A Reconfigurable High-Performance Multiplier Based on Multi-Granularity Design and Parallel Acceleration

Feng Jing
Institute of Automation, Chinese Academy of Sciences, University of Chinese Academy of Sciences
95 Zhongguancun East Road, 100190, Beijing, China
fengjing2015@ia.ac.cn

Zijun Liu, Xiaojun Ma, Guo Yang, Guo Peng and Donglin Wang
Institute of Automation, Chinese Academy of Sciences,
95 Zhongguancun East Road, 100190, Beijing, China
{zijun.liu & xiaojun.ma & guoyang2014& guopeng2014 &donglin.wang }@ia.ac.cn

Abstract—**This paper proposes a reconfigurable high performance multiplier (RHPM) based on multi-granularity design and parallel acceleration. Capable of supporting multiple precisions for different processing requirements, the RHPM can perform one 32x32, two 16x16, or four 8x8bit unsigned/signed multiplication, or one 16x16, or two 8x8bit complex number multiplication. The structures of the partial product generator and the partial product accumulator are improved in the paper, so as to reuse most of the hardware resources. Compression can be completed automatically by means of recording the validity of every bit in the partial product array which accelerates the computation dramatically. The RHPM is implemented with TSMC 28nm technology, exhibiting a 0.68s of the critical path delay, while consuming only 0.6281mW in power. Results show its significant superiority in terms of performance and power efficiency compared with our previous work or other similar products.**

Keywords-compression; high speed; multi-granularity ; parallel; power efficient ; reuse; reconfigurable

I. INTRODUCTION

The RHPM is applied in the second version of the high performance mathematical processor (MaPU) which has been taped out [1]. MaPU is a 512bits SIMD (Single Instruction Multiple Data) arithmetic instruction architecture of complex vector, divided into eight 64-bit parallel data paths. Each path includes two internal 32-bit multipliers.

The fields of multimedia processing applications, wireless communication and high-performance computing have the characteristics that containing data-intensive and complex computations. Therefore, multiplier plays an essential role in the digital signal processor which would determine the execution time of the whole processor program.

In digital signal processing domain, 8-bit, 16-bit and 32-bit data formats are used widely for different scenarios. A large number of digital signal processing algorithms, such as FFT, need to perform complex operations. The reusable structure of multi-granularity parallel multiplier is designed in this paper, supporting word, short and byte types of data. That is, the RHPM realizes one 32x32-bit, two 16x16-bit or four 8x8-bit signed/unsigned real number multiplication, one 16x16-bit or two 8x8-bit complex number multiplication. The RHPM implements multiple granularities in parallel, increasing the throughput by giving different outputs at the same time, decreasing the power by a novel automatic compression with the indicated valid bit of the partial product array, decreasing the area by using the identical multiplier for 32-bit, 16-bit and 8-bit multiplications.

After the design is completed, we use Synopsys Design Complier to synthesize the RTL level code. Compared to others, we come to the conclusion that this kind of multiplier has the characteristics, high performance, small area and low power.

II. RELATED WORK

Most traditional sub-word parallel algorithms [2] can be divided into two categories, generating large width result from the operation of small width results and reusing the maximum width operation to generate small width result.

In the M unit of TI6678 [3], the large data width multiplier is composed of some small width multipliers and accumulation units. That is, smallest multiplier is used to constitute large data width multipliers. Firstly, we calculate multiplication results of four 16x16-bit. Then we take the four results as a group to obtain 32x32-bit multiplication results. However, in this way, the delay cannot be effectively ensured.

In the paper [4], the 64-bit multiplier hardware resource is reused while performing small width multiplication operation. The multiplication array consists of sixteen multiplication units, in the form of a 4x4 array arrangement. Each multiplication unit is similar to 16x16 multiplier, and contains the Booth encoding and partial products compression internally. Sixteen arithmetic units form a 64-bit multiplication. Four arithmetic units form the path1 to realize 32-bit multiplication. When it performs 16-bit multiplication, each unit is an operation path. This method would cause overlapping between different SIMD paths, so it is necessary to mark the location of arithmetic unit However, the realization of design becomes more complex, and wastes the power and area.

III. ARCHITECTURE

As shown in figure 1, the architecture of RHPM, as we all know that the high speed multiplication contains three main

steps. We reduce the number of partial products with modified Booth encoding. The 3-2 compressor and Wallace tree are used to achieve compressing the array quickly. After obtaining two intermediate sums from the previous operation, we use a fast adder to get the final result. The multi-granularity parallel in generation and compression of partial product array is proposed in this paper, which is the key to the design.

