Lab 4 - Discrete Fourier Transform (DFT) and an Application

Section	L01	L02	L03	L04
Lab Date	Nov. 10	Nov. 11	Nov. 12	Nov. 13
Due Date for Report	Nov. 21	Nov. 21	Nov. 21	Nov. 21

Assessment: 5% of the total course mark.

Objectives:

- To gain experience in implementing and analyzing DFT in `MATLAB` and in using FFT in TMS 320 DSP processor for FMCW radar signal processing.

Assessment:

- Your grade for this lab will be based on your ability to analyze and implement DFT in `MATLAB` and to understand the FFT implementation on the DSP processor.
- Clearly label all plots and their axes (points for style will be deducted otherwise)
- Please attend the lab section to which you have been assigned
- You can complete this lab with one lab partner
- By the end of the lab session, you must demonstrate to your TA the `MATLAB` code and the TMS 320 DSP processor part.
- All MATLAB source code and c code (updated "RadarProcessor_lab4.c") must be submitted on Avenue to Learn by the end of your designated lab session.
- Each pair of students should complete one lab report together. The report has to be submitted by **11:59 pm on Nov. 21, 2025**.

Experiments:

1. **Introduction to the DFT**
   The discrete Fourier transform (DFT) is <u>not</u> the same as the discrete-time Fourier transform from Lab 3. The main difference is in the resulting frequency resolution. When the DFT is computed for an array of length N, the frequency resolution is inherently determined. In contrast, the result of the DTFT has continuous or infinite frequency resolution and thus requires that frequencies be specified for the purpose of computation and plotting in `MATLAB`.
   (a) Create a `MATLAB` function for computing the DFT of an arbitrary length `N` input array `x`, for the set of frequency sampling points `k = 0:length(x)-1`.
   Hint: Either update one or two lines in your DTFT function to compute the DFT, or try writing a new function for the DFT that implements the matrix formulation of the DFT.
   (b) Use your function to compute the DFTs of the 5 rectangular signals of different lengths `N`:

$$x[n] = \begin{cases} 1 & 0 \le n \le 15 \\ 0 & 16 \le n \le N-1 \end{cases}$$

   with `N = 16,32,64,128,256`.

(c) Describe what happens to the magnitude of the output as the zero-padding at the tail end of the rectangular signal increases in length.

(d) Use the function that you created in Lab 3 to compute and plot the DTFT of the same input set. What frequency vector `w` do you need to provide as an input to the DTFT function to return the same outputs as the DFT?

(e) Use the built-in `MATLAB` command for the Fast Fourier Transform `fft()` to compute the DFT for the input set. Compare the results of these three Fourier transform methods.

2. **DFT Resolution**

(a) Load the `.wav` file `aaa.wav` using MATLAB `audioread()` function. This is a synthesized vowel /A/. What is the sampling frequency?

(b) Plot the waveform (i.e., in time-domain) of the first 300 samples. You should see a natural repetition. What is the period of this natural repetition?

(c) Take the magnitude of the DFT of one natural period and compare it with that of two natural periods. How does the spectrum change?

(d) Zero-pad the same two signal pieces to 1024 points and plot the magnitude of the DFTs. From section 1 above, zero-padding should change the frequency sampling resolution. Is the frequency sampling resolution changing as you would expect?

(e) Compute and plot the magnitude of the DFT of the signal windowed at 1, 2, 3, 4 and 5 times the natural period. Using your knowledge of the effects of windowing on spectral resolution, explain the change in the spectrum due to an increase in the number of samples used to compute the DFT.

3. **FFT Application on FMCW Radar Signal Processing with the TMS 320 DSP Processor**

(a) **Background Knowledge**

In the previous lab, you used time-domain signal processing technique to find the targets in the detection range. In this lab, you will learn how the frequency-domain technique is applied to extract the range information of the targets. Specifically, the Fast Fourier Transform (FFT), which is an efficient algorithm for calculating DFT and is widely used in real-world applications.

Recall that the reference signal (transmitted signal) and the received signal have a time shift δt when a target is present in the coverage (see Figure 1).

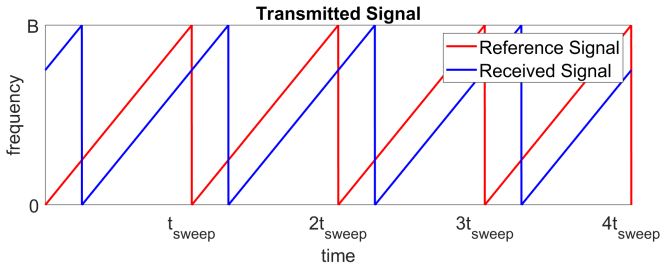

Figure 1: Frequency vs. time plots of the transmitted signal and the received signal.

The dechirped signal is generated by multiplying the reference signal with the conjugate of the received signal.

$$x_{dechirp} = x_{ref} \cdot x^*_{received} \tag{1}$$

In this lab, the dechirped signal is provided with the transmitted and received signals. Hence, you do not need to calculate the dechirped signal yourself.
The signals are in complex numbers, and the conjugate of the received signal is the phase-inversed version of the received signal. Therefore, the resultant signal of the product of the reference signal and the conjugate received signal is a narrow band signal with the dominant frequency at beat frequency f_b. The beat frequency is equal to the frequency difference of the reference signal and the received signal (see Figure 2).

