Experiment 3: Fast Fourier Transform

Bruno Korst - bkf@ece.utoronto.ca

Abstract
In this experiment, you will probe the Fast Fourier Transform (FFT) algorithm by applying different inputs to it and by changing some of its parameters. This will be done first in simulation, using existing FFT blocks in `Simulink`, and then by using an FFT program in real-time on a DSP platform. On the real-time implementation, you will transform your time domain display on the oscilloscope into a frequency domain display. You will then experiment with using FFTs of different lengths to assess their resolution and computation time, and will analyze signals with multiple sinusoidal components. Finally, you will explore particular aspects of the butterfly structure of the FFT implementation by changing some parameters on an 8 point Decimation-in-Time FFT model.

Keywords
FFT — length — frequency bins — resolution — spectral leakage

Contents

Introduction

The purpose of this lab is to lead you through some of the issues involving the implementation of Fast Fourier Transforms. This will be done first by simulating a spectrum analyzer and changing some parameters on both the input signal and the FFT algorithm. Then you will build a basic spectrum analyzer using a DSP platform and an oscilloscope.Finally, a closer look at the implementation of an FFT will provide insight on the butterfly structure commonly used in code. By the end of the lab, you will have a good understanding of the trade-off between length of the FFT and resolution, and how it applies to analyzing the frequency domain of a signal. You will also have a better understanding of the implementation process and its limitations. The main steps are as follows:

- Simulate and run in real-time FFTs of different lengths for one sinusoid only as your input;
- Simulate and run in real-time FFTs of different lengths for two sinusoids, observing how the change in resolution affects your output;
- Observe the trade-off between resolution and computation time as applied to different types of inputs;
- Simulate an 8-point DIT FFT with one sinusoid as input, calculate and change parameters on the butterfly structure of the FFT implementation.

Note that the lab **answer sheet** is to be done in groups of two students. You should print the answer sheet to bring to the lab. In the lab you will follow this procedure in pdf, and write your answers in the answer sheet.

1. Background Reading and Preparation

The primary source for you to look at is any DSP textbook, such as [5] or [3]. Some excellent complementary sources are [4] and [2]. These will help you to better understand the mathematical and practical issues that may arise in this lab. For the practical side of DSP, you can look at [1] and [6]. These are extremely helpful references on DSP real-time implementation.

It is beneficial for you, even for your future professional practice, to become acquainted with the tools used in this course, so do practice often if you can.

2. Experiment

The lab is divided into three main parts. In the first part you will simulate a system that performs an FFT on a single sinusoid for a given number of FFT points. Then, you will use two sinusoids added as your input and vary the length of the FFT, or the number of FFT points. Your objective in this second part of the lab will be to actually display two sinusoids on the scope by making the right decision regarding the FFT resolution. Thus, you are simulating a somewhat crude spectrum analyzer. Finally, you will look at the butterfly structure of the FFT implementation, which is typically what is done in code to implement an FFT.

2.1 The FFT of a Single Sinusoid

The question to be asked here is: what happens to the output of my FFT routine when I change frequency and amplitude of my input? We know that the Fourier Transform is *linear*, therefore it has the properties of *homogeneity* and *additivity* [**?**]. This is a question that also involves spectral leakage (perhaps a time to do a 2 minute search on it...). To summarize, your N-point FFT partitions produces N frequency *bins* on the output. The output represents two mirror-imaged halves of the spectrum you desire to analyze, and it spans from DC (0 Hz) to F_s (your sampling rate). If the frequency of your single-sinusoid input falls exactly on one of these *bins*, you have no leakage. Otherwise, the FFT will result in magnitude components for the frequency bin closest to where the input frequency is, and for the adjacent bins.

2.1.1 Simulation

The system to be implemented is very straightforward: the input is a sinusoid (time domain) and the output is the magnitude FFT of it (frequency domain). Using `Simulink`, create a simple system with a discrete-time sine block (called `DSP Sine Wave`), a `Buffer` block, a `Magnitude FFT` block and a `Array Plot` block. Open a new model in `Simulink` and connect these blocks together. Since the FFT is an operation performed on blocks of data (more on this later), it is necessary for you to store incoming data in a buffer prior to calculating the FFT. This same procedure also takes place for the actual implementation of an FFT on a microprocessor. With these three blocks above, you have effectively created your own spectrum analyzer. However, the look and feel of the display of a real spectrum analyzer is slightly different. The block `Spectrum Analyzer` provided in the `Simulink` library resembles more closely what the equipment in the lab displays. Add this block to your model, so that you can understand how your spectrum display relates to a more realistic one. Your system should look like the one presented in Figure 1.

