Complex Binary Adder Designs and their Hardware Implementations

Tariq Jamil[1], Medhat Awadalla[2], Iftaquaruddin Mohammed[3]
Department of Electrical and Computer Engineering
Sultan Qaboos University
AlKhod, OMAN

***Abstract*—Complex Binary Number System (CBNS) is (-1+j)-based on binary number system which facilitates both real and imaginary components of a complex number to be represented as single binary number. In this paper, we have presented three designs of nibble-size complex binary adders (ripple-carry, decoder-based, minimum-delay) and implemented them on various Xilinx FPGAs. The designs of base2 4-bit binary adder have also been implemented so that statistics of different adders can be compared.**

***Keywords*—Complex number; complex binary; adder; ripple carry; decoder; minimum delay**

I. Introduction

Complex numbers play important roles in various areas of electrical and computer engineering but their representation and treatment in the realm of computing are based on a *divide-and-conquer* technique wherein real part of the complex number is dealt with separately and imaginary part of the complex number is handled separately. Thus, addition of two complex numbers (a+jb) and (c+jd) involves two separate additions: (a+c) for the real parts and (b+d) for the imaginary parts. To facilitate single-unit representation of a complex number which will ultimately result in the reduction of arithmetic operations for the real and imaginary components of complex numbers, Complex Binary Number System (CBNS) with (-1+j)-base has been proposed in the scientific literature [1-4]. In this paper, we are going to present three designs of nibble-size complex binary adder circuits and their implementations on various Xilinx FPGAs. For the sake of comparison, we'll also present implementation of base2 nibble-size adder so that relative complexity of different adder designs can be appreciated.

II. Complex Binary Number System

A. Binary Representation

The value of an n-bit binary number with base (-1+j) can be written in the form of a power series as follows: $a_{n-1}(-1+j)^{n-1} + a_{n-2}(-1+j)^{n-2} + a_{n-3}(-1+j)^{n-3} + \ldots + a_2(-1+j)^2 + a_1(-1+j)^1 + a_0\ (-1+j)^0$ 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). This is analogous to the ordinary binary number system power series of $a_{n-1}(2)^{n-1} + a_{n-2}(2)^{n-2} + a_{n-3}(2)^{n-3} + \ldots + a_2(2)^2 + a_1(2)^1 + a_0\ (2)^0$ except that the bases are different. Details about how to convert a given complex number into (-1+j)-base complex binary number representation can be found in [1-4]. By the application of the conversion algorithms mentioned in these publications, a given complex number can be represented as a single binary entity. For example, the complex number 2019+j2019 has the binary representation in base (-1+j) as: 1110100000001110100001100110.

B. Addition Algorithm

The binary addition of two complex binary 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 except for the last case when 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].

III. Adder Designs

A. Ripple-Carry

The block diagram of a 4-bit Complex Binary Ripple-Carry Adder (CBRCA) is shown in Fig. 1 [5].

The adder performs the addition of two 4-bit complex binary numbers A ($a_3a_2a_1a_0$) and B ($b_3b_2b_1b_0$) and generates a 4-bit (-1+j)-radix result (Sum) and up to 8 Extended-Carries. It consists of the Addition Unit, the Extended-Carry Generation Unit, the Zero Detection Unit, and the Output Generation Unit.

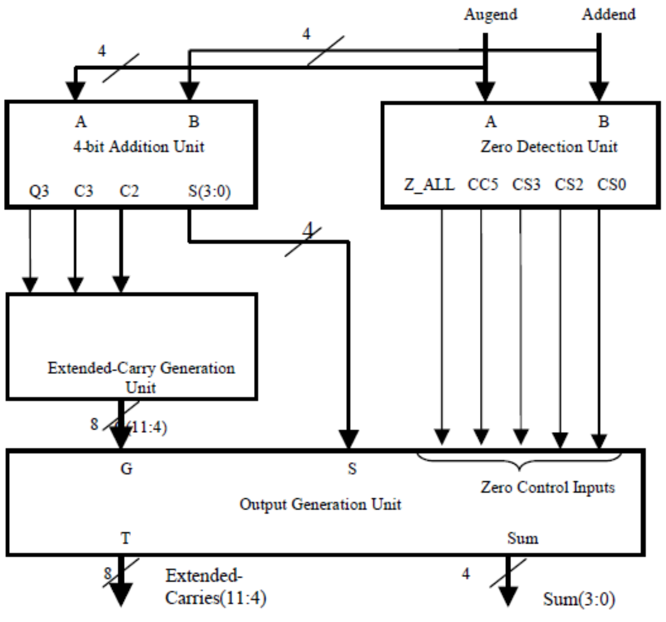

Fig. 1. Block Diagram of a 4-bit Complex Binary Ripple-Carry Adder [5].

The Addition Unit is structured from 4 semi-cascaded stages. Each stage is responsible of generating one of the bits of the result *(S_0-S_3)*. The carry generated from the addition of two bits in (-1+j) radix representation at stage *i* produces a carry that should be propagated to stages *i+2* and *i+3*. Since no carry-in(s) to the adder is assumed, stages 0 and 1 are easily implemented using two half-adders. Stage 2 is implemented using a full-adder with a carry-in generated from stage 0. While for stage 3, a specially designed 4-input binary variables adding component is implemented [6]. Stage 3 performs the addition of bits b_3 and a_3 of the Addend and Augend with possible two carries referred to by K_{31} and K_{32}, which may be generated from stages 0 and 1, respectively. Stage 3 produces bit S_3 of the result and two carry bits, C_3 and Q_3, according to the 4 binary variables truth table for the addition stage. C_3 is a normal carry due to adding three ones (1+1+1), and Q_3 is an extended carry due to adding four ones (1+1+1+1) in (-1+j) radix representation. C_3 should propagate to stages 5 and 6, and Q_3 to stages 7, 9, 10, and 11. Since the adder performs 4-bit (-1+j)-base complex number addition, the carries C_2, C_3, and Q_3 are taken to the inputs of the Extended-Carries unit, in order to generate all the necessary carries. All carries generated by stages 2 and 3 are handled by dummy stages in the Extended-Carry Generation Unit, referred to by stages 4 to 11. Each stage of the Extended-Carries Unit is responsible of generating one Extended-Carry bit. The Boolean equations for stages 0, 1, and 2 are obvious from the use of half-adders and full-adder circuits. For stage 3, the Boolean equations of the outputs are found from the minimization of 4-variable Karnaugh maps. These are,

$$S_3 = a_3 \oplus b_3 \oplus K_{31} \oplus K_{32} \quad (1)$$

$$C_3 = \bar{a}_3 K_{31} K_{32} + b_3 K_{31} \bar{K}_{32} + a_3 \bar{b}_3 K_{31} + a_3 \bar{K}_{31} K_{32} + a_3 b_3 \bar{K}_{31} + b_3 \bar{K}_{31} K_{32} \quad (2)$$

$$Q_3 = a_3 b_3 K_{31} K_{32} \quad (3)$$

The S_3 expression is a 4-input odd function that can be implemented by EXCLUSIVE-OR gates, the Q_3 expression is a 4-input AND function, and the C_3 is a sum-of-product expression that can be implemented by a two-stage logic (e.g., AND-OR, or NAND-NAND).

