

Chapter 2

Conversion Algorithms

Abstract In this chapter, algorithms for conversion of complex numbers into complex binary number system (CBNS) will be described. We'll start with integers, then explain how fractional numbers can be converted into CBNS, and finally how to represent floating point numbers into the new number system. Along the way, we'll also describe how imaginary numbers can be converted into CBNS. Once the algorithms for conversion of real and imaginary parts of a complex number (whether integer, fraction, or floating point) are known, we'll describe how a given complex number can be represented as single-unit binary string consisting of 0 and 1s.

2.1 Conversion Algorithms for Integers

Let's first begin with the case of a positive integer N (in decimal number system) [1, 2]. To represent N in CBNS, we follow these steps:

- (i) Express N in terms of powers of 4 using the repeated division process. That is, repeatedly divide N by 4 keeping track of the remainders.

Examples:

$$2012_{10} = (1, 3, 3, 1, 3, 0)_{\text{Base } 4}$$

$$2000_{10} = (1, 3, 3, 1, 0, 0)_{\text{Base } 4}$$

$$60_{10} = (3, 3, 0)_{\text{Base } 4}$$

- (ii) Now convert the Base 4 number $(\dots n_5, n_4, n_3, n_2, n_1, n_0)$ to Base -4 by replacing each digit in the odd location $(n_1, n_3, n_5, \dots)$ with its negative to get $(\dots -n_5, n_4, -n_3, n_2, -n_1, n_0)$.

Examples:

$$2012_{10} = (-1, 3, -3, 1, -3, 0)_{\text{Base } -4}$$

Table 2.1 Equivalence between normalized base -4 and $(-1+j)$ -base CBNS representation [1]

Normalized base -4	CBNS representation base $(-1+j)$
0	0000
1	0001
2	1100
3	1101

$$2000_{10} = (-1, 3, -3, 1, 0, 0)_{\text{Base } -4}$$

$$60_{10} = (3, -3, 0)_{\text{Base } -4}$$

- (iii) Next, we normalize the new number, i.e., get each digit in the range 0–3, by repeatedly adding 4 to the negative digits and adding a 1 to the digit on its left. This operation will get rid of negative numbers but may create some digits with a value of 4 after the addition of a 1. To normalize this, we replace 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 but this process will definitely terminate. What is interesting to note is that, with negative bases, all integers (positive or negative) have a unique positive representation.

Examples:

$$\begin{aligned} 2012_{10} &= (-1, 3, -3, 1, -3, 0)_{\text{Base } -4} \\ &= (1, 3, 4, 1, 2, 1, 0) = (1, 2, 0, 1, 2, 1, 0)_{\text{Normalized}} \end{aligned}$$

$$\begin{aligned} 2000_{10} &= (-1, 3, -3, 1, 0, 0)_{\text{Base } -4} \\ &= (1, 3, 4, 1, 1, 0, 0) = (1, 2, 0, 1, 1, 0, 0)_{\text{Normalized}} \end{aligned}$$

$$\begin{aligned} 60_{10} &= (3, -3, 0)_{\text{Base } -4} \\ &= (4, 1, 0) = (-1, 0, 1, 0) = (1, 3, 0, 1, 0)_{\text{Normalized}} \end{aligned}$$

- (iv) Lastly, we replace each digit in the normalized representation by its equivalent binary representation in CBNS, as per Table 2.1.

These equivalences can be verified to be correct by calculating the power series for each CBNS representation as follows:

$$0000 = 0 \times (-1+j)^3 + 0 \times (-1+j)^2 + 0 \times (-1+j)^1 + 0 \times (-1+j)^0$$

$$0001 = 0 \times (-1+j)^3 + 0 \times (-1+j)^2 + 0 \times (-1+j)^1 + 1 \times (-1+j)^0$$

$$1100 = 1 \times (-1+j)^3 + 1 \times (-1+j)^2 + 0 \times (-1+j)^1 + 0 \times (-1+j)^0$$

$$1101 = 1 \times (-1+j)^3 + 1 \times (-1+j)^2 + 0 \times (-1+j)^1 + 1 \times (-1+j)^0$$

