A Complex-Number Multiplier Using Radix-4 Digits

Belle W. Y. Wei, He Du, and Honglu Chen

Department of Electrical Engineering, College of Engineering,
San Jose State University, San Jose, CA 95192-0084, bwei@sparta.sjsu.edu

Abstract - This paper describes the design of a 16x16 complex-number multiplier developed as part of the arithmetic datapath of a complex-number digital signal processor. The complex-number multiplier internally uses binary signed digits for fast multiplication and compact layout. It employs the traditional three-multiplication scheme while minimizing the logic and delay associated with the three extra pre-multiplication binary additions which that scheme requires. The minimization comes from producing the redundant binary sum for each of the pre-multiplication binary additions with minimal hardware, and then recoding the redundant sums as radix-4 multiplier operands. The radix-4 operands halve the number of summands to be added in each of the three real multiplier units. Furthermore, an additional factor of two reduction in the number of summands is effectuated by our coding scheme for representing binary signed digits. The result is a fast and compact complex-number multiplier.

1. Introduction

Complex-number multipliers process and filter signals arising from Fast Fourier Transform (FFT), quadrature signal representations, or Hilbert transforms. They are time-critical components for radar, satellite, and digital modulation applications.

A direct implementation of a complex-number multiplication uses four multipliers with significant chip area and delay: $(A+jB)\ (C+jD) = R+jI$ where $R = AC-BD$ and $I = AD+BC$. One method in reducing the multiplication count is the use of modular arithmetic. Examples are Quadratic Residue Number System (QRNS), Quadratic Like Residue Number System (QLRNS), Modified Quadratic Residue Number System (MQRNS), and Polynomial Residue Number System (PRNS)[1][2][3][4][5]. For instance, a complex multiplication in QRNS requires two real multiplications instead of four. The trade-off is a large number of adders used for preprocessing and postprocessing. If we balance a number of design issues like dynamic range, data width, scaling, RNS-to-binary conversion, post/preprocessing, and circuit pipeline, it is doubtful that these RNS systems are as competitive as conventional binary implementations. This is especially true in view of recent advances in the development of commercial floating processors with fast and compact binary multipliers [6][7][8][9].

Among binary complex-number multipliers, a common implementation scheme is to merge real (or imaginary) products in one adder tree summing up twice the number of partial products, i.e, $R = AC-BD$, without reducing logic complexity [10]. Another method is to make $m_0 = (A+B)\ (C+D)$, $m_1 = AC$, $m_2 = BD$, and then compute $R = m_1 - m_2$ and $I = m_0 - m_1 - m_2$. The resulting circuit has one less multiplication but three more additions than the conventional scheme.

Our complex-number multiplier uses three binary multipliers with minimal additional hardware for extra additions due to two design innovations. The first is the use of a particular coding of redundant binary digits (BSD)[11] such that producing redundant binary sums from binary numbers uses minimal hardware [12][13]. The second is the "recoding" of redundant binary numbers into radix-4 digits which are used as a multiplier operand for subsequent multiplication operations. The recoding is

*This work was supported in part by NSF grant MIP-9321143.

accomplished by applying two "filtering" factors to the target redundant binary number, and this leads to simple logic.

In Section 2, we present the computational algorithm and architecture of our complex-number multiplier. Section 3 discusses two key components of our design: redundant binary addition and the radix-4 recoding of redundant binary numbers. Section 4 describes the circuit implementation and layout as part of a complex-number multiplier-and-accumulator (MAC) unit. The final section summarizes our design.

2. Architecture

Given $(A+jB)(C+jD) = R+jI$ where $R = AC-BD$ and $I = AD+BC$, a complex-number multiplication can be written as:

$$m_0 = (C-D) \times B \quad (1)$$

$$m_1 = (A-B) \times C \quad (2)$$

$$m_2 = (A+B) \times D \quad (3)$$

$$R = m_1 + m_0 \quad (4)$$

$$I = m_2 + m_0. \quad (5)$$

This method requires three additions to generate pre-multiplication sums, and three multiplications and two additions for the final results, a saving of one multiplication at a cost of three extra additions. However, we can implement these three additions at minimal additional cost, thus realizing savings in both chip area and speed. Figure 1 presents a block diagram for the above computation in which the pre-multiplication sums are in redundant binary (RB) forms produced by *Redundant Binary Coders*. These RB values are then filtered into a form suitable for radix-4 recoding. The filtering and recoding functions are performed by *Binary Signed Digit* (BSD) *Recoders*. The resultant radix-4 digits, in the same format as Booth recoded multiplier digits, feed the subsequent *Redundant Binary Multipliers*. The products are then added and converted to binary outputs by *Redundant Binary Adder and Converter*.

