Industrial Engineering and Applications
L.-C. Tang (Ed.)

doi:10.3233/ATDE230112

A FPGA Embedded DSP Supporting Parallel Multiple Low Bit-Width Multiply-Accumulate Operations

MiaoWang [a], Zhihong Huang [b, 1], Gang Cai [b], Junxuan Wang [a]
[a] *School of Communications and Information Engineering*
Xi'an University of Posts and Telecommunications
Xi'an 710121, China
[b] *Aerospace Information Research Institute, Chinese Academy of Sciences*
Ehiway Microelectronic Science and Technology Co.Ltd.
Beijing 100094, China

Abstract. With the continuous development of big data and hardware computing platforms, deep learning has been substantially applied in many intelligent scenarios. Recent studies have shown that using low bit-width networks in deep learning inference can effectively improve the overall performance of accelerator by reducing the computational ability requirements while maintaining the recognition accuracy of accelerator. Among them, low bit-width convolutional operations such as 8bit and 4bit are widely used in applications such as graph recognition. FPGA chip is the core key device of digital system, due to the excellent reconfigurability of FPGA, it has become one of the mainstream platforms in the field of deep learning accelerator. The current mainstream FPGAs are composed of higher bit-width multipliers due to the need to adapt to different computing application requirements, and the DSP module resources are used to perform low bit-width convolutional operations, which only occupy part of the multiplier bit-width, thus wasting a large amount of hardware on chip resources. Therefore, this paper proposes a DSP architecture of using large bit-width multipliers to compute low bit-width multiplications in parallel, so that the new DSP can realize double 8bit and 4bit multiply-accumulate operations without adding multipliers, and can support any combination of signed and unsigned data operations. The design is based on the commercial Stratix IV DSP architecture, and the overall circuit is designed with SMIC 14nm standard CMOS process. The experimental results show that when calculating the same number of 4-bit and 8-bit multiply-accumulate operations, the resource consumption area of the improved DSP is reduced by 43.5% and the speed is increased by 48%.

Keywords. Deep learning, FPGA, DSP, Low bit-width, Multiply-accumulation

1. Introduction

In recent years, artificial intelligence has developed rapidly, and deep learning algorithms have shown more advantages than traditional algorithms in many fields such as natural language processing, target detection, image classification and speech recognition, but their large amount of data and computation also put pressure on current

[1] Corresponding Author: Zhihong Huang, *huangzhihong@mail.ie.ac.cn*

hardware computing platforms[1]-[2]. The mainstream CNN models in deep learning algorithms even contain millions of parameters, and their computation consumes a large amount of hardware resources. Studies have shown that in some scenarios, the use of low-precision network models can achieve the goal of significantly improving accelerator performance and reducing memory usage while meeting accuracy requirements[3].

The current mainstream accelerated computing platforms include CPUs, GPUs, ASICs, and FPGAs. Among them, FPGAs have dynamic reconfigurable features and the hardware logic resources can be programmed to adapt to different layers of deep learning algorithms, thus becoming a widely used accelerated computing platform[4]. The parallelism of FPGAs for accelerated computing is usually related to the number of embedded DSPs. The embedded DSP modules in commercial FPGAs usually consist of fixed high bit-width multipliers. The DSP module realizes large bit-width multiplication or multiply-accumulate operations through cascade. While for lower bit-width convolution operations, the low utilization of multiplier bit-width makes the DSP not efficient in achieving low bit-width multiply-accumulate operations[5]-[6].

To solve this problem, this paper proposes a method of using a large bit-width multiplier to compute low bit-width multiplications in parallel, custom-designing the multipliers of the DSP so that the DSP can achieve double the number of multiply-accumulate operations in one cycle without increasing the number of multipliers, thus significantly improving the performance of FPGAs supporting low bit-width data convolution operations.

The rest of this paper is organized as follows. Section II describes the design implementation of the customized multiplier. The specific architecture of the DSP and the implementation of the multiply-accumulate function are given in Section III. Section IV compares the experimental results. Section V summarizes the full paper.

2. Customized Multiplier Design

The customized multiplier in this design is 18 bits wide and can be configured into different functional modes by control signals. The customized multiplier can implement one set of 18-bit multiplication operations, two sets of 8-bit multiplication operations and two sets of 4-bit multiplication operations, supporting any combination of signed and unsigned numbers.

