Hybrid Floating Point/Logarithmic Number System Processor

C Y Sheng1,2, R C Ismail2,3, S Z M Naziri2, M N M Isa2, S A Z Murad2, A Harun2

1Intel Malaysia Sdn. Bhd., Kulim Kedah, Malaysia
2School of Microelectronic Engineering, Universiti Malaysia Perlis, Malaysia
3School of Computing and Informatics, Albukhary International University, Malaysia

Abstract: Hybrid Floating Point/Logarithmic Number System processor is an Arithmetic Logic Unit with hybrid architecture in which its data computation involves Floating Point (FLP) and Logarithmic Number System (LNS). LNS processor has high performance but requires complicated hardware to support its function, especially LNS addition and subtraction. Therefore, hybrid processor is proposed to perform multiplication/division in LNS, addition/subtraction in FLP. Through merging FLP and LNS, data computation can be done in a faster, precise and less complicated way. The proposed research is a 32-bit Hybrid FLP/LNS processor, which involving 32-bit fixed point data format and 32-bit single precision FLP format. The EDA tools used in developing and simulating this project is based on Synopsys Design Compiler and Altera Quartus II, and the Hardware Description Language used is Verilog HDL. Logical synthesis of this project is done by using Synopsys Design Compiler and its area, timing and power are validated.

Keywords: Floating Point, Logarithmic Number System, Arithmetic Logic Unit, Hybrid

1. Introduction

The earliest invention of digital signal processing is Fixed Point (FP) number system. It is a fast and simple number system, but it has limited data range and precision. After that, FLP number system was proposed to substitute FP number system. Although FLP number system processor is slow and complex, it has wider magnitude range than FP system, which is $2^{-128}$ to $2^{128}$. Based on [1], LNS is proposed as an alternative to FLP number system. LNS is fast in multiplication and division, but quite complicated in addition and subtraction, because it requires a large lookup table in which its size grows exponentially with the number bits. To optimize the hardware speed and size, Hybrid FLP/LNS processor is proposed by Lai, as in [2] and [3]. Hence, the aims and motivations of the proposed research is to utilize the advantage of FLP in addition and subtraction, whereas utilize the advantage of LNS in multiplication and division.

2. Literature review

2.1 Floating Point

Compared to FP, FLP is better because of its extensive range of values and high precision. Its application is quite wide, it can be used in artificial neural network, digital signal processing, finance and some other fields. It can be used to calculate extremely small number like mass of electron and extremely large number like the distance between Sun and Earth. Since the system is slow and complex, the speed and performance are always be the most important parameter in designing FLP unit.
There is a design proposed in [4], attempting to propose a FLP unit architecture with improved performance. In this design, parallelism is added in the design to manipulate FLP numbers with the behaviour of FP which is meticulous. With this, the high-speed requirement is able to be achieved.

In 1985, a widely used Floating Point format was introduced, which is IEEE 754 FLP representation. There are two types of them, the first is single precision and the second is double precision. Double precision floating point consists of 64 bits whereas single precision consists of 32 bits, as shown in Table 1.

|                        | Sign | Exponent | Emin | Mantissa | Bias |
|------------------------|------|----------|------|----------|------|
| Single Precision       | 1    | 8        | +127 | 23       | 127  |
| Double Precision       | 1    | 11       | +1023| 52       | 1023 |

The single precision floating point structure is symbolize as follows [4]:

\[ X = (-1)^{x_s} (X_m \times 2^{x_e}) \]

\(x_s\): Sign bit of X.
\(X_m\): mantissa of X.
\(x_e\): exponent of X.

The content of IEEE 754 Floating Point Representation are sign bit, exponent bit and mantissa bit, as shown in Table 1 and Figure 1. The sign bit consists of only one bit, which is the MSB. In single precision FLP number, sign bit is bit 31 whereas in double precision, sign bit is bit 63. If the sign bit is 0, the value is positive. If the sign bit is 1, the value is negative. For exponent bit, single precision FLP number has 8 bit, which is from bit 30 down to bit 23.

