# Spectrum Analyzer: Introduction to Fast Fourier Transform and Power Spectra (ECE 420 specific)*

Matt Kleffner
Douglas L. Jones

Based on *Spectrum Analyzer: Introduction to Fast Fourier Transform (ECE 320 specific)*† by Matt Kleffner

This work is produced by OpenStax-CNX and licensed under the
Creative Commons Attribution License 1.0‡

**Abstract**

In this lab you will explore DSP FFT implementations and power spectral density (PSD) estimators. You will also learn how to write DSP code in C and the trade-offs between writing in C versus assembly.

## 1 Introduction

In this lab you are going to apply the **Fast Fourier Transform** (**FFT**) to analyze the spectral content of an input signal in real time. You will also explore algorithms that estimate a stationary random signal's **Power Spectral Density** (**PSD**). Finally, you will be introduced to using the C environment and code optimization in a practical application. This knowledge will be applied in optimizing a reference implementation of a PSD estimator.

## 2 Fast Fourier Transform

First, samples of the **power spectrum** of a deterministic signal will be calculated via the magnitude squared of the FFT of the windowed signal. You will transform a 1024-sample block of input data and send the power spectrum to the output for display on the oscilloscope. After computing the FFT of a 1024-sample block of input data, you will then compute the squared magnitude of the sampled spectrum and send it to the output for display on the oscilloscope. In contrast to the systems you have implemented in the previous labs, the FFT is an algorithm that operates on blocks of samples at a time. In order to operate on blocks of samples, you will need to use interrupts to halt processing so that samples can be transferred.

*Version 1.2: Sep 23, 2004 3:10 pm -0500
†http://cnx.org/content/m11828/1.2/
‡http://creativecommons.org/licenses/by/1.0

The FFT can be used to analyze the spectral content of a signal. Recall that the FFT is an efficient algorithm for computing the **Discrete Fourier Transform** (**DFT**), a frequency-sampled version of the **DTFT**.

DFT:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-\left(i \frac{2\pi}{N} nk\right)} \tag{1}$$

where $n \wedge k \in \{0, 1, \ldots, N-1\}$

Your implementation will include windowing of the input data prior to the FFT computation. This is simple a point-by-point multiplication of the input with an analysis window. As you will explore in the prelab exercises, the choice of window affects the shape of the resulting spectrum.

A block diagram of the spectrum analyzer you will implement in the lab, including the required input and ouput locations, is depicted in Figure 1.