2.1. Customized Multiplier Architecture Design

Fig.1 shows the structure of the customized multiplier, including the following parts: the data pre-processing block, which is used to reorganize the multiplicand and split the multiplicand into two sets of data, multiplicand1 and multiplicand2; the partial product generation block is used to Booth encode the multiplier and generate partial product with the output data of the data pre-processing block; the correction block is used to correct the partial product in the double 8bit multiplication mode, so that the partial product array can meet the requirements of two sets of 8bit multiplication operations; the data mux, used to select the partial product array of the multiplier in different modes; the compressor tree reduction block, which is used to compress the partial product array and get the final operation result.

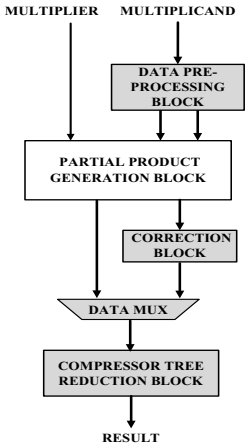

Figure 1. Structure diagram of customized multiplier

In the normal multiplication operation, the data is input normally; in the double multiplication operation, the input data of the multiplier is spliced in the way shown in Fig.2 The left Fig.2 shows the data input format of double 4bit multiplier, in which a and c represent the multiplicands of 4bit, and b and d represent the multipliers of two sets of sign bit expansion to 6bit; the right Fig.2 shows the data input format of double 8bit multiplier, in which a and c represent the multiplicands of two sets of 8bit, and b and d represent the multipliers of two sets of 8bit.

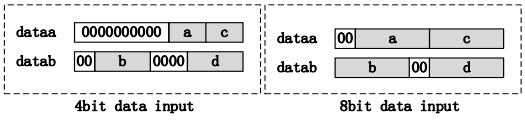

Figure 2. Double multiplication mode data input format

2.2. Data Pre-processing Block Circuit Design

Since the customized multiplier can calculate any combination of signed and unsigned numbers of 18bit data, the actual data involved in the operation is 19bit×18bit. In order to make the multiplier support two sets of 4bit and 8bit multiplication operations, this design preprocesses multiplicand by the data pre-processing block and splits multiplicand into multiplicand1 and multiplicand2. For the normal multiplication operation, the symbolic control signal Signa and the highest bit of data are logically combined to obtain the data symbolic bits according to the circuit in Fig.3.a, and the data symbolic bits are spliced with multiplicand to obtain multiplicand1 and multiplicand2. For the double multiplication operation, the symbolic control signal signa is logically combined with the highest bit of the two sets of data to obtain the symbolic bit of the data, and the two sets of data are spliced with the symbolic bit and then the symbolic bit is expanded to 19 bits to generate multiplicand1 and multiplicand2 according to the circuit in Fig.3.b. When X=4, the multiplier performs a double 4-bit multiplication operation; when X=8, the multiplier performs a double 8-bit multiplication operation. When X=4, the multiplier performs double 4-bit multiplication; when X=8, the multiplier performs double 8-bit multiplication.

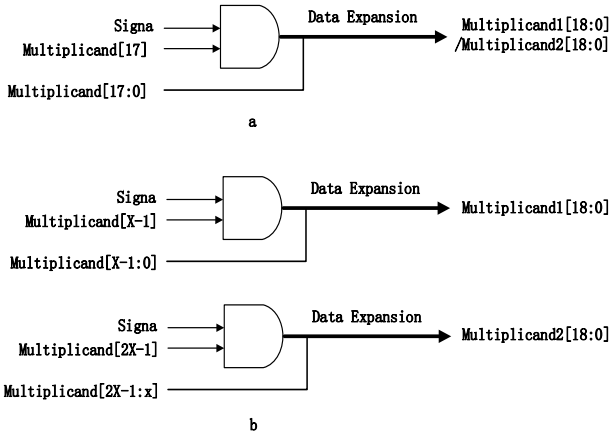

Figure 3. Data pre-processing block circuit

2.3. Partial Product Generation Block Optimization Design

This design uses Booth radix-4 encoding for the multiplier[7], as shown in Equation 1, and groups the multipliers in such a way that every three digits are used as a group and every two groups overlap by one, which will eventually produce 10 groups of partial products.

