A Reconfigurable Multiple-Precision Floating-Point Dot Product Unit for High-Performance Computing

Wei Mao[1], Kai Li[1], Xinang Xie[1], Shirui Zhao[1], He Li[2] and Hao Yu[1]
[1] School of Microelectronics, Southern University of Science and Technology, Shenzhen, China
[2] Department of computer science and technology, University of Cambridge, London, UK

***Abstract*—There is an emerging need to optimize floating-point (FP) dot product units (DPU) for high-performance scientific computing as well as training deep learning models. Due to different precision requirements of applications, a reconfigurable multiple-precision DPU operation can largely reduce the cost of area and power. However, the existing methods could result in redundant bits for unit multipliers, but also leave idle hardware resources for the operations in different precisions. In this paper, a reconfigurable multiple-precision FP DPU design is proposed for high-performance computing (HPC) applications. The FP DPU can be reconfigured as follows. A bit-partitioning method is provided to minimize the redundant bits with a configurable mixed-precision multiplier for three-mode operations: 20 half-precision Dot Product (DP), 5 single-precision DP, and 1 double-precision DP operations. Any of the modes can be executed in two successive clock cycles without idle hardware resources. The proposed design is realized by using the UMC 55-nm process with simulation results. Compared with the existing multiple-precision FP methods, the proposed DPU achieves 88.9% and 35.8% area-saving performance for FP16 and FP32 operations, respectively. Moreover, when using benchmarked HPC applications where multiple precisions can be used, the proposed reconfigurable DPU can accelerate up to 4× and 20× maximum throughput rates when compared with fixed FP32 and FP64 operations, respectively.**

Keywords—Multiple-precision, partitioning, floating point, SIMD, dot product unit, mix-precision, multiplier.

I. INTRODUCTION

The IEEE 754-2008 standard introduces multiple precisions of floating-point (FP) numbers, such as 16-bit half precision (FP16), 32-bit single precision (FP32) and 64-bit double precision (FP64) [1]. In general, the higher the precision, the more costly and slower computation will be. While high precision has the advantage of accuracy, low precision has the advantage of improved power and speed [2]. Therefore, FP numbers of various precisions are often used in many intensive computing applications to improve the performance, especially accelerating scientific computations with multiple-precision algorithms. Besides, the accurate description of model in deep learning research often requires the collaborative completion of multiple-precision FP data and algorithms [3]. The rapid development of hybrid precision methods for training acceleration also requires the support for multiple-precision FP formats [4]. In addition, performing simulations and other numerical computations in chemistry, physics and other fields often requires combining single and double precision arithmetic. In order to meet the computational requirements in such complex scenarios, designing a reconfigurable dot product unit (DPU) for multiple-precision operations is very important [5].

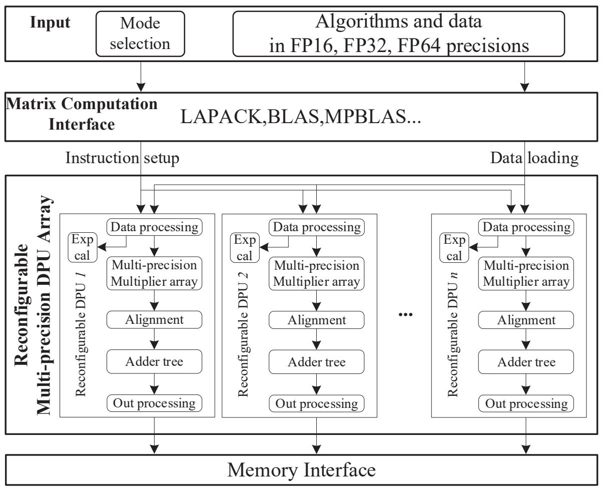

Fig. 1. The computational flow of the reconfigurable HPC system.

There are many works that present designs with multiple-precision multiplication operations. In [6-7], multiple precision operations are implemented by dual-mode multiplication units. Those designs can support one FP64 or two FP32 operations in parallel. In [8], LU factorization and iterative refinement solvers in high precision are sped up by using FP16 and mixed-precision algorithms. In [9], a single instruction multiple data (SIMD) architecture is designed for multiple-precision FP vector products. But the multiple-precision multiplication is achieved by truncating the high-bit mantissa only. In [10], a FP architecture supporting multi-precision operations in the range FP16~FP128 is proposed. However, the architecture is based on a 15-bit multiplier, the bit selection of which is optimized to support 128-bit quadruple-precision multiplication. This setup results in large bit redundancy when doing FP arithmetic in other precisions.

