A New Decimal Antilogarithmic Converter

Dongdong Chen*, Yu Zhang*, Daniel Teng*, Khan Wahid†, Moon Ho Lee†, Seok-Bum Ko*†
*Department of Electrical and Computer Engineering, University of Saskatchewan
Campus Drive 57, Saskatoon, SK, Canada, seokbum.ko@usask.ca
†Institute of Information and Communication, Chonbuk National University
Jeonju 561-756, Korea, moonho@chonbuk.ac.kr

Abstract—This paper presents a new design and implementation of a 32-bit decimal floating-point (DFP) antilogarithmic converter based on the digit-recurrence algorithm with selection by rounding. The converter can calculate the accurate antilogarithm \((10^{x_{\text{dec}}})\) of the 32-bit DFP numbers which are defined in the IEEE 754-2008 standard. The sequential architecture of the proposed 32-bit DFP antilogarithmic converter is implemented on Xilinx Virtex-II Pro P30 FPGA device. The proposed architecture occupies 2,315 out of 13696(16%) slices and can obtain a faithful 32-bit DFP antilogarithm in 11 clock cycles running at 51.5 MHz. The 7-digit decimal fixed-point (FXP) antilogarithmic converter is an essential operational part of the 32-bit DFP antilogarithmic converter. We transform it to a 7-digit decimal exponential converter to compare with a 24-bit binary FXP exponential converter. The compared results show that the 7-digit decimal exponential converter occupies 2.18 times more area and 1.66 times slower than the 24-bit binary FXP exponential converter.

I. INTRODUCTION

Nowadays, there are many commercial demands for DFP arithmetic operations such as financial analysis, tax calculation, phone billing, internet based applications, and e-commerce [1]. This trend gives rise to further development on DFP arithmetic unit which can perform more accurate calculations compared with a binary floating-point (BFP) arithmetic unit. Due to the significance of DFP arithmetic, the IEEE 754-2008 standard for floating-point arithmetic [2] includes it into the specifications. Decimal arithmetic unit, as a main part of a decimal processor, is attracting more and more researchers’ attention. The decimal-encoded formats and arithmetic have been implemented in IBM system z10 processor [3].

The logarithm and antilogarithm operations are useful arithmetic concept in many areas of science and engineering. Some applications, such as logarithmic number system (LNS) and digital signal processing, are implemented by using a logarithmic unit to replace the normal computer arithmetic. Moreover, the decimal logarithmic and antilogarithmic operation as a decimal arithmetic operation is defined in the new IEEE 754-2008 standard [2]. With the improvement of basic decimal arithmetic units, more complex DFP elementary operations such as logarithm, antilogarithm, exponential, trigonometric, etc. would be the next useful building blocks.

This paper is organized as follows: Section II presents an overview of a 32-bit DFP antilogarithm operation. In Section III, a 7-digit FXP decimal antilogarithmic digit-recurrence algorithm with selection by rounding and its related architecture are described. Section IV presents and analyzes the implementation and comparison results. In Section V, we summarize the conclusions.

II. 32-BIT DECIMAL FLOATING-POINT ANTILOGARITHM

The current IEEE 754-2008 draft specifies a 32-bit DFP format represented as:

\[ X = (-1)^{s} \times 10^{e} \times \text{coefficient} \]  

(1)

In (1), where \( s \) is in the range of \([-6, -59] \), \( e \) is in the range of \([-52, 62] \), and the coefficient represents an integer. If the DFP numbers with absolute values are larger than the largest DFP number \((X_{\text{max}} = 9999999 \times 10^{90})\) then overflow occurs; Similarly, if they are less than the smallest 32-bit DFP number \((X_{\text{min}} = 10^{-101})\) then underflow occurs. When the absolute value of DFP number is less than \(0 \times 10^{-101}\) and larger than \(0000001 \times 10^{-101}\), it will produce subnormal.

