Design of a Versatile and Cost-Effective Hybrid Floating-Point/LNS Arithmetic Processor

Chichyang Chen

Department of Information Engineering and Computer Science

Feng Chia University, Taichung, Taiwan 407

Tel: +886-5-24517250 ext. 3738 Email: cychen@fcu.edu.tw

Paul Chow

Department of Electrical and Computer Engineering

University of Toronto, Toronto, ON M5S 3G4, Canada

Tel: 416-978-2402 Email:pc@eegc.utoronto.ca

ABSTRACT

LNS (logarithmic number system) arithmetic has the advantages of high-precision and high performance in complex function computation. However, the large hardware problem in LNS addition/subtraction computation has made the large word-length LNS arithmetic implementation impractical. In this research, we proposed a hybrid floating-point (FLP)/LNS processor that can utilize the FLP multiplication-addition-fused (MAF) unit and the FLP division unit for implementing the computation of LNS addition/subtraction. With unified representation format in FLP and LNS numbers, this hybrid processor is versatile because it can execute the FLP-to-LNS and LNS-to-FLP conversions easily, without any extra hardware cost, in addition to the FLP multiplication-addition/subtraction, FLP division, and LNS addition/subtraction instructions. It is cost-effective because the FLP hardware is shared by the LNS unit. A 32-bit hybrid FLP/LNS processor is implemented on the Xilinx Virtex II multimedia FF896 development board. From the synthesis results, the hardware of the 32-bit hybrid processor is at most three times that of a 32-bit pure FLP processor. Our proposed hybrid FLP/LNS approach has made the design of very large word-length LNS arithmetic processors become practical.

Categories and Subject Descriptors

B.2.4 [**High speed arithmetic**]: algorithms and cost/performance.

General Terms: Algorithms and Design.

Keywords:

Logarithmic number system (LNS) arithmetic, Floating-point arithmetic, Logarithmic computation, Exponential computation.

1. INTRODUCTION

Floating-point (FLP) arithmetic unit is an essential component in many scientific and engineering systems. For example, in 3D computer graphics and visual simulations, the major execution units are the FLP multiplication-add-fused (FLP MAF) unit and the FLP divide/square root unit. Due to the scaling for overflow prevention in fixed-point (FXP) arithmetic, FLP arithmetic is usually preferred than FXP arithmetic in the

hardware or software design [1].

In radix-2 logarithmic number system (LNS), a number X is represented as a signed-exponent word, x. Its value is $X = S_X 2^x$, where S_X denotes the sign of X. For $A = S_A 2^a$, $B = S_B 2^b$, and $C = S_C 2^c$, arithmetic in LNS is performed in the following manner:

$$\textit{Multiplication} \quad [C = A \times B] \quad c = a + b \quad \textit{and} \;\; S_C = S_A \oplus S_B$$
$$\textit{Division} \quad [C = A \div B] \quad c = a - b \quad \textit{and} \;\; S_C = S_A \oplus S_B$$
$$\textit{Addition} \quad [C = A + B] \quad c = a + \Phi_2(v) \;\; \textit{and} \;\; S_C = S_A$$
$$(|A| \geq |B|) \; \textit{with} \quad v = a - b \quad \textit{and} \;\; \Phi_2(v) = \log_2\left(1 + 2^{-v}\right)$$
$$\textit{Subtraction} \quad [C = A - B] \quad c = a + \Psi_2(v) \;\; \textit{and} \;\; S_C = S_A$$
$$(|A| \geq |B|) \; \textit{with} \quad v = a - b \quad \textit{and} \;\; \Psi_2(v) = \log_2\left(1 - 2^{-v}\right)$$

S_A, S_B, and S_C denote the signs of A, B, and C, respectively. The above equations reveal that multiplication and division in the LNS require only one additive operation. Square and square root operations can also be performed efficiently by simple shifting. Another advantage of the LNS is its better precision performance than that of the FLP system [2]

However, the addition and subtraction in LNS arithmetic require the computation of the functions Φ_2 and Ψ_2, which is usually performed by table-lookup operation. A problem in the development of large word-length LNS arithmetic is the exponential increase of this table size. In order to reduce the hardware cost for computing these two functions, many approaches have been proposed, either to reduce the size of the tables [3], to compute [4-5] or to avoid [6] the computation. However, we can expect that hardware cost in these computational methods will still increase dramatically as the word length increases. Another problem of LNS arithmetic is that high precision in LNS subtraction is very difficult to obtain [6].