For this simulation, you will display only the magnitude of your calculated sequence, even though the output of the FFT is a set of complex numbers yielding magnitude and phase information. The model and results presented below have a sampling rate of 8KHz and 256 for the FFT size and the buffer size. The simulation time is set to `inf`. The model in Figure 1 yields the results presented in Figures 2(a) and 2(b).

As a general rule, always set the buffer size with the same number of samples as the FFT points. If the buffer is smaller than the number of points in the FFT you are trying to compute, the simulation will produce an error; if it is larger, the display will interpret the size of the buffer as reference for the frequency axis. Feel free to change the display mode to whatever you feel is better to answer the questions and report the results. You can also set the running time from *inf* to 10.0, if you do not want the simulation to run continuously.

After you explore a little bit, set the sinusoid to 500Hz, $1V_p$ (peak!). Run the system (click on the "play" icon).

Now move to the answer booklet, answer the questions pertaining to the single sinusoid simulation.

2.1.2 Implementation

Now that you know what to expect from your system, run `Code Composer Studio` and open the project named `ECE431_Exp03_FFT64`. Take a moment to look at the programs comprising the project implementing an FFT. Try to identify important routines within the program, such as the generation of twiddle factors, the loop implementing the *butterflies*, the location of input and output buffers, etc.

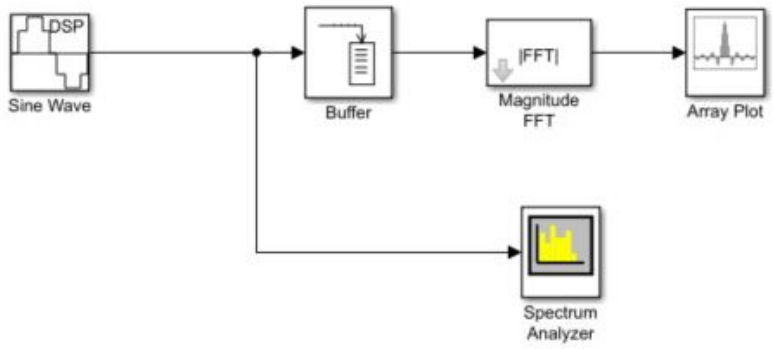
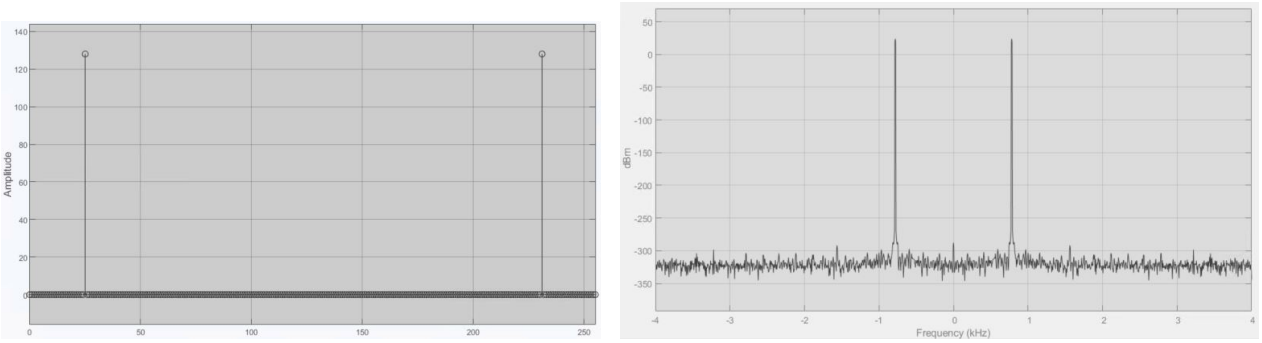


Figure 1. FFT Simulation for a Single Sinusoid



(a) Array Plot Display

(b) Spectrum Analyzer Display

Figure 2. Output as Displayed on Two Different Displays

Make sure you connect inputs and outputs to the instruments. Set the signal generator to a 2kHz, $1V_{pp}$ sinusoid. Set the output impedance of the generator to *High Z*. Compile the project and make it run. You should see a display similar to the one shown in Figure 3. Adjust the horizontal sweep to provide you with the best view (like the one in the picture), and set the trigger to the channel with the single periodic impulse. *Note: Your scope is set to display time domain, but it is actually displaying the result of an FFT. That is, it is displaying the frequency domain*