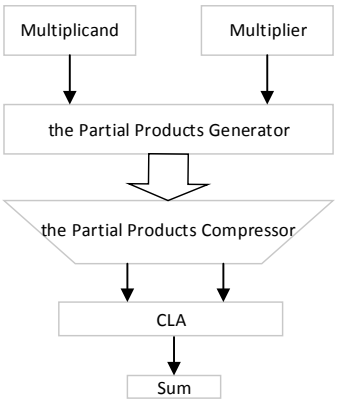

Figure1. The architecture of the RHPM

IV. MULTI-GRANULARITY PARALLEL MULTIPLICATION

A. Booth Encoding and Sign Extension

1) Modified Booth Encoding

In the multiplication accumulation operation, the partial products generator mainly uses the modified booth algorithm (MBA) to reduce the number of partial products greatly, thus enhancing the operating speed of the entire multiplication accumulator [5][6]. The multiplier and multiplicand are 32bits variables. We divide the multiplier X into four segments {X3, X2, X1, X0}. In each segment, the right-most bit is the first bit, and if it is the lowest bit of this granularity, a zero should be extended to the right of it. Otherwise, the previous segment last bit should be copied and extended to the right-most. In this way, we obtain a 36bits variable called $X^{'}$.

In word mode:
$\{X3^{'},X2^{'},X1^{'},X0^{'}\}=\{X3,X2[7],X2,X1[7],X1,X0[7],X0,0\}$ (1)
In short mode:
$\{X3^{'},X2^{'}\}=\{X3,X2[7],X2,0\}$, $\{X1^{'},X0^{'}\}=\{X1,X0[7],X0,0\}$ (2)
In byte mode:
$X3^{'}=\{X3,0\}$, $X2^{'}=\{X2,0\}$, $X1^{'}=\{X1,0\}$, $X0^{'}=\{X0,0\}$ (3)

For multiplicand Y, we do the same extension as the multiplier, and we know that extending a zero at the lowest bit changes the value into twice of the former.

Take the $Y0^{'}$ as example, we use radix-4 Booth algorithm to encode according to the each 3-bit of $X3^{'}$, $X2^{'}$, $X1^{'}$ and $X0^{'}$ separately. In this way, we obtain coded variable called BoothBit which represent one value in a set of {0, Y, -Y, 2Y, -2Y}. In the table 1, the value of variable S is 0 for unsigned operands and 1 for signed operands. The variable OBit represents the highest bit of the partial product. The variable EBit represents the sign bit of the sign extended partial product. The value of variable SBit is 1 when the booth encoding term is negative. The value of EBit is equal to the value of SBit while the inputs are unsigned number. A partial product is made up of the OBit and the BoothBit, and having corresponding the EBit and the SBit.

TABLE I. RADIX-4 BOOTH ENCODING

$X^{'}$	PP_i	BoothBit	OBit	EBit	SBit
000	0	0	0	0	0
001	Y	$Y^{'}[8:1]$	S& $Y^{'}[8]$	S& $Y^{'}[8]$	0
010	Y	$Y^{'}[8:1]$	S& $Y^{'}[8]$	S& $Y^{'}[8]$	0
011	2Y	$Y^{'}[7:0]$	$Y^{'}[8]$	S& $Y^{'}[8]$	0
100	-2Y	~ $Y^{'}[7:0]$	~ $Y^{'}[8]$	~(S& $Y^{'}[8]$)	1
101	-Y	~ $Y^{'}[8:1]$	~(S& $Y^{'}[8]$)	~(S& $Y^{'}[8]$)	1
110	-Y	~ $Y^{'}[8:1]$	~(S& $Y^{'}[8]$)	~(S& $Y^{'}[8]$)	1
111	0	0	0	0	0

2) The Limited Sign Extension

Take 16x8-bit multiplication as an example, we use radix-4 booth algorithm to obtain five items of partial products. To accumulate these products, we have to extend the sign bit. The conventional algorithm of sign extension will result in the waste of hardware resources, and also increase power for computing. So we use the limited sign extension technique called ES Coding to simplify the extension [7].

The reason why we separate the ES encoding and the partial product encoding is that for different operating widths the divisions of ES encoding are different. If we put them together, it would cause the increase of critical path delay. However, separating them, then putting the ES encoding results in the compressing period will not generate additional delay.

