Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2021) 29: 3150 – 3164

doi:10.3906/elk-2101-56

HC-FFT: highly configurable and efficient FFT implementation on FPGA

Pakize ERGÜL[1,2,*], **H. Fatih UĞURDAĞ**[1], **Doğancan DAVUTOĞLU**[2]
[1]Department of Electrical and Electronics Engineering, Faculty of Engineering, Özyeğin University, İstanbul, Turkey
[2]TÜBİTAK BİLGEM, Kocaeli, Turkey

Received: 13.01.2021 • **Accepted/Published Online:** 12.06.2021 • **Final Version:** 30.11.2021

Abstract: FFT is one of the basic building blocks in many applications such as sensors, radars, communications. For some applications, e.g., real-time spectral monitoring and analysis, FFT needs to be "run-time configurable" so that the system is real-time. When examining the previous work on configurable real-time (FPGA-based) FFT implementations, we see that the degree of configurability is less than what is desired. In this paper, a new FFT architecture is proposed, which has a high degree of run-time configurability and yet does not compromise area or throughput. The configurable parameters of this design are the number of FFT points (up to 64K), forward versus inverse mode, output order (natural or bit-reversed), and the number of streams (up to 4). The proposed FFT architecture (HC-FFT) is designed using a parallel and pipelined radix-2 multipath delay commutator (MDC) FFT structure. HC-FFT was implemented on a Xilinx Kintex Ultrascale FPGA and was verified against the Xilinx FFT IP. Besides its high degree of run-time configurability, HC-FFT is quite efficient and offers a very high throughput of 87 Gbps with a quite reasonable area.

Key words: Fast Fourier transform, run-time configurable FFT, multipath delay commutator

1. Introduction

Fast Fourier transform (FFT) is an algorithm known to everybody and is used in a myriad of signal processing applications. There are many software libraries for a variety of languages/tools that implement FFT. It is a complicated function and has many parameters. The software implementations usually support many options for many parameters.

This paper is on the hardware (FPGA) realization of FFT. Hardware design is not only about satisfying functionality but also about meeting area, timing, and power consumption requirements. Hardware design of functions such as FFT is much more difficult than software design due to hardware requirements. Therefore, not only hardware library components (a.k.a., intellectually property or IP) are hard to find for FFT but also they support few options for few parameters. Even this by itself is a good enough reason for hardware engineers to design their own FFT. If we cannot foresee the exact parameter values ahead of time, it is best to design an FFT IP Generator rather than a fixed design instance with fixed parameter choices. An FFT IP Generator is a program written in a regular programming language and takes in the selected parameter values and produces a fixed design in a hardware description language.

The above scenario proposes a compile-time configurable FFT. However, sometimes we need run-time configurability. That is when we need to compute FFT with different parameter options during a single run of the device that needs to compute FFT. On FPGAs, this can be achieved by feeding a different bitstream on

*Correspondence: pakize.ergul@tubitak.gov.tr

to the FPGA as need arises. That is acceptable if all possible designs for all possible parameter combinations are precompiled, and if we are allowed switch times in the order of milliseconds. (Also, we need either a host computer or a large enough Flash memory.) If there are too many parameter combinations or the switch time has to be in the order of micro or nanoseconds, then it means that we are in need of true run-time configurability. That is, we need a single design (i.e. single FPGA bitstream) that supports all those parameter combinations we may use. An FPGA-based run-time configurable FFT IP that supports many parameters has not been sufficiently addressed in the literature or the commercial world. That is what this work targets. As in any hardware design work, our goal was to also come up with an efficient hardware architecture, i.e. efficient in terms of area, throughput, and latency.

The most pressing configuration parameters in an FFT design are: (i) size, (ii) mode, (iii) output order, and (iv) number of streams.

FFT size needs to be configurable in, for instance, flexible communication devices that support multiple protocols or multiple data rates. Each of these may require a different FFT size. Flexibility may be in the form of adaptability to different channel characteristics in a portable device. Spectral analysis devices, for example, have to flexible and need to support various FFT sizes [22].

FFT mode, whether we need regular FFT or inverse FFT (IFFT) depends on the signal processing problem at hand. A communications transceiver needs both, hence no need for configurable mode. On the other hand, a receiver usually needs regular FFT, while a transmitter usually needs IFFT. Therefore, if our device is required to be configured as a receiver-only or transmitter-only, it needs its FFT mode to be configurable. Obviously, such examples can be given for other usecases such as filtering [24].

Output order is a challenging issue in FFT. According to Cooley–Tukey FFT algorithm [6], reordering is required to arrange the input or output order depending on the chosen decimation type. For decimation in frequency (DIF) FFT architectures, the input can be applied directly in a natural order. Despite this, the output is obtained in bit-reversed order. On the other hand, decimation in time (DIT) FFT architectures require bit-reversed order for input data, while the output data is in a natural order.

Supporting multiple data streams (i.e. concurrent input signals) from multiple sources is a desired feature for FFT blocks as is the case in multiple-input multiple-output (MIMO) systems [3, 5]. We can always support multiple streams by using as many FFT blocks as the number of streams. However, that is too costly in terms of area and power.

The rest of the paper is organized as follows. Section 2 discusses the literature. Section 3 introduces the basics of FFT. Section 4 explains our proposed design architecture, HC-FFT. While in Section 5, implementation results are provided, Section 6 concludes the paper.

2. Previous work

There is a plethora of work in the literature on FFT hardware. Since our contribution to the literature is in run-time configurable FFT hardware, we will limit ourselves to such work in this section. We start by listing and comparing them in Table 1 with our work, HC-FFT.

In Table 1, size is the supported FFT size in tun-time configuration. Max size shows supported maximum FFT size for related designs. Mode shows the supported FFT calculation mode as inverse (I) and forward (F). In this column, I/F is used for a design that supports inverse and forward modes together. On the other hand, I or F is used the design for support only one of these modes. This kind of designs is only configurable at

design time for this mode. Therefore, it can be said that I/F is a much better design than I or F in terms of configurability. Out. order is the FFT output order and is divided into natural order (N) and bit-reversed order (B). In this column, N/B represents design feature that supports natural and bit-reversed output order together. #Stream is the number of streams, which are handled in parallel in FFT structure.

Table 1. Configurable parameters and the works in the literature.