The Extended-Carry Generation Unit consists of 8 dummy stages, 4-11. They handle the propagated carries from stages 2 *(C_2)* and 3 *(C_3, Q_3)* in the Addition Unit. The Dummy stages 4, 5, 6, and 8 are implemented using half adders, and dummy stages 7, 9, 10, and 11 are implemented using full-adders. The unit would generate the extended carries *(C_4-C_{11})* as inputs to the Output Generation Unit.

The Zero Detection Unit determines the conditions necessary to generate special output results based on the recognition of specific patterns for the Addend and the Augend. All conditions considered are based on the Zero Rule for the (-1+j) radix number representation. Assuming 4-bit (-1+j) radix numbers, the unit receives inputs for the Addend *(b_3 b_2 b_1 b_0)* and the Augend *(a_3 a_2 a_1 a_0)*, and generates five control signals: *CS0*, *CS1*, *CS3*, *CC5*, and *Z_ALL*. The five control signals are generated based on the patterns detected for the Addend *($b_3b_2b_1b_0$)* and the Augend *($a_3a_2a_1a_0$)*, which satisfy the Zero Rule. Table 1 lists all the minterms of the input that will generate special output results.

The Boolean expressions characterizing each control output are defined below.

1) CS0 Control Output: CS0 controls the summation bit S_0 according to table 1. Its Boolean expression is described as:

$$CS0 = \sum(111, 126, 231, 239, 246, 254)$$
$$CS0 = (a_2\, a_1\, b_2\, b_1)(\overline{a_0}\, b_3\, b_0 + a_0\, b_3\, \overline{b_0} + a_3\, b_3\, (a_0 \oplus b_0)) \quad (4)$$

2) CS2 Control Output: CS2 controls the summation bit S_2 according to table 1. Its Boolean expression is described as:

$$CS2 = \sum(119, 127, 247, 255)$$
$$CS2 = (a_2\, a_1\, a_0\, b_2\, b_1\, b_0) \quad (5)$$

3) CS3 Control Output: CS3 controls the summation bit S_3 according to table 1. Its Boolean expression is described as:

$$CS3 = \sum(63, 123, 127, 183, 238, 239, 243, 247, 254)$$
$$CS3 = (a_1\, b_1)(\overline{a_3}\, a_0\, b_3\, b_2\, b_0 + a_3\, a_0\, \overline{b_3}\, b_2\, b_0 + a_3\, a_2\, b_3\, b_2\, \overline{b_0} + \overline{a_3}\, a_2\, a_0\, b_3\, b_0 + a_3\, a_2\, a_0\, \overline{b_3}\, b_0 + a_3\, a_2\, \overline{a_0}\, b_3\, b_2) \quad (6)$$

4) CC5 Control Output: CC5 controls the extended carry bits C_5 *and* C_6 according to table 1. Its Boolean expression is described as:

$$CC5 = \sum(191, 251, 255)$$
$$CC5 = (a_3\, a_2\, a_1\, a_0\, b_3\, b_1\, b_0 + a_3\, a_1\, a_0\, b_3\, b_2\, b_1\, b_0) \quad (7)$$

5) Z_ALL Control Output: Z_ALL controls generating all zeros in the sum and extended carry bits according to Table 1. Its Boolean expression is described as:

$$Z\_ALL = \sum(55, 110, 115, 230)$$
$$Z\_ALL = (a_1\, b_1)(\overline{a_3}\, a_0\, \overline{b_3}\, b_0\, (a_2 \oplus b_2) + a_2\, \overline{a_0}\, b_2\, \overline{b_0}\, (a_3 \oplus b_3)) \quad (8)$$

The Output Generation Unit receives the control signals, *(CS0, CS2. CS3, CC5, Z_ALL)*, from the Zero Detection Unit, the result of addition *(S_0-S_3)* and the extended-carries *(C_4-C_{11})*. Then it determines the actual Sum bits *(Sum_0-Sum_3)* and the actual Extended-Carry bits *(T_4-T_{11})* according to the control signals described above.

B. Decoder-Based

The design of a nibble-size decoder-based adder involves the following steps[7]: (i) Generation of a truth table with two 4-bit operands --- operand A with $a_3a_2a_1a_0$ bits and Operand B with $b_3b_2b_1b_0$ bits --- addition of these two operands produces twelve outputs which are labeled as $c_{11}c_{10}c_9c_8c_7c_6c_5c_4s_3s_2s_1s_0$.

The truth table (Table 2) has a total of $2^8 = 256$ minterms. (ii) We have used a 8x256 decoder to implement this truth table. For this purpose, we expressed each output in sum-of-minterms form as shown on the next page. (iii) Finally, these expressions have been implemented using the decoder and OR gates as shown in Fig. 2.

