FPGA Implementation of Parallel Fast Fourier Transform

Sara M. Hassan [1,*] **, A.M. ELMahdy** [1]

[1] *Electronics and Communications Engineering Department, Modern Academy for Engineering and Technology, Cairo, Egypt*

A R T I C L E I N F O.

Article history:
Received 13 March 2024
Revised 6 May 2024
Accepted 12 May 2024
Available online 29 May 2024

Keywords:

Parallel
FFT
FPGA
VHDL
MATLAB

A B S T R A C T

The Fast Fourier Transform (FFT) is a fundamental signal processing technique. FFT implementations with high throughput are required by modern higher-speed signal processing and communications protocols such as 4G LTE, 5G, the Internet of Things (IoT), and so on. Furthermore, the FFT resolution mandated by the preceding standards differs depending on the mode of operation. As a result, it's highly desired to own an FFT implementation which not just supports the high throughput requirement, but additionally expandable to handle any configurable N point FFT resolution. This study simulates and implements the Parallel 256-point radix-2 Fast Fourier Transform. The Parallel FFT technique improves system performance and makes it faster. The simulation was carried out in MATLAB (version R2023b), and the implementation was carried out in the Virtex 6 XC6VLX240T FPGA kit using VHDL codes written in the Xilinx package version 14.7, with ModelSim (version SE-64 10.6d) used to present the simulation results.

1. Introduction

In order to minimize the computational complexity of mathematical calculations from a magnitude associated with the order O(N^2) for Discrete Fourier Transform (DFT) to O(NlogN) arithmetic operations, Cooley-FFT Tukey's method first implemented this symmetry in DFT operation [1-5]. FFT is utilized in a wide range of technical applications, including radar processing, higher throughput image processing, speech-recognition, and Compressed data. In a similar vein, large throughput FFTs are needed for the functioning of higher-speed communication protocols such as 4G LTE & LTE-Advanced, 5G communication-systems, and the growing Internet of Things (IoT). Furthermore, the FFT resolution needed for the execution of these higher-speed applications is affected by multiple methods of operating [6-14]. The size of an FFT spans from 64 to 2048 points for 3G, 4G, and WiMAX communication protocols [15-17]. In comparison to 4G LTE, the 5G FFT size transmission is predicted to be higher. It's highly desired for there to be an adaptable FFT structure that could be resized or modified based on operational needs avoiding having to go via the onerous process of redesigning every N-point FFT resolution. In light of changing FFT accuracy needs for various modes. The radix-2 butterfly block, suggested by Cooley-Tukey [18-22], is the foundation of the fundamental FFT design. Using the symmetric features of the FFT, all

*** Corresponding author**

E-mail address: *sara.hassan@eng.modern-academy.edu.eg*

standard FFT techniques divide the dimension N-FFT onto two pieces, specifically an odd half with an even half, so significantly decrease the total number of multiplications [23-26].

Cooley-Tukey introduced the FFT for the first time in order to simplify the arithmetic, Cooley-Tukey took advantage of the symmetry in the DFT. In their work, they subdivided a N-points sequence to N/2 odd and even subsequences. The radix 2 approach, which is the simplest fundamental algorithm, has gained prominence. For an N-point FFT, log2N stages are needed, with N /2 butterflies in each step [26-29]. The FFT algorithms used in numerous earlier articles [30-37] are created automatically by Xilinx cores rather than by computer programs; as a result, they cannot be altered or observed in almost Xilinx versions. The Xilinx FFT intellectual property (IP) cores can only be used to insert standard specifications; they cannot be used to modify specific design specifications. Previous research sought to speed up the FFT process in a variety of ways, including replicating the algorithm with some mathematical operations reduced. However, when the input size increases, the acceleration advantage reduces [38-41]. The paralleling approach enables for any input size to be used. This study presents high acceleration parallel FFT MATLAB simulation with real time hardware FPGA implementation to boost system performance and perform faster. Section two of the study is titled "Simulation of FFT." FFT Implementation is introduced in Section 3, while the Conclusions are presented in Sections 4.

2. Simulation of FFT

In this part, MATLAB is used to simulate an FFT signal with a size of 256 points (version R2023b). In DFT, a discrete-time signal is represented by a series of sinusoidal-functions. The DFT for a complex-valued discrete-time sequence x(n) of N-point is realized by Equation (1).

$$X(K) = \sum_{n=0}^{N-1} x(n) W_N^{Kn} \ , k = 0, 1, 2, \dots, N. \tag{1}$$

W_N^{kn} is the phase factor provided by Equation (2).

$$W_N^{Kn} = e^{\frac{-j2Kn\pi}{N}} \tag{2}$$

The input function x(n) discrete-time domain value in the frequency domain is represented by the function X(K). N complex multiplications and (N-1) complex additions are needed for the direct analysis of a single-point DFT. As a result, an N-point DFT employs N^2 complex-multiplications and N(N.1) complex- additions, rendering the computing cost of DFT proportionate to O (N^2). The direct calculation of the DFT makes computing prohibitively demanding for huge numbers of N-points. FFT is a group of algorithms that accelerates DFT calculations by taking advantage of their symmetry. The speed-up given by the FFT technique is achieved by reducing the number of complex-additions from N(N-1) in DFT methods to $N\log_2(N)$, and the number of complex multiplications from N^2 to (N/2) x $\log_2(N)$. Figure 1 depicts the four input single frequency signals, whereas Figure 2 depicts the amplitudes of the FFT output signals.

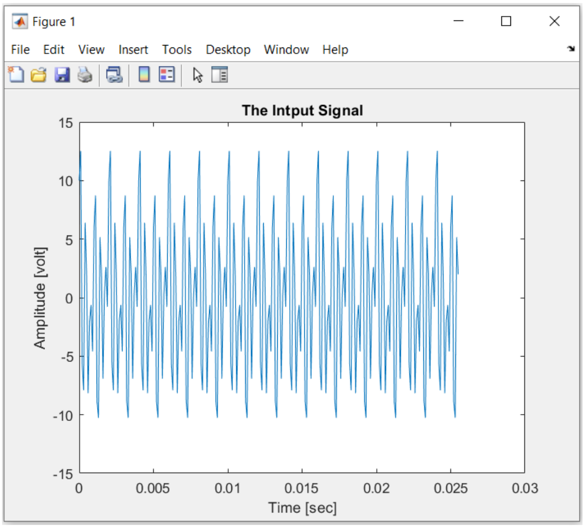


Fig. 1. The Input Signals.