The code you are running implements a 64 point FFT in real-time. The magnitude portion of the result (output of the routine) is displayed on one channel, while the other channel only provides a reference, or a "beacon". This reference is actually the location of DC on the very first "bin" of your resulting FFT, and will be used for your measurements. This reference impulse is displayed as the bottom signal on Figure 3 above. This reference impulse is your trigger reference.

The DSP board continuously outputs the block of 64 numbers resulting from the calculation. Since you only have a single sinusoid as your input, and it is within the Nyquist limit, your output should display **TWO** "impulses": one between DC and $f_s/2$ and the other between $f_s/2$ and f_s. (This frequency is Hz. If the DFT of $x[n]$ is $X[k]$, then k and f are related by $(k/N) = (fT_s)$.) All other values are close to zero, as there is no other spectral component. Since this happens continuously, you will see more than two impulses on your scope. However, the period that you are interested in happens between two DC (or reference) spikes. You have studied the periodic nature of the Fourier Transform. The scope is actually displaying *repeated* versions of the same results. Though it *looks* the same as your textbook picture for the Fourier transform, the interval calculated is from DC to f_s. The results are displayed back to back, repeatedly, i.e., between two reference "DC" spikes you will have the interval from DC to f_s.

For a better understanding of what is happening, increase the frequency of the sinusoid from 2kHz to 10kHz. When you see the peaks moving you know which channel is displaying the magnitude of your sinusoid. Place the frequency back at 2kHz and the magnitude at $1V_{pp}$.

On some of the questions you will be required to measure the frequency displayed on the scope. Since you have references at DC and f_s, your best option will be to use cursors to find the time difference between one of the references and your desired signal. A simple calculation will yield the frequency value. Feel free to adjust the horizontal span of your scope to provide you

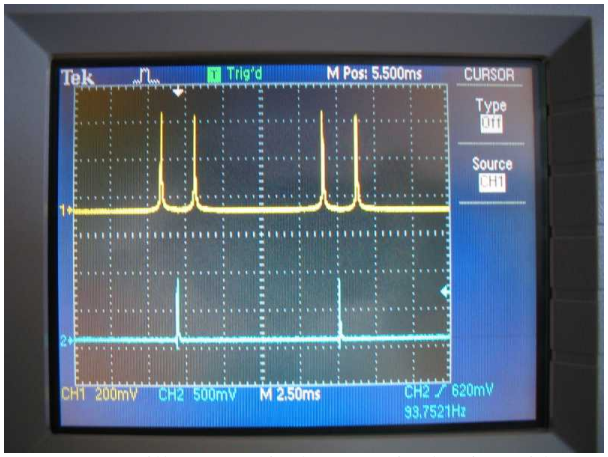


Figure 3. Oscilloscope Display For Single Sinusoid Input

with a better picture. The relative time value between cursors is displayed on the right side under the label *Delta*. The picture you should see on the scope is displayed in Figure 4. Answer the questions pertaining to the single sinusoid implementation on the answer booklet.