3. Redundant Binary Coder and Binary Signed Digit Recoder

Among four circuit blocks of a complex-number multiplier, *Redundant Binary Multipliers* and *Redundant Binary Adder and Converters* have been documented

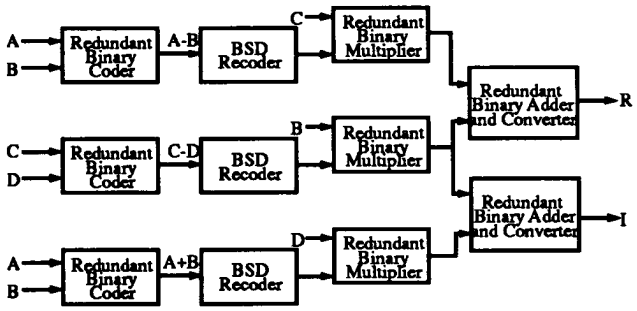


Figure 1. A Complex-Number Multiplier

extensively in the literature [9][12][13][14][15]. It is *Redundant Binary Coders* and *BSD Recoders* that merit special discussion. A *Redundant Binary Coder* adds up (or subtracts) a pair of 2's complement numbers and generates the corresponding sum (or difference) in redundant binary format. Consider the sum of two $(n+1)$-bit binary numbers A and B as follows:

$$A+B\,(\text{or } A-B) = d_n 2^n + \sum_{i=0}^{n-1} d_i 2^i + g2^0 \quad (6)$$

where

$$A = (a_n a_{n-1} \ldots a_1 a_0),$$

$$B = (b_n b_{n-1} \ldots b_1 b_0),$$

$$a_n, a_{n-1}, \ldots, a_1, a_0, b_n, b_{n-1}, \ldots, b_1, b_0 \in \{0,1\},$$

$$d_n, d_i, g \in \{\bar{1},0,1\}.$$

Let two coding bits 00, 01, 10, and 11 represent $\bar{1}$, 0, 0, and 1 respectively. The resulting logic for a redundant binary sum digit is trivial, and is summarized in Table 1 in which a BSD d_i ($d_i \in \{\bar{1},0,1\}$) has two coding bits $d_i^- d_i^+$. The corresponding *Redundant Binary Coder* has only inverters which along with the correction term g are integrated with the following *BSD Recoder*. We thus obtain the pre-multiplication sum in redundant binary form at minimal costs.

Table 1. Redundant Binary Sum Digits for Adding Two Binary Numbers

	d_n		$d_i (0 \le i \le n-1)$		$g\,(g^- g^+)$
	d_n^-	d_n^+	d_i^-	d_i^+	
$A+B$	$\bar{a}_n$	$\bar{b}_n$	a_i	b_i	−1 (00)
$A-B$	$\bar{a}_n$	b_n	a_i	$\bar{b}_i$	0 (01)

The redundant binary sum described above is

then presented as a multiplier to its following multiplication circuit. Typically a binary multiplier operand is Booth recoded into radix-4 digits to reduce the number of partial products. Since the redundant binary multiplier operand is not in binary format, it cannot use Booth recoding directly. If one adopts "Booth-like" strategy of scanning consecutive BSD digits, the resulting digit may be -3 or 3. To solve this problem, we first filter the redundant binary multiplier, X_{SD}, by two filtering factors F_{SD} and G_{SD}, and then recode the filtered output, Y_{SD}. The filtering operation is:

$$\begin{aligned} Y_{SD} &= (X_{SD} + F_{SD}) + G_{SD} \\ &= W_{SD} + G_{SD} \end{aligned}$$

where

$$X_{SD} = \sum_{i=0}^{n} x_i \cdot 2^i, x_i = \{\bar{1}, 0, 1\}$$

$$Y_{SD} = \sum_{i=0}^{n} y_i \cdot 2^i, y_i = \{\bar{1}, 0, 1\}$$

$$W_{SD} = \sum_{i=0}^{n} w_i \cdot 2^i, w_i = \{\bar{1}, 0, 1\}.$$

Constants F_{SD} and G_{SD} are filtering factors defined as $F_{SD} = (...0\bar{1}0\bar{1})_{SD}$ and $G_{SD} = (...0101)_{SD}$. The filtered redundant number, Y_{SD}, features that its value is the same as the original X_{SD} and the product of even-odd adjacent digits cannot be one, that is, $y_{i+1}y_i \neq 11$ or $y_{i+1}y_i \neq \bar{1}\bar{1}$ for even i ($i = 0,2,4...$). Consequently, recoding Y_{SD}'s even-odd digit pair eliminates the possibility of producing +3 or -3 values.