In this research, an architecture that can combine the FLP MAF unit, FLP division unit, and LNS unit into one single arithmetic processor, called "**hybrid FLP/LNS processor**", is proposed. The first advantage of this hybrid processor is that the FLP and LNS number representations can be designed in a uniform and compatible manner. Furthermore, the hardware for performing the FLP-to-LNS and LNS-to-FLP conversions is embedded within the hybrid FLP/LNS processor. There is no extra software and hardware effort needed for the two conversions. Secondly, the hybrid processor is a functionally versatile processor. It can perform FLP MAF operation, FLP division, LNS addition/subtraction, and the FLP-to-LNS and LNS-to-FLP conversions. Thirdly, and most importantly, this approach can

allow the FLP MAF unit and the FLP division unit be fully shared by the LNS unit. This advantage can effectively solve the large hardware problem of LNS arithmetic, and result in a cost-effective FLP/LNS arithmetic processor.

In the proposed hybrid processor, the Φ_2 and Ψ_2 functions are computed by the following computations: exponential computation of $2^{-0.v_F}$, right shifting by v_I bits, addition/subtraction by one, and finally the logarithmic computation followed by an addition. We denote v_I and v_F as the integer and fractional parts of v, respectively. Among these computations, the exponential and the logarithmic computations can be used to compute the LNS-to-FLP and FLP-to-LNS conversions, respectively. In the implementation of the exponential computation, a full word-length multiplication is needed, which can be performed in the FLP MAF unit. On the other hand, the shifting that is used to perform the normalization in the FLP MAF unit can also be adopted to perform the right shifting in the computation of the Φ_2 and Ψ_2 functions. Finally, the FLP division in this hybrid processor is implemented by using the division-by-reciprocal algorithm, which can be easily modified to compute the logarithmic function in the Φ_2 and Ψ_2 functions. As a result, the hardware of the FLP unit can be fully shared by the LNS unit. A 32-bit hybrid FLP/LNS processor is implemented on a Xilinx Virtex II multimedia FF896 development board. From the synthesis results, the proposed hybrid processor can save roughly about one quarter the hardware of the processor that has separate FLP and LNS datapaths. Furthermore, the hardware of the 32-bit hybrid processor is at most three times that of a 32-bit pure FLP processor. It is concluded that practical design of very large word-length LNS arithmetic processors is possible by using our proposed hybrid FLP/LNS approach.

In the following, we first describe the format of the FLP and LNS number representation, and the precision requirement of the LNS subtraction, in Section 2 and 3, respectively. The architecture and algorithms for the FLP MAF unit, the exponential unit, the FLP division and logarithmic unit, and the hybrid processor are described in Section 4, 5, 6, and 7, respectively. The synthesis and test results are presented in Section 8. Conclusions are made in Section 9.

2. Format of the FLP and LNS Number Representation

In the IEEE 754 single-precision standard, the format of a 32-bit FLP number X consists of a sign bit S_X, eight-bit exponent E_X, and an unsigned 23-bit fractional mantissa, M_X. With 127-bias and hidden-one, the value of such an FLP number is defined to be $\quad X = (-1)^{S_X}(1.0 + 0.M_X)2^{E_X - 127}$.

For an LNS number x, its representation consists of a sign bit s_x and an unsigned 31-bit exponent $e_x = e_I.e_F$, where e_I is 8-bit and e_F is 23-bit. With the bias of 127 in the integer part of the exponent, the value of an LNS number is equal to $x = (-1)^{S_x}2^{e_I.e_F - 127}$.

3. Precision Requirement in LNS Subtraction

For FLP arithmetic, if we denote the correct value of the arithmetic operation to be $2^{E_{correct}}(1 + m_{correct})$ and assume that the maximum error in the mantissa computation is ε. Then, the maximum relative error of the arithmetic operation is defined as

$$\text{Maximum relative error}_{\text{FLP operation}} =$$
$$|\frac{2^{E_{correct}}(1 + m_{correct}) - 2^{E_{correct}}(1 + m_{correct} + \varepsilon)}{2^{E_{correct}}(1 + m_{correct})}| = \varepsilon.$$

For our designed 32-bit FLP MAF unit, $\varepsilon = 2^{-1}ulp = 2^{-24}$.