This paper proposes a reconfigurable multiple-precision FP DPU unit. Fig. 1 shows the computational flow of the reconfigurable HPC system. The algorithms and data in multiple precisions are passed through the matrix computation interface to the DPU array. The DPU array, which is constructed by the proposed DPU units, can support multiple-precision calculations according to the mode selection. The DPU can be reconfigured to perform the FP32 and FP64

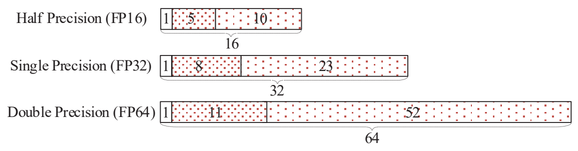

Fig. 2. The IEEE formats of FP numbers in FP16, FP32 and FP64.

computations. To construct the hardware for high-precision operations, the FP32 and FP64 mantissas are split according to the bit width of the unit multiplier. Thanks to the reconfiguration structure, twenty FP16 operations, five FP32 operations and one FP64 operation can be executed in two successive clock cycles. The main contributions of this paper are summarized as follows.

- A reconfigurable FP DPU unit based on a novel bit-partitioning method with minimized redundant bits is proposed to support FP16, FP32 and FP64 precisions.
- For each precision, the unit multipliers of the reconfigurable system are all employed for operations without standby multipliers being gated.
- The fused multiplier can support three mixed-precision multiplication operations. In addition, the operations can be executed in parallel.

The remainder of this paper is organized as follows. Section II introduces the existing multiple-precision DPU designs and the proposed reconfigurable architecture. Section III analyses the proposed bit-partitioning method. Section IV covers the circuit implementation and design considerations. Section V shows the performance comparison based on the experimental results, followed by the conclusion in Section VI.

II. Reconfigurable Floating-Point Computing

In the IEEE 754 standard, FP numbers contains three parts, i.e., sign (S), exponent (E) and mantissa (M). Fig. 2 illustrates the bit widths of FP16, FP32 and F64. The value encoded in a floating point format can be obtained by

$$Value = (-1)^S \times (1.M) \times 2^{E-bias} \tag{1}$$

The significant digit number at each precision equals the corresponding mantissa bit width plus one. For FP16, FP32 and F64, the significant digit numbers are 11, 24 and 53, the bias are 15, 127 and 1023, respectively. To build the reconfigurable multiple-precision DPU, the high-precision significant bits are partitioned into several subparts, whose bit widths should be no less than the significant digit number of a low-precision FP16 value. Thus, the hardware for high-precision subparts can be used for the low-precision operations by reconfiguration.

Currently, many FP multiplication units have been proposed to support multiple precisions in one architecture [6-7,9-10]. By using a reconfigurable system instead of duplication, the hardware costs including area and power are largely reduced. In [6-7], the multiple-precision multiplier for FP32 and FP64 is implemented by using the subword-parallel strategy. Based on the completed FP64 multiplier with 53-bit operands, both the input and partial production generation part are partitioned into two subparts with redundant bits filled with isolation bits zero. Thus, the multiplier can implement two FP32 multiplication or one FP64 multiplication once. However, one FP64 multiplier can only reconfigured to support two FP32 multiplication, the hardware utilization rate is only 50% in FP32 operation. In [10], the sum-apart strategy is adopted for multiple-precision DPU design. The multiplication array based on the 15-bit width unit multipliers is constructed at first, then the multiplication result is obtained by shifting and accumulating the results of the unit multipliers according to the multiple-precision partitioning method. The design can support the DP operations of four precisions FP16~FP128. However, the 15-bit width selection is optimized for FP128 only. When the 15-bit unit multiplier is used for FP16, FP32 and FP64 operations, the redundant bits are large. In addition, by using the sum-apart strategy, the hardware utilization rate is decreasing along with lowering the precisions due to the gated idle unit multipliers [11].