$$\begin{aligned} y &= -2^{n-1}\cdot y_{n-1} + 2^{n-2}\cdot y_{n-2} + 2^{n-3}\cdot y_{n-3} + \cdots + 2^{1}\cdot y_{1} + 2^{0}\cdot y_{0} \\ &= 2^{n-2}\cdot\left(-2y_{n-1} + y_{n-2} + y_{n-3}\right) + \cdots + 2^{1}\cdot\left(-2y_{2} + y_{1} + y_{0}\right) \end{aligned} \tag{1}$$

Table 1 shows the encoding method of Booth radix-4 encoding, where X represents the multiplicand, Y represents the multiplier, NEG, X1, X2P, ZP represents the control signal, and pp_i represents the partial product, which can be obtained by the action of the control signal and the multiplier. When the ADD signal is 1, the coding value is negative. In the table, y_{2i+1}、y_{2i}、y_{2i-1} respectively represents the three consecutive digits of the multiplier, and the possible values include {000, 001, 010, 011, 100, 101, 110, 111} 8 cases, corresponding to a total of 5 coding methods, respectively {-2X, -X, 0, X, 2X}, where X is the value of the multiplicand, 2X means to shift the multiplicand to the left by one bit, - X means to reverse and add one to the multiplicand, and - 2X means to reverse and add one to the multiplicand after moving the multiplicand to the left by one bit. Because only - X and - 2X need to add 1, which requires a 20-bit adder, and will introduce a large delay, so when the coding value is negative, add a bit 1 to the lowest bit of the partial product, and the compressor tree reduction block will uniformly sum it.

Table 1. Booth radix-4 encoding table

y_{2i+1}	y_{2i}	y_{2i-1}	NEG	X1	X2P	ZP	ADD	PP_i
0	0	0	0	0	1	1	0	0
0	0	1	0	1	0	1	0	X
0	1	0	0	1	0	0	0	X
0	1	1	0	0	1	0	0	2X
1	0	0	1	0	1	0	1	-2X
1	0	1	1	1	0	0	1	-X
1	1	0	1	1	0	1	1	-X
1	1	1	1	0	1	1	0	0

The expressions of each control signal can be obtained according to the truth table, and in order to reduce the energy consumption generated by burrs in the circuit, the encoding method used in this design synchronizes all paths in the encoder and partial product generator to ensure that the control signals have the same generation delay[8]. The specific encoding circuit is shown in Fig.4.

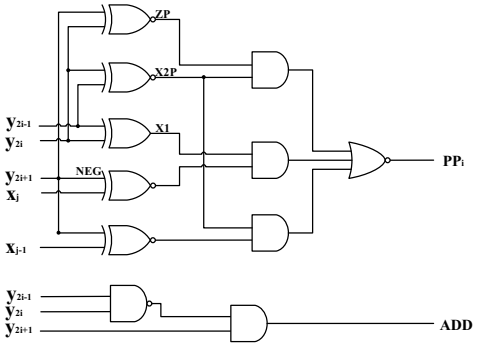

Figure 4. Booth radix-4 encoding circuit

In the partial product generation block, multiplier[9:0] is encoded with the encoding grouping as $\{y_{-1}\, y_0\, y_1\}, \{y_3\, y_2\, y_1\}, \{y_5\, y_4\, y_3\}, \{y_7\, y_6\, y_5\}, \{y_9\, y_8\, y_7\}$, which acts with multiplicand1 to generate the partial products r0-r4, encoding multiplier[17:9], encoding grouping as $\{y_{11}\, y_{10}\, y_9\}, \{y_{13}\, y_{12}\, y_{11}\}, \{y_{15}\, y_{14}\, y_{13}\}, \{y_{17}\, y_{16}\, y_{15}\}, \{y_{17}\, y_{17}\, y_{17}\}$, with multiplicand2 acts to generate partial products r5-r9. The three multiplication modes of the multiplier multiplex the partial product generation circuit, and the partial product array is shown in Fig.5. Since the highest data valid bit multiplier[17] can only produce three combinations of {000}, {001} and {111}, when multiplier is performing Booth radix-4 encoding, the lower 18 bits of r9 are taken as the valid value of the partial product.