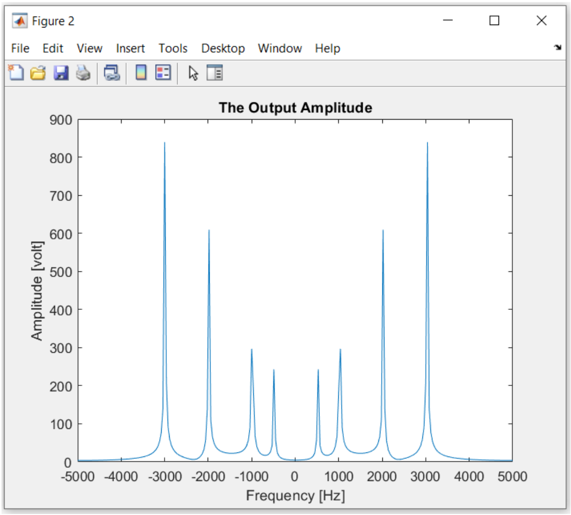


Fig. 2. The FFT Amplitudes.

3. Implementation of FFT

Various multidimensional indexing mappings for the input and output sequences can be used to categorize all FFT approaches. These are based on Equation (2), an N-length DFT transform. In general, it is sufficient to study merely the two-factor instance since greater dimensions could be constructed by just replacing one of these components iteratively. In order to simplify this description, the Cooley-Tukey FFT algorithm will only be presented via the two-dimensional indexed transform. The time-index n is transformed by Equation (3), which has the constants A and B $\in Z$, $N = N_1N_2$. Equation (4) is obtained by applying a different index mapping k for the resultant frequency domain, in which C and D $\in Z$ are constants. In contrast, for a prime-factor algorithm (PFA), the factors N_1 and N_2 need to be a coprime, not primes in and of themselves. This is important to note since the Cooley-Tukey method can actually create FFTs through two-factors, $N = N_1N_2$, whose are coprime.

$$n = An_1 + Bn_2 \; mod \; N \quad \begin{cases} 0 \le n_1 \le N_1 - 1 \\ 0 \le n_2 \le N_2 - 1 \end{cases} \tag{3}$$

$$k = Ck_1 + Dk_2 \; mod \; N \quad \begin{cases} 0 \le k_1 \le N_1 - 1 \\ 0 \le k_2 \le N_2 - 1 \end{cases} \tag{4}$$

Any factorization of N is conceivable, hence the Cooley Tukey FFT is the most adaptable of all the FFT algorithms. The Cooley Tukey FFTs are the most popular, with transform length N being a power of a basis r. These algorithms are frequently also known as radix r algorithms. The index mapping proposed by Cooley Tukey is the most basic. The mapping illustrated below is based on Equation (3) and assumes that A = N_2 and B = 1, as in Equation (5). According to the range that n_1 and n_2 are valid for, the modulo reduction indicated by Equation (3) does not need to be explicitly computed. Selecting C = 1 & D = N_1 yields the subsequent mapping for the inverse mapping from Equation (4) Cooley and Tukey to Equation (6). In this case, omitting the modulo computation is also an option. Assuming that n and k have now been substituted for the proper values in W_N^{kn} by Equations (5) and (6), find Equation (7). Considering that since W has order N = N_1N_2, it concludes that W_N^{N1} = W_{N2} and W_N^{N2} = W_{N1}, which simplifies the Equation (7) to Equation (8). As a result, in the DFT Equations (9) and (10), if now replaces Equation (2) with Equation (8).

$$n = N_2 n_1 + n_2 \quad \begin{cases} 0 \le n_1 \le N_1 - 1 \\ 0 \le n_2 \le N_2 - 1 \end{cases} \tag{5}$$

$$k = k_1 + N_1 k_2 \quad \begin{cases} 0 \le k_1 \le N_1 - 1 \\ 0 \le k_2 \le N_2 - 1 \end{cases} \tag{6}$$

$$W_N^{nk} = \; W_N^{N_2 n_1 k_1 + N_1 N_2 n_1 k_2 + n_2 k_1 + N_1 n_2 k_2} \tag{7}$$

$$W_N^{nk} = W_{N_1}^{n_1 k_1} W_N^{n_2 k_1} W_{N_2}^{n_2 k_2} \tag{8}$$

$$X[k_1, k_2] = \sum_{n_2=0}^{N_2-1} W_{N_2}^{n_2 k_2} \underbrace{\left(W_N^{n_2 k_1} \underbrace{\sum_{n_1=0}^{N_1-1} x[n_1, n_2] W_{N_1}^{n_1 k_1}}_{N_1 - point\ transform}\right)}_{\bar{x}[n_2, k_1]} \tag{9}$$

$$= \underbrace{\sum_{n_2=0}^{N_2-1} W_{N_2}^{n_2 k_2} \, \bar{x}[n_2, k_1]}_{N_2 - point\ transform} \tag{10}$$

The Cooley-Tukey approach differentiates itself from previous FFT algorithms by the ability to choose the factors for N freely. As a result, a radix r method with N = r^S is feasible. The most widely used approaches include those having a base of r = 2 that don't include any multiplications. For instance, in Equations (11) and (12), the following index mapping appears with r = 2 and S phases.

$$n = 2^{S-1}n_1 + \cdots + 2n_{S-1} + n_S \tag{11}$$

$$k = k_1 + 2k_2 + \cdots + 2^{S-1}k_S \tag{12}$$

With S > 2, the two-point DFT is commonly represented by a butterfly in the signal flow graph, as seen in Figure 3 for an eight-point transform. The representation of the signal's path graph is now reduced by taking use of the reality that all incoming arrows at a node are combined, and the constants used for coefficient multipliers are represented by the value of a factor with an arrow. In the radix r technique with $\log_r(N)$ steps, a similar form of the twiddle factor emerges for each group. The signal's flow diagram in Figure 3 indicates that because the data are no longer needed for the future computations, the calculation could be performed situated, which means that the storage region with a butterfly can be rewritten. For the reason that the twiddle factor only applies to every second arrow, $\log_2(N)$ N/2 represents the total quantity of twiddle factor multipliers in the radix 2 transformation. The method shown in Figure 3 is known as a decimation in frequency (DIF) approach because it begins by splitting originally created DFT to smaller DFTs in the frequency domain. Whereas the numerical index of values for frequency is in slightly inverted order, the input values are generally in natural order. Table 1 displays the typical values for the DIF radix 2 approach.