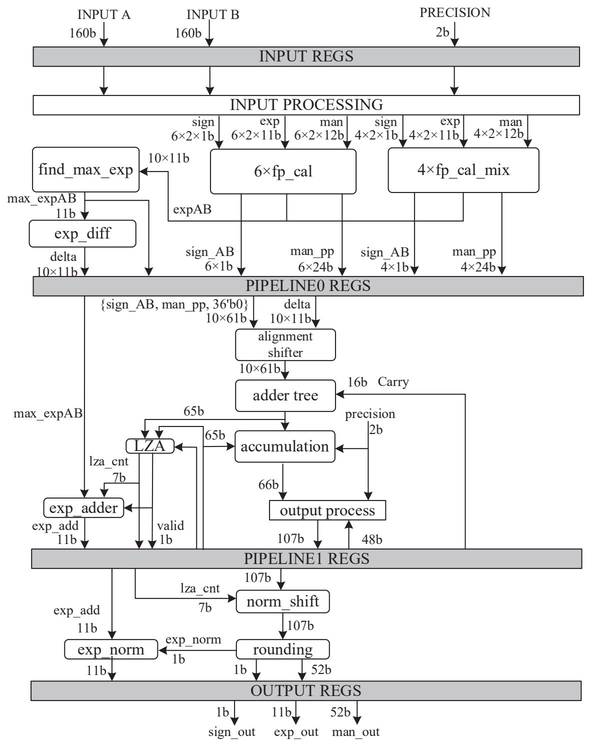

Fig. 3. Architecture of 10-input PEs based on the reconfigurable adder tree.

The proposed reconfigurable multiple-precision DPU is designed to support FP16, FP32 and FP64 precisions by using SIMD architecture. We use the bottom-up combination method to construct the whole structure. Firstly, the unit multiplier is designed to implement the FP16 operation. Then, the unit multipliers are shifted and accumulated by using alignment shifter and reconfigurable adder tree to generate the high-precision DP results according to the bit-partitioning method. In Fig. 3, the multiple-precision DPU with ten multipliers is proposed. 10 multipliers are composed of 6 conventional multipliers fp_cal and 4 mixed-precision fused multipliers fp_cal_mix. The fp_cal and fp_cal_mix modules not only calculate the products of the mantissa, but also generate the

TABLE I. THE ANALYSIS OF RECONFIGURABLE HPC SYSTEM PERFORMANCE ACCORDING TO VARIOUS BIT WDITHS OF THE UNIT MULTIPLIER

	Half Precision, FP16				**Single Precision, FP32**				**Double Precision, FP64**			
Bit width of unit mul.	**Significant digits**	**No. of segments**	**Redundant bits**	**Redundant bits per segment**	**Significant digits**	**No. of segments**	**Redundant bits**	**Redundant bits per segment**	**Significant digits**	**No. of segments**	**Redundant bits**	**Redundant bits per segment**
11	11	1	0	0	24	3	9	3	53	5	2	0.40
12	**11**	**1**	**1**	**1**	**24**	**2**	**0**	**0**	**53**	**5**	**7**	**1.40**
13	11	1	2	2	24	2	2	1	53	5	12	2.40
14	11	1	3	3	24	2	4	2	53	4	3	0.75
15	11	1	4	4	24	2	6	3	53	4	7	1.75
16	11	1	5	5	24	2	8	4	53	4	11	2.75
17	11	1	6	6	24	2	10	5	53	4	15	3.75
18	11	1	7	7	24	2	12	6	53	3	1	0.33
19	11	1	8	8	24	2	14	7	53	3	4	1.33
20	11	1	9	9	24	2	16	8	53	3	7	2.33
...												

addition of exponential results. The sign is also decided by those modules. The fp_cal_mix module also supports 3 kinds of mixed-precision multiplications. The search module find_max_exp is used to find the max exponent of 10 inputs. Ten exponential subtraction modules exp_diff are used to calculate the difference between the max exponent and each exponent of 10 inputs. Once the multiplication results and exponent differences are generated, all the results are processed and shifted to the right by the alignment shifter module according to the corresponding exponent differences. Then the shifted results are passed to the 10-input adder tree for accumulation. 10-input adder tree are provided to support the multiple-precision DP operations. In this way, the reconfigurable design is proposed to support 20 FP16 DP, 5 FP32 DP, and 1 FP64 DP operations within two clock cycles.

III. MULTIPLE-PRECISION BIT-PARTITIONING METHOD

To implement the multiple-precision DPU operations in SIMD architecture, the significant digits of high precisions are partitioned into several segments. Each segment is used as one operand of a unit multiplier. To maximize the hardware usage, the bit width of the unit multiplier should be analyzed comprehensively.

