Noname manuscript No.
(will be inserted by the editor)

Efficient Implementation of Complex Multipliers on FPGAs Using DSP Slices

Pedro Paz · Mario Garrido

Received: date / Accepted: date

Abstract In this paper, we propose two efficient implementations of complex multipliers on field-programmable gate arrays (FPGAs) using DSP slices. The first implementation aims for high throughput and the second one for low area. By mapping these circuits to the DSP slices in the FPGA, the proposed implementations have the advantage that they only require three DSP slices. Experimental results show that the proposed high-throughput implementation saves hardware resources with respect to previous approaches, while reaching the highest achievable clock frequency. Alternatively, the proposed low-area implementation reduces the amount of hardware resources even further at the cost of reducing the clock frequency.

Keywords Complex multiplier, FPGA, rotator, FFT, DSP slice

This work was supported in part by Comunidad de Madrid through the call "Ayudas de Estímulo a la Investigación de Jóvenes Doctores de la Universidad Politécnica de Madrid" under Project APOYO-JOVENES-21-TL23SB-116-I4FOMC; and in part by MCIN/AEI/10.13039/501100011033 and "ESF Investing in your future" under Grant RYC2018-025384-I.

P. Paz
Dept. of Electronic Engineering
Universidad Politécnica de Madrid
Madrid, Spain
E-mail: p.pazm@alumnos.upm.es

M. Garrido
Dept. of Electronic Engineering
Universidad Politécnica de Madrid
Madrid, Spain
E-mail: mario.garrido@upm.es

1 Introduction

The multiplication of two complex numbers is an elementary operation, commonly used in digital signal processing algorithms such as the fast Fourier transform (FFT) [10,13]. Large and highly parallel FFT architectures require hundreds of such multipliers. Thus, optimizing a complex multiplier provides significant savings when the improvement is applied to all the complex multipliers in the architecture.

The direct computation of a complex multiplication requires four real multiplications and two real additions [10,11,13,14,18]. However, the number of real multiplications can be reduced to three by rearranging the operations, at the cost of increasing the number of additions to five [4,7,8,12,14–16]. Several architectures based on this approach are presented in [16] and have been used in Xilinx IP cores [4]. In [12], these architectures are used as an alternative to implement multiplierless rotators. Further optimization of these architectures apply adder compression to simplify the circuit [8]. Finally, a detailed comparison of the direct computation with respect to the three-multipliers approach in terms of area, power, throughput, and latency is provided in [15].

Nowadays, it is common to use field-programmable gate arrays (FPGAs) for the development of final products. However, implementations do not always take full advantage of the hardware resources of the FPGA. A better utilization of the FPGA leads to more efficient designs in terms of area, throughput, and power consumption.

On FPGAs, the real multipliers and adders in a complex multiplier can be implemented by using logic blocks. However, FPGAs usually contain dedicated modules to compute arithmetic operations. In Xilinx FPGAs, these modules are called DSP slices [2,5] and allow higher clock frequency. 7 Series FPGAs from Xilinx [3] contain a block named DSP48E1 [2]. Ultrascale and Ultrascale+ FPGAs from Xilinx [6] contain a block named DSP48E2 [5]. This paper presents two efficient implementations of a complex multiplier, suitable for DSP48E1 and DSP48E2 slices. The first implementation targets maximum throughput and low area, and achieves the highest clock frequency allowed by the device, whereas the second implementation aims for a balance between area and performance.

This paper is organized as follows. In Section II, we review previous structures to reduce the number of real multipliers in a complex multiplication, as well as the slices available in Xilinx 7 Series FPGAs and the DSP architecture. In Section III, we present the proposed complex multipliers. In Section IV, implementation results and comparison with the state-of-the-art are provided. Finally, the main conclusions of this work are summarized in Section V.