The reason why the filtering operation produces the desired result hinges upon the peculiarities of redundant binary addition. A redundant binary addition takes two steps as shown in Figure 2. The first step adds x_i and y_i, and generates an intermediate sum (s_i) and an intermediate carry (c_i) : $x_i + y_i = 2c_i + s_i$ where x_i, y_i, c_i, and $s_i \in \{\bar{1}, 0, 1\}$. The second step produces the final sum z_i by summing s_i and c_{i-1}, the intermediate carry from the less significant position; that is, $z_i = s_i + c_{i-1}$. In order to render a carry-propagation free addition, the choice of the intermediary carry and sum digits should be such that the final sum digit z_i is neither 2 nor -2.

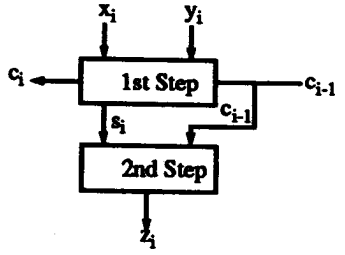


Figure 2. Two Steps of a Redundant Binary Addition

Tables 2 and 3 tabulate the two-step redundant binary addition of even-odd digit positions for the first filtering operation $W_{SD} = X_{SD} + F_{SD}$. Tables 2.a and 2.b list the first-step addition in generating the intermediate sum and carry digits for the even and odd positions respectively. Variable t_{2k} (or t_{2k+1}) indicates whether the corresponding intermediate carry belongs to {1,0} or {$\bar{1}$, 0}. This information from the less significant position is used to assign intermediate sum digit values, i. e. the second row of Table 2.a, such that the second-step addition results in a BSD sum digit as shown in Table 3. Equipped with these tables, we can show that the even-position sum digit (w_{2k}) cannot be 1:

For x_k=1, s_{2k}=0, c_{2k-1}=0 $\Rightarrow$ w_{2k}=0,
c_{2k-1}=1 $\Rightarrow$ w_{2k}=1;
x_k=0, s_{2k}=1, then c_{2k-1}=0 $\Rightarrow$ w_{2k}=1,
s_{2k}=1, then c_{2k-1}=1 $\Rightarrow$ w_{2k}=0;
x_k=1, s_{2k}=0, c_{2k-1}=0 $\Rightarrow$ w_{2k}=0,
c_{2k-1}=1 $\Rightarrow$ w_{2k}=1.

That is, the filtering produces the $w_{2k+1}w_{2k}$ output in the range of [-3, 2].

Table 2.a The First Step of

$$W_{SD} = X_{SD} + F_{SD}$$

for an Even Digit Position

t_{2k}	$x_{2k}f_{2k}$ (f_{2k}=$\bar{1}$)	$x_{2k-1}f_{2k-1}$ (f_{2k-1}=0)	$c_{2k}s_{2k}$
t_{2k} = 0	$\bar{1}\bar{1}$	-	$\bar{1}0$
	$0\bar{1}$	t_{2k-1} = 1	$0\bar{1}$
		t_{2k-1} = 0	$\bar{1}1$
	$1\bar{1}$	-	00

Table 2.b The First Step of

$W_{SD} = X_{SD} + F_{SD}$

for an Odd Digit Position

t_{2k+1}	$x_{2k+1}f_{2k+1}$ $(f_{2k+1}=0)$	$x_{2k}f_{2k}$ $(f_{2k}=\bar{1})$	$c_{2k+1}s_{2k+1}$
$t_{2k+1} = 1$	00	-	00
	10	-	01
$t_{2k+1} = 0$	$\bar{1}0$	-	T1

