ISSN: 2277-9655
Impact Factor: 1.852

IJESRT

INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

Efficient Implementation of 64-Point FFT/IFFT for OFDM on FPGA

Mr. Shreyas D. Deshmukh[*1], Mrs. Deepali Sale[2]
[*1,2] Lecturer, DYPIET, Pimpri, Pune-18, India
shreyasdesh09@gmail.com

Abstract

Orthogonal Frequency Division Multiplexing (OFDM) is a multi-carrier modulation technique which divides the available spectrum into many carriers. OFDM uses the spectrum efficiently compared to FDMA.
With the rapid growth of digital wireless communication in recent years, the need for high-speed mobile data transmission has increased. FPGAs have become key components in the implementation of high performance DSP systems.

The objective of this paper is to design and efficiently implement FFT and IFFT blocks required for a base band OFDM transmitter and receiver on FPGA hardware. IFFT/ FFT blocks are complex to implement and main blocks of OFDM system i.e. it consumes more resources. So, a efficient technique used here in which FFT/IFFT is implemented in such a way that it consumes very less resources. This module of 64-point FFT and IFFT is designed using VHDL programming language. In this work, a pure VHDL design, integrated with some intellectual property (IP) blocks, is employed to implement an OFDM transmitter and receiver. The proposed design is map and test on Xilinx Virtex 5 FPGA and for simulation, synthesis and implementation XILINX ISE 13.1 software is used.

Keywords: Orthogonal Frequency Division Multiplexing (OFDM), Field Programmable Gate Array (FPGA), Fast Fourier Transform (FFT), Quadrature Amplitude Modulation (QAM), VHDL (VHSIC Hardware Description Language).

Introduction

Orthogonal Frequency Division Multiplexing is a special case of multicarrier transmission, where a single data stream is transmitted over a number of lower-rate subcarriers. The main advantage of OFDM is their robustness to channel fading in wireless environment.

OFDM can be seen as either a modulation technique or a multiplexing technique. In OFDM, multiplexing is applied to independent signals but these independent signals are the part of one main signal. In OFDM, the signal itself is first split into independent channels, modulated by data and then re-multiplexed to create the OFDM carrier [1].

OFDM is a technique especially suitable for wireless communication due to its resistance to inter-symbol interference (ISI) and inter-carrier interference (ICI). In single carrier system, if signal get fade or interfered then entire link gets failed where as in multicarrier system, only a small percentage of the subcarriers will be affected.

FFT/IFFT are the complex and important block of OFDM system, it also requires much of the resources. So its efficient implementation regarding power and resources is must. So in this paper for implementation of FFT, very efficient and innovative technique is proposed by Koushik Maharatna, Eckhard Grass, and Ulrich Jagdhold [2] is used. This paper also gives comparison with other implementation techniques and with available IPs for FFT from various vendors.

Basic of OFDM system is discussed in the section II. The concepts like guard band and cyclic prefix are discussed. Also advantages and disadvantages and of OFDM is also discussed in this section.

Section III is Implementation of FFT/IFFT modules. In this section, detail description of FFT/IFFT block is given and how to implement these blocks on FPGA is also explained in this section.
In Section IV various other ideas regarding efficient implementation of FFT/IFFT such as complex multiplication, number representation is discussed. It also gives the comparison of implemented architecture with various other available IPs. Section V outlines the conclusion.

Basic OFDM System

In OFDM, each subcarrier has an integer number of cycles within a given time interval, and the number of cycles by which each adjacent subcarrier differs is exactly one, in time domain. Due to this, the spectrum of each carrier has a null at the center frequency of the other carriers in the system, in frequency domain. This property accounts for orthogonality between the

ISSN: 2277-9655
Impact Factor: 1.852

subcarriers [1]. Because an OFDM receiver, essentially calculates the spectrum values at those points that correspond to the maxima of individual subcarriers, it can demodulate, each subcarrier free from any interference from other subcarriers.

The generation of OFDM signal started from serial to parallel converter. The input data is in serial form and need to convert into parallel format, since QAM (Quadrature Amplitude Modulation) module requires parallel input to process data. These parallel converted data is mapped to appropriate symbol, with the help of amplitude modulation mapping bank. The parallel symbols are transformed from frequency domain into time domain, using IFFT module. Now, the signals are added with a cyclic prefix and converted into serial format, before being transmitted.