Design	Size	Max size	Mode	Out. order	#Streams
HC-FFT	13	64K	I/F	N/B	1, 2, 4
Xilinx	14	64K	I or F	N	4
[13]	14	128K	I/F	B	1
[27]	4	1K	F	N	1
[28]	5	2K	F	N	1
[29]	8	2K	F	N	1
[30]	4	4K	F	-	1
[31]	1	2K	F	N	2
[32]	4	4K	F	N	1
[33]	7	32K	F	B	1
[20]	4	1K	I/F	N	1
[15]	5	2K	F	N	4
[12]	12	32K	F	N	1
[17]	7	512	F	N	1
[19]	4	1K	F	N	1
[16]	10	1024K	F	-	1
[14]	8	8K	F	-	1
[18]	5	2K	F	N	1

While HC-FFT has configurability in all of the 4 dimensions in Table 1, the competition has configurability in at most 2 dimensions. The works in [13, 20] have configurable size and mode. Although all other works have only configurable size, [15] draws attention as it is a multistream solution. [13] stands out compared to HC-FFT by supporting 14 different sizes. Now, we will go over the works in Table 1 one by one. In [27], radix-4 MDC decimation in frequency (DIF) parallel pipeline FFT processor is presented for a very high-speed orthogonal frequency-division multiplexing (OFDM) communication systems. Such FFT processor supports the computation of one stream of variable-length FFT with 16, 64, 256, and 1024 pts. The design produces outputs in the natural order. In addition to that, a new reordering block design is presented, which allows easy generalization for different parallelization degree and radix. In [28], a new class of FFT architecture is proposed that combines the flexibility and programmability of memory-based designs with the high throughputs available from array-based hardware. In the paper, according to different systems requirements, 7 different FFT architectures are presented. Variable-length FFT processor is given in the third architecture, which supports 128, 256, 512, 1024, and 2048-point FFT calculation with one stream and natural output order for 802.11ax and LTE protocols. In [29], a scalable and run-time configurable FFT processor is proposed which allow 8 different size from 16 pt to 2048 pts. The main target of the design is OFDM based communication systems. The design in [29] can handle one stream FFT calculation and produce natural ordered outputs. In [30], a

reconfigurable multi-precision FFT block is proposed for coherent optical OFDM (CO-OFDM) systems. This FFT architecture can be reconfigured dynamically and supports 64, 256, 1024, and 4096-point FFT. In [31], a novel FFT processor that processes two independent data streams simultaneously is proposed for various high-speed real-time applications. This processor generates outputs in natural order. [31] shows that the architecture can be extended for different FFT sizes, although the size cannot be configured at run-time. A high-performance and resource-efficient FPGA implementation is proposed in [32]. This design has been developed for applications with high performance needs such as OFDM systems. The proposed FFT processor can handle one stream data and generate output in natural order. In [32], the architecture is extented for 512/1024/2048 and 4096-point FFT. In [33], a reconfigurable and variable-size FFT architecture is proposed. In this work, the multimode synthetic-aperture radar (SAR) imaging professing applications is targeted. The proposed FFT processor supports 7 different FFT sizes from 512-point to 32768-point. It handles one stream natural order (N) inputs and produces the outputs in bit-reversed order (B).

In [13], a large FFT processor supports 14 different sizes, namely, from 16 to 128K (K = 1024) points (pts), all powers of 2 (also applies to the rest of the previous works). The main target of this work is synthetic aperture radar (SAR) and sensor signal processing systems. The design in [13] supports IFFT as well. It produces the outputs in bit-reversed order (B) and can handle only 1 stream. In [20], a configurable size and precision FFT processor is proposed. The architecture is based on single path delay feedback (SDF) pipeline architecture. It mainly targets WiFi, Wimax, and MIMO-OFDM applications. It supports 4 different sizes up to 1K, IFFT, natural order output (N), and 1 stream. In [15], a design with configurable size and 4 streams is proposed. The design mainly targets LTE, Wimax, and MIMO-OFDM domains. Special attention is given to area minimization. It supports 5 different sizes up to 2K, and natural order output (N). The designs below all support natural order output (N) and 1 stream except [14, 16]. Those two do not report the output order. In [12], a configurable size design is presented. The design mainly targets SAR and OFDM domains. It supports 12 different sizes up to 32K. Also, they claim to support an efficient data scaling technique. This work gives special attention to area minimization. The design in [17] supports 7 different sizes up to 512 and mainly targets OFDM based applications. It uses radix-4 common factor algorithm and SDF architecture to reduce the number of twiddle factors and (see Section 3 for definition) complexity. The work in [19] supports 4 different sizes up to 8K and targets OFDM based standards such as DVB, DAB, and ADSL. It uses SDF architecture and a mixed-radix approach. A dynamic scaling approach is adopted in the design and internal data is formatted as self-defined floating point to improve processor performance. In [16], an ultralong variable-size (up to 1M pts with 10 different sizes) pipelined FFT processor is presented for applications such as communications, imaging, radar processing, and spectrum analysis. This paper focuses on reducing the required memory size. The proposed solution in this paper is based on decomposing a one-dimensional FFT into multidimensional FFT calculations repeatedly and applying an efficient twiddle factor memory compression method. The work in [14] supports 8 different sizes up to 8K and targets flexible communication systems, where a single device integrates various wired and wireless communication standards. Resource usage optimization and efficient implementation are the two main concerns in this paper. Therefore, a dynamic address generator scheme and the CORDIC (coordinate rotational digital computer) technique are performed for twiddle factor multiplication to provide a conflict free in-place memory replacement scheme for intermediate data storage. [18] supports 5 different sizes up to 2K. It targets WiFi and WiMax. The architecture is designed based on radix-2 SDF. In this paper, we compare our work to also Xilinx FFT IP Generator v9.1 [26], which we also use in verification. The IP has 14 different sizes

up to 64K. Xilinx is listed on the third row in Table 1. The Xilinx IP has extra configurable parameters at compile-time.

3. FFT Basics

The N-point discrete Fourier transform (DFT) input sequence $x[n]$ defined as follows, where $W_N^{nk} = e^{-j(\frac{2\pi}{N})\alpha}$ and $\alpha = nk$:

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{nk}, k = 0, 1, ..., N-1 \tag{1}$$

N represents the size of DFT, $x[n]$ are the samples in time domain, and $X[k]$ represent output values of DFT in the frequency domain. W_N^{nk} is called as twiddle factor, and they are complex numbers that define rotations in a complex plane. The rotation operation is represented by the $\otimes$ in Figure 1 and throughout the article. When the size of DFT is the power of two, Cooley–Tukey FFT [6] algorithm is the most widely used method. N-point Cooley–Tukey requires $O(NlogN)$ arithmetic operations instead of the straight-forward implementation with $O(N^2)$ operations. Radix-2 DIF and DIT are the simplest algorithms for reducing the complexity of the design.

