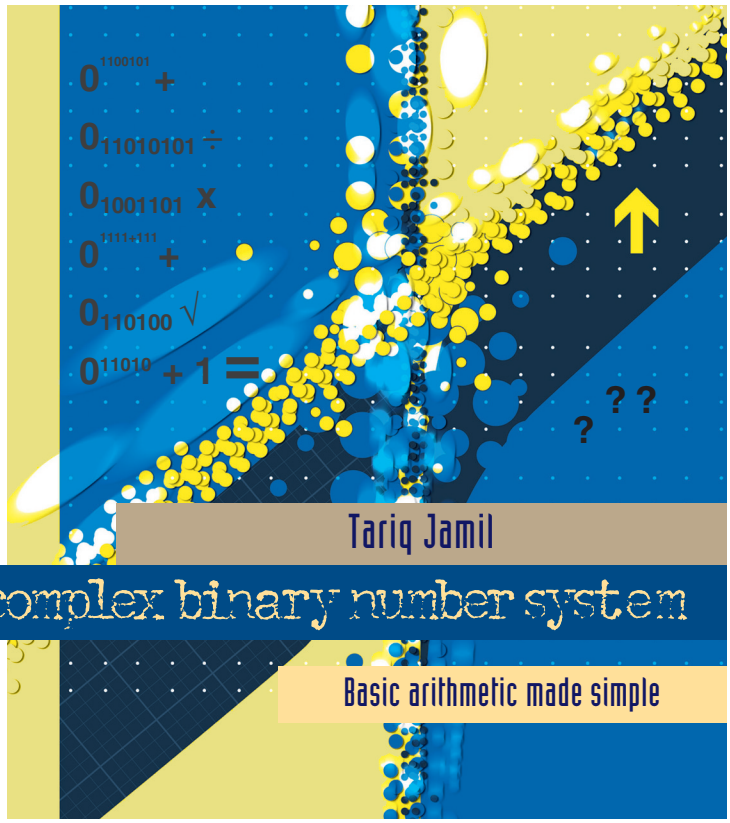

Complex numbers play a unique and very important role in a wide variety of computing applications. History has recorded that in 1545, Cardano initiated the use of $\sqrt{-1}$ during investigation of the roots of polynomials. Later in 1777, the famous mathematician Euler introduced the use of abbreviation i for $\sqrt{-1}$ and originated the $a + ib$ notation to represent complex numbers. (In electrical and computer engineering, we use the symbol j, instead of i, presumably because it is easier to distinguish between the number 1 and j than between 1 and i.)

The Fast Fourier Transform in most digital signal processing algorithms and the geometric analysis of pixels in graphics and image processing owe their advantage to complex numbers. Yet despite their widespread applications in digital signal and image processing, complex numbers are treated as distant relatives of real numbers. Even with 100-million transistors on a single IC-chip, virtually the entire complex arithmetic still involves the "divide-and-conquer" technique. That is, a complex number is broken up into its real and imaginary parts. Operations are then carried out on each part as if it was a part of the real arithmetic. Finally, the overall result of the complex operation is obtained by accumulation of the individual results.

This unequal treatment of complex numbers has long been viewed as unjust by both mathematicians and engineers. This has resulted in efforts to define binary numbers with bases other than 2. In 1960, Donald E. Knuth described a "quater-imaginary" number system with base $2j$ and analyzed the arithmetic operations of numbers using this imaginary base. However, he was unsuccessful in providing a division algorithm. He considered this failure as a main obstacle towards hardware implementation of any imaginary-base number system.

Walter Penney, in 1964, attempted to define a complex number system, first by using a negative base of –4 and then by using a complex number (–1+j) as the base. However, the main problem encountered with using these bases was again the inability to formulate an efficient division process.

Stepanenko utilizes the base $j\sqrt{2}$ in which the even powers of the base yield real numbers and the odd powers of the base result in imaginary numbers. He was partly successful in resolving the division problem as an "all-in-one" operation. However, Stepanenko's algorithm lacks the confirmation that it will always converge in all circumstances.

In this article, we aim to provide a simplified tutorial to the complex binary number system. The examples of arithmetic operations will show that a binary number system with a complex base (such as –1+j) is advantageous for today's computer designs.