Table I summarizes the segment numbers and redundancy performance for the DPU implemented in three precisions. The number of FP16 significant digits restricts the minimum bit width of the unit multiplier to 11. From the table, it shows that the bit width of 11 results in zero redundant bits for FP16. The redundant bits of FP64 is also small with only 0.4 per segment. However, the redundant bits of FP32 increases to 3 per segment. Along with bit width increasing larger than 11, the redundant performance is getting worsen for FP16 and FP32. The redundant performance is fluctuated for FP64 due to the segment numbers changing. For bit width of 12, the redundant bits of FP32 is minimized to zero. The redundant bits per segment of FP16 and FP64 are 1 and 1.4, respectively. The sum of these two values is also the smallest when comparing the sums under other bit width conditions. Thus, the bit width of 12 is selected for the unit multiplier.

By using 12-bit unit multiplier, the segment numbers for FP16, FP32 and FP64 are 1, 2 and 5, respectively. In Fig. 4, 24-bit operands for FP32 multiplication are partitioned into two 12-bit segments, i.e. a_1~a_0 and b_1~b_0. The multiplication can be

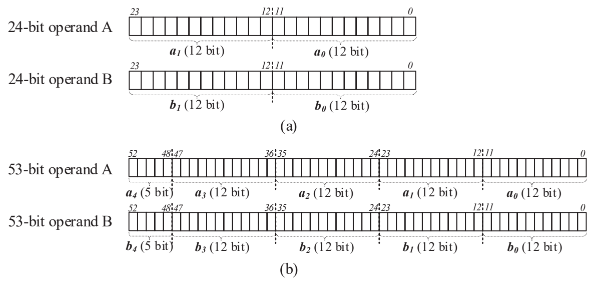

Fig. 4. The partitioning methods for the operands in FP32 and FP64 multiplications: (a) the 24-bit operands are divided into 2 12-bit segments; (b) the 53-bit operands are divided into 5 segments with 5:12:12:12:12 bit ratio.

TABLE II. SUMMARY OF THE MUTLIPLICATIONS TYPES AND THE REQUIRED UNIT MULTIPLIERS FOR PARTITIONED MULTIPLICATION IN THREE PRECISIONS

Data Type	Bit Width	Bit Ratio for Segments	Multiplication Types	No. of Unit Multipliers
FP16	11	12	1 times 12b×12b	1
FP32	24	12:12	4 times 12b×12b	4
FP64	53	5:12:12:12:12	16 times 12b×12b 8 times 12b×5b 1 times 5b×5b	25

realized by using four unit multipliers. The process of shifting and accumulating unit multiplier results is described as:

$$A\times B=\sum_{i=0}^{1}\sum_{j=0}^{1}(a_i\times b_j)\ll((i+j)\times 12) \quad (2)$$

where ≪ means the left-shift operation. The results of the unit multiplier for $a_i \times b_j$ are shifted according to the product of 12 multipling the sum of the segment numbers. Then all the shifted results are accumulated to obtain the final high-precision multiplication results. For the 24-bit multiplication, the unit multipliers shift the results of 0, 12, 12, 24 sequentially. For FP64 multiplication, the 53-bit operands cannot be partitioned into segments with equal bits. By using 12-bit multiplier, the operands are partitioned into 5 segments $a_4 \sim a_0$ and $b_4 \sim b_0$ with 5:12:12:12:12 bit ratio. Similar to the FP32 multiplication, the FP64 operation can be realized by using 25 12-bit multipliers. As shown in Eq. (3), the results of the unit multipliers are shifted and accumulated to get the final results.

$$A\times B=\sum_{i=0}^{4}\sum_{j=0}^{4}(a_i\times b_j)\ll((i+j)\times 12) \quad (3)$$

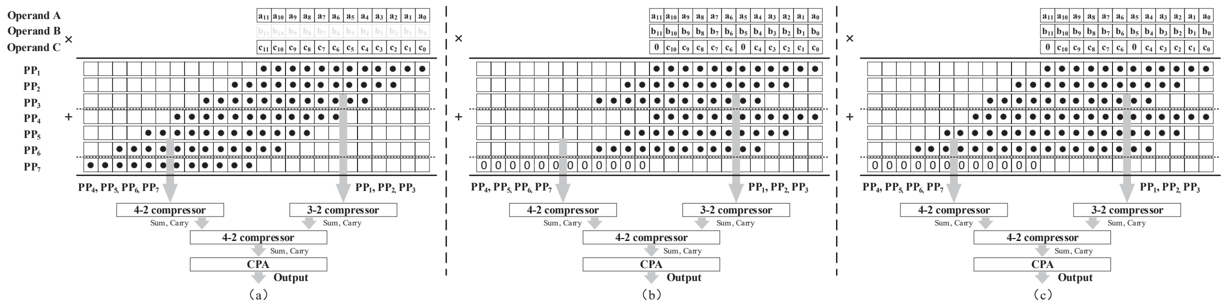

Fig. 5. The proposed fused multiplier supports 3 kinds of multiplications:(a) 12b×12b operations, (b) two 12b×5b operations, (c) 12b×5b and 17b×5b operations.

