Impact of Shift Operations on (−1+j)-Base Complex Binary Numbers

Tariq Jamil
Department of Electrical and Computer Engineering, Sultan Qaboos University, Al-Khod 123, OMAN
Email: tjamil@squ.edu.om

***Abstract*—Complex numbers play a very important role in various applications of electrical and computer engineering. These days, arithmetic operations dealing with these numbers rely on a "divide-and-conquer" technique wherein a complex number is broken into its real and imaginary parts and then, after representing each part in binary number system, operation is carried out on each part as if part of the real arithmetic. Thus, addition of two complex numbers requires two separate additions, and their multiplication requires four individual multiplications, one subtraction and one addition. In an effort to reduce the number of arithmetic operations within the realm of complex arithmetic, binary number system with base (−1+j), called complex binary number system, has been proposed in the literature which allows a complex number to be represented as single-unit instead of two separate units as in the base-2 binary number system. In this paper, the effects of shift operations on complex binary numbers have been examined and mathematical equations describing their behavior have been obtained. Analysis of these equations leads to the conclusion that the impact of shift operations on a complex binary number is, to a large extent, similar to typical multiply-by-2 (for per-bit shift-left) and divide-by-2 (for per-bit shift-right) operations of traditional base-2 binary number.**

***Index Terms*—complex number, complex binary number, shift-left, shift-right.**

I. Introduction

Complex numbers play a very important role in engineering applications such as digital signal processing and image processing. Thus design of an efficient approach to handle complex arithmetic will result in better performance in such applications. These days, arithmetic operations dealing with complex numbers involve, to a large extent, application of a "divide-and-conquer" technique, whereby a complex number is broken up into its real and imaginary parts and then operations are carried out on each part as if it were part of the real-arithmetic. Finally, the overall result of the complex operation is obtained by accumulation of the individual results. For instance, addition of two complex numbers requires two separate additions (one for the real parts and one for the imaginary parts) while multiplication of the same two complex numbers requires four multiplications, one subtraction, and one addition. Efforts in defining binary numbers with bases other than 2, which facilitate one-unit representation of complex numbers include work by Knuth [1], Penney [2,3], and Stepanenko [4]. Jamil *et. al.* [5,6] have presented a detailed analysis of (−1+j)-base and (−1−j)-base complex binary number systems and elaborated on how addition, subtraction, multiplication, and division of two such complex binary numbers can be accomplished. In this paper, we have investigated the effects of logical shift-left and logical shift-right operations on the binary representation of complex numbers in (−1+j)-base Complex Binary Number System (CBNS) and have obtained mathematical equations describing their behavior when subjected to these operations.

This paper is organized as follows: In Section II, a detailed introduction to (−1+j)-base complex binary number system, along with algorithms for converting a given number into CBNS, has been presented. This is followed, in Section III, by a discussion of the effects of shift-left operations on complex numbers represented in CBNS format. In Section IV, effects of shift-right operations on complex binary numbers are explored. Conclusions and suggestions for future research are presented in Section V, which are followed by Acknowledgment and References.

II. (−1+j)-Base Complex Binary Number System (CBNS)

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

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

where the coefficients $a_{n-1}, a_{n-2}, a_{n-3}, \ldots, a_2, a_1, a_0$ are binary (either 0 or 1).

A. Conversion algorithm for integers

Let's first begin with the case of positive integers *N*. To represent *N* in the (−1+j)-base complex binary number

system, we express N in terms of powers of 4 using division process. Thus

$$N_{base\ 4} = \sum q_i\, 4^i \qquad (2)$$

This "normalized" representation is unique when $0 \le q_i < 4$. In that case the non-zero 'digits' …, $q_5, q_4, q_3, q_2, q_1, q_0$ are called the base 4 representation of N. If the constraint on q_i is removed, then we call it an "un-normalized" base 4 representation of N, which is not unique. 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 $q_1, q_3, q_5, \ldots$ with its negative to get

$(\ldots, q_5, q_4, q_3, q_2, q_1, q_0)_{base\ 4}$
$= (\ldots, -q_5, q_4, -q_3, q_2, -q_1, q_0)_{base\ -4}$ (un-normalized)

We normalize the new number (i.e. get each digit in the range 0 to 3) by repeatedly using the operation of adding 4 to the negative digits and adding a 1 to the digit on its left. This operation will get rid of the negative numbers, but might create some digits with a value of 4 after the addition of a 1. To normalize this, we replace the 4 by a 0 and subtract a 1 from the digit on its left. Of course this subtraction might once again introduce negative digits which will be normalized by the previous method. As an example

2007_{base10}
$= (1,3,3,1,1,3)_{base\ 4} = (-1, 3, -3, 1, -1, 3)_{base\ -4}$
$= (1,3,4,1,2,3,3)_{base\ -4} = (1,2,0,1,2,3,3)_{base\ -4}$ (normalized)

To represent the given number in the base (–1+j), we replace each digit in base –4 representation with the given four bit sequence ($0 \rightarrow 0000$, $1 \rightarrow 0001$, $2 \rightarrow 1100$, $3 \rightarrow 1101$) which yields:

$2007_{base10} = (1,2,0,1,2,3,3)_{base\ -4}$
$= 0001\ 1100\ 0000\ 0001\ 1100\ 1101\ 1101_{base\ -1+j}$
$= 11100000000011100011011101_{base\ -1+j}$

To convert a negative integer into (–1+j)-base representation, we simply multiply the representation of the corresponding positive integer with 11101 (equivalent to $-1_{base\ -1+j}$) according to the multiplication algorithm given in Ref.[5]. Thus

-2007_{base10}
$= (0001\ 1100\ 0000\ 0001\ 1100\ 1101\ 1101) \times (11101)$
$= 1100\ 0000\ 0000\ 1101\ 1100\ 0001_{base\ -1+j}$

B. Conversion algorithm for imaginary numbers

To obtain binary representation of a given positive or negative imaginary number in CBNS, we simply multiply (according to the multiplication algorithm given in Ref.[5]) the corresponding (–1+j)-base representation of positive or negative integer with 11 (equivalent to $j_{base\ 10}$) or 111 (equivalent to $-j_{base\ 10}$), as required. Thus, given

