International Journal of Innovative Research in Electronics and Communications (IJIREC)
Volume 1, Issue 7, October 2014, PP 25-30
ISSN 2349-4042 (Print) & ISSN 2349-4050 (Online)
www.arcjournals.org

Design of 64point Fast Fourier Transform by Using Radix-4 Implementation

N.Omkar

M.Tech(VLSI)
GokaRaju RangaRaju College of Engg & Tech

Dr.T.C. Sarma

H.O.D ECE
GokaRaju RangaRaju College of Engg & Tech

Abstract: *The Fast Fourier Transform (FFT) is very significant algorithm in signal processing, to obtain environmental status and wireless communication. This paper explains the high performance 64 point FFT by using Radix-4 algorithm. Radix-4 has the advantage of parallel computations. This is simulated using VHDL, using Xilinx ISE 10.1 and simulated using ModelSIM6.5e. Here we shown the architectures of 32 point FFT withradix-2 and 64-point FFT with radix-4. Finally, the high performance 64-point FFT processor their architecture and simulation graphs are shown.*

Keywords: *FFT, radix-2, radix-4*

1. Introduction

Fast Fourier rework (FFT) processor is wide used in different applications, like local area network, image method, spectrum measurements, measuring device and transmission communication services [1]. However, the FFT formula is a hard task and it should be exactly designed to urge an efficient implementation. If the FFT processor is created flexible and quick enough, a transportable device equipped with wireless transmission is possible. Therefore, an efficient FFT processor is needed for period of time operations [2] and planning a quick FFT processor may be a matter of nice significance.

A fast Fourier transform (FFT) is the advanced version of discrete fourier transform (DFT). A Fourier transform converts time (or space) to frequency and vice versa. An FFT rapidly computes such transformations. As a result, fast Fourier transforms are widely used for many applications in engineering, science, and mathematics. The basic ideas were popularized in 1965, but some FFTs had been previously known as early as 1805[2]. Fast Fourier transforms have been described as "the most important numerical algorithm of our lifetime"

There are many different FFT algorithms involving a wide range of mathematics, from simple complex-number arithmetic to group theory and number theory; this article gives an overview of the available techniques and some of their general properties, while the specific algorithms are described in subsidiary articles linked below.

The DFT is obtained by decomposing a sequence of values into components of different frequencies. This operation is useful in many fields (see discrete Fourier transform for properties and applications of the transform) but computing it directly from the definition is often too slow to be practical. An FFT is a way to compute the same result more quickly: computing the DFT of N-points in the naive way, using the definition, takes $O(N^2)$ arithmetical operations, while a FFT can compute the same DFT in only O(N log N) operations. The difference in speed can be enormous, 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 roughly proportional to N / log(N). This huge improvement made the calculation of the DFT 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 best-known 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 $e^{-\frac{2\pi i}{N}}$ is the twiddle factor, an Nth 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 the same as the DFT, but with the opposite sign in the exponent and a 1/N factor, any FFT algorithm can easily be adapted for it.

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

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi k\frac{n}{N}} \qquad k = 0,\dots,N-1.$$

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 a lower complexity score is impossible[3](Johnson and Frigo, 2007).

To reduction the computations of complex multiplications and additions FFT is used. Calculating the computations using DFT involves N^2 number of 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 Cooley–Tukey algorithm, for N a power of 2, can compute the same result with only (N/2)log$_2$(N) complex multiplications (again, ignoring simplifications of multiplications by 1 and similar) and Nlog$_2$(N) complex additions. In practice, actual performance on modern computers is usually dominated by factors other than the speed of arithmetic operations and the analysis is a complicated subject (see, e.g., Frigo& Johnson, 2005), but the overall improvement from O(N^2) to O(N log N) remains.

The native implementation of the N-point digital Fourier transform involves calculating the scalar product of the sample buffer (treated as an N-dimensional vector) with N separate basis vectors. Since each scalar product involves N multiplications and N additions, the total time is proportional to N^2 (in other words, it's an O(N^2) algorithm). However, it turns out that by cleverly re-arranging these operations, one can optimize the algorithm down to O(N log(N)), which for large N makes a huge difference. The optimized version of the algorithm is called the fast Fourier transform, or the FFT.