In Table II, the multiplication types with the corresponding times and the required total unit multipliers for three precisions are provided. For FP16 and FP32, 1 and 4 unit multipliers are required, respectively. For FP64, 25 unit multipliers are required by using the conventional designs. To reduce the number of the unit multipliers, fused multiplier with supporting mixed-precision multiplication is proposed. The fused multiplier can be implemented in three modes, i.e. 12b×12b operation, two 12b×5b parallel operation, 12b×5b and 17b×5b parallel operation. Table II shows there are total 16 12b×12b, 8 12b×5b and 1 5b×5b operations are required for FP64 multiplication. Based on the fused multiplier, the computing process can employ 16 unit multipliers for 12b×12b operation, 4 fused multipliers for parallel 8 12b×5b operation and 1 5b×5b operation. We remark that one 12b×5b operation and one 5b×5b operation can be merged as one 17b×5b operation. Thus, total 20 multipliers are needed for FP64 multiplication. In Fig. 3, the proposed work employs 10 unit multipliers including 6 conventional unit multipliers and 4 fused unit multipliers to implement the multiple-precision operations in 2 cycles. 4 fused unit multipliers have to be used in cycle 2 due to bit width constraints of the adder tree.

IV. Circuit Implmentation

A. Mixed-precision Fused Multiplier

The conventional radix-4 booth multiplier consists of three steps: partial product generation (PPG), partial product compression (PPC) and carry propagate addition (CPA). In Fig. 5(a), operand A and operand C are the multiplicand and multiplicator, respectively. The partial products (PPs) are generated by using the radix-4 booth encoder and processing operand A based on every three bits of the operand C with additional zero bits to the tail during the PPG step. Then the PP array is processed by the 4:2 or 3:2 compressors in the PPC step. Finally, the results are summed by CPA operation.

We proposed a mixed-precision fused radix-4 booth multiplier to meet the requirements of the bit-partitioning method for hardware reuse. The multiplier can perform one 12b×12b, two 12b×5b, or parallel 12b×5b and 17b×5b operations. For unsigned booth coding, zero bits must be added in the beginning of the multiplier operand for formula correction. When implementing unsigned 12b×12b operation, seven PPs are generated with booth coding. In Fig. 5, the PPs are separated into two parts PP_1~PP_3 and PP_4~PP_7, which are processed by 4:2 and 3:2 compressors, respectively.

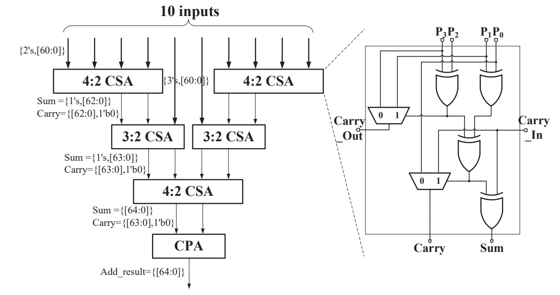

Fig. 6. The structure of 10-input adder tree.

For 12b×5b parallel operations, four inputs are required, i.e., two 12-bit multiplicands and two 5-bit multiplicators. In Fig. 5(b), two multiplicands are stored in operands A and B, two 5-bit multiplicators are stored in 12-bit operand C with adding zero bits before each MSB bit. When implementing 12b×5b operation, booth coding starts with first 5 bits (c_4~c_0) in operand C and generate the PPs by processing the operand A. Then the booth coding continues with the second 5 bits (c_{10}~c_6) in operand C and generate the PPs by processing the operand B. Due to adding bit zero as c5 and c11 before the MSB bits, PP_1~PP_3 and PP_4~ PP_6 generations are only based on the c4~c0 and c10~c6, respectively. PP_7 is zero due to c_{11} equal to zero, which does not affect PPG of 12×5 operation. In this way, two independent 12×5 operations during booth coding ensure the parallel operations. For conventional booth multiplier, the shifting bits of first six rows are 0, 2, 4, 6, 8, 10. For the proposed one, the shifting bits are 0, 2, 4, 0, 2, 4. In comparison, only the shifting bits of the second 3 rows are rearranged. All the following compressors and CPA can be reused.