Table 3. The Second Step of a Redundant Binary Addition

s_i	c_{i-1}	w_i, y_i
0	1	1
0	T	T
1	0	1
1	T	0
T	0	T
T	1	0
0	0	0

Similar tables (Tables 4.a and 4.b) can be constructed for the subsequent filtering operation $Y_{SD} = W_{SD} + G_{SD}$, i.e., adding "1" to each w_{2k} digit. The resultant y digits are $y_{2k} = w_{2k} + 1 \in [0,1]$ and $y_{2k+1} = w_{2k+1} \in [-1,0,1]$. We can prove $y_{2k+1}y_{2k} \neq 11$ with Tables 4.a and 4.b:

$$y_{2k+1} = 1 \Rightarrow s_{2k+1}=0 \text{ and } c_{2k} = 1 \Rightarrow y_{2k}=0.$$

Table 4.a The First Step of

$Y_{SD} = W_{SD} + G_{SD}$

for an Even Digit Position

t_{2k}	$w_{2k}g_{2k}$ $(g_{2k}=1)$	$w_{2k-1}g_{2k-1}$ $(g_{2k-1}=0)$	$c_{2k}s_{2k}$	$y_{2k} = s_{2k} + t_{2k-1}$
$t_{2k} = 1$	01	$t_{2k-1} = 1$	1T	0
		$t_{2k-1} = 0$	01	1
	T1	-	00	0 or 1

Table 4.b The First Step of

$Y_{SD} = W_{SD} + G_{SD}$

for an Odd Digit Position

t_{2k+1}	$w_{2k+1}g_{2k+1}$ $(g_{2k+1}=0)$	$w_{2k}g_{2k}$ $(g_{2k}=1)$	$c_{2k+1}s_{2k+1}$
$t_{2k+1} = 1$	10	-	1T
$t_{2k+1} = 0$	00	-	00
	T0	-	0T

After the filtering operation, the recoding becomes trivial: $\sigma_{2k} = 2y_{2k+1} + y_{2k}$ and $\sigma_{2k} \in \{0, \pm 1, \pm 2\}$.

The logic for *Binary Signed Digit* (BSD) *Recoders* can be derived with the BSD coding of 00 for $\bar{1}$, 01 or 10 for 0, and 11 for 1. Let $x_i^- x_i^+$ and $y_i^- y_i^+$ be coding bits for x_i and y_i respectively, and

$$r_i = (x_i^- \oplus x_i^+)$$
$$p_i = (x_{i-1}^- + x_{i-1}^+)(x_{i-2}^- + x_{i-2}^+) + x_{i-1}^- x_{i-1}^+$$
$$q_i = x_{i-2}^- x_{i-2}^+ \overline{r_{i-1}} + r_{i-1}\overline{x_{i-2}^- x_{i-2}^+}.$$

Then the filtered output is:

$$y_{2k}^- = 1$$
$$y_{2k}^+ = (\overline{r_{2k-1}} p_{2k-1}) \oplus (r_{2k} \oplus (x_{2k-1}^- + x_{2k-1}^+))$$
$$y_{2k+1}^- = \overline{r_{2k+1}} \oplus p_{2k+1}$$
$$y_{2k+1}^+ = p_{2k-1} q_{2k+1}.$$

The recoded radix-4 digit σ_{2k} is:

$$(\sigma_{2k} = 0) = \overline{y_{2k}^+}\ (y_{2k+1}^- \oplus y_{2k+1}^+)$$
$$(\sigma_{2k} = 1) = y_{2k}^+\ (y_{2k+1}^- + y_{2k+1}^+)$$

$$(\sigma_{2k} = \bar{1}) = y_{2k}^{+}\ \overline{(y_{2k+1}^{-} + y_{2k+1}^{+})}$$
$$(\sigma_{2k} = 2) = y_{2k+1}^{-} y_{2k+1}^{+}$$
$$(\sigma_{2k} = \bar{2}) = \overline{y_{2k}^{+}}\ \overline{(y_{2k+1}^{-} + y_{2k+1}^{+})}.$$

4. Implementation

Our complex-number multiplier was implemented as part of a complex-number MAC, core component of a complex-number digital signal processor. The MAC has three pipeline stages and supports multiplication-and-accumulation and direct multiplication as shown in Figure 3. The multiplier part consists of three *RB Coder/BSD Recoders*, three *RB Multipliers*, and two *Redundant Binary Adder and Converters*. A *RB Coder/ BSD Recoder* inputs a pair of 16-bit binary numbers and produces 9 radix-4 digits. These 9 recoded radix-4 digits constitute the multiplier operand for the *RB Multiplier*. Adding 9 binary partial products in the adder tree of the *RB Multiplier* requires three ($\lceil \log_2 9/2 \rceil$) tree levels in which the factor of two reduction in the number of summands comes from BSD coding discussed above. The redundant binary multiplication products produced by three *RB Multipliers* are added in the fourth-level *RB Adder* according to Equations (4) and (5). The redundant binary results then feed an accumulation redundant binary adder, *Acc RBA*, for fast accumulation. The accumulated redundant binary sum is then converted to binary in a *Redundant Binary Converter* implemented by a 41-bit adder.