$2007_{base10} = 11100000000011100011011101_{base\ -1+j}$
then

$j2007_{base10} = (11100000000011100011011101) \times (11)$
$= 1000000001100110100011 1_{base\ -1+j}$
And
$-j2007_{base10}$
$= (11100000000011100011011101) \times (111)$
$= 111010000000111010001000011_{base\ -1+j}$

C. Conversion algorithm for fractions

The procedure for finding the binary equivalent for fractions in base (–1+j) is based on the usual approach to obtaining ordinary binary representations. Any fraction F can be expressed uniquely in terms of powers of $½ = 2^{-1}$ such that

$$F = r_0 = f_1.\, 2^{-1} + f_2.\, 2^{-2} + f_3.\, 2^{-3} + f_4.\, 2^{-4} + \ldots \text{to machine limit} \qquad (3)$$

Then the coefficients f_i and the remainders r_i are given as follows:

Initially
if $2r_0 - 1 < 0$ then $f_1 = 0$ and set $r_1 = 2r_0$
or if $2r_0 - 1 \ge 0$ then $f_1 = 1$ and set $r_1 = 2r_0 - 1$
then
if $2r_i - 1 < 0$ then $f_{i+1} = 0$ and $r_{i+1} = 2r_i$
or if $2r_i - 1 \ge 0$ then $f_{i+1} = 1$ and $r_{i+1} = 2r_i - 1$

We continue this process until $r_i = 0$ or the machine limit has been reached. Then, for $\forall f_i = 1$, we replace its associated 2^{-i} according to the sequence ($2^{-1} \rightarrow 1.11$, $2^{-2} \rightarrow 1.1101$, $2^{-3} \rightarrow 0.000011$, $2^{-4} \rightarrow 0.00000001$) [for $i > 4$, refer to Ref.[5]).

As an example
let $F = r_0 = 0.4375_{base10}$

Initially
$2r_0 - 1 = 2(0.4375) - 1 = -0.125 < 0$
$\Rightarrow f_1 = 0,\ r_1 = 2r_0 = 2(0.4375) = 0.875$
Then
$2r_1 - 1 = 2(0.875) - 1 = 0.75 > 0$
$\Rightarrow f_2 = 1,\ r_2 = 2r_1 - 1 = 2(0.875) - 1 = 0.75$
Continuing according to the algorithm, we have
$2r_2 - 1 = 2(0.75) - 1 = 0.5 > 0$
$\Rightarrow f_3 = 1,\ r_3 = 2r_2 - 1 = 2(0.75) - 1 = 0.5$
$2r_3 - 1 = 2(0.5) - 1 = 0$ (STOP)
$\Rightarrow f_4 = 1,\ r_4 = 0$

Thus
$0.4375_{base\ 10} = 0.2^{-1} + 1.\, 2^{-2} + 1.\, 2^{-3} + 1.\, 2^{-4}$
$= 0(1.11) + 1(1.1101) + 1(0.000011) + 1(0.00000001)$
$= 1.11011101_{base\ -1+j}$
(addition according to the algorithm given in Ref.[5])

It is likely that most fractions will not terminate as our example, until the machine limit is reached. For example,

$0.351_{base\ 10} = 1.1101110011001100000111001 10\ldots_{base\ -1+j}$

In that case, it is up to the user to terminate the process when certain degree of accuracy has been achieved.

D. 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 [5]. Once again, all rules for obtaining negative integer and positive/negative imaginary number representations in base (–1+j), as discussed previously, are equally applicable for obtaining negative floating point and positive/negative imaginary floating point representations in the proposed new base.

For example
$2007.4375_{base\ 10}$
= 11100000000001110011011101 + 1.11011101
= $11100000000001110011000000.11011101_{base\ -1+j}$
And
$j2007.4375_{base\ 10}$
= 11100000000001110011000000.11011101x 11
= $1000000001100110100011 1.01000111_{base\ -1+j}$

Having known the conversion algorithms, the binary representation for any given complex number can be easily obtained, as shown by the following example:
$(2007.4375 + j2007.4375)_{base10}$
= $11100000000001110011000000.11011101_{base\ -1+j}$
+ $1000000001100110100011 1.01000111_{base\ -1+j}$
= $11101000000001110100010000000.10000110_{base-1+j}$

III. Shift Left Operations in Complex Binary Number System

To investigate the effects of shift-left (1,2,3, and 4-bits) operations on a complex number represented in CBNS format, a computer program was developed in C++ language which allowed (i) variations in magnitude and sign of both real and imaginary components of a complex number to be generated automatically in a linear fashion, and (ii) decomposition of the complex number after the shift-left operation, represented in CBNS format, into its real and imaginary components[7]. The length of the original binary bit array was restricted to 800 bits and 0s were padded on the left-side of the binary data when the given complex number required less than maximum allowable bits for representation in CBNS format. As an example, consider the following complex number:

Before padding
$90+j90_{base10} = 110100010001000_{base-1+j}$
After padding
$90+j90_{base10} = 0...0110100010001000_{base-1+j}$
such that the total size of binary array is 800 bits. Shifting this binary array by 1-bit to the left will yield $0\ldots0110100010001000\underline{\mathbf{0}}_{base-1+j}$ ensuring that total array-size remains 800 bits by removing one 0 from the left-side and appending it to the right-side of the number. Similarly, shifting of the original binary array by 2,3, or 4-bits to the left will yield $0011010001000100\underline{\mathbf{00}}_{base-1+j}$ (notice two 0s appended on the right-side of the array),$0...0110100010001000\underline{\mathbf{000}}_{base-1+j}$ (notice three 0s appended on the right-side of the array), $0\ldots0110100010001000\underline{\mathbf{0000}}_{base-1+j}$ (notice four 0s appended on the right-side of the array), ensuring all the time that total array-size remains 800 bits by removing two 0s, three 0s, and four 0s respectively from the left-side of the original array.

