AUT Journal of Electrical Engineering

AUT J. Elec. Eng., 53(1) (2021) 1-10
DOI: 10.22060/eej

# New Insight on the Application of Binary Coded Chiliad (BCC) Encoding for Decimal Arithmetic

M. Dorrigiv[1,*]

[1] Department of Electrical and Computer Engineering, Semnan University, Semnan, Iran

**ABSTRACT:** The densely packed decimal (DPD) encoding for secondary and primary storage of three binary coded decimal (BCD) digits is included in the IEEE 754-2019 standard for decimal floating-point arithmetic. Binary coded chiliad (BCC) representation of 3 BCD digits (i.e. radix-1000) will achieve equi-efficient packing. The primary advantage is BCC operands can be directly manipulated by arithmetic operations, while DPD operands have to undergo DPD-to-BCD and reverse conversions afore and ahead of each arithmetic operation. Therefore, we are interested in designing the arithmetic unit that receives BCC operands and produces BCC results, following previous BCC works. Compared to the equivalent BCD or other radix-10 arithmetic, prospects show that equally efficient arithmetic units are feasible for BCC arithmetic, as even better performance has been achieved in the case of addition. Therefore, we demonstrate the IEEE 754-2019 compatibility of the BCC Encoding in this paper. Consequently, for the DPD-to-BCD expansion and the reverse compression, the DPD-to-BCC converter, and the reverse blocks, we show the delay, area, and power dissipation. The findings show a substantial delay (83%), area (27%), and power (29%) overhead. However, as the number of conversions in the latter case is much less than the former, overall power dissipation is expected to decrease considerably.

**Review History:**
Received:
Revised:
Accepted:
Available Online:

**Keywords:**
Binary coded chiliad encoding
decimal computer arithmetic
IEEE 754-2019
densely packed decimal encoding
encoding conversion

## 1. INTRODUCTION

Many ancient civilizations' numeral systems use ten and their powers to represent numbers, probably for there are ten fingers on two hands and people have begun to count with their fingers. Binary representation is also used internally by most modern computer hardware and software systems. Although many early computers, such as the ENIAC or the IBM 650, used decimal representation internally [1].

A long-standing practice is the implementation of decimal arithmetic operations on binary processors, which has its origins in the normal form of arithmetic for humans. Another justification is that, for many commercial and banking applications, the binary representation of certain decimal values is not sufficiently accurate [2]. However, at the dawn of the digital computer industry, the lack of 10-level logic devices compelled designers to use software-simulated radix-10 arithmetic operations carried out on binary digital processors [3]. Afterward, design engineers attempted to build specialized hardware units for decimal arithmetic operations with the rapid development of the computer industry.

In response to the new demands for high-performance decimal computations, research on the hardware realization of decimal arithmetic has been revived in the past two decades. Hardware decimal units and decimal arithmetic instruction subsets comprise several recently commercialized general-purpose digital processors. For instance, IBM included a decimal floating-point hardware unit in the z-196 [4] server chips and increased it in the next z-family servers [5]. On each of the cores on the 12-core chip of the newest z15 servers, the decimal floating-point (DFP) accelerator feature is present. Fujitsu also announced the Sparc64 X processor that includes an accelerator called SWoC (Software on Chip) to speed up cryptography and decimal calculation operations [5]. Correspondingly, SilMinds' primary focus has also been on creating an extensible range of decimal floating-point arithmetic IP cores for financial applications over the past few years [6].

Binary coded decimal (BCD) encoding of radix-10 numbers is commonly used for the implementation of decimal arithmetic on digital processors. However, the encoding efficiency of this representation is rather low (i.e., $10/16 = 0.625$), which results in a waste of storage. To remedy this problem, one of the solutions that are recommended by IEEE 754-2019 (revision of IEEE 754-2008) standard for decimal floating-point (FP) arithmetic [7] is the densely packed decimal (DPD) 10-bit encoding of three BCD digits, where encoding efficiency rises to $1000/1024 = 97.66\%$. Thereafter, almost all of the hardware realizations of decimal arithmetic operators that we have encountered [8-15], assume DPD-encoded inputs and outputs. However, decimal arithmetic operations cannot

*Corresponding author's email: email

**Table 1. DPD Encoding (Compression from 3 BCD digits)**

| Case | $B$ | $C$ | $D$ | $aei$ | $pqr$ | $stu$ | $v$ | $wx$ | $y$ |
|---|---|---|---|---|---|---|---|---|---|
| 1 | $\mathcal{S}$ | $\mathcal{S}$ | $\mathcal{S}$ | 000 | $bcd$ | $fgh$ | 0 | $jk$ | $m$ |
| 2 | $\mathcal{S}$ | $\mathcal{S}$ | $\mathcal{L}$ | 001 | $bcd$ | $fgh$ | 1 | 00 | $m$ |
| 3 | $\mathcal{S}$ | $\mathcal{L}$ | $\mathcal{S}$ | 010 | $bcd$ | $jkh$ | 1 | 01 | $m$ |
| 4 | $\mathcal{L}$ | $\mathcal{S}$ | $\mathcal{S}$ | 011 | $jkd$ | $fgh$ | 1 | 10 | $m$ |
| 5 | $\mathcal{L}$ | $\mathcal{L}$ | $\mathcal{S}$ | 100 | $jkd$ | $00h$ | 1 | 11 | $m$ |
| 6 | $\mathcal{L}$ | $\mathcal{S}$ | $\mathcal{L}$ | 101 | $fgd$ | $01h$ | 1 | 11 | $m$ |
| 7 | $\mathcal{S}$ | $\mathcal{L}$ | $\mathcal{L}$ | 110 | $bcd$ | $10h$ | 1 | 11 | $m$ |
| 8 | $\mathcal{L}$ | $\mathcal{L}$ | $\mathcal{L}$ | 111 | $00d$ | $11h$ | 1 | 11 | $m$ |

be performed on DPD operands before expanding to BCD or other arithmetic-friendly encodings that are directly manipulatable (e.g., 4-2-2-1 [16]); hence cost and delay overhead due to the required conversions. For example, DPD-to-BCD expansion and the reverse compression require four gates at each end of the critical delay path, which leads to 10.5 FO4 delay overhead per operation, while for instance two $16$-digit FP BCD operands can be added in $26.1$ FO4 time [10]; hence 29% delay overhead.