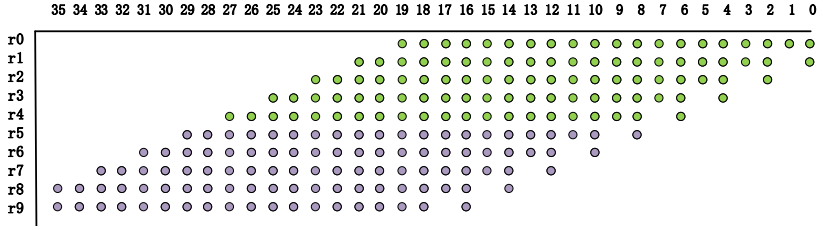

Figure 5. Partial product generation array diagram

The partial products generated by the Booth radix-4 encoding algorithm are signed numbers, and the highest bit represents the sign bit. Therefore, when summing the partial products, it is usually necessary to expand the sign bit of the partial products of each line, and then perform the compressed summation. This operation requires additional circuit resources. In this paper, the sign bit calculation is optimized by the method of pre-summing the sign bit [9]. Formula 2 accumulates and sums all the sign bits and simplifies them. The result of the simplification is to inverse the sign bits of the product of each part, and then add them with the constant data.

$$\begin{aligned} \text{sign} &= s_0 \sum_{n=19}^{35} 2^n + s_1 \sum_{n=21}^{35} 2^n + \cdots + s_6 \sum_{n=31}^{35} 2^n + s_7 \sum_{n=33}^{35} 2^n \\ &= \sum_{i=0}^{7} \overline{s}_i \cdot 2^{19+2i} + 2^{19} + 2^{20} + 2^{22} + 2^{24} + 2^{26} + 2^{28} + 2^{30} + 2^{32} + 2^{34} + 2^{35} \end{aligned} \tag{2}$$

After the operation, there is no need to expand the symbolic bits in the later calculation, only the highest bit of each partial product needs to be inverted and a new row is added, i.e., there are 10 partial products originally, and now there are 11 partial products. The new row is r10 = {11010101010101010110} with a bit width of 18 bits, and the new partial product array is shown in Fig.6.

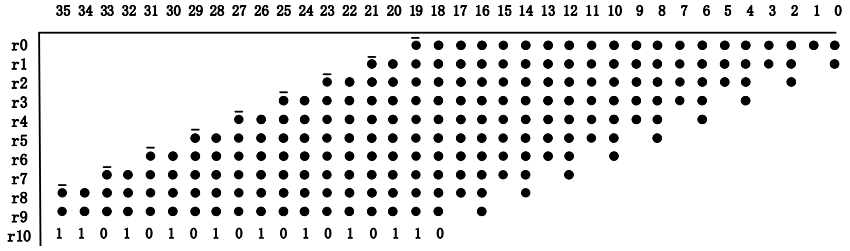

Figure 6. New partial product array diagram

2.4. Partial Product Correction Circuit

When the customized multiplier performs a double 8bit multiplication operation, the bit width of multiplier cannot meet the shared use of two sets of 8bit multipliers, so it is necessary to correct the value of r4. The correction processing circuit is shown in Fig7. When the multiplier D is an unsigned number and the highest bit is 1, sel1 selects Mutiplicand1[19:0] as the value of r4, and in other cases selects 20 'b0 as the value of r4.

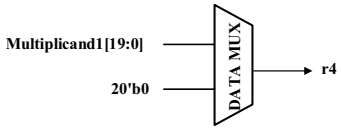

Figure 7. Partial product correction circuit

2.5. The Compressor Tree Reduction Block

As shown in Fig.8 to Fig.10, the compression tree unit compresses the partial product array with a total of four compression stages: stage1, stage2, stage3 and stage4 [10]. Among them, stage1 are compressed by means of 3-2 compressors; stage2 are compressed by means of a combination of 3-2 compressors and 4-2 compressors; stage3 are compressed by means of a combination of 3-2 compressors, 4-2 compressors and 5-2 compressors, and stage4 are implemented by using an adder.