Table I presents an overall summary of the effect on the signs of the complex numbers, represented in CBNS format, because of 1,2,3, and 4-bit shift-left operations.

TABLE I
EFFECT ON SIGNS OF COMPLEX NUMBERS IN CBNS FORMAT AFTER SHIFT-LEFT OPERATIONS

Before Shift-Left		After Shift-Left (1-bit)	
Real	Imaginary	Real	Imaginary
+	0	–	+
–	0	+	–
0	+	–	–
0	–	+	+
+	+	–	0
+	–	0	+
–	+	0	–
–	–	+	0
Before Shift-Left		**After Shift-Left (2-bits)**	
Real	Imaginary	Real	Imaginary
+	0	0	–
–	0	0	+
0	+	+	0
0	–	–	0
+	+	+	–
+	–	–	–
–	+	+	+
–	–	–	+
Before Shift-Left		**After Shift-Left (3-bits)**	
Real	Imaginary	Real	Imaginary
+	0	+	+
–	0	–	–
0	+	–	+
0	–	+	–
+	+	0	+
+	–	+	0
–	+	–	0
–	–	0	–
Before Shift-Left		**After Shift-Left (4-bits)**	
Real	Imaginary	Real	Imaginary
+	0	–	0
–	0	+	0
0	+	0	–
0	–	0	+
+	+	–	–
+	–	–	+
–	+	+	–
–	–	+	+

To find out the effect of shift-left operation on the magnitudes of the complex numbers, we chose to vary the magnitude of real and imaginary components of the original complex numbers in a linear fashion (Fig. 1) and for the complex numbers obtained after the shift-left operation, we obtained mathematical equations describing their behavior, as given in Figs. 2-5. To fully understand the variations in the sign and magnitude of the complex numbers before and after the shift-left operations, graphs were drawn in Microsoft Excel as shown in the Figs 6-13.

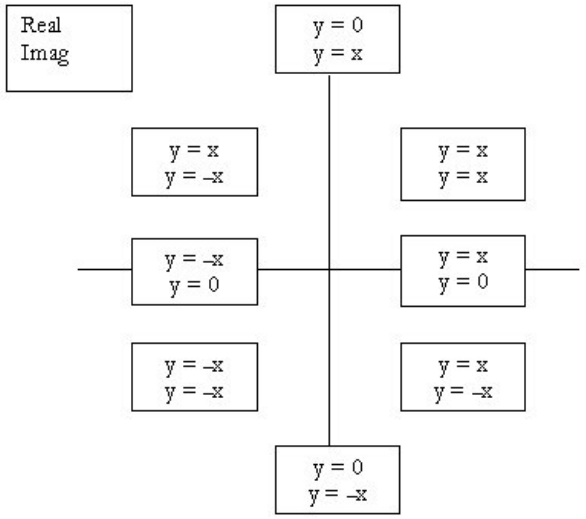

Figure 1. Mathematical equations describing variations in magnitudes of complex numbers in the Cartesian plane (before shift-left or shift-right)

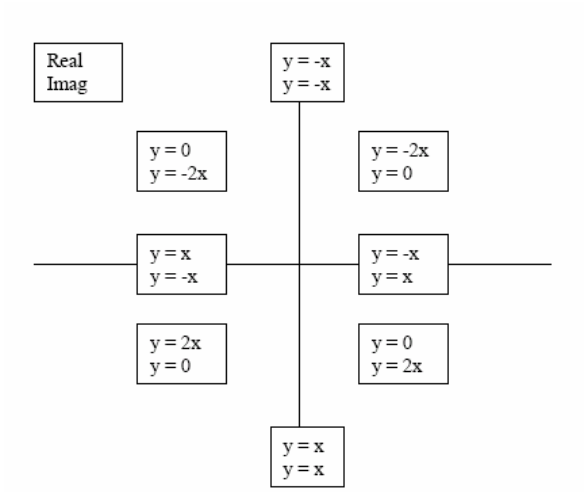

Figure 2. Mathematical equations describing variations in magnitudes of complex numbers in the Cartesian plane (after shift-left by 1-bit)

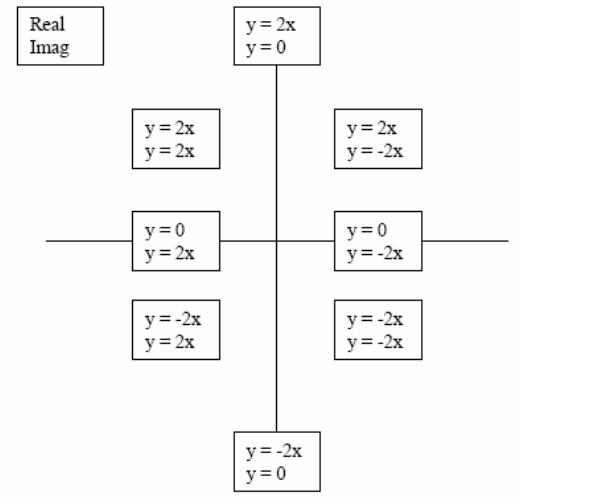

Figure 3. Mathematical equations describing variations in magnitudes of complex numbers in the Cartesian plane (after shift-left by 2-bits)

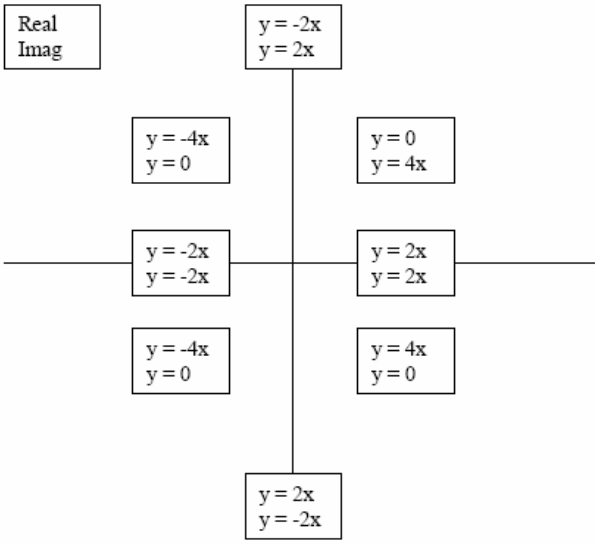

Figure 4. Mathematical equations describing variations in magnitudes of complex numbers in the Cartesian plane (after shift-left by 3-bits)