Each three BCD digits, such as $B\ C\ D = 100B + 10C + D$ that represent $[0, 999]$, can be equivalently encoded as a $10$-bit radix-1000 digit [17] to be hereafter referred to as binary coded chiliad (BCC) digit [18]. The BCC encoding efficiency is the same as in the DPD encoding. However, it has been shown that efficient decimal adders can be realized that directly manipulate BCC operands [18], with the obvious advantage that the aforementioned conversion overhead per operation is removed. Nevertheless, such conversions are required only as I/O processing at the input and output ports of the processor. It surely is the case that if a particular computation dictates one conversion per arithmetic operation (as is the case for all DPD applications) our BCC scheme would not be recommended. However, note that the decimal arithmetic hardware has been realized in commercial processors in response to demands of, for instance, monetary and billing applications which are known for undertaking several fixed-point operations before reporting a result. Accordingly, we have used TELCO as a benchmark for our evaluations (see Section 6).

The conditional speculative mixed BCD/binary addition scheme of [19, 20] have extended to radix-1000 operands in [18] as the opening work on BCC addition. To decide on $+24\left(=2^{10}-10^{3}\right)$ speculation through using the 7 most significant bits (MSBs) of similarly weighted digits of addition operands. The second work we have encountered [21] studies all other possible speculation possibilities (i.e., besides from the 7-bit case of [18], all the other five cases are based on 2-6 bits).

The notations are used for delay and cost measures throughout the paper, are $\ddot{a}_g$ and $\#_g$ (delay and cost of a simple gate). Subsequently, circuit synthesis will be used to indorse the estimates, as the former model overlooks the consequence of fan-out.

This paper expands our previous research on BCC arithmetic [22] by (1) providing new details on the BCC Encoding compatibility of IEEE 754-2019, (2) adding a case study to support the proposed architecture as a benchmark., and (3) presentation and analysis of results that estimate the number of required DPD-to-BCD and reverse conversions in the conventional DPD-only processing environment concerning the BCC case due to BCC-to-DPD input and reverse output conversions.

The remainder of this work is structured as follows. A history of the IEEE 754-2019 standard is given in section 2. The BCC encoding is briefly covered in Section 3. Section 4 addresses the effect of BCC encoding on the addition of FP and the compatibility of IEEE 754-2019. Implementation details are provided in Section 5. Section 6 is devoted to defining the TELCO benchmark as a case study to analyze the architectures, and Section 7 offers final remarks.

## 2. IEEE 754-2019 STANDARD

The IEEE 754-2008 standard revised the IEEE 754-1985, for FP representation and arithmetic [7], and provides two standards for storage of decimal FP numbers, of which the DPD is popular in hardware realization of decimal FP units. Mike F. Cowlishaw devised DPD in 2002 [23] as an enhancement of Chen–Ho encoding, which was incorporated into the IEEE 754-2019 [7] and IEE/ISO/IEC 60559:2020 [24] standards for DFP. It uses a Huffman code, picking several combinations of digits by leading indicator bits. Similar to Chen-Ho encoding, DPD categorizes each decimal digit into one of two classes:

- $\mathcal{S}$ ("small" digits 0 through 7): Three more bits are required to specify the value of small digits after it is known that a digit is in class $\mathcal{S}$.
- $\mathcal{L}$ ("large" digits 8 and 9): One bit is required to differentiate between the values 8 (i.e., 1000) or 9 (i.e., 1001), once a digit in class $\mathcal{L}$ has been indicated.

The decoding (DPD to 3 BCD digits expansion) and encoding (3 BCD digits to DPD compression) patterns are described in Tables 1 and 2, respectively. Starting with the MSB (i.e., $a$) of the most significant digit (denoted as MSD), the 12 bits of three BCD digits (i.e., $B$, $C$, and $D$) have been represented by letters $a$ through $m$ (excluding $l$, for clarity). Once more starting with the MSB (i.e., $p$), the 10

**Table 2. DPD Decoding (Expansion to 3 BCD Digits)**

| Case | $vwxst$ | $B = abcd$ | $C = efgh$ | $D = ijkm$ |
|---|---|---|---|---|
| 1 | $0\cdots\cdot$ | $0pqr$ | $0stu$ | $0wxy$ |
| 2 | $100\cdot\cdot$ | $0pqr$ | $0stu$ | $100y$ |
| 3 | $101\cdot\cdot$ | $0pqr$ | $100u$ | $0sty$ |
| 4 | $110\cdot\cdot$ | $100r$ | $0stu$ | $0pqy$ |
| 5 | 11100 | $100r$ | $100u$ | $0pqy$ |
| 6 | 11101 | $100r$ | $0pqu$ | $100y$ |
| 7 | 11110 | $0pqr$ | $100u$ | $100y$ |
| 8 | 11111 | $100r$ | $100u$ | $100y$ |
| | [a] NB: "·" stands for don't care. | | | |

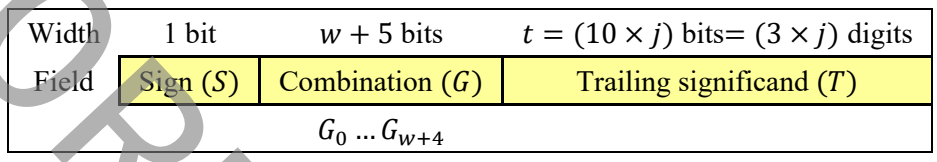


**Fig. 1. The storage format of DFP numbers in [7]**

**Table 3. Parameters for defining the basic DFP number format of Fig. 1**

| Format name | Decimal-32 | Decimal-64 | Decima-l28 | Decimal-$k$ ($k \geq 32$) |
|---|---|---|---|---|
| Storage width ($k$) | 32 | 64 | 128 | $1 + (w + 5) + t$ |
| Trailing significand field width ($t$) | 20 | 50 | 110 | $15 \times k/16 - 10$ |
| Combination field with ($w + 5$) | 11 | 13 | 17 | $k/16 + 9$ |
| Number of significand digits ($p$) | 7 | 16 | 34 | $9 \times k/32 - 2$ |
| Exponent bias, $E - q$ | 101 | 398 | 6176 | $E_{\max} + p - 2$ |
| $E_{max}$ | +96 | +384 | +6144 | $+3 \times 2^{k/16+3}$ |
| $E_{min}$ | −95 | −383 | −6143 | $-3 \times 2^{k/16+3} + 1$ |

bits of the encoded DPD digits are represented by letters $p$ through $y$.