In LNS arithmetic, we denote the correct value of the arithmetic operation to be $I_{correct}.F_{correct}$ and assume that the maximum error in the exponent computation is ε_l. Then, the maximum relative error of the arithmetic operation will be

$$\text{Maximum relative error}_{\text{LNS operation}} = $$
$$|\frac{2^{I_{correct}.F_{correct}} - 2^{I_{correct}.F_{correct} + \varepsilon_l}}{2^{I_{correct}.F_{correct}}}| = 2^{\varepsilon_l} - 1 \approx \varepsilon_l \ln 2$$

In order to have a comparable precision performance with the FLP arithmetic, for the LNS addition/subtraction of our hybrid processor, we must ensure that the maximum error in the computation of the $\log_2(1 \pm 2^{-v})$ be less than $2^{-24}(\ln 2)^{-1} \approx 1.443 \cdot 2^{-24}$. In the computation of LNS subtraction, when v is very close to zero, the value of $(1 - 2^{-v})$ is also very close to zero. The maximum error in 2^{-v} computation should be less than $1.443 \cdot 2^{-47}$.

4. Architecture of the FLP MAF Unit

Fig. 1 shows the block diagram of a typical FLP MAF unit, which is modified from the design in [7].

4.1 The first stage

In the first stage, the multiplication and alignment stage, the mantissas m_B and m_C are multiplied. At the same time, the alignment of the two operands of the FLP addition is accomplished by right shifting the mantissa m_A.

To increase the precision of the exponential function computation when the value of $(1 - 2^{-v})$ is less than 2^{-23}, the word length of the result of the Mantissa Adder is increased to 98 bits (28 integer bits and 70 fractional bits). The two 24×24 multipliers in the right-upper corner are used to multiply m_B and $mCp2$, and m_C and $mBp2$, respectively. $mBp2$ and $mCp2$ are the low-order fractional parts of the exponential values of $2^{0.e_{F1}}$ and $e^{0.e_{F2}2^{-8}\ln 2}$, respectively, as will be explained in Section 5.
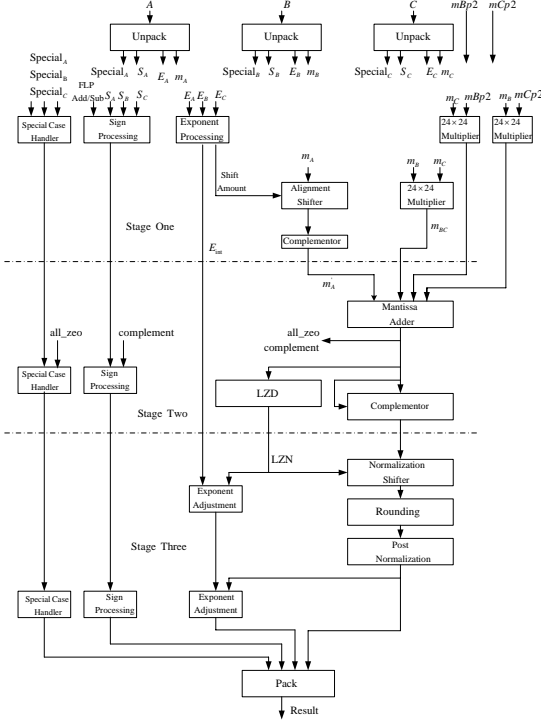
Fig. 1 Architecture of the FLP MAF unit.

4.2 The second and third stages

In the second stage, the sign reversion and addition stage, a 74-bit adder is used to add the aligned and negated mantissa $m_A^{'}$ and the mantissa product m_{BC}. Leading-zero detection (LZD) is performed after this mantissa addition to detect the number of leading zero bits of the result. The value of the intermediate exponent $\exp_{int1}$ is adjusted. In the third stage, the normalization and rounding stage, the normalization, rounding, and post-normalization are performed according to the IEEE 754 standard.