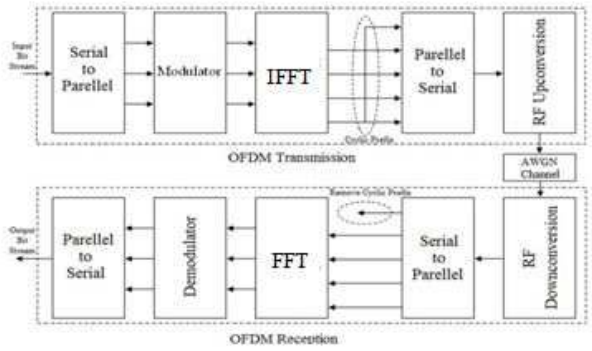


Fig. 1 Basic OFDM system

The received data is in serial format, since FFT input is in parallel, a module which use to converts from serial to parallel is required. Before applying data to the FFT unit, cyclic prefix is removed. Output from FFT is demodulated, using de-mapping module. To demodulate the subcarriers using QAM modulations, reference phase and amplitude of the constellation, on each subcarrier are required. The output of de-modulating module is converted back to serial format, through parallel to serial converter, to get the transmitted data [1].

Guard Band, Cyclic Prefix & Zero Padding

An OFDM system is defined by the IFFT/FFT length-N, the underlying modulation technique (BPSK/QPSK/QAM), supported data rate, etc. The FFT/IFFT length N defines the number of total subcarriers present in the OFDM system. For example, an OFDM system with N=64, provides 64 subcarriers. In reality, not all the subcarriers are utilized for data transmission. Some subcarriers are reserved for pilot carriers (used for channel estimation/equalization and to combat magnitude and phase errors in the receiver) and some are left unused to act as guard band.

OFDM system do not transmit any data on the subcarriers, that are near the two ends of the transmission band (not necessarily at the ends of the bands, implementation may differ). These subcarriers are collectively called guard band. The reservation of subcarriers to guard band, helps to reduce the out of band radiation and thus eases the requirements on transmitter front-end filters.

Advantages and Disadvantages of OFDM

The OFDM transmission scheme has the following advantages:

1. OFDM is an efficient way to deal with multipath; for a given delay spread, the implementation complexity is significantly lower than that of single carrier system with an equalizer.
2. In relatively slow time varying channels, it is possible to significantly enhance the capacity by adapting the data rate per subcarrier according to the signal to noise ratio of the particular subcarrier.
3. OFDM is robust against narrowband interference, because such interference affects the only a small percentage of the subcarrier.
4. OFDM makes single frequency networks possible, which is especially attractive for broadcasting application.

On the other hand, OFDM also has some drawbacks compared to single carrier modulation:

1. OFDM is more sensitive to frequency offset and phase noise.
2. OFDM has a relatively large peak to average power ratio, which tends to reduce power efficiency of power amplifier.

Thus, in this section, we presented the basic OFDM system, advantages and disadvantages of OFDM. In next section, implementation of FFT/IFFT module is explained.

Fast Fourier Transform

The Fast Fourier Transform (FFT) and Inverse Fast Fourier Transform (IFFT) are derived from the main function, namely Discrete Fourier Transform (DFT / IDFT). The idea of using FFT/IFFT instead of DFT/IDFT is that, the computation of the function can be made faster and the number of calculations required in case of FFT is very less, as compared to DFT, which is the main criterion for implementation in the digital signal processing [3]. In DFT, the computation for N-point DFT will be calculated one by one for each point. While for FFT/IFFT, the computation is done simultaneously and this method saves quite a lot of time. Below is the equation showing the DFT and from this, the equation is derived to get FFT function.