A *BSD Recoder* implements the radix-4 encoding logic and its circuit is shown in Figure 4. The circuit inputs 6 bits, 2 bits for each digit position, and produces five unencoded radix-4 outputs. It has six gates on its critical path if we treat an XOR (XNOR) gate as one 1.5 gate, an inverter as one 0.5 gate, all other gates as one gate. The 1.2 μ CMOS implementation has a latency of 3.8 ns that includes a one-stage driver. Compared with a regular Booth recoder inputting a binary operand, a *BSD Recoder* circuit is twice as complicated and slow.

The *Redundant Binary Multiplier* includes Booth decoders generating partial products (PPG), and redundant binary adders (RBA). Figure 5 shows a redundant binary adder circuit adding a pair of BSD x_i and y_i and generating one BSD z_i. The circuit has five gates on its critical path and a latency of 3 ns.

5. Discussion and Summary

We have implemented a 16-b × 16-b complex-number MAC using a BSD radix-4 recoding scheme. The radix-4 recoding coupled with our coding of binary signed digits allows us to use the three-mulplication scheme with minimal extra hardware for the three extra pre-multiplication binary additions. In addition, our BSD codes halve the number of summands to be added up in the multiplier's adder tree. Table 5 summarizes the design characteristics of our complex-number MAC, and Figure 6 shows the circuit's micrograph.

Table 5. Characteristics of a 16 x 16 Complex-Number MAC

Minimum Feature	1.2 μm (drawn), two-level metal
Transistor Count	57,634
Size	30.8 mm^2
Clock Rate	100 MHz
Power	26.2 mW

Our design of using radix-4 recoding does not reduce logic complexity when compared with the implementation of merging real (or imaginary) products in one adder tree to produce real (or imaginary) parts [10]. Both require four levels of redundant binary addition. Our advantage lies in the dense routing and compact layout of distributing fewer partial products to be summed up in one multiplier unit. Furthermore, our radix-4 recoding scheme is general, and is useful in computations such as division/square-root [16].