2.2 Logarithmic Number System Arithmetic

Logarithmic arithmetic is already introduced since 1975 [6]. This arithmetic can overcome the slowness of multiplication and division. However, it has a tradeoff which is being complex in addition and subtraction. Hence, LNS is only restricted to some low-precision application. To perform addition and subtraction in LNS, large lookup table (LUT) is required. To increase its precision, the Read Only Memory (ROM) size must be increased as well. In 1999 [1], the Gravity Pipe Supercomputer (GRAPE) won a Nobel Prize, in which using the LNS arithmetic unit. There are also applications of LNS in speech recognition and DNA sequencing [1]. Other than that, LNS is also commonly used in Graphical Processing Unit (GPU) [7] and [8]. GPU performs a lot of heavy and massive calculations concurrently. Some example of calculations inside GPU are multiplication, division, powering, square,
square root, trigonometric function and reciprocate. These complicated operations are converted into step-by-step calculation in multiplication and division. LNS is simpler in multiplication and division because multiplication and division in linear is converted into addition and subtraction in logarithmic domain.

The LNS representation in Figure 2 of a number \( x \) consists of the number’s sign \( S_x \) and the binary logarithm \( L_x \) of its magnitude. LNS representation equivalent to the 32-bit (single precision) IEEE standard FP format [2] has a 31-bit logarithm part that forms a 2’s-complement fixed-point value ranging from \(-128\) to approximately \(+128\). The real numbers represented are signed and have magnitudes ranging from \(2^{-128}\) to \(~2^{+128}\). Exponents range from \(-126\) to \(+127\) because exponents of \(-127\) (all 0s) and \(+128\) (all 1s) are reserved for special numbers.

**Figure 2:** Binary Storage Format for LNS [5]

LNS multiplication and division are defined as:

**Multiplication:**
\[
S_p = S_x + S_y, \quad L_p = L_x + L_y
\]

**Division:**
\[
S_q = S_x + S_y, \quad L_q = L_x - L_y
\]

Given \( x \) and \( y \), with \(|x| \geq |y|\), \( z = x \pm y \) is computed as:

\[
S_z = S_x, \quad L_z = \log_2 |x \pm y|
\]

\[
= \log_2 |x(1 \pm y / x)|
\]

\[
= \log_2 |x| + \log_2 |1 \pm y / x|
\]

\[
= \log_2 |x| + \log_2 |1 \pm 2(L_y - L_x)|
\]

### 2.3 Hybrid FLP/LNS Arithmetic Unit

Based on [9], FLP has large dynamic range and high precision, but it is slow and complex. The complexity in FLP will cause the temporal inefficiency. The performance of FLP in hardware implementation is only the half of the software implementation.

According to [2], LNS has several advantages over FLP. One of them is the simplicity in multiplication, division, square, square root and others. The second advantage is that LNS has uniform geometric error characteristics over the entire range of values. One downside of LNS is that large LUT is needed in performing addition. Therefore, a large size of ROM is needed to contain such a large LUT. To overcome the complex hardware problem weakness, hybrid FLP/LNS processor is proposed. Hybrid Number System (HNS) is the data computation involving combination of FLP and LNS. In this design, Anti-logarithmic Programmable Logic Array (PLA) is used as a substitute to multiplier and a 64k bit ROM is needed in the 32-bit FLP to LNS conversion. In HNS processor, all input and output data are in IEEE 754 Floating Point format.

Based on [10], the accuracy of the hybrid LNS processor proposed depends on the LUT size used in conversion. The larger the LUT size, the more memory address space it will consume. LNS processor can be compared at the cost of converting numbers to and from the logarithmic domain.
This paper proposed two types of architecture of processor to compute Discrete Cosine Transform. The first is pure LNS processor which all the numerical calculations are performed in logarithmic domain. The second is hybrid LNS processor, in which it performs addition and subtraction in linear, multiplication and division in logarithmic domain. Hybrid LNS does not need a LUT to convert between linear and logarithmic domains. The hybrid LNS architecture is smaller than pure LNS architecture. Figure 3 and 4 depicts the architecture proposed in [10].