ISSN: 2277-9655
Impact Factor: 1.852

The discrete Fourier transform (DFT), A(r) of a complex data sequence B(k) of length N, where $r, k \in \{0,1,2, \ldots, N-1\}$, can be described as

$$A(r) = \sum_{k=0}^{N-1} B(k) e^{-\frac{2\pi rk}{N}} \qquad (3.1)$$

The DFT equation can be re-written into:

$$A(r) = \sum_{k=0}^{N-1} B(k)\, W_N^{rk} \qquad (3.2)$$

The quantity W_N^{rk} is defined as:

$$W_N^{rn} = e^{-\frac{2\pi rk}{N}} = \cos\left(\frac{2\pi rk}{N}\right) - j \sin\left(\frac{2\pi rk}{N}\right)$$

Here the secret lies between DFT and FFT/IFFT, where the function above is called Twiddle Factor. The number of Twiddle Factors used depends on the number of points in FFT/IFFT [3].

64-point FFT

The conventional Cooley–Tukey radix-2 FFT algorithm requires 192 complex butterfly operations, for a 64-point FFT computation. A radix-2 butterfly unit requires one complex multiplication and two complex additions. On top of this butterfly unit, one needs memory to store the complex twiddle factors and complex intermediate data, complicated addressing logic and control circuitry. Combining all these circuit modules, it is expected that the required resources of the entire processor will be quite high.

The fixed point 32-bit word-width 64-point FFT is realized, by decomposing it into a two-dimensional structure of 8-point FFTs [2]. This approach reduces the number of required complex multiplications, compared to the conventional radix-2 64-point FFT algorithm. The complex multiplication operations are realized using dedicated two-input digital multiplier. The processor completes one parallel-to-parallel (i.e., when all input data are available in parallel and all output data are generated in parallel) 64-point FFT computation in 25 cycles. The main motivation of this work is to derive and investigate an alternative architecture for FFT/IFFT computation with moderate silicon area i.e. less use of resources.

We know that, the DFT A(r) of a complex data sequence B(k) of length N, where $r, k \in \{0,1,2, \ldots, N-1\}$, can be described as

$$A(r) = \sum_{k=0}^{N-1} B(k) W_N^{rk} \qquad (3.1)$$

where $W_N^{rk} = e^{-2\pi rk/N}$. Let us consider that N=MT, $r = s + tT$ and $k = l + Mn$, where $s, l \in \{0,1,2,..,7\}$ and $m, t \in \{0,1,2,..,7\}$. Applying these values in (3.1) and simplifying, one gets

$$A(s+Tt) = \sum_{l=0}^{M-1} W_M^{lt} \left[W_{MT}^{sl} \sum_{m=0}^{T-1} B(l + Mn)\, W_T^{sm} \right] \qquad (3.3)$$

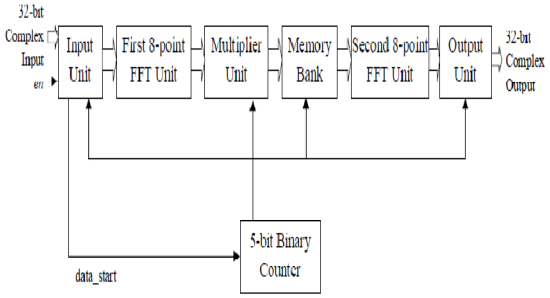
Equation (3.3) means that it is possible to realize the FFT of length N by first decomposing it into one M and one T-point FFT where N=MT, and then combining them. This essentially results in a two dimensional structure instead of a one-dimensional structure of FFT. Now considering M = T = 8, one may formulate the 64-point FFT as

$$A(s+8t) = \sum_{l=0}^{7} \left[W_{64}^{sl} \sum_{m=0}^{7} B(l + 8m)\, W_8^{sm} \right] W_8^{lt} \qquad (3.4)$$

Equation (3.4) suggests that, it is possible to express the 64-point FFT in terms of a two dimensional structure of 8-point FFTs plus 64 complex inter-dimensional constant multiplications. However, since $s, l \in \{0,1,2,..,7\}$, the number of required nontrivial complex multiplications is 49. At first, appropriate data samples (every eighth data of the incoming data sequence) undergo an 8-point FFT computation, followed by eight multiplications with the inter-dimensional constants or twiddle factors (W_{64}^{sl}). Eight such computations are needed to generate a full set of 64 intermediate data, which once again, undergo a second 8-point FFT operation with the appropriate data ordering. As in the case of first 8-point FFT, again eight such computations are required. Proper reshuffling of the data coming out from the second 8-point FFT generates the final output of the 64-point FFT.

