Design and Simulation of FFT Processor Using Radix-4 Algorithm Using FPGA

N. Amarnath Reddy, D. Srinivasa Rao and J. Venkata Suman

GMR Institute of Technology
Asst. Prof. Dept. of ECE, GMRIT, RAJAM

amar.nagu02@gmail.com, srinivasarao.d@gmrit.org, venkatasuman.j@gmrit.org

Abstract

A parallel and pipelined Fast Fourier Transform (FFT) processor for use in the Orthogonal Frequency division Multiplexer (OFDM) and WLAN, unlike being stored in the traditional ROM. The twiddle factors in our pipelined FFT processor can be accessed directly. A novel simple address mapping scheme and the modified radix 4 FFT also proposed. FPGA was majorly used to develop the ASIC IC's to which was implemented. Here we simulated and synthesized the 256- point FFT with radix-4 using VHDL coding and 64 point FFT Hardware implementation we designed code using System C. Finally, the pipelined 256-point FFT processor can be completely implemented within 19.103ns.

Keywords- FFT, DIT, Radix-4, FPGA

1. INTRODUCTION

A fast Fourier transform (FFT) is an algorithm to compute the discrete Fourier transform (DFT) and it’s inverse. A Fourier transform converts time (or space) to frequency and vice versa; 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. 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 discrete Fourier transform is obtained by decomposing a sequence of values into components of different frequencies. This operation is useful in many fields (see DFT 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 naïve 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 [2] 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 $e^{rac{2\pi i}{N}}$ is an N-the 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 [5] and produces exactly the same result as evaluating the DFT definition directly; the only difference is that an FFT is much faster. Let $x_0, \ldots, x_N$ be complex numbers. The DFT is defined by the formula

$$X_k = \sum_{n=0}^{N-1} x_n e^{-2\pi i kn/N} \text{ for } k = 0, \ldots, N-1 \quad (1)$$

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 $\Theta(N \log N)$ operations (technically, $O$ only denotes an upper bound), although there is no known proof that a lower complexity score is impossible. (Johnson and Frigo, 2007)

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 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 $N\log_2(N)$ complex[12] 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 [4] (see e.g., Frigo & Johnson, 2005), but the overall improvement from $O(N^2)$ to $O(N \log N)$ remains.

The naive 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

$$V[k] = \sum_{n=0}^{N-1} W_N^{kn}v[n]$$

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

$$V[k] = \sum_n \text{ even } W_N^{kn}v[n] + \sum_n \text{ odd } W_N^{kn}v[n]$$

$$= \sum_{r=0}^{N/2-1} W_N^{k(2r)}v[2r] + \sum_{r=0}^{N/2-1} W_N^{k(2r+1)}v[2r+1]$$

$$= \sum_{r=0}^{N/2-1} W_N^{k(2r)}v[2r] + W_N^k \sum_{r=0}^{N/2-1} W_N^{k(2r)}v[2r+1]$$

$$= \left( \sum_{r=0}^{N/2-1} W_N^{k(r)}v[2r] \right) + W_N^k \left( \sum_{r=0}^{N/2-1} W_N^{k(r)}v[2r+1] \right)$$

Where we have used one crucial identity:

$$W_N^{k(2r)} = e^{-2\pi i * 2kr / n}$$

Copyright © 2013 SERSC

54
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:

\[ w_{N/2}^{m+n/2} = W_{N/2}^m W_{N/2}^n = W_{N/2}^m e^{-2\pi i m} = \cos(-2\pi) + i \sin(-2\pi) = 1 \]

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:

\[ V[k] = W_2^0 y[0] + W_2^1 y[1], \quad k = 0,1 \]

The two components are:

\[ V[0] = W_2^0 y[0] + W_2^1 y[1] \]

\[ V[1] = W_2^0 y[0] + W_2^1 y[1] \]

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

**Figure 1. Butterfly Calculation**

**Figure 1.1. 4-point Fourier Transform**

**Figure 1.2. Generic Butterfly Graph**
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 $W^k_4$ 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 $N=4$).