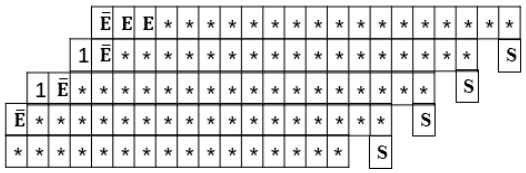

Figure2. The simplified sign-extension method. The variable S and E represent the SBit and the EBit of the table1 respectively. For unsigned operands, the sign extension makes $x_9=x_8=0$, so the last item of partial product is 0 or the multiplicand Y. For the signed operands, the sign extension makes $x_9=x_8=x_7$, so the last item of partial product is 0.

B. Generation of the partial product array

Based on the above preparation works, we start to construct the original partial product arrays for different granularities. The RHPM can support three width modes, and real number or complex number operation.

1) Word mode

For 32x32-bit multiplication, we get seventeen items of partial product (pp0~pp16). We arrange the OBit and four BoothBit items of each partial product in a line, separating the ES coding. The size of the partial product array called WordOrgData is 19x64bits in figure3. From the figure2 illustration, we know that the sixteenth line (pp16) is 0 or Y. The seventeenth line consists of the sign extension of pp1~pp15 and the SBit of pp0~pp15. The eighteenth line is the sign extension of pp0.

In addition, we construct a valid array for partial product array which is equal to its length and width. The shadow part of the partial product array indicates these bits are valid, so the value of the same position on valid array is 1. Otherwise the valid value of this bit is 0. Whether the data is valid provides the basis for the subsequent reordering and automatic compression operations later.

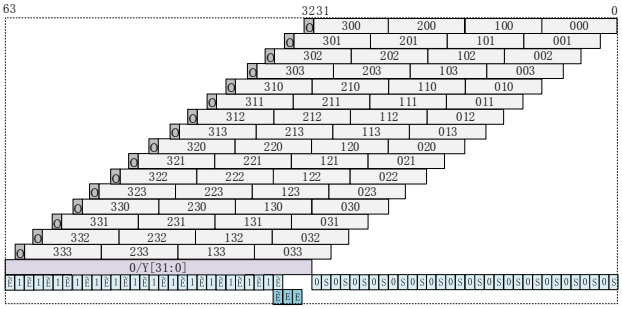

Figure3. The description of WordOrgData, xxx referring to BoothBitxxx. The first number represents one of the {Y3', Y2', Y1', Y0'}. The second number represents one of the {X3', X2', X1', X0'}. The third number represents the segment of X'_i.

2) Short mode

We divide the X' and Y' into high and low two parts separately. For the sake of brevity, the block1, block2, block3 and block4 represent L*L, H*L, H*H and L*H respectively. Each block has eight rows respectively which includes the OBit and two BoothBit items. We piece them together to an array called ShortOrgData in figure 4.

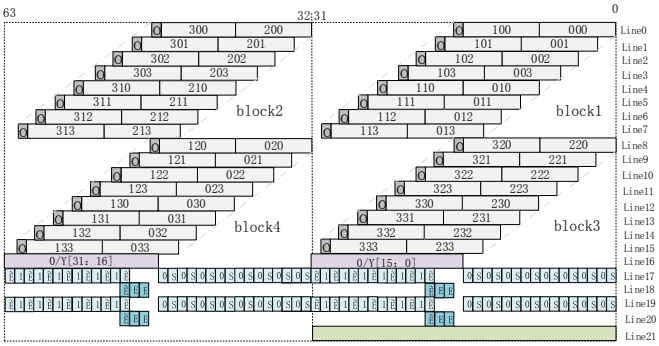

Figure4. ShortOrgData is 22x64bits.The seventeenth line are the sign extension line1~line7 and the SBit of line0~line7. The nineteenth line are the sign extension line9~line15 and the SBit of line8~line15. The eighteenth and twentieth line are the sign extension of line0 and line8 separately. The twenty-first line is an additional line for complex number which will explain later.

- two 16x16bit real number multiplication

For the short mode of real number, the two 16x16bit multiplications are independent. The result is a combination of two parts {H*H, L*L}, so the block1 and block3 are valid and the rest are invalid. For the consideration of irrelevance, we shift the block3 to the location of the block4.