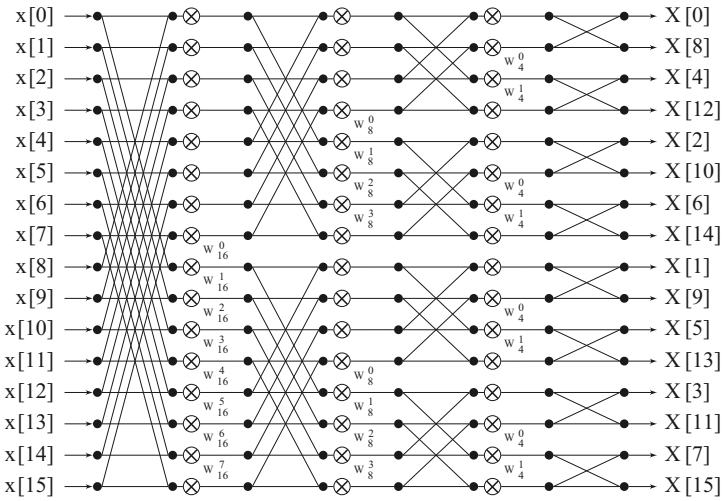


Figure 1. Flow-graph of 16-point radix-2 DIF FFT.

Figure 1 shows the flow-graph of 16-point radix-2 DIF FFT. The numbers at the input represent input sample index n for every $x[n]$ in time domain, and the numbers at the output represent the result of FFT $X[k]$ in frequency domain. Radix-2 based N-point FFT calculation is executed in $log_2 N$ stages. In the radix-2 algorithm, a butterfly is the basic operation, and butterflies calculate 2-point FFTs in each stage.

4. HC-FFT: proposed architecture

In this section, the proposed FFT architecture is explained in 6 subsections. The first subsection explains the 4-parallel architecture that takes in and also outputs 4 pts in a single cycle. Variable-size and multistream operation, IFFT, input rearrangement and output reordering are covered in the subsequent subsections.

4.1. 4-parallel radix-2 DIF FFT architecture

Figure 2 shows a 16-point FFT architecture, and it consists of R2, ⊗, and S blocks. Radix-2 butterfly can be shown in Figure 3a. R2 blocks perform addition and subtraction operations in a radix-2 butterfly. ⊗ symbols, which are located in the output branch of the R2 blocks, denote rotation operations as mentioned before in Section 3. Rotation operation is performed by multiplying the R2 block outputs with related twiddle multiplier values. Values above the ⊗ symbols represent α for W_N^{α}.

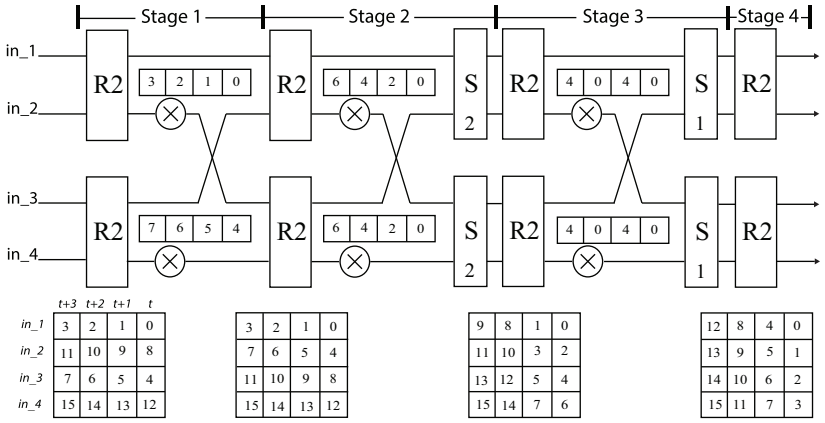
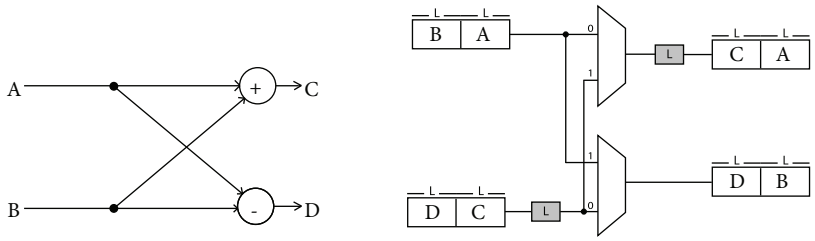


Figure 2. 4-parallel radix-2 16-point DIF FFT structure.

The input data index orders for different FFT stages are given in the tables, which are under the R2 blocks in Figure 2. While sample indexes in vertical arrive in the butterfly processor at the same time, in the horizontal, indexed data samples enter sequentially at different times, from right to left. As can be seen in Figue 2, the order of data index differs at each stage. Therefore, shuffling blocks are used to reorder the outputs of the current stage to feed the next stage. This block has already been used in various FFT architectures in the literature such as [1, 11]. A detailed block diagram of the shuffle block is given for L-length buffers in Figure 3b.



(a) R2 Block (b) Shuffle Block

Figure 3. Basic blocks.

Shuffle block consists of two L-length buffers and two multiplexers. The purpose of this circuit is to swap the B part of the upper input with the C part of the lower input. Therefore, select bit is set to 0 during first L cycles, thus A part of the data is stored in output buffer and C part of the data is stored in input buffer. For the second L cycles, the select bit is set to 1. Thereby, D part is stored in input buffer and C part passed to the output buffer while A part and B parts of the data provided to the output. When select bit changes again to 0, C and D part of data is provided to output. This way, the interchange of data is complete.

In Figure 2, the number given inside the S block is the buffer length. For P-parallel N-point FFT, length of the buffers in shuffling blocks can be obtained by calculating $L = N/2^{s+1}$ at any stage $s \in [p, n-1]$ where $n = log_2 N$ and $p = log_2 P$.

4.2. Wide-range variable-size FFT

The proposed FFT processor has been designed to support FFT sizes of 2^n, where n is between 4 and 16. The number of stages required to calculate 64K-point FFT is $16 = log_2 64K$.

Variable-size FFT structure will be explained over two consequent FFT sizes. 16-point and 32-point FFT configurations shown in Figures 2 and 4. Figure 2 includes 4 stages for N = 16 and Figure 4 includes 5 stages for N = 32. For these two figures given, the buffer width of the shuffle block in the stage after first stage increases as the FFT length increases. From this point of view, to calculate 16-point FFT using 32-point FFT architecture, multiplexer can be added between stages to select to drive the first stage outputs or previous stage outputs to next stage according to selected FFT size. The new architecture can be shown as in Figure 5.

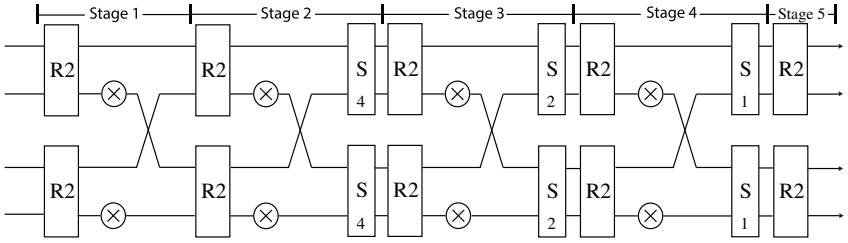


Figure 4. 4-parallel radix-2 32-point DIF FFT structure.