During compression, for instance, if the decimal number is 399 with two $\mathcal{L}$ digits (namely, 0011 1001 1001), then the 7th row of Table 1 describes the pattern to be used (because the sequence $aei$ is 011). Therefore, the indicator bits, $v$, and $wx$, are set to 1 and 11 in this row, and therefore the encoding is 011 101 1 111. Similarly, if the decimal numbers were 020 (namely, 0000 0010 0000, with three $\mathcal{S}$ digits) then the 1st row of Table 1 describes the output 000 010 0 000.

As 24 of the 10-bit DPD values (i.e., 1024) are unused, any of the four possible combinations could have been for the values $pq$ in the 8th row of Table 1.

During expansion, if the encoded bits are 011 101 1 111, which are corresponding by the 7th row of Table 2 (since $vwxst$ bits are 11110), hence giving 0011 1001 1001. Therefore, reversing the first compression example.

Implementation details for mappings described by Tables 1 and 2 are provided in Section 5.

A DFP number is encoded in $k$ bits (a finite DFP number $\{S, q, C\}$, infinity or a NaN) using the following three fields, detailed in Fig. 1:

- $S$: A 1-bit sign field, which encodes the coefficient sign.
- $G$: A combo field of $(w+5)$-bit, containing the binary biased exponent of $(w+2)$-bit $E = q + bias$ and the 4 most significant $p$-digit coefficient bits. The value of the exponent's 2 most significant bits can't be 3.
- $T$: A $(10\times j)$-bit trailing significand field, encoding $p-1=3\times j$ trailing digits of significand using DPD, or binary integer values using BID from $0$ to $2^{10\times j-1}$.

For example, Table 3 shows the format encoding parameter values corresponding to the basic decimal formats of [7].

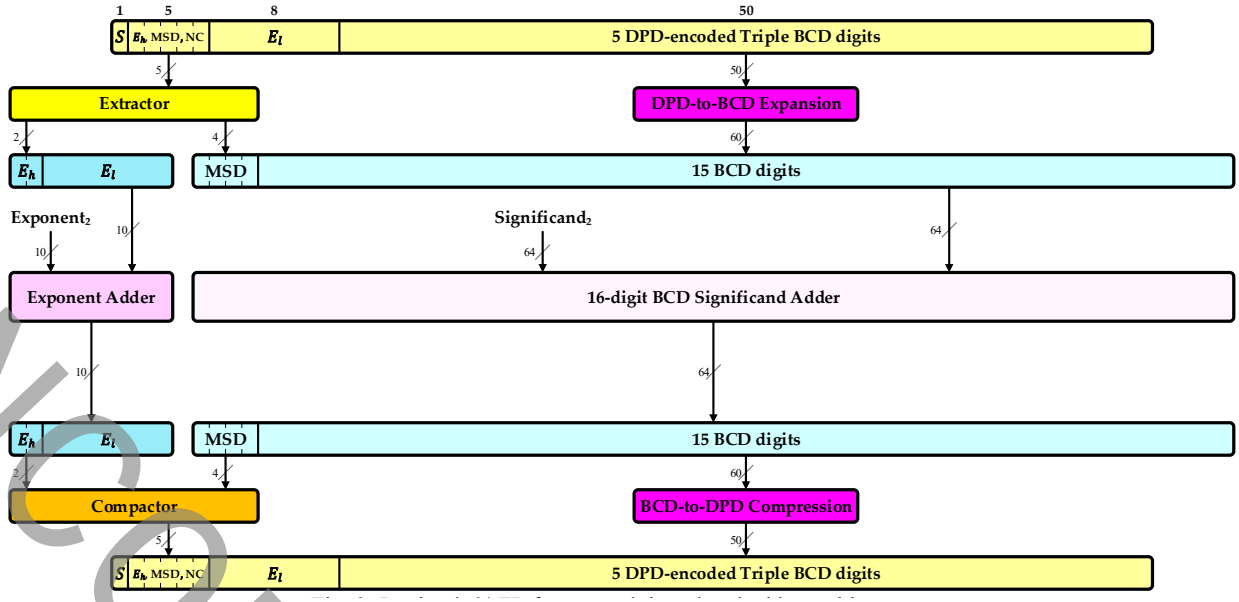


**Fig. 2. Decimal-64 FP format and the related adder architecture**

Decimal number formats are specified for any multiple of 32 bits of at least 32 bits.

## 3. RADIX-1000 REPRESENTATION OF DECIMAL NUMBERS

Each 3 BCD digits (e.g., $B\ C\ D = 100B + 10C + D \in [0, 999]$) that entail storage can be identically represented in 10 bits as a number in radix-1000. Therefore, the efficiency of such encoding that is known as binary coded chiliad (also known as declet or binary coded millennium [17]) is alike to DPD. Nevertheless, BCC numbers can directly be manipulated by arithmetic operators (opposite to DPD), that no expansion/compaction is needed on retrieval of operands and storage of results in/from processor's registers and memory; henceforth substantial latency and power savings are gained. Nonetheless, only one conversion from IEEE 754-2019 standard DPD inputs to BCC is needed at the input ports, and just at the output ports the reverse conversion is required.

The best radix-10 adder for BCD operands that uses conditional speculation [19], has been already extended for BCC operands [18], where speculation constant of BCC (i.e., $24 = (11000)_2$) has three trailing zero, while that of BCD (i.e., $6 = (110)_2$) has one in binary representation. Therefore, the three (in contrast to one in BCD) least significant bits of BCC operands have been left out in the evaluation of speculation condition. Moreover, simpler logical expressions for asserting the speculation condition have been proposed in [21]. To do this, more trailing bits of each BCC digit are leaving out. As such, [21] show 30%, 27%, and 17% advantages in power, area, and power delay product (PDP) measures, respectively. Moreover, lower time constraints are met by the proposed designs. Notice that these enhancements are only due to BCC addition and do not show any additional savings resulting from the absence of DPD-to-BCC and the reverse conversions before and after each arithmetic operation.

To conclude, we notice that within the discussion sessions for determining the representation of decimal numbers in FP as a portion of the IEEE 754-2008 standard (superseded by IEEE 754-2019 [7]), it has been proposed to use the radix-1000 representation of decimal numbers, but not endorsed [17].

## 4. THE IMPACT OF BCC ENCODING ON FLOATING-POINT ADDITION AND THE IEEE 754-2019 COMPATIBILITY

Most of the DFP units that are implemented in the industrial solutions opted for the BCD representation as an internal format to have efficient decimal computations. Therefore, while complying with the DPD standard, we are inspired to design a decimal hardware architecture that can use BCC for intermediate results. Following the common practice of DPD-to-BCD expansion and the reverse compression, before and after decimal arithmetic operations, we present the required DPD-to-BCC and BCC-to-DPD conversions and study their impact on decimal addition.