- one 16x16bit complex number multiplication

The complex number multiplication requires four multiplication operations and two addition operations. An ordinary digital signal processor may need six clock cycles to complete the complex number multiplication. However, if the multiplier supports the complex operation, then the result can be obtained after one clock cycle, and the performance is improved by six times. On the other hand, it can reduce the difficulty of programming and improve the efficiency of code by supporting the instruction level complex operation.

Take the 32-bit operand as a 16-bit complex number, and the lower 16bits are the real component and the upper 16bits are the imaginary component. In this way, the real portion of the result will be (L*L-H*H), and the imaginary portion of the result will be (H*L+ L*H). The block1, block2, block3, and block4 are valid. It is worth noting that the block3 which represents H*H need to take the NOT operation. Only when the bit is valid, we take the NOT operation, in the Verilog HDL description of ~data = ~valid + data & valid. The twenty-first line in figure4 is the ~valid part in the operation. In addition, we take the complex as signed format, so the sixteenth line is zero.

$$(L+Hj) * (L+Hj) = (H*L+L*H)\ j+ (L*L-H*H) \quad (4)$$

3) Byte mode

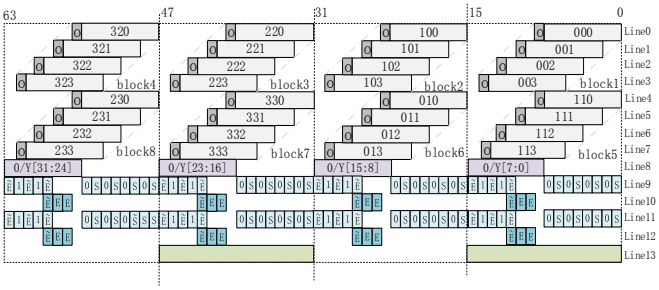

Figure5. For the byte mode ,the partial product array called ByteOrgData is 14x66bits. The ninth line are the sign extension of line1~line3 and the SBit of line0~line3. The eleventh line are the sign extension of line5~line7 and SBit line4~line7. The tenth and twelfth line are the sign extension of line0 and line4.

- four 8x8bit real number multiplication

The block1, block3, block5 and block7 are valid. We should shift the block5 and the block 7 to the location of the block6 and the block 8 separately.

- two 8x8bit complex number multiplication

All of the blocks are valid. We know that the block5 and block7 need to take the NOT operation, so the thirteenth line is the combined ~valid of this two parts.

C. Compression of the partial products

1) Reodering and Union

In the conventional compression of the partial products, a large number of bits are not utilized, so the power consumption can't be ensured. Reordering technique of partial products is applied on the RHPM which can solve this problem.

The parallelogram structure of partial products is rearranged into tree structure. We sort every valid bit in each column of valid array, and corresponding valid bit of partial product array values are shifted to the top of the column. This helps to reduce the total wire length in the cell placement and reduce resource consumption compared to parallelogram structure.

We union the reordered partial product arrays of three modes, and then get a new 64x18bit array which has eighteen items of partial products finally.

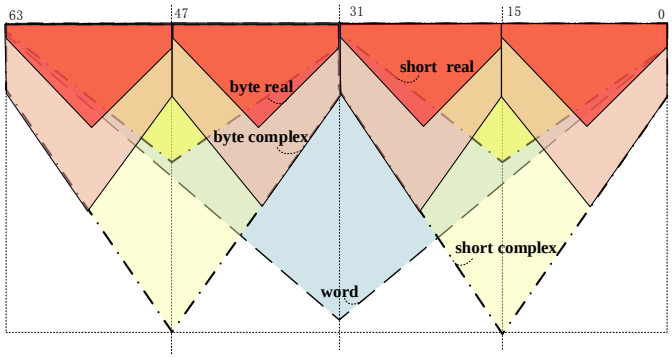

Figure6. The union of three mode, a blue area enclosed by the dashed line is the word mode. The yellow area enclosed by the dash dot line is the short mode. The red area enclosed by the solid line is the byte mode. In addition, the deep color is the real number operation, the light is complex number opertion.

2) The structure of compression tree

In the stage of compression, the Wallace tree has the advantage in reducing carry propagation delays, accelerating the addition operation. The Wallace tree structure compresses the partial products in parallel, generating new partial product arrays which are compressed again and again until it remains the last two lines [8]. The eighteen lines of partial products that we get are recorded as PP0, PP1, PP2…PP17. As shown in figure 7, after four levels of CSA, we obtain the two intermediate product. The final sum and carry are then sent to the CLA to generate the final result product.