TABLE. I. APPLICATION OF THE ZERO RULE TO 4-BIT ADDITION OPERANDS

MINTERM (Dec.)	$a_3a_2a_1a_0b_3b_2b_1b_0$ (Hex.)	$C_{11}C_{10}C_9C_8C_7C_6C_5C_4$ (Hex.)	$S_3S_2S_1S_0$ (Hex.)
55	37	00	0
63	3F	00	8
110	6E	00	0
111	6F	00	1
115	73	00	0
119	77	00	4
123	7B	00	4
126	7E	00	1
127	7F	00	C
183	B7	00	8
191	BF	06	0
230	E6	00	0
231	E7	00	1
238	EE	00	8
239	EF	00	9
243	F3	00	8
246	F6	00	1
247	F7	00	C
251	FB	06	0
254	FE	00	9
255	FF	06	4

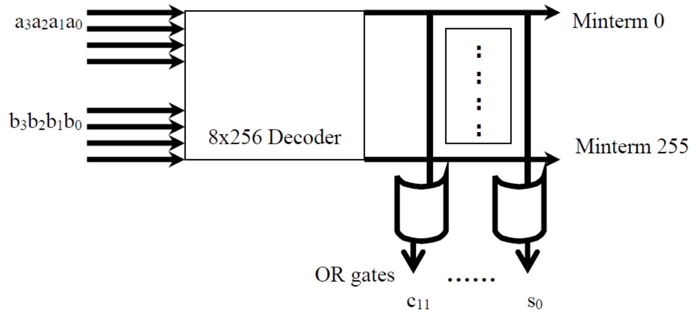

Fig. 2. Block Diagram of a 4-bit Complex Binary Complex Binary Adder using Decoder.

$c_{11} = \Sigma$ (187) $c_{10} = \Sigma$ (187) $c_9 = \Sigma$ (187)

$c_8 = \Sigma$ (29,31,61,89,91,93,95,121,125,149,151,157,159,181,189, 204,205,206,207,209,211,213, 215,217, 219,220,221,222,223, 236,237,241,245,249,252,253)

$c_7 = \Sigma$ (29,31,61,89,91,93,95,102,103,118,121,125,149,151,157, 159, 181,187,189,204, 205,206,207,209,211,213,215,217,219, 220,221,222,223, 236,237,241, 245,249,252,253)

$c_6 = \Sigma$ (25,27,29,31,42,43,46,47,51,57,58,59,61,62,89,91,93, 95,102,103,106,107,118,121,122,125,136,137,138,139,140, 141,142,143,145,147,149,151,152,153,154,155,156,157,158, 159,162,163,166,167,168, 169,170,171,172,173,174,175,177, 178,179,181,182,184,185,186,188,189,190,191,200,201,202, 203,204,205,206,207,209,211,213,215,216,217,218,219,220, 221, 222,223,226, 227,232,233,234,235,236,237,241,242,245, 248,249,250,251,252, 253,255)

$c_5 = \Sigma$ (21,23,25,27,42,43,46,47,51,53,57,58,59,62,68,69,70, 71,76,77,78,79,81,83,84,85,86,87,92,94,100, 101,102,103, 106,107,108,109,113,116,117,118,122,124,136,137,138,139, 140,141,142,143,145,147,152,153,154,155,156, 158,162, 163,166,167,168,169,170,171,172,173,174,175,177,178,179, 182,184,185,186,188,190,191,196,197,198,199,200, 201,202, 203,212, 214,216, 218,226,227,228, 229,232,233,234,235, 242,244,248,250,251,255)

$c_4 = \Sigma$ (21,23,29,31, 34,35,38,39,42,43,46,47,50,51,53,54,58, 59,61,62,68,69,70,71,76,77,78,79, 81,83,84,85,86,87,89,91,92, 93,94,95,98,99,100,101,106,107,108,109,113,114,116,117, 121,122,124,125,149,151,157,159,162,163,166,167,170,171, 174,175,178,179, 181,182,186,187,189,190, 196,197,198, 199,204,205,206,207,209,211,212,213,214,215,217,219,220, 221, 222, 223,226,227, 228,229,234,235,236,237,241,242, 244,245,249,250,252,253)

$s_3 = \Sigma$ (8,9,10,11,12,13,14,15,17,19,21,23,24,26,28,30, 34,35, 38,39,40,41,44,45,49,50,53,54,56, 59,60,63,72,73,74,75,76, 77,78,79,81,83,85,87,88,90,92,94,98,99,102,103,104,105, 108,109,113,114,117,118,120,123,124,127,128,129,130,131, 132, 133,134,135,144,146,148,150,153,155,157, 159,160,161, 164,165,170, 171,174,175,176,179,180,183,185,186,189,190, 192, 193,194,195,196,197,198,199,208,210,212,214,217,219, 221,223,224,225,228,229,234,235,238,239,240,243,244,247, 249, 250,253,254)

$s_2 = \Sigma$ (4,5,6,7,12,13,14,15,17,19,20,22,25,27,28,30,36,37,38, 39,44,45,46,47,49,51,52,54,57, 59,60,62,64,65,66, 67,72,73, 74,75,80,82,85,87,88,90,93,95,96,97,98,99,104,105,106,107, 112, 114,117,119,120,122,125,127,132,133,134,135,140,141, 142,143,145,147,148,150,153, 155,156, 158,164,165,166,167, 172,173,174,175,177,179,180,182,185,187,188,190, 192,193, 194,195,200, 201,202,203,208,210,213,215,216,218,221,223, 224,225,226,227,232,233,234,235,240,242,245, 247,248,250, 253,255)

$s_1 = \Sigma$(2,3,6,7,10,11,14,15,18,19,22,23,26,27,30,31,32,33,36, 37,40,41,44,45,48,49,52,53,56,57, 60,61,66,67,70,71,74,75, 78,79,82,83,86,87,90,91,94,95,96,97,100,101,104,105,108, 109,112, 113,116,117,120,121,124,125,130,131,134, 135,138, 139,142,143,146,147,150,151,154,155,158,159,160,161,164, 165,168, 169,172,173,176,177,180,181,184,185, 188,189,194, 195,198,199,202, 203,206,207,210,211,214,215,218,219, 222,223,224,225,228,229,232,233,236,237,240,241,244, 245, 248, 249,252,253)

s_0= Σ (1,3,5,7,9,11,13,15,16,18,20,22,24,26,28,30,33,35,37,39, 41,43,45,47,48,50,52,54,56,58,60,62,65,67,69,71,73,75,77,79, 80,82,84,86,88,90,92,94,97,99,101,103,105,107,109,111,112, 114,116,118,120,122,124,126,129,131,133,135,137,139,141, 143,144,146,148,150,152,154,156,158,161,163,165,167,169, 171,173,175,176,178,180,182,184,186,188,190,193,195,197, 199,201,203,205,207,208,210,212,214,216,218,220,222,225, 227,229,231,233,235,237,239,240,242,244,246,248,250,252, 254)

C. Minimum-Delay

The truth table of the 4-bit complex binary adder, given in Table 2, was entered, one output at a time, into online Karnaugh Map [8] and simplified Boolean expression for each output was obtained. To facilitate use of online K-Map, the inputs were labeled as ABCD for the augend and EFGH for the addend. The outputs were labeled as JKLMPRSTUWYZ. The simplified expression for each output was implemented on Xilinx FPGAs and statistics for the circuit were obtained.