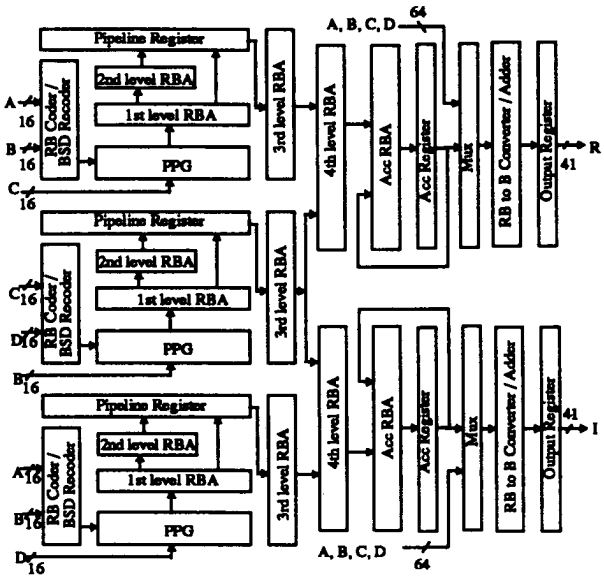


Figure 3. Floor Plan of a 16 x 16 Complex-Number Multiplier-and-Accumulation

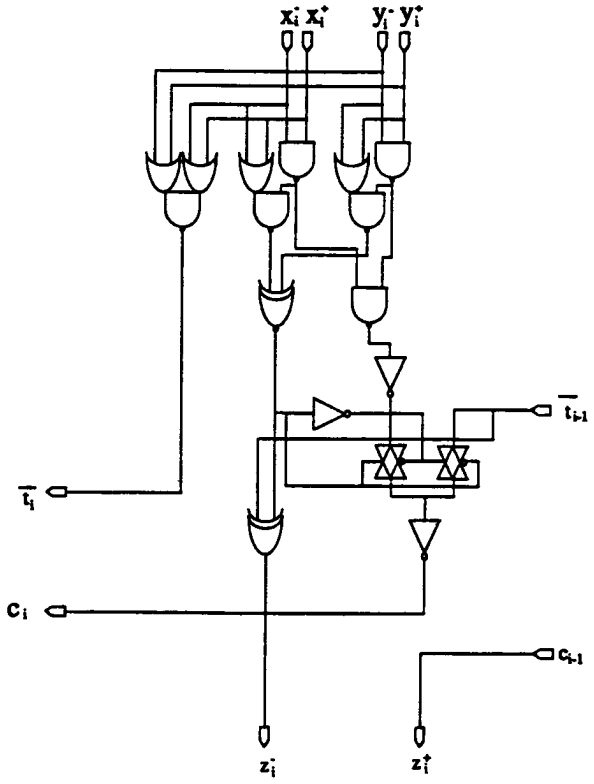
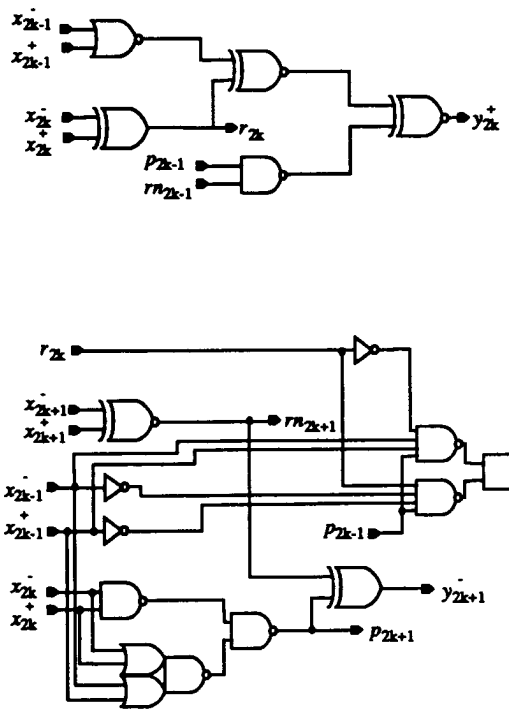


Figure 5. A Redundant Binary Adder



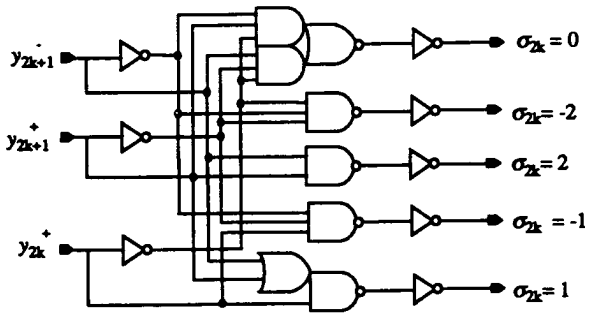


Figure 4. A Radix-4 Recoder of Redundant Binary Numbers

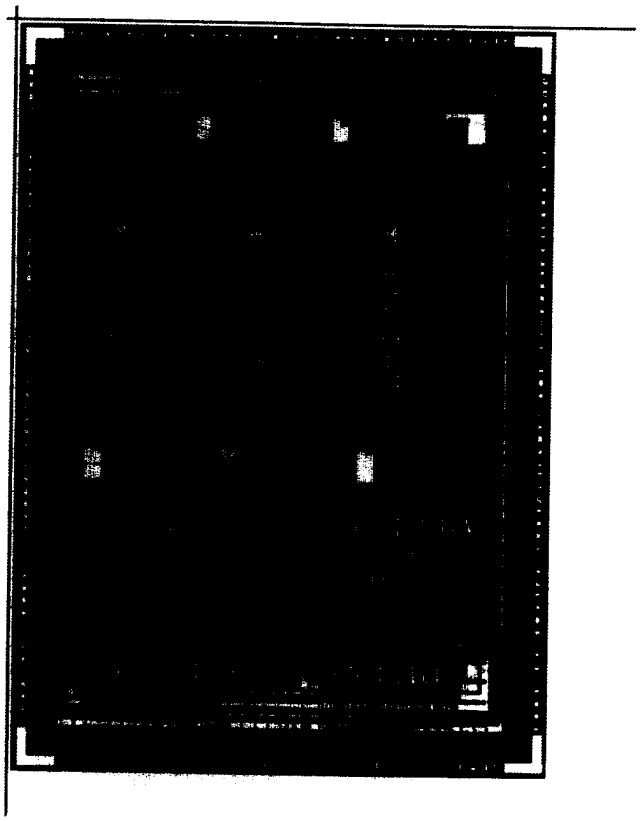


Figure 6. A Micrograph

References

[1] S.H. Leung, "Application of Residue Number Systems to Complex Digital Filters," in *Proceedings of Fifteenth Asilomar Conference on Circuits, Systems, and Computers*, pp. 70 - 74, November 1981.