The 32-bit DFP antilogarithm operation is defined as:

\[ P = \text{Anti-log}_{10}(X) = 10^{X} \]  

(2)

When all 32-bit DFP numbers are smaller than \(\log_{10}(X_{\text{min}})\), the antilogarithm result, \( P \), satisfies the condition of underflow and returns the smallest 32-bit DFP number \((X_{\text{min}} = 10^{-101})\); when all 32-bit DFP numbers are larger than \(\log_{10}(X_{\text{max}})\), \( P \) satisfies the condition of overflow and return the largest DFP number \((X_{\text{max}} = 9999999 \times 10^{90})\). Moreover, if the \( X \) is \( \text{NaN} \) (Not-a-Number), the antilogarithmic converter will simply return \( \text{NaN} \), and if the \( X \) is infinite, the antilogarithmic converter will simply returns \( \text{Inf} \). Therefore, the 32-bit DFP antilogarithmic calculation is transformed to a FXP decimal antilogarithmic calculation as shown in equation (3), where \( X \in [-101, 96.99999] \). The \( 10^{X_{\text{frac}}} \) represents the coefficient of a 32-bit DFP antilogarithm and the \( X_{\text{int}} \) represents the exponent of the 32-bit DFP antilogarithm. We will then focus on the algorithm and the architecture of the 7-digit FXP decimal antilogarithmic converter which can produce 7-digit accurate antilogarithm results.

\[ \text{Anti-log}_{10}(X) = 10^{X_{\text{int}}} \times X_{\text{frac}} = 10^{X_{\text{int}}} \times 10^{X_{\text{frac}}} \]  

(3)

III. FXP DECIMAL ANTILOGARITHMIC CONVERTER

A. Overview of Algorithm

The antilogarithm of \( X_{\text{frac}} \) is achieved by (4), where \( X_{\text{frac}} \) is a 7-digit decimal FXP number with the range of \((-1, 1)\):

\[ 10^{X_{\text{frac}}} = e^{X_{\text{frac}} \times \ln(10)} \]  

(4)
To obtain $10^{X_{frac}}$, the digit-recurrence algorithm to calculate $e^m$ is summarized as (5), where $m = X_{frac} \times \ln(10)$ in the range of $(-\ln(10), \ln(10))$.

$$L[j+1] = m - \sum_{i=1}^{j} \ln(f_i)$$  \hspace{1cm} (5)

If (6) is satisfied:

$$m - \sum_{i=1}^{j} \ln(f_i) \to 0$$  \hspace{1cm} (6)

Then,

$$\sum_{i=1}^{j} \ln(f_i) \to m$$  \hspace{1cm} (7)

Thus,

$$10^{X_{frac}} = e^m = \prod_{i=1}^{j} f_i$$  \hspace{1cm} (8)

$f_i$ is defined as $f_i = 1 + e_j 10^{-j}$ by which $m$ is transformed to 0 by successive subtraction of $\ln(f_i)$. This form of $f_i$ allows the use of a decimal shift-and-add implementation.

According to (5) and (8), the corresponding recurrences for transforming $m$ and computing the antilogarithm are presented as follows:

$$L(j+1) = L[j] - \ln(1 + e_j 10^{-j})$$  \hspace{1cm} (9)

$$E(j+1) = E[j] \times (1 + e_j 10^{-j})$$  \hspace{1cm} (10)

In (9) and (10), $j \geq 1$, $L[1] = m$ and $E[1] = 1$. The digits $e_j$ are selected so that $L(j+1)$ converges to 0. After performing the last iteration of recurrence, the results are:

$$L(N+1) \approx 0$$  \hspace{1cm} (11)

$$E(N+1) \approx e^m = 10^{X_{frac}}$$  \hspace{1cm} (12)

To have the selection function for $e_j$, a scaled remainder is defined as

$$W[j] = 10^j \times L[j]$$  \hspace{1cm} (13)

Thus,

$$L[j] = W[j] \times 10^{-j}$$  \hspace{1cm} (14)

Substituting (14) into (9) yields

$$W[j+1] = 10W[j] - 10^{j+1} \times \ln(1 + e_j 10^{-j})$$  \hspace{1cm} (15)

According to (15), the digits $e_j$ are selected as a function of leading digits of scaled remainder in a way that the residual $W[j]$ remains bounded.

### B. Selection by Rounding

The selected redundant digits are achieved through rounding to the integer part of the scaled remainder indicated as (16), where $e_j \in \{-9, -8, -7, ..., 0, ..., 7, 8, 9\}$.

$$e_j = \text{round}(W[j])$$  \hspace{1cm} (16)

when $e_j$ is selected by rounding as in (16):

$$-0.5 \leq W[j] - e_j \leq 0.5$$  \hspace{1cm} (17)

Because $|e_{j+1}| \leq 9$,

$$-9.5 < W[j+1] < 9.5$$  \hspace{1cm} (18)

The equation (15) can be written as:

$$W[j+1] = 10(W[j] - e_j) - 10^{j+1} \times \ln(1 + e_j 10^{-j}) + 10e_j$$  \hspace{1cm} (19)