For instance, adding two complex numbers (a+jb) and (c+jd) in today's arithmetic requires two separate additions, (a+c) and (b+d), while multiplying the same two complex numbers requires four multiplications (ac), (ad), (bc), (bd), one subtraction (j^2bd=–bd), and one addition (ac+ j(ad+bc)+(–bd)). This process can be effectively reduced to just one complex addition, or one multiplication and one addition respectively for the given cases if each complex number is represented as one unit instead of two individual units.

The Base –1+j

The value of an n–bit binary number with base –1+j can be written in the form of a power series as follows:

$$a_{n-1}(-1+j)^{n-1}+ a_{n-2}(-1+j)^{n-2}+ a_{n-3}(-1+j)^{n-3}+...+ a_2(-1+j)^2 +a_1(-1+j)^1+a_0(-1+j)^0$$

where the coefficients $a_{n-1},a_{n-2},a_{n-3},...,a_2,a_1,a_0$ are binary (either 0 or 1). This is analogous to the ordinary binary number system power series of:

$$a_{n-1}(2)^{n-1} + a_{n-2}(2)^{n-2} + a_{n-3}(2)^{n-3} + ... + a_2(2)^2 +a_1(2)^1 +a_0(2)^0$$

except that the bases are different.

Binary representation for complex numbers

Conversion algorithm for integers. Let's first begin with the case of positive integers N. To represent N in the complex binary number system, we follow these steps:

1. Express N in terms of powers of 4 using division process.

2. Now convert the base 4 number (..., q_5, q_4, q_3, q_2, q_1,q_0) to base –4 by replacing each digit in odd location (q_1, q_3, q_5, ...) with its negative to get(..., $-q_5$, q_4,$-q_3$, q_2, $-q_1$, q_0).

3. Normalize the new number (i.e., get each digit in the range 0 to 3) by repeatedly adding four to the negative digits and adding a one to the digit on its left. If the digit is 4, replace it by a zero and subtract a one from the digit on its left.

4. To represent the given number in the base –1+j, we replace each digit in base –4 representation with the corresponding four bit sequence (0 →0000 ; 1→0001 ; 2 →1100 ; 3 →1101).

Thus,

2000_{base10} = (1,2,0,1,1,0,0) $_{base\ -4}$ = 0001 1100 0000 0001 0001 0000 0000 $_{base\ -1+j}$ = 1 1100 0000 0001 0001 0000 $0000_{base\ -1+j}$

This can be verified by computing the power series:

$(-1+j)^{24} + (-1+j)^{23} + (-1+j)^{22} + (-1+j)^{12} + (-1+j)^8 = 2000$

To convert a negative integer into (–1+j)-base representation, we simply multiply the representation of the correspond-

 0278-6648/01/$17.00 © 2001 IEEE

ing positive integer with 11101 (equivalent to $-1_{base\ -1+j}$) according to the multiplication algorithm (given under arithmetic operations for complex numbers section). Thus

-2000_{base10} = (1 1100 0000 0001 0001 0000 0000) x (11101)

= $1100\ 0000\ 0000\ 1101\ 0000\ 0000_{base\ -1+j}$

This can be verified by computing the power series:

$(-1+j)^{23} + (-1+j)^{22} + (-1+j)^{11} + (-1+j)^{10} + (-1+j)^8 = -2000$

Conversion algorithm for imaginary numbers. To represent a positive or negative imaginary number in (–1+j)-base representation, we multiply the corresponding complex binary representation of the positive or negative integer with 11 (equivalent to $j_{base\ 10}$) or 111 (equivalent to $-j_{base\ 10}$), as appropriate. Thus

$j2000_{base10}$ = (1 1100 0000 0001 0001 0000 0000) x (11)

= $100\ 0000\ 0011\ 0011\ 0000\ 0000_{base\ -1+j}$

And

$-j2000_{base10}$= (11101000000010000) x (111)

= $111\ 0100\ 0000\ 0111\ 0111\ 0000\ 0000_{base\ -1+j}$

Conversion algorithm for fractions. The procedure for finding the binary equivalent in base (–1+j) for real fractions is very similar to the algorithm presented before for the integers. For imaginary fractions, the same procedure, just outlined for the imaginary numbers, can be followed.

