The Discrete Fourier transform is used to produce frequency analysis of discrete non-periodic signals. The FFT is another method of achieving the same result, but with less overhead involved in the calculations. The representation of finite Fourier Transform is given in \{X(\omega)\} of the sequence of \{x(n)\} is continuous function in frequency domain Fourier Transform, it is not computationally convenient to be represented by the sequence of \{x(n)\}. However, the sequence of \{x(n)\} can be represented by sampling the spectrum \{X(\omega)\}. This frequency domain representation leads to the discrete Fourier Transform (DFT), which is an important algorithm for performing frequency analysis of discrete-time signal.

Application using frequency analysis of discrete-time signals in digital signal processor is the most convenient method especially designed digital hardware (Praxis and Melonakos, 1996 and Schoukens et. Al, 2004). Frequency analysis is performed on a discrete-time signal \{x(n)\} by converting the time-domain sequence to an equivalent frequency-domain presentation. As straightforward representation of the Fourier transform is illustrated in Figure 1.1 (Brigham, 1974). From the figure, it shows the essence of the Fourier transform of a waveform is to decompose or implement the discrete-time signal-processing algorithms and systems it is used to convert the samples in time domain to frequency domain. The Fast Fourier Transform (FFT) is simply a fast (computationally efficient) way to calculate the Discrete Fourier Transform (DFT). The wide usage of DFT’s in Digital Signal Processing applications is the motivation to implement FFT’s. Almost every branch of engineering and science uses Fourier Methods. The words “frequency,” “period,” “phase,” and “spectrum” is important parts of an engineer’s vocabulary.
separate the waveform into a sum of sinusoids of different frequency. Thus, the pictorial representation of the Fourier transforms is a diagram which displays the amplitude and frequency of each of the determined sinusoids.

The Fourier Transform identifies or distinguishes the different frequency sinusoids and their respective amplitudes which combine to form an arbitrary waveform. Mathematically, this relationship is stated as (Bracewell, 1978).

II. FOUNDAMENTALS

Radix-2 FFT algorithm is called the decimation-in-frequency algorithm, is obtained by using the divide-and-conquer approach. To derive the algorithm, we begin by splitting the DFT formula into two summations, one of which involves the sum over the first N/2 data points and the second sum involves the last N/2 data points. Thus we obtain

\[ X(k) = \sum_{n=0}^{N/2-1} x(n) W_N^{nk} + \sum_{n=N/2}^{N-1} x(n) W_N^{nn} \]

Since \[ W_N^{N/2} = (-1)^k \]

Now, let us split (decimate) \( X(k) \) into the even- and odd-numbered samples. Thus we obtain

\[ X(2k) = \sum_{n=0}^{N/2-1} x(n) + x \left( n + \frac{N}{2} \right), \quad k = 0, 1, \ldots, \frac{N}{2} - 1 \]

\[ X(2k + 1) = \sum_{n=0}^{N/2-1} \left[ x(n) - x \left( n + \frac{N}{2} \right) \right], \quad k = 0, 1, \ldots, \frac{N}{2} - 1 \]

Where we have used the fact that \( W_N^2 = W_N^{N/2} \)

The computational procedure above can be repeated through decimation of the \( N/2 \)-point DFTs \( X(2k) \) and \( X(2k+1) \). The entire process involves \( v = \log_2 N \) stages of decimation, where each stage involves \( N/2 \) stages of the type. Consequently, the computation of the N-point DFT via the decimation-in-frequency FFT requires \( (N/2)\log_2 N \) complex multiplications and \( N\log_2 N \) complex additions, just as in the decimation-in-time algorithm.

III. RADIX-4 DIF FFT ALGORITHM

In this paper, two improved radix-4 FFT algorithms have been proposed. It has been shown that by re-indexing a subset of the output samples resulting from the conventional decompositions in the radix-4 FFT algorithms, a substantial reduction in the number of twiddle factor evaluations or accesses to the lookup table can easily be obtained. These savings have been achieved without imposing any additional computational or structural complexity in the algorithms. The basic idea for improved FFT algorithms introduced in this paper can also be applied to other higher radices DIT and DIF FFT algorithms as well as to multidimensional FFT algorithms.