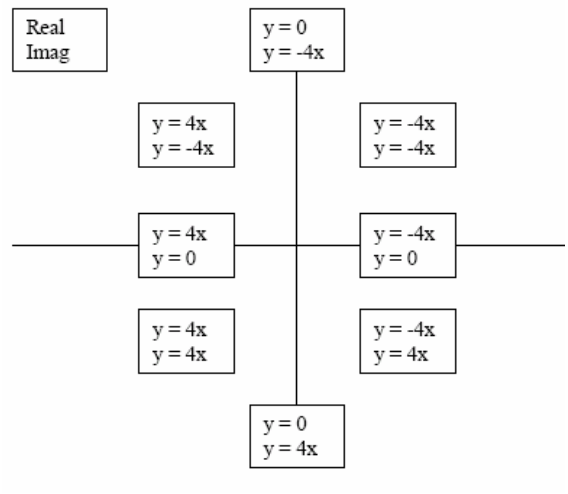

Figure 5. Mathematical equations describing variations in magnitudes of complex numbers in the Cartesian plane (after shift-left by 4-bits)

Assuming positive real-only complex numbers (no imaginary part) represented in CBNS format before the shift, the effect on the real and imaginary parts of the complex numbers after the 1,2,3,4-bits shift-left operations are shown in Fig. 6. The case of negative real-only complex number (no imaginary part) is shown in Fig. 7.

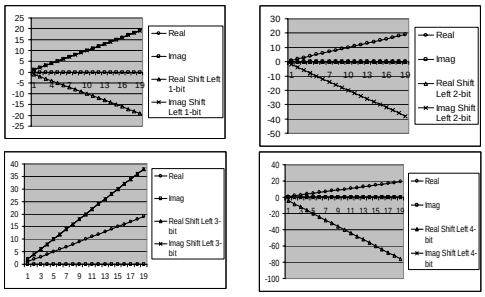

Figure 6. Effects of shift-left operations on sign and magnitude of a positive real-only complex number (1,2,3,4-bits)

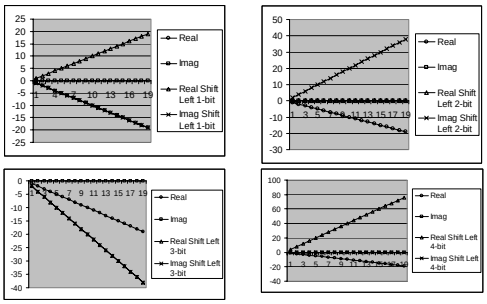

Figure 7. Effects of shift-left operations on sign and magnitude of a negative real-only complex number (1,2,3,4-bits)

The cases of positive and negative imaginary-only complex numbers (no real part) are presented in Figs. 8-9 respectively.

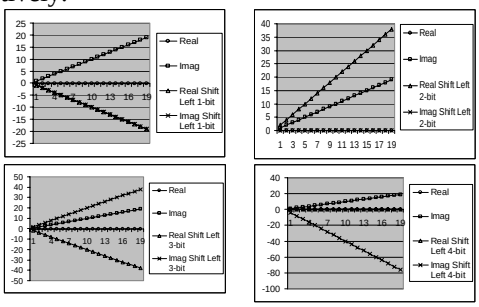

Figure 8. Effects of shift-left operations on sign and magnitude of a positive imaginary-only complex number (1,2,3,4-bits)

The four cases of ±Real±Imaginary complex numbers represented in CBNS format before the shift, and effects of 1,2,3,4-bits shift-left operations on the sign and magnitude of complex numbers are presented in Figs 10-13 respectively.

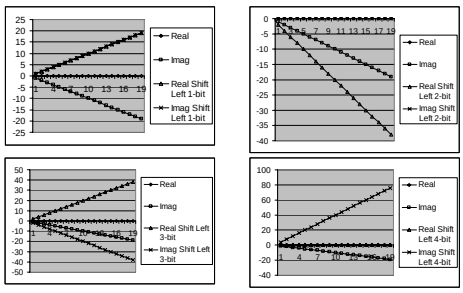

Figure 9. Effects of shift-left operations on sign and magnitude of a negative imaginary-only complex number (1,2,3,4-bits)

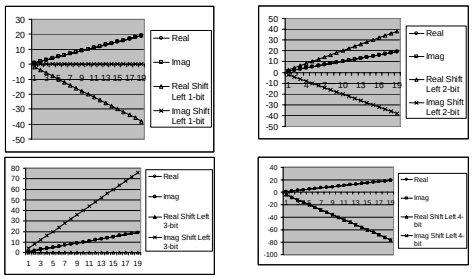

Figure 10. Effects of shift-left operations on sign and magnitude of a +Real+Imaginary complex number (1,2,3,4-bits)

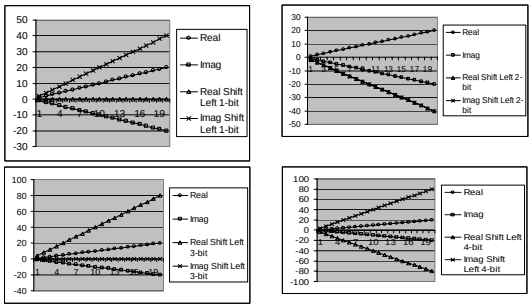

Figure 11. Effects of shift-left operations on sign and magnitude of a +Real–Imaginary complex number (1,2,3,4-bits)