When the customized multiplier is in double 4bit multiplication mode, as shown in Fig.7, the orange part is the partial product array of two sets of 4bit operations c × d and a × b, and after stage1, the two sets of summation and carry value are spliced to the high 9 bits and low 9 bits of the data in stage4 of the fourth compression stage, and finally the result of two sets of 4bit multiplication operations is obtained. When the customized multiplier is in double 8bit multiplication mode, as shown in Figure 8, the blue part is the partial product array of two 8bit operations c × d and a × b, and after stage1 and stage2, the two sets of summation and carry value are spliced to the high 17 bits and low 17 bits of stage4 data, and finally two sets of 8bit multiplication results are obtained; when the customized multiplier is in normal multiplication mode, as shown in Figure 9,

the black part is a set of partial product array of 18bit operation a × b, and after stage1, stage2, stage3 and stage4, a set of 18bit multiplication operation results are obtained.

The operation of stage4 is actually implemented in the form of a 36-bit adder. Since the design is a combinational circuit, the core of the chosen adder is the pursuit of adder speed, and after comparing the performance of various adders, the Koggle_Stone tree adder is chosen to achieve the final summation [11], The final customized multiplier output result can be a set of normal multiplier results or two sets of low bit width multiplication operation results.

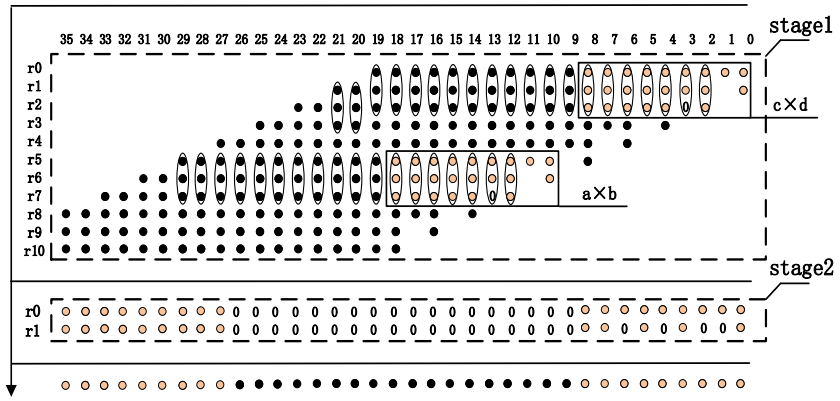