When the number of data points \( N \) in the DFT is a power of 4 (i.e., \( N = 4^v \)), we can, of course, always use a radix-2 algorithm for the computation. However, for this case, it is even more efficient computationally to employ a radix-4 FFT algorithm. Let us begin by describing a radix-4 decimation-in-time FFT algorithm briefly. We split or decimate the N-point input sequence into four subsequence’s,

\[ x(4n), x(4n + 1), x(4n + 2), x(4n + 3), n = 0, 1, \ldots, \frac{N}{4} - 1 \]
Thus the four \( N/4 \)-point DFTs \( F(l,q) \) obtained from the above equation are combined to yield the N-point DFT. The expression for combining the \( N/4 \)-point DFTs defines a radix-4 decimation-in-time butterfly, which can be expressed in matrix form as

\[
\begin{bmatrix}
X(0, q) \\
X(1, q) \\
X(2, q) \\
X(3, q)
\end{bmatrix} =
\begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & -j & -1 & j \\
1 & -1 & 1 & -1 \\
1 & j & -1 & -j
\end{bmatrix}
\begin{bmatrix}
W_{N/4}^0 F(0, q) \\
W_{N/4}^1 F(1, q) \\
W_{N/4}^2 F(2, q) \\
W_{N/4}^3 F(3, q)
\end{bmatrix}.
\]

The radix-4 butterfly is depicted in figure TC.3.9a and in a more compact form in figure TC.3.9b. Note that each butterfly involves three complex multiplications, since \( W_N^0 = 1 \), and 12 complex additions.

By performing the additions in two steps, it is possible to reduce the number of additions per butterfly from 12 to 8. This can be accomplished by expressing the matrix of the linear transformation mentioned previously as a product of two matrices as follows:

\[
\begin{bmatrix}
X(0, q) \\
X(1, q) \\
X(2, q) \\
X(3, q)
\end{bmatrix} =
\begin{bmatrix}
1 & 0 & 1 & 0 \\
0 & 1 & 0 & -1 \\
1 & 0 & -1 & 0 \\
0 & 1 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
W_{N/4}^0 F(0, q) \\
W_{N/4}^1 F(1, q) \\
W_{N/4}^2 F(2, q) \\
W_{N/4}^3 F(3, q)
\end{bmatrix}.
\]
Where we have used the property $W_N^{4kN} = W_N^{5kN}/4$.

Note that the input to each $N/4$-point DFT is a linear combination of four signal samples scaled by a twiddle factor. This procedure is repeated $v$ times, where $v = \log_4 N$.

IV. SPLIT-RADIX FFT ALGORITHMS

An inspection of the radix-2 decimation-in-frequency flow graph shown in figure TC.3.8 indicates that the even-numbered pints of the DFT can be computed independently of the odd-numbered points. This suggests the possibility of using different computational methods for independent parts of the algorithm, with the objective of the number of computations. The split-radix FFT (SRFFT) algorithms exploit this idea by using both radix-2 and radix-4 decomposition in the same FFT algorithm. First, we recall that in the radix-2 decimation-in-frequency FFT algorithm, the even-numbered samples of the $N$-point DFT are given as

$$X(2k) = \sum_{n=0}^{N/2-1} [x(n) + x(n + \frac{N}{2})] W_N^n W_{N/2}^{2nk}, \quad k = 0, 1, \ldots, N/2 - 1$$

A radix-2 suffices for this computation. The odd-numbered samples $\{X(2k+1)\}$ of the DFT require the pre-multiplication of the input sequence with the twiddle factors $W_N^n$. For these samples radix-4 decomposition produces some computational efficiency because the four-point DFT has the largest multiplication-free butterfly. Indeed, it can be shown that using a radix greater that 4 does not result in a significant reduction in computational complexity.

If we use a radix-4 decimation-in-frequency FFT algorithm for the odd-numbered samples of the $N$-point DFT, we obtain the following $N/4$-point DFTs:

$$X(4k + 1) = \sum_{n=0}^{N/4-1} [x(n) + x(n + \frac{N}{4})] W_N^n W_{N/4}^{2kn}$$

$$X(4k + 2) = \sum_{n=0}^{N/4-1} [x(n) + x(n + \frac{N}{4})] W_N^n W_{N/4}^{2kn}$$

$$X(4k + 3) = \sum_{n=0}^{N/4-1} [x(n) + x(n + \frac{N}{4})] W_N^n W_{N/4}^{2kn}$$

Fig.4 shows the flow graph for an in-place 32-point decimation-in-frequency SFFT algorithm.