![Log2Lin converter](image1)

**Figure 3:** The Log2Lin converter proposed in [10]

![Lin2Log converter](image2)

**Figure 4:** The Lin2Log converter proposed in [10]

3. **Design Methodology**

All processors in our world is just about processing numbers. ALU is the core part of it that carry out the arithmetic operation. Whenever instructed by the processor, the ALU performs an operation on one or more values. These values, called operands, are typically obtained from two registers, or from one register and a memory location. The result of the operation is then placed back into a given destination register or memory location. The status outputs indicate any special attributes about the operation, such as whether the result was zero, negative, or if an overflow or carry occurred. Each architecture has its own unique ALU features, and this can vary greatly from one processor to another. The ALU architecture proposed in this study is illustrated in Figure 5.
From Figure 5, the data input is 32-bit FP format data input. If the desired operation is addition or subtraction, it will take the left hand side path to compute data. On the other hand, if the desired operation is multiplication or division, the system will take the right hand side path to compute data and get the result data output.

In the proposed architecture, addition and subtraction are performed in FLP. The FP format data is first converted to FLP representation in IEEE 754 format, before it will perform addition and subtraction. After that the result is converted back to FP number. In single precision FLP, it has totally 32 bit. Sign consists of 1 bit, exponent has 8 bits and mantissa has 23 bit [11].

\[ F = (-1)^s \times (2^{e-127}) \times (1.m) \]

On the other hand, multiplication and division are performed in logarithmic form. The method of conversion used is iterative method proposed in [12]. After being converted to logarithmic form, multiplication and division is performed by using FP addition and subtraction in logarithmic form [5]. Lastly, the result is converted back to linear FP format data. The logarithmic to linear conversion technique is referred in [13-14]. The calculating algorithm examples are presented in the following part.

### 3.1 Floating Point Conversion

Step 1: Covert the decimal base number to binary.

\[
263.3d = 100000111.0100110011... \_b \\
= 1.000001110100110011... \_b \times 2^8
\]

Step 2: Convert this binary form to IEEE 754 Floating Point format.
Exponent, $e = 127_d + 8_d$
$= 135_d$
$= 1000\ 0111_b$

| Sign | Exponent | Mantissa |
|------|----------|----------|
| 0    | 1000\ 0111 | 0000\ 0111\ 0100\ 1100\ 1100\ 110 |

### 3.2 Floating Point Addition

$X = 0100\ 0010\ 0000\ 1111\ 0000\ 0000\ 0000\ 0000$
$Y = 0100\ 0001\ 1010\ 0100\ 0000\ 0000\ 0000\ 0000$

**Step 1:** Convert $X$ and $Y$ to exponent format.

For $X$,
- Sign, $s = 0$
- Exponent, $e = 1000\ 0100_b$
- $= 132_d - 127_d$
- $= 5_d$
- $X = 1.0001111b \times 2^5$

For $Y$,
- Sign, $s = 0$
- Exponent, $e = 1000\ 0011_b$
- $= 131_d - 127_d$
- $= 4_d$
- $Y = 1.01001b \times 2^4$

**Step 2:** Align the exponent of $X$ and $Y$, the smaller exponent is adjusted.

$Y = 1.01001b \times 2^4$
$= 0.101001b \times 2^5$

**Step 3:** Perform Addition

$X + Y = (1.0001111b + 0.101001b) \times 2^5$
$= 1.1100001 \times 2^5$

Exponent, $e = 5_d + 127_d = 132_d$
$= 1000\ 0100_b$

| Sign | Exponent | Mantissa |
|------|----------|----------|
| 0    | 1000\ 1000 | 1100\ 0010\ 0000\ 0000\ 0000\ 0000 |

### 3.3 Linear to Logarithmic Conversion

$P = 184_d = 1011\ 1000_b$
Convert $P$ into logarithmic form, up to 1 fractional bit.
Since it is 8 bits, $\log_2 (8-1) = \log_2 (7) = 2. +$. Hence, integer portion, $I_{max} = 2$
Step 1: I = 2
\[ A = 1011\ 1000_b \]
\[ A \geq 2^2 \cdot 2^2 = 0001 \ 0000_b \] Yes, therefore \( C_2 = 1 \)
\[ A = A \cdot 2^-(2^2) = 1011 \ 1000_b \cdot 0.0001 = 1011.1_b \]
Computed partial logarithm is now 1xx.x_b

Step 2: I = 1
\[ A = 1011.1_b \]
\[ A \geq 2^{2^1} = 100_b \] Yes, therefore \( C_1 = 1 \)
\[ A = A \cdot 2^{-(2^1)} = 1011.1_b \cdot 0.01 = 10.111_b \]
Computed partial logarithm is now 11x.x_b