According to (17), (18) and (19), the numerical analysis is processed as follows:

$$10^{j+1} \times \ln(1 + e_j 10^{-j}) - 10e_j > -4.5$$  \hspace{1cm} (20)

$$10^{j+1} \times \ln(1 + e_j 10^{-j}) - 10e_j < 4.5$$  \hspace{1cm} (21)

The numerical analysis results show that when $j \geq 2$, the conditions (20) and (21) are satisfied. So the selection by rounding is only valid for iterations $j \geq 2$ and $e_1$ can be achieved by a look-up table $I$ which is created according to inequality (22):

$$-9.5 < W[2] = 100 \times m - 10^2 \times \ln(1 + e_1 10^{-1}) < 9.5$$  \hspace{1cm} (22)

Based on (22), the numerical analysis results show that 1) The look-up table can not achieve positive input numbers $m$. For tuning $m$ to negative number, the fraction part of positive DFP input number $X_{frac}$ should be firstly adjusted to negative by $X_{frac} - 1$ and its corresponding integer part $X_{int}$ is adjusted by $X_{int} + 1$. 2) Choosing 1-digit $e_1 \in \{-9, -8, -7, ..., 0, ..., 7, 8, 9\}$ can not create a look-up table to achieve all the input numbers $m$. Therefore the 4-digit of $e_1$ is extended so that all the input numbers $m$ can be achieved. The look-up table $I$ is constructed by a size of $2^5 \times 16\ ROM$ in which the values of $e_1$ is stored as shown in Table 1.

<table>
<thead>
<tr>
<th>Range of m</th>
<th>10's complement BCD</th>
<th>$e_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>[-0.08, 0.00]</td>
<td>&quot;0000000000000000&quot;</td>
<td>0.000</td>
</tr>
<tr>
<td>[-0.19, -0.09]</td>
<td>&quot;1001000000000000&quot;</td>
<td>-1.000</td>
</tr>
<tr>
<td>[-0.20, -0.30]</td>
<td>&quot;1000000000000000&quot;</td>
<td>-2.000</td>
</tr>
<tr>
<td>[-0.31, -0.44]</td>
<td>&quot;0111000000000000&quot;</td>
<td>-3.000</td>
</tr>
<tr>
<td>[-0.45, -0.59]</td>
<td>&quot;0110000000000000&quot;</td>
<td>-4.000</td>
</tr>
<tr>
<td>[-0.60, -0.77]</td>
<td>&quot;0101000000000000&quot;</td>
<td>-5.000</td>
</tr>
<tr>
<td>[-0.78, -0.88]</td>
<td>&quot;0100010100000000&quot;</td>
<td>-5.500</td>
</tr>
<tr>
<td>[-0.89, -1.00]</td>
<td>&quot;0100000000000000&quot;</td>
<td>-6.000</td>
</tr>
<tr>
<td>[-1.01, -1.13]</td>
<td>&quot;0011010000000000&quot;</td>
<td>-6.500</td>
</tr>
<tr>
<td>[-1.14, -1.29]</td>
<td>&quot;0011000000000000&quot;</td>
<td>-7.000</td>
</tr>
<tr>
<td>[-1.29, -1.37]</td>
<td>&quot;0010100011010000&quot;</td>
<td>-7.250</td>
</tr>
<tr>
<td>[-1.38, -1.47]</td>
<td>&quot;0010010100010000&quot;</td>
<td>-7.500</td>
</tr>
<tr>
<td>[-1.48, -1.57]</td>
<td>&quot;0010000100100100&quot;</td>
<td>-7.750</td>
</tr>
<tr>
<td>[-1.58, -1.69]</td>
<td>&quot;0010000000000000&quot;</td>
<td>-8.000</td>
</tr>
<tr>
<td>[-1.70, -1.82]</td>
<td>&quot;0001100110100000&quot;</td>
<td>-8.250</td>
</tr>
<tr>
<td>[-1.83, -1.98]</td>
<td>&quot;0001010101000000&quot;</td>
<td>-8.500</td>
</tr>
<tr>
<td>[-1.99, -2.16]</td>
<td>&quot;0001001010100000&quot;</td>
<td>-8.750</td>
</tr>
<tr>
<td>[-2.17, -2.26]</td>
<td>&quot;0001000101001010&quot;</td>
<td>-8.875</td>
</tr>
<tr>
<td>[-2.27, -2.38]</td>
<td>&quot;0001000000000000&quot;</td>
<td>-9.000</td>
</tr>
</tbody>
</table>
C. Approximation of Logarithm