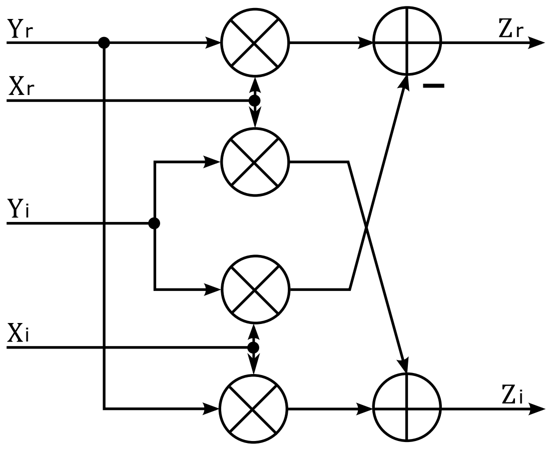

Fig. 1: Direct implementation of a general complex multiplier using four real multipliers and two real adders.

2 Background

2.1 Structures to compute complex multiplications

The multiplication of two complex numbers $(X_r + jX_i)$ and $(Y_r + jY_i)$ is defined as

$$\begin{aligned} Z_r &= X_r Y_r - X_i Y_i, \\ Z_i &= X_r Y_i + Y_r X_i, \end{aligned} \tag{1}$$

where Z_r is the real part of the result and Z_i is its imaginary part. Based on (1), Fig. 1 shows a direct implementation of a complex multiplier, which requires four real multipliers and two real adders. This implementation is common in the literature [10,11,13,18]. Note that the complexity of adders and subtractors is the same. Based on this, we will refer to all of them as adders throughout the paper.

Instead of using four real multipliers, a structure with three real multipliers [16] can be obtained by rewriting (1) as

$$\begin{aligned} Z_r &= X_r \cdot (Y_r - Y_i) + Y_i \cdot (X_r - X_i), \\ Z_i &= X_i \cdot (Y_r + Y_i) + Y_i \cdot (X_r - X_i). \end{aligned} \tag{2}$$

It can be observed that the term $Y_i \cdot (X_r - X_y)$ is shared for the calculations of Z_r and Z_i, so it only needs to be computed once. Fig. 2 shows the implementation of a complex multiplier based on (2). Apart from three multipliers, this structure uses five adders instead of the two adders needed in the direct implementation. This leads to area reduction, since multipliers require significantly more area than adders. The area of a real multiplier can be estimated as *WL* adders, where *WL* is the word length of the inputs [17], e.g., for a word length of 16 bits, the area of a multiplier is comparable to the area of 16 adders.

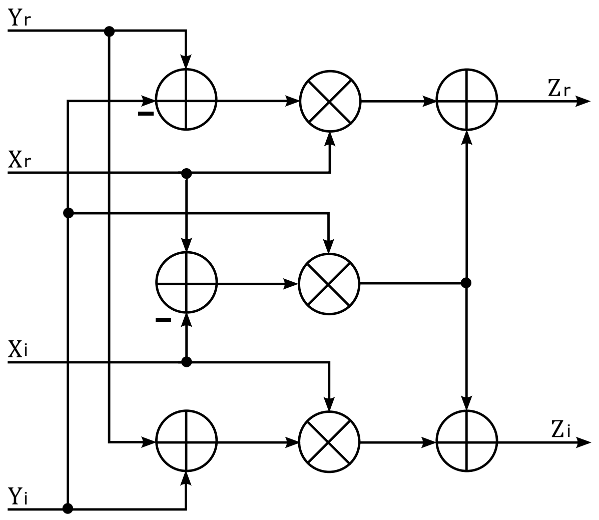

Fig. 2: Implementation of a complex multiplier using three real multipliers and five real adders.

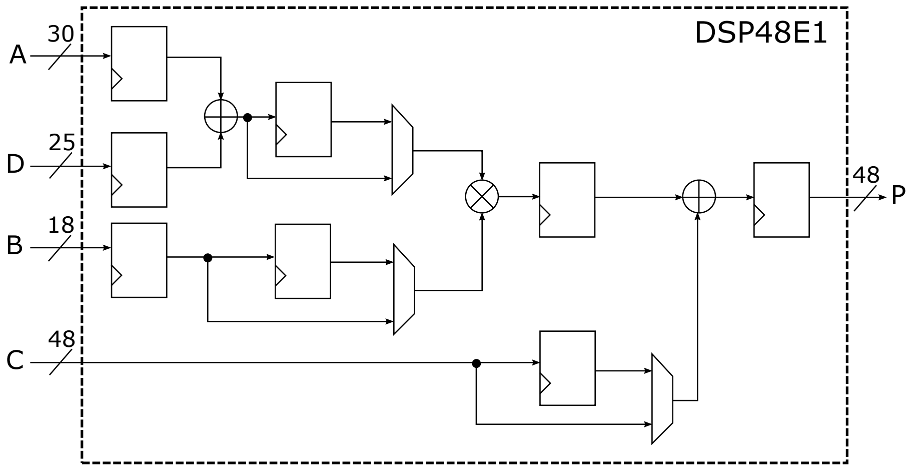

Fig. 3: Simplified scheme of DSP48E1.