Figure 8. 4bit multiplication partial product processing flow

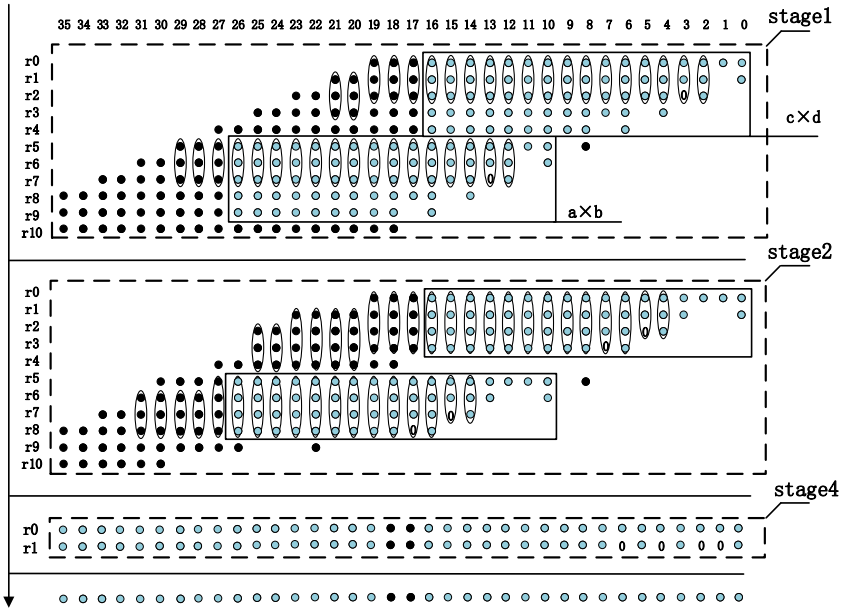

Figure 9. 8bit multiplication partial product processing flow

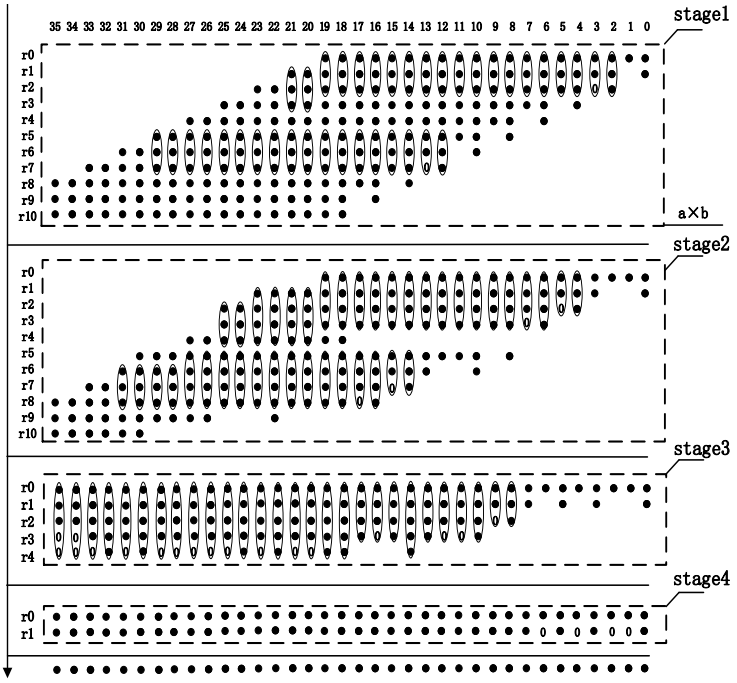

Figure 10. 18bit multiplication partial product processing flow

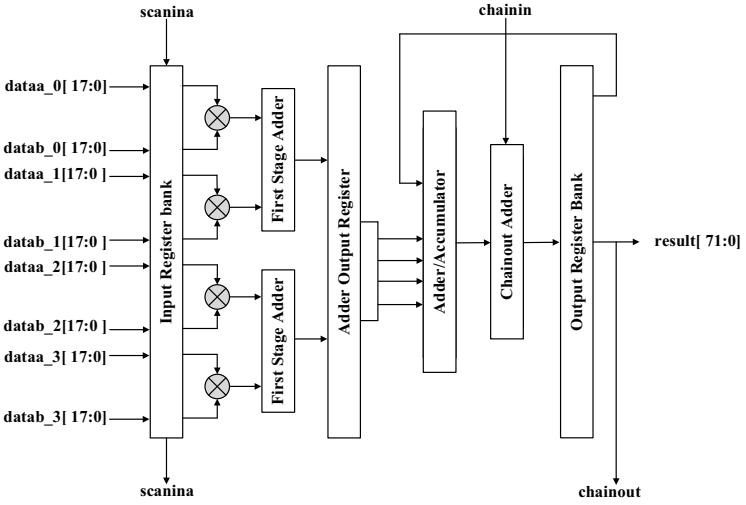

Figure 11. Architecture diagram of DSP module

3. DSP Module Architecture Design

The design is implemented with reference to Altera Stratix IV DSP and optimized to output the multiply-accumulate operation results of up to 8 operands in one beat and up

to 16 operands in two beats, thus achieving the purpose of multiplexing the multiplier resources, and the computational efficiency is twice as before.

3.1. DSP Module Composition

Fig.11 shows the block diagram of the DSP module proposed in this paper, which consists of: the input register bank for pre-processing the input data to be computed in order to choose whether to store it or not; the multiplication unit, including four groups of customized multipliers for performing ordinary multiplication and double low bit width multiplication; two first-stage adders for adding the results of the multiplier unit separately; the adder output register is used to store the operation results obtained from the summation process; the adder/accumulator is used to perform the secondary addition operation or the accumulation operation; the chain adder is used to perform the chain addition operation; and the output register is used to store the final operation results.