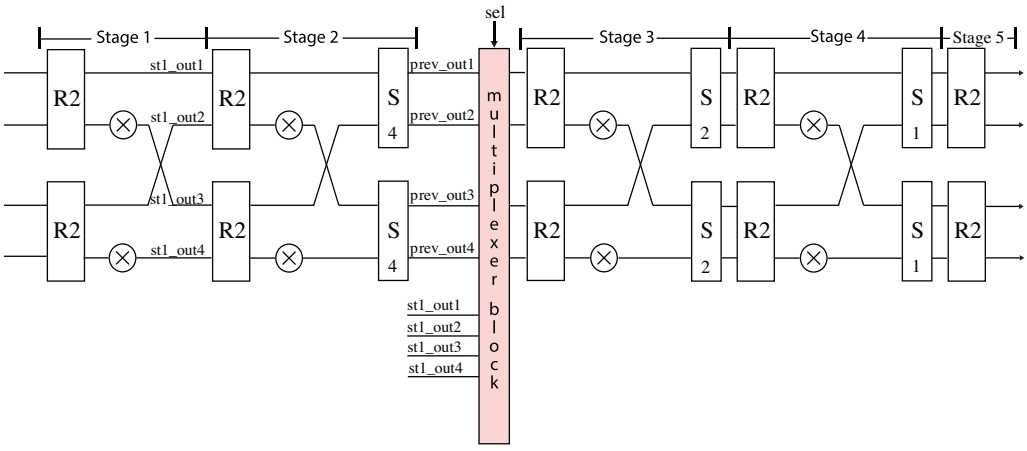


Figure 5. Variable-size FFT structure for 16/32-point FFT.

Proposed architecture includes 16 stages to support 64K-point FFT. According to configured FFT size, some of the stages will be inactivated. Regardless of the selected FFT size, the first stage and last three stages are always used. Because, the supported smallest FFT size is 16 points in the proposed architecture. Therefore, outputs of the FFT are always taken from last stage. Depending on the configured FFT size, the selection bits control the stages which come after the first stage.

Twiddle values, which are used for the rotation in the stages, are precomputed and stored in separate ROMs for each stage. Instead of storing all possible twiddle values for each stage, only the required twiddle values are stored in ROMs, thereby memory usage is optimized. Stored twiddle values for the respective stages are given in Figure 6.

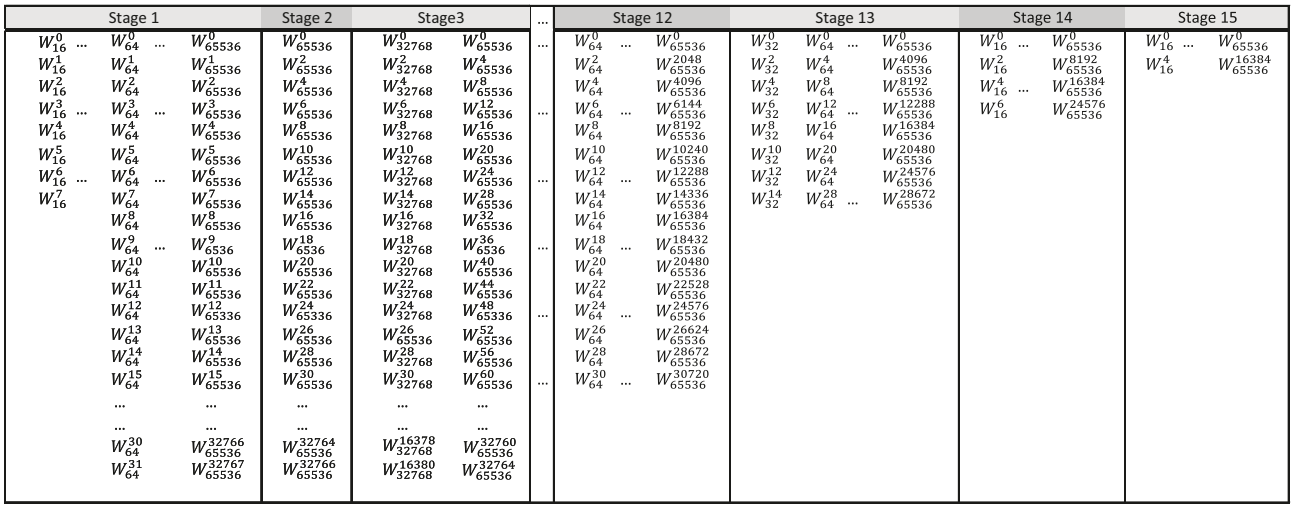


Figure 6. Twiddle multipliers used for each FFT stage.

As can be seen in Figure 6, stage 1 is the common stage for the FFT size which is supported by HC-FFT. Hereby, it should be said that used twiddle values for 64k-point FFT in stage 1 include other small FFT sizes twiddle values. Therefore, it will be sufficient to store the twiddle values which is used for the largest FFT size running in stage 1 in ROM. In other stages, it is seen that the same values are used regardless of the FFT length. For example, stage 13 is used for both 32-point FFT and 64-point FFT calculation. The equality of twiddle values given for stage 13 is can be formulated in Eq. 2.

$$W_{32}^{2} = W_{64}^{4} = W_{65536}^{4096} \tag{2}$$

It is shown in Eq. 3 that these values are equal to each other. If twiddle value formulates as follows: $W_N^{\alpha} = e^{-j(\frac{2\pi}{N})\alpha}$.

$$W_{32}^{2} = e^{-j(\frac{2\pi\times 2}{32})} = e^{-j(\frac{2\pi\times 2\times 2}{32\times 2})} = e^{-j(\frac{2\pi\times 4}{64})},\ W_{32}^{2} = W_{64}^{4} \tag{3}$$

When we examine Figure 6, the twiddle values used throughout the consecutive stages starting from stage 1 are decreasing. The area used in twiddle ROMs for any stage $s \in [1, n-1]$ can be formulated as 2^{n-s}. Based on this, the total area used for twiddle values can be calculated as in Eq. 4.

$$\sum_{s=1}^{n-1} 2^{n-s} = \sum_{s=1}^{n-1} 2^{log_2 N} \times 2^{-s} = \sum_{s=1}^{n-1} N \times 2^{-s} \approx N \tag{4}$$

4.3. Multistreaming

To illustrate combining the variable-size and multistreaming features, examination of 16-point 2-stream and 16-point 1-stream structures can be a good starting point. This example was chosen because it can be generalized

and makes it easier to examine the proposed structures. The sample data is first driven to the input of the reorder block, which will be explained in detail under the reorder circuit design section. The output of this circuit is driven to the first stage of FFT structure. For 1-stream 16-point FFT, reordered samples for the first stage are given in Figure 2 under first FFT stage. For 2-stream 16-point FFT, input reorder circuit reorders the FFT inputs with respect to the order index given in Figure 7 under first stage. 16-point FFT is calculated in four stages and the used twiddle values in each stage are the same for 1-stream, 2-stream, or 4-stream options. However, during the FFT calculation, the number of processed data in FFT stages varies depending on the number of streams. Therefore, the length of the shuffle blocks change according to the number of streams. As seen in Figure 2, shuffle-2 and shuffle-1 blocks are used in the FFT circuit for 1-stream 16-point FFT. When we examine the 2-stream 16-point FFT circuit given in Figure 7, the length of the shuffle-2 block increases to two, while the length of the last shuffle block remains constant.