Some decimal arithmetic applications extensively use fixed-point decimal data (e.g., accounting [25]), where no particular problem occurs in using BCC encoding. In this section, we discuss the peculiarities of the BCC encoding on FP decimal addition. Fig. 2 depicts the Decimal-64 FP format and the related adder architecture, where the 50-bit significand is due to fifteen less significant DPD encoded BCD digits, the most significant BCD digit (denoted as MSD) is embedded in the combination field, and details of extraction and compaction circuits are shown in Section 5 (see Fig. 4). This 13–bit combination field also contains the exponent (embedded two MSBs $E_h$ and eight least significant bits $E_l$) and number classification (NC) information (i.e., NaN cases and $\pm\infty$).

Since the trailing 50 bits of the significand are not directly

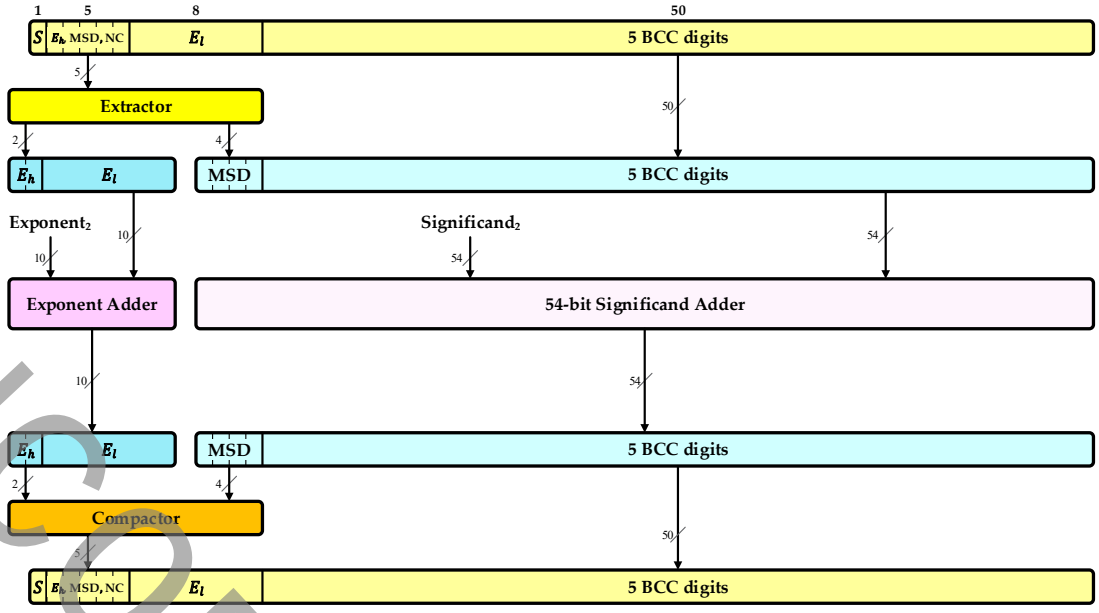


Fig. 3. BCC-64 FP format and the related adder architecture

manipulatable, DPD-to-BCD (or other arithmetic friendly encoding of decimal digits) expansion and the reverse compression are required before and after each arithmetic operation. This leads to extra power dissipation and $4\ \delta_g$ delay overhead per conversion. Additionally, wider or extra registers are required to accommodate the converted significand (e.g., 64 bits in case of BCD) and extracted exponent (e.g., 10 bits in case of Decimal-64). For instance, some research reports have used 75-bit [12] or 83-bit [9] specialized registers for decoded Decimal-64 DPD operands. Additionally, a DPD-based commercial realization of Decimal-128 uses separate registers for significand (144-bit) and exponent (16-bit) [5].

### 4-1- The Intermediate BCC FP Format

The forward DPD-to-BCC conversion, only affects the trailing 50 bits of DPD significand, which is to be converted to five BCC digits comprising to trailing 50 bits of BCC significand. Hence, the resulted BCC-64 FP operand perfectly fit in the commonly used 64-bit registers and memory words (see Fig. 3), while the trailing 50 bits (i.e., five BCC digits) are readily available to be directly manipulated. The MSD and exponent extraction, out of the combination field, and the reverse compaction are undertaken exactly in the same way as in the DPD-to-BCD expansion and the reverse compression. The cost and delay of forward extraction (backward compaction) as can be easily Figured out from [7, 23], and shown in Section 5, correspond to only seven and two gates (six and two gates), respectively.

Note that the MSD extraction is obviously off the critical delay path of addition operation in both DPD and BCC cases. However, the combination field compaction is off the critical delay path only in the BCC case, since the MSD (i.e., a BCD digit) is available two gates earlier than BCC digits (see [21]). Nevertheless, the main advantages of BCC encoding are:

1. The BCC significand is directly manipulatable. No interoperation conversion is required until a result is to be reported to an output device, where BCC-to-DPD conversion is required. The required logic for DPD-to-BCC and the reverse conversions, which can be implemented within the I/O processor, are given in Section 5.
2. The significand adder is 54-bit wide in comparison to a 64-bit adder in the DPD case.

### 4-2- BCC Exponent Base

When the exponent difference of two BCC FP addition operands is not a multiple of 3, the required alignment shift operation is nontrivial and rather complex. The same problem, noted in addition to two BID-encoded [7] decimal FP operands [26] may be accure more seriously. However, the BCC alignment complexity can be mitigated by allowing the BCC exponent base to be 1000; thus, leading to multiple-of-3 exponents and simple BCC shifts.

### 4-3- Mixed BCC/binary FP Adders

Double (quadruple) precision binary FP addition according to IEEE 754-2019 standard requires 53-bit (113-bit) binary adders, while the proposed BCC FP addition scheme is based on 54-bit (114-bit) binary adders for Decimal-64 (-128) operands. Therefore, the latter can be shared by the binary FP addition unit, where only one bit of the available capacity is not used. However, in the common case where the DPD operands are converted to 64-bit (136-bit) BCD numbers, 11 (23) bits of the additional capacity are unused, while normally contribute to additional power dissipation.