3.2. DSP Module Multiply-accumulate Function Introduction

In the normal multiply-accumulate mode, the customized multiplier is configured to function as a normal multiplier and the operands are input normally. The DSP in this mode can implement single multiplier multiply-accumulate, two multiplier multiply-accumulate, three multiplier multiply-accumulate, and four multiplier multiply-accumulate operations. The output data of the multiplication operation unit is added by the first adder and the second adder respectively to obtain the sum of two multiplication additions, and then sent to the accumulator module to perform the addition operation with the output data of the previous cycle to obtain the value of 18bit multiply-accumulate operation. The result of the accumulator operation can be selected whether to continue the chain addition operation or not, and finally the result is output directly or after registering.

In the double multiply-accumulate mode, the customized multipliers are configured as double multiply function. 8 sets of data a1 × b1, c1 × d1, a2 × b2, c2 × d2, a3 × b3, c3 × d3, a4 × b4, c4 × d4 are spliced to the DSP module input port according to the rules. After multiplication operation unit calculation, 8 sets of multiplication operation results are obtained. Since the two sets of operation results of each customized multiplier output are separated by protection bits, thus the data can directly enter the first stage adder and the second stage adder for addition calculation, and the two outputs obtained contain the sum of four sets of multiplications. The two outputs and the sum are split into four data, and the accumulator module performs the addition operation with the output data of the previous cycle to obtain eight sets of 4bit or 8bit multiply-accumulate values.

4. Comparison of Experimental Results

In order to compare with the existing Stratix-IV DSP IP core on the same platform and process, this paper completes its DSP IP design using the same design methodology and flow, called Altera DSP, based on the existing Stratix-IV datasheet and the corresponding design documentation [12]-[13]. A whole DSP contains two identical DSP architecture, which is completed by the common multiplier design. Next, the common multiplier is replaced by the customized multiplier and the improved DSP is called enhanced DSP. The study was performed with standard SMIC 14nm CMOS process and the logic

synthesis of both designs are finished by Synopsys DC tool to obtain area and timing information respectively. The maximum frequency of the DSP module was obtained by adjusting the clock period to detect timing violations.

Table 2 shows a comparison of the Altera Stratix IV DSP, the Altera DSP, and the enhanced DSP module proposed in this paper, which supports double multiply-accumulate operations. Among them, the Altera DSP has the smallest critical path delay and area with a frequency of up to 800 MHZ and an area of 18272.1 μm^2, which is mainly caused by the use of more advanced process nodes, while the enhanced DSP has a 3.9% performance degradation due to the enhancement of the customized multiplier double multiplication operation function, compared to the Altera DSP, and the area is increased by 13.0%.

Table 2. Comparison of different DSP block logic synthesis results

	Stratix-IV DSP	**Altera DSP**	**Enhanced DSP**
Process Technology	SMIC 40nm	SMIC 14nm	SMIC 14nm
Area/ μm^2	No data	18272.1	20645.2
Maximum Frequency/MHZ	550	800	769.2
Critical path delay/ns	1.82	1.25	1.30

Table 3 shows the comparison of the improved functions of the two DSP blocks. Due to the improvement of the parallelism of the low bit width operation of the customized multiplier, the enhanced DSP can support 16 sets of multiplication, 16 sets of multiply-addition and 16 sets of multiply-accumulation for 4 bit. For 8bit operation, due to the bit width limitation of the DSP block output port, it can support up to 9 sets of multiplication, 16 sets of multiply-addition and 16 sets of multiply-accumulation. In addition, enhanced DSP can also support all functions of Altera DSP.

Table 3. Comparison of DSP block functions

DSP Block	**4bit Multiplication**	**8bit Multiplication**	**4bit Multiply-Addition**	**8bit Multiply-Addition**	**4bit Multiply-Accumulation**	**4bit Multiply-Accumulation**
Altera DSP	8	8	8	8	8	8
Enhanced DSP	16	9	16	16	16	16

Considering the computing requirements of low precision networks in deep learning, this paper mainly focuses on the improvement of DSP's performance in low bit wide multiply-accumulate operation. Table 4 shows the comparison between the occupied area and the calculation time when using both DSPs to perform the 4bit and 8bit multiplication related operations. When performing the same number of 4bit multiplication, 4bit and 8bit multiply-addition, multiply-accumulate operations, the total on-chip resource occupation area of the enhanced DSP is reduced by 43.5% and the speed is increased by 48% compared with the Altera DSP. When performing the same number of 8bit multiplication operations, the area loss will be 0.43%, however the time performance will also be increased by 8.4%.