As an example, binary representation in base (–1+j) for 0.351_{base10} is obtained as follows:

Step 1: $0.351_{base10} = 0.11213123..._{base\ 4}$

The base 4 representation has been obtained as follows:

0.351x4 = **1**.404; 0.404x4 = **1**.616; 0.616x4 = **2**.464; 0.464x4 = **1**.856; 0.856x4 = **3**.424; 0.424x4 = **1**.696; 0.696x4 = **2**.784; 0.784x4 = **3**.136 and so on...

Step 2: Converting from base 4 to base –4 representation, as before, yields

$0.351_{base10} = 0.(-1)1(-2)1(-3)1(-2)3..._{base\ -4}$

Step 3: After normalization, we have

$0.351_{base10} = 1.32221223..._{base\ -4}$

Step 4: And finally replacing each base –4 digit with its equivalent four-bit sequence gives

0.351_{base10} = 1.1101 1100 1100 1100 0001 1100 1100 1101...$_{base\ -1+j}$

Similarly,

$j0.351_{base10}$ = (1.1101 1100 1100 1100 0001 1100 1100 1101) x (11)

= $0.010001000011010000011001101..._{base\ -1+j}$

Conversion algorithm for floating point numbers. To represent a floating-point positive number in the new base, we add the integer and fractional representations according to the addition rules given in the next section. Once again, all rules for obtaining negative integer and positive/negative imaginary number representations in base (–1+j) are equally applicable for obtaining negative floating point and positive/negative imaginary floating point representations in the proposed new base.

For example, $60.4375_{base\ 10}$

= 1 1 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 + 1 . 1 1 0 1 1 1 0 1 = $11101000000010001.11011101_{base\ -1+j}$

And $j60.4375_{base\ 10}$

= 11101000000010001.11011101 x 11

= $111000000110000.01000111_{base\ -1+j}$

Similarly, $55.6875_{base\ 10}$

= 0001 1101 0000 1101 1101 + 1.11001101

= $0001\ 1101\ 0000\ 1100\ 0000.1100\ 1101_{base\ (-1+j)}$

And $j55.6875_{base\ 10}$

= 0001 1101 0000 1100 0000.1100 1101 x 11

= $111011101000000.00110111_{base\ (-1+j)}$

Having known the conversion algorithms, as described previously, the binary representation for any given complex number can be easily obtained, as shown by the following example:

$(55.6875 + j55.6875)_{base10}$

= $0001\ 1101\ 0000\ 1100\ 0000.1100\ 1101_{base\ (-1+j)}$

+ $111\ 0111\ 0100\ 0000.0011\ 0111_{base\ (-1+j)}$

= $110\ 0000\ 1000\ 0000.1110\ 0110_{base(-1+j)}$

Arithmetic operations

Addition. The binary addition of two complex numbers follows these rules: 0 + 0 = 0; 0 + 1 = 1; 1 + 0 = 1; 1 + 1 = 1100. These rules are very similar to the traditional binary arithmetic. The exception is the last case: when two 1s are added, the sum is zero and (instead of just one carry) two carries are generated. These two carries propagate towards the two adjoining positions after skipping the immediate neighbor of the sum column. That is, if two numbers with 1s in position n are added, this will result in 1s in positions n+3 and n+2 and 0s in positions n+1 and n in the sum. Similar to the ordinary computer rule where 1 + 111 ... (to limit of machine) = 0, we have 11 + 111 = 0 [zero rule].