## 5. IMPLEMENTATION DETAILS

This section provides the required logical expressions for

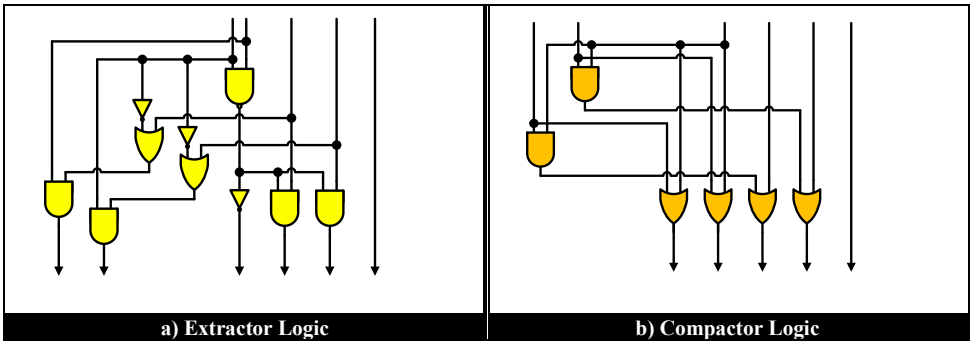


**Fig. 4. Details of the extractor (a) and compactor (b) blocks in Figs. 2 and 3**



the DPD-to-BCD expansion, BCD-to-DPD compression, extraction, and compaction. Moreover, details of the proposed DPD-to-BCC and BCC-to-DPD conversions are comprised.

### 5-1- DPD-to-BCD Expansion and BCD-to-DPD Compression

Since the DPD digits are not directly manipulatable, DPD-to-BCD expansion and reverse compression are required before and after each arithmetic operation. This requirement is depicted in the architecture of Decimal-64 FP of Fig. 2 and entails delay, cost, and power dissipation. Recall that each DPD digit $pqrstuvwxy$ expanded to three BCD digits $B = abcd$, $C = efgh$, and $D = ijkm$ before performing arithmetic operations in the architecture of Decimal-64 FP format. Likewise, reverse compression is required for storing the results.

The following logic expressions describing each DPD output bit during compression with $47\#_g$ cost and $4\ddot{a}_g$ delay:

$$p = (b \wedge \bar{a}) \vee (j \wedge a \wedge \bar{i}) \vee (f \wedge a \wedge \bar{e} \wedge i)$$

$$q = (c \wedge \bar{a}) \vee (k \wedge a \wedge \bar{i}) \vee (g \wedge a \wedge \bar{e} \wedge i)$$

$$r = d$$

$$s = \left(f \wedge \bar{e} \wedge \left(\overline{a \wedge i}\right)\right) \vee \left(j \wedge \left(\bar{a} \wedge e \wedge \bar{i}\right)\right) \vee (e \wedge i)$$

$$t = \left(g \wedge \bar{e} \wedge \left(\overline{a \wedge i}\right)\right) \vee \left(k \wedge \left(\bar{a} \wedge e \wedge \bar{i}\right)\right) \vee (a \wedge i)$$

$$u = h$$

$$v = a \vee e \vee i$$

$$w = a \vee (e \wedge i) \vee (j \wedge \bar{e} \wedge \bar{i})$$

$$x = e \vee (a \wedge i) \vee (k \wedge \bar{a} \wedge \bar{i})$$

$$y = m$$

Likewise, the logic expressions describing each output bit during expansion with $51\#_g$ cost and $4\ddot{a}_g$ delay are as follows:

$$a = (v \wedge w) \wedge (\bar{x} \vee \bar{s} \vee (s \wedge t))$$

$$b = p \wedge (\bar{v} \vee \bar{w} \vee (x \wedge s \wedge \bar{t}))$$

$$c = q \wedge (\bar{v} \vee \bar{w} \vee (x \wedge s \wedge \bar{t}))$$

$$d = r$$

$$e = v \wedge ((\bar{w} \wedge x) \vee (w \wedge x \wedge (s \vee \bar{t})))$$

$$f = (s \wedge (\bar{v} \vee (v \wedge \bar{x}))) \vee (p \wedge v \wedge w \wedge x \wedge \bar{s} \wedge t)$$

$$g = (t \wedge (\bar{v} \vee (v \wedge \bar{x}))) \vee (q \wedge v \wedge w \wedge x \wedge \bar{s} \wedge t)$$

$$h = u$$

$$i = v \wedge ((\bar{w} \wedge \bar{x}) \vee (w \wedge x \wedge (s \vee t)))$$

$$j = (w \wedge \bar{v}) \vee (s \wedge v \wedge \bar{w} \wedge x) \vee (p \wedge v \wedge w \wedge (\bar{x} \vee (\bar{s} \wedge \bar{t})))$$

$$k = (x \wedge \bar{v}) \vee (t \wedge v \wedge \bar{w} \wedge x) \vee (q \wedge v \wedge w \wedge (\bar{x} \vee (\bar{s} \wedge \bar{t})))$$

$$m = y$$

### 5-2- EXTRACTION AND COMPACTION

The MSD and exponent extraction, out of the combination field, and the reverse compaction in the architecture of Fig. 3 are undertaken exactly in the same way as in the architecture of Fig. 2. The cost and delay of forward extraction as is depicted in Fig. 4 (a) (backward compaction as is depicted in Fig. 4 (b)) corresponds to only $7\#_g$ and $2\delta_g$ ($6\#_g$ and $2\delta_g$), respectively.

### 5-3- The Proposed DPD-to-BCC and BCC-to-DPD Conversions

This subsection provides the required logic for DPD-to-BCC and BCC-to-DPD conversions, which can be implemented within the I/O processor. Fig. 5 depicts a straightforward solution for DPD-to-BCC (a) and BCC-to-DPD (b), where BCD serves as an intermediate format. DPD-

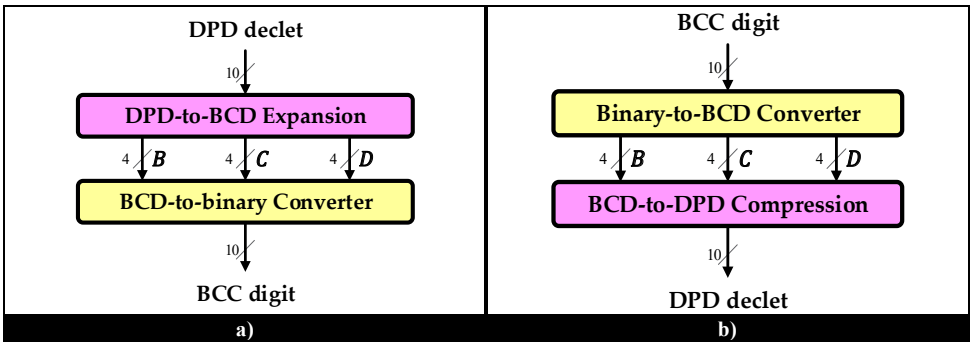


Fig. 5. The proposed DPD-to-BCC (a) and BCC-to-DPD (b) converters

to-BCD expansion and the reverse compression are the same as ones in subsection 5.1 and logics for BCD-to-binary and the reverse converters are given here.