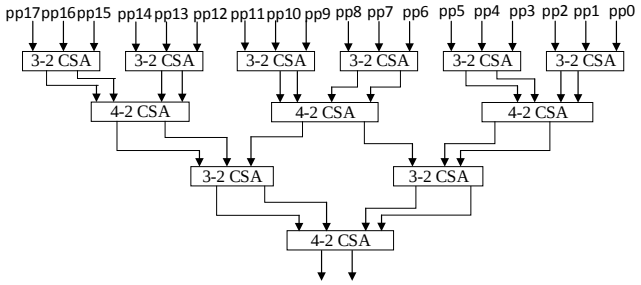

Figure7. The structure of compression tree. At first, use six 3-2 compressors compress 18 parts into 12 items of intermediate results. After the second compression of the 4-2 compressor [9] , we use the 3-2 compressor and 4-2 compressor compress six intermediate results after the second level and four intermediate results which generated subsequently.

V. RESULT

Based on the 28nm standard cell library, we use Synopsys Design Compiler to synthesize the RHPM. Table 2 shows the comparison of delay, area and power with other multipliers. Compared with single word mode implemented in RHPM, 16-bit, 8-bit real number and complex number multiplication are added in, but only cause 19.4% increase of area. Under the same conditions, compared with the multiplier in the previous version of MaPU [1] which have not realized the complex multiplication, the power reduce 60% and the area reduce 34.24%. As for the delay, the RHPM has two pipelines, while the previous version has four pipelines.

Paper [10] performs one 32-bit, two 16-bit and four 8-bit multiplications in 65nm process, but has not realize complex multiplication. With every generation improved in CMOS process, the delay decreases 30%, and the area reduces 30%. According this law, compared with FT-SIMD in [10], the RHPM has superiority on 64% reduction in power and 52.78% reduction in delay.

TABLE II. EXPERIMENTAL RESULTS AND COMPARSION

Multiplier	***Delay(ns)***	***Area(um²)***	***Power(mW)***
The single word mode	0.57	5438	0.5236
Ref.[1]	0.49	9872	1.5724
Ref.[10]	3.6	15593	4.39
The RHPM	**0.68**	**6493**	**0.6281**

VI. CONCLUSION

This paper proposes a novel high-performance multi granularity and parallel multiplier, supporting one 32x32, two 16x16 or four 8x8bit signed/unsigned multiply operation, or one 16x16 or two 8x8bit complex multiply operation. The RHPM has the advantages of high performance, small area and low power compared with other multipliers.

REFERENCES

[1] Wang, D., et al. MaPU: A novel mathematical computing architecture. in High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on. 2016. IEEE.

[2] Lee Y, Avizienis and Efficiency R, Bishara A. Exploring the Tradeoffs between Programmability in Data-Parallel Accelerators[C]. Proceedings of the IEEE International Symposium on Computer Architecture. San Jose, USA, June, 2011: 129~140.

[3] Timothy Anderson, Duc Bui. A 1.5 GHz VLIW DSP CPU with integrated floating point and fixed point instructions in 40nm CMOS [C]. 20th IEEE Symposium on Computer Arthmetic. 2011:8286

[4] WU Hu-cheng, LIU Yang-xu-rui, LIU Jiar-ping. Architecture of a High Performance SIMD Multiplication Array, Microelectronics&Computer Vol.31, no.3, 2014

[5] P.E.Madrid, B.Miller and E.E.Swartzlander, "Modified Booth Algorithm for High Radix Fixed -Point Multiplication," IEEE Transactions on Very Large Scale Integeration (VLSI)Systems, Vol.1, no.2, pp164-167 June 1993R

[6] Xu Bangjian. DSP algorithm and system structure[M], Beijing, National Defence Industry Press, 2001.1

[7] Ych Wenchang, Jen Chcinwci. High-speed booth encoded parallel multiplier design [J].Transactions on Computers, 2000, X9(7):692-701

[8] R. P Brent, H, T. Kung. A Regular Layout for Parallel Adders, liEE Transcations on Computers. 1982.

[9] Huey Ling. High*Speed Binary Adder IBM Journal of Research and Development. 1981.

[10] Li Guoqiang. FT_SIMD: Design of A High-Performance Multiplier Computer Engineering & Science Vol.34, no.1,2012