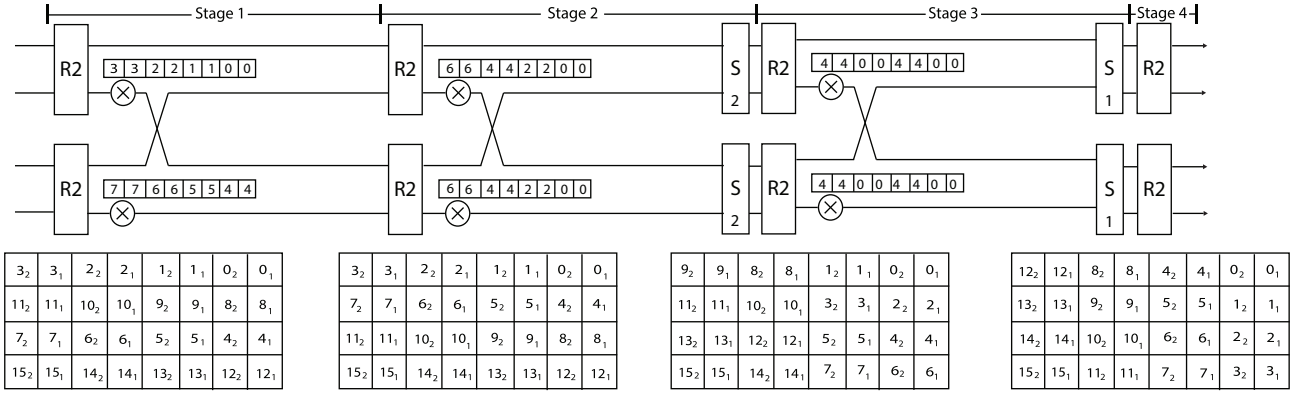


Figure 7. 2-Stream 16-point DIF FFT structure.

In Figure 7, first stream inputs are indexed as 0_1, 1_1, 2_1, ..., 15_1, and second stream inputs are indexed as 0_2, 1_2, 2_2, ..., 15_2. In 1-stream 16-point FFT circuit, 4 calculations are performed in each stage, while 8 calculations are performed in each stage for 2-stream. Therefore, twiddle values used in stages are repeated twice. Also, in 4-stream FFT circuits, twiddle values will repeat themselves 4 times. For 4-stream 16-point FFT, shuffle-8 replaces shuffle-2, while shuffle-4 replaces shuffle-1. For 4-stream FFT circuits, it can be said that the length of the shuffle blocks in a 1-stream circuit is quadrupled. To combine variable-size and multistreaming structures based on the described configurations, the only thing that will change is shuffle blocks on the used stages since stages and twiddle values remain constant. This can be accomplished by selecting the relevant shuffle block using the multiplexer.

4.4. Input rearrangement circuit

In previous titles such as designing 4-parallel radix-2 DIF FFT architecture and multistreaming, input index order for the first stage of HC-FFT is given in the related architecture figures. As seen in Figures 2 and 7, the data that are entering at the same time into the first stage of HC-FFT, are not received by the system simultaneously or consecutively in time. Therefore, the input sequence order should be rearranged for the first stage of FFT. For this purpose, previously described shuffle blocks were used to rearrange the input order.

Explanation of the input rearrangement circuit will be made for 16-point and 32-point FFT sizes, because of their simplicity and generalizable structure. FFT samples for 16-point provided to the architecture as shown in left side of the Table 2. The rearrangement circuit consists of shuffle blocks that are also used in the FFT

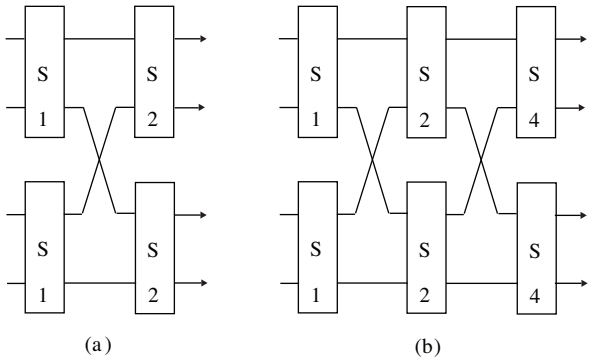


Figure 8. Input rearrangement circuit for (a) 16-point and (b) 32-point FFT.

stages. It is used to adapt sample order to the processing sequence in the FFT. The purpose of the circuit is to rearrange the sample data order which is given in left side of the Table 2 as the order in given right side of the Table 2. For 16-point and 32-point FFT, used input rearrangement circuits are given in Figure 8. For 16-point FFT, samples arriving at the terminals of the rearrangement circuit first enter the shuffle-1 block. Then, the output of shuffle-1 blocks enter shuffle-2 blocks by replacing of the lower output of the shuffle-1 block and the upper output of the shuffle-1. For 16-point FFT, the outputs of the shuffle-2 block are driven to the first FFT stage. The 32-point input rearrangement circuit includes an additional shuffle block, unlike the 16-point rearrangement circuit. For 16-point, the first stage inputs of FFT are taken from the output of the shuffle-2 block, while for 32-point the outputs of the shuffle-2 block are connected to the shuffle-4 block. The inputs of the first FFT stage are taken from output of shuffle-4 block for 32-point FFT.

Table 2. Input and output of input rearrangement circuit for 16-point FFT.

t+3	t+2	t+1	t		t+3	t+2	t+1	t
x_{12}	x_8	x_4	x_0		x_3	x_2	x_1	x_0
x_{13}	x_9	x_5	x_1	=	x_{11}	x_{10}	x_9	x_8
x_{14}	x_{10}	x_6	x_2		x_7	x_6	x_5	x_4
x_{15}	x_{11}	x_7	x_3		x_{15}	x_{14}	x_{13}	x_{12}

As can be seen from the 16-point and 32-point input rearrangement circuits in Figure 8, as the FFT increases in size, a shuffle block is added to the end of the rearrangement circuit. The input rearrangement circuit is designed for large FFT sizes but the circuit can also be used either smaller FFT sizes. Input rearrangement circuit which is designed for 65536-point consists of 28 shuffle blocks and this circuit can be used for 65536-point and smaller FFT sizes. For 65536-point FFT, inputs of the first FFT stage are taken from shuffle-8192 blocks, and for 32768-point FFT from shuffle-4096 blocks.