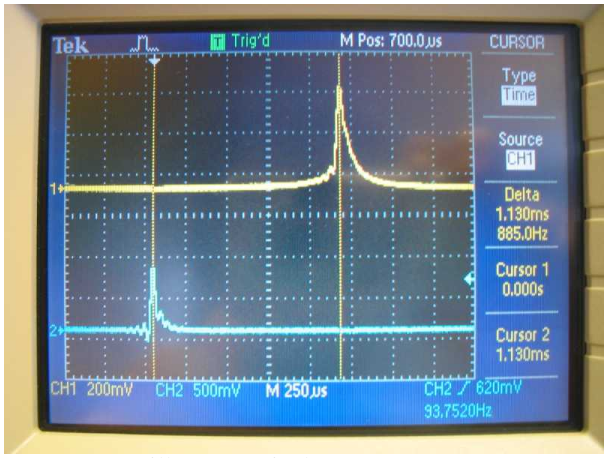


Figure 4. Oscilloscope Display For Cursor Measurement

2.2 FFT Resolution for Two Sinusoids

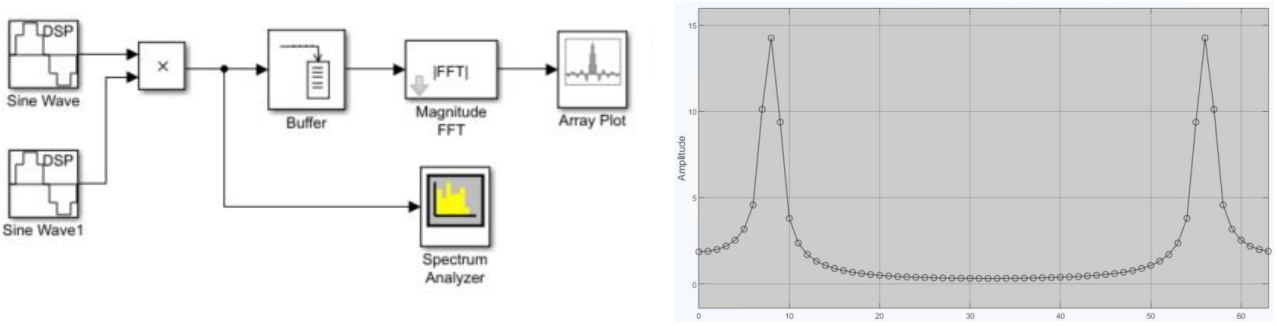
Now your objective is to establish a good enough length for the FFT to resolve a more complex signal. In a practical setting, this would be analogous to selecting the resolution of the spectrum analyzer to view, say, a carrier signal modulated in amplitude by a sinusoid.

As you know by now, a greater number of FFT points will increase the resolution in the frequency domain, since it decreases the spaces between adjacent frequency points. In other words, the resulting sequence has a greater number of "bins" for the same frequency interval.

2.2.1 Simulation

Add one `DSP Sine` and one `Product` to the previous system you simulated, so that your system looks like the one in Figure **??** below. Set one of the sinusoids to be 1kHz and the other 50 Hz. Your `Buffer` and `FFT` blocks should be set to 64 still. What you intend to visualize is the spectrum components resulting from the multiplication of two sine waves, one at higher frequency than the other. You saw this exact operation in high school, but you did not have Fourier then! Here you get to see what will happen in the frequency domain.

Run your simulation. While you run it, you will notice that the curve moves; the picture displayed on Figure **??** is a snapshot of the running system. Run the system and answer the questions on your answer booklet.



(a) Simulation Model (b) Frequency Domain (**weird**) Display

Figure 5. Simulation of FFT for Two Sinusoids

2.2.2 Implementation

The arbitrary signal generators that you have on your workstation allow for signals to be designed or recorded, and then loaded onto the generator. The signal you will use in this part of the lab is under the function button labelled `Arb` on the generators.

After you select the appropriate signal, make sure your output button is pressed or no signal will be input to the DSP. Your objective here is to find the components of your input signal.

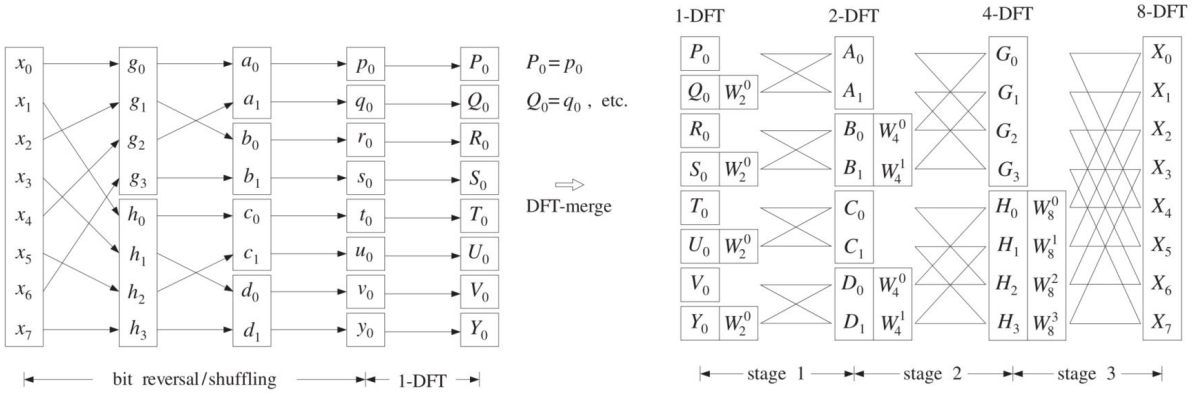
Since the workstations have two types of signal generators, the signal you will use in this part of the lab has been downloaded onto the instrument. The procedure to set the signal generators properly is described on the implementation section of the answer booklet below.