The realization of 12b×5b and 17b×5b parallel operation is based on the splicing of 3 operands. For 12b×5b, it is the same with two 12×5 parallel operation by multiplying operand A and first 5 bits in operand C. For 17b×5b, the 17-bit multiplicand is composed by first 5 bits in operand C and the operand B. Then the operation is achieved by multiplying 17-bit multiplicand and the second 5 bits in operand C. The 17b×5b operation also generates 3 PPs, PP_4~PP_6. With shifting these 3 PPs with 0, 2 and 4 bits, the total bits are still within 24-bit widths, which avoids the overflow. The computing blocks are also reused.

B. Reconfigurable Adder Tree

The DPU design employs ten multipliers in one cycle. Thus, a 10-input adder tree is required for accumulation. Fig. 6

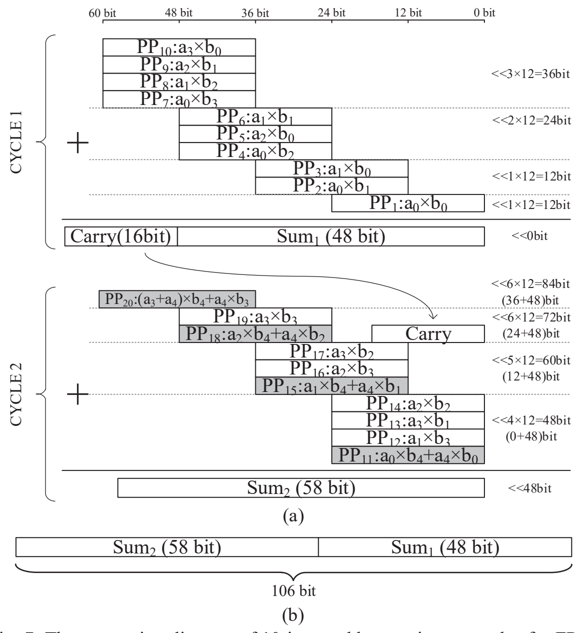
Fig. 7. The computing diagram of 10-input adder tree in two cycles for FP64 mode.

presents the internal structure, which employs two 4:2 CSAs, two 3:2 CSAs and one 4:2 CSA in three levels. A CPA is followed to calculate the final result. To speed up the computing process, the 4:2 CSA is custom designed with only 1.5× transmission delay of the 3:2 CSA.

For PPs accumulation of FP16 operation, all the 10 multiplication results are shifted by the alignment shifters according to the exponent differences only. Then the shifted results are signed and sent as the inputs of the adder tree. For FP32 accumulation, the ten low-order PPs are shifted according to the exponent differences and the sums of the segment numbers in Eq. (2). The sum in the first cycle is stored in the output registers, which is accumulated with the sum in the second cycle to obtain the final results of five FP32 operations. In Fig. 7, the FP64 accumulation process is illustrated in detail. The ten low-order PPs are shifted according to the exponent differences and the sums of the segment numbers in Eq. (3). Then the shifted PPs are added to get 48-bit sum and 16-bit carry signal. The carry signal is propagated to the second cycle. In the second cycle, the PPs in grey rectangle PP_{11}, PP_{15}, PP_{18} and PP_{20} are generated by the fused multipliers. The other six PPs are generated by the conventional unit multipliers. In this way, the higher 58-bit addition result obtained from the second cycle and the lower 48-bit addition result obtained from the first cycle are spliced together to obtain the multiplication result with 106-bit lossless accuracy. To do sign bit calculation, in Fig. 6, the sign bits are extending 2-bit or 1-bit in each level of the adder tree.

C. Normalization and Rounding

To normalize and round the results, the LZA [10] used in the proposed is designed to leading zero and leading one. Then the lza_cnt is added to exp_max. In the normalization pipeline, after norm shift, the rounding is designed to support roundTiestoAway rounding mode [1].