2.2 Structure of a slice by Xilinx

Slices are the most basic elements in FPGAs. They are used to implement logic functions and, in general, any digital circuit based on logic gates and registers. In Xilinx 7 Series FPGAs, each slice contains four look-up tables (LUT), eight 1-bit registers and several multiplexers to configure the interconnections inside each slice [1].

There are two types of slices, known as SLICEL and SLICEM, which differ in the capabilities of the LUTs. Besides implementing logic functions, each of the four LUTs in the SLICEM can be configured as 32x1 bits distributed random-access memory (RAM) or a 2-bit shift register. The delay of these shift registers can be set from 1 to 32 clock cycles. The SLICEM configured as shift register is specially useful when a signal needs to be registered for several clock cycles in order to balance the pipeline delays between paths of a design.

In Xilinx Ultrascale and Ultrascale+ FPGAs by Xilinx, the slices are replaced by CLB slices, which contain 8 LUTs and 16 registers, instead of four LUTs and 8 registers as reviwed for Xilinx 7 Series slices. CLB slices can also be of the type SLICEM or SLICEL.

2.3 Structure of DSP48E1 and DSP48E2 slices by Xilinx

The DSP48E1 slice [2] is the module dedicated to compute arithmetic operations in Xilinx 7 Series FPGAs [3]. Ultrascale and Ultrascale+ FPGAs from Xilinx [6] containt the module named DSP48E2 [5]. The only difference between the two models relevant to this work is the width of the D port. Fig. 3 shows a simplified scheme of the DSP48E1 slice with the sub-modules relevant to this work. The DSP slice has four inputs (A, B, C and D) and one output (P). The DSP slice can compute operations of the type:

$$P = \pm[(D \pm A) \cdot B] \pm C, \tag{3}$$

i.e., one addition or subtraction followed by a multiplication and another addition or subtraction.

The paths from inputs A, B, and D to P have four pipeline registers, whereas the path from C to P has two pipeline registers. The pipeline registers can be bypassed using an alternative path. Bypassing the registers reduces the latency in terms of clock cycles, but also increases the propagation delay, leading to a lower clock frequency.

3 Proposed complex multipliers

Digital design is usually based on dealing with a trade-off between performance and area. For this reason, two complex multipliers are proposed in this paper. The first one aims for high throughput and targets applications where maximum performance is sought. The second one aims for low area and targets applications where area and power consumption are critical. Both complex multipliers use DSP48E1 or DSP48E2 slices to compute the real multiplications and additions. Also, both approaches are based on the structure with three multipliers and five adders described in Section 2.1.

The proposed approach allows for complex multipliers with word length of up to 18 bits in fixed-point 2's complement format. The limit of 18 bits is set by the word length of the B input in the DSP48E1 and DSP48E2. However, in the proposed implementations we have chosen a word length of 16 bits due to two facts. First, 16 bits is a typical word length used in many designs [8, 9, 15, 18]. Second, each slice in the FPGA has eight 1-bit storage elements [1]. As 16 is multiple of 8, using 16-bit words leads to more efficient designs in terms of slice utilization. Furthermore, in the proposed implementations, no quantization of the internal signals neither the output values is carried out. After presenting the proposed complex multipliers in Sections 3.1 and 3.2, Section 3.3 discusses several considerations related to the word length of the designs.

