INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH & DEVELOPMENT

ISSN: 2278 – 0211 (Online)

Design Of Radix-4 FFT In VHDL Using Simulink

Shruti Khandelwal
M.Tech.Scholar, ET&T Department , DIMAT Raipur

Prof. Pankaj Gulhane
Lecturer ET&T, ET&T Department , DIMAT Raipur

Abstract:

FFT is suitable for high speed environment because it provides the transfer of data at a very high speed. Main focus of this paper is to design an FFT with the help of MATLAB and simulink along with system generator (SysGen). Such tools take as their input a high-level representation of an application written in MATLAB R2007a and generate RTL (Register Transfer Level) implementation for an FPGA. The RTL code is synthesized using Xilinx Project Navigator XILINX ISE 9.2i and simulated using Model Sim5.8c simulator providing superior performance making it an increasingly preferred choice of many engineers today.

Key words: *FFT, SysGen , MATLAB, Simulink, DFT, Butterfly Algorithm, Xilinx System Generator*

1.Introduction

This paper comprises of FFT i.e. Fast Fourier Transform which is an efficient algorithm to compute the discrete Fourier transform (DFT) and its inverse. There are many distinct FFT algorithms involving a wide range of mathematics, from simple complex-number arithmetic to group theory and number theory. An FFT is a way to compute the same result more quickly: computing a DFT of N points in the naive way, using the definition, takes $O(N^2)$ arithmetical operations, while an FFT can compute the same result in only $O(N \log N)$ operations. The difference in speed can be substantial, especially for long data sets where N may be in the thousands or millions—in practice, the computation time can be reduced by several orders of magnitude in such cases, and the improvement is roughlyproportional to $N / \log(N)$. This huge improvement made many DFT-based algorithms practical; FFTs are of great importance to a wide variety of applications, from digital signal processing and solving partial differential equations to algorithms for quick multiplication of large integers.

The aim of this work is to design and implement the FFT using MATLAB along with simulink. In this work FFT is designed using SysGen after that results are verified and then compared with the previous work which shows the present work provides better performance than the previous work.

1.1. FFT System Model

FFT algorithms depend upon the factorization of N, but there are FFTs with $O(N \log N)$ complexity for all N, even for prime N. Many FFT algorithms only depend on the fact that $e^{-2\pi i/N}$ is an N^{th} primitive root of unity, and thus can be applied to analogous transforms over any finite field, such as number-theoretic transforms. Since the inverse DFT is DFT, the opposite sign . in the exponent and a $1/N$ factor, any FFT algorithm can easily be adapted for it. It has been described as "the most important numerical algorithm of our lifetime".

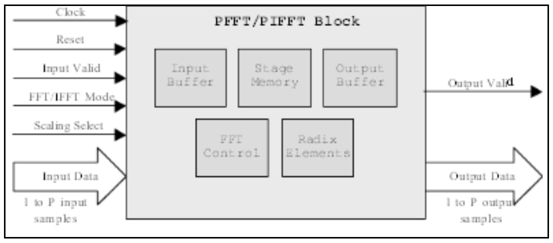

Figure 1: Block diagram of FFT [5]

An FFT computes the DFT and produces exactly the same result as evaluating the DFT definition directly; the only difference is that an FFT is much faster. (In the presence of round-off error, many FFT algorithms are also much more accurate than evaluating the DFT definition directly, as discussed below)

Let x_0, ..., x_{N-1} be complex numbers. The DFT is defined by the formula

$X_k = \sum_{n=0}^{N-1} x_n e^{-2\pi kn/m}$, where k=0 to N-1----(a)[2]

Evaluating this definition directly requires $O(N^2)$ operations: there are N outputs X_k, and each output requires a sum of N terms. An FFT is any method to compute the same results in O(N log N) operations. More precisely, all known FFT algorithms require Θ(N log N) operations (technically, O only denotes an upper bound), although there is no known proof that better complexity is impossible.