Until here, how the input rearrangement circuit works for 1-stream FFTs has been explained. The same input rearrangement circuit is also used for 2-steam and 4-stream FFTs with a few differences. The first stage inputs for 1-stream 16-point FFT are taken from the output of the shuffle-2 block, while the first stage inputs of the 2-stream 16-point FFT are taken from shuffle-4 and for 4-stream from the output of the shuffle-8 block. According to configured FFT size and stream number, related shuffle blocks outputs are driven to first stage of FFT.

4.5. Reorder circuit

In the proposed design, DIF FFT structure is used. Therefore, while the input data order should be natural order, FFT output is in bit-reversed order. The bit reversal circuit is one of the challenging block in the FFT design. [1, 2] and [3] are some of works which are mentioned about this issue. [2] is the one of the work in literature which specially discuss this problem and proposed a solution to reorder the FFT output. Although [1] is focused on parallel radix-2^k MDC FFT, it briefly mentioned about possible total memory usage of reorder block. In [3], bit reversal is calculated and proposed bit-reversal circuit for eight-parallel FFT structure.

Table 3. Last stage output order for 16-point FFT.

t+3	t+2	t+1	t		t+3	t+2	t+1	t
x_{12}	x_8	x_4	x_0		$X[3]$	$X[1]$	$X[2]$	$X[0]$
x_{13}	x_9	x_5	x_1	=	$X[11]$	$X[9]$	$X[10]$	$X[8]$
x_{14}	x_{10}	x_6	x_2		$X[7]$	$X[5]$	$X[6]$	$X[4]$
x_{15}	x_{11}	x_7	x_3		$X[15]$	$X[13]$	$X[14]$	$X[12]$

Our proposed output reorder block consists of 8-RAM blocks and this 8 RAM blocks behave such as single RAM block with RAM addressing unit. The main reason for using 8 RAM blocks in this way is that the proposed architecture is designed for a 4-parallel sample structure. Eight RAM blocks were required since 4 samples of FFT data had to be written to the RAM each cycle considering the order to adapt the FFT output data order. Output reorder block will be explained over the same FFT sizes, just as the 16-point and 32-point FFT sizes were used when explaining previous design parts before. For 16-point FFT, output data order is given in Table 3.

For 16-point and 32-point FFTs, the addressing rules differ from each other for writing the output data to RAMs and reading the data from RAMs. First of all, if we examine the addressing of 16-point FFT results during writing and reading, it is sufficient to use a 4-bit address for 16 data. 16-point FFT outputs, X[0], X[8], X[4], X[12], X[2], X[10], X[6], X[14], X[1], x[9], X[5], X[13], X[3], X[11], X[7], X[15] are written to addresses 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 and 15 respectively. Write addressing rule can be explained for 16-point. {waddr(3),waddr(2)} select the main RAM block group, waddr(0) addresses the RAM block which is located in the main group and finally, waddr(1) selects the data located in selected RAM block. For 32-point FFT outputs, 5 bit address is used for 32 data samples. Main RAM groups are represented by {waddr(4), waddr(3)} and selected RAM in main group is pointed by waddr(0) bit. Output data address in RAM is shown by {waddr(1),waddr(2)}.

Output data of X[25] are written to waddr "10011" in case of 32-point FFT size. Address waddr is decoded as {waddr(4), waddr(3)} = "10", {waddr(1), waddr(2)} = "10", waddr(0) = "1" and points address "10" of RAM-1 in the main RAM group "10". After the write operation for 16-point and 32-point FFTs, RAM group structure and data locations are given in Figure 9 for output reorder block. Also, the depth of each RAM block in the reorder block is selected 8192 to provide the memory area required for the 65536-point FFT. The data in the circle in the RAM blocks show the 16-point FFT outputs, while the data given in the rectangle show the 32-point FFT outputs.

During the reading process, the read address is decoded as follows. For 16-point FFT and 4 bit address, {raddr(0),raddr(1)} shows the main RAM group, raddr(3) select the RAM in main RAM group and raddr(2) is the data address in selected RAM block. For 32-point and 5-bit address, while {raddr(0),raddr(1)} again

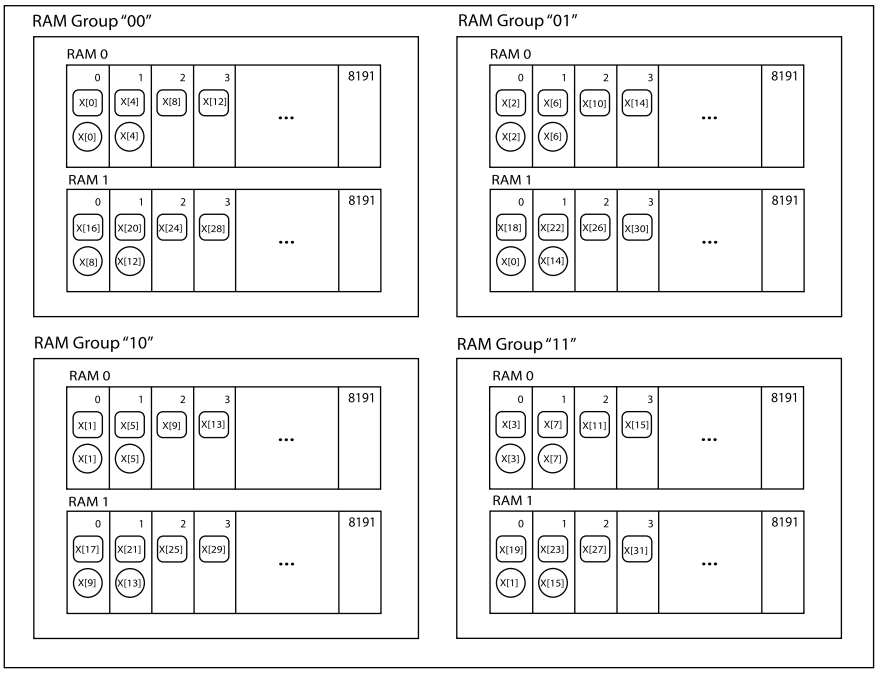


Figure 9. Output reorder RAM block.

shows the main RAM group, selected RAM is pointed by raddr(4) and data address in RAM is represented by {raddr(3),raddr(2)}. For 2-stream and 4-stream FFT circuits, the same output circuit is used with no change. For 2-stream 16-point FFT, there are 32 data samples to reorder. Therefore, 5 bits are required for addressing, so we can say that 2-stream 16-point output reorder circuit is the same with 1-stream 32-point output reorder circuit. Also, 4-stream 16-point FFT output circuit is the same with 1-stream 64-point FFT's output reorder circuit.