5. Exponential Unit

In this section, we first explain the algorithm for computing the exponential value $2^{0.e_F}$, then we describe the architecture of the exponential unit in the hybrid processor.

5.1 The algorithm

To compute the exponential function $2^{0.e_F}$, with $0.e_F$ being the 23-bit fractional part of an unsigned 32-bit fixed-point number e, we first split the operand $0.e_F$ into two parts, $0.e_F = 0.e_{F1}e_{F2}$, where $0.e_{F1}$ is 8-bit and $0.e_{F2}$ is 15-bit. Then the value of the exponential function $2^{0.e_F}$ will become the product of the two terms, $2^{0.e_{F1}}$ and $2^{0.e_{F2} \cdot 2^{-8}}$. The value of the first term $2^{0.e_{F1}}$ can be easily obtained by looking up a pre-computed table.

To compute the value of the second term $2^{0.e_{F2} \cdot 2^{-8}}$, we approximate $2^{0.e_{F2} \cdot 2^{-8}}$ as

$$2^{0.e_{F2} \cdot 2^{-8} \cdot \ln 2} = e^{0.w_{F2} \cdot 2^{-8}}$$
$$= 1 + 0.w_{F2} 2^{-8} + \frac{1}{2} 0.w_{F2}^2 2^{-16} + \frac{1}{3!} 0.w_{F2}^3 2^{-24} + \frac{1}{4!} 0.w_{F2}^4 2^{-32}$$

$0.w_{F2} = 0.e_{F2} \cdot \ln 2$. In order to minimize the hardware cost, the third-order and the fourth-order terms are computed as

$$\frac{1}{3!} 0.w_{F2}^3 2^{-24} + \frac{1}{4!} 0.w_{F2}^4 2^{-32} \approx \frac{1}{3!} 0.w_{F21}^3 2^{-24} + \frac{1}{4!} 0.w_{F21}^4 2^{-32} + \frac{1}{2} 0.w_{F21}^2 0.w_{F22} 2^{-8} 2^{-24}$$

(1)

$0.w_{F2}$ is divided into two parts as $0.w_{F2} = 0.w_{F21} + 0.w_{F22} \cdot 2^{-8}$. $0.w_{F21}$ and $0.w_{F22}$ are designed to be 8-bit and 15-bit wide, respectively. Thus, the sum of the first two terms in (1) can be obtained by looking up a table with only 256 entries. The term $\frac{1}{2} 0.w_{F21}^2 0.w_{F22} 2^{-8} 2^{-24}$ in (1) can be easily computed by hardware circuit.
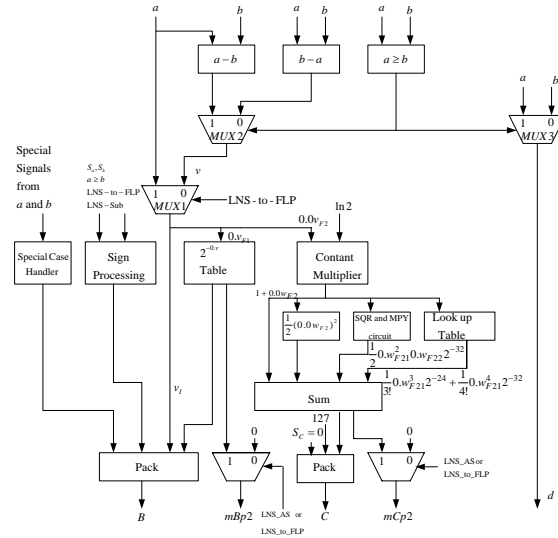

Fig. 2 Architecture of the exponential unit

5.2 The architecture

To perform the LNS-to-FLP conversion, we need to convert an LNS number $x = (-1)^{S_x} 2^{e_I.e_F - 127}$ into an FLP number $X = (-1)^{S_X} (1.0 + 0.M_X) 2^{E_X - 127}$ through the following computations: $S_X = S_x$, $E_X = e_I$, and $1.0 + 0.M_X = 2^{0.e_F}$. If these computations are performed by the FLP MAF unit ($B \times C + A$), the three FLP operands can be assigned in the following manner:

$A = 0.0$, $S_B = S_x$, $E_B = e_I$, $M_B = 2^{e_{F1}}$, and
$S_C = 0$, $E_C = 127$, $M_C = 2^{0.e_{F2} \cdot 2^{-8}}$.

