

# FPGA implementation of Cooley-Tukey

Garrett Massman and Cory Walker

September 13, 2014

## Overview

In this project we will implement the Cooley-Tukey Fast Fourier Transform algorithm on a Xilinx FPGA using VHDL. The primary motivation for this is to digitally analyze musical signals, regardless of the instruments they come from. Using the Cooley-Tukey algorithm, we can calculate the FFT in  $O(n \log_2(n))$  time. As far as implementation, the project will be split into several main components. First, we will need to sample our audio for a set amount of time and store the data into a buffer in memory. Then, the digital signal will need to be processed using a complex ALU. Finally, the data will need to be sent to an output buffer for application specific usage. This can be in the form of a 7-segment hex display (for instance if we want to print out the fundamental frequency), or a microcontroller (if we want to plot the entire sampled spectrum by passing it to a more capable platform such as Matlab).

## Theory

To understand the discrete Fourier Transform, one must first analyze its analogous continuous time form. This is written most simply as

$$X(\omega) = \int_{-\infty}^{\infty} x(t) e^{-j\omega t} dt$$

which defines the transform of a continuous time signal  $f(t)$ . Without getting into too many details, it defines the signal as a combination of sinusoids, making it a very useful tool for real world applications.

The discrete Fourier Transform (DFT) is simply a reduction of the continuous Fourier Transform into a discrete sample space. In other words, if we let  $x_n$  represent a sampled version of the continuous time function  $x(t)$  with a total of  $N$  samples, we can replace the integral with a summation over the series, as shown below.

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{j2\pi kn}{N}}$$

As with the continuous time case, this series gives us a glimpse into the component elements of our original signal. However, it is rather costly to compute, requiring a time complexity of  $O(N^2)$ . For each value  $X_k$ , a series of values from  $n = 0$  to  $N - 1$  must be generated and summed, using up valuable computer resources. Thus, calculating the DFT in this way is very inefficient, and we must instead turn to the Fast Fourier Transform.

In order to perform this operation more quickly, we can utilize the Cooley-Tukey FFT algorithm. However, the one requirement is that our input is strictly a power of 2. That is because the algorithm works by recursively finding the FFT of smaller and smaller sample sizes of  $x_n$ , which arises from the fact that the transform itself is periodic. The transform function can be broken into the sum of its even and odd components,

$$X_k = \sum_{m=0}^{N/2-1} x_{2m} e^{-\frac{2\pi i}{N}(2m)k} + \sum_{m=0}^{N/2-1} x_{2m+1} e^{-\frac{2\pi i}{N}(2m+1)k}$$

This equation can further be simplified by making the substitutions

$$E_k = \sum_{m=0}^{N/2-1} x_{2m} e^{-\frac{2\pi i}{N}(2m)k} \quad O_k = \sum_{m=0}^{N/2-1} x_{2m+1} e^{-\frac{2\pi i}{N/2}mk}$$

giving us

$$X_k = E_k + e^{-\frac{2\pi i}{N}k} O_k$$

The expression  $e^{-\frac{2\pi i}{N}k}$  is commonly called the *twiddle factor*. Furthermore, because of the periodicity of the transform, we can calculate respective even and odd components simultaneously

$$E_k = E_{k+\frac{N}{2}} \quad O_k = O_{k+\frac{N}{2}}$$

## In Mathematica

Describe Mathematica code, how precomputed tables can help us.

## FPGA considerations

Because we are working with a very specialized type of hardware, we must take into consideration some limiting parameters. In particular, the Spartan3Es loaded on each BASYS board only have 72K RAM. This means that if we want a real time FFT, we will need to optimize the space we have available, and only write to it when it is absolutely necessary. We must also consider the quality of our digital input signal. There are various ADC PMODs available for the BASYS boards, which range in quality from 4.8 KHz to 1 MHz in speed. If we are only sampling audio, we probably won't need to gather any frequencies above 1200 Hz, so the lower end model should be fine.

Once our data has been collected and stored inside the FPGA, it needs to be processed. This is perhaps the most challenging part of the project, as we need to implement what will be called an FTC (Fourier Transform Controller) as well as a CALU (Complex Arithmetic Logic Unit). The FTC's primary function will be to keep track of the data as it moves from RAM, into the CALU, and back out into the real world. This will require some heavily synchronized logic and very tight timing control to be carried out properly. It will work in sync with the SPI input master to tell the CALU when data is ready to be read, and then clear the input buffer once the CALU has finished. It will also need to monitor the movement of data from the output buffer into whatever output stream requires it. The data may be sent directly to a microcontroller or into another component for post processing.

