Prototyping design of a flexible DSP block with pipeline structure for FPGA

Hanyang Xu, Jian Wang, and Jinmei Lai

ASIC and System State Key Laboratory, Fudan University,
825 Zhangheng Road, 201203, Shanghai, China
a) 11110720012@fudan.edu.cn
b) jmlai@fudan.edu.cn

Abstract: Embedded hard DSP block effectively improves FPGA performance for arithmetic circuits. This paper proposes a novel DSP architecture. By adopting a new Compressor Array, the proposed DSP can additionally support multi-operand addition which current commercial devices do not support. This makes the DSP block more versatile to cover a wider range of applications. But supporting multi-operand addition will significantly increase routing congestion. To alleviate timing degeneration caused by the more congestion routing, we implement a pipelined design in the Compressor Array. The proposed DSP block is fabricated in 1P10M 65 nm bulk CMOS process, Test results show a 53.7% reduction in critical path delay compared to the Field Programmable Compressor Tree (FPCT).

Keywords: DSP, compressor, pipeline, FPGA

Classification: Integrated circuits

References

[1] I. Kuon, et al.: “FPGA architecture—survey and challenges,” Foundations and Trends in Electronic Design Automation 2 (2007) 135 (DOI: 10.1561/1000000005).
[2] P. A. Jamieson and J. Rose: “Enhancing the area-efficiency of FPGAs with hard circuits using shadow clusters,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 18 (2010) 1696 (DOI: 10.1109/TVLSI.2009.2026651).
[3] A. Cevrero, et al.: “Field programmable compressor trees acceleration of multi-input addition on FPGAs,” ACM Trans. TRETS 2 (2009) 13 (DOI: 10.1145/1534916.1534923).
[4] S. Mirzaei, et al.: “High speed FIR filter implementation using add and shift method,” Proc. ACM FPGA’06 (2006) 231 (DOI: 10.1145/1117201.1117251).
[5] A. K. Verma and P. Ienne: “Improved use of the carry-save representation for the synthesis of complex arithmetic circuits,” Conf. IEEE ICCAD (2004) 791 (DOI: 10.1109/ICCAD.2004.1382683).
[6] C.-Y. Chen, et al.: “Analysis and architecture design of variable block-size motion estimation for H.264/AVC,” IEEE Trans. Circuits Syst. 53 (2006) 578 (DOI: 10.1109/TCSL.2005.858488).
[7] S. Sriram, et al.: “A 64 channel programmable receiver chip for 3G wireless infrastructure,” Proc. IEEE Custom Integrated Circuits (2005) 59 (DOI: 10.1109/CICC.2005.1568607).
1 Introduction

Field Programmable Gate Arrays (FPGAs) offer many advantages compared to Application Specific Integrated Circuits (ASICs), including prototype verification, reconfiguration capability and reduced non-recurring engineering costs. But the advantages of FPGA comes at a significant cost in area, power consumption, and delay, according to [1], an FPGA requires 20 to 35 times more area and consumes 10 times more dynamic power than an ASIC, but the speed of an FPGA is roughly 3 to 4 times slower than an ASIC. These gaps are tolerable for control-dominated circuits, but unacceptable for arithmetic-dominated circuits. Generally, arithmetic circuits do not map well on lookup table (LUT), which is the primary building block in FPGA. Integrating DSP block into FPGA is an effective way to close the existing gap between FPGA and ASIC.

Since DSP block is very efficient at implementing specific functions, yet go to waste if unused. The primary target to design a DSP block is to improve efficiency. One way to achieve this goal is by adopting shadow clusters [2]. Thus the routing resources can be shared by shadow cluster when DSP block is not used. Another way is to make the DSP block as flexible as possible, thus a much wider range of applications can benefit from the DSP block [3].

Multi-Operand addition occurs in a variety of signal processing applications, such as FIR filters [4]; Arithmetic transformations [5]; motion estimation [6] and correlators used in wireless communication [7]; In commercial FPGA device,
multi-operand addition is realized on general logic resource, which is comprised of LUTs and carry chains [8, 9, 10], so the performance is limited.