[2] Ramasamy Krishnan, "Conventional Binary Number System (BNS) versus Residue Number System (RNS) Digital Signal processing Architecture Suitable for Complex Digital Filtering," in *Processing of Twenty-third Asilomar Conference on Circuits, Systems and Computers*, Pacific Grove, CA, November 1989.

[3] R. Krishnan, G.A. Jullien and W.C. Miller, "Complex Digital Signal Processing Using Quadratic Residue Number Systems," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP - 34, No. 1, February 1985.

[4] Michael A. Soderstrand, Gregory D. Poe, "Applications of Quadratic-like Complex Residue Number System Arithmetic to Ultrasonics," in *Proceedings 1984 ICASSP*, San Diego, CA, Mar. 1984.

[5] Alexander Skavantzos, Zarir B. Sarkari, Thanos Stouraitis, "A Complex DSP Processor Using Polynomial Encoding," *IEEE International Conference on Acoustics, Speech and Signal Processing*, v II (of 4), pp. 1310 - 1313, 1989.

[6] M. Birman, G. Chu, L. Hu, J. Mcleod, N. Bedard, F. Ware, L. Torban, C. M. Lim, "Design of a High-Speed Arithmetic Datapath," *IEEE International Conference on Computer Design (ICCD 88)*. pp. 214 - 216.

[7] James Miller, Ben Roberts, Paul Madiand, "High Performance Circuits for the i486 Processor," *IEEE International Conference on Computer Design (ICCD 89)*, pp. 188 - 192.

[8] Donald Steiss *et al.*, "A 65MHz Floating-Point Coprocessor for a RISC Processor," *ISSCC Digest of Technical Papers*, pp. 94-95, February 1991.

[9] Junichi Goto, Kouichi Ando, Toshiaki Inoue, Masaki Ishida, Masakazu Yamashina Hachiro Yamada, Tadayoshi Enomoto, "A 250-MHz 16b 1-Million Transistor BiCMOS Super-High-Speed Video Signal Processor," *ISSCC Digest of Technical Papers*, pp.254 - 255, February 1991.

[10] Vojin Oklobdzija, David Villeger and Thierry Soulas, "An Integrated Multiplier for Complex Numbers," *Journal of VLSI Signal Processing*, vol. 7, no. 3, pp. 213-222, May 1994.

[11] A. Avizienis, "Signed-Digit Number Representations for Fast Parallel arithmetic," *IRE Transactions on Electronic Computer*, Vol. EC-10, No. 9, pp. 389 - 400, September 1961.

[12] Xiaoping Huang, Wen-Jung Liu, and Belle W. Y. Wei, "A High-Performance CMOS Redundant Binary Multiplication (MAC) Unit," *IEEE Transactions on Circuits and Systems-I: Fundamental Theory and Applications*, vol. 41, no. 1, pp 33-39, Jan. 1994.

[13] Xiaoping Huang, Belle W. Y. Wei, Honglu Chen, Yuhai H. Mao, "High-Performance VLSI Multiplier with a New Redundant Binary Coding," *Journal of VLSI Signal Processing*, Vol. 3, pp. 283 - 291, 1991.

[14] Catherine Chow, James Robertson, "Logical Design of A Redundant Binary Adder," *4th Symposium on Computer Arithmetic*, pp. 109 - 115, October 1978.

[15] Shigeo Kuninobu, Tomotsu Nishiyama, Hisakazu Edamatsu, Takashi Taniguchi, Naofumi Takagi, "Design of High Speed MOS Multiplier and Divider Using Redundant Binary Representation," *8th Symposium on Computer Arithmetic*, pp. 80 - 86, 1987.

[16] Milos D. Ercegovac, Tomas Lang, "On Recoding in Arithmetic Algorithms," in *Proceedings of Twentysixth Asilomar Conference on Circuits, Systems, and Computers*, November 1994.