The IFFT can be performed by first swapping the real and imaginary parts of the incoming data at the primary input, then performing the forward FFT on them and once again swapping the real and imaginary parts of the data at the output. This method, allows to perform the FFT and IFFT, without changing any of the internal coefficients.

Architecture of 64-Point FFT/IFFT



ISSN: 2277-9655
Impact Factor: 1.852

Fig. 2 Block Diagram of 64-point FFT

The block diagram of the 64-point FFT/IFFT processor derived from (3.4) is depicted in Fig. 2. It consists of an input unit, two 8-point FFT units, a multiplier unit, an internal memory block, an output register bank and a 5-bit binary counter that acts as the master controller for the entire architecture.

Input Unit:

The Input Unit, consists of an register bank (reg(0 to 63)), having 32-bit word length that can store 64 complex data. Upper 16 bits of each register are used to store real part and lower 16 bits for imaginary part of the complex number. The input unit is equipped with two single-bit signals i.e. en and data_start. The assertion of the en signal indicates the presence of a serial valid data stream, at the input of the register bank and subsequently, the input unit starts its operation. The en signal remains at logic 1 for the next 76 cycles after its assertion. After assertion of the en signal, at every clock cycle, the input data are serially inputted only at the 63rd position of the input register bank (reg(63)) and in the successive clock cycles the complex data inside the register bank having index i is shifted to the (i-1) th position where iϵ{0,1,2,…,63}. And the data_start signal is used to start the control unit i.e. 5-bit binary control counter. With the assertion of data_start signal, control unit starts the counting and control the various processes.

The input register bank has eight complex 32-bit fixed hard-wired outputs, corresponding to the register position index 8j, where jϵ{0,1,2,…,7}. When the input register bank is completely full, the appropriate data (a data octet consisting of every eighth data starting with index position 0), is treated as the input to the 8-point FFT as stated in (3.4). This data octet, in the input buffer automatically gets self-aligned with the hard-wired outputs and is delivered to the first 8-point FFT unit. In the next cycle, the same procedure is executed once again because of the shifting of the ith data sample to the (i-1)th sample position.

If this data shifting scheme were not deployed, a parallel multiplexing scheme for all the 64 complex input data to the 8-point FFT input would be needed. This would result in massive multiplexing and a large number of global connections. With the present scheme the data multiplexing and the number of global connections are substantially reduced.

Multiplier Unit:

As stated in Section 3.4.2, 49 nontrivial inter-dimensional constants are to be multiplied to the intermediate results coming out from the first 8-point FFT unit. However, a close observation of these constants reveals that only nine sets of them are unique. They are (1,0), (0.995178, 0.097961), (0.980773, 0.195068), (0.956909, 0.290283), (0.923828, 0.382629), (0.881896, 0.471374), (0.831420, 0.555541), (0.773010, 0.634338), (0.707092, 0.707092), where, in each set, the first entry corresponds to the cosine function (the real part) and second one corresponds to the sine function (the imaginary part) in the expansion of W_{64}^{sl}.

The entire inter-dimensional constant multiplication operation can be carried out using only these nine sets of constants by appropriate swapping of their real and imaginary parts and choosing the appropriate sign. However, the first set of these constants is trivial (1, 0). Thus, in practice, there are eight sets of nontrivial constants required for carrying out the inter-dimensional constant multiplication operation. The implication is that, we require a storage space for only these eight sets of constants instead of 49. Thus, compared to the conventional DIF FFT algorithm, significantly less storage space in this scheme is needed.

On the other hand, because of the full parallel implementation of the first 8-point FFT unit, the respective computation can be carried out in a single clock cycle, which provides a significant leverage in the overall computation time. In the final design of the multiplier unit, eight such complex multiplier units corresponding to the eight sets of the inter-dimensional constants are placed in parallel as shown in Fig. 3 as Const1,…,Const8. Using this arrangement, theoretically, one needs to spend only eight clock cycles all together to finish the entire operation. However, in our case, this actually results in a requirement of 12 clock cycles. The four additional clock cycles come from the fact that, in the case of some of multiplication operations, the same constant needs to be reused.