Examples:

$$\begin{aligned} 2012_{10} &= (1, 2, 0, 1, 2, 1, 0)_{\text{Normalized}} \\ &= 0001 \ 1100 \ 0000 \ 0001 \ 1100 \ 0001 \ 0000 \\ &= 1110000000001110000010000_{\text{Base } (-1+j)} \end{aligned}$$

$$\begin{aligned}
2000_{10} &= (1, 2, 0, 1, 1, 0, 0)_{\text{Normalized}} \\
&= 0001\ 1100\ 0000\ 0001\ 0001\ 0000\ 0000 \\
&= 1110000000001000100000000_{\text{Base } (-1+j)} \\
60_{10} &= (1, 3, 0, 1, 0)_{\text{Normalized}} \\
&= 0001\ 1101\ 0000\ 0001\ 0000 \\
&= 11101000000010000_{\text{Base } (-1+j)}
\end{aligned}$$

To convert a negative integer into CBNS format, we simply multiply the representation of the corresponding positive integer with 11101 (equivalent to $(-1)_{\text{Base } (-1+j)}$) according to the multiplication algorithm given in [Chap. 3](#). Thus,

$$\begin{aligned}
-2012_{10} &= 1110000000001110000010000 \times 11101 \\
&= 110000000000110111010000_{\text{Base } (-1+j)} \\
-2000_{10} &= 1110000000001000100000000 \times 11101 \\
&= 110000000000110100000000_{\text{Base } (-1+j)} \\
-60_{10} &= 11101000000010000 \times 11101 \\
&= 1000111010000_{\text{Base } (-1+j)}
\end{aligned}$$

To obtain binary representation of a positive or negative imaginary number in CBNS, we multiply the corresponding CBNS representation of positive or negative integer with 11 (equivalent to $(+j)_{\text{Base } (-1+j)}$) or 111 (equivalent to $(-j)_{\text{Base } (-1+j)}$) according to the multiplication algorithm given in [Chap. 3](#). Thus,

$$\begin{aligned}
+j2012_{10} &= 1110000000001110000010000 \times 11 \\
&= 10000000000010000110000_{\text{Base } (-1+j)} \\
-j2012_{10} &= 1110000000001110000010000 \times 111 \\
&= 111010000000111010001110000_{\text{Base } (-1+j)} \\
+j2000_{10} &= 1110000000001000100000000 \times 11 \\
&= 10000000011001100000000_{\text{Base } (-1+j)} \\
-j2000_{10} &= 1110000000001000100000000 \times 111 \\
&= 111010000000111011100000000_{\text{Base } (-1+j)} \\
+j60_{10} &= 11101000000010000 \times 11 \\
&= 111000000110000_{\text{Base } (-1+j)}
\end{aligned}$$

$$\begin{aligned}
 -j60_{10} &= 11101000000010000 \times 111 \\
 &= 11000001110000_{\text{Base } (-1+j)}
 \end{aligned}$$

Having obtained CBNS representations for all types of integers (real and imaginary), it is now possible for us to represent an integer complex number (both real and imaginary parts of the complex number are integers) simply by adding the real and imaginary CBNS representations according to the addition algorithm given in [Chap. 3](#). Thus,

$$\begin{aligned}
 2012_{10} + j2012_{10} &= 1110000000001110000010000_{\text{Base } (-1+j)} \\
 &\quad + 10000000000010000110000_{\text{Base } (-1+j)} \\
 &= 1110100000001110100011100000_{\text{Base } (-1+j)}
 \end{aligned}$$

$$\begin{aligned}
 -60_{10} - j2000_{10} &= 1000111010000_{\text{Base } (-1+j)} \\
 &\quad + 111010000000111011100000000_{\text{Base } (-1+j)} \\
 &= 11101000000001101011010000_{\text{Base } (-1+j)}
 \end{aligned}$$

2.2 Conversion Algorithms for Fractional Numbers

The procedure for finding the binary equivalent in base $(-1 + j)$ for real fraction and imaginary fraction is very similar to the procedure explained for integers in [Sect. 2.1](#). As an example, CBNS representation for 0.351_{10} is obtained as follows [\[2\]](#):