The standard strategy to speed up an algorithm is to divide and conquer. We have to find some way to group the terms in the equation

$P[k] = \Sigma_{n=0..N-1} W_N^{kn}\, p[n]$

Let's see what happens when we separate odd ns from even ns (from now on, let's assume that N is even):

$P[k] = \Sigma_{n\ \text{even}} W_N^{kn}\, p[n] + \Sigma_{n\ \text{odd}} W_N^{kn}\, p[n]$

$= \Sigma_{r=0..N/2-1} W_N^{k(2s)}\, p[2s] + \Sigma_{r=0..N/2-1} W_N^{k(2s+1)}\, p[2s+1]$

$= \Sigma_{s=0..N/2-1} W_N^{k(2s)}\, p[2s] + \Sigma_{s=0..N/2-1} W_N^{k(2s)}\, W_N^{k}\, p[2s+1]$

$= \Sigma_{s=0..N/2-1} W_N^{k(2s)}\, p[2s] + W_N^{k}\, \Sigma_{r=0..N/2-1} W_N^{k(2s)}\, p[2s+1]$

$= (\Sigma_{s=0..N/2-1} W_{N/2}^{ks}\, p[2s]) + W_N^{k}\, (\Sigma_{s=0..N/2-1} W_{N/2}^{ks}\, p[2s+1])$

where we have used one crucial identity:

$W_N^{k(2s)} = e^{-2\pi i*2ks/N} = e^{-2\pi i*ks/(N/2)} = W_{N/2}^{ks}$

Notice an interesting thing, the two sums are nothing else but N/2-point Fourier transforms of, respectively, the even subset and the odd subset of samples. Terms with k greater or equal N/2 can be reduced using another identity:

$W_{N/2}^{m+N/2} = W_{N/2}^{m} W_{N/2}^{N/2} = W_{N/2}^{m}$

Which is true because $W_m^m = e^{-2\pi i} = \cos(-2\pi) + i \sin(-2\pi) = 1$.

If we start with N that is a power of 2, we can apply this subdivision recursively until we get down to 2-point transforms.

We can also go backwards, starting with the 2-point transform:

$P[k] = W_2^{0*k}\, p[0] + W_2^{1*k}\, p[1], \quad k=0,1$

The two components are:

$P[0] = W_2^0\, p[0] + W_2^0\, p[1] = p[0] + W_2^0\, p[1]$

$P[1] = W_2^0\, p[0] + W_2^1\, p[1] = p[0] + W_2^1\, p[1]$

We can represent the two equations for the components of the 2-point transform graphically using the, so called, *butterfly*

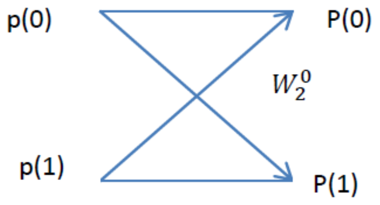


Fig.1.1. *Butterfly calculation*

Furthermore, using the divide and conquer strategy, a 4-point transform can be reduced to two 2-point transforms: one for even elements, one for odd elements. The odd one will be multiplied by W_4^k. Diagrammatically, this can be represented as two levels of butterflies. Notice that using the identity $W_{N/2}^n = W_N^{2n}$, we can always express all the multipliers as powers of the same W_N (in this case we choose N=4)

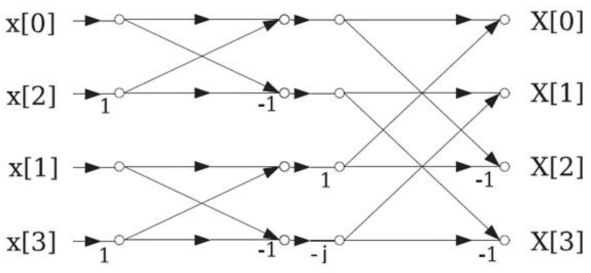


Fig 1.2. *Diagrammatical representation of the 4-point Fourier transforms calculation*

I encourage the reader to derive the analogous diagrammatical representation for N=8. What will become obvious is that all the butterflies have similar form.

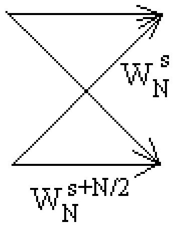


Fig 1.3. *Generic butterfly graph*

2. Radix 4 FFT

The decimation-in-time (DIT) radix-4 FFT recursively partitions a DFT into four quarter-length DFTs of groups of every fourth time sample. The outputs of these shorter FFTs are reused to compute many outputs, thus greatly reducing the total computational cost. The radix-4 decimation-in-frequency FFT groups every fourth output sample into shorter-length DFTs to save computations. The radix-4 FFTs require only 3/4th as many complex multiplies as the radix-2 FFTs.