Start `Code Composer Studio` and load the project `ECE431_Exp03_FFT256` onto it. Compile and run the project. Make sure you have an output displayed on the scope. Answer the questions on Section 2.2 of your answer booklet. Later, you may want to use the FFT capability of the oscilloscope for comparison with the 1024 point FFT you have obtained for a sinusoid.

2.3 Butterflies

In this section you will experiment with a given model in `Simulink` of an 8-point Decimation-In-Time FFT implemented using a butterfly structure. You will find it at `http://www.comm.utoronto.ca/~bkf/comm/ECE431`, under the FFT experiment.

Figure 6 shows the representation of the structure used in the model (from [4]). Open the model and explore the structure to see how it maps to the one given below.



(a) Decomposition (b) DFT Merge in 3 stages

Figure 6. 8-point DIT FFT Butterfly Structure

The structure presented has three stages, and the input to the first stage (the input is in *time domain*!) has two main characteristics:

- It is buffered. That is, the FFT algorithm works on *groups, arrays* or *frames* of samples. On Figure 6(a) above, the input buffer size is 8, and it takes in x_0 - x_7;

- It is ordered in bit-reversed order. That is the *address* in bits of the location of each input sample is read *in reverse*. Address `0010` is then read as `0100`, for example. This is also shown in 6(a), where the input to the `1-DFT` is a bit-reversed version of x_0 - x_7.

The reading of the addresses in bit-reversed order is done to account for the initial *decomposition* of the time-domain signal, needed to proceed with the rest of the FFT algorithm. After that is done, then the algorithm proceeds with the calculation of the frequency spectrum (that is the *DFT Merge*).

Note that if you were to implement such algorithm using code in a very fast manner (FAST is the first F in FFT), you would at the very least need storage for buffering the input (it is a frame, or buffer-based process) and for the base-function factors, known as *twiddle factors* (those are the W_N^k shown on Figure 6(b) above). The number of these factors needed varies with the size of the FFT, so if the size of the FFT will not change (that is, you are happy with the resolution chosen), then they can be calculated once, ahead of the algorithm and stored in a static array.

As far as the processing goes, you would need additions and multiplications (real and complex) of signed floating-point numbers. Note also that the bit-reversed addressing is typically a feature in the assembly language of many processors, and that speeds up the process as well. Since you have taken APS105, you are now familiar with all the tools to implement an FFT for an embedded processor, which is what just ran multiple times on your smartphone as you were reading this paragraph...

Move now to the answer sheet and address the final questions there.

3. Conclusion

In this lab, you simulated and implemented a Fast Fourier Transform algorithm, and tested its limitations for different input signals. You have likely verified that for a greater number of FFT points, a finer frequency resolution is achieved. This is done at the cost of greater memory use (to store longer input and output sequences), and greater processing time, which, depending on your real-time constraints, may pose an impediment to long FFT calculations. Since the FFT algorithm by its very nature is *fast*, processing time may be lengthened by multiple external memory reads to retrieve buffered input data and write output data.

Acknowledgments

Thanks to all the students who have provided input on the previous versions of this experiment.

References

[1] R. Chassaing. *Digital Signal Processing and Applications with the c6713 and c6416 DSK*. Wiley, 2004.

[2] Jervis B. Ifeachor, E. *Digital Signal Processing, A Practical Approach, 2nd. Ed.* Prentice-Hall, 2001.

[3] Schafer R. W. Oppenheim A. V. *Discrete-Time Signal Processing, 3rd. Ed.* Prentice-Hall, 2009.

[4] S.J. Orfanidis. *Introduction to Signal Processing*. Prentice Hall, 1996.

[5] Manolakis D. K. Proakis J. G. *Digital Signal Processing, Principles, Algorithms and Applications, 4th. Ed.* Prentice-Hall, 2006.

[6] Wright C. H. G. Welsh, T. B. and M. G. Morrow. *Real-Time Digital Signal Processing, From Matlab to C with the TMS320C6X DSK*. CRC Press, 2006.