Fig. 6 depicts a weighted bit set (WBS) that collectively represents the binary equivalent of three BCD digits $B\ C\ D$, which arithmetic's value can be expressed as in (1), where $abcd$, $efgh$, and $ijkm$, represent the equivalent of $B$, $C$, and $D$, respectively.

$$100B + 10C + D = 800a + 400b + 200c + 100d + 80e + 40f + 20g + 10h + 8i + 4j + 2k + m \quad (1)$$

The 4-deep WBS is reduced to a 2-deep one via half adders, full adders, and (4; 2) compressors, as appropriate, followed by an 8-bit adder that produces the final BCC digit. Bit dependencies such as $ab = 0$ and $eg = 0$, and the like are used to simplify the required logic.

The reverse BCC-to-BCD converter is illustrated in Fig. 7, which is designed based on the work of [27].

### 5-4- Synthesis Results

We have described all the blocks listed in Figs. 2 and 3 with HDL for a more accurate evaluation and also for verification (with 100,000 random inputs to check accuracy). Such codes are used by Synopsys Design Compiler to synthesize all units through the standard method of TSMC 0.90 $\grave{\imath}m$ CMOS technology under normal operating conditions (core voltage 1.2 $V$ and operating temperature 25 °C).

Table 4 shows the delay, area, and power dissipation for the DPD-to-BCD expansion and the reverse compression, the DPD-to-BCC converter, and the reverse blocks. As was expected, the results show a substantial delay, area, and power overhead. However, note that total power dissipation is expected to considerably reduce since the number of conversions in the latter case is far less than that of the former. To illustrate this reduction in the number of needed conversions, Section 6 studies a case study.

## 6. CASE STUDY

The TELCO benchmark is a decimal number system based on telephone billing application developed by IBM to analyze the balance between I/O time and calculation

| $2^9$ | $2^8$ | $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
|---|---|---|---|---|---|---|---|---|---|
| $a$ | $a$ | $b$ | $e$ | $f$ | $e$ | $i$ | $j$ | $k$ | $m$ |
| | $b$ | $c$ | $c$ | $a$ | $g$ | $f$ | $g$ | $h$ | |
| | | | $d$ | $d$ | $b$ | $h$ | $d$ | | |
| | | | | | | $c$ | | | |
| $a$ | $u_8$ | $u_7$ | $u_6$ | $u_5$ | $u_4$ | $u_3$ | $u_2$ | $u_1$ | $m$ |
| | $v_8$ | $v_7$ | $v_6$ | $v_5$ | $v_4$ | | $v_2$ | | |
| $s_9$ | $s_8$ | $s_7$ | $s_6$ | $s_5$ | $s_4$ | $s_3$ | $s_2$ | $s_1$ | $s_0$ |

Fig. 6. BCD-to-binary converter block in Fig. 5

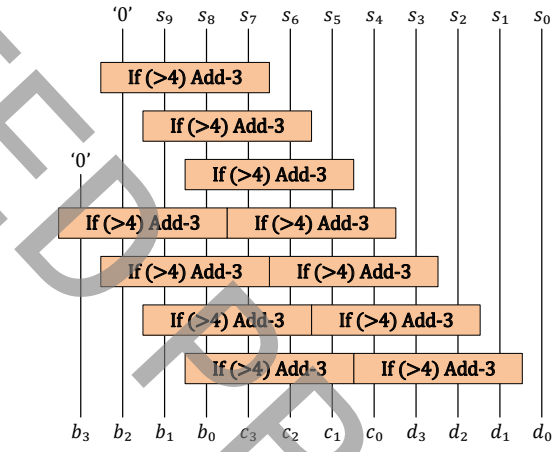


Fig. 7. 10-bit binary to 3-digit BCD numbers converter in block 4

time in a billing system for a telecommunications company [28]. The benchmark offers an example of a compliant set of multiplication and addition operations for IEEE 754-2019. The conversions from the IEEE-754 DPD to computationally efficient formats of BCD and BCC, which take place as a case study within the operations of a telephone billing application, through the TELCO benchmark, are the subject of our assessment.

Note that conversion errors might lead to incorrect

**Table 4. Synthesis Results for Converters**

| Converter | Critical Path | | Area | | Power | |
|---|---|---|---|---|---|---|
| | [*ns*] | *Ratio* | *NAND2* | *Ratio* | [*μW*] | *Ratio* |
| DPD-to-BCD | 0.175 | 1.00 | 1697 | 1.00 | 440 | 1.00 |
| BCD-to-DPD | 0.178 | 1.02 | 1785 | 1.05 | 479 | 1.09 |
| DPD-to-BCC | 0.320 | 1.83 | 2095 | 1.23 | 566 | 1.29 |
| BCC-to-DPD | 0.313 | 1.79 | 2160 | 1.27 | 518 | 1.18 |