There are several academic proposals to improve the performance of multi-operand addition. Reference [3] proposed an alternative block called Field Programmable Compressor Tree (FPCT). FPCT can eliminate much of the routing overhead compared to the compressor tree synthesized on LUTs, so it is very good at implementing multi-operand addition. However, FPCT is less efficient at implementing other functions. For example, when performing an $M \times N$-bit multiplication, only $M + N$ bits signals are required to be routed to a dedicated multiplier, but $M \times N$ bits signals are required to be routed to a FPCT block, this is because the Partial Product Generator (PPG) has to be synthesized on LUTs when implementing multiplication with FPCT. Reference [11] overcomes the disadvantage of FPCT by integrating PPG and CSlices in one block, thus when performing multiplication, the $M \times N$ bits of partial products do not need to go through FPGA’s general routing structure. The disadvantage in [11] is its Baugh-Wooley PPG, the number of partial products in a Baugh-Wooley PPG is equal to the bit width of the multiplicand. Reference [12] adopted a Radix-4 Booth PPG which reduces the number of partial products by half compared to [11], thus reduce the area of Partial Product Reduction (PPR) unit.

In this paper, we introduce a novel DSP architecture supporting multi-operand addition through bypassing the inputs of the Compressor Array. The lower bits of the partial products are extended and aligned to the Least Significant Bit (LSB) of the entire compressor array. By doing this, the compressor array and interconnections between DSP inputs and the multiplexers are still in a very regular form, which makes it easy to insert registers into the array to form a pipelined structure. When switching between multiplication and multi-operand addition, only the selection bit of the multiplexer need to be changed. So the complexity of configuration is significantly reduced. Combined with a Post Single Instruction Multiple Data (SIMD) adder and Control Logic, the proposed DSP block can implement multiplication, multi-operand addition, symmetric rounding, accumulation, multiply accumulation, barrel shifter and counter functions.

2 Architecture

2.1 Pipelined multiplier supporting multi-operand addition

The fundamental building block of our proposed DSP is the pipelined Compressor Array. By multiplexing the input signal of the Compressor Array, the multiplier can be configured into a multi-operand adder.

To fully exploit the advantage of pipeline design, the Compressor Array should be in a very regular form. Vedic Multiplier in [13] is based on basic 4-bit multiplier which has a fine regularity, but less efficient in implementing compressor array. Array Multiplier has a fine regularity in compressor array, and good efficiency in compression, but has too many partial products. Therefore we adopt Radix-4 Booth architecture. The number of partial products is reduced by half compared to Array Multiplier. Meanwhile, Compressor Array in Radix-4 Booth architecture keeps in a very regular form.
The dark circle and dark triangle in Fig. 1 represents partial products generated by booth encoder, to obtain the final multiplication result, all the partial products should be sign extended to the 35th column. Sum of the sign bits in partial-product array can be expressed as Eq. (1).

\[
\text{Sign} = S_0 \cdot \sum_{n=18}^{35} 2^n + S_1 \cdot \sum_{n=20}^{35} 2^n + S_2 \cdot \sum_{n=22}^{35} 2^n + S_3 \cdot \sum_{n=24}^{35} 2^n + S_4 \cdot \sum_{n=26}^{35} 2^n + S_5 \cdot \sum_{n=28}^{35} 2^n + S_6 \cdot \sum_{n=30}^{35} 2^n + S_7 \cdot \sum_{n=32}^{35} 2^n + S_8 \cdot \sum_{n=34}^{35} 2^n
\]

(1)

The extra compressors used to compress the extended sign bits will result in an area and power penalty. We adopt a sign-generate algorithm proposed in [14] to eliminate sign extending. According to [14], the sum of sign extension bits in Eq. (1) can be rewrote as Eq. (2).

\[
\text{Sign} = \sum_{i=0}^{8} S_i \cdot 2^{18+2i} + 2^{18} + 2^{19} + 2^{21} + 2^{23} + 2^{25} + 2^{27} + 2^{29} + 2^{31} + 2^{33} + 2^{35}
\]

(2)

According to Eq. (2), sign extension can be simplified into 2 steps: inverting the sign bit of each partial product, then adding a constant 1010101010101010101 starting at the 18th column. The entire array before compression is shown in Fig. 1.