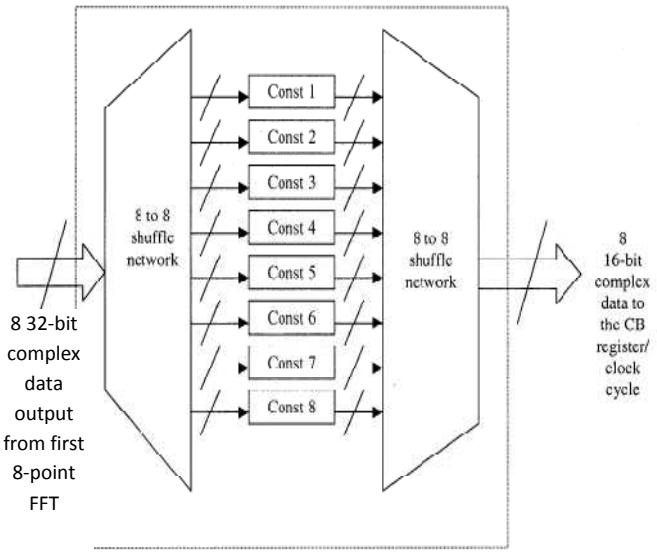


Fig. 3 Block Diagram of Multiplier Unit

ISSN: 2277-9655
Impact Factor: 1.852

In Table I, for the purpose of simplicity, we assume that the first set of data arrives from the 8-point FFT at zeroth time instant. At different time instants, the constants to be used for the multiplication, with the incoming data are indicated by 1, whereas 0 indicates the unused constants. At the time instant T=0, effectively no multiplication operation is needed, as the zeroth block of data (i.e., 8-point FFT output at T=0) from the 8-point FFT has to be multiplied by (1, 0) and therefore, none of the constants are used. The processing of first, third, fifth, and seventh blocks of data (i.e. 8-point FFT output at T=1, 3, 5 and 7) requires one clock cycle each where all constants except const8 are involved in the multiplication process. On the other hand, the processing of second and sixth block of data (i.e., 8-point FFT output at T=2 and 6) requires two cycles each. This is due to the fact that const2, const4 and const6 are reused two times, in successive clock cycles for the complex multiplication operation. For processing the fourth block of data, one needs to spend four clock cycles as const4 is reused four times, whereas const8 is reused two times.

Table I. Utilization of the different constants during the 49 complex multiplication operations

Con s-tant	Block data from 8 point FFT	0th	1st	2nd		3rd	4th				5th	6th		7th
	Time instance	0	1	2	3	4	5	6	7	8	9	10	11	12
const1 (0.9951-j0.0980)		'0'	'1'	'0'	'0'	'1'	'0'	'0'	'0'	'0'	'1'	'0'	'0'	'1'
const2 (0.9807-j0.1951)		'0'	'1'	'1'	'1'	'1'	'0'	'0'	'0'	'0'	'1'	'0'	'0'	'1'
const3 (0.9569-j0.2902)		'0'	'1'	'0'	'0'	'1'	'0'	'0'	'0'	'0'	'1'	'0'	'0'	'1'
const4 (0.9238-j0.3827)		'0'	'1'	'1'	'1'	'1'	'1'	'1'	'1'	'1'	'1'	'1'	'1'	'1'
const5 (0.8819-j0.4714)		'0'	'1'	'0'	'0'	'1'	'0'	'0'	'0'	'0'	'1'	'1'	'1'	'1'
Const6 (0.8314-j0.5550)		'0'	'1'	'1'	'1'	'1'	'0'	'0'	'0'	'0'	'1'	'1'	'1'	'1'
const7 (0.7730-j0.6343)		'0'	'1'	'0'	'0'	'1'	'0'	'0'	'0'	'0'	'1'	'0'	'0'	'1'
const8 (0.7071-j0.7071)		'0'	'0'	'1'	'0'	'0'	'1'	'1'	'0'	'0'	'0'	'1'	'0'	'0'

Thus, the data coming out of the first 8-point FFT block at even time instants has to be kept for more than one clock cycle until the multiplication of the full set is completed. A straight forward strategy to do this is to suspend the operation of the 8-point FFT unit, during those clock cycles. This also implies a suspension of downward shifting of the data in the input unit, for those clock cycles. However, upon completion of the complex multiplication for the respective set of 8-point FFT data, the downward shifting of the data in the input register bank and the 8-point FFT operation resumes once again. Apart from these eight sets of constants, the multiplier unit also has two 8-to-8 32-bit complex shuffle networks at its input and output, respectively, as shown in Fig. 3. The input shuffle network, routes the data from the first 8-point FFT unit to the appropriate constants (const1,…,const8) and the output shuffle network maps the multiplied data to the appropriate index position of the Internal Memory Block MB.