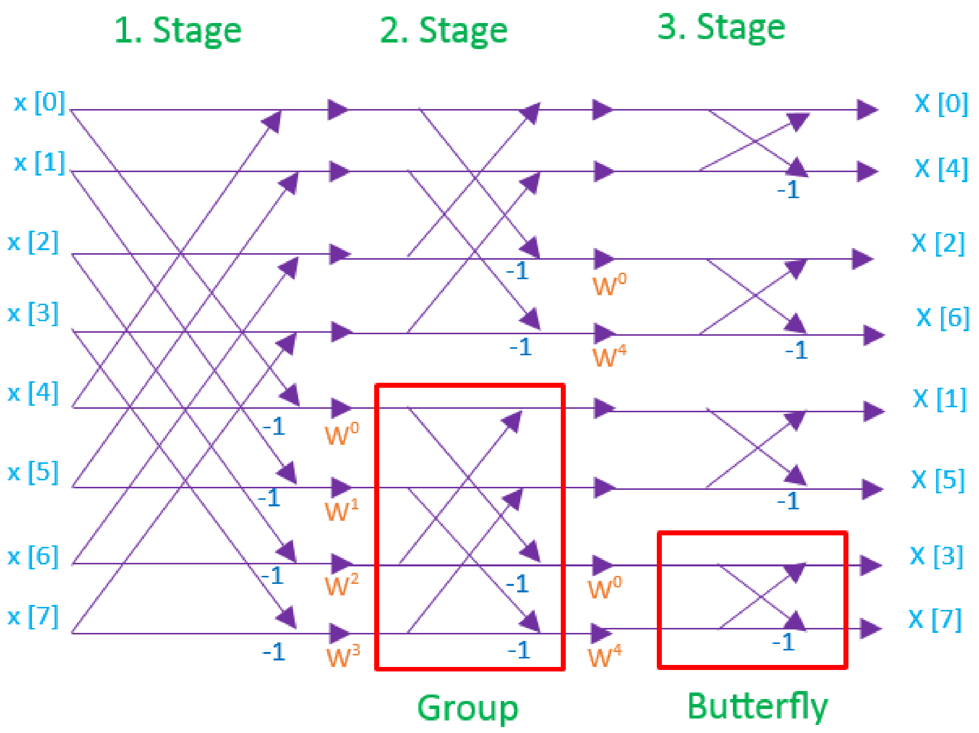


Fig. 3. Length-8 Algorithm for Radix-2 Decimation in Frequency.

Table 1. Frequency Decimation Using Radix-2 FFT.

	Stage 1	Stage 2	Stage 3	...	Stage $\log_2(N)$
Number of groups	1	2	4	...	N/2
Butterflies per group	N/2	N/4	N/8	...	1
Increment exponent twiddle factors	1	2	4	...	N/2

3.1 FFT Radix 2 Implementation

This part implements the Cooley Tukey Length 256 FFT approach by writing VHDL code with the Xilinx package version 14.7 and exhibiting ModelSim simulation results. The radix 2 FFT could be effectively executed by the butterfly processing that has a sophisticated multiplier to determine the twiddle factors as well as the butterfly itself. A radix 2 butterfly processing unit is made up of a complex adder, A complex subtractor and a complex multiplier are used for the twiddle factors. To obtain the complex multiplies utilizing the twiddle factor, the four real multipliers and two add-subtract procedures are typically employed. Even so, because one operand is precomputed, it is also feasible to construct a complex multiplication using just three real multipliers and three addition-subtraction actions. The approach includes of three multiplications, two subtractions, and one addition. When constructing a full-size FFT, all of the data is merged and constructed in accordance with the strategy described in Table 1. Following that, the code is written to update the butterfly data and increase the number of dual nodes, twiddle factors, and group sizes. The FFT RTL Schematic design is shown in Figure 4. The results of the ModelSim simulation start for the tri-angular input values x(n) = [20, 40, 60, 80, 100, 120, 140, 160, 0, 0...] are shown in Figure 5. The first eight numbers are the only ones that have nonzero values. The following can be used to construct the test sequence in MatLab:[Xz= [(1:8)*20, zeros (1,248)]; and Yn=fft(Xz);].

First, start by forcing the inputs (valid and reset) with ones, then force them with zeroes to load the values into the register as indicated by xr_out and xi_out (the real and imaginary first 8 eight register files). Figure 6 shows how the FFT values are shown on the output ports as soon as the reverse state is changed to forward. To show FFT values at the output ports fftr & ffti (the real and imaginary output values), setting the fft_valid flag to one at the same time. The test data presented in Figure 6 corresponds to the expected values from the MATLAB simulation shown in Figure 7.

720 714 698 671 634 587 532 471 403 ... (real).

0 -82 -159 -260 -313 -380 -439 -490 -532 ... (imag).