Step 3: I = 0
\[ A = 10.111_b \]
\[ A \geq 2^{2^0} = 10_b \] Yes, therefore \( C_0 = 1 \)
\[ A = A \cdot 2^{-(2^0)} = 10.111_b \cdot 1.01110100_b \]
Computed partial logarithm is now 111.x_b

Step 4: I = -1
\[ A = 1.0111_b \]
\[ A \geq 2^{2^1} = 10_b \] Yes, therefore \( C_{-1} = 1 \)
\[ A = A \cdot 2^{-(2^1)} = 1.0111_b \cdot 1.011010100_b \]
Computed logarithm is 111.1_b

Log_2 1011 1000_b = 111.1_b
Log_2 184_d = 7.5_d

3.4 Logarithmic Multiplication and Division

Multiply:
\[ R_1 = XY \text{ where } \log_b |R1| = \log_b |XY| = \log_b |X| + \log_b |Y| \] \[ S_{R1} = S_X \oplus S_Y \]

Divide:
\[ R_2 = X/Y \text{ where } \log_b |R2| = \log_b |X/Y| = \log_b |X| - \log_b |Y| \] \[ S_{R2} = S_X \oplus S_Y \]

3.5 Logarithmic to Linear Conversion

\[ X = 2^\log_2 (X) = (1 - Z)(-1)^{2^{12.05}} \]
Log_2 184_d = 7.5_d
\[ 2^{7.5} = 181_d \approx 184_d \]
Step 1: Antilog(7.5) = 2^7 * 2^{0.5}
\[ = 181_d \approx 184_d \]
4. Results and Discussions

4.1 RTL Schematic

As shown in Figure 6, the hybrid FLP/LNS processor module (HNSP) has 4 input pins and 2 output pins. The inputs consist of clock (clk), 32-bit input A and B, and 2-bit operation selector pins. On the other hand, the outputs consist of a 32-bit multiplication/division output and a 32-bit addition/subtraction output.

In this project, the decimal input and output data is represented in FP format. For addition and subtraction, the 32-bit FP format inputs would be first converted to 32-bit IEEE 754 Floating Point Representation, before it can be performed in FLP format. After completed the computation, the FLP format addition/subtraction result would be converted back to FP format. For multiplication and division, the 32-bit FP format inputs would be first converted to base-2 logarithm, and then only it will perform FP addition or subtraction. Once completed the computation, the addition/subtraction result of FP adder is converted back to Antilog FP format.

As shown in Figure 7, there are nine sub modules inside the HNSP top module: u0 Control Unit, u1 Fixed Point to Floating Point Converter A, u2 Fixed Point to Floating Point Converter B, u3 Floating Point Adder, u4 Floating Point to Fixed Point Converter, u5 Fixed Point Linear to Logarithmic converter A, u6 Fixed Point Linear to Logarithm converter B, u7 Fixed Point Adder and lastly u8 Fixed Point Logarithm to Linear Converter.
4.2 Results Waveform

From Figure 8, the operation input is ‘00’, which indicates the desired operation is addition. The first data input number_inA is positive FP binary value ‘1011.001001’, which is 11.140625 in decimal. The second data input number_inB is positive FP binary value ‘11.001001’, which is 3.140625 in decimal. The computed answer is fixed point binary value positive ‘1110.01001’, which is positive 14.28125 in decimal, shown in output port number_outAS. Another output pin number_outMD stays 0 because the current operation is addition, this port is reserved for the multiplication and division output.

![Figure 8: Result of Addition, 11.140625 + 3.140625 = 14.28125](image)

On the other hand, the operation input ‘01’, ‘10’ and ‘11’ respectively represents subtraction, multiplication and division. The result of these operations are presented in Figure 9, 10 and 11. The results of multiplication and division is less accurate because the lookup table included in antilog is only 3-bit. To get an accurate result, the lookup table size must be at least 15-bit, which requires large ROM size.