3.1 High-throughput complex multiplier

Fig. 4 represents the scheme of the proposed complex multiplier for maximum throughput. This complex multiplier requires three DSP slices and 8 additional slices to balance pipeline paths. In Fig. 4, each DSP slice is labeled with DSP48E1/2, since they can be implemented either with DSP48E1 or DSP48E2 slices, and the identifier UPPER, MIDDLE, or LOWER. Each DSP is delimited with a dotted line rectangle. Above each adder and multiplier, the operation that they compute is displayed. Slices are also limited with dotted line rectangles. They are labeled with the word SLICES and two numbers that identify the slices. Labels of the form tn inside registers indicate the time n when data arrives to these registers relative to the arrival of the inputs.

In Fig. 4, the UPPER DSP slice computes the term $Y_i \cdot (X_r - X_y)$ in (2), i.e., an addition followed by a multiplication. Since the last adder in the DSP slice can not be bypassed, zero is added to the previous result. To achieve maximum clock frequency, every pipeline stage in the DSP is used, meaning that this computation requires 4 clock cycles. As the term $Y_i \cdot (X_r - X_y)$ is shared between the real and imaginary parts, this partial result needs to be fed to the two other DSP slices. The MIDDLE DSP slice computes the expression $X_r \cdot (Y_r - Y_i)$ in the first adder and the multiplier. The final adder adds this expression with the result from the UPPER DSP, which provides the real part of the complex multiplication. Likewise, in the LOWER DSP slice, the term $X_i \cdot (Y_r - Y_i)$ is computed and then added to the result from the UPPER DSP slice, which leads to the imaginary part of the complex multiplication.

In order to achieve maximum clock frequency, the result from the UPPER DSP slice needs to be registered, both at the output of the UPPER DSP slice and at the input of the other two DSP slices. Otherwise, the maximum clock frequency achievable is decreased due to the delay introduced by the interconnection of the DSP slices.

To compute the complex multiplication correctly, the MIDDLE and LOWER DSP slices need the result from the UPPER one three clock cycles after the data are fed to these DSPs. The result of the UPPER DSP is delivered to the second adder in the MIDDLE and LOWER DSPs five clock cycles after the input data are delivered to the UPPER DSP slice, i.e., four clock cycles from the UPPER DSP latency plus an additional cycle from the input register in the C to P path of the MIDDLE and LOWER DSP slices.