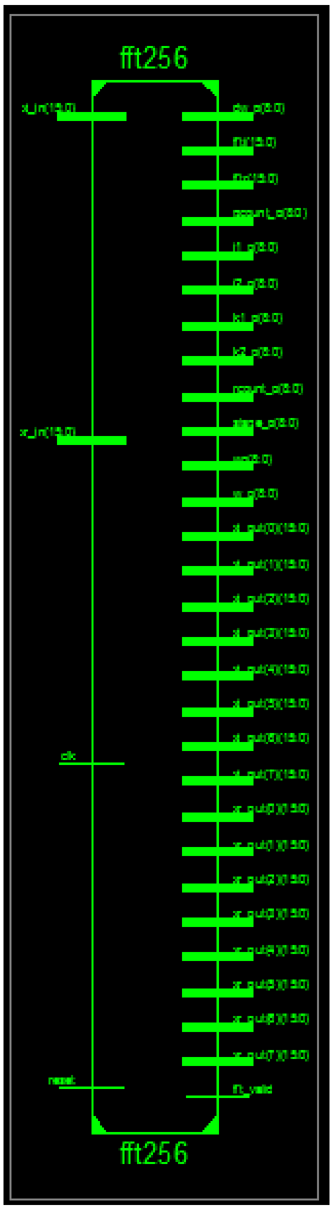
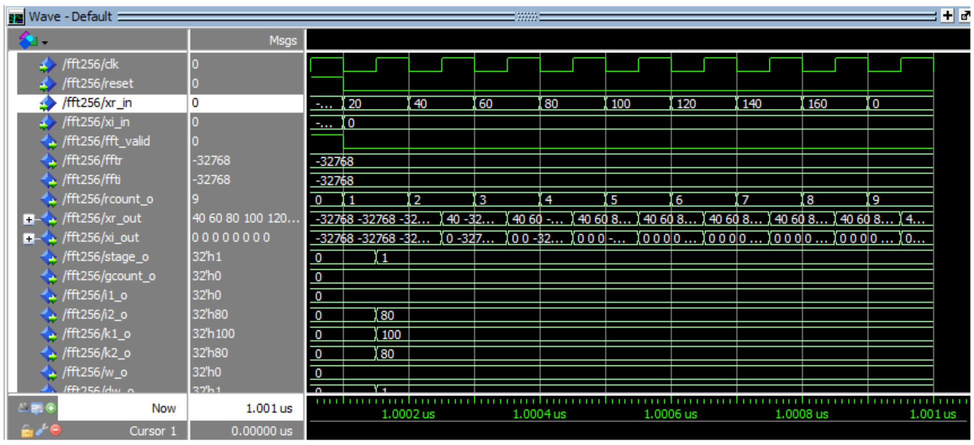
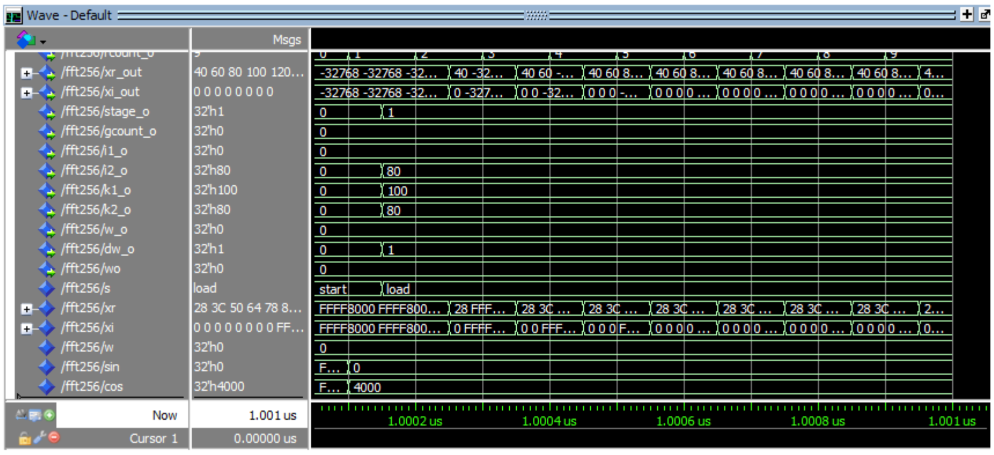


Fig. 4. RTL Schematic diagram.

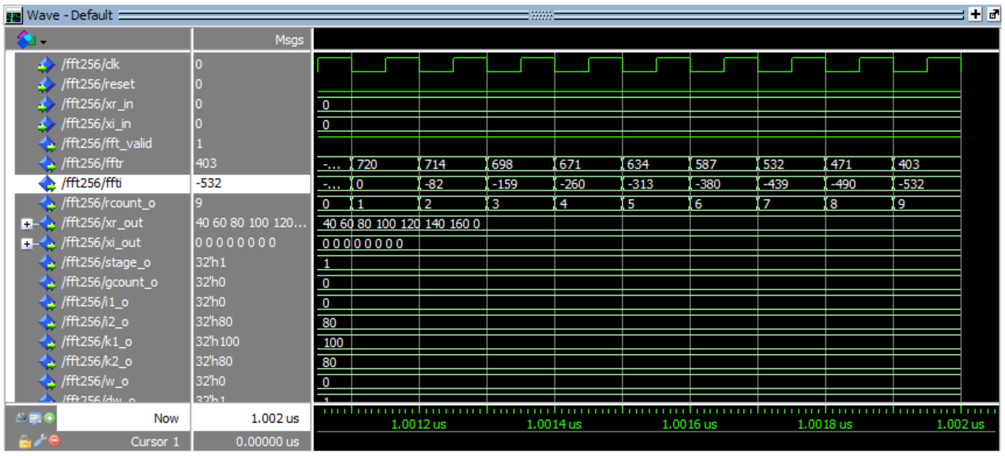


(a)

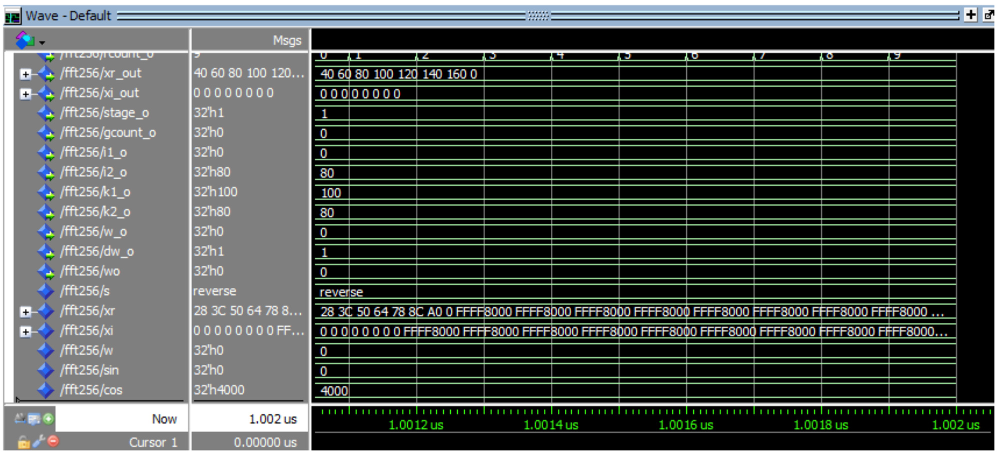


(b)

Fig. 5. The FFT ModelSim Simulation (Start then load of the input frame), (a) xr_out and xi_out (the real and imaginary first eight register files); (b) Start and load states.