![Figure 9: Result of Subtraction, 11.140625 - 3.140625 = 8](image)

![Figure 10: Result of Multiplication, 11.140625 * 3.140625 = 32](image)

![Figure 11: Result of Division, 11.140625 / 3.140625 = 3.25](image)
4.3 Performance Report

The timing, area and power report generated after logical synthesis have been analysed. The slack time is obtained by using data required time to subtract data arrival time, the slack is 0, which meets the minimum requirement (at least zero and positive value ns) of setup hold timing requirement. In this zero slack case, the design is working on an exact frequency and there is no margin available. The design will face setup violation if the slack has a negative value. The speed of the project is 49 MHz. The final area size of the chip is 4.4 mm$^2$ and the total cell area is 3.9 mm$^2$. The total dynamic power is 126.6891 uW whereas the cell leakage power is 180.4094 uW. Dynamic power is the power required when the logic switch states. Cell leakage power is the power required when the transistor is in cutoff state and the diode is in reversed bias.

5. Conclusion

The Hybrid Floating Point/Logarithmic Number System Processor can perform its function well and smoothly in computing addition, subtraction, multiplication and division. This is because FLP addition/subtraction and LNS multiplication/division are separated and their datapath are not mixed. This design is versatile because it can perform conversion between FP-FLP and Linear-Logarithm, depending on the selected operation. In conclusion, the practical design of very large word-length LNS arithmetic processor is possible by using the FLP/LNS approach.

References

[1] M. Chugh and B. Parhami, “Logarithmic arithmetic as an alternative to floating-point: A review,” Conf. Rec. - Asilomar Conf. Signals, Syst. Comput., pp. 1139–1143, 2013.
[2] F. Shi Lai, “The Efficient Implementation and Analysis of a Hybrid Number System Processor,” IEEE Trans. Circuits Syst. II Analog Digit. Signal Process., vol. 40, no. 6, pp. 382–392, 1993.
[3] F. S. Lai and C. F. Eric Wu, “A Hybrid Number System Processor with Geometric and Complex Arithmetic Capabilities,” IEEE Trans. Comput., vol. 40, no. 8, pp. 952–962, 1991.
[4] S. Palekar and N. Narkhede, “High Speed and Area Efficient Single Precision Floating Point Arithmetic Unit,” pp. 1950–1954, 2016.
[5] R. C. Ismail, M. K. Zakaria, and S. A. Z. Murad, “Hybrid logarithmic number system arithmetic unit: A review,” ICCAS 2013 - 2013 IEEE Int. Conf. Circuits Syst. "Advanced Circuits Syst. Sustain., pp. 55–58, 2013.
[6] E. E. Swartzlander and a. G. Alexopoulos, “The Sign / Logarithm Number System,” vol. C, no. December, pp. 1238–1242, 1975.
[7] M. Lastras and B. Parhami, “A logarithmic approach to energy-efficient GPU arithmetic for mobile devices,” Conf. Rec. - Asilomar Conf. Signals, Syst. Comput., pp. 2177–2180, 2013.
[8] D. M. Ellaithy, M. A. El-Moursy, G. H. Ibrahim, A. Zaki, and A. Zekry, “Double logarithmic arithmetic technique for GPU,” Proc. ICCES 2017 12th Int. Conf. Comput. Eng. Syst., vol. 2018–Janua, pp. 373–376, 2018.
[9] F. Taylor, “A hybrid floating-point logarithmic number system processor,” IEEE Trans. Circuits Syst., vol. 32, no. 1, pp. 92–95, 2004.
[10] P. Lee, “An evaluation of a hybrid-logarithmic number system DCT/IDCT algorithm,” Proc.
IEEE Int. Symp. Circuits Syst., no. 3, pp. 4863–4866, 2005.

[11] D. R. Prakash Rao, N. Dhanunjaya Rao, D. K. Naveen, and P. Ramya, “Implementation of the standard floating point MAC using IEEE 754 floating point adder,” Proc. 2nd Int. Conf. Comput. Methodol. Commun. ICCMC 2018, no. Iccmc, pp. 717–722, 2018.

[12] D. K. Kostopoulos, “An Algorithm for the Computation of Binary Logarithms,” IEEE Trans. Comput., vol. 40, no. 11, pp. 1267–1270, 1991.

[13] N. R. Rani, V. Subbiah, and L. Sivakumar, “Design of Logarithm Based Floating Point Multiplication,” vol. 11, no. 2, pp. 1352–1358, 2016.

[14] M. Chaudhary and P. Lee, “An improved two-step binary logarithmic converter for FPGAs,” IEEE Trans. Circuits Syst. II Express Briefs, vol. 62, no. 5, pp. 476–480, 2015.