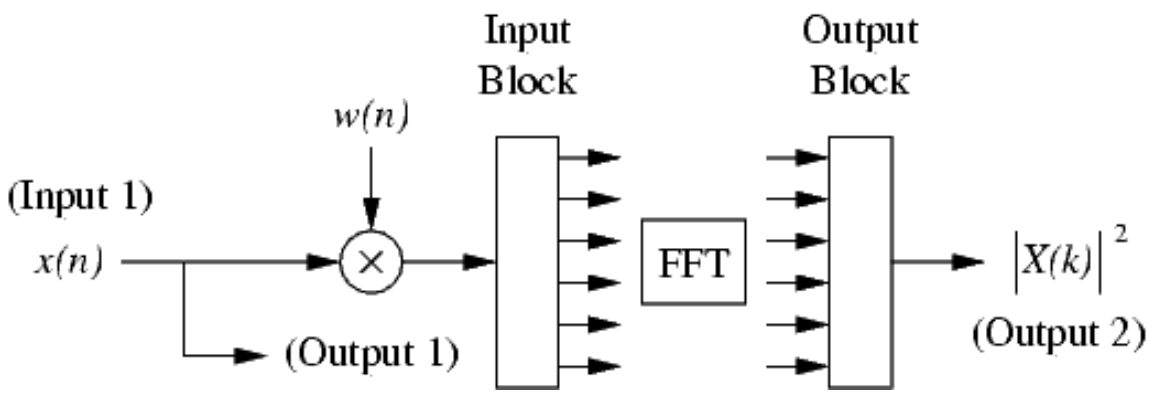


**Figure 1:** FFT-based spectrum analyzer

## 3 Pseudo-Noise Sequence Generator

Second, you will generate a colored, psuedo-noise (PN) sequence as input to the power spectrum algorithm. The noise sequence will be generated with a linear feedback shift register, whose operation is as shown in Figure 2 (Pseudo-Noise Generator). This PN generator is simply a shift-register and an XOR gate. Bits 0, 2, and 15 of the shift-register are XORed together and the result is shifted into the lowest bit of the register. This lowest bit is the output of the PN generator, and the highest bit is discarded in the shift process. The LSB is used to generate a value of $\pm M$ and this sequence will repeat with a period of $2^B - 1$, where $B$ is the width in bits of the shift register and $M$ is a constant. The power spectral density of this sequence is flat (white) and it will be "colored" via a fourth-order IIR filter. PN generators of this type are a useful source of random data bits for system testing. They are especially useful as a data model in simulating communication systems as these systems tend to randomize the bits seen by the transmission scheme so that bandwidth can be efficiently utilized. However, this method will not produce very "random" numbers. For more on this, see *Pseudorandom number generator*[_number_generator?], *Linear feedback shift register*[_feedback_shift_register?], and chapter 7, section 4 of *Numerical Recipes*[?].

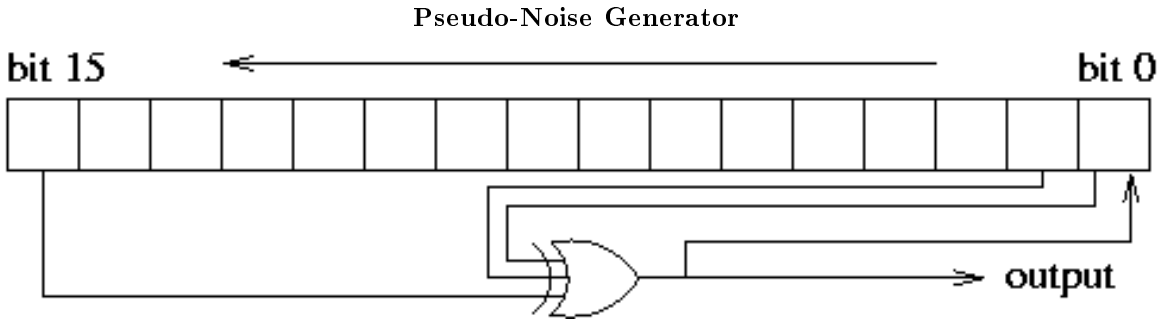


**Figure 2**

## 4 Power Spectral Density Estimation

The direct-power-spectrum (DPS) algorithm outlined above is insufficient for estimating the PSD of a stationary noise signal because the variance of the estimated PSD is proportional to the value of the actual PSD. For the third part of this lab you will try to reduce the variance of the PSD estimate by windowing the autocorrelation of the noise signal and computing the fft.

The autocorrelation of a sequence is the correlation of the sequence with itself:

$$R[m] = \sum_{i=0}^{N-1-|m|} x[i]\, x[i+|m|] \tag{2}$$

where $m \in \{-(N-1), -(N-2), \ldots, N-1\}$

For random signals, the autocorrelation here is an estimate of the actual autocorrelation. As $|m|$ is increased, the number of samples used in the autocorrelation decreases. The windowed-DPS algorithm is equivalent to taking the FFT of the autocorrelation of the windowed data. Windowing of the data adds even more noise to the autocorrelation estimate, since the autocorrelation is performed on a distorted version of the original signal. An improvement can be made by constructing an accurate estimate of the autocorrelation (using a rectangular window), applying a window and computing the FFT. The motivation for applying the window at the latter stage is that emphasis should be given to accurate autocorrelation values while less accurate values should be de-emphasized or discarded.

A good empirical characterization of a random process requires sufficient data, and both of the PSD-estimation algorithms defined above can be extended to accomodate more data. There is one caveat, however: many real-world processes are modeled as **short-time** stationary processes (non-stationary models are hard to deal with), so there is a practical limit to how much data is available for a PSD estimate. Additional data is added to the direct-PSD estimation algorithm by adding multiple spectra together, thereby smoothing the PSD estimate. Additional data is added to the windowed-autocorrelation method by computing the autocorrelation of the total data set before windowing. You will explore the windowed-autocorrelation method on the DSP.

## 5 Compiling and Optimization

A second objective of this lab exercise is to introduce the TMS320C549 C environment in a practical DSP application. The C environment provides a fast and convenient way to implement a DSP system using C

and assembly modules. You will also learn how to optimize a program and when to use C or assembly. In future labs, the benefits of using the C environment will become clear as larger systems are developed.