The other instruction that needs to compute exponential function is the LNS addition/subtraction. We need to compute the value of $(1 \pm 2^{-v}) = (1 \pm 2^{-v_I} 2^{-v_F})$. This computation can be performed by the FLP MAF unit by assigning the three operands of the FLP MAF unit as: with $v^{'} = v_I^{'}.v_F^{'} = 127 - v_I.v_F$,

$A = 1.0$

$$S_B = \begin{cases} 0, \text{ if LNS addition,} \\ 1, \text{ if LNS subtraction} \end{cases}, \ E_B = v_I', \ M_B = 2^{v_{F2}'},$$

and $S_C = 0$, $E_C = 127$, $M_C = 2^{0.v_{F2}' \cdot 2^{-8}}$

We can divide each of M_B and M_C into two parts as,

$$M_B = m_B + mBp2 \text{ and } M_C = m_C + mCp2 . \ m_B \text{ and}$$

m_C are the mantissas of the FLP numbers B and C, respectively. $mBp2$ and $mCp2$ are the least significant 24 bits of M_B and M_C, respectively. Then,

$$\begin{aligned} m_{BC} &= (m_B + mBp2)(m_C + mCp2) \\ &\approx m_B m_C + m_B \cdot mCp2 + m_C \cdot mBp2. \end{aligned}$$
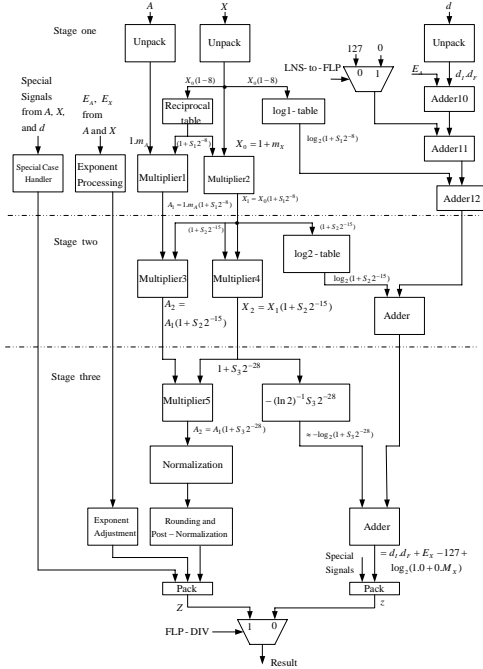

Fig. 3 The architecture of the FLP division and logarithmic unit

6. FLP Division and Logarithmic Unit
6.1 Division algorithm

For a normalized operand, $X = 1 + x$, $0 \leq x = \sum_{i=1}^{23} x_i 2^{-i} < 1$, its reciprocal can be obtained by three processing stages. In the first stage, the approximate value of the reciprocal of X, denoted as a normalization factor $(1 + S_1 2^{-8})$, with S_1 being a negative 8-bit integer, is obtained from a lookup table addressed by the eight leading fractional bits of X, $x_i, \ 1 \leq i \leq 8$. Then, the first remaining term

$$X_1 = 1 - \sum_{i=1}^{28} x_i^{(1)} 2^{-i}, \ x_i^{(1)} = 1, \text{ or } 0, \text{ is computed as}$$

$$X_1 = X(1 + S_1 r^{-1}).$$

We can easily show that the leading fractional bits $x_1^{(1)}$ of X_1, $1 \leq i \leq 7$, should be all zero.

In the second stage, the normalization factor is chosen to be $(1 + S_2 2^{-15}) = (1 + \sum_{i=8}^{15} x_i^{(1)} 2^{-i})$. The second remaining term X_2 becomes

$$X_2 = 1 + \sum_{i=15}^{28} x_i^{(2)} 2^{-i} = X_1(1 + S_2 2^{-15}).$$

The leading fourteen fractional bits of X_2 are all zero, and $| X_2 - 1 | < 2^{-14}$.