- (i) Repeated multiplication by 4 gives:

$$\begin{aligned}
 0.351 \times 4 &= 1.404; \\
 0.404 \times 4 &= 1.616; \\
 0.616 \times 4 &= 2.464; \\
 0.464 \times 4 &= 1.856; \\
 0.856 \times 4 &= 3.424; \\
 0.424 \times 4 &= 1.696; \\
 0.696 \times 4 &= 2.784; \\
 0.784 \times 4 &= 3.136;
 \end{aligned}$$

and so on. Thus

$$0.351_{10} = 0.11213123 \dots_{\text{Base } 4}$$

- (ii) Converting the Base 4 number $(\dots n_5, n_4, n_3, n_2, n_1, n_0)$ to Base -4 by replacing each digit in the odd location $(n_1, n_3, n_5, \dots)$ with its negative to get $(\dots -n_5, n_4, -n_3, n_2, -n_1, n_0)$ yields:

$$0.351_{10} = 0.(-1)1(-2)1(-3)1(-2)3 \dots_{\text{Base } -4}$$

- (iii) After normalization, we have:

$$0.351_{10} = 1.32221223 \dots_{\text{Base } -4}$$

Table 2.2 Equivalence between fractional coefficients and $(-1+j)$ -base CBNS representation [1]

i	2^{-i}	CBNS representation base $(-1+j)$
1	2^{-1}	1.11
2	2^{-2}	1.1101
3	2^{-3}	0.000011
4	2^{-4}	0.00000001

(iv) And, finally replacing each Base -4 digit with its equivalent four-bit binary sequence as given in Table 2.1 gives:

$$0.351_{10} = 1.11011100110011000001 \dots \text{Base } (-1+j)$$

Similarly,

$$\begin{aligned} j0.351_{10} &= (1.11011100110011000001 \dots) \times (11) \\ &= 0.0100010000110100001100 \dots \text{Base } (-1+j) \end{aligned}$$

A more elaborate mathematical algorithm which can be easily programmed in some high-level language for converting fractional numbers into CBNS is described in [1]. According to this algorithm, any fraction F can be expressed uniquely in terms of powers of $1/2 = 2^{-1}$ such that

$$F = r_0 = f_1 \cdot 2^{-1} + f_2 \cdot 2^{-2} + f_3 \cdot 2^{-3} + f_4 \cdot 2^{-4} + \dots \quad (2.1)$$

Then the coefficients f_i and 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 $(2r_0 - 1) \geq 0$ then $f_1 = 1$

and set $r_1 = (2r_0 - 1)$

Then,

if $(2r_i - 1) < 0$ then $f_{i+1} = 0$ and set $r_{i+1} = 2r_i$

or if $(2r_i - 1) \geq 0$ then $f_{i+1} = 1$

and set $r_{i+1} = (2r_i - 1)$

This process continues until $r_i = 0$ or the machine limit has been reached. Then, $\forall f_i = 1$, replace its associated 2^{-i} according to Table 2.2 (only the first four values of i are listed in the table; for $i > 4$, refer to Table 2.3).

As an example, let

$$F = r_0 = 0.4375_{10}$$

Initially,

$$(2r_0 - 1) = 2(0.4375 - 1) = -0.125 < 0$$

$$\Rightarrow f_1 = 0 \text{ and } r_1 = 2r_0 = 2(0.4375) = 0.875$$

Then,

$$(2r_1 - 1) = 2(0.875) - 1 = 0.75 > 0$$

$$\Rightarrow f_2 = 1 \text{ and } r_2 = (2r_1 - 1) = 2(0.875) - 1 = 0.75$$

Continuing according to the algorithm, we have

$$(2r_2 - 1) = 2(0.75) - 1 = 0.5 > 0$$

$$\Rightarrow f_3 = 1 \text{ and } r_3 = (2r_2 - 1) = 2(0.75) - 1 = 0.5$$

$$(2r_3 - 1) = 2(0.5) - 1 = 0(\text{STOP})$$

$$\Rightarrow f_4 = 1 \text{ and } r_4 = 0$$

Thus,

$$\begin{aligned} 0.4375_{10} &= 0.2^{-1} + 1.2^{-2} + 1.2^{-3} + 1.2^{-4} \\ &= 0.(1.11) + 1(1.1101) + 1(0.000011) \\ &\quad + 1(0.00000001) = 1.11011101_{\text{Base } (-1+j)} \end{aligned}$$