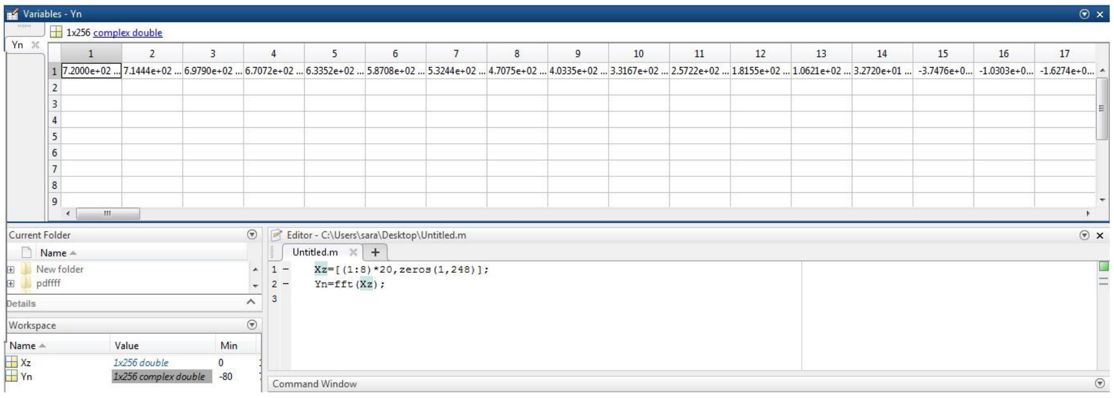
(a)



Signal	Msgs	Wave value
/fft256/xr_out	40 60 80 100 120...	40 60 80 100 120 140 160 0
/fft256/xi_out	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
/fft256/stage_o	32'h1	1
/fft256/gcount_o	32'h0	0
/fft256/i1_o	32'h0	0
/fft256/i2_o	32'h80	80
/fft256/k1_o	32'h100	100
/fft256/k2_o	32'h80	80
/fft256/w_o	32'h0	0
/fft256/dw_o	32'h1	1
/fft256/wo	32'h0	0
/fft256/s	reverse	reverse
/fft256/xr	28 3C 50 64 78 8...	28 3C 50 64 78 8C A0 0 FFFF8000 FFFF8000 FFFF8000 FFFF8000 FFFF8000 FFFF8000 FFFF8000 FFFF8000 ...
/fft256/xi	0 0 0 0 0 0 0 0 FF...	0 0 0 0 0 0 0 0 FFFF8000 FFFF8000 FFFF8000 FFFF8000 FFFF8000 FFFF8000 FFFF8000 FFFF8000 FFFF8000...
/fft256/w	32'h0	0
/fft256/sin	32'h0	0
/fft256/cos	32'h4000	4000

(b)

Fig. 6. FFT ModelSim Simulation in the reverse state, (a) ports fftr and ffti (the real and imaginary output values); (b) reverse state changes to forward.



Yn: 1x256 complex double

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	7.2000e+02 ...	7.1444e+02 ...	6.9790e+02 ...	6.7072e+02 ...	6.3352e+02 ...	5.8708e+02 ...	5.3244e+02 ...	4.7075e+02 ...	4.0335e+02 ...	3.3167e+02 ...	2.5722e+02 ...	1.8155e+02 ...	1.0621e+02 ...	3.2720e+01 ...	-3.7476e+0...	-1.0303e+0...	-1.6274e+0...

```
Xz=[(1:8)*20,zeros(1,248)];
Yn=fft(Xz);
```

Name	Value	Min
Xz	1x256 double	0
Yn	1x256 complex double	-80

Fig. 7. Matlab Simulation for the Triangular Input Test Sequence.

3.2 FPGA Utilization

In this part, the Virtex 6 (XC6VLX240T) FPGA kit is used for the implementation. Figure 8 depicts the FPGA kit in use. Eight embedded multipliers and about 34,340 logic elements (LEs) are used in the design, which has a registered performance of $F_{(max)}$=31.12MHz using the Time Quest slow modelling (85C). The utilized (LEs) in The VHDL design is going to be decreased for an optimizing objective area and cosine and sine LUTs, it will then be synthesized as embedded Memory Blocks (M9K) blocks. The LEs generated from the VHDL LUTs are designed to be fast. The design goals and strategies part of the Xilinx ISE package version 14.7 were used to develop the area optimization strategy. Table 2 shows the FPGA consumption when the area reduction design aim is used.

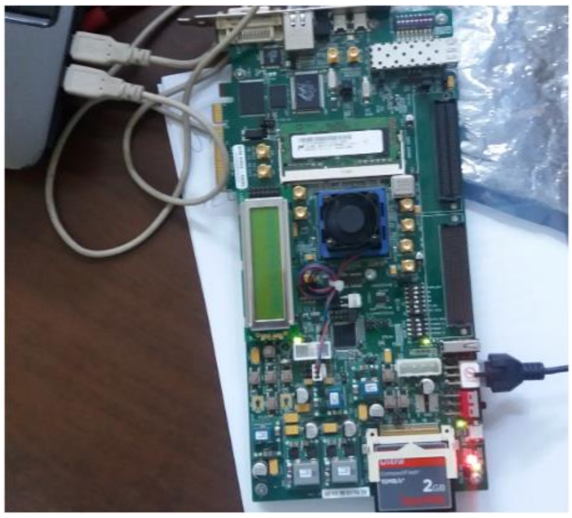
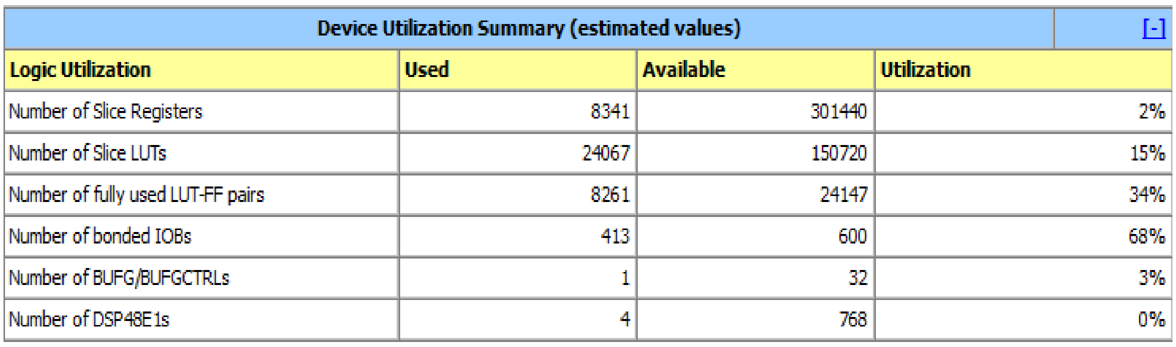


Fig. 8. FPGA Kit for Virtex 6 (XC6VLX240T).