Subtraction. The binary subtraction of two complex numbers follows these rules: 0 – 0 = 0; 0 – 1 = *; 1 – 0 = 1; 1 – 1 = 0. Three of the four conditions listed in these rules are the same as for subtraction in the ordinary binary system. For the case where 1 is subtracted from 0 (* case in the rules), the following algorithm applies:

Assuming our minuend is $a_n a_{n-1} a_{n-2} ... a_{k+4} a_{k+3} a_{k+2} a_{k+1} a_k \mathbf{0} a_{k-1} .... a_3 a_2 a_1 a_0$

and subtrahend is $b_n b_{n-1} b_{n-2} ... b_{k+4} b_{k+3} b_{k+2} b_{k+1} \mathbf{1} b_{k-1} .... b_3 b_2 b_1 b_0$

Then, the result of subtracting 1 from 0 is obtained by changing

$a_k \rightarrow a_k+1$; $a_{k+1} \rightarrow a_{k+1}$(unchanged); $a_{k+2} \rightarrow a_{k+2}+1$; $a_{k+3} \rightarrow a_{k+3}+1$; $a_{k+4} \rightarrow a_{k+4}+1$and $b_k \rightarrow 0$.

Example: Subtract (1+3j) from 2

Solution: In base (–1+j) notation, we have

2 – (1+3j) = 1100 – 1010 ≡ 0100 – 0010 ≡ 111110 – 000000 (by algorithm) = $111110_{base(-1+j)}$

This can be verified to be equivalent to 1-3j by evaluating the power series:

$(-1 + j)^5 + (-1 + j)^4 + (-1 + j)^3 + (-1 + j)^2 + (-1 + j)^1$

Multiplication. The multiplication process of two complex binary numbers is similar to multiplying two ordinary binary numbers. However, while adding the intermediate results of multiplication, the rules given previously for adding complex numbers should be followed. The zero rule plays an important role in reducing the number of summands resulting from intermediate multiplications.

Example: Multiply (2 – j)(1 + 2j)

Solution: The binary representations of the given complex numbers in base –1+j are

2 – j = 2 + (–j) = 1100 + 111 = $111011_{base\ -1+j}$

1 + 2j = 0001 + 1110100 = $1110101_{base\ -1+j}$

Now $(2 - j)(1 + 2j) = 111011_{base\ -1+j}$ x $1110101_{base\ -1+j}$

Blue-faced **1**s in Box A help us to recognize the pattern 111 + 11 which results in 0 (zero rule).

For verification,

$1100111_{base\ -1+j} = (-1 + j)^6 + (-1 + j)^5 + (-1 + j)^2 + (-1 + j)^1 + (-1 + j)^0 = (4 + 3j)$

Division. The division algorithm is based on determining the reciprocal of the divisor (denominator) and then multiplying it with the dividend (numerator) according to the multipli-

cation algorithm.

Thus

$(a+jb) \div (c+jd) = (a+jb)(c+jd)^{-1} = (a+jb)z$ where $z = w^{-1}$ and $w = c+jd$

We start with our initial approximation of z setting $z_0 = (-1+j)^{-k}$ where k is obtained from the representation of w such that

$$w \equiv \sum_{i=-\infty}^{k} a_i (-1+j)^{-i}$$

in which $a_k \equiv 1$ and $a_i \equiv 0$ for $i > k$.

The successive approximations are then obtained by $z_{i+1} = z_i (2 - wz_i)$. If the values of z do not converge, we correct our initial approximation to $z_0 = j(-1 + j)^{-k}$ which will definitely converge. Having calculated the value of z, we just multiply it with (a+jb) to obtain the result of the division. In the following examples, for the sake of clarity, decimal numbers have been used to explain the converging process of the division algorithm.

Let (a + jb) = 1 + j2, and w = 1 + j3. Our calculations for approximation of $z = w^{-1}$ then begin as follows:

$1 + j3 = 1010_{base\ -1+j} = 1.(-1 + j)^3 + 0.(-1+j)^2 + 1.(-1 + j)^1 + 0.(-1+j)^0 \Rightarrow k = 3$

Therefore,

$z_0=(-1+j)^{-3} = 0.25–j0.25$

$z_1 = 0.125–j0.375$

$z_2 = 0.09375–j0.28125,$

$z_3 = 0.099609375–j0.2988281250$

$z_4 = 0.09999847412–j0.2999954224$

$z_5 = 0.1–j0.3$

$z_6 = 0.1–j0.3$ (converging).