The CALU will have one primary job, which is to actually perform the FFT. From its perspective, it will only the addresses of an input buffer and an output buffer, as well as a flag that tells it when to start and stop. Because all of our calculations are being done inside of an FPGA, the CALU will be most composed of combinational logic at its core. In theory (i.e., provided enough gates), all that needs to be built is the base case of the FFT, which

takes in two numbers and outputs their sum and difference.

$$y_0 = x_0 + e^{-\frac{2\pi i}{N}k}x_1 \qquad y_1 = x_0 - e^{-\frac{2\pi i}{N}k}x_1$$

This portion of the computation is commonly called a *butterfly*, because the system's diagram looks vaguely like a butterfly. Each of the twiddle factors included above can be calculated beforehand and stored in a lookup table for quick access before each iteration. Because we will be performing the same type of computation with them each time, they are essentially constants in our system.

## Block diagram

Over the past few weeks we have taken what we know and outlined a basic block diagram of our device:

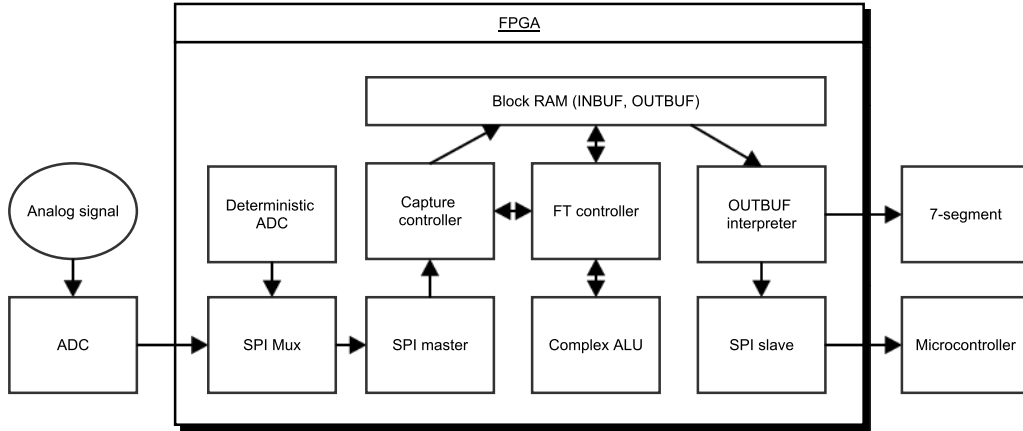


Figure 1: Block diagram of the FFT chip

Now that we have divided the device into several logical blocks, we can now split up the work accordingly.

## Roadmap

Describe the steps, in order, that need to be done.

## Resources

List of resources that we plan to use.

## Possible extensions

Describe the extra things we might add if we have time.

## Conclusion

## References

- [1] Doin, Jonny. *SPI Master/Slave Interface*. OpenCores, 16 May 2011. Web. 13 Sept. 2014. <[http://opencores.org/project,spi\\_master\\_slave](http://opencores.org/project,spi_master_slave)>.
- [2] Reynwar, Ben. *FFT on an FPGA*. FFT on an FPGA. N.p., n.d. Web. 13 Sept. 2014. <[http://www.reynwar.net/ben/docs/fft\\_dit/index.html](http://www.reynwar.net/ben/docs/fft_dit/index.html)>.
- [3] Satoh, Keiichi, Jubee Tada, Kenta Yamaguchi, and Yasutaka Tamura. *Complex Multiplier Suited for FPGA Structure*. Computers and Communications (2008): 341-44. Web. 13 Sept. 2014.
- [4] Wikipedia contributors. *Cooley–Tukey FFT algorithm*. Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 27 Jun. 2014. Web. 13 Sep. 2014.
- [5] Wikipedia contributors. *Discrete Fourier transform*. Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 2 Sep. 2014. Web. 13 Sep. 2014.
- [6] Roberts, Michael J. *Signals and Systems: Analysis Using Transform Methods and MATLAB*. New York: McGraw Hill, 2012. Print.