Table 2. The Virtex 6 System Utilization with the Area Reduction Design Objective.



Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	8341	301440	2%
Number of Slice LUTs	24067	150720	15%
Number of fully used LUT-FF pairs	8261	24147	34%
Number of bonded IOBs	413	600	68%
Number of BUFG/BUFGCTRLs	1	32	3%
Number of DSP48E1s	4	768	0%

3.3 Parallel FFT

This section provides a parallel FFT design that is appropriate to work with these FPGA implementations. It is made up for numerous parallel/pipelines and the front-end butterfly-shaped circuit for refining and disseminating incoming data. The FFT in Figure 9 was developed in the fixed-point and blocks floating-point models employing by Schematic creator constructed with VHDL codes, Subsequently the simulation and evaluated on the Virtex 6 version XC6VLX240T. The Radix-2^2 technique is used by the Schematic generator for parallel FFT pipelines. Table 3 exhibits the Quad FFT system's Virtex 6 FPGA resource area consumption. While comparing quad utilization to single usage in Table 2, it is apparent that the quad technique employs more kit logic elements, increasing the system's complexity. Even though the Quad approach takes additional FPGA resources, it still makes use of a modest Virtex 6 resource block. According to Table 3, the number of slice registers and LUTs used is 7% and 12%, respectively. Remarkably, the quad methodology enhanced the percentage of completely LUT-FF pairs (utilization effectiveness) from 34% in the single method to 56%.

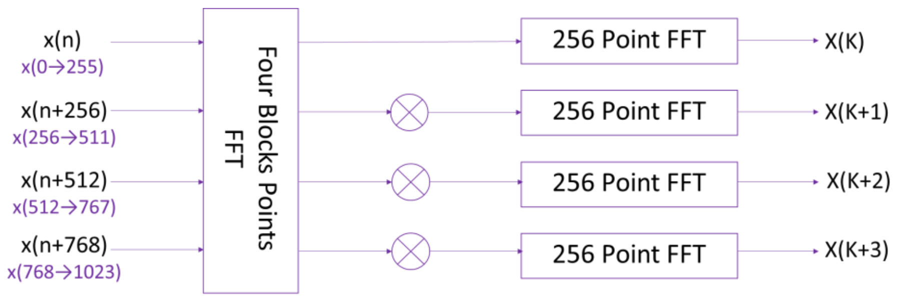
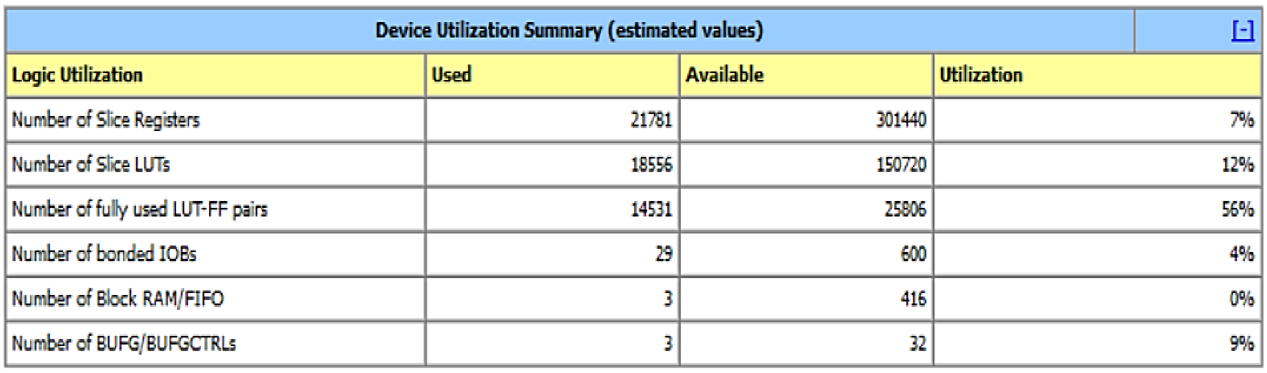


Fig. 9. 1024 point Quad Pipeline Radix 2^2 FFT.

Table 3. The Virtex 6 Quad System Area Utilization.



Device Utilization Summary (estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	21781	301440	7%
Number of Slice LUTs	18556	150720	12%
Number of fully used LUT-FF pairs	14531	25806	56%
Number of bonded IOBs	29	600	4%
Number of Block RAM/FIFO	3	416	0%
Number of BUFG/BUFGCTRLs	3	32	9%

Table 4. FFT Fixed points on the XC6VLX240T.

Pipeline Method	Slices Registers	Block RAMs	Block MULTs	Speed (MHz)	Latency (cycles)	Throughput Msps	Transform Time (s)	Area x Transform Time Product
Single	8341	23	34	173	482	175	1.67	13929
Quad	21781	41	123	152	142	651	0.45	8787

The outcomes of single and pipeline implementations are shown in Table 4. The table solely contains FFT computations with fixed-points. The throughput of these FFTs can be calculated in samples per second by multiply the clock value by the total quantity of pipes. A table evidently indicates the nature of the quadruple FFT pipeline has one-fourth the transforming time of a single-pipeline FFT while being only two to three times as large. The throughput increases by around four times as a result of the parallel concept, which sends four parallel blocks of data (1024 points) in the quad method instead of a single method (256 data points), and the delay amount reduces due to the pipeline method being used. The pipeline processing concept aims to reduce processing time by reducing the number of clocks consumed by the entire system, as each new clock has a new FFT process that will be started and another one will be finished, whereas in the normal single method, the FFT process does not begin until the previous FFT process is completed.