TABLE III. THE PERCENTAGES OF SIMULATED POWER AND AREA BREAKDOWN FOR THE BLOCKS OF THE DOT PRODUCT UNIT

Building Blocks	Area		Power	
	Value (um^2)	Percentage (%)	Value (mW)	Percentage (%)
Con. multiplier	1944.6	5.48	0.22	6.13
Fused multiplier	2491.5	7.03	0.26	7.24
Adder tree	2479.51	7.00	0.23	6.40
Delta calculation	991.18	2.80	0.09	2.50
Signed calculation &Align. Shifter	5968.85	16.84	0.56	15.60
normalization	4365.76	12.31	0.35	9.74
Total multipliers	21633.6	61.04	2.36	65.74
Total	354378.9	100	3.59	100

V. EXPERIMENTAL RESULTS

The performance evaluation of the proposed work is implemented. By constructing the proposed design in RTL level, the area occupation and power consumption values are obtained from the synthesis results by using Synopsys Design Compiler. The simulation is based on UMC 55-nm process. The state-of-the-art DPU designs [6-7,10] are implemented for direct comparisons. In addition, the multiple-precision performance evaluation is provided.

A. Power and Area Breakdown of the DPU unit

In Table III, the power and area performance of the main building blocks of the DPU unit are summarized. It shows that the multiplier block occupies the largest percentages of the power and area, which are 61.04% and 65.74%, respectively. The second dominant part, which is the alignment shifter with sign bit calculation, the corresponding percentages are 16.84% and 15.60%. Those two parts are accounting for 77.88% and 81.34% of the total area and power. Due to the addition of mode selection and new function implementation in the fused multiplier, the area and power consumption is larger than the conventional multiplier. The results show that there are only 11.2% area and 7.3% power overheads when comparing the proposed multiplier array with traditional multiplier array of 10 conventional unit multipliers. In addition, the critical path is 3.52ns.

B. Performance comparison between conventional multiple-precision multipliers and the proposed multiplier

We have demonstrated that the multiplier part dominates the hardware resources. We now investigate the performance comparison of the conventional multiple-precision multipliers and the proposed multiplier. To evaluate the proposed design, multiple-precision multipliers in [6-7] and [10] are constructed.

Table IV summarizes the area performance of each multiplier. The total area of multiplier in [6-7] represents the 53-bit array multiplier area. The total area of multiplier in [10] represents the total 64 15-bit unit multipliers for multiplication array. For the proposed work, the total area represents the total 20 12-bit unit multipliers (16 con. multipliers and 4 fused multipliers). The rate for each precision represents the utilization rate of the total hardware (area) per operation. The area per precision unit equals total area being divided by the

TABLE IV. PERFORMANCE COMPARISON OF MULTIPLIERS IN MULTIPLE PRECISIONS

Multiplier Types	Total	FP16 Unit		FP32 Unit		FP64 Unit	
	Area (um^2)	Area (um^2)	Rate (%)	Area (um^2)	Rate (%)	Area (um^2)	Rate (%)
[6-7]	26972.4	-	-	13486.1	50	26972.2	100
[10]	156089.6	19511.2	12.5	39022.4	25	78044.8	50
Proposed	**43267.2**	**2163.36**	**100**	**8653.44**	**100**	**43267.2**	**100**

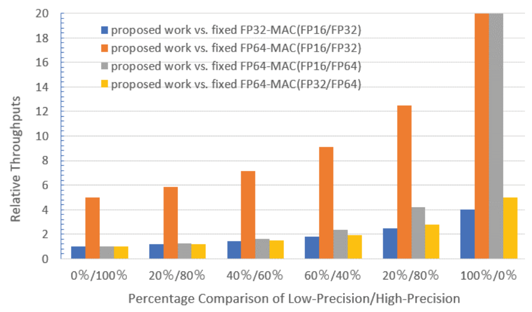

Fig. 8. Throughput comparison among fixed operations and the proposed multiple-precision dot product operations.

operation number. For [6-7], the system supports 2 FP32 and 1 FP64 operations. For [10], the system supports 8 FP16, 4 FP32 and 2 FP64 operations. The proposed design supports 20 FP16, 5 FP32 and 1 FP64 operations. From the table, it shows that the multiplier in [6] achieves the smallest unit area for FP64, which benefits from the direct 53-bit multiplier design. The proposed work achieves smallest unit area for FP16 and FP32, which saves at least 88.9% and 35.8% when comparing with the corresponding areas of [6-7] and [10], respectively. The proposed work also saves 44.6% area for FP64 when comparing with [10]. In addition, the utilization rates of the proposed work are 100% without gated idle hardware for all 3 precisions.