Internal Memory Block (MB):

The MB is used for temporary storage of the 64 complex data, coming from the multiplier unit. Similar to the input unit, it has eight hard-wired outputs corresponding to the registers having the positional indices 8j where jϵ{0,1,2,…,7}. These outputs are directly connected to the input of the second 8-point FFT unit.

Output Unit:

For the output unit, we follow the same strategy as with the input unit. The Output Unit, consists of an register bank (reg(0 to 57)), having 32-bit word length that can store 57 complex data. Here, the i th data coming from the second 8-point FFT unit is directly mapped to the (i-1) th position of the output unit (where iϵ{0,1,2,…,7}) by hard-wired connection. The final output in serial form is taken from the zeroth position of the output unit as soon as the first data arrives. At every cycle, as the new data arrives from the second 8-point FFT unit at the (8i) th positions of the output unit, the old data corresponding to those positions are shifted downwards by one position and the data output mechanism proceeds in the same way.

The Control Mechanism:

The controller for the overall architecture is a simple 5-bit binary counter. The counter starts counting from 0, with the assertion of the signal data_start from the input unit, when the 63rd position of the input register bank is filled. At count number 25, the first output data is available at the zeroth position of the output register bank. All required internal computation is completed and the complete set of output data is stored the output register bank when the count of the master control counter reaches 31. At this point the master control counter is reset to zero and can be reactivated when the

next set of input data fills the 63rd position of the input register bank. However, in the meantime, the output control counter of the output unit controls the serial data output mechanism.

Other Points Related to Implementation of Transform

Complex Multiplication

The most costly part of the FFT is the complex multiplication. By general method, we need 4 simple multiplications and 2 simple additions for 1 complex multiplication. That is why we need an efficient solution to execute the multiplication. First, we separate the complex numbers into real part and imaginary part:

$$W = Wr + jWi$$
$$A = Ar + jAi \quad \text{where } (j^2 = -1).$$

Then complex multiplication of A and W can be done as follow:

$$W.A = (Wr + jWi)(Ar + jAi)$$
$$= Ar.(Wr + Wi) - Wi.(Ar + Ai)$$
$$+j(Ar.(Wr + Wi) + Wi.(Ai - Ar))$$

As W does not change, we do not have to calculate $(Wr + Wi)$ at runtime. We have to calculate $Wri = (Wr + Wi)$ once and save it in memory. Then we get

$$W.A = Ar.Wri - Wi.(Ar + Ai)$$
$$+j(Ar.Wri + Wr.(Ai - Ar))$$

and only three real valued multiplications and three additions/subtractions are required. In this method, we require one more addition instead of multiplication. But as resources require for addition as compared to multiplication is very less and also multiplication is very power hungry process, hence this method is efficient as compared to general method.

Number Representation

For the number representation, a fixed point 16-bitword-width scheme is used. For floating point representation of number, 32 bits are required. We can avoid use of such a large number of bits, by using fixed point representation of number.

The twiddle factor W_N^{rk} is complex, with the magnitude of the real part and the imaginary part of W is between zero and one. The twiddle factors, with which we have to performed complex multiplication are $W^1 = 0.707 - j\,0.707$, $W^3 = -0.707 - j\,0.707$ and $W^2 = -1j$. In the proposed implemented algorithm, we multiply the other twiddle factors by 256, i.e. in digital logic arithmetic, left-shift the number by 8 and round-up the number. So, finally the twiddle factor becomes 181-j181 and -181- j181. Now, with this new twiddle factor values, we perform complex multiplication of butterfly unit output. Thus, if before complex multiplication, 8 bits are required for representation of real or imaginary number (butterfly unit output), then after complex multiplication, for real or imaginary number representation, 16 bits are required. This extra requirement of 8 bits is because of, multiplication of twiddle factor by 256. Original multiplication answer is obtained, by dividing the number by 256 i.e. right shifting the number by 8.