The radix-2, radix-4 decimation-in-time and decimation-in-frequency fast Fourier transforms (FFTs) gain their speed by reusing the results of smaller, intermediate computations to compute multiple DFT frequency outputs. The radix-4 decimation-in-time algorithm rearranges the discrete Fourier transform (DFT) equation into four parts, sums over all groups of every fourth discrete-time index n=[0,4,8,…,N−4], n=[1,5,9,…,N−3], n=[2,6,10,…,N−2] and n=[3,7,11,…,N−1] as in Equation 1. (This works out only when the FFT length is a multiple of four.) Just as in the radix-2 decimation-in-time FFT, further mathematical manipulation shows that the length-N DFT can be computed as the sum of the outputs of four length-N4 DFTs, of the even-indexed and odd-indexed discrete-time samples, respectively, where three of them are multiplied by so-called twiddle factors WkN=e−(i2πkN), W2kN, and W3kN.

This is called a decimation in time because the time samples are rearranged in alternating groups, and a radix-4 algorithm because there are four groups. Figure 1 graphically illustrates this form of the DFT computation

It is this reuse that gives the radix-4 FFT its efficiency. The computations involved with each group of four frequency samples constitute the radix-4 butterfly, which is shown in 1.2. Through further rearrangement, it can be shown that this computation can be simplified to three twiddle-factor multiplies and a length-4 DFT! The theory of multi-dimensional index map shows that this must be the case, and that FFTs of any factorable length may consist of successive stages of shorter-length FFTs with twiddle-factor multiplications.

3. Implementation

FPGA

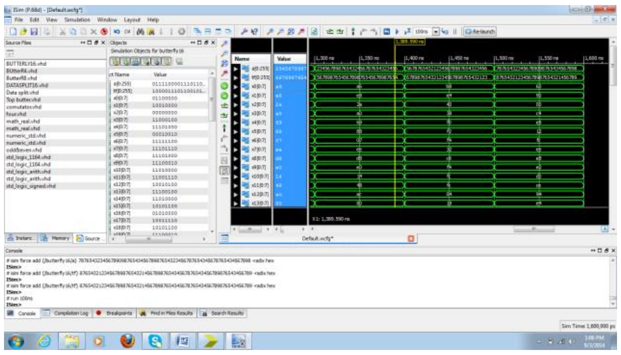
The Field Programmable Gate Array is majorly used for generation ASIC IC's to the computations. They offer more speed in execution process. So, for generation ASIC IC's FPGA's are majorly used. The 64 FFT with radix 4 is simulated and synthesized as well as implemented on the FPGA of below configuration.

Table 3.1. *Configuration of FPGA*

Property Name	Value
Family	Spartan 3E-250
Device	XC3SAN
Package	TQG144
Speed Grade	-3

4. Simulation Results

The RTL view of the butterfly structure obtained after the simulation of the 64-point FFT block, Decimation in time domain is shown next and also the internal architecture of the butterfly block is shown.



```
# isim force add {/butterfly16/a} [illegible] -radix hex
ISim>
# isim force add {/butterfly16/tf} [illegible] -radix hex
ISim>
# isim force add {/butterfly16/tf} [illegible] -radix hex
ISim>
# run 100ns
ISim>
```

Fig 3.1. *Waves forms View of A Butterfly Component Used In 64-Point FFT*

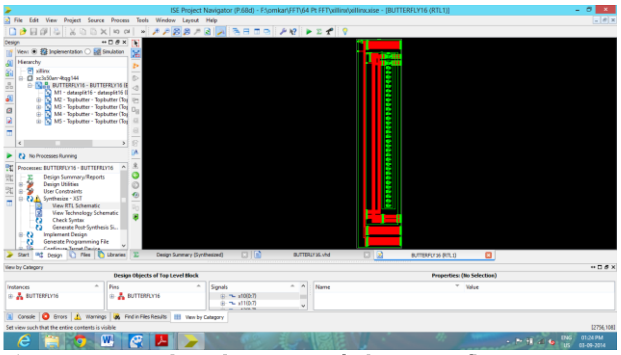
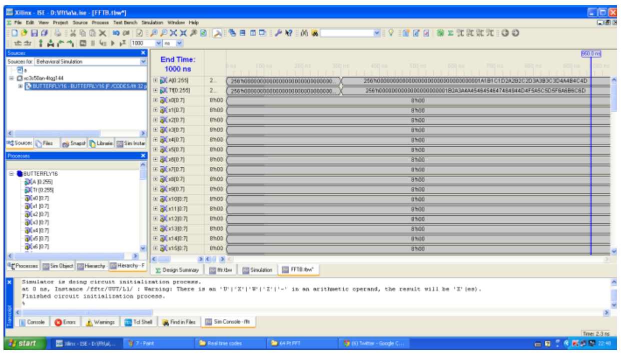
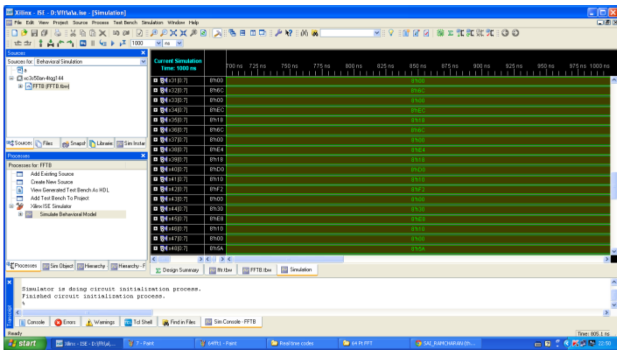