The simplified expressions obtained for outputs are:

$$J = K = L = A\overline{B}CDE\overline{F}GH$$

$$\begin{aligned} M = {} & \overline{C}DEFH + DEF\overline{G}H + B\overline{C}DEH + A\overline{C}DFH + ADF\overline{G}H \\ & + AB\overline{C}EF + ABEF\overline{G} + AB\overline{C}DH + ABD\overline{G}H \end{aligned}$$

$$\begin{aligned} P = {} & \overline{C}DEFH + DEF\overline{G}H + B\overline{C}DEH + BDE\overline{G}H + A\overline{C}DFH \\ & + ADF\overline{G}H + AB\overline{C}EF + ABEF\overline{G} + AB\overline{C}DH \\ & + ABD\overline{G}H + \overline{A}BC\overline{DE}FG + \overline{A}BC\overline{E}FG\overline{H} \\ & + A\overline{B}CDE\overline{F}GH \end{aligned}$$

$$\begin{aligned} R = {} & A\overline{C}E + AE\overline{G} + \overline{C}DEH + DE\overline{G}H + A\overline{C}DH + AD\overline{G}H \\ & + ABE\overline{F} + \overline{B}C\overline{D}EG + C\overline{D}E\overline{F}G + \overline{B}CEG\overline{H} \\ & + CE\overline{F}G\overline{H} + ADEFH + A\overline{B}C\overline{D}G + AC\overline{DF}G \\ & + A\overline{B}CG\overline{H} + AC\overline{F}G\overline{H} + A\overline{B}D\overline{EF}H \\ & + \overline{AB}CD\overline{F}GH + \overline{A}BC\overline{DE}FG + \overline{A}BC\overline{E}FG\overline{H} \end{aligned}$$

$$\begin{aligned} S = {} & A\overline{BD}E + A\overline{D}E\overline{F} + A\overline{B}E\overline{H} + AE\overline{FH} + \overline{B}C\overline{D}EG \\ & + C\overline{D}E\overline{F}G + \overline{B}CEG\overline{H} + CE\overline{F}G\overline{H} + \overline{A}B\overline{CD}F \\ & + \overline{A}B\overline{DE}F + B\overline{C}\overline{D\overline{E}}F + \overline{A}B\overline{D}F\overline{G} \\ & + B\overline{DE}F\overline{G} + \overline{A}B\overline{C}F\overline{H} + \overline{A}B\overline{E}F\overline{H} \\ & + B\overline{CE}F\overline{H} + \overline{A}BF\overline{GH} + B\overline{E}F\overline{GH} \\ & + A\overline{BC}E\overline{F} + A\overline{B}E\overline{FG} + A\overline{B}C\overline{D}G + AC\overline{DF}G \\ & + A\overline{B}CG\overline{H} + AC\overline{F}G\overline{H} + \overline{AC}D\overline{E}FH \\ & + \overline{A}D\overline{E}F\overline{G}H + \overline{AB}DE\overline{F}H + A\overline{B}D\overline{EF}H \\ & + ABCE\overline{F}G + \overline{AB}CD\overline{F}GH + \overline{A}B\overline{CDEG}H \\ & + \overline{A}B\overline{C}D\overline{E}GH + \overline{A}BCD\overline{EG}H + ACDEFGH \end{aligned}$$

$$\begin{aligned} T = {} & B\overline{C}F + BF\overline{G} + \overline{C}DFH + DF\overline{G}H + \overline{B}C\overline{D}G + \overline{B}C\overline{F}G \\ & + C\overline{DF}G + \overline{B}CG\overline{H} + C\overline{F}G\overline{H} + B\overline{C}DGH \\ & + B\overline{C}DEH + BCD\overline{G}H + AB\overline{C}DH \\ & + \overline{A}B\overline{CDEG}H \end{aligned}$$

$$\begin{aligned} U = {} & \overline{ACD}E + \overline{AD}E\overline{G} + \overline{AC}E\overline{H} + \overline{A}E\overline{GH} + A\overline{CDE} + A\overline{DEG} \\ & + A\overline{CEH} + A\overline{EGH} + \overline{AC}D\overline{E}H + \overline{A}D\overline{EG}H \\ & + \overline{A}C\overline{DE}G + \overline{A}C\overline{E}G\overline{H} + A\overline{C}DEH \\ & + ADE\overline{G}H + AC\overline{D}EG + ACEG\overline{H} \\ & + \overline{A}CDEGH + ACD\overline{E}GH + \overline{A}B\overline{CDFG}H \end{aligned}$$

$$\begin{aligned} W = {} & \overline{BD}F + B\overline{FH} + \overline{AB}F\overline{H} + \overline{BE}F\overline{H} + \overline{B}FG\overline{H} + \overline{B}D\overline{F}H \\ & + \overline{B}CF\overline{H} + B\overline{DF}G + B\overline{D}E\overline{F} + BDFH \\ & + BC\overline{DF} + AB\overline{DF} + \overline{AC}D\overline{EFG}H \\ & + A\overline{BC}EF\overline{G}H \end{aligned}$$

$$Y = \overline{C}G + C\overline{G} + \overline{A}B\overline{CEF}H$$

$$\begin{aligned} Z = {} & D\overline{H} + \overline{BD}H + \overline{D}GH + \overline{D}FH + \overline{D}EH + C\overline{D}H + A\overline{D}H \\ & + \overline{A}B\overline{C}D\overline{EFG} \end{aligned}$$

The logic diagram of minimum-delay complex binary adder is given in Fig. 3.

The logic diagram of each adder output (J,K,L,M,P,R,S,T, U,W,Y,Z) is shown in Figs.4-15.