4. Conclusion

The requirement for FFT to demonstrate spectrum analysis is greater. FFT is necessary for a variety of technical applications such as radar processing, speech recognition, data compression, and picture and audio processing. High throughput FFTs are additionally necessary for the functioning of high-speed communication protocols involving 4G LTE and LTE-Advanced, 5G communication systems, and the developing Internet and IoT. Furthermore, different modes of operation have distinct requirements for the FFT resolution required to achieve these high-speed applications. The Fast Fourier Transform (FFT) 256-point radix 2 single and parallel (Quad pipeline) is simulated and implemented in this article. The simulation was carried out using MATLAB (version R2023b), and the implementation was carried out using the Virtex 6 XC6VLX240T FPGA kit by writing VHDL codes with the Xilinx package version 14.7 and demonstrate the simulation results in ModelSim (version SE-64 10.6d). The findings of the MATLAB simulation are identical to the results of the Modelsim simulation. The system exhibits good resource use due to the design goal of reducing area and adequate computation performance. The design goal of area reduction reduces system usage by 17.31%. This research's simulation and implementation outcomes demonstrated that it was successful in accomplishing its purpose. The system makes effective use of resources while maintaining an acceptable computation speed. Low order radix methods, like for instance radix 2, necessitate difficult recursive decompositions for higher N FFT points processing, resulting in a substantially greater number of phases, or stages. Consequently, they are incapable of meeting the throughput requirements for applications with high speeds. For that, parallel processing and the pipeline are employed so as to reduce the amount of time consumed and boost the system throughput. Clearly, such a parallel pipelines technique Could give rise to exceedingly as large throughputs, which are only limited by the available amount of chip resources. It additionally enables the trade-off between chip areas and improving throughput. Attempting to use fewer hardware resources would be another goal. The complexity of the system will thereafter decline.

References

[1] Lakshmanan, R., Pichler, A., & Potts, D. (2022). Nonequispaced fast Fourier transform boost for the Sinkhorn algorithm. arXiv preprint arXiv:2201.07524. https://doi.org/10.48550/arXiv.2201.07524

[2] Verma, M., Chatterjee, S., Garg, G., Sharma, B., Arya, N., Kumar, S., ... & Verma, M. K. (2023). Scalable Multi-node Fast Fourier Transform on GPUs. SN Computer Science, 4(5), 625. https://doi.org/10.48550/arXiv.2202.12756

[3] He, Y., Podobas, A., Andersson, M. I., & Markidis, S. (2022, August). FFTc: An MLIR Dialect for Developing HPC Fast Fourier Transform Libraries. In European Conference on Parallel Processing (pp. 80-92). Cham: Springer Nature Switzerland. https://doi.org/10.48550/arXiv.2207.06803

[4] Gasmi, A. (2022). What is Fast Fourier Transform? (Doctoral dissertation, Société Francophone de Nutrithérapie et de Nutrigénétique Appliquée). https://hal.science/hal-03741810

[5] Smyk, R., & Czyżak, M. (2020). On implementation of FFT processor in XILINX FPGA using high-level synthesis. Poznan University of Technology Academic Journals. Electrical Engineering, (104), 17-33. https://doi.org/10.21008/j.1897-0737.2020.104.0002

[6] Nor, N. S. M., Malim, N. H. A. H., Rostam, N. A. P., Thomas, J. J., Effendy, M. A., & Hassan, Z. (2022). Automated classification of eight different Electroencephalogram (EEG) bands using hybrid of Fast Fourier Transform (FFT) with machine learning methods. Neuroscience Research Notes, 5(1), 116-116. https://doi.org/10.31117/neuroscirn.v5i1.116

[7] Chu, K. U., & Ho, Y. H. (2022). Max Fast Fourier Transform (maxFFT) Clustering Approach for Classifying Indoor Air Quality. Atmosphere, 13(9), 1375. https://doi.org/10.3390/atmos13091375

[8] Jayan, G., & Nair, A. K. (2018, March). Performance analysis of filtered OFDM for 5G. In 2018 international conference on wireless communications, signal processing and networking (WiSPNET) (pp. 1-5). IEEE. https://doi.org/10.1109/WiSPNET.2018.8538544

[9] Dawa, M., & Abdallah, F. B. (2018). FPGA implementation of the IEEE 802.16 physical layer for SHVC video transmission. Journal of Circuits, Systems and Computers, 27(12), 1850190. http://dx.doi.org/10.1142/S0218126618501906

[10] Sekhar, R. C., & Nagamani, D. Optimaization of PAPR using Non-Linear Companding with Weighting Function for MIMO-OFDM Systems. http://dx.doi.org/E10840275S419/19©BEIESP

[11] Gonzalez, Y., & Prati, R. C. (2022). Acoustic Descriptors for Characterization of Musical Timbre Using the Fast Fourier Transform. Electronics 2022, 11, 1405. https://doi.org/10.3390/electronics11091405

[12] Locharla, G. R., Mahapatra, K. K., & Ari, S. (2018). Variable length mixed radix MDC FFT/IFFT processor for MIMO-OFDM application. IET Computers & Digital Techniques, 12(1), 9-19. https://doi.org/ 10.1049/iet-cdt.2017.0018

[13] Kaur, S., Chaudhary, G., Dinesh Kumar, J., Pillai, M. S., Gupta, Y., Khari, M., ... & Parra Fuente, J. (2022). Optimizing fast fourier transform (FFT) image compression using intelligent water drop (IWD) algorithm. https://doi.org/10.9781/ijimai.2022.01.004

[14] Shawahna, A., Haque, M. E., & Amin, A. (2019). JPEG image compression using the discrete cosine transform: an overview, applications, and hardware implementation. arXiv preprint arXiv:1912.10789. https://doi.org/10.48550/arXiv.1912.10789

[15] Hassan, S. M., Hamza, G. G., & Zekry, A. (2021). Simulation of LTE-Advanced Downlink Physical Layer Transceiver. Simulation, 7(37). https://doi.org/10.5120/cae2021652889

[16] Fadhil, M. J. (2018). High Rate Data Processing System of 6x6 MIMO_OFDM Using FPGA Technique with Spatial Algorithm. Engineering and Technology Journal, 36(7), 723-732. https://doi.org/10.30684/etj.36.7A.4

[17] Jallouli, K., Mazouzi, M., Ahmed, A. B., Monemi, A., & Hasnaoui, S. (2018, December). Multicore MIMO-OFDM LTE Optimizing. In 2018 International Conference on Internet of Things, Embedded Systems and Communications (IINTEC) (pp. 166-170).

https://doi.org/10.1109/IINTEC.2018.8695290

[18] Knopp, T., Boberg, M., & Grosser, M. (2023). NFFT. jl: Generic and Fast Julia Implementation of the Nonequidistant Fast Fourier Transform. SIAM Journal on Scientific Computing, 45(3), C179-C205. https://doi.org/10.48550/arXiv.2208.00049