C. Multiple-precision Thoughput evaluation

For both the HPC and machine learning applications, the combination operations of different-precision data are required. Accordingly, it is of great significance to compare the throughput of fixed operations and proposed work under multiple precisions of different proportions. For example, when dealing with multiple-precision inputs of FP16 and FP32, one fixed FP32 DPU can only be used for one FP16 operation or one FP32 operation. For proposed work, the operations in both precisions can be implemented without waste of resources. When the proportion of FP16 and FP32 is 100%/0%, the resources occupied by one FP32 operation in the proposed work can be used to calculate four FP16 operations, so the relative throughput is 4 when compared with the fixed FP32 DPU. In Fig. 8, the relative throughputs of the proposed work compared to fixed FP32 and fixed FP64 operations under the proportions of FP16 and FP32 are 0%/100%, 20%/80%, 40%/60%, 60%/40%, 80%/20% and 100%/0% are listed. Besides, the relative throughputs of the proposed work compared to the fixed FP64 DPU when the multiple precisions are FP32 and FP64, FP16 and FP64 are also listed. It shows that as the proportion of low-precision FP data increases, the relative throughput increases. When compared with the fixed FP32 DPU under the FP16 and FP32 multiple-precision data, the increase is the slowest with maximum 4 relative throughput. When compared with the fixed FP64 DPU under the multiple-precision data with FP16, the relative throughput of the proposed work increases to maximum 20 for 100%/0% proportion.

VI. CONCLUSION

In this paper, a reconfigurable multiple-precision FP DPU is designed. Based on the proposed work, the reconfigurable architecture supports multiple-precision dot-product operations. Experimental results show that at least 88.9% and 35.8% area-saving performance improvement for FP16 and FP32 precision are obtained when compared with other state-of-the art designs. The utilization rates keep 100% for all 3-precision operations. For the throughput performance, maximum 20× improvement is achieved when compared with fixed operations.

ACKNOWLEDGMENT

This work was supported by the Science and Technology Innovation Committee Foundation of Shenzhen (Grant No. JCYJ20180504170454381), and the Key-Area Research and Development Program of Guangdong Province (Grant No. 2019B010142001), and Huawei Contact YBN2019115023.

REFERENCES

[1] IEEE Standard for Floating-Point Arithmetic, IEEE Std 754-2008, Aug. 2008.

[2] I. Sourdis, D. A. Khan, A. Malek, S. Tzilis, G. Smaragdos and C. Strydis, "Resilient Chip Multiprocessors with Mixed-Grained Reconfigurability," in *IEEE Micro*, vol. 36, no. 1, pp. 35-45, Feb. 2016.

[3] M. Blott, L. Halder, M. Leeser, and L. Doyle, "QuTiBench: Benchmarking Neural Networks on Heterogeneous Hardware," in *J. Emerg. Technol. Comput. Syst.* vol. 15, no. 4, Dec. 2019.

[4] V. Sze, Y. Chen, T. Yang and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," in *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295-2329,

[5] F. Dinechin, P. Echeverría, M. López-Vallejo, and B. Pasca, "Floating-Point Exponentiation Units for Reconfigurable Computing," in *ACM Trans. Reconfigurable Technol. Syst*. vol. 6, no. 1, May 2013.

[6] K. Manolopoulos, D. Reisis, and V. A. Chouliaras, "An efficient dual-mode floating-point multiply-add fused unit," in *Proc. 17th IEEE Int. Conf. Electron. Circuits Syst.*, pp. 5–8, Dec. 2010.

[7] L. Huang, L. Shen, K. Dai, and Z. Wang, "A new architecture for multiple-precision floating-point multiply-add fused unit design," in *Proc. 18th IEEE Symp. Comput. Arithmetic*, pp. 69–76, Jun. 2007.

[8] A. Haidar, S. Tomov, J. Dongarra and N. Higham, "Harnessing GPU Tensor Cores for Fast FP16 Arithmetic to Speed up Mixed-Precision Iterative Refinement Solvers," in *SC18: The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC), Dallas, TX, US*, pp. 603-613, 2018.

[9] S. Kuang, C. Liang, and M. Chang, "A Multi-functional Multi-precision 4D Dot Product Unit with SIMD Architecture," in *Springer Arab J Sci Eng*, vol. 41, pp. 3139–3151, Aug. 2016.

[10] H. Zhang, D. Chen and S. Ko, "Efficient Multiple-Precision Floating-Point Fused Multiply-Add with Mixed-Precision Support," in *IEEE Transactions on Computers*, vol. 68, no. 7, pp. 1035-1048, July 2019.

[11] V. Camus, L. Mei, C. Enz and M. Verhelst, "Review and Benchmarking of Precision-Scalable Multiply-Accumulate Unit Architectures for Embedded Neural-Network Processing," in *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 4, pp. 697-711, Dec. 2019.