Now,

$0.1–j0.3 = 0.0011110010111100010111100...{}_{base\ -1+j}$

So $(1+j2) \div (1+j3)$

$= (1+j2) \times (1+j3)^{-1}$

$= 1\ 1\ 1\ 0\ 1\ 0\ 1\ _{base\ -1+j} \times 0.0011110010111100010111100...{}_{base\ -1+j}$

$= 1.1111100101111001011110010111 1...{}_{base\ -1+j} = 0.7–j0.1$

As another example, let w = –28–j15, then

–28–j15

$= 11011010011_{base\ -1+j}$

$= 1.(-1+j)^{10}+1.(-1+j)^9+0.(-1+j)^8+1.(-1+j)^7+1.(-1+j)^6+0.(-1+j)^5+1.(-1+j)^4+0.(-1+j)^3+0.(-1+j)^2+1.(-1+j)^1+ 1.(-1+j)^0 \Rightarrow k = 10$

We begin by choosing $z_0 = (-1+j)^{-10} = j0.03125$

$z_1 := –0.02734375+j0.0478515625$

$z_2 := –0.05861282347–j0.0007009506465$

$z_3 := –0.02227897905+j0.05242341786$

$z_4 := –0.07257188256+j0.005664255473$

$z_5 := 0.01375684714+j0.06682774792$

$z_6 := –0.1198139965+j0.1209880304$

$z_7 := 0.1873379177–j0.5740439146$

$z_8 := –4.643184188–j11.58680236$

$z_9 := –4778.731320+j1299.184773$

(not converging)

So we correct our initial approximation to:

$z_0 = j(-1+j)^{-10} = –0.03125$

$z_1:= (0.03515625+j0.0146484375$

$z_2:=(0.02626419066+j0.01577854156$

$z_3:=(0.02775239316+j0.01496276824$

$z_4:=(0.02775050252+j0.01486605276$

$z_5:=(0.02775024777+j0.01486620416$

$z_6:=(0.02775024777+j0.01486620416$

$z_7:=(0.02775024777+j0.01486620416$

$z_8:=(0.02775024777+j0.01486620416$

$z_9:=(0.02775024777+j0.01486620416$

(converging)

The converging value of z_9 can be represented in base (–1 + j). It can then be multiplied with any given complex number to obtain the result of dividing the given complex number by –28 – j15, as in the previous example.

Conclusion

Conversion algorithms and arithmetic procedures for a (–1 + j)-base binary number allows a given complex number to be represented as one unit. This should simplify the operations involving complex numbers in today's microprocessors. With the division process secure, we can implement the usual algorithms for calculating functions and processes such as logarithms, exponentials and trigonometric functions. Currently, work is underway to write Java applets for the algorithms. We are planning to design an arithmetic unit based on the new binary system which will then be implemented using Field Programmable Gate Arrays.

Read more about it

• D. Joyce, "The Origin of Complex Numbers and the notation 'i'." Available:

[http://www.math.toronto.edu/mathnet/questionCorner/complexorigin.html]

• L. Geppert, "The 100-million Transistor IC," *IEEE Spectr.*, pp. 23-60, July 1999.

• D. Knuth, "An Imaginary Number System," *Communications of the ACM*, pp. 245-247, 1960.

• W. Penney, "A Numeral System with a Negative Base," *Mathematics Student Journal*, pp. 1-2, May 1964.

• W. Penney, "A Binary System for Complex Numbers," *Journal of the ACM,* pp. 247-248, Apr. 1965.

• V. Stepanenko, "Computer Arithmetic of Complex Numbers," *Cybernetics and System Analysis,* vol. 32, no. 4, pp. 585-591, 1996.

Box A

```
                1110101
                 111011
====================
                1110101
               1110101
              0000000
             1110101
            1110101
           1110101
 ====================
          000001000111
              11
             11
             11
====================
              1100111
```

About the author

Dr. Tariq Jamil is an Assistant Professor in the Information Engineering Department at Sultan Qaboos University, Oman. He received his B.Sc. (Honors) degree in electrical engineering from the NWFP University of Engineering and Technology, Pakistan, in 1989 and M.S. and Ph.D. degrees in computer engineering from the Florida Institute of Technology in 1992 and 1996, respectively. He can be contacted by e-mail at: <tjamil@squ.edu.om>.