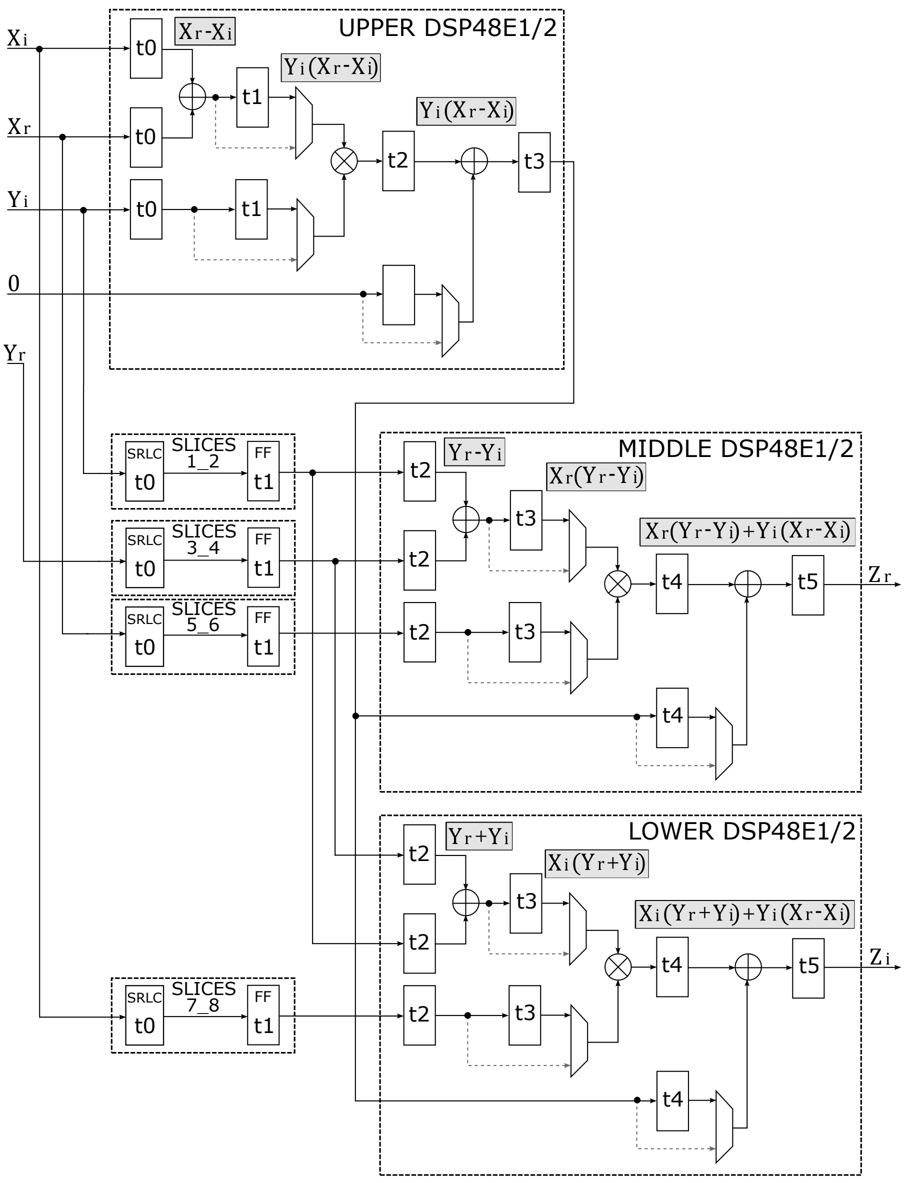

Fig. 4: Proposed complex multiplier using 3 DSP slices for high throughput.

To balance the pipeline, two additional registers are needed in each of the inputs of the MIDDLE and LOWER DSP slices. The registers for the data Y_r and Y_i can be shared between the two DSPs, meaning that four two-cycle delays are required. These registers can be implemented with the memory elements available in the slices. Since the data word length is 16 bits and each slice contains eight 1-bit registers, two slices are necessary to implement each register. Since four signals need to be registered for two clock cycles, a total of 16 slices would be required. Using the SLICEM configured as shift registers, as described in Section 2.2, the number of slices required is halved. In this implementation, we configure each LUT as a shift register for a single clock cycle delay, which is denoted as SRLC in Fig. 4. The additional 8 registers in the SLICEM are labeled as FF in Fig. 4. This allows us to generate the 2-cycle delay over 8 bits in a single slice. By applying this efficient mapping technique, the high-throughput complex multiplier only requires three DSP slices and 8 slices.

3.2 Low-area complex multiplier

The high-throughput complex multiplier proposed in previous section is designed to be able to work at the highest clock frequency achievable by the DSP slices.

However, maximum throughput might not be the main goal in certain applications. Often, designs look for a trade-off between throughput, area, power consumption and latency. For this reason, we propose an additional complex multiplier implementation, where only 3 DSP slices are required.

This complex multiplier is shown in Fig. 5 and it uses the same format as that in Fig. 4. Additionally, in Fig. 5 the registers in grey without a time identifier indicate that the registers are bypassed.

The MIDDLE DSP in Fig. 5 computes the shared term between the real and imaginary parts of the complex multiplication, i.e., $Y_i \cdot (X_r - X_i)$. To reduce the latency in this computation, the pipeline stage between the first adder and the multiplier is bypassed. Thus, this term is calculated in three clock cycles and fed to the UPPER and LOWER DSP slices. These two DSP slices use every pipeline stage, except for the input register in the path from C to P in Fig. 3. Thereby, the partial result from the MIDDLE DSP can be added in the UPPER and LOWER DSP slices three clock cycles after the data are fed to the complex multiplier without additional registers to balance pipeline. However, since several pipeline registers are bypassed, the maximum achievable frequency is reduced.