Fig 3.2. *Internal Architecture of The Butterfly Component*



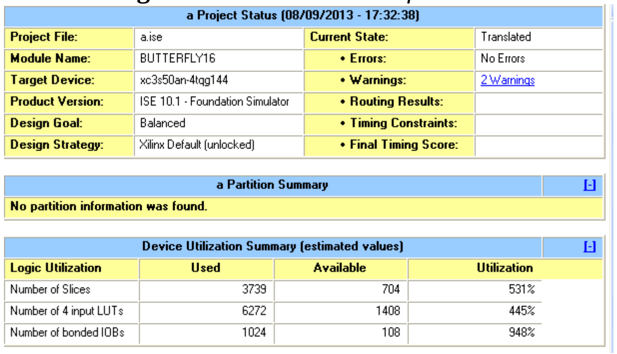
```
Simulator is doing circuit initialization process.
at 0 ns, Instance /fftc/UUT/L1/ : Warning: There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, the result will be 'X'(es).
Finished circuit initialization process.
```

Fig 3.3. *Simulation result of 64 FFT*



```
Simulator is doing circuit initialization process.
Finished circuit initialization process.
```

Fig 3.4. *Simulation result of 64 FFT*

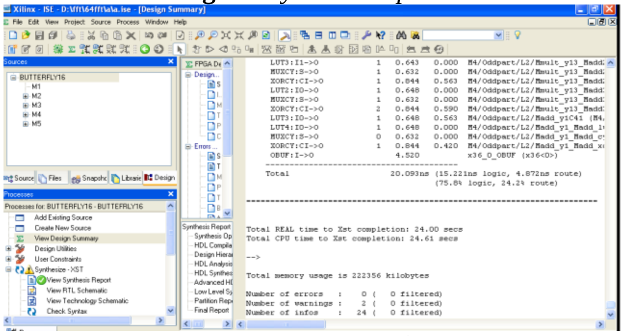


a Project Status (08/09/2013 - 17:32:38)			
Project File:	a.ise	**Current State:**	Translated
Module Name:	BUTTERFLY16	• **Errors:**	No Errors
Target Device:	xc3s50an-4tqg144	• **Warnings:**	2 Warnings
Product Version:	ISE 10.1 - Foundation Simulator	• **Routing Results:**	
Design Goal:	Balanced	• **Timing Constraints:**	
Design Strategy:	Xilinx Default (unlocked)	• **Final Timing Score:**	

a Partition Summary [-]
No partition information was found.

Device Utilization Summary (estimated values) [-]			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	3739	704	531%
Number of 4 input LUTs	6272	1408	445%
Number of bonded IOBs	1024	108	948%

Fig 3.5. *Synthesis report*



```
LUT3:I1->O            1   0.643   0.000  M4/Oddpart/L2/Mmult_y13_Madd
MUXCY:S->O            1   0.632   0.000  M4/Oddpart/L2/Mmult_y13_Madd
XORCY:CI->O           1   0.844   0.563  M4/Oddpart/L2/Mmult_y13_Madd
LUT2:I0->O            1   0.648   0.000  M4/Oddpart/L2/Mmult_y13_Madd
MUXCY:S->O            1   0.632   0.000  M4/Oddpart/L2/Mmult_y13_Madd
XORCY:CI->O           2   0.844   0.590  M4/Oddpart/L2/Mmult_y13_Madd
LUT3:I0->O            1   0.648   0.563  M4/Oddpart/L2/Madd_y1C41 (M4,
LUT4:I0->O            1   0.648   0.000  M4/Oddpart/L2/Madd_y1_Madd_l
MUXCY:S->O            0   0.632   0.000  M4/Oddpart/L2/Madd_y1_Madd_c
XORCY:CI->O           1   0.844   0.420  M4/Oddpart/L2/Madd_y1_Madd_x
OBUF:I->O                 4.520          x36_0_OBUF (x36<0>)
----------------------------------------
Total                    20.093ns (15.221ns logic, 4.872ns route)
                                  (75.8% logic, 24.2% route)

=========================================================================


Total REAL time to Xst completion: 24.00 secs
Total CPU time to Xst completion: 24.61 secs

-->

Total memory usage is 222356 kilobytes

Number of errors   :    0 (   0 filtered)
Number of warnings :    2 (   0 filtered)
Number of infos    :   24 (   0 filtered)
```