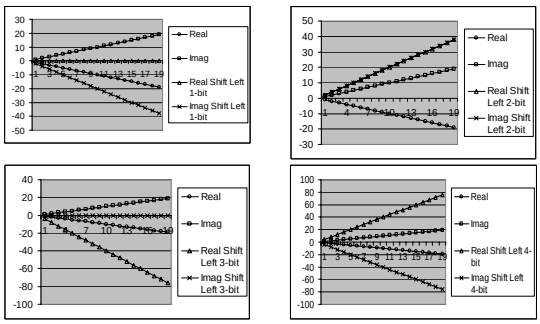

Figure 12. Effects of shift-left operations on sign and magnitude of a –Real+Imaginary complex number (1,2,3,4-bits)

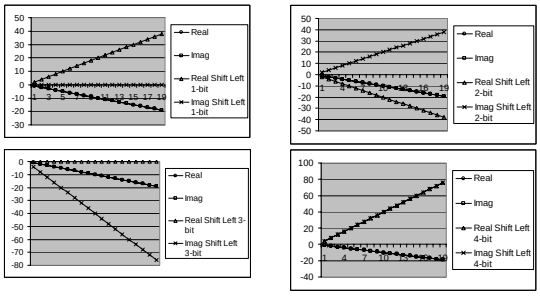

Figure 13. Effects of shift-left operations on sign and magnitude of a –Real–Imaginary complex number (1,2,3,4-bits)

Characteristic equations describing the behavior of complex numbers in CBNS format after shift-left operations are given in Table II.

IV. Shift Right Operations in Complex Binary Number System

To investigate the effects of shift-right (1,2,3, and 4-bits) operations on a complex number represented in CBNS format, a computer program was developed in C++ language which allowed (i) variations in magnitude and sign of both real and imaginary components of a complex number to be generated automatically in a linear fashion, and (ii) decomposition of the complex number after the shift-right operation, represented in CBNS format, into its real and imaginary components[8]. The length of the original binary bit array was restricted to 800 bits and 0s were padded on the left-side of the binary data when the given complex number required less than maximum allowable bits for representation in CBNS format. As an example, consider the following complex number:

TABLE II
CHARACTERISTIC EQUATIONS DESCRIBING COMPLEX NUMBERS IN CBNS FORMAT AFTER SHIFT-LEFT OPERATIONS

Type of Complex Number	Shift-Left by 1-bit		Shift-Left by 2-bits		Shift-Left by 3-bits		Shift-Left by 4-bits	
	$Real_{new}$	$Imag_{new}$	$Real_{new}$	$Imag_{new}$	$Real_{new}$	$Imag_{new}$	$Real_{new}$	$Imag_{new}$
Positive Real Only	$-Real_{old}$	$+Real_{old}$	0	$-2Real_{old}$	$+2Real_{old}$	$+2Real_{old}$	$-4Real_{old}$	0
Negative Real Only	$-Real_{old}$	$+Real_{old}$	0	$-2Real_{old}$	$+2Real_{old}$	$+2Real_{old}$	$-4Real_{old}$	0
Positive Imag Only	$-Imag_{old}$	$-Imag_{old}$	$+2Imag_{old}$	0	$-2Imag_{old}$	$+2Imag_{old}$	0	$-4Imag_{old}$
Negative Imag Only	$-Imag_{old}$	$-Imag_{old}$	$+2Imag_{old}$	0	$-2Imag_{old}$	$+2Imag_{old}$	0	$-4Imag_{old}$
+Real +Imag	$-2Real_{old}$	0	$+2Real_{old}$	$-2Imag_{old}$	0	$+4Imag_{old}$	$-4Real_{old}$	$-4Imag_{old}$
+Real –Imag	0	$-2Imag_{old}$	$-2Real_{old}$	$+2Imag_{old}$	$+4Real_{old}$	0	$-4Real_{old}$	$-4Imag_{old}$
–Real +Imag	0	$-2Imag_{old}$	$-2Real_{old}$	$+2Imag_{old}$	$+4Real_{old}$	0	$-4Real_{old}$	$-4Imag_{old}$
–Real –Imag	$-2Real_{old}$	0	$+2Real_{old}$	$-2Imag_{old}$	$+4Real_{old}$	0	$-4Real_{old}$	$-4Imag_{old}$

Before padding
$90+j90_{base10} = 110100010001000_{base-1+j}$
After padding
$90+j90_{base10} = 0...0110100010001000_{base-1+j}$
such that the total size of binary array is 800 bits. Shifting this binary array by 1-bit to the right will yield **0**0…011010001000100 $_{base-1+j}$ ensuring that total array-size remains 800 bits by removing one 0 from the right-side and inserting one 0 on the left-side of the number. Similarly, shifting of the original binary array by 2,3, or 4-bits to the right will yield **00**0…01101000100010$_{base-1+j}$ (notice two 0s inserted on the left-side of the array), **000**0…0110100010001$_{base-1+j}$ (notice three 0s inserted on the left-side of the array), **0000**0…011010001000$_{base-1+j}$ (notice four 0s inserted on the left-side of the array), ensuring all the time that total array-size remains 800 bits by removing two 0s, three 0s, and four 0s respectively from the right-side of the original array. Table III presents an overall summary of the effect on the signs of the complex numbers, represented in CBNS format, because of 1,2,3, and 4-bit shift-right operations.

To find out the effect of shift-right operation on the magnitudes of the complex numbers, we chose to vary the magnitude of real and imaginary components of the original complex numbers in a linear fashion (Fig. 1) and for the complex numbers obtained after the shift-right operation, we obtained mathematical equations describing their behavior, as given in Figs. 14-17. To fully understand the variations in the sign and magnitude of the complex numbers before and after the shift-right operations, graphs were drawn in Microsoft Excel as shown in the Figs. 18-21.

Assuming positive real-only complex numbers (no imaginary part) represented in CBNS format before the shift, the effect on the real and imaginary parts of the complex numbers after the 1,2,3,4-bits shift-right operations is shown in Fig. 18.