The values of logarithm \( \ln(1 + e_j 10^{-j}) \) in (15) can be achieved by another look-up table II. However, with the number of iteration increasing, the size of the table will become prohibitively larger. Therefore, a method for reducing the size of table is necessary, which can achieve a significant reduction in the overall hardware requirement. Equation (23) is the series expansion of logarithm function \( \ln(1 + x) \):

\[
\ln(1 + x) = x - \frac{x^2}{2} + \ldots
\]

After iteration \( j = k_1 \), the values of \( \ln(1 + e_j 10^{-j}) \) can be approximated by \( e_j 10^{-j} \). Since this antilogarithmic converter aims to guarantee 7-digit accuracy, the series approximation can be used in the iterations when the constraint \( x^2 / 2 < 10^{-7} \) is met. Here \( x = e_j 10^{-j} \), so

\[
e_j^2 10^{-2j} / 2 < 10^{-7}
\]

The numerical analysis of (24) shows that after the fifth iteration, while the values of \( -\ln(1 + e_j 10^{-j}) \) does not need to be stored in table, the values of \( -e_j 10^{-j} \), instead, will be used for approximation. The look-up table II is constructed by a size of \( 27 \times 40 \) ROM.

D. Error Evaluation

A MATLAB simulation model completely consistent with the hardware implementation of the proposed 7-digit FXP antilogarithmic converter is set up based on the algorithm described above. The MATLAB simulation model proves that at least 10-digit precision is necessary for \( W[j] \) in order to obtain correct \( e_j \) during 8 iterations. Furthermore, 100,000 7-digit random decimal FXP numbers in the range of \((-1, 0]\) are simulated as test vectors in the MATLAB model. The maximum absolute error of calculation of \( 10^X_{frac} \) is in the range of \( -6.93 \times 10^{-8} \leq E_{absolute} \leq 8.99 \times 10^{-8} \).

E. Architecture

Fig. 1 shows a sequential architecture of the proposed 7-digit FXP decimal antilogarithmic converter \(^1\). The hardware implementation of this antilogarithmic converter includes two stages. The \( e_j \) is obtained with selection by rounding in the stage 1. After \( e_j \) is achieved, the antilogarithm results will be produced in the stage 2. All variables in this architecture are represented with 10’s complement number system. In the architecture, the 10-digit decimal CLA adder is implemented based on 1-digit decimal CLA adder described in literature [5]. The Constant Multiplier and Multi logic are implemented based on the literature [6].

1) Cycle Process: In the first clock cycle, the first 7-digit FXP decimal number \( X_{frac} \) is multiplied with 7-digit constant \( \ln(10) \) to obtain \( m \), then \( m \) is complemented with 8-bit zero to achieve \( 100 \times m \). In the first iteration (2\textsuperscript{nd} clock cycle), while the 4-digit \( e_1 \) is achieved from the look-up table I, the corresponding logarithm value of \( -\ln(1 + e_1 10^{-1}) \) is obtained from look-up table II. The logarithm value is shifted to the left to obtain \( -10^2 \times \ln(1 + e_1 10^{-1}) \) which is selected by Mux2. Since \( m \) is in the range of \([-\ln(10), 0]\), it is converted by 9's complement converter and selected by Mux3. Finally \( W[2] \) is obtained by subtracting \( 10^2 \times \ln(1 + e_1 10^{-1}) \) from \( 100 \times m \) in 10-digit CLA adder and \( e_2 \) is obtained by rounding \( W[2] \) to integer in Rounding logic. In the stage 2, \( e_1 \), obtained from stage 1, is chosen by Mux4 and complemented with zero and shifted in Shifter \((\times 10^2)\). Meanwhile, the value of 1 is chosen in Mux5. Finally, the exponential result \( E[2] \) of the first iteration is obtained by 10-digit CLA adder.

From the second to the fifth iteration (3\textsuperscript{rd} to 6\textsuperscript{th} clock cycle), \( 10xW[j] \) obtained from Shifter Reg is selected by Mux3 and its corresponding logarithm value of \( -\ln(1 + e_j 10^{-j}) \), obtained from look-up table II is shifted to the left to obtain

\(^1\)Note that the proposed architecture can be transformed to a decimal exponential converter by removing the Reg1 and Constant Mul.
$10^{23} \times \ln(1+e_1 10^{-j})$. Thus $W[j+1]$ (refer to (15)) is obtained by subtracting $10^{23} \times \ln(1+e_1 10^{-j})$ from $10W[j]$ in 10-digit CLA adder and $e_{j+1}$ is obtained by rounding $W[j+1]$ to the integer. In the stage 2, $e_j$, obtained from stage 1, is multiplied with previous $E[j-1]$ in Mult logics to achieve $e_j \times E[j-1]$. Then this value is shifted to obtain $10^{-j} e_j E[j-1]$. Finally, the exponential result $E[j]$ is obtained by adding $10^{-j} e_j E[j-1]$ and $E[j-1]$ in 10-digit CLA adder.