To perform multiplication, the partial products and corresponding sign extension bits should be compressed. These bits are represented by dark circle, dark triangle, the inverted sign bits and the constant value in 9th row. To perform multi-operand addition, signals in a rectangular region from column 0 to column 18 should be compressed. We insert multiplexers in the array to choose between different signals, by doing this, the array remains in a regular form.

As shown in Fig. 1, there are at most 10 digits to be compressed in one column in the partial product array. According to [15], a (10:2) compressor will have less delay compared to combinations of multi-stage (3:2) compressor, but one stage of (10:2) compressor cannot take advantage of pipeline design. So we adopt a combination of (4:2) compressor and (3:2) compressor in the design of compressor array.

A (4:2) compressor not only compresses 4 bits into 2 bits, it also absorbs an extra carry in signal and generates an extra carry out signal. Considering the situation in Fig. 2(a), if a (4:2) compressor is used to compress data in column n.
Compression result will be 3 bits in column $n+1$ and 1 bit in column $n$ in the next stage. This is because the extra carry out signal cannot be absorbed by another (4:2) compressor in column $n+1$. But if a (3:2) compressor is used in column $n$ as shown in Fig. 2(b), the compression result will be 2 bits in column $n+1$ and 2 bits in column $n$ in the next stage. Compared to Fig. 2(a), result in Fig. 2(b) is more balanced, and delay in Fig. 2(b) will be smaller since (3:2) compressor is used instead of (4:2) compressor. So in the design of the compressor array, selection between (4:2) compressor and (3:2) compressor is based on which one makes the compression results more balanced.

We divide the entire multiplier into 3-stage pipeline. The first stage pipeline contains booth encoding circuits and multiplexers. The second and third stage pipeline contains the compressor array. In Fig. 3, the shadowed ellipse represents (3:2) compressor, and the white ellipse represents (4:2) compressor.

![Fig. 2. Compression results of (4:2) compressor and (3:2) compressor](image)

![Fig. 3. Pipeline division of the multiplier](image)

### 2.2 Post SIMD adder

In order to enhance the flexibility of the DSP block, we adopt a SIMD design in the post adder. As shown in Fig. 4, the SIMD adder supports execution of one 64-bit addition/subtraction, two groups of 32-bit addition/subtraction and four groups of 16-bit addition/subtraction.
Within each 16-Bit Adder, we implement a combination of Kogge-Stone (KS) Tree [16] parallel prefix and Carry Select Adder (CSA). The structure of KS Tree is shown in Fig. 5(a). The black circle in the figure represents the dot operator \([17]\), which is defined in Eq. (3). \(g\) and \(g'\) represent bit generate signals, while \(p\) and \(p'\) represent bit propagate signals as defined in Eq. (4).

\[
(g, p) \odot (g', p') = (g + (p \cdot g'), p \cdot p')
\]

\[
g = A \cdot B \quad p = A \oplus B
\]

A complete KS Tree as shown in Fig. 5(a) can generate all the carry signals from \(C0\) to \(C15\), but the high-density structure results in more interconnections and longer distance between different computation nodes. This significantly deteriorates the performance. To solve this problem, we divide the 16-bits addition into four CSAs, with each CSA computes 4-bits addition. Therefore the KS Tree only needs to generate carry signals every four bits, which makes a sparse tree structure as shown in Fig. 5(b). Another advantage of this approach is: the 4-bits CSA and the KS tree can work in parallel, thus the final result is generated as long as the carry signal is valid from the KS Tree.

![Fig. 4. SIMD structure of the post adder](image)

![Fig. 5. Sparse tree structure based on Kogge-Stone tree](image)

### 2.3 The overall DSP architecture

The entire DSP consists of a pipelined multiplier, a post-SIMD adder and Control Multiplexers (CM) as shown in Fig. 6. The function of the entire DSP is controlled by multiplexer CM1, CM2 and CM3. Through different combinations of signals selected by CM1, CM2 and CM3, the DSP block can implement multiplication, multi-operand addition, symmetric rounding, accumulation, multiply accumulation, barrel shifter and counter functions.