5. Implementation results

The proposed FFT architecture, HC-FFT, was implemented (i.e. integrated, simulated, synthesized, placed, routed) using Xilinx Vivado tool. The top-level and several layers below the top-level have been created using Vivado's "schematic block design" capabilities. Lower layers were created through RTL design in VHDL, which was about 3000 lines. The design was verified through simulation in Vivado and then later validated on a Xilinx Kintex UltraScale (US) FPGA. We dropped the design in our testbench, which was about 2000 lines of VHDL. We also wrote around 800 lines of MATLAB code to produce test vectors for verification and look-up tables (i.e. ROMs) for the design part of the project. The design was verified and validated against the Xilinx FFT IP Generator v9.1. HC-FFT and Xilinx IP were fed the same inputs (both 32-bit fixed-point) and their outputs were compared. The average relative error (i.e. the deviation divided by the Xilinx IP's output) was less than 0.00001 in all configurations.

Table 4 compares the synthesis results of HC-FFT with results of [13, 27, 32] as well as the Xilinx IP. The implementation results in the Table 4 are given for Virtex-7 FPGA. In Table 1, we had a wider list of previous works. Some of those works are not in Table 4 because their implementation results are not available for Virtex-7. In Table 4, size is the maximum number of FFT pts supported. MS/s is the throughput or million

Table 4. Comparison of synthesis results of HC-FFT with the previous work for Virtex-7 FPGA.

Design	Size	MS/s	MHz	LUT6	FF	DSP	BRAM	Latency
HC-FFT	64K	1350	339	20k	36k	240	396	49k
Xilinx	64K	372	372	6k	8k	24	550	196k
HC-FFT*	128K	1355	339	21k	38k	255	792	97k
[13]	128K	352*	352	14k	12k	88	428	-
HC-FFT*	1K	1261	339	12.5k	22.5k	150	6	858
[27]	1K	8889	555	13k	34k	192	148	110
HC-FFT*	4K	1327	339	15k	27k	180	24	3k
[32]	4K	400*	200	20k	-	-	-	2k

points per second. MHz column lists the maximum clock frequencies. Lat. is the latency in clock cycles. Note that the cycle-time can be smaller than the latency. On the other hand, LUT6, FF (flip-flop), DSP Slices, and Block RAM constitute the area of an implementation. We already showed that HC-FFT stands out in terms of its run-time configurability. Table 4 shows that it also stands out in terms of throughput. Note that the cells in Table 4 with a "-" show results that are not reported in the respective references and "*" is used to indicate possible maximum values.

HC-FFT processes 4-parallel samples in each cycle, however, Xilinx FFT IP processes 1 sample per cycle. The throughput of HC-FFT, in the case of 1-stream 64K-point FFT, can be calculated as $16K/(16K+64) \times 4 \times 339 = 1350$ MS/s, where 16K comes from the 16K cycles with valid data, 64 cycles is the gap between subsequent FFTs, 4 is the number of parallel data ports, and 339 MHz is the clock frequency. Similarly, the throughput of the Xilinx FFT IP, in the case of 64K-point FFT, can be calculated as $64K/64K \times 1 \times 372 = 372$ MS/s, where 64K/64K indicates no gap needed between subsequent FFTs, 1 indicates that Xilinx IP supports only 1 data port, and 372 MHz is the clock frequency. The latency of Xilinx FFT IP is 196k cycle for a 64K-point FFT, whereas our FFT core has a latency 49k cycles. As shown in Table 4, the Xilinx IP has competitive area but its throughput is not competitive.

Normalized values are given for 1K-point HC-FFT in 3rd line of the Table 4. Throughput of [13] is not reported but the possible maximum throughput can be 352MS/s according to given clock frequency and single data path architecture. According to Table 4, [13] is area-efficient but throughput remains very low compared to HC-FFT. [27] is has a pretty high throughput rate. However, considering the used resources, it can be said that HC-FFT is more area-efficient design. [27] uses 148 BRAM, 192 DSP, 34k FF and 13k LUT6 while HC-FFT uses only 6 BRAM, 150 DSP, 22.5k FF and 12.5k LUT6. Additionally, this design has only one configuration option, which is FFT size. It supports only 4 different sizes i.e. 16, 64, 256 and 1024 points. In terms of configurability, there are lack of options in [27] compared to our proposed design. Some parts of the utilization components are not reported in [32]. Therefore, area comparison cannot be made. Comparison can only be made in terms of used LUT6 and it can be said that HC-FFT use less LUT6 than [32]. Also, the maximum throughput of the design is calculated as 400MS/s for 2-parallel sample architecture. It can be said that our design outperforms [32] by terms of throughput and maximum clock frequency.

6. Conclusion

This paper has proposed a highly run-time configurable FFT design. HC-FFT can be configured in run-time for mode (F/I), size (12 different size from 16 to 64K), output order (N/B) and number of stream (1/2/4). Considering these features, HC-FFT allows 144 ($Mode \times Stream \times Size \times OutputOrder$) different configuration combination. Considering its configurable architecture, it is seen that there is no work that matches HC-FFT's configurability degree. HC-FFT achieves this with reasonable area. Our design also excels with its throughput of 87 Gbps, a rate that exceeds the peak data transfer rate of most interfaces. This high throughput is mainly because there is a negligible amount of stall (a gap of only 64 cycles) between consecutive FFTs, hence a very high pipeline efficiency. That is, a new FFT can start before the current finishes. The design has been verified against the Xilinx FFT IP. All of these features make the proposed design a good choice especially for real-time spectrum analysis systems.

References

[1] Garrido M, Grajal J, Sanchez MA, Gustafsson O. Pipelined radix-2^k feedforward FFT architectures. IEEE Transactions on Very Large Scale Integration Systems 2013; 21 (1): 23-32. doi: 10.1109/TVLSI.2011.2178275

[2] Chen S, Huang S, Garrido M, Jou S. Continuous-flow parallel bit-reversal circuit for MDF and MDC FFT architectures. IEEE Transactions on Circuits and Systems I: Regular Papers 2014; 61 (10): 2869-2877. doi: 10.1109/TCSI.2014.2327271

[3] Yoshizawa S, Orikasa A, Miyanaga Y. An area and power efficient pipeline FFT processor for 8×8 MIMO-OFDM systems. In: IEEE International Symposium of Circuits and Systems; Rio de Janeiro, Brazil; 2011. pp. 2705-2708.