From the sixth to the eighth iteration ($7 \text{th}$ to $9 \text{th}$ clock cycle), the logarithm result is obtained by $e_4 10^{-j}$ rather than from look-up table II. The values of $10^{23} e_j 10^{-j}$ is subtracted by $10^{j} \times W[j]$ to achieve $W[j+1]$. In the stage 2, the process for obtaining $E[j]$ is the same as in previous iterations. After 8 times iterations (9 clock cycles), the final exponential results are obtained and they are faithful.

IV. ANALYSIS OF IMPLEMENTATION RESULTS

A 32-bit DFP antilogarithmic converter is modeled with VHDL and implemented in Virtex-II PRO P30 FPGA configuration. Due to the page limit, we go directly into the analysis of its implementation results. The proposed 32-bit DFP antilogarithmic converter is synthesized with XST and placed and routed by Xilinx ISE 9.1. It occupies 1 out of 16 GCLK I/O block, 66 out of 644 I/O blocks, and 2,315 out of 13696 slices. The maximum clock frequency and latency are 51.5 MHz and 11 clock cycles respectively. The critical path of the proposed architecture is in the stage 2 of the 7-digit DFP decimal antilogarithmic converter which is highlighted in Fig. 1 (dotted line) and the details of critical path are available in Table II.

TABLE II

DETAILS OF CRITICAL PATH.

<table>
<thead>
<tr>
<th>Reg6</th>
<th>Mult</th>
<th>Mul-4</th>
<th>Shifter</th>
<th>CLA</th>
<th>Round</th>
<th>Total(ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.599</td>
<td>7.839</td>
<td>1.539</td>
<td>1.100</td>
<td>6.794</td>
<td>0.545</td>
<td>19.42</td>
</tr>
</tbody>
</table>

The 7-digit decimal FXP antilogarithmic converter can be readily transformed to a 7-digit decimal FXP exponential converter by removing the Reg1 and Constant Mul from the proposed architecture. Since there is no comparable decimal exponential converter, we compare the 7-digit decimal FXP exponential converter with a 24-bit radix-8 binary FXP exponential converter [4], because 1) they have similar dynamic range for the normalized coefficients ($2^{23} < 10^7 < 2^{24}$); 2) they are implemented by same digit-recurrence algorithm with selection by rounding; and 3) the radix-10 is close to radix-8. For the purpose of comparison, the proposed decimal FXP exponential converter is synthesized with a TMSC 0.18-um standard cell library. The synthesis results show that the worse case path and area are 7.54 ns and 122,564.53 unit respectively. Since the timing and area evaluation units in [4] are $\tau$ and $fa$ (1 $\tau$ = the delay of 1-bit full adder, 1$fa$ = the area of 1-bit full adder), we use the same units to represent the delay and area of decimal FXP exponential converter in this paper.

The compared results are shown in Table III which indicates that the proposed architecture is 1.66 times slower and 2.18 times larger than the radix-8 binary FXP exponential converter, because the number, in the form of BCD code, is less efficient than binary number in the radix-8 binary FXP exponential converter and needs more resource to be implemented.

V. CONCLUSIONS

In this paper, we develop a 32-bit DFP antilogarithmic converter based on the digit-recurrence algorithm with selection by rounding. Furthermore, we compare the proposed 7-digit decimal FXP exponential converter with a 24-bit radix-8 binary FXP exponential converter. The compared results show that the decimal FXP exponential converter is slower and occupies more areas than the binary FXP exponential converter. The presented architecture, however, can be optimized to achieve a faster speed and occupy a smaller area. Moreover, the proposed 32-bit DFP antilogarithmic converter can be scaled to a 64-bit or 128-bit converter which is compliant with decimal64 or decimal128 format, by the larger CLA adders and ROM with more iterations.

ACKNOWLEDGMENTS

This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), the Korea Research Foundation, and a Korean Federation of Science and Technology Societies grant funded by the South Korean government (MOEHRD, Basic Research Promotion Fund).

REFERENCES