In the third stage, the normalization factor is chosen to be $(1 + S_3 2^{-28}) = (1 - \sum_{i=15}^{28} x_i^{(2)} 2^{-i})$, and the value of the reciprocal of X and quotient can be computed as

$$\frac{1}{X} \approx (1 + S_1 2^{-8})(1 + S_2 2^{-15})(1 + S_3 2^{-28}) \quad \text{and}$$

$$\frac{A}{X} = A(1 + S_1 2^{-8})(1 + S_2 2^{-15})(1 + S_3 2^{-28}) \quad (2)$$

6.2 Logarithmic computation

According to (2), the logarithmic value of X can be computed as

$$\begin{aligned} &\log_2(X) \\ &= -[\log_2(1 + S_1 2^{-8}) + \log_2(1 + S_2 2^{15}) + \log_2(1 + S_3 2^{-28})]. \end{aligned} \quad (3)$$

The first and the second terms in (3) can be obtained by looking-up a table with 256 entries. The third term can be computed as $\log_2(1 + S_3 2^{-28}) \approx S_3 2^{-28}(\ln 2)^{-1}$.

6.3 The Architecture

Fig. 3 shows the architecture of this unit. For FLP division operation, the output Q is equal to $Q = \frac{A}{X}$, which is an FLP number. In this case, the d input is ignored and is assigned as FLP zero value. The d value is the larger one among the inputs of a and b.

For FLP-to-LNS operation, the input $X = (-1)^{S_X}(1.0 + 0.M_X)2^{E_X - 127}$ is an FLP number and the output z is an LNS number, whose value should be $z = (-1)^{S_z} 2^{z_I.z_F - 127}$, with

$$S_z = S_X \text{ and } z_I.z_F = E_X + \log_2(1.0 + 0.M_X).$$

In this case, the input A is assigned as a FLP 1.0 value and the input d is assigned as 0.0.

For LNS addition/subtraction, the output z is an LNS number, the value of the LNS input d should be added to the logarithmic value. Therefore, the value of the A input should be assigned as an FLP 1.0 value. With $d = (-1)^{S_d} 2^{d_I.d_F - 127}$, the

value of the output z should be $z = (-1)^{S_z} 2^{z_I.z_F - 127}$, where $S_z = S_d$ and

$$z_I.z_F = d_I.d_F + \log_2(X)$$
$$= d_I.d_F + E_X + \log_2(1.0 + 0.M_X)$$
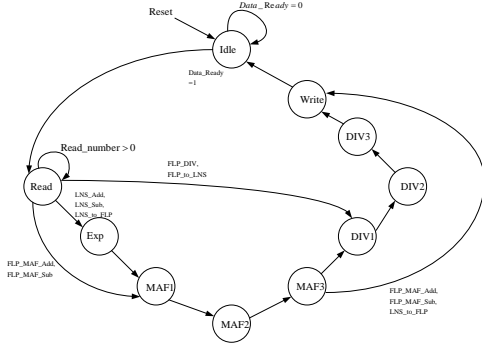
Fig. 4 The FSM of the control unit of the hybrid FLP/LNS processor
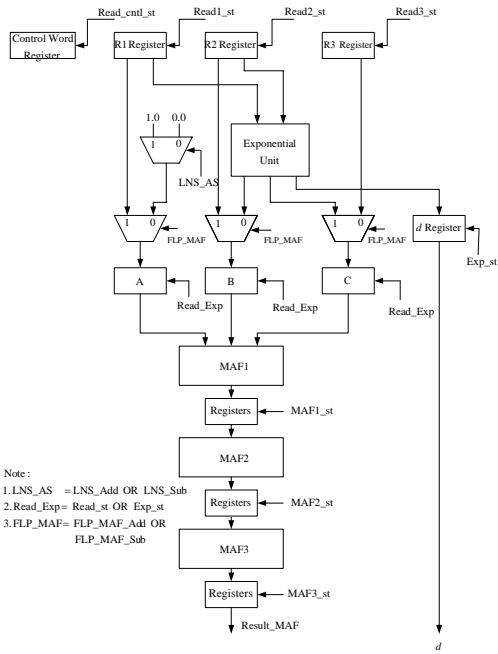
Fig. 5 The block diagram of the data path of the hybrid FLP/LNS processor (part 1)

7. PROCESSOR ARCHITECTURE

In this section, we describe the design of the control and datapath of the hybrid processor.