Fig 3.6. *Timing Report of 64FFT*

5. CONCLUSION

In this paper, a 32 point FFT with radix-2 and 64point with radix-4 processor was designed using FPGA System successfully. The processor use VHDL language to describe the circuit, use Xilinx ISE8.1i software to build the model, and use ModelSim SE 6.2e software for simulation.

REFERENCES

[1] R. W. Chang, “Synthesis of Band-Limited Orthogonal Signals for Multichannel Data Transmission,” Bell System Tech. J., vol. 45, pp. 1775–1796, Dec. 1966.

[2] ETS 300401, ETSI, “Digital Audio Broadcasting (DAB);DAB to mobile, portable and fixed receivers,” 1995.

[3] ETSI EN 300 744, “Digital video broadcasting (DVB);framing structure, channel coding, and modulation for digital terrestrial television,” 2001.

[4] European IST Project, “Power Aware Communications for Wireless OptiMised personal Area Networks (PACWOMAN),” http://www.imec.be/pacwoman/.

[5] S.B. Weinstein and P. M. Ebert, “Data transmission by frequency division multiplexing using the discrete Fourier transform,” IEEE Transactions on Communications, vol. 19, pp. 628–634, Oct. 1971.

[6] A.Peled and A. Ruiz, “Frequency domain data transmission using reduced computational complexity algorithms,” in Int. Conf. Acoustic, Speech, Signal Processing, Denver, CO, 1980, pp. 964–967.

[7] L.J. Cimini, “Analysis and Simulation of a Digital Mobile Channel Using Orthogonal Frequency Division Multiplexing ,” IEEE Transactions on Communications, vol. 33, pp. 665–675, July 1985.

[8] ETSI TS 101 475, “Broadband Radio Access Networks (BRAN); HIPERLAN Type 2 Physical (PHY) layer, v1.1.1,” 2000, http://portal.etsi.org/bran/.

[9] IEEEstd 802.11a, “High-speed Physical Layer in 5 GHz Band,” 1999, http://ieee802.org/.

[10] IEEEstd 802.11g, “High-speed Physical Layer in 2.4 GHz Band,” 2003, http://ieee802.org/.

[11] J. Bingham, “Multicarrier Modulation for Data Transmission: An Idea Whose Time Has Come,” IEEE Communications Magazine, vol. 8, pp. 5–14, May 1990.

[12] O. Edfors, M. Sandell, J. van de Beek, D. Landström and F. Sjöberg, “An introduction to orthogonal frequency-division multiplexing,” TULEA 1996:16, Div. of Signal Processing, Luleå, Tech. Rep., a University of Technology, Luleå 1996.

[13] J. W. Cooley and J. W. Tukey, “An Algorithm for Machine Calculation of Complex Fourier Series,” Math. Comput., vol. 19, pp. 297–301, Apr. 1965.

[14] R. Grunheid, E. Bolinth, and H. Rohling, “A blockwise loading algorithm for the adaptive modulation technique in OFDM systems,” in Proc. of Vehicular Technology Conference, VTC 2001 Fall, Atlantic City, NJ, USA, Oct. 7-11 2001, pp. 948–951.

[15] J. G. Proakis, Digital Communications. McGraw-Hill, 2001.

[16] M. Russel and G. Stuber,“Interchannel interference analysis of OFDM in a mobile environment,” in Proc. IEEE Vehic. Technol. Conf., vol. 2, Chicago, IL, 1995, pp. 820–824.

[17] N. Petersson, “Peak and power reduction in multicarrier systems,” 2002, licentiate thesis, Lund University, Sweden.

[18] S.Johansson, “ASIC Implementation of an OFDM Synchronization Algorithm,” 2000, Licentiate Thesis, Lund University, Sweden.

[19] R. Morrison, L. J. Cimini, and S. K. Wilson, “On the Use of a Cyclic Extension in OFDM,” in Proc. of Vehicular Technology Conference, VTC 2001 Fall, vol. 2, Atlantic City, NJ, USA, Oct. 7-11 2001, pp. 664–668.