3.3 Word length considerations

The proposed implementations receive inputs represented in 2's complement with 16 bits for the real part of the data and 16 bits for the imaginary one. Thus, input values are in the range $[-2^{15}, 2^{15} - 1]$. Considering the direct implementation with 4 DSPs and 2 adders in Fig. 1, the range of numbers after the multiplication is $[-2^{30} + 2^{15}, 2^{30}]$. Then, two such numbers are added together, leading to the output range $[-2^{31} + 2^{16}, 2^{31}]$. This result requires 33 bits, as it is inside the range $[-2^{32}, 2^{32} - 1]$. Note that 32 bits can not cover the output range completely, because the highest possible output value that

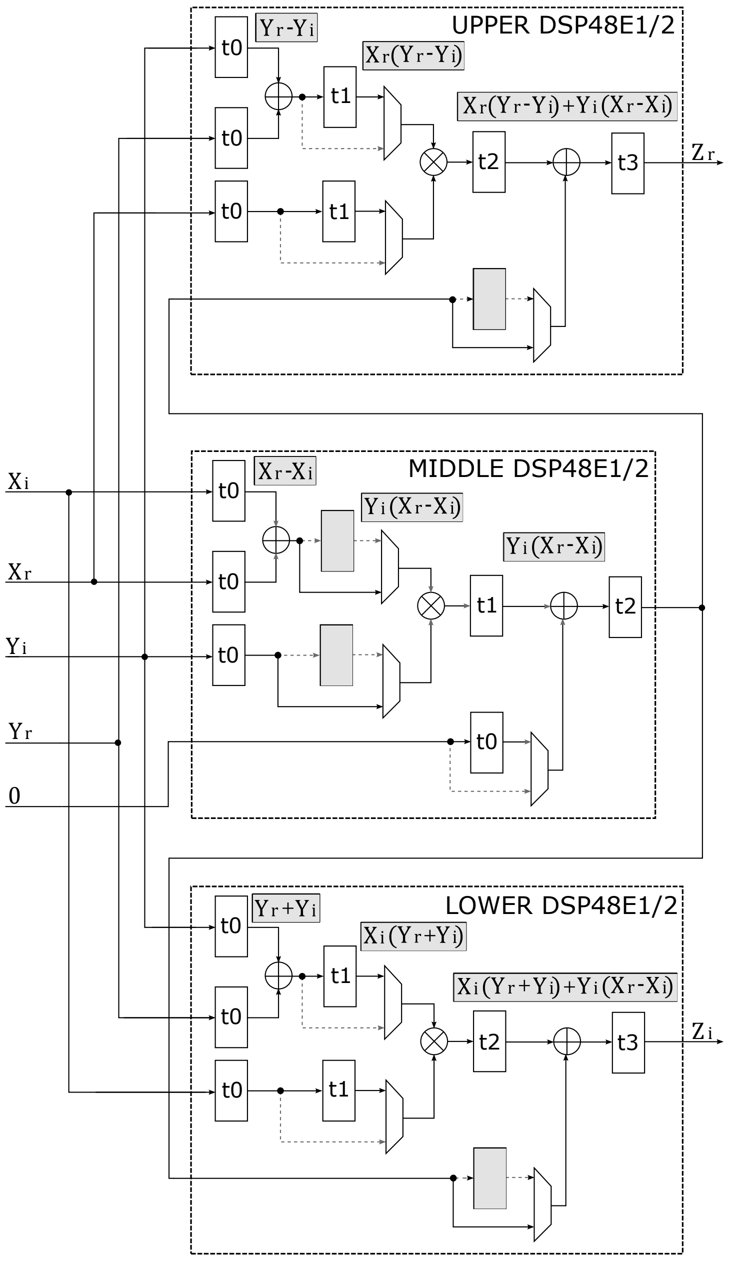

Fig. 5: Proposed complex multiplier using 3 DSP slices for low area utilization.

can be represented with 32 bits is $2^{31} - 1$ and the output range includes the value 2^{31}.