Main Features and Comparison

From (3.4), one can infer that a complete 64-point FFT computation can be carried out, using 49 nontrivial complex multiplication with the inter-dimensional constants, excluding 8-point FFT which need 4 real multiplication. On the other hand, the number of nontrivial complex multiplications for the conventional 64-point radix-2 DIT FFT is 66. Thus, the present approach results in a reduction of about 24% for complex multiplication compared to that required in the conventional radix-2 64-point FFT. By the idea proposed in section 4.2, number of simple multiplication required in every complex multiplication is only 3 instead of 4. So, it also results in 25% resource saving, in every complex multiplication. This reduction of arithmetic complexity further enhances the scope for realizing a 64-point FFT processor with less resource.

The algorithm-to-architecture mapping in the present design was done with the aim to reduce multiplexing and attendant global wirings. The strategy of downward shifting of the data in conjunction with the self-alignment of it, to the fixed hard-wired connections at the input and output register bank, effectively reduces the multiplexing and global wiring compared to the conventional implementation, by a factor of 64 and 8 at the input and output of the input unit and by a factor of 8 and 64 at the input and output of the output unit. This massive reduction of signal multiplexing and global wiring implies a better utilization of silicon area, reduction of routing overhead, and lower power consumption. The effectiveness of the algorithm-to-architecture mapping methodology adopted here can be better appreciated considering the following comparison.

The performance of the processor has been compared with some commercially available 64-point FFT/IFFT IP cores. This is shown in Tables II. It is evident from Table II that the proposed processor requires the smallest number of clock cycles compared to all other available commercial IP cores, and thus, is expected to be less power hungry than the others.

Table II. The performance comparison of the proposed FFT/IFFT processor with the commercially available 64-point FFT/IFFT IP cores

IP Core	Number of Cycle
Xilinx	192
Altera	112
Proposed	25

ISSN: 2277-9655
Impact Factor: 1.852

Table III gives device utilization summery for 64-point FFT. The use of resources of proposed technique is also compared with other techniques [4], [5]. It is observed that some technique requires more number of multiplier [4] and also some more other resources [4] [5]. Thus, proposed technique is an efficient way to implement 64-point FFT. But this implementation technique requires more BRAM.

Table III. Device Utilization Summery of Proposed Technique for 64-point FFT

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	2989	69120	4%
Number of Slice LUTs	5080	69120	7%
Number of fully used LUT-FF pairs	1967	6102	32%
Number of bonded IOBs	48	640	7%
Number of Block RAM/FIFO	64	148	43%
Number of BUFG/BUFGCTRLs	1	32	3%
Number of DSP48Es	32	64	50%

Conclusion

The main aim of the paper is to implement the FFT/IFFT blocks of OFDM system on FPGA using VHDL language. The OFDM system is designed on Xilinx project navigator for different number of subcarrier i.e. FFT and IFFT points. FFT and IFFT are important and complex blocks in OFDM system which consumes lots of resources. So, its efficient implementation in terms of available resources is must. In this project, design for efficient implementation of FFT and IFFT modules is proposed and implemented. Design is implemented on Virtex 5 FPGA using Xilinx synthesis tool, tested for different data patterns and results are compared with theoretical expected results. By manually entering transmitted data at receiver, the recovery of the original required data is done. The results are matching with expected results.

References

[1] R.Van Nee, R. Prasad Publication by Artech House,"OFDM for wireless Multimedia Communications.", e-book.
[2] Koushik Maharatna, Eckhard Grass, and Ulrich Jagdhold, "A 64-Point Fourier Transform Chip for High-Speed Wireless LAN Application Using OFDM"
[3] J. W. Cooley and J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," Mathematics of Computation, vol. 19, no. 90, pp. 297–301, 1965.
[4] Wang Xudong, Liu Yu, "Special-purpose computer for 64-point FFT based on FPGA"
[5] J. M. Rudagi et.al., "An Efficient 64-point pipelined FFT Engine"