TABLE III
EFFECT ON SIGNS OF COMPLEX NUMBERS IN CBNS FORMAT AFTER SHIFT-RIGHT OPERATIONS

Before Shift-Right		After Shift-Right (1-bit)	
Real	Imaginary	Real	Imaginary
+	0	–	–
–	0	+	+
0	+	+	–
0	–	–	+
+	+	0	–
+	–	–	0
–	+	+	0
–	–	0	+
Before Shift-Right		**After Shift-Right (2-bits)**	
Real	Imaginary	Real	Imaginary
+	0	0	+
–	0	0	–
0	+	–	0
0	–	+	0
+	+	–	+
+	–	+	+
–	+	–	–
–	–	+	–
Before Shift-Right		**After Shift-Right (3-bits)**	
Real	Imaginary	Real	Imaginary
+	0	+	–
–	0	–	+
0	+	+	+
0	–	–	–
+	+	+	0
+	–	0	–
–	+	0	+
–	–	0	+
Before Shift-Right		**After Shift-Right (4-bits)**	
Real	Imaginary	Real	Imaginary
+	0	–	0
–	0	+	0
0	+	0	–
0	–	0	+
+	+	–	–
+	–	–	+
–	+	+	–
–	–	–	0

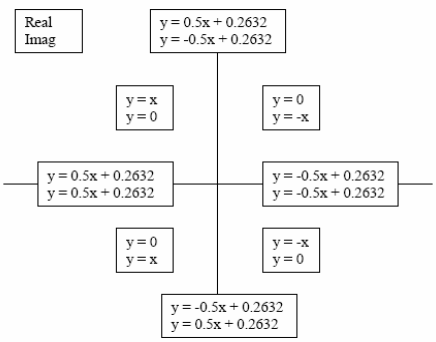

Figure 14. Mathematical equations describing variations in magnitudes of complex numbers in the Cartesian plane (after shift-right by 1-bit)

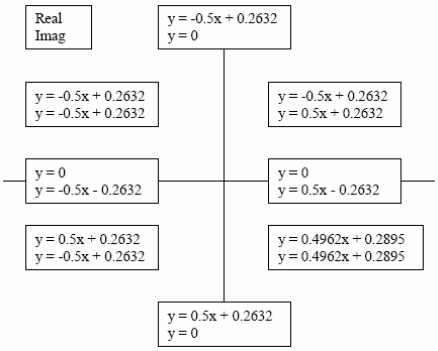

Figure 15. Mathematical equations describing variations in magnitudes of complex numbers in the Cartesian plane (after shift-right by 2-bits)

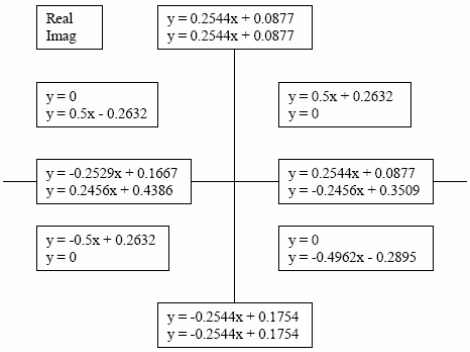

Figure 16. Mathematical equations describing variations in magnitudes of complex numbers in the Cartesian plane (after shift-right by 3-bits)

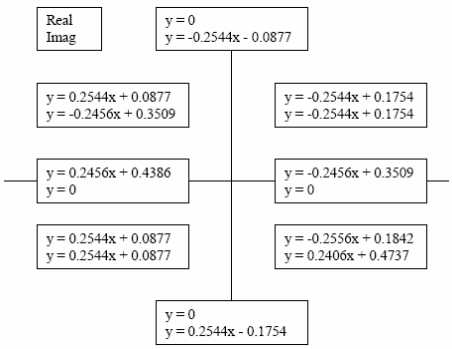

Figure 17. Mathematical equations describing variations in magnitudes of complex numbers in the Cartesian plane (after shift-right by 4-bits)

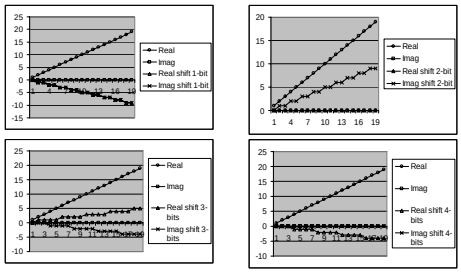

Figure 18. Effects of shift-right operations on sign and magnitude of a positive real-only complex number (1,2,3,4-bits)

The case of negative real-only complex number (no imaginary part) is shown in Fig. 19.

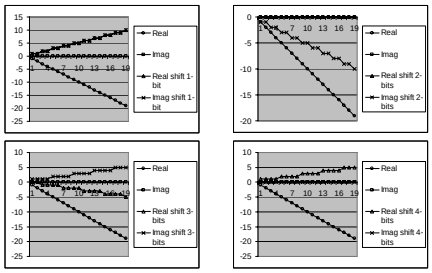

Figure 19. Effects of shift-right operations on sign and magnitude of a negative real-only complex number (1,2,3,4-bits)

The cases of positive and negative imaginary-only complex numbers (no real part) are presented in Figs. 20-21 respectively.

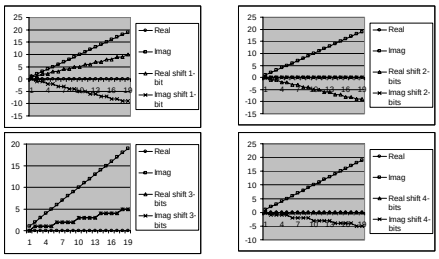

Figure 20. Effects of shift-right operations on sign and magnitude of a positive imaginary-only complex number (1,2,3,4-bits)

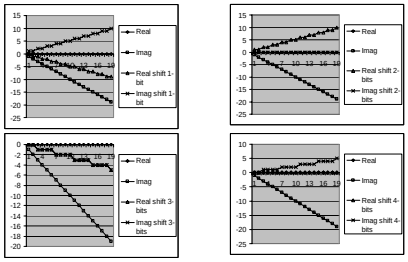

Figure 21. Effects of shift-right operations on sign and magnitude of a negative imaginary-only complex number (1,2,3,4-bits)

The four cases of ±Real±Imaginary complex numbers represented in CBNS format before the shift, and effects of 1,2,3,4-bits shift-right operations on the sign and magnitude of complex numbers are presented in Figs 22-25 respectively.

