Design and VHDL Implementation of 64-Point FFT using Two Structure 8-Point FFT/IFFT

Patel Kajalben Ramanbhai  
PG Student  
Department of Electronics & Telecommunication Engineering  
D. N. Patel Institute of Technology Shahada, Maharashtra India

Prof. S. P. Patil  
Assistant Professor  
Department of Electronics & Telecommunication Engineering  
D. N. Patel Institute of Technology Shahada, Maharashtra India

Abstract

In this paper, we present a novel fixed-point 32-bit word-length Radix-2 64-point FFT processor. The 64-point FFT is realized by decomposing it into a two-dimensional structure of 8-point FFTs. 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 shift-and-add operations. Thus, the processor does not use a two-input digital multiplier. It also does not need any RAM or ROM for internal storage of coefficients. The Fast Fourier Transform (FFT) is one of the rudimentary operations in field of digital signal, image processing and FFT processor is a critical block in all multicarrier systems used primarily in the mobile environment. Fast Fourier transform (FFT) is an efficient implementation of the discrete Fourier transform (DFT). FFT blocks are complex to implement and it consumes more resources. So, a efficient technique used here in which FFT is implemented in such a way that it consumes very less resources. This module of 64-point FFT is designed using VHDL programming language. In this work, a pure VHDL design, integrated with some intellectual property (IP) blocks and simulation, synthesis and implementation XILINX ISE 13.2 software is used.

Keywords: FFT, IP Core and VHDL

I. INTRODUCTION

Fourth-generation wireless and mobile systems are currently the focus of research and development. Broadband wireless systems based on orthogonal frequency division multiplexing (OFDM) will allow packet-based high-data-rate communication suitable for video transmission and mobile Internet applications. Fast Fourier transform (FFT) is one of the key components for various signal processing and communications applications such as software defined radio and OFDM. A typical FFT processor is composed of butterfly calculation units, an address generator and memory units[9]. This study is primarily concerned with improving the performance of the address generation unit of the FFT processor by eliminating the complex critical path components. Pease observed that the two data addresses of every butterfly differ in their parity. Parity check can be realized by modulo-r addition in hardware[6].

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[7]. 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.

The rest of this paper is organized as follows. Section II is Implementation of FFT modules. In this section, detail description of FFT block multiplier Block, internal memory block is given and how to implement these blocks on FPGA is also explained in this section.

In Section III 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. In Section IV experimental result and discussion Section V outlines the conclusion.

II. 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 [18]. 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 [10].

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
The DFT equation can be re-written into:

\[ A(\tau) = \sum_{k=0}^{N-1} B(k) W_N^{\tau k} \]  

(2)

The quantity is defined as:

\[ W_N^{\tau k} = e^{-\frac{2\pi i k \tau}{N}} = \cos\left(\frac{2\pi \tau k}{N}\right) - j \sin\left(\frac{2\pi \tau k}{N}\right) \]

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].

### A. 64-point FFT

The conventional Cooley–Turkey 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, serial-to-parallel and parallel-to-serial converters at the inputs and outputs, respectively, 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 [13]. 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

\[ A(\tau) = \sum_{k=0}^{N-1} B(k) W_N^{\tau k} \]

(3)

Where \( W_N = e^{-\frac{2\pi i k \tau}{N}} \). Let us consider that \( N=MT, \tau = s + \tau t \text{and} k = i + Mn \), where \( s, i \in \{0,1,2,\ldots,T\} \text{ and } m, n \in \{0,1,2,\ldots,M\} \).

Applying these values in (3) and simplifying, one gets

\[ A(\tau + \tau t) = \sum_{s=0}^{T-1} \sum_{i=0}^{T-1} \sum_{m=0}^{M-1} \sum_{n=0}^{M-1} B(i + Mn) W_N^{s i} W_N^{m n} \]

(4)

Equation (4) 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

\[ A(\tau + 8\tau t) = \sum_{s=0}^{7} \sum_{i=0}^{7} \sum_{m=0}^{7} \sum_{n=0}^{7} B(i + 8n) W_N^{s i} W_N^{m n} \]

(5)

Equation (5) 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 \( s, i \in \{0,1,2,\ldots,7\} \), 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 \( W_N^{s i} \).

However, the number of nontrivial multiplications required, for each set of 8-point FFT results is actually seven since the zeroth term of the first 8-point FFT, gets multiplied with 1. 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 (every eighth data forms an input data set, for the second 8-point FFT). 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.