To illustrate the savings of an FFT, consider the count of complex multiplications and additions. Evaluating the DFT's sums directly involves N^2 complex multiplications and N(N − 1) complex additions [of which O(N) operations can be saved by eliminating trivial operations such as multiplications by 1]. The well-known radix-2 butterfly algorithm, for N a power of 2, can compute the same result with only (N/2) $\log_2$ N complex multiplies (again, ignoring simplifications of multiplications by 1 and similar) and N $\log_2$N complex additions. In practice, actual performance on modern computers is usually dominated by factors other than arithmetic and is a complicated subject but the overall improvement from $O(N^2)$ to O(N log N) remains. FFT has an extremely high throughput through programmable parallelism of input data samples. It has ultra high speed continuous time operation and is area efficient too. It allows us to relate events in time domain to events in frequency domain. The FFT of the simple sinusoidal wave is given below:

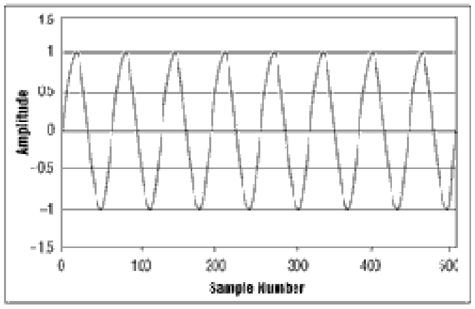

Figure 2(a): Sine wave

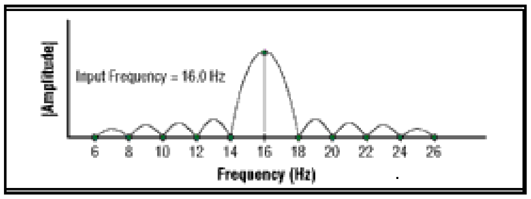

Figure 2(b): FFT of sine wave

1.2.FFT with Butterfly Algorithm

A butterfly is a portion of the computation that combines the results of smaller discrete Fourier transforms (DFTs) into a larger DFT, or vice versa (breaking a larger DFT up into subtransforms). The name "butterfly" comes from the shape of the data-flow diagram in the radix-2 case, as described below. The same structure can also be found in the Viterbi algorithm, used for finding the most likely sequence of hidden states.

Most commonly, the term "butterfly" appears in the context of the Cooley–Tukey FFT algorithm, which recursively breaks down a DFT of composite sizen = rm into r smaller transforms of size m where r is the "radix" of the transform. These smaller DFTs are then combined via size- r butterflies, which themselves are DFTs of size r (performed m times on corresponding outputs of the sub-transforms) pre-multiplied by roots of unity (known as twiddle factors). This is the "decimation in time" case; one can also perform the steps in reverse, known as "decimation in frequency", where the butterflies come first and are post-multiplied by twiddle factors. The butterfly can also be used to improve the randomness of large arrays of partially random numbers, by bringing every 32 or 64 bit word into causal contact with every other word through a desired hashing algorithm, so that a change in any one bit has the possibility of changing all the bits in the large array.

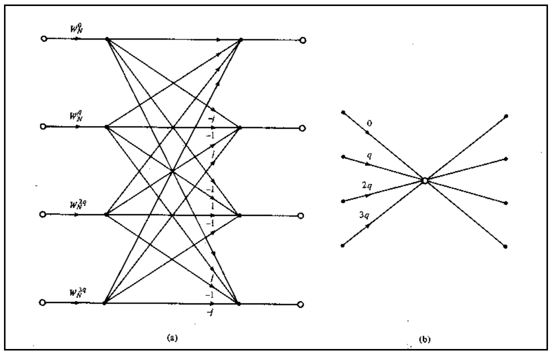

Figure 3: Data flow diagram of Radix-4 FFT[2]

2.Implementation Details

FFT is now considered as a mature and well established technology. Its main advantage is that it allows transmission over highly frequency selective channels at a low receiver implemetation cost. MATLAB is an excellent tool for algorithm development and data analysis . Now-a-days 90% of the algorithm used today originate as MATLAB models. Simulink is a graphical tool, which lets a user to implement graphically.

Equation (a) can be rewritten as

$X(k) = \sum_{n=0}^{N-1} x(n)\, W_N^{nk}$ ---------(B)[6] The quantity W_N^{nk} is

defined as

$W_N^{nk} = e^{-j2\pi nk/N}$ -------------(C) [6]

This factor is also called twiddle factor and is calculated. Here 64 point DIT , Radix-4 FFT has been designed using simulink in MATLAB and implemented on FPGA using VHDL.[2]

Xilinx System Generator(XSG) is a tool which offers block libraries that plugs into simulink tool to create HDL designs from MATLAB. It provides many features such as system resource estimation to take full advantage of FPGA resources, hardware co-simulation and accelerated simulation through hardware in the loop co-simulation which give many orders of simulation performance increase. It also provides a system integration platform for the design of DSP FPGA's that allows the RTL , simulink, MATLAB and C/C++ components of a system to come together in a single simulation implementation environment.

Here 64 point DIT , Radix-4 FFT has been designed using simulink in MATLAB.