Table 1: Comparison of complex multipliers on FPGA

Parameters	Rao [14]	Rao [14]	Du [7]	Xilinx DL [4]	Xilinx DSP [4]	Direct Imp.	Proposed HT	Proposed LA
Approach	DL	DL	DL	DL	DSP	DSP	DSP	DSP
Real Multipliers	4	3	3	3	3	4	3	3
Input Word Length	32	32	16	16	16	16	16	16
Approximated	No	No	Yes	No	No	No	No	No
Device	V7	V7	A7	V7	V7	V7	V7	V7
f_{max} (MHz)	-	-	198	470	740	740	740	465
Latency (ns)	25.9	27.5	-	12.7	8.1	5.4	8.1	8.6
Latency (cycles)	-	-	-	6	6	4	6	4
DSP Slices	0	0	0	0	3	4	3	3
Slices	-	-	-	289	16	0	8	0
LUTs	10416	8096	836	999	32	0	32	0
FFs	-	-	128	938	64	0	64	0
Power (mW)	187	128	-	175	40	30	37	17

The proposed complex multipliers do not apply any quantization in the calculations. Therefore, the number that results from the complex multiplication is exactly the same as in the case of the direct implementation. As a consequence, the range of output values is also $[-2^{31} + 2^{16}, 2^{31}]$, which is covered with 33 bits.

In certain applications, the inputs of the complex multiplier have properties that guarantee that the most negative value of the 2's complement representation of the inputs will not appear in all four inputs simultaneously. In this case, the outputs will never reach the value 2^{31} and 32 bits are enough to represent the output. This occurs, for instance, when one of the complex inputs receives constant coefficients and we know in advance that no coefficient takes the most negative value for the real and imaginary components simultaneously.

4 Implementation results

Table 1 compares different complex multipliers implemented on FPGAs. The approaches are classified into those that used only distributed logic (DL) and those that use DSP slices (DSP).

The proposed high-throughput (HT) and low-area (LA) complex multipliers have been implemented on a Xilinx Virtex 7 XC7VX330TFFG1157-3 FPGA. In these implementations, the Xilinx DSP48E1 and SRL32 primitives for the DSP slices and the shift registers, respectively, have been instantiated manually, and the results have been obtained using Vivado 2020.2.

Among previous works in Table 1, the approach in [14] includes two implementations of a 32×32-bit multiplier on a Virtex 7 using 4 and 3 real multipliers, respectively. The approach in [7] calculates an approximated complex multiplier using a configuration with 3 real multipliers. The comparison also includes two Xilinx IPs that use 3 real multipliers to calculate the complex multiplications. One of them is implemented in distributed logic and the other one uses DSP slices. Finally, a direct implementation of the complex multiplier according to Fig. 1 is provided. All the results are for Virtex 7 (V7) except for [7], which provides experimental results for an Artix 7 (A7).

By comparing approaches that use distributed logic and those that use DSP slices, it can be observed that the latter allow for higher clock frequency and lower power consumption. This higher clock frequency results in lower latency in ns. Regarding area, [7] and the Xilinx DL implementations require 836 and 999 LUTs, respectively. Compared to approaches that use DSP slices, it can be observed that each DSP slice saves approximately 200 to 300 slices.

By comparing approaches that use DSP slices, all of them use three DSP slices, except for the direct implementation, which uses four. The direct implementation does not require additional slices, and obtains the maximum clock frequency, which is 740 MHz, at the minimum latency. The Xilinx DSP and the proposed high-throughput complex multiplier also achieve the maximum clock frequency. These implementations require a few additional slices, but reduce the number of DSP slices by one. Compared to the Xilinx DSP, the proposed high-throughput complex multiplier halves the number of slices, as it takes advantage of the shift registers available in the SLICEM. In fact, it improves or has the same value in all figures of merit. The low-area complex multiplier obtains the minimum number of slices, DSP slices, latency and power consumption and achieves a clock frequency of 465 MHz. The savings in power consumption with respect to the other approaches based on DSPs range from 43.3% to 57.5%. This is due to the facts that the low-area complex multiplier only requires three DSP slices and no slice, and its clock frequency is lower.

As a result, the proposed implementations together with the direct implementation are optimized solutions for three different scenarios. The direct implementation is preferable when high throughput and low latency is required. The high-throughput implementation is suitable for high throughput and low area requirements. Finally, the low area implementation is preferable for low latency and low area requirements.

5 Conclusions

In this paper, two efficient implementations for complex multipliers on FPGA using DSP slices have been presented. The proposed high-throughput complex multiplier obtains a clock frequency of 740 MHz, which is the highest achievable frequency. This complex multiplier only requires three DSP slices and eight slices, which reduces the area compared to equivalent implementations in terms of throughput. The proposed low-area complex multiplier achieves the minimum latency and power consumption. It uses only three DSP slices and no slices, and achieves a clock frequency of 465 MHz. Together with the direct implementation, the proposed designs reusult in optimized solutions to cover different scenarios.