2. RADIX-4 FFT

The decimation-in-time (DIT) radix-4 FFT recursively partitions a DFT [5] 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 75% as many complex multiplies [6] as the radix-2 FFTs. The 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[23] frequency outputs. The radix-4 decimation-in-time algorithm rearranges the discrete Fourier transform (DFT) equation into four parts. The DFT[22] sums over all groups of the 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[22] FFT, further mathematical manipulation shows that the length-N DFT can be computed as the sum of the outputs of four length-N/4 DFTs, of the even-indexed and odd-indexed discrete-time samples, respectively, where three of them are multiplied by so-called twiddle factors $W_N^k = e^{-(i2\pi kn/N)}$, $W_N^{2k}$, and $W_N^{3k}$.

![Figure 2. Radix-4 DIT Structure](image-url)
This is called 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 Figure 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 maps shows that this must be the case, and that FFTs of any factorable length may consist of successive stages of shorter-length FFT [25] with twiddle-factor multiplications in between.

3. FPGA Implementation

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 [21] are majorly used. The 64 FFT with radix 4 is simulated and synthesized as well as implemented on the FPGA of below configuration.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Family</td>
<td>Spartan 3</td>
</tr>
<tr>
<td>Device</td>
<td>XC3S200</td>
</tr>
<tr>
<td>Package</td>
<td>TQG144</td>
</tr>
<tr>
<td>Speed Grade</td>
<td>-4</td>
</tr>
</tbody>
</table>

4. Simulation Results:

The RTL view of the butterfly structure obtained after the simulation of the 256-point FFT block, Decimation in time domain is shown next and also the internal architecture of the butterfly block is shown.

Figure 4.1. RTL View of a Butterfly Component Used In 256-Point FFT
Figure 4.2. Internal Architecture of the Butterfly Component

Figure 4.3. Simulation result of 256 FFT

Figure 4.4. Synthesis Report
Figure 4.5. Timing Report of 256 FFT

Hardware implementation was through system C coding and its results are as follows:

![Figure 4.5. Timing Report of 256 FFT](image)

Figure 4.6. Timing Report on Hardware

5. Conclusion

In this project it is shown that a baseband ASIC can be fast and at the same time flexible with a low power consumption. The term fast do not refer to extreme clock frequencies but to the fact that no part of the designs needs more than one clock cycle to process a sample once the pipe is filled. Hence, the design does not need to be clocked any faster than the requested bandwidth and compared to modern CMOS technology this is a low number, in the order of 5-100 MHz. There are several advantages with a low clock frequency, first it is possible use a low power/low speed cell library with low static leakage current and secondly, it is easier to create a clock tree. Since flexibility is achieved with independent modules, where the operation mode decides if a module should be used or not. The unused modules are not clocked and hence only consume static power.
leakage power and when you implement 256 point FFT in FPGA than better results are out.

References

Authors

Nagu Amarnath Reddy Pursuing M.Tech (VLSI & ES) in GMRIT, Rajam, A.P, India and Received B.Tech (ECE) from Newton’s institute of engineering under JNTUK. Major areas are interested in DSP and VLSI, ES.

D. Srinivasa RaO has completed B.Tech in Electronics and Communication Engineering from JNTU, Hyderabad. He completed M.E from Anna University with specialization in Communication Systems. Presently he is working as an Assistant Professor in Department of ECE, GMR Institute of Technology, Rajam. Major research areas include wireless communications, Information theory and coding. He is Life Member of ISTE.

Jami Venkata Suman has completed B.Tech from Tontadarya College of Engineering, under VTU, Belgaum, Karnataka. He Received Master of Technology in VLSI System Design from Annamacharya Institute of Technology and Sciences, Rajampet, under JNTU, Hyderabad, A.P and Master of Business Administration in HRM and MRKT from A.U, Visakhapatnam, A.P. He is currently working as an Assistant Professor in the Department of Electronics and Communication Engineering at GMR Institute of Technology, Rajam, A.P. Major Research areas include Radar, Signal Processing and VLSI. Life Member of ISTE.