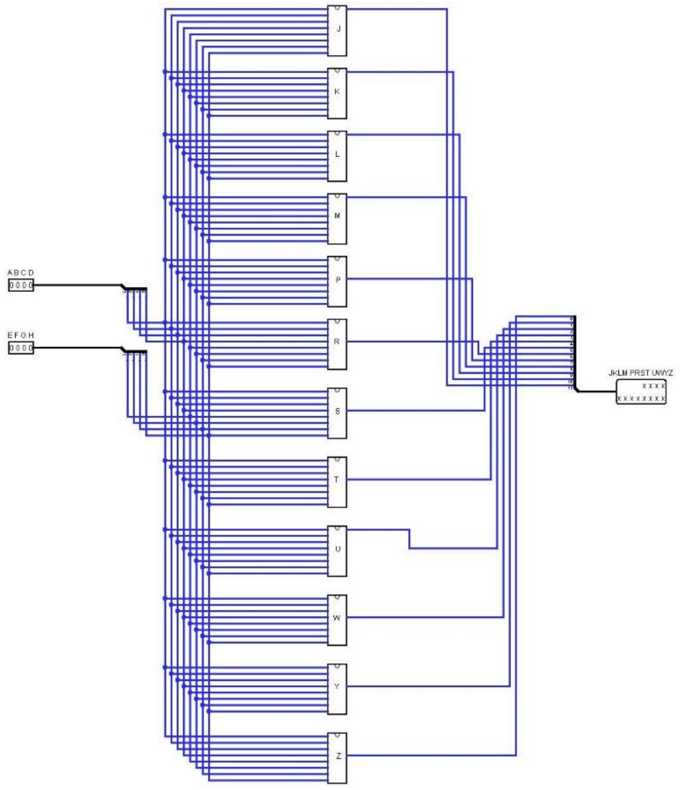

Fig. 3. Block Diagram of a 4-bit Complex Binary Minimum-Delay Adder.

TABLE. II. TRUTH TABLE OF A 4-BIT COMPLEX BINARY ADDER
(MINTERM: $A_3A_2A_1A_0$ ADD $B_3B_2B_1B_0 = C_{11}C_{10}C_9C_8C_7C_6C_5C_4S_3S_2S_1S_0$)

0	00000000 000000000000	64	01000000 000000000100	128	10000000 000000001000	192	11000000 000000001100
1	00000001 000000000001	65	01000001 000000000101	129	10000001 000000001001	193	11000001 000000001101
2	00000010 000000000010	66	01000010 000000000110	130	10000010 000000001010	194	11000010 000000001110
3	00000011 000000000011	67	01000011 000000000111	131	10000011 000000001011	195	11000011 000000001111
4	00000100 000000000100	68	01000100 000000110000	132	10000100 000000001100	196	11000100 000000111000
5	00000101 000000000101	69	01000101 000000110001	133	10000101 000000001101	197	11000101 000000111001
6	00000110 000000000110	70	01000110 000000110010	134	10000110 000000001110	198	11000110 000000111010
7	00000111 000000000111	71	01000111 000000110011	135	10000111 000000001111	199	11000111 000000111011
8	00001000 000000001000	72	01001000 000000001100	136	10001000 000001100000	200	11001000 000001100100
9	00001001 000000001001	73	01001001 000000001101	137	10001001 000001100001	201	11001001 000001100101
10	00001010 000000001010	74	01001010 000000001110	138	10001010 000001100010	202	11001010 000001100110
11	00001011 000000001011	75	01001011 000000001111	139	10001011 000001100011	203	11001011 000001100111
12	00001100 000000001100	76	01001100 000000111000	140	10001100 000001100100	204	11001100 000111010000
13	00001101 000000001101	77	01001101 000000111001	141	10001101 000001100101	205	11001101 000111010001
14	00001110 000000001110	78	01001110 000000111010	142	10001110 000001100110	206	11001110 000111010010
15	00001111 000000001111	79	01001111 000000111011	143	10001111 000001100111	207	11001111 000111010011
16	00010000 000000000001	80	01010000 000000000101	144	10010000 000000001001	208	11010000 000000001101
17	00010001 000000001100	81	01010001 000000111000	145	10010001 000001100100	209	11010001 000111010000
18	00010010 000000000011	82	01010010 000000000111	146	10010010 000000001011	210	11010010 000000001111
19	00010011 000000001110	83	01010011 000000111010	147	10010011 000001100110	211	11010011 000111010010
20	00010100 000000000101	84	01010100 000000110001	148	10010100 000000001101	212	11010100 000000111001
21	00010101 000000111000	85	01010101 000000111100	149	10010101 000111010000	213	11010101 000111010100
22	00010110 000000000111	86	01010110 000000110011	150	10010110 000000001111	214	11010110 000000111011
23	00010111 000000111010	87	01010111 000000111110	151	10010111 000111010010	215	11010111 000111010110
24	00011000 000000001001	88	01011000 000000001101	152	10011000 000001100001	216	11011000 000001100101
25	00011001 000001100100	89	01011001 000111010000	153	10011001 000001101100	217	11011001 000111011000
26	00011010 000000001011	90	01011010 000000001111	154	10011010 000001100011	218	11011010 000001100111
27	00011011 000001100110	91	01011011 000111010010	155	10011011 000001101110	219	11011011 000111011010
28	00011100 000000001101	92	01011100 000000111001	156	10011100 000001100101	220	11011100 000111010001
29	00011101 000111010000	93	01011101 000111010100	157	10011101 000111011000	221	11011101 000111011100
30	00011110 000000001111	94	01011110 000000111011	158	10011110 000001100111	222	11011110 000111010011
31	00011111 000111010010	95	01011111 000111010110	159	10011111 000111011010	223	11011111 000111011110
32	00100000 000000000010	96	01100000 000000000110	160	10100000 000000001010	224	11100000 000000001110
33	00100001 000000000011	97	01100001 000000000111	161	10100001 000000001011	225	11100001 000000001111
34	00100010 000000011000	98	01100010 000000011100	162	10100010 000001110000	226	11100010 000001110100
35	00100011 000000011001	99	01100011 000000011101	163	10100011 000001110001	227	11100011 000001110101
36	00100100 000000000110	100	01100100 000000110010	164	10100100 000000001110	228	11100100 000000111010
37	00100101 000000000111	101	01100101 000000110011	165	10100101 000000001111	229	11100101 000000111011
38	00100110 000000011100	102	01100110 000011101000	166	10100110 000001110100	230	11100110 000000000000
39	00100111 000000011101	103	01100111 000011101001	167	10100111 000001110101	231	11100111 000000000001
40	00101000 000000001010	104	01101000 000000001110	168	10101000 000001100010	232	11101000 000001100110
41	00101001 000000001011	105	01101001 000000001111	169	10101001 000001100011	233	11101001 000001100111
42	00101010 000001110000	106	01101010 000001110100	170	10101010 000001111000	234	11101010 000001111100
43	00101011 000001110001	107	01101011 000001110101	171	10101011 000001111001	235	11101011 000001111101
44	00101100 000000001110	108	01101100 000000111010	172	10101100 000001100110	236	11101100 000111010010
45	00101101 000000001111	109	01101101 000000111011	173	10101101 000001100111	237	11101101 000111010011
46	00101110 000001110100	110	01101110 000000000000	174	10101110 000001111100	238	11101110 000000001000
47	00101111 000001110101	111	01101111 000000000001	175	10101111 000001111101	239	11101111 000000001001
48	00110000 000000000011	112	01110000 000000000111	176	10110000 000000001011	240	11110000 000000001111
49	00110001 000000001110	113	01110001 000000111010	177	10110001 000001100110	241	11110001 000111010010
50	00110010 000000011001	114	01110010 000000011101	178	10110010 000001110001	242	11110010 000001110101
51	00110011 000001110100	115	01110011 000000000000	179	10110011 000001111100	243	11110011 000000001000
52	00110100 000000000111	116	01110100 000000110011	180	10110100 000000001111	244	11110100 000000111011
53	00110101 000000111010	117	01110101 000000111110	181	10110101 000111010010	245	11110101 000111010110
54	00110110 000000011101	118	01110110 000011101001	182	10110110 000001110101	246	11110110 000000000001
55	00110111 000000000000	119	01110111 000000000100	183	10110111 000000001000	247	11110111 000000001100
56	00111000 000000001011	120	01111000 000000001111	184	10111000 000001100011	248	11111000 000001100111
57	00111001 000001100110	121	01111001 000111010010	185	10111001 000001101110	249	11111001 000111011010
58	00111010 000001110001	122	01111010 000001110101	186	10111010 000001111001	250	11111010 000001111101
59	00111011 000001111100	123	01111011 000000001000	187	10111011 111010010100	251	11111011 000001100000
60	00111100 000000001111	124	01111100 000000111011	188	10111100 000001100111	252	11111100 000111010011
61	00111101 000111010010	125	01111101 000111010110	189	10111101 000111011010	253	11111101 000111011110
62	00111110 000001110101	126	01111110 000000000001	190	10111110 000001111101	254	11111110 000000001001
63	00111111 000000001000	127	01111111 000000001100	191	10111111 000001100000	255	11111111 000001100100