TABLE IV
CHARACTERISTIC EQUATIONS DESCRIBING COMPLEX NUMBERS IN CBNS FORMAT AFTER SHIFT-RIGHT OPERATIONS

Type of Complex Number	Shift-Right by 1-bit		Shift-Right by 2-bits		Shift-Right by 3-bits		Shift-Right by 4-bits	
	$Real_{new}$	$Imag_{new}$	$Real_{new}$	$Imag_{new}$	$Real_{new}$	$Imag_{new}$	$Real_{new}$	$Imag_{new}$
Positive Real Only	$-\frac{1}{2}Real_{old} + \frac{1}{4}$	$-\frac{1}{2}Real_{old} + \frac{1}{4}$	0	$+\frac{1}{2}Real_{old} - \frac{1}{4}$	$\frac{1}{4}\ Real_{old}$	$-\frac{1}{4}\ Real_{old}$	$-\frac{1}{4}Real_{old}$	0
Negative Real Only	$-\frac{1}{2}Real_{old} + \frac{1}{4}$	$-\frac{1}{2}Real_{old} + \frac{1}{4}$	0	$+\frac{1}{2}Real_{old} - \frac{1}{4}$	$\frac{1}{4}\ Real_{old}$	$-\frac{1}{4}\ Real_{old}$	$-\frac{1}{4}Real_{old}$	0
Positive Imag Only	$\frac{1}{2}Imag_{old} + \frac{1}{4}$	$-\frac{1}{2}Imag_{old} + \frac{1}{4}$	$-\frac{1}{2}Imag_{old} + \frac{1}{4}$	0	$\frac{1}{4}\ Imag_{old}$	$\frac{1}{4}\ Imag_{old}$	0	$-\frac{1}{4}Imag_{old}$
Negative Imag Only	$\frac{1}{2}Imag_{old} + \frac{1}{4}$	$-\frac{1}{2}Imag_{old} + \frac{1}{4}$	$-\frac{1}{2}Imag_{old} + \frac{1}{4}$	0	$\frac{1}{4}\ Imag_{old}$	$\frac{1}{4}\ Imag_{old}$	0	$-\frac{1}{4}Imag_{old}$
+Real +Imag	0	$-Imag_{old}$	$-\frac{1}{2}Real_{old} + \frac{1}{4}$	$\frac{1}{2}Imag_{old} + \frac{1}{4}$	$\frac{1}{2}Real_{old} + \frac{1}{4}$	0	$-\frac{1}{4}Real_{old}$	$-\frac{1}{4}Imag_{old}$
+Real –Imag	$-Real_{old}$	0	$\frac{1}{2}Real_{old} + \frac{1}{4}$	$\frac{1}{2}Imag_{old} + \frac{1}{4}$	0	$\frac{1}{2}Imag_{old} - \frac{1}{4}$	$-\frac{1}{4}Real_{old}$	$-\frac{1}{4}Imag_{old}$
–Real +Imag	$Real_{old}$	0	$\frac{1}{2}Real_{old} + \frac{1}{4}$	$-\frac{1}{2}Imag_{old} + \frac{1}{4}$	0	$\frac{1}{2}Imag_{old} - \frac{1}{4}$	$-\frac{1}{4}Real_{old}$	$-\frac{1}{4}Imag_{old}$
–Real –Imag	0	$-Imag_{old}$	$-\frac{1}{2}Real_{old} + \frac{1}{4}$	$\frac{1}{2}Imag_{old} + \frac{1}{4}$	$\frac{1}{2}Real_{old} + \frac{1}{4}$	0	$-\frac{1}{4}Real_{old}$	$-\frac{1}{4}Imag_{old}$

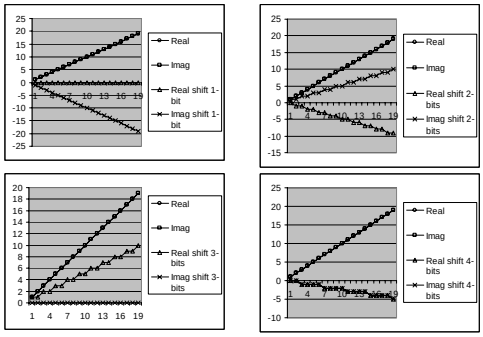

Figure 22. Effects of shift-right operations on sign and magnitude of a +Real+Imaginary complex number (1,2,3,4-bits)

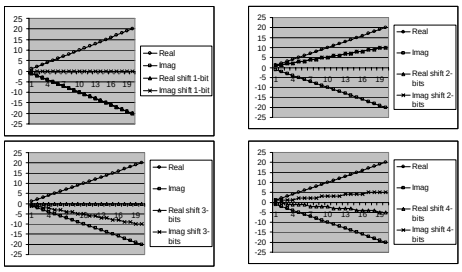

Figure 23. Effects of shift-right operations on sign and magnitude of a +Real–Imaginary complex number (1,2,3,4-bits)

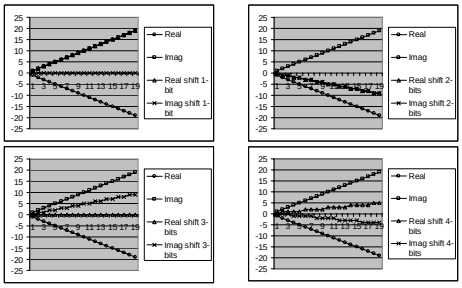

Figure 24. Effects of shift-right operations on sign and magnitude of a –Real+Imaginary complex number (1,2,3,4-bits)

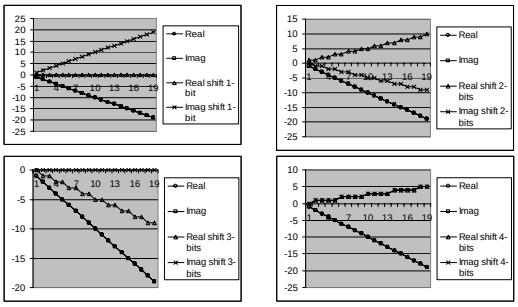

Figure 25. Effects of shift-right operations on sign and magnitude of a –Real–Imaginary complex number (1,2,3,4-bits)

Characteristic equations describing the behavior of complex numbers in CBNS format after shift-right operations are given in Table IV.