[4] Boopal PP, Garrido M, Gustafsson O. A reconfigurable FFT architecture for variable-length and multi-streaming OFDM standards. In: IEEE International Symposium on Circuits and Systems; Beijing, China; 2013. pp. 2066-2070.

[5] Li S, Xu H, Fan W, Chen Y, Zeng X. A 128/256-point pipeline FFT/IFFT processor for MIMO OFDM system IEEE 802.16e. In: IEEE International Symposium on Circuits and Systems; Paris, France; 2010. pp. 1488-1491.

[6] Cooley JW, Tukey J. An Algorithm for the machine calculation of complex Fourier series. Mathematics of Computation 1965; 19 (90): 297-301. doi: 10.2307/2003354

[7] Kumar GP, Krishna BT, Pushpa K. Optimized pipelined fast Fourier transform using split and merge parallel processing units for OFDM. Wireless Personal Communications 2020; 117 (4). doi: 10.1007/s11277-020-07471-3

[8] O'Brien J, Mather J, Holland B. A 200 MIPS single-chip 1 k FFT processor. In: IEEE International Solid-State Circuits Conference; New York, NY, USA; 1989. pp. 166-167.

[9] Sunada G, Jin J, Berzins M, Chen T. COBRA: an 1.2 million transistor expandable column FFT chip. In: IEEE International Conference on Computer Design: VLSI in Computers and Processors; Cambridge, MA, USA; 1994. pp. 546-550.

[10] Baas BM. A low-power, high-performance, 1024-point FFT processor. IEEE Journal of Solid-State Circuits 1999; 34 (3): 380-387. doi: 10.1109/4.748190

[11] He S, Torkelson M. Design and implementation of a 1024-point pipeline FFT processor. In: IEEE Custom Integrated Circuits Conference; Santa Clara, CA, USA; 1998. pp. 131-134.

[12] Langemeyer S, Pirsch P, Blume H. A FPGA architecture for real-time processing of variable-length FFTS. In: IEEE International Conference on Acoustics, Speech and Signal Processing; Prague, Czechia; 2011. pp. 1705-1708.

[13] Xie Y, Feng Y, Yang C, Xie Y, Chen H. Design of a large point FFT processor with configurable transform length. In: IET International Radar Conference; Hangzhou, China; 2015. pp. 1-5.

[14] Gautam V, Ray KC, Haddow P. Hardware efficient design of Variable Length FFT Processor. In: IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems; Cottbus, Germany; 2011. pp. 309-312.

[15] Karachalios A, Nakos K, Reisis D, Alnuweiri H. A new FFT architecture for 4 × 4 MIMO-OFDMA systems with variable symbol lengths. In: International Conference on Innovations in Information Technology; Al Ain, United Arab Emirates; 2009. pp. 80-84.

[16] He C, Qiang W, Zhenbin G, Hongxing W. A pipelined memory-efficient architecture for ultra-long variable-size FFT processors. In: International Conference on Computer Science and Information Technology; Singapore; 2008. pp. 357-361.

[17] Manish B, Sangeeta N. Fast performance pipeline re-configurable FFT processor based on radix-2^2 for variable length N. International Journal of Electrical and Electronic Engineering and Telecommunications 2019; 8 (3): 163-170. doi: 10.18178/ijeetc.8.3.163-170

[18] Netto R, Güntzel JL. A high throughput configurable FFT processor for WLAN and WiMax protocols. In: VIII Southern Conference on Programmable Logic; Bento Goncalves, Brazil; 2012. pp. 1-5.

[19] He J, Ma L, Xu X. A configurable FFT processor. In: IET International Conference on Wireless, Mobile and Multimedia Networks; Beijing, China; 2010. pp. 246-249.

[20] Yang C, Xie Y, Chen L, Chen H, Deng Y. Design of a configurable fixed-point FFT processor. In: IET International Radar Conference; Hangzhou, China; 2015. pp. 1-4.

[21] Cho I, Shen C, Tachwali Y, Hsu C, Bhattacharyya SS. Configurable, resource-optimized FFT architecture for OFDM communication. In: IEEE International Conference on Acoustics, Speech and Signal Processing; Vancouver, Canada; 2013. pp. 2746-2750.

[22] Eddington C, Ray B. Using the parallel FFT for multigigahertz FPGA signal processing. Xcell Journal 2013; 82: 51-55.

[23] Iglesias V, Grajal J, Sánchez MA, López-Vallejo M. Implementation of a real-time spectrum analyzer on FPGA platforms. IEEE Transactions on Instrumentation and Measurement 2015; 64 (2): 338-355. doi: 10.1109/TIM.2014.2344411

[24] Verma R. FPGA implementation of fast Fourier transform (FFT) based finite impulse response (FIR) filter using VHDL. International Journal of Engineering and Computer Science 2014; 3 (3).

[25] Wang S, Inkol R, Rajan S, Patenaude F. FFT filter-bank-based wideband detection: coherent vs. non-coherent integration. In: IEEE Instrumentation and Measurement Technology Conference; Singapore; 2009. pp. 1327-1331.

[26] Xilinx Corp. Fast Fourier Transform v9.1 LogiCORE IP Product Guide PG109. San Jose, CA, USA: Xilinx Inc., 2020.

[27] Bruno JS, Almenar V, Valls J. FPGA Implementation of a 10 GS/s variable-length FFT for OFDM-based optical communication systems. Microprocessors and Microsystems 2019; 64: 195-204. doi: 10.3390/electronics7070116

[28] Nash JG. Distributed-memory-based FFT architecture and FPGA implementations. Electronics 2018; 7: 116. doi: 10.1016/j.micpro.2018.12.002

[29] Revanna D, Anjum O, Cucchi M, Airoldi R, Nurmi J. A scalable FFT processor architecture for OFDM based communication systems. In: International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation; Agios Konstantinos, Greece; 2013. pp. 19-27.

[30] Abdoli H, Nikmehr H. A flexible CO-OFDM using reconfigurable multi-precision FFT. IEEE Communications Letters 2017; 21: 1997-2000. doi: 10.1109/LCOMM.2017.2706675

[31] Glittas AX, Sellathurai M, Lakshminarayanan G. A normal I/O order radix-2 FFT architecture to process twin data streams for MIMO. IEEE Transactions on Very Large Scale Integration Systems 2016; 24 (6): 2402-2406. doi: 10.1109/TVLSI.2015.2504391

[32] Changela A, Zaveri M, Verma D. FPGA implementation of high-performance, resource-efficient radix-16 CORDIC rotator based FFT algorithm. Integration 2020; 73: 89-100. doi: 10.1016/j.vlsi.2020.03.008

[33] Wang J, Xie Y, Li B, Yang C, Hu S. The reconfigurable pipelined variable-point FFT processor design. In: IEEE International Conference on Signal, Information and Data Processing; Chongqing, China; 2019. pp. 1-4.