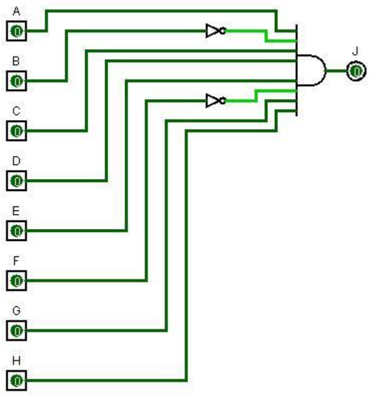

Fig. 4. Logic Diagram for Output J.

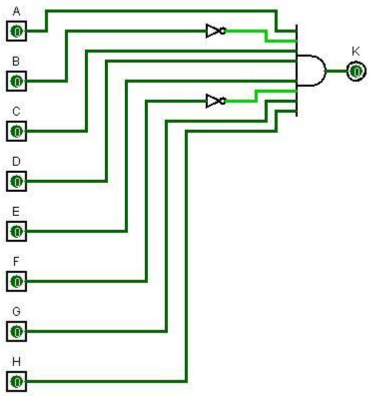

Fig. 5. Logic Diagram for Output K.

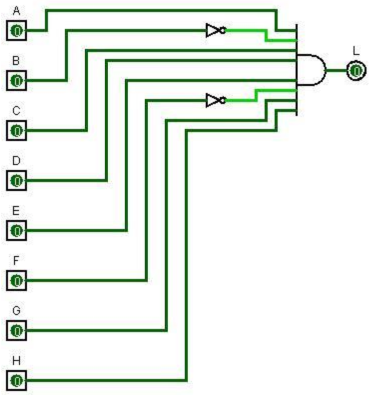

Fig. 6. Logic Diagram for Output L.

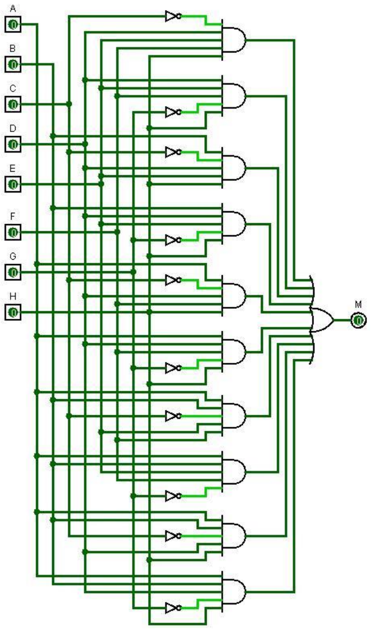

Fig. 7. Logic Diagram for Output M.

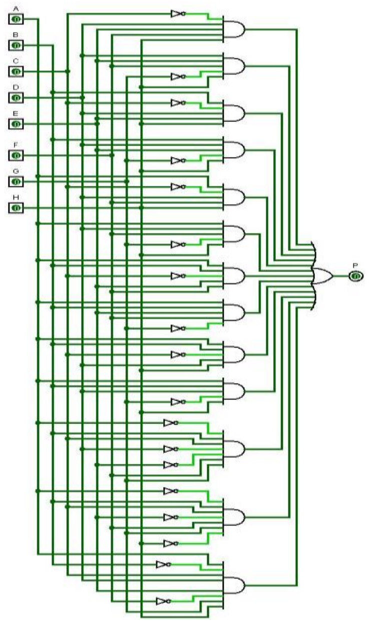

Fig. 8. Logic Diagram for Output P.

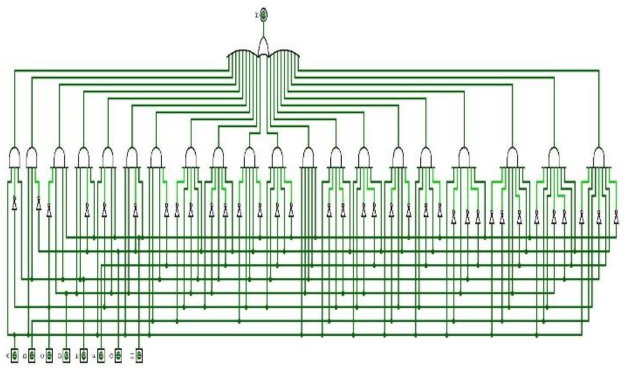

Fig. 9. Logic Diagram for Output R.