### B. Architecture of 64-Point FFT

The block diagram of the 64-point FFT/IFFT processor derived from (5) is depicted in Fig. 1. It consists of an input unit, two 8-point FFT units, a multiplier unit, an internal memory block [2], an output register bank and a 5-bit binary counter that acts as the master controller for the entire architecture. All sub module Discuss step by step.
1) **Input Unit**

The Input Unit, consists of a 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 iє{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 jє{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 (5). 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 as shown in fig 3. 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 [12].

2) **Multiplier Unit**

As stated in this Section, 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 $e^{j\pi/8}$.

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 [2]. 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. 2 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.

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.

---

**Fig. 1: Block Diagram of 64-point FFT**
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 straightforward 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. 2.

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.

3) 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 j ∈ {0,1,2,…,7}. These outputs are directly connected to the input of the second 8-point FFT unit.

4) 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 i ∈ {0,1,2,…,7}) by hard-wired connection [14]. 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.

5) 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.

III. OTHER POINTS RELATED TO IMPLEMENTATION OF TRANSFORMER

A. 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:

\[
W = W_r + jW_i
\]
\[
A = A_r + jA_i
\]

Where \( j^2 = -1 \)

Then complex multiplication of \( A \) and \( W \) can be done as follow:
\[ W.A = (W_r + jW_i)(A_r + jA_i) = A_r(W_r + W_i) - W_i(A_r + A_i) + j(A_r(W_r + W_i) + W_i(A_i - A_r)) \]

As \( W \) does not change, we do not have to calculate \((W_r + W_i)\) at runtime. We have to calculate \(W_{ri} = (W_r + W_i)\) once and save it in memory. Then we get

\[ W.A = A_r W_{ri} - W_i(A_r + A_i) + j(A_r W_{ri} + W_r(A_i - A_r)) \]

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 multiplication as compared to addition is very less and also multiplication is very power hungry process, hence this method is efficient as compared to general method.

### B. Number Representation

For the number representation, a fixed point 16-bit word-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 \( W_{rk} \) 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

\[ W^4 = 0.707 - j 0.707, W^3 = -0.707 - j 0.707 \text{ and } W^2 = -1j. \]

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.

### IV. EXPERIMENT AND RESULT DISCUSSION

#### A. Input unit

The **Input Unit** consists of 8 register banks and each register bank is of 8 registers of 32 bit. All these register banks are arranged one above the other, in such a way that, the output of one unit is given as input to the other register bank below it. These register banks are made from a Distributed RAM Shift Register and at every clock data is shifted one place down. Also, the output of each register bank is given to the output of **Input Unit**, which is treated as the input to the 8-point FFT unit. This is clearly shown in Fig. 3.

![Input Unit Diagram](image)

**Fig. 3:** Input Unit

The output of Input Unit is applied to the first 8-point FFT block. Design of 8-point FFT is already explained in the previous section, in details. Fig. 4 shows the simulated output of first 8-point FFT block, when input to the Input Unit is 1+1i,........64+64i
in sequence. In this way data is processed block by block and the simulated output of each block is shown in Fig. 6 and Fig. 7. Fig. 6 shows the output of Memory Block and Fig. 7 shows the output of second 8-point FFT block. Output Unit gives output in serial form and thus its output is difficult to observe in diagram form.

![Fig. 4: Simulation result of first FFT block](image)

B. Identify the Headings

Headings, or heads, are organizational devices that guide the reader through your paper. There are two types: component heads and text heads.

Component heads identify the different components of your paper and are not topically subordinate to each other. Examples include ACKNOWLEDGMENTS and REFERENCES, and for these, the correct style to use is “Heading 5.” Use “figure caption” for your Figure captions, and “table head” for your table title. Run-in heads, such as “Abstract,” will require you to apply a style (in this case, italic) in addition to the style provided by the drop down menu to differentiate the head from the text.

Text heads organize the topics on a relational, hierarchical basis. For example, the paper title is the primary text head because all subsequent material relates and elaborates on this one topic. If there are two or more sub-topics, the next level head (uppercase Roman numerals) should be used and, conversely, if there are not at least two sub-topics, then no subheads should be introduced. Styles named “Heading 1,” “Heading 2,” “Heading 3,” and “Heading 4” are prescribed.