7.1 Control unit design

The control unit is implemented as a finite state machine (FSM), as shown in Fig. 4. Associated with each state, a control signal is generated and is named as StateName_st. After the system resets, the FSM enters the Idle state and is waiting for the Data_ready signal from the Microblaze (MB) processor. Once the Data-Ready signal is raised, the FSM enters into the Read state. In this state, the first data read from the MB processor through the FSL link is stored in the Control Word Register.

7.2 The data path design

In Fig. 5 (part 1), the Exponential unit generates the values of B and C operands for the FLP MAF unit, and also the d operand for the LNS addition/subtraction instructions. For the FLP-MPY-Add and FLP-MPY-Sub instructions, the inputs of the A, B, and C registers are the values from the R1, R2, and R3 registers, respectively. For LNS-Add, LNS-Sub, and LNS-to-FLP instructions, the input values of the B and C registers come from the outputs of the Exponential unit. The value of the A register should be assigned as 1.0 for the LNS-Add and LNS-Sub instructions, and as 0.0 for the LNS-to-FLP instruction, as explained in Section 6.3.
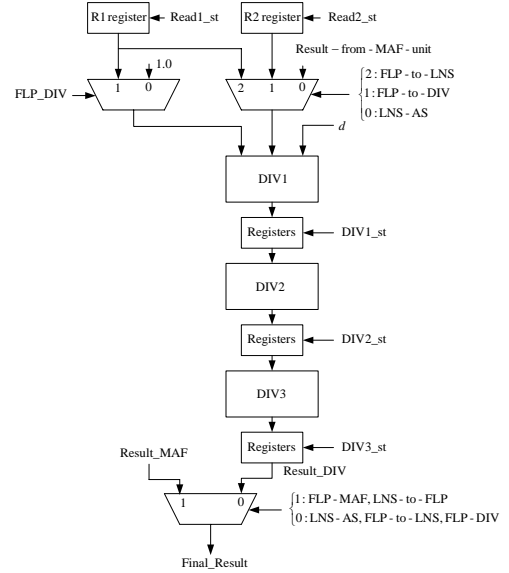

Fig. 5 The block diagram of the data path of the hybrid FLP/LNS processor (part 2)

In Fig. 5 (part 2), the first input operand (the dividend) to the FLP division and logarithmic unit comes from the R1 register. For the other instructions, this operand is assigned value 1.0. The second input operand (the divisor X) comes from the R1 register, if the instruction is FLP-to-LNS conversion. If the instruction is FLP division, the second operand should be the value of register R2. If the instruction is LNS addition/subtraction, the second operand should be from the result of the FLP MAF unit.

8. SYNTHESIS and TEST RESULTS

The hybrid FLP/LNS processor is synthesized by using the Xilinx EDK 6.3.

8.1 Timing and hardware cost

From the controller design of the hybrid FLP/LNS processor, we can easily derive the number of clock cycles that are needed to execute each of the seven kinds of the instructions in the datapath, which is listed in Table 1. The FPGA device of the development board is 2v2000ff896-4. The device utilization summary is listed in Table 2.

In order to investigate the extra hardware cost that is required for adding the FLP-to-LNS, LNS-to-FLP, and LNS-Add and LNS-Sub instructions into the FLP arithmetic processor, we

remove the circuits in the hybrid processor that are used for executing the four LNS-related instructions, and then we estimate the hardware cost after this modification. The device utilization summary of the FLP processor is listed in Table 2.

Table 1. The execution clock cycles of the instructions in the FLP/LNS datapath.

Instruction	LNS-to-FLP	FLP-to-LNS	LNS Add and Sub	FLP_MPY_ Add and Sub	FLP_DIV
No. of cycles	4	3	7	3	3

Table 2. The number of slices used for the FLP/LNS processor.

	Used number	Percentage used
Hybrid FLP/LNS	3415	31
FLP processor	1274	11

8.2 Test results

The hybrid FLP/LNS processor is tested thoroughly in the manner described in the following. The test is divided into three phases. The first phase is to test all the possible cases of the special values of the operands. The second phase is to test the cases with all the possible values of the integer part of the LNS numbers or the exponent of the FLP numbers, by fixing the values of the fractional parts. The third phase is to test as many cases as possible with different values of the fractional parts or the mantissas of the operands, while the values of their integer parts or exponents are fixed. The number of the test cases for each of the seven instructions is more than 3×2^{23}. The maximum errors of these cases for all the instruction are all less than 2^{-23}.