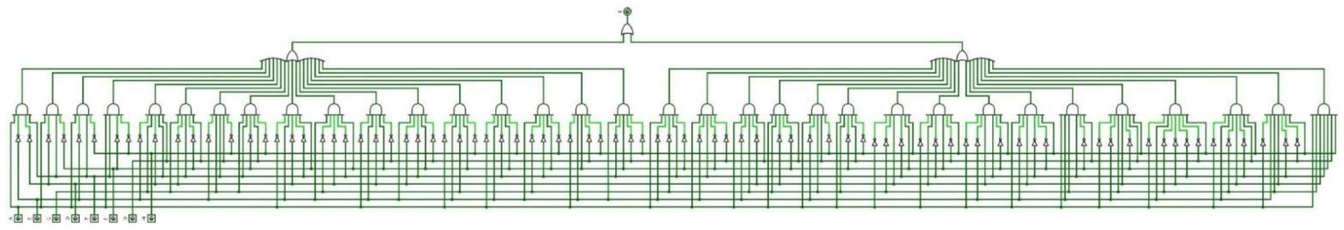

Fig. 10. Logic diagram for Output S

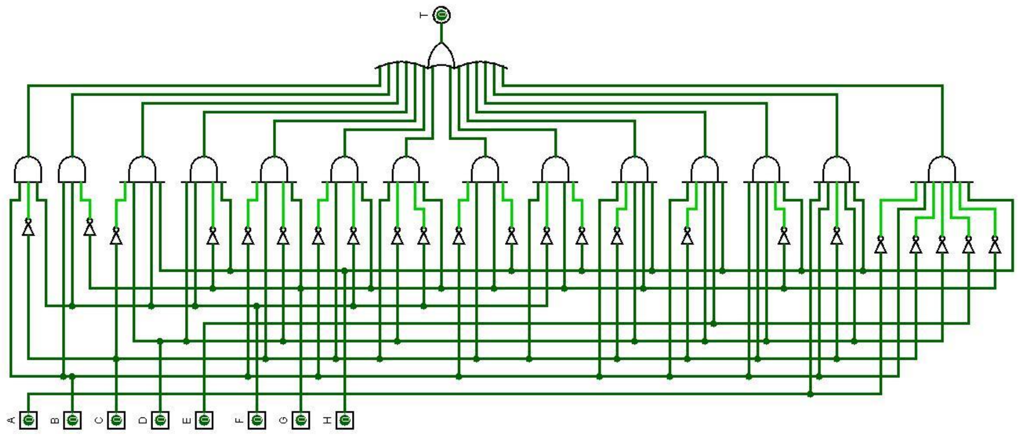

Fig. 11. Logic diagram for Output T

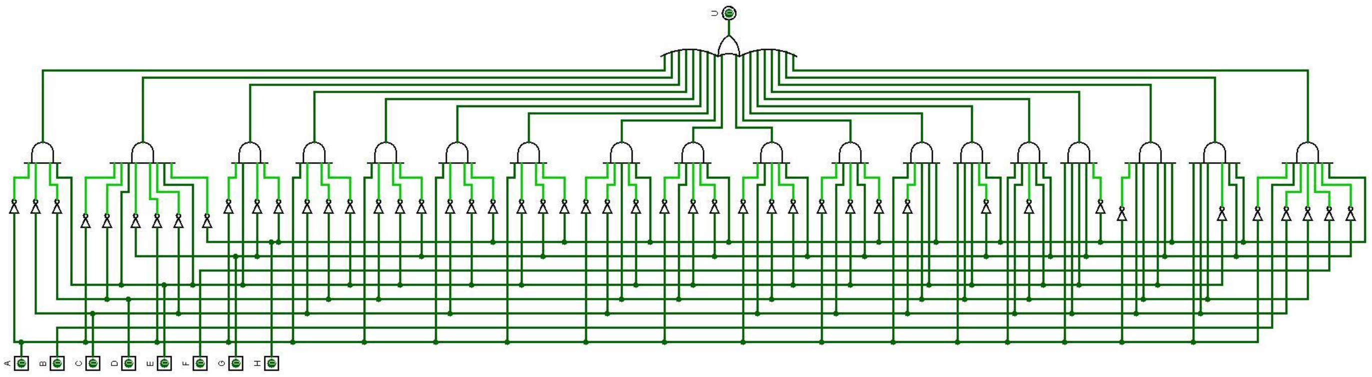

Fig. 12. Logic diagram for Output U

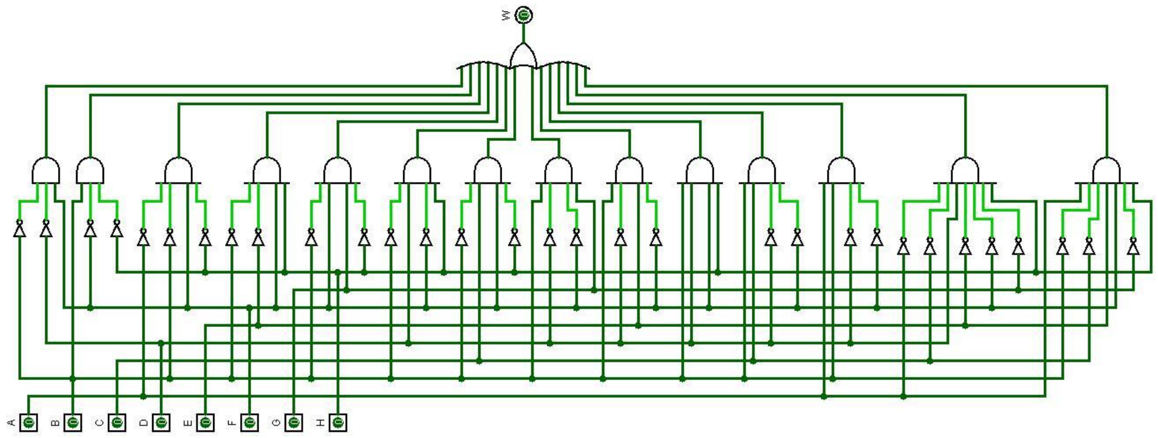

Fig. 13. Logic Diagram for Output W.

