Efficient Multiple-Precision Posit Multiplier

Hao Zhang and Seok-Bum Ko
Department of Electrical and Computer Engineering, University of Saskatchewan, Canada
Email: {hao.zhang, seokbum.ko}@usask.ca

Abstract—Posit number system has been recently widely applied in many fields of applications. For different applications, the precision requirements are usually different. In addition, the transprecision computing paradigm, which is proposed for energy efficient computation, even requires different precision in each computation step. To support computations of various precision in a single hardware architecture, in this paper, a unified architecture of multiple-precision posit multiplier is proposed. The proposed posit multiplier supports the commonly used Posit(8, 0), Posit(16, 1), and Posit(32, 2) formats, where one Posit(32, 2), or two parallel Posit(16, 1), or four parallel Posit(8, 0) multiplications can be accomplished each time. Each module of the proposed posit multiplier is carefully tailored for resource sharing among three supported precision formats. Compared to the Posit(32, 2) multiplier, the proposed multiple-precision multiplier adds the support for parallel low-precision posit multiplications with only 12.8% more area and 15.4% more power. The proposed architecture can be used in posit-enabled general-purpose processor designs.

I. INTRODUCTION

Posit number system [1] has recently been widely used in many fields of applications to take the place of conventional IEEE floating-point numbers [2]. It uses a regime field to further scale the exponent and thus can provide much larger dynamic range than the floating-point formats. This can benefit many applications, such as deep learning training [3], where a large dynamic range is expected from the numeric format. In addition, posit number format uses tapered precision, and thus its number distribution is non-uniform. This distribution fits well with the data distribution of many applications which makes posit formats efficient in meeting both dynamic range and precision requirements. Due to these advantages, many applications, such as deep learning [4] and scientific computation [5], nowadays use posit formats.

In order to facilitate the use of posit formats in applications, in recent years, extensive research works have been focused on the design of posit arithmetic units [6]–[11]. In [6], architecture generator for posit adder and subtractor is proposed. In addition to adder, in [7], the design of posit based multiplier is discussed. The architecture of posit divider is proposed in [8] along with improved adder and multiplier designs. Power optimization techniques for posit multiplier considering the variable mantissa bit-width is proposed in [9]. In addition to these basic arithmetic units, the fused posit arithmetic units, including the multiple-accumulate unit [10] and the positquire operation unit [11], are also available in the literature.

The above mentioned generators can be used to generate arithmetic units for specific applications. However, only one precision format is supported by each of these arithmetic units. In order to support multiple applications in a general-purpose processor, multiple arithmetic units that support different precisions are required. In such a configuration, when running one application, only one arithmetic unit is working while the others are not used. This will lead to area and power waste. Therefore, a multiple-precision arithmetic unit [12] in which different precision modes share the resources is required. As posit arithmetic becomes popular in applications, many general-purpose processors enable the support of posit arithmetic operations [13]. Therefore, a multiple-precision posit arithmetic unit is desired. On the other hand, the transprecision computing paradigm [14], which is proposed for energy efficient computation and is widely applied in edge computing, requires multiple computation precisions in a single application. Therefore, a multiple-precision posit arithmetic unit is also needed in this scenario.

In this paper, an efficient multiple-precision posit multiplier is proposed as a complementary design to the posit arithmetic literature. The proposed posit multiplier supports the commonly used Posit(8, 0), Posit(16, 1), and Posit(32, 2) formats, where one Posit(32, 2), or two parallel Posit(16, 1), or four parallel Posit(8, 0) multiplications can be accomplished each time. All three precision modes share the same hardware resources. Each module of the proposed posit multiplier is carefully tailored for resource sharing among three supported precision formats. Implementation results show that the proposed posit multiplier supports three commonly used precision formats with only 12.8% area overhead and 15.4% power overhead compared to a 32-bit posit multiplier. The proposed architecture can be used in posit-enabled processor designs and the proposed techniques can be modified and applied to support other precision modes.

The rest of the paper is organized as follows: Section II presents the background of posit format and posit multiplier architecture. The proposed multiple-precision posit multiplier architecture is presented in Section III. In Section IV, the implementation results and their analysis are presented. Finally, Section V concludes the whole paper.

II. BACKGROUND

The general format of a posit number is shown in Fig. 1. A posit format is defined with nb and es, where nb represents the total bit-width and es represents the exponent bit-width. As shown in Fig. 1, the regime, exponent, and fraction (or mantissa) share the bit positions. The regime part always exists which is a binary sequence of zeros ended by a single one bit or a sequence of ones ended by a single zero bit. If the regime
The proposed multiple-precision posit multiplier is designed based on the architecture shown in Fig. 2. It accepts one set of 32-bit input operands, or two sets of 16-bit operands, or four sets of 8-bit operands. The precision mode is controlled by the MODE signal. MODE is a 2-bit signal. In 8-bit precision (PM), MODE=00, and similarly, in 16-bit (PH) and 32-bit (PS) precision, MODE=01 and 10, respectively. The main components shown in Fig. 2 work in a specific precision mode under the control of MODE signal.

A. Posit Component Extraction

The general architecture of a posit component extraction module is shown in Fig. 3. To accommodate multiple-precision operations, some changes are required for the design. The architecture of the modified complementer (COMP) is shown in Fig. 4, where the subscript represents the bit position and \( m \) represents the MODE signal. In 32-bit mode, the whole operand is XORed with the most significant bit (MSB) of the operand. When doing incrementing operation, the carry output from former incremenetor is passed to the next incremenetor. For 16-bit mode, the lower half operand is XORed with operand bit 15 and carry propagation of incrementer is interrupted after the second incremenetor. Similarly, in 8-bit mode, the four parts work independently with each other.