(the addition is according to the algorithm given in [Chap. 3](#))

It is likely that most fractions will not terminate as this example, until the machine limit has been reached, e.g.

$$0.351_{10} = 1.110111001100110000011 \dots_{\text{Base } (-1+j)}$$

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

In general, to find CBNS representation of any 2^{-i} , express i as $4s + t$ where s is an integer and $0 \leq t < 4$. Then, depending upon value of t , 2^{-i} can be expressed as given in [Table 2.3](#). All rules for obtaining negative integer and positive/negative imaginary number representations in CBNS, as discussed previously, are equally applicable for obtaining negative fractional and positive/negative imaginary fractional representations in the new base.

A complex number which has only fractional real and imaginary parts can be represented in CBNS simply by adding the CBNS representations of each part according to the addition algorithm described in [Chap. 3](#). Thus,

$$\begin{aligned} (0.351 + j0.351)_{\text{Base } 10} &= 1.1101110011001100000111001100 \dots_{\text{Base } (-1+j)} \\ &\quad + 0.01000100001101000011001101 \dots_{\text{Base } (-1+j)} \\ &= 0.011010001111010111110001001 \dots_{\text{Base } (-1+j)} \end{aligned}$$

Table 2.3 Equivalence between value of “ t ” and $(-1+j)$ -base CBNS representation [1]

t	CBNS representation base $(-1+j)$
0	$0.0 \dots (8s-1)$ zeroes followed by 1
1	$0.0 \dots (8s-1)$ zeroes followed by 111
2	$0.0 \dots (8s-1)$ zeroes followed by 11101
3	$0.0 \dots (8s-1)$ zeroes followed by 11

2.3 Conversion Algorithms for Floating-Point Numbers

To represent a floating-point positive number in CBNS, we add the corresponding integer and fractional representations according to the addition algorithm described in Chap. 3. Once again, all rules for obtaining negative integer and positive/negative imaginary number representations, as discussed previously, are equally applicable for obtaining negative floating-point and positive/negative imaginary floating-point representations in CBNS. For example,

$$\begin{aligned}
 60.4375_{10} &= 11101000000010000_{\text{Base } (-1+j)} + 1.11011101_{\text{Base } (-1+j)} \\
 &= 11101000000010001.11011101_{\text{Base } (-1+j)} \\
 j60.4375_{10} &= (11101000000010001.11011101_{\text{Base } (-1+j)}) \times (11) \\
 &= 111000000110000.01000111_{\text{Base } (-1+j)}
 \end{aligned}$$

Adding these two CBNS representation using the addition algorithm outlined in Chap. 3 gives:

$$\begin{aligned}
 &(60.4375 + j60.4375)_{10} \\
 &= 11101000000010001.11011101_{\text{Base } (-1+j)} \\
 &\quad + 111000000110000.01000111_{\text{Base } (-1+j)} \\
 &= 10000011101110.1000011_{\text{Base } (-1+j)}
 \end{aligned}$$

In the above example, we have been able to represent a complex number (both real and imaginary parts are floating point numbers) in a single binary string. Thus, following the procedures outlined in this chapter, we can represent any complex number into CBNS format which is characterized by a single-unit string of bits (0 or 1).

References

1. T. Jamil, N. Holmes, D. Blest, *Towards implementation of a binary number system for complex numbers. Proceedings of the IEEE Southeastcon*, pp. 268–274 (2000)
2. T. Jamil, The complex binary number system: Basic arithmetic made simple. *IEEE Potentials* **20**(5), 39–41 (2002)



<http://www.springer.com/978-81-322-0853-2>

Complex Binary Number System

Algorithms and Circuits

Jamil, T.

2013, XII, 83 p. 11 illus., Softcover

ISBN: 978-81-322-0853-2