Data availability: Data sharing not applicable to this article as no datasets were generated or analysed during the current study.

References

1. Xilinx 7 series FPGAs configurable logic block user guide (2016). Https://www.xilinx.com/support/documentation/user_guides/ ug474_7Series_CLB.pdf
2. Xilinx - 7 Series DSP48E1 Slice (2018). Https://www.xilinx.com/ support/documentation/user_guides/ug479_7Series_DSP48E1.pdf
3. Xilinx 7 series FPGAs data sheet (2020). Https://www.xilinx.com/ support/documentation/data_sheets/ds180_7Series_Overview.pdf
4. Xilinx - Complex multiplier v6.0 (2021). Https://docs.xilinx.com/ v/u/en-US/pg104-cmpy
5. Xilinx Ultrascale architecture DSP Slice (2021). Https://docs.xilinx.com/v/u/en-US/ug579-ultrascale-dsp
6. Xilinx Ultrascale architecture and product data sheet: Overview (2022). Https://docs.xilinx.com/v/u/en-US/ds890-ultrascale-overview
7. Du, J., Chen, K., Yin, P., Yan, C., Liu, W.: Design of an approximate FFT processor based on approximate complex multipliers. In: Proc. IEEE Comp. Soc. Annual Symp. VLSI, pp. 308–313 (2021)
8. Fonseca, M.B., Martins, J.B.S., da Costa, E.A.C.: Design of pipelined butterflies from radix-2 FFT with decimation in time algorithm using efficient adder compressors. In: Proc. IEEE Latin American Symp. Circuits Syst., pp. 1–4 (2011)
9. Garrido, M., Malagón, P.: The constant multiplier FFT. IEEE Trans. Circuits Syst. I **68**(1), 322–335 (2021)
10. Garrido, M., Qureshi, F., Takala, J., Gustafsson, O.: Hardware architectures for the fast Fourier transform. In: S.S. Bhattacharyya, E.F. Deprettere, R. Leupers, J. Takala (eds.) Handbook of Signal Processing Systems, third edn. Springer (2019)
11. Ingemarsson, C., Källström, P., Qureshi, F., Gustafsson, O.: Efficient FPGA mapping of pipeline SDF FFT cores. IEEE Trans. VLSI Syst. **25**(9), 2486–2497 (2017)
12. Macleod, M.D.: Multiplierless implementation of rotators and FFTs. EURASIP J. Adv. Signal Process. **2005**(17), 2903–2910 (2005)
13. Qureshi, F., Takala, J., Bhattacharyya, S.: Rotators in fast Fourier transforms. In: S.S. Bhattacharyya, M. Potkonjak, S. Velipasalar (eds.) Embedded, Cyber-Physical, and IoT Systems: Essays Dedicated to Marilyn Wolf on the Occasion of Her 60th Birthday. Springer International Publishing (2020)
14. Rao, K.D., Gangadhar, C., Korrai, P.K.: FPGA implementation of complex multiplier using minimum delay Vedic real multiplier architecture. In: Proc. IEEE Uttar Pradesh Section Int. Conf. Electr. Comp. Electron. Eng., pp. 580–584 (2016)
15. Takala, J., Punkka, K.: Scalable FFT processors and pipelined butterfly units. J. VLSI Signal Process. Syst. Signal Image Video Tech. **43**(2), 113–123 (2006)
16. Wenzler, A., Luder, E.: New structures for complex multipliers and their noise analysis. In: Proc. IEEE Int. Symp. Circuits Syst., vol. 2, pp. 1432–1435 (1995)
17. Yang, C.H., Yu, T.H., Markovic, D.: Power and area minimization of reconfigurable FFT processors: A 3GPP-LTE example. IEEE J. Solid-State Circuits **47**(3), 757–768 (2012)
18. Žádník, J., Takala, J.: Low-power programmable processor for fast Fourier transform based on transport triggered architecture. In: Proc. IEEE Int. Conf. Acoust. Speech Signal Process., pp. 1423–1427 (2019)