The architecture of the leading zero detector (LZD) is shown in Fig. 5. The complemented operand is first rearranged according to the precision mode and it is shown in the upper part of Fig. 5. For lower precision modes, the sign bits are removed and zero bits are added in those position for leading zero detection. Then, four LZDs are applied for counting leading zeros in PM mode. LZD-1 and LZD-2 are then combined for counting leading zeros in PH mode for operand set 1. And similarly, LZD-3 and LZD-4 are combined for operand set 2 in PH mode. Finally, those two results are
Further combined for leading zero count in PS mode. The leading one detector (LOD) uses similar architecture except that when rearrange operands, one bits are added in those sign bit positions instead of zero bits.

The left shifter also needs changes to accommodate multiple-precision operations. The modified architecture is shown in Fig. 6. The arrangement of data in different precision modes are shown in the upper part. Four small shifters are applied for PM mode. For PS and PH modes, the data shifted out from one shifter may need to be sent to the next shifter as the shift_in data.

B. Unified Mantissa Multiplier

After the exponent extraction module, the mantissa to be multiplied are obtained. As the minimum regime width is 2-bit, the mantissa including the implicit bit is \((n_\text{b} - e_\text{s} - 2)\)-bit. The rearranged mantissa for multiple-precision multiplication is shown in Fig. 7. The mantissa is then divided into four 7-bit parts and a total of 16 7-bit small Booth multipliers [15] are applied.

The products of these small multipliers are then selected and accumulated for different precision modes, as shown in Fig. 8. In Fig. 8, the two digits represent the part number of the operands, for example, PD13 represents the product of M1 of the multiplicand and M3 of the multiplier. In each mode, only the grey parts are selected and accumulated. The bit positions to extract product are also shown in Fig. 8.

C. Addition and Normalization

The mantissa multiplier generates carry-save format products which will be then added by a carry propagate adder.

The multiple-precision carry propagate adder is similar to the incrementer shown in Fig. 4. The MODE signal is used to control whether carry is propagated through two small adders. Then, the normalization is similar to the shifter shown in Fig. 6, except that a right shift is used for normalization.
D. Posit Component Packing and Rounding

The general architecture of a posit component packing and rounding module is shown in Fig. 9. To accommodate multiple-precision operation, modules, including COMP, right shifter, and rounding, need to be changed. Here the COMP and right shifter are similar to the ones shown in Fig. 4 and Fig. 6, respectively, except a right shifter is used here. For rounding operation, four rounding units are implemented for PM mode. In PH mode, the first and third units are used and in PS mode, only the first unit is used.

Finally, the results are packed into a 32-bit vector which contain one 32-bit result or two 16-bit results or four 8-bit results.

IV. RESULTS AND ANALYSIS

The proposed architecture is implemented with Verilog HDL. Simulations with extensive testing vectors for each operational mode are performed to verify the proposed design. The verified design is then synthesized in Synopsys Design Compiler using STM-90nm technology with normal case parameters. Delay and area metrics are generated from synthesis process. Power estimation is then performed with the synthesized netlist in Synopsys PrimeTime PX.

The delay, area, power, and energy consumption of the proposed design are compared with those standalone designs and the results are shown in Table I. Compared to Posit(32, 2) multiplier design, the proposed multiplier has a 6% delay overhead. It comes from the extra multiplexers used for managing different operation modes. In terms of area, the proposed design consumes 12.8% more resource. This overhead also comes from the extra multiplexers used in the proposed design. However, with this small overhead, the proposed multiplier can support parallel low precision posit multiplications in addition to the 32-bit operation. Compared to the combined multiple-precision posit multiplier and Posit(32, 2) multiplier, the proposed design has a much smaller area. In terms of power consumption, the proposed design has a 15.4% overhead compared to the Posit(32, 2) multiplier. This is due to the extra area consumed by the proposed design. Energy per operation is also compared among the proposed design and those standalone designs. Due to the sharing of resources, the energy per operation is increased for the proposed design. However, with the proposed architecture, multiple precision modes are supported in a single architecture.

Compared to Posit(8, 0) and Posit(16, 1) multipliers, the proposed design has a much larger delay. However, as the proposed design supports parallel low precision operations, the throughput of low-precision operations can be compensated.

The synthesis results of each module in the proposed multiple-precision posit multiplier and Posit(32, 2) multiplier are shown in Table II. The input process contains the posit component extraction and the sign and exponent processing. The output process contains posit component packing and rounding. In terms of area, the overhead of the adder is relatively small because the resource sharing of the adder part is relatively simple. For the mantissa multiplier, the area overhead comes from the selection of partial product and the extraction of carry-save format product. For input and output process, about 15% area overhead is obtained due to the complexity of these two modules.

V. CONCLUSION

In this paper, an efficient multiple-precision posit multiplier architecture is proposed. The proposed architecture supports the most commonly used Posit(8, 0), Posit(16, 1), and Posit(32, 2) formats in a unified architecture. Each component of the posit multiplier is carefully designed and tailored for resource sharing. Compared to the Posit(32, 2) multiplier, the proposed multiplier has minor area and power overhead. The proposed posit multiplier can be used in posit-enabled processors or in transprecision computing applications.
REFERENCES