In this paper, we propose to design an 32-point FFT and find its computation time. Here our goal is to implement Radix-4 64-point FFT in hardware using hardware language (VHDL) here time constrains is measured with the help of Xilinx FPGA (Field Programmable Gate Array). First we consider single butterfly suited for FPGA implementation. Then we had seen the result of complete FFT. From above result we get the time taken by each block in hardware language. Thus, FPGA resources are effectively utilized because the designed structures fitted for FPGA structures.

This paper concentrates on the development of the fast Fourier Transform (FFT), based on Decimation-in-time (DIT) domain, Radix-2 algorithm, Radix-4 Decimation-in-Frequency algorithm this paper uses VHDL as a design entity, and their synthesis by Xilinx Synthesis Tool on Vertex kit has been done. The input of fast Fourier transform has been given by a PS2 KEYBOARD using a test bench and output has been displayed using the waveforms on the Xilinx ISE 10.1. The synthesis results show that the computation for calculating the 32-point Fast Fourier transform is efficient in terms of speed.

V FPGA

The introduction of field programmable gate arrays (FPGAs), has made it feasible to provide hardware for application specific computation design. The changes in designs in FPGA’s can be accomplished within a few hours, and thus result in significant savings in cost and design cycle. FPGAs offer speed comparable to dedicated and fixed hardware systems for parallel algorithm. The 32-point FFT proposed in this paper is been simulated and synthesized using the Xilinx ISE10.1.
VI CONCLUSION

This paper proposes a novel radix-4 FFT processor based on FPGA for WLAN, using Verilog HDL as Hardware description language and Quartus II as Design and Synthesis tool. To achieve high-throughput, pipelined architectures have been used in the butterfly unit and the whole system performance can be greatly improved due to adopting a novel simple address mapping scheme. For radix-2 system, this mapping scheme is better and simpler than most of others. The design is implemented on a FPGA chip. And this pipelined FFT completes a complex 64-point FFT within 2.1μs. The hardware testing result explains that it can meet the requirements of the WLAN.

VII REFERENCES

[1] Alan V. Oppenheim, Ronald W. Schafer and John R. Buck, Discrete-Time Signal Processing, Prentice Hal, second edition, 1999.

[2] Mandeep Singh Balwinder Singh Pawan Verma, Harpreet Kaur, “VHDL implementation of FFT/IFFT blocks for OFDM”, International Conference on Advances in Recent Technologies in Communication and Computing, 2009.

[3] James W. Cooley and John W. Tukey, “An algorithm for the machine calculation of complex fourier series”, Math Comput., pp. 297 – 301, 1965.

[4] Volnei A. Pedroni, Circuit Design with VDHL, MIT Press, ISBN 0-262-16224-5, 2004.

[5] T. Starr, M. Sorbara, J. M. Cioffi and P. J. Silvermann, DSL Advance, Prentice Hall, 2003.

[6] Rd. J.; Ordaz-Moreno A.; Vite-Frias, J. A.; Romero-Troncoso, “VHDL core for 1024-point radix-4 fft computation”, International Conference on Reconfigurable Computing and FPGAs 2005, ReConFig 2005, pp. 4 – 24, 9 2005.

[7] A. Álvarez-Marquina E. Martínez de Icaya C. González- Consejero, V. Rodellar and P. Gonzalez-Vilda, “A portable hardware design of a FFT algorithm”, Latin American Applied Research, 2007.

[8] Randy Yates, “Fixed-point arithmetic: An introduction”, 2009, http://www.digitalsignallabs.com.

[9] David Bishop, “Fixed point package user’s guide”.

[10] Frank Vahid, Digital Design with RTL Design, VHDL and Verilog, John Wiley and Sons, second edition, 2011.

[11] J. A. C. Bingham, “Multicarrier modulation for data transmission: an idea whose time has come,” IEEE Communication Magazine, vol. 28, no. 5, pp. 5-14, May 1990.

[12] J. Palicot and C. Roland, “FFT: a basic function for a reconfigurable receiver,” 10th International Conference on Telecommunications, vol. 1, pp. 898-902, March 2003.