V. CONCLUSIONS AND FUTURE RESEARCH

Complex binary number system holds a very promising future because of its ease in representing complex numbers and in simplifying arithmetic operations involving such numbers. With the design of arithmetic algorithms[5], formulation of characteristics equations describing effects of multiple-bit shift-left [7] and shift-right operations [8], and hardware implementations of arithmetic circuits [9,10,11,12], CBNS provides a new avenue of research in the realm of computer arithmetic.

For future research in CBNS, it is highly desirable to design and implement an Arithmetic Unit (AU) incorporating all the arithmetic operations (addition, subtraction, multiplication, division, shift-left, and shift-right) defined so far, and then to compare the performance of ordinary base-2 AU with the newly designed CBNS AU.

ACKNOWLEDGMENT

The author gratefully acknowledges the support provided by Sultan Qaboos University (Oman) for research on complex binary number system through Internal Research Grant Nos. IG/ENG/IENG/01/01, IG/ENG/ECED/04/03, and IG/ENG/ECED/06/02.

REFERENCES

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

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

[3] W. Penney,"A Binary System for Complex Numbers", *Journal of the ACM*, April 1965, pp. 247-248.

[4] V. Stepanenko, "Computer Arithmetic of Complex Numbers", *Cybernetics and System Analysis*, 1996, Vol. 32, No. 4, pp. 585-591.

[5] Tariq Jamil, N. Holmes, and David Blest, "Towards Implementation of a Binary Number System for Complex Numbers", *Proceedings of the IEEE SoutheastCon 2000*, Nashville, Tennessee (USA), April 7-9, 2000, pp. 268-274.

[6] Tariq Jamil and David Blest, "Analysis and Implementation of (−1−j)-base Complex Binary Number System", *Proceedings of the International Conference on Communication, Computer, and Power*, Muscat (Oman), 12-14 February 2001, pp. 76-81.

[7] Tariq Jamil and Usman Ali, "An Investigation into the Effects of Multiple-bit Shift-Left Operations on (−1+j)-base Representation of Complex Numbers," *Proceedings of the IEEE International Conference on Computer and Communication Engineering (ICCCE'06)*, Kuala Lumpur (Malaysia), May 9-11, 2006, Vol. I, pp. 549-554.

[8] Tariq Jamil and Usman Ali, "Effects of Multiple-bit Shift-Right Operations on Complex Binary Numbers," *Proceedings of the IEEE SoutheastCon 2007*, Richmond, Virginia (USA), March 22-25, 2007, pp. 759-764.

[9] Tariq Jamil, Bassel Arafeh, and Amer AlHabsi, "Hardware Implementation and Performance Evaluation of Complex Binary Adder Designs," *Proceedings of the 7th World Multiconference on Systemics, Cybernetics, and Informatics*, Orlando, Florida (USA), July 27-30, 2003, Vol. II, pp. 68-73.

[10] Johnny Goode, Tariq Jamil, and Dale Callahan, "A Simple Circuit for Adding Complex Numbers," *WSEAS Transactions on Information Science and Applications*, Vol. 1, No. 1, July 2004, pp. 61-66.

[11] Tariq Jamil, Amir Arshad Abdulghani, and Ahmad Al-Maashari, "Design of a Nibble-size Subtractor for (−1+j)-base Complex Binary Numbers," *WSEAS Transactions on Circuits and Systems*, Vol. 3, No. 5, July 2004, pp. 1067-1072.

[12] Tariq Jamil, Ahmad Al-Maashari, and Amir Arshad Abdulghani, "Design and Implementation of a Nibble-size Multiplier for (−1+j)-base Complex Binary Numbers," *WSEAS Transactions on Circuits and Systems*, Vol. 4, No. 11, November 2005, pp. 1539-1544.

Tariq Jamil is an assistant professor in the Department of Electrical and Computer Engineering at Sultan Qaboos University (Oman) where he teaches and does research in the areas of computer architecture, parallel processing, computer arithmetic, and cryptography. Before joining the faculty at SQU in year 2000, he had been a lecturer at the University of New South Wales, Sydney (Australia) and the University of Tasmania, Launceston (Australia).

He holds a B.Sc. (Honors) degree in electrical engineering from the NWFP University of Engineering and Technology (Pakistan) obtained in 1989 and M.S. and Ph.D. degrees in computer engineering from the Florida Institute of Technology (USA) obtained in 1992 and 1996 respectively. He has authored several publications in refereed international conferences and journals and has been a recipient of research grants from the Australian Research Council and the SQU. He has also been a visiting professor to COMSATS Institute of Information Technology, Abbottabad (Pakistan) during 2004 and has delivered an invited lecture on Complex Binary Number System at System-on-Chip (SoC) seminar, held in Tampere (Finland) in November 2002. On account of his outstanding academic achievements and for contributions to activities related to the computing discipline, he was awarded the IEEE Computer Society/Upsilon Pi Epsilon Honor Society Award for Academic Excellence (1996) and his biography has been published in such renowned directories as Marquis's Who's Who in the World (USA), Who's Who in Finance and Industry (USA), Who's Who in Science and Engineering (USA), Dictionary of International Biography (UK), and Outstanding People of the 20th Century (UK).

Dr. Jamil is a senior member of IEEE (USA) and a member of IEEE Computer Society (USA), IET (UK), a Chartered Engineer (UK), and a registered Professional Engineer (Pakistan). He has been Student Editor of IEEE Potentials (1995-1997), an IEEE Student Branch Counselor and/or ACM Student Chapter Sponsor at various universities in Pakistan (1997), Australia (1997-2000), and Oman (2001-2003).