For example, when implementing multi-operand addition, CM1 choose signal MP1, CM2 choose signal MP2 and CM3 choose signal GND. Here MP1 and MP2
are the compression results from the multiplier. Then MP1 and MP2 are added together in the post-SIMD adder to generate the final multi-operand addition result.

3 Experimental results

The proposed design is fabricated in 1P10M 65 nm bulk CMOS process. Fig. 7 shows the layout and test platform.

We use a Built In Self Test (BIST) method to verify the function and performance of the DSP block. A golden model is realized by behavioral description without any specification on circuit structure. A Linear Feedback Shift Register (LFSR) is used to generate pseudo-random test vectors. In each test cycle, the proposed DSP block and the golden model get the same test stimulus from the LFSR. But the clock frequency of the golden model is divided by four through a frequency divider. Thus the operating frequency of the DSP block is four times of the golden model. At the end of each test cycle, output of the proposed DSP block
and the golden model is compared. If the results are equal, test procedure will continue, otherwise test procedure will stop. Test result shows the comparison results are equal until the clock frequency of the proposed DSP is up to 713.16 MHz.

Since multiplication has the maximum critical path delay, the performance of multiplication restricts the overall performance of the DSP block. We compare the critical path delay of different multiplications with FPCT in [3], DSP Block in [12] respectively. Table I shows comparison results between different architectures.

| DSP Features | Ref. [3] | Ref. [12] | Our DSP |
|--------------|----------|-----------|---------|
| m9 × 9       | 2.60     | 1.89      | 1.40    |
| m12 × 12     | 3.60     | 2.43      | 1.40    |
| m18 × 18     | 4.65     | 3.15      | 1.40    |
| m20 × 20     | 5.40     | -         | 1.40    |

Except for multiplication which can be implemented merely on DSP block, we also evaluate benchmarks like fir3, fir6, hpoly and g72x, the implementation of these benchmarks including both DSP blocks and other general logics of FPGA. We cannot achieve the evaluation results by test directly since we only tape out a DSP block. These benchmarks are evaluated in the following way: first we synthesize the design on Xilinx’s Virtex-5 device, the synthesize result is an ncd file. Then we extract timing information of the design from the ncd file by Xilinx’s Timing Analyzer Tool. By doing this, we get the delay value of all the general logics used in the design. The final critical path delay is achieved by adding up the delay of the general logics and the proposed DSP block. We compare the evaluation result with FPCT in [3] and DSP Block in [12]. Table II listed the comparison result. It shows an average 53.7% and 34% reduction in critical path delay compared with the architecture in Reference [3] and Reference [12] respectively. Fig. 8 illustrates the comparison results in histogram.

| DSP Features | Ref. [3] | Ref. [12] | Our DSP |
|--------------|----------|-----------|---------|
| fir3         | 3.76     | 3.21      | 1.49    |
| fir6         | 6.79     | 4.53      | 2.24    |
| hpoly        | 5.55     | 3.36      | 2.78    |
| g72x         | 4.31     | 3.22      | 2.93    |

The FPCT in [3] is realized on Altera’s FPGA device, area of the FPCT is evaluated on the number of Adaptive Logic Modules (ALMs) used instead of the layout area. Therefore we only compare area with Reference [12] which is synthesized on a 90nm CMOS process.
Our DSP block is fabricated using 65 nm process, core area is 30290 um². DSP block in Reference [12] is implemented on 90 nm process, core area is 41439 um². Area comparison result is summarized in Table III.

Since area comparison is unreasonable between different processes, we normalized the area in 90 nm process by multiplying a factor of 65/90. Comparison result shows area of the proposed DSP block is 1.2% larger than the DSP block in [12].

### 4 Conclusion

In this paper, we proposed a new DSP architecture for FPGAs. By modifying compressor arrays, the proposed DSP block supports multi-operand additions, which is not supported by commercial devices currently. This improves the efficiency of the DSP block since more applications can be fitted into it. The final design is fabricated in 65 nm 1P10M bulk CMOS process. Test results show an average 53.7% and 34% reduction in critical path delay compared with FPCT in [3] and DSP block in [12] respectively. While area of the proposed DSP is 1.2% larger compared to the DSP block in [12].

### Acknowledgments

This work is supported by State Key Laboratory of ASIC and System of Fudan University funded project, No. 2015MS007.