[19] Orenes-Vera, M., Sharapov, I., Schreiber, R., Jacquelin, M., Vandermersch, P., & Chetlur, S. (2023, June). Wafer-Scale Fast Fourier Transforms. In Proceedings of the 37th International Conference on Supercomputing (pp. 180-191). https://doi.org/10.1145/3577193.3593708

[20] Onimus, F., Gelebart, L., & Brenner, R. (2022). Polycrystalline simulations of in-reactor deformation of recrystallized Zircaloy-4 tubes: Fast Fourier Transform computations and mean-field self-consistent model. International Journal of Plasticity, 153, 103272. https://cea.hal.science/cea-03758977

[21] Shen, Y. L., & Wai, R. J. (2022). Fast-Fourier-transform enhanced progressive singular-value-decomposition algorithm in double diagnostic window frame for weak arc fault detection. IEEE Access, 10, 39752-39768. https://doi.org/10.1109/ACCESS.2022.3165793

[22] Pal Nandi, B., Jain, A., Tayal, D. K., & Narang, P. A. (2022). High performing sentiment analysis based on fast Fourier transform over temporal intuitionistic fuzzy value. Soft Computing, 1-15. https://doi.org/10.1007/s00500-021-06444-3

[23] Kulkarni, P., Hogade, B. G., & Kulkarni, V. (2019). Designing of radix-2 butterfly for digital signal processor for FFT computation. In Information and Communication Technology for Intelligent Systems: Proceedings of ICTIS 2018, Volume 2 (pp. 603-610). Springer Singapore. https://doi.org/10.1007/978-981-13-1747-7_59

[24] Nirmala, N., & Sumathi, S. (2019). An area-efficient FFT processor for the OFDMA transceiver communication system. Concurrency and Computation: Practice and Experience, 31(14), e4875. https://doi.org/10.1002/cpe.4875

[25] Bahoura, M. (2020, August). Multirate Hardware of Architecture for Real-Time Fourier Transform Analysis/Synthesis. In 2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS) (pp. 1096-1099). IEEE. https://doi.org/10.1109/MWSCAS48704.2020.9184635

[26] Singhal, A., Goen, A., & Mohapatrara, T. T. (2017). Design and implementation of fast fourier transform (FFT) using VHDL code. International Journal of Emerging Research in Management & Technology, 6(8), 268-272. https://doi.org/10.23956/ijermt.v6i8.150

[27] Fankhauser, E., & Wassner, J. (2018, October). FPGA Implementation of a Multi-Channel Continuous-Throughput FFT Processor. In 2018 IEEE International Workshop on Signal Processing Systems (SiPS) (pp. 181-186). IEEE. https://doi.org/10.1109/SiPS.2018.8598373

[28] Hsiao, V., Nau, D., & Dechter, R. (2022, May). Fast Fourier Transform Reductions for Bayesian Network Inference. In International Conference on Artificial Intelligence and Statistics (pp. 6445-6458). PMLR. https://doi.org/PMLR 151:6445-6458, 2022.

[29] Joshi, A., Gupta, D. A., & Jaipuriyar, P. (2019). Implementation of low complexity FFT, ADC and DAC blocks of an OFDM transmitter receiver using Verilog. Journal of Information Processing Systems, 15(3), 670-681. https://doi.org/10.3745/JIPS.03.0119

[30] Hassan, S. M., Zekry, A., Bayomy, M. A., & Gomah, G. (2013). Software Defined Radio Implementation of LTE Transmitter Physical Layer. International Journal of Computer Applications, 74(8), 41-46. https://doi.org/10.5120/12909-0065

[31] Hassan, S. M., & Zekry, A. (2015). FPGA implementation of LTE downlink transceiver with synchronization and equalization. Communications on Applied Electronics, 2(2), 1-11. https://doi.org/ 10.5120/cae-1663

[32] Hassan, S. M., & Zekry, A. (2017). FPGA implementation of LTE-advanced downlink physical layer transceiver. International Journal of Electronics & Communication Technology (IJECT), 8(2), 9-18. https://doi.org/IJECT/82/1/A-480

[33] Sunseri, J., Slepian, Z., Portillo, S., Hou, J., Kahraman, S., & Finkbeiner, D. P. (2023). sarabande: 3/4 point correlation functions with fast Fourier transforms. RAS Techniques and Instruments, 2(1), 62-77. https://doi.org/10.1093/rasti/rzad003

[34] Tsoeunyane, L., Winberg, S., & Inggs, M. (2018). Automatic configurable hardware code generation for software-defined radios. Computers, 7(4), 53. http://dx.doi.org/10.3390/computers7040053

[35] Morales-Velazquez, L., & Guillén-García, E. (2018, November). FPGA Real-time FFT Portable Core, Design and Implementation. In 2018 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC) (pp. 1-7). IEEE. https://doi.org/10.1109/ROPEC.2018.8661404

[36] Šušteršič, T., & Peulić, A. (2019). Implementation of face recognition algorithm on field programmable gate array (FPGA). Journal of Circuits, Systems and Computers, 28(08), 1950129. https://doi.org/10.1142/S0218126619501299

[37] Sundararajan, D. D., & Sundararajan, D. (2021). Fast Computation of the DFT. Digital Signal Processing: An Introduction, 319-347. https://doi.org/10.1007/978

[38] Li, Q., Zuo, D., Feng, Y., & Wen, D. (2024). Research on High-Performance Fourier Transform Algorithms Based on the NPU. Applied Sciences, 14(1), 405. https://doi.org/10.3390/app14010405

[39] Qian, C., & Yucel, A. C. (2020). On the compression of translation operator tensors in FMM-FFT-accelerated SIE simulators via tensor decompositions. IEEE Transactions on Antennas and Propagation, 69(6), 3359-3370. https://doi.org/10.1109/TAP.2020.3030981

[40] Tao, T., Wang, H., & Zhao, K. (2021). Efficient simulation of fully non-stationary random wind field based on reduced 2D hermite interpolation. Mechanical Systems and Signal Processing, 150, 107265. https://doi.org/10.1016/j.ymssp.2020.107265

[41] Arda, S. E., Krishnakumar, A., Goksoy, A. A., Kumbhare, N., Mack, J., Sartor, A. L., ... & Ogras, U. Y. (2020). DS3: A system-level domain-specific system-on-chip simulation framework. IEEE Transactions on Computers, 69(8), 1248-1262. https://doi.org/10.48550/arXiv.2003.09016