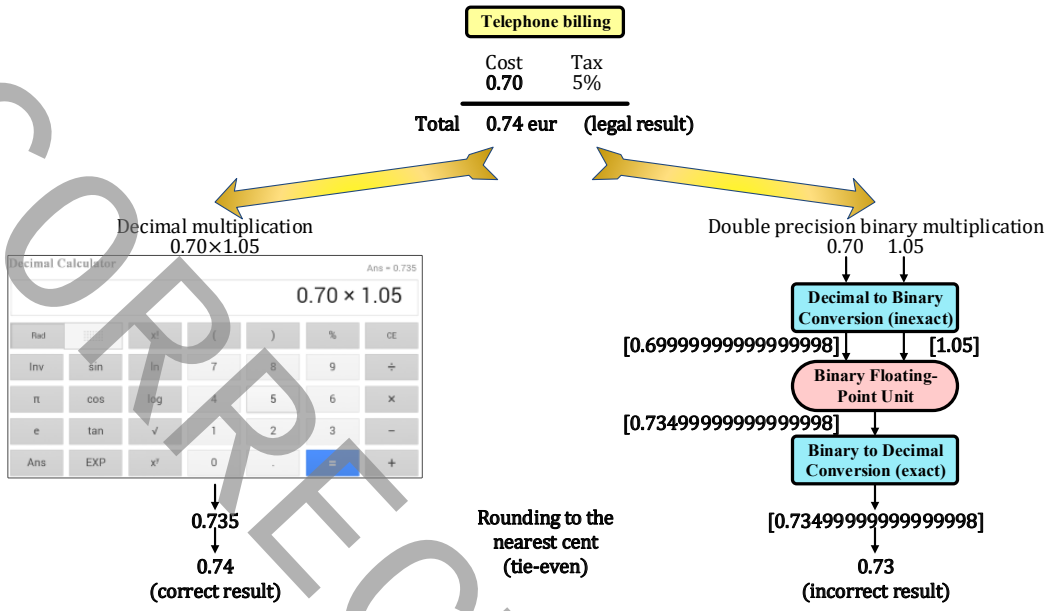


**Fig. 8. An example of calculating a decimal tax using a binary floating-point (reproduced from [20])**

results when a decimal calculation is performed using binary floating-point arithmetic. Fig. 8 presents an example of a 5% tax levied on a telephone call at a rate of EUR 0.70, rounded to the nearest cent (ties to even). The result is one cent less than expected (tax calculations are governed by law), using a double-precision binary floating-point for the calculation. These systemic one-cent errors add up, so the annual losses for a cell phone company with millions of calls a day could amount to over EUR 1 million [2].

Note that these errors are caused by a lack of binary floating-point precision and not by rounding. Although erroneous, these kinds of precisely rounded decimal results are enforced by legal and financial criteria and are the intended results of a decimal measurement.

Therefore, for financial calculations or any application based on decimal human-oriented arithmetic, binary floating-points cannot be used for they can neither comply with legal requirements nor provide precisely rounded decimal results.

For processing telephone bills, the TELCO benchmark executes the following code illustrated in Fig. 9 on decimal data, where the number of calls per telephone subscriber (i.e., $n$) is typically several thousand. The number of required DPD-to-BCD and reverse conversions amount to ($18n+3$) in the conventional DPD-only processing environment,

```
T = 0;
for i = 1 to n do
    P = secs × Drate;
    B = P × Btax;
    D = P × Dtax;
    C[i] = P + B + D;
    T = T + C[i];
end for;
```

**Fig. 9. Pseudo-code for the computations in TELCO benchmark**

while the BCC case is only ($2n+4$) due to BCC-to-DPD conversions at the input and reverse at the output.

The number of conversions for each number of calls is shown in Fig. 10 for the DPD and BCC architectures. The prescribed number of calls is 1000-10000 per 1000 call intervals. The gap between the DPD and the BCC curves clearly shows the superiority of the BCC encoding.

## 7. CONCLUSIONS

Densely packed decimal (DPD) encoding is the commonly used by one of the two encodings prescribed by the IEEE

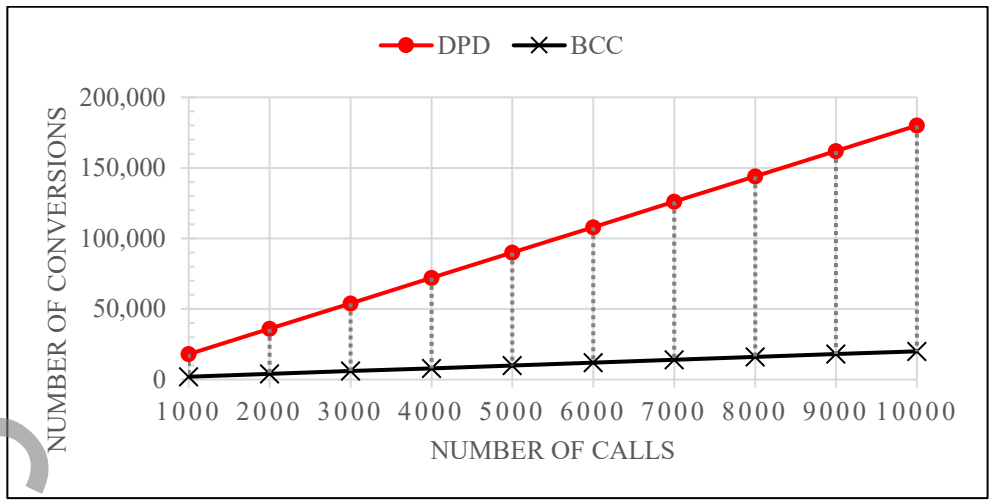


**Fig. 10. Number of conversions per number of calls for TELCO benchmark**

754-2019 standard for FP decimal number representation and arithmetic. Pre (post) conversion, to (from) arithmetic-friendly decimal encodings (for instance, BCD) is mandatory, for every arithmetic operation which results in additional latency, power, and area consumption. While abiding by IEEE 754-2019 DPD encoding for decimal data exchange and secondary storage, we spot that the inherently manipulaTable binary coded chiliad (BCC) encoding of decimal numbers can be favored for internal representation of decimal numbers due to succeeding properties:

Similar encoding effectiveness as DPD for storage of BCC operands (i.e., 10 bits per three BCD digits), in registers and primary memory.

• Latency, area, and power savings due to anticipating the necessity for pre/post conversions meant for every arithmetic operation. The only once required DPD-to-BCC and the reverse conversions at the I/O ports can be delegated to I/O processors, as is normally the case in DPD/BCD processing environment

• Prospects for BCC arithmetic that are at least equally efficient compared to the equivalent BCD or other radix-10 arithmetic, as in the case of addition even better performance has been achieved.

• BCC-64 (BCC-128) equi-width registers and binary double (quadruple) precision operands, which are important for the sharing of decimal/binary hardware.

• To mitigate the complexity of BCC FP alignment shifts, a base-1000 exponent format can be used.

Work is ongoing to investigate the impact of BCC encoding on the processor design as well as on unified add / subtract, multiplier, divider, and decimal FP units.

## NOMENCLATURE

| | |
|---|---|
| · | stands for don't care |
| $\mathcal{S}$ | "small" digits 0 through 7 |
| $\mathcal{L}$ | "large" digits 8 and 9 |
| $p$ | number of significand digits |
| $S$ | sign field |
| $G$ | combination field |
| $T$ | trailing significand |
| $E_l$ | embedded eight least significant bits of exponent $E$ |
| $E_h$ | embedded two MSBs of exponent $E$ |
| $E_{min}$ | the minimum value of exponent $E$ |
| $E_{max}$ | the maximum value of exponent $E$ |
| $ns$ | nanosecond |
| ì W | microwatt |
| $ä_g$ | delay of a simple gate |
| $\#_g$ | cost of a simple gate |

## REFERENCES

[1] W. Buchholz, "Fingers or fists? (the choice of decimal or binary representation)," in *Communications of the ACM*, pp. 3–11, 1959.