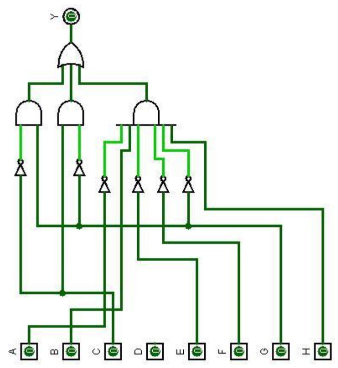

Fig. 14. Logic diagram for Output Y.

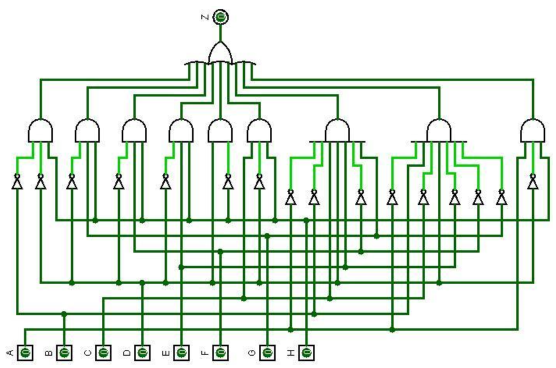

Fig. 15. Logic Diagram for Output Z.

IV. Results

Complex binary adder designs presented in the previous section have been implemented on Xilinx[9] FPGAs and various statistics pertaining to each design are given in Tables 3-5.

TABLE. III. FPGA Implementation Results of Complex Binary Ripple-Carry Adder (CBRCA) and base 2 Adder [5]

	Complex Binary Ripple-Carry Adder Implementations on FPGA Devices			Base 2 Ripple-Carry Adder ImplementatioN
	XC 4003E	Spartan XCS05	Virtex XCV50	Virtex XCV50
Number of external IOBs[a]	20/80 (32%)	20/80 (32%)	20/94 (21%)	13/94 (13%)
Number of CLBs[b] (Slices[c])	24/100 (24%)	24/100 (24%)	31/768 (4%)	6/768 (1%)
Number of 4 input LUTs[d]	42/200 (21%)	42/200 (21%)	59/1536 (3%)	9/1536 (1%)
Number of 3 input LUTs	13/100 (13%)	13/ 100 (13%)		
Number of bonded IOBs	20/61 (32%)	20/ 61 (32%)	20/ 94 (21%)	13/94 (13%)
Gate count	310	310	354	54
Average connection delay (ns)	2.808	3.506	1.640	1.525
Maximum combinational delay (ns)	35.680	45.995	24.839	15.389

[a]IOBs: Programmable Input/Ouput Blocks.
[b]CLBs: Configurable Logic Blocks.
[c]Slice: Each Virtex CLB contains 4 logic cells organized in two similar slices.
[d]LUTs: Lookup Tables.

TABLE. IV. FPGA Implementation Results of Complex Binary Decoder-based Adder and base 2 Adder [5]

	Complex Binary Decoder-based Adder Implementation on FPGA Device	Base 2 Decoder-based Adder Implementation
	Virtex V50CS144	Virtex V50CS144
Number of external IOBs	20/94 (21%)	13/94 (13%)
Number of CLBs(Slices)	455/768 (59%)	391/768 (50%)
Number of 4 input LUTs	857/1536 (55%)	755/1536 (49%)
Number of bonded IOBs	20/94 (21%)	13/94 (13%)
Gate count	5142	4530
Average connection delay (ns)	3.179	3.169
Maximum combinational delay (ns)	32.471	28.442

TABLE. V. FPGA IMPLEMENTATION RESULTS OF COMPLEX BINARY MINIMUM-DELAY ADDER [10]

	Complex Binary Minimum-Delay Adder Implementations on FPGA Devices		
	Virtex4 XC4VLX15	Virtex5 XC5VLX30	Virtex XCV100
Number of external IOBs	20/240 (8%)	20/220 (9%)	20/180 (11%)
Number of CLBs(Slices)	27/6144 (>1%)	25/19200 (1%)	27/1200 (2%)
Number of 4 input LUTs	52/12288 (>1%)	0/27	52/2400 (2%)
Number of bonded IOBs	20/240 (6%)	20/220 (9%)	20/180 (11%)
Gate count	330	175	330
Maximum net delay (ns)	5.028	2.790	8.856
Maximum combinational delay (ns)	7.827	4.776	17.001

ACKNOWLEDGMENT

We gratefully acknowledge the support provided by Sultan Qaboos University through Internal Research Grant No. IG/ENG/IENG/16/03 in facilitating and encouraging an environment conducive to research and academic excellence in the Sultanate of Oman.

REFERENCES

[1] T. Jamil, N. Holmes, and D. 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.

[2] T. Jamil, "The complex binary number system – basic arithmetic made simple," IEEE Potentials, Vol. 20, No. 5, pp. 39-41, 2002.

[3] D. Blest and T. Jamil, "Division in a binary representation for complex numbers," International Journal of Mathematical Education in Science and Technology, Vol. 34, No. 4, pp, 561-574, 2003.

[4] T. Jamil, "Complex Binary Number System – Algorithms and Circuits," Springer, 2013.

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

[6] B. Arafeh, T. Jamil, and A. AlHabsi, "A Nibble-size Ripple-carry Adder for (-1+j)-base Complex Binary Numbers," Proceedings of the International Arab Conference on Information Technology (ACIT 2002), Doha (Qatar), 16-19 December 2002, Vol. I, pp. 207-211.

[7] T. Jamil, B. Arafeh, and A. AlHabsi, "Design of a Nibble-size Adder for (-1+j)-base Complex Binary Numbers," Proceedings of the 6th World Multiconference on Systemics, Cybernetics, and Informatics (SCI 2002), Orlando, Florida (USA), 14-18 July 2002, Vol. V, pp. 297-302.

[8] www.32x8.com

[9] www.xilinx.com

[10] T. Jamil, M. Awadalla, and I. Mohammed, "Design and Implementation of a Complex Binary Adder," Medwell Journal of Engineering and Applied Sciences, ISSN: 1818-7803, Vol. 13, No. 7, 2018, pp. 1813-1828.