid-State Circuits*, Vol. 31, No. 6, pp. 773–783, June 1996.

[10] A. D. Booth,"A signed binary multiplication technique," *Quart. J. Mech. Appl. Math.*, Vol. 4, pp. 236–240, August 1951.