[2] M.F. Cowlishaw, "Decimal floating-point: algorism for computers," in *Proceedings of the 16th IEEE Symposium on Computer Arithmetic (ARITH '03)*, pp. 104–111, 2003.

[3] W.S. Brown, and P.L. Richman, "The Choice of Base," in *Communications of the ACM*, Vol. 12, pp. 560–561, 1969.

[4] S.R. Carlough, A. Collura, S.M. Mueller, and M. Kroener, "The IBM zEnterprise-196 Decimal Floating-Point Accelerator," in *Proceedings of the 20th IEEE Symposium on Computer Arithmetic (ARITH '11)*, pp. 139–146, 2011.

[5] C. Jacobi, and C. Webb, "History of IBM Z mainframe processors," in *IEEE Micro*, pp. 1–10, 2020.

[6] H.A.H. Fahmy, "Decimal Floating Point Number System," in *Embedded Systems Design with Special Arithmetic and Number Systems, Springer*, 2017.

[7] IEEE Standards Committee, "IEEE 754-2019 Standard for Floating-Point Arithmetic," Revision of IEEE 754-2008, IEEE Computer Society Standard, pp. 1–84, 2019.

[8] L.-K. Wang, M.J. Schulte, J.D. Thompson, and N. Jairam, "Hardware Designs for Decimal Floating-Point Addition and Related Operations," in *IEEE Transactions on Computers*, Vol. 58, No. 3, pp. 322–335, 2009.

[9] L.-K. Wang, and M.J. Schulte, "A Decimal Floating-Point Adder with Decoded Operands and a Decimal Leading-Zero Anticipator," in *Proceedings of the 19th IEEE Symposium on Computer Arithmetic (ARITH '09)*, pp. 125–134, 2009.

[10] Á. Vázquez, and E. Antelo, "A High-Performance Significand BCD Adder with IEEE 754-2008 Decimal Rounding," in *Proceedings of the 19th IEEE Symposium on Computer Arithmetic (ARITH '09)*, pp. 135–144, 2009.

[11] M.A. Erle, B.J. Hickmann, and M.J. Schulte, "Decimal Floating-Point

Multiplication," in *IEEE Transactions on Computers*, Vol. 58, No. 7, pp. 902–916, 2009.

[12] C. Minchola, and G. Sutter, "An FPGA IEEE 754-2008 Decimal Floating-Point Multiplier," in *International Conference on Reconfigurable Computing and FPGAs*, pp. 59–64, 2009.

[13] C. Lichtenau, S. Carlough, and S.M. Mueller, "Quad Precision Floating Point on the IBM z13," in *Proceedings of the 23rd IEEE Symposium on Computer Arithmetic (ARITH '16)*, pp. 87-94, 2016.

[14] A.A. Wahba, and H.A.H. Fahmy, "Area Efficient and Fast Combined Binary/Decimal Floating-Point Fused Multiply Add Unit," in *IEEE Transactions on Computers*, Vol. 66, No. 2, pp. 226-239, 2017.

[15] R. Mian, M. Shintani, and M. Inoue, "Cycle-Accurate Evaluation of Software-Hardware Co-Design of Decimal Computation in RISC-V Ecosystem," in *Proceedings of the 32nd IEEE International System-on-Chip Conference (SOCC)*, pp. 412-417, 2019.

[16] Á. Vázquez, E. Antelo, and P. Montuschi, "A New Family of High-Performance Parallel Decimal Multipliers," in *Proceedings of the 18th IEEE Symposium on Computer Arithmetic (ARITH '07)*, pp. 195–204, 2007.

[17] IEEE 754-2008 Working Group Mail Archive, available at http://grouper.ieee.org/groups/754/email/msg00801.html, 2003, retrieved on June 9, 2021.

[18] S. Emami, M. Dorrigiv, and G. Jaberipur, "Radix-10 Addition with Radix-1000 Encoding of Decimal Operands," in *Proceedings of the 16th CSI International Symposiums on Computer Architecture & Digital Systems*, pp. 139–144, 2012.

[19] Á. Vázquez, and E. Antelo, "Conditional Speculative Decimal Addition," in *Proceedings of the 7th Conference on Real Numbers and Computers*, pp. 47–57, 2006.

[20] Á. Vázquez, "High-Performance Decimal Floating-Point Units," Ph.D. dissertation, Univ Santiago de Compostela, 2009.

[21] M. Dorrigiv, and G. Jaberipur, "Conditional speculative mixed decimal/binary adders via binary-coded-chiliad encoding," in *Computers & Electrical Engineering*, Vol. 50, pp. 39–53, 2016.

[22] M. Dorrigiv, "The IEEE 754-2019 Compatibility of the Binary Coded Chiliad (BCC) Encoding," in *Proceedings of the 20th CSI International Symposiums on Computer Architecture & Digital Systems*, in print, 2020.

[23] M. Cowlishaw, "Densely packed decimal encoding," in *IEE Proceedings - Computers and Digital Techniques*, Vol. 149, No. 3, pp. 102–104, 2002.

[24] IEE/ISO/IEC 60559:2020, "ISO/IEC/IEEE International Standard - Floating-point arithmetic," *International Organization for Standardization*, pp. 1–86, 2020.

[25] J.M. Muller, N. Brisebarre, F. de Dinechin, C. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres, "Handbook of Floating-Point Arithmetic," *Birkhäuser*, 2010.

[26] C. Tsen, S. Gonzalez-Navarro, and M.J. Schulte, "Hardware Design of a Binary Integer Decimal-based Floating-Point Adder," in *Proceedings of the 25th IEEE International Conference on Computer Design*, pp. 288–295, 2007.

[27] J.D. Nicoud, "Iterative Arrays for Radix Conversion," in *IEEE Transactions on Computers*, Vol. C-20, No. 12, pp.1479–1489, 1971.

[28] IBM Corporation, "The 'telco' benchmark," available at http://speleotrove.com/decimal/telco.html, retrieved on September 27, 2020.

**HOW TO CITE THIS ARTICLE**

*Dorrigiv, M. (2021). New Insight on the Application of Binary Coded Chiliad (BCC) Encoding for Decimal Arithmetic. AUT J. Elec. Eng., 53(1): 1-10.*

**DOI:** 10.22060/eej***