Table 4. Comparison of different DSP blocks for different bit-width multiply-accumulate operations

Realization Of Different Arithmetic Cases	**DSP**	**Area /μm^2**	**Area Change Rate**	**Time Required /ns**	**Time Change Rate**
16 sets of 4bit multiplication	Altera DSP	36544.2	-43.5%	2.5	-48%
	Enhanced DSP	20645.2		1.3	
	Altera DSP	36544.2	-43.5%	2.5	-48%

16 sets of 4bit multiply-addition	Enhanced DSP	20645.2		1.3	
16 sets of 4bit multiply-accumulation	Altera DSP Enhanced DSP	36544.2 20645.2	-43.5%	2.5 1.3	-48%
16 sets of 8bit multiplication	Altera DSP Enhanced DSP	36544.2 20645.2	-43.5%	2.5 1.3	-48%
72 sets of 8bit multiply-addition	Altera DSP Enhanced DSP	164448.9 165161.6	0.43%	11.35 10.4	-8.4%
16 sets of 8bit multiply-accumulation	Altera DSP Enhanced DSP	36544.2 20645.2	-43.5%	2.5 1.3	-48%

5. Conclusion

In this paper, we propose a method of using a large bit-width multiplier to implement low bit-width multiplication in parallel, , custom-design the multiplier to replace the traditional multiplication module, design an improved DSP module architecture, and experimentally compare the performance with the traditional DSP. Compared with the original structure, the enhanced DSP can reduce the resource usage area by 43.5% and increase the speed by 48% when calculating the same number of 4bit and 8bit multiply-accumulate operations with low bit-width convolution. The improved DSP makes the FPGA more adaptable to the computational requirements of low-precision networks in deep learning.

References

[1] Wang, K. F. , and X. H. Huang . "Research of Image Recognition of Plant Diseases and Pests Based on Deep Learning." International Journal of Cognitive Informatics and Natural Intelligence (IJCINI) 15(2021).

[2] Canziani, A. , A. Paszke , and E. Culurciello . "An Analysis of Deep Neural Network Models for Practical Applications.", 10.48550/arXiv.1605.07678. 2016.

[3] Boutros, S. Yazdanshenas and V. Betz, "Embracing Diversity: Enhanced DSP Blocks for Low-Precision Deep Learning on FPGAs," 2018 28th International Conference on Field Programmable Logic and Applications (FPL), Dublin, Ireland, 2018, pp. 35-357, doi: 10.1109/FPL.2018.00014.

[4] Skliarova, I. . "A Survey of Network-Based Hardware Accelerators." (2022).

[5] Sommer, J. , et al. "DSP-Packing: Squeezing Low-precision Arithmetic into FPGA DSP Blocks." (2022).

[6] Rasoulinezhad, Seyed Ramin , et al. "PIR-DSP: An FPGA DSP Block Architecture for Multi-precision Deep Neural Networks." 2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM) IEEE, 2019.

[7] Sma, B , and B. Dja . "An Optimistic Design of 16-Tap FIR Filter with Radix-4 Booth Multiplier Using Improved Booth Recoding Algorithm - ScienceDirect." Microprocessors and Microsystems (2020).

[8] Fried, R. . "Algorithms for Power Consumption Reduction and Speed Enhancement in High-Performance Parallel Multipliers." Patmos (2000).

[9] Annaratone S. Digital CMOS circuit design[M]. Springer Science & Business Media, 2012.

[10] Rao, M Jagadeshwar , and S. Dubey . "A high speed and area efficient Booth recoded Wallace tree multiplier for fast arithmetic circuits." Microelectronics & Electronics IEEE, 2012:220-223.

[11] Knowles S. A family of adders[C]//Proceedings 14th IEEE Symposium on Computer Arithmetic (Cat. No. 99CB36336). IEEE, 1999: 30-34.

[12] Stratix III D H. Vol. 1[J]. Chapter, 2007, 6: 6-1.

[13] Stratix V. Device handbook, volume 1: Device interfaces and integration[J]. Altera, June, 2012