C. Multiplier Unit

![Fig. 5: Simulation Result of multiplier block](image)

Fig. 5 shows the output of multiplier unit, when the output of first 8-point FFT block (as shown in Fig. 4) is applied to the multiplier unit. Multiplier Unit consist of Input Data Shuffle Network which has 8 outputs of 32 bit each, multiplication block in which multiplication with 8 constants is carried out and Output Data Shuffle Network. For implementation of Data Shuffle Network, we use Case Select Statement and perform required operations on the Input and Output data of Multiplier Unit. These required operations must have to perform on proper clock cycle and these operations include 2’s complement of number, exchanging real and imaginary part of complex number etc. In multiplication block, we have to perform 8 complex multiplications with 8 complex constants in every clock cycle. By general method, it requires 32 simple multiplications and 16 simple additions. Thus, total number of simple multiplication in multiplier unit is 24 results in 25% of resources saving.
D. Internal Memory Block (MB)

As shown in Fig. 6 Internal memory Block consist of 8 Block RAM units are used, for the construction of MB. Each Block RAM unit used here, is a simple dual port RAM, has 256 bit input and 32 bit output. First multiplier block output i.e. 256 bits, have to store in a Block RAM 1, at a certain clock cycle and at next clock cycle, new computed multiplier block output, have to store in Block RAM 2 [13]. In this way, we have to fill all 8 Block RAMs. We have to wait, until all 8 Block RAMs are completely filled. Then, at a given time, the input to the second FFT unit must be a 32 bit value at index 0 of all 8 block RAMs, at next clock cycle value at index 1 must be assigned to the second FFT block.

Fig. 6: Output of Internal Memory Block

E. Output Unit

The Output Unit has an input of 256 bit and 32 bit output. After the first data arrives from FFT unit, at every clock cycle, Output Unit gives a 32-bit complex number as output. It consists of 8 register banks, out of which 1 bank is of 2 registers, 1 bank is of 8 registers, 6 banks is of 7 registers, each of 32 bit. The Output Unit also consists of, 8 32-bit 2-input multiplexers. All these register banks are arranged one above the other, in such a way that, the output of one unit is given as input to the other register bank lower to it, through multiplexer. At the top, we have bank of 8 register, below it 6 banks of 7 register and at the bottom bank of 2 register [14]. These register banks are made from a Distributed RAM Shift Register and at every clock data is shift one place down. The input of the Output Unit, with index 7 is treated as the input to the topmost register bank of 8 register and the input with index 6 to 1, to the register banks of 7 register but through the multiplexer. The input with index 0 is applied to the register bank with 2 register, through multiplexer and output of this register bank is connected to the output of output unit. For first 8 clock cycles, after the first data arrives from FFT unit, input to the output unit is connected to the input of register banks through multiplexers. And for the remaining 56 clock cycles, output of upper register bank is connected to the input of lower register bank to it through multiplexers. simulated result of second FFT block as shown in fig.7.

Fig. 7: Output of Second 8-point FFT Block
F. Main Features and Comparison

From (eq.5), 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 [4].

Table II 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 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>
<thead>
<tr>
<th>Logic Utilization</th>
<th>Proposed by Wang Xudong(4) 64-point</th>
<th>Proposed by Asmita Haveliya(5) 32-point</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Slice Registers</td>
<td>2989</td>
<td>3596</td>
</tr>
<tr>
<td>Number of Slice LUTs</td>
<td>5080</td>
<td>5717</td>
</tr>
<tr>
<td>Number of fully used LUT-FF pairs</td>
<td>1967</td>
<td>2615</td>
</tr>
<tr>
<td>Number of bonded IOBs</td>
<td>48</td>
<td>402</td>
</tr>
<tr>
<td>Number of Block RAM/FIFO</td>
<td>64</td>
<td>16</td>
</tr>
<tr>
<td>Number of BUFG/BUFGCTRLs</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Number of DSP48Es</td>
<td>32</td>
<td>40</td>
</tr>
</tbody>
</table>

V. Conclusions

This architecture is based on a decomposition of the 64-point FFT into two 8-point FFTs so that the resulting algorithm-to-architecture mapping is well suited for silicon implementation. The main aim of the paper is to efficient implement the FFT blocks, multiplier block, internal memory block in VHDL language. FFT complex blocks in OFDM system which consumes lots of resources. So, its efficient implementation in terms of available resources is must. This Design done Xilinx synthesis tool, tested for different data patterns and results are compared with theoretical expected results.

REFERENCES

[10] Wen Fan, Chii-sing Choy; “Robust, low complexity and energy efficient downlink baseband receiver design for MB-OFDM UWB system.”, IEEE transactions on circuits and system 1549-8328/26.02 2011 IEEE.
[12] Kitaek Bae, Student Member, IEEE, Jeffrey G. Andrews, Senior Member, IEEE, and Edward J. Powers, Life Fellow, IEEE, “Quantifying an Iterative Clipping and Filtering Technique for reducing PAPR in OFDM”. IEEE Transactions on wireless communications, vol. 9, no. 5, may 2010

[14] Mr. Shreyas D. Deshmukh & Mrs. Deepali sale “efficient implementation of 64-point FFT/IFFT for OFDM on FPGA” IJESRT ISSN: 2277-9655 December 2013.