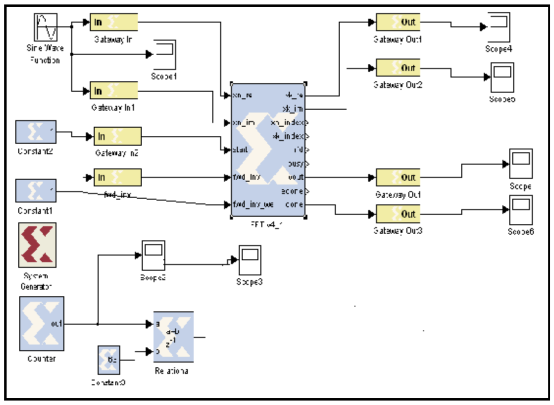

Figure 4: Implementation of FFT in MATLAB using simulink.

3.Simulation Results

FFT is designed using MATLAB and synthesized using Xilinx Project Navigator Xilinx ISE 9.2i. Results are verified using Modelsim XE 5.8c simulator.Simulation results verified that the FFT would perform with suitably low bit error for the full range of coding and modulation schemesand for various of all channel impairments. VHDL functional verification confirmed that the HDL design exhibited good performance.

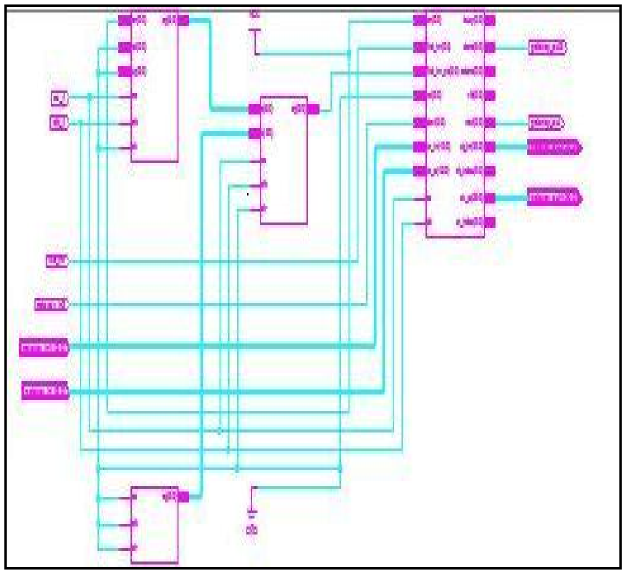

Figure 5: Internal RTL view of FFT

4.Conclusion

The main focus of this work is to show the capability of designing and simulating FFT . This work's main emphasis was on designing and simulation of synthesizable VHDL code of the COFDM transceiver using Xilinx's ISE 9.1i and simulated using ModelSim XE 5.8c simulator.

The simulated output of the required FFT is given :

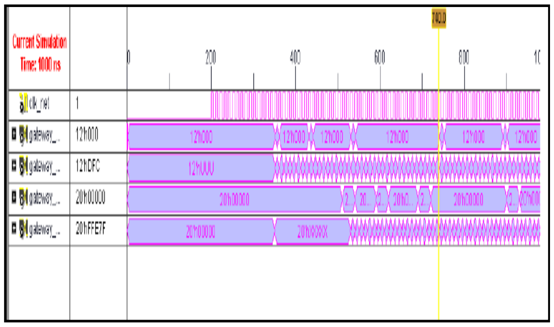

Figure 6: Simulation result of FFT

The logic utilization of FFT compared with the previous ones:

Logic utilization	**Used**	**available**	**Utilization**
No. of slice registers	5133	9312	55%
No. of fully used bit slices	3265	9312	35%
No. of bonded IOBs	91	232	39%
No. of BUFG	17	32	52%
No. of DSP48Es	24	32	75%

Table 1: Comparison with previous FFT's

5.Reference

1. S. B Weinstein and P.M. Ebert, "Data Transmission by Frequency Division Multiplexing Using the Discrete Fourier Transform", IEEE Transactions on Communication Technology", vol. com-19 , pp. 628-634, October 1971.
2. Simon H ay kin, "Communication Systems", Jhon Wiley & Sons, Inc., 4th Edition, ISBN 0- 471-17869-1, 2001.
3. "High level Synthesis tools for Xilinx FPGA" by the staff of Berkley Design Technology In.
4. System Generator: The State-of-art FPGA Design tool for DSP Applications ,by Sparsh Mittal , Saket Gupta, and S. Dasgupta from Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee.
5. Product brief, Parallel N-point FFT/IFFT core, RAD communications.
6. " Design of COFDM Transceiver using VHDL" by Hemant Kumar Sharma, Sanjay P. Sood and Balwinder Singh.