8.3 Discussions

From Table 1, we can see that the datapath delay of the LNS add/sub instruction is about 2.3 times that of the FLP multiply-and-add/sub instruction. However, the LNS multiply/divide instructions take only one clock cycle. From our experiments, the maximum clock rate of the FLP/LNS processor is less than half the clock rate of the FLP processor. One method to improve the clock rate of the FLP/LNS processor is to divide the exponential stage in the FLP/LNS processor into two stages and further optimize its circuit design.

Comparing the hardware cost of the FLP/LNS processor and the pure FLP processor in Table 2, we can see that the hardware cost of the FLP/LNS processor is roughly about 2.5 to 3 times that of the FLP processor. The extra 1.5 to 2 times the hardware cost of the FLP processor is mainly due to the high-precision cost in the LNS subtraction. Our proposed hybrid processor can save at least about one quarter

$$\left(\frac{\text{Cost of FLP processor(1)}}{\text{Cost of FLP processor(1)} + \text{Cost of LNS/FLP processor(3)}} \right)$$ the hardware of the processor with separate FLP and LNS datapaths.

9. CONCLUSIONS

This research has proposed a versatile and cost-effective hybrid FLP/LNS arithmetic processor. It is versatile because it can execute the FLP-to-LNS and LNS-to-FLP conversions, the FLP multiplication-addition/subtraction and FLP division operations, and the LNS addition/subtraction operations in one single datapath with uniform data representation format. It is cost-effective because it allows the FLP hardware be shared by the LNS computation. This 32-bit hybrid FLP/LNS processor is implemented on the Xilinx Virtex II multimedia FF896 development board. From the synthesis results, we found that the hardware of the 32-bit hybrid processor is at most three times that of a 32-bit pure FLP processor. It is concluded that practical design of very large word-length LNS arithmetic processors is possible by using our proposed hybrid FLP/LNS approach.

10. Acknowledgement

11. REFERENCES

[1] Ki-Il Kum, Jiyang Kang, and Wonyong Sung, "AutoScaler for C: an optimizing floating-point to integer C program converter for fixed-point digital signal processing," IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing, vol. 47, no. 9, Sept. 2000, pp. 840-848.

[2] E. E. Swartzlander Jr., et al., "Sign/logarithm arithmetic for FFT implementation," *IEEE Tran. on Computers,* vol. 32, no. 6, pp. 526-534, June 1983.

[3] M. L. Frey and F. J. Taylor, "A table reduction technique for logarithmically architected digital filters," *IEEE Transactions on Acoustics, Speech, and Signal Processing,* vol. ASSP-33, no. 3, pp. 718-719, June 1985.

[4] J. N. Coleman, E. I. Chester, C. I. Softley, and J. Kadlec, "Arithmetic on the European logarithmic microprocessor," *IEEE Transactions on Computers*, vol. 49, no. 7, pp. 702-715, July 2000.

[5] Chichyang Chen, Rui-Lin Chen, and Chih-Huan Yang, "Pipelined computation of very large word-length LNS addition/subtraction with polynomial hardware cost," *IEEE Transactions on Computers*, vol. 49, no. 7, pp. 716-726, July 2000.

[6] Mark G. Arnold, Thomas A. Bailey, John R. Cowles, and Jerry J. Cupal, "Redundant logarithmic arithmetic," *IEEE Tran. on Computers,* vol. 39, pp. 1077-1086, Aug. 1990.

[7] E. Hokenek, R. Montoye, and P. W. Cook, "Second-generation RISC floating-point with multiply-add fused," *IEEE J. Solid-State Circuits,* vol. 25, no. 5, pp. 1207-1213, 1990.

[8] Chichyang Chen, Rui-Lin Chen, and Ming-Hwa Sheu, "A hardware algorithm for fast logarithmic computation with exponential convergence rate," *Journal of the Chinese Institute of Engineers*, vol. 28, no. 4, pp. 749-752, July, 2005.