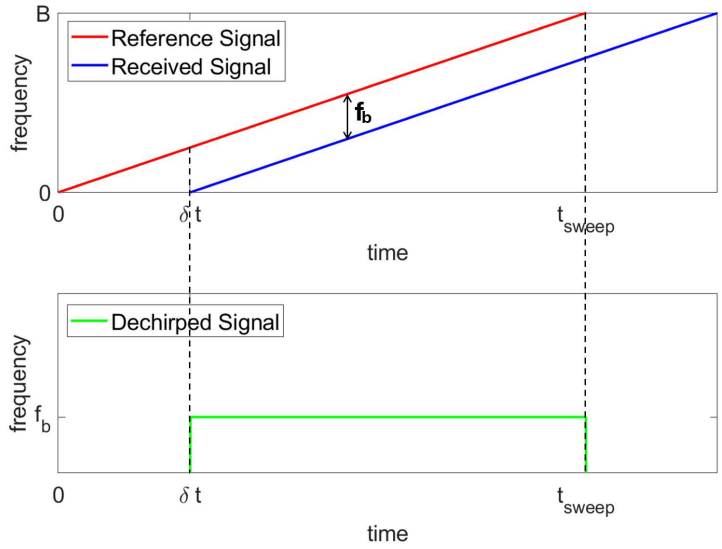

Figure 2: The dechirp operation creates a narrow band signal at beat frequency.

By evaluating the FFT of the dechirped signal, the beat frequency f_b will present as the peak and the value of f_b can be found. Since the slope of the linear modulation is known, the time delay δt can be evaluated with $\frac{f_b}{slope}$. Therefore:

$$\delta t = \frac{f_b \cdot t_{sweep}}{B} \tag{2}$$

Once the time delay is known, similar to the previous lab, once the time shift is found, the range of the target can be calculated.
The equation for finding the range is given as follows:

$$r = \frac{ct_{sweep}}{2B} \frac{f_s n_{peak}}{N_{FFT}} \tag{3}$$

where n_{peak} is the sample index of the peak(s), c is the speed of light, f_s is the sampling frequency, t_{sweep} is the duration of each sweep, B is the bandwidth of the reference signal, and N_{FFT} is the number of point of the FFT. The following table listed the parameters you may need and their values:

Parameter	**Value**
Time of a single sweep (t_{sweep})	3.3356 μs
Bandwidth (B)	150 MHz
Sampling rate (f_s)	150 MHz
Number of points of FFT (N_{FFT})	512

(b) **Lab Procedure**

i. Import the source file into the project:

A. Create a copy of the project that you worked on in lab 3 and rename as "RadarProcessor_Lab4" and open this project in CSS.

B. Delete "RadarProcessor_Lab3.c" from this project.

C. Right click the project under the explorer, and choose "Add Files..".

D. Direct to the folder "Y:/COE4TL4/RadarProcessor/src", and add the source code "RadarProcessor_lab4.c" to the project. MAKE SURE TO CHOOSE THE OPTION OF "COPY FILES", NOT "LINK FILES".

E. Include the DSP library path.

- Right click the project under the explorer tab, click "Properties", unfold "Build", and then unfold "C6000 Compiler". Click "Include Options", and add the path "Y:/COE4TL4/dsplib/include" .
- In the same window, under "Build", unfold "C6000 Linker", click "File Search Path", and add the path "Y:/COE4TL4/dsplib/lib/dsp67x.lib".

ii. Make sure that the dechirped signal is used: comment the line "#include "data_rec.h"" and uncomment the line "#include "data_dechirp.h"".

iii. Go through the source code "RadarProcessor_lab4.c" and the header file "FFT_helper.h" and understand the functions. You don't have to understand the FFT() function. There are two options to call FFT functions in the code, one function is written with C code, and the other one is the FFT function from the DSP library, which takes full advantage of the underlying hardware of the DSP processor. Note that the two implementations handles the complex numbers in different ways.

iv. Use the provided FFT function "FFT()" to detect the target.

A. The function "prepare_fft_data()" loads the data in the data file (the one you include the path) and transfer that to the complex data that can be used in the "FFT()" function. Complete this function (Replace the comment "// needed to fill" under the function definition "prepare_fft_data()").

B. Compile and run the code. Observe the result on oscilloscope.

C. Record the CPU time shown in the console of CCS. Note that the CPU time is in clock ticks, not in seconds.

v. Use the DSP library FFT function "DSPF_dp_cfftr2()" to detect the target.

A. Uncomment "#define USE_FFT_LIB" on the top of the "RadarProcessor_lab4.c" file to switch to the DSP library implementation.

B. Similar to the previous section, complete the function "prepare_fft_data_for_lib()". Note that the data type that is used to store the complex data is different from the previous one. Replace the comment "// needed to fill" under the function definition "prepare_fft_data_for_lib()".

C. Complete the prepare functions calling part. The function you want to use are "prepare_w_fft_lib()", and "prepare_fft_data_for_lib()". Make sure that the parameters you pass are correct.

D. Complete the FFT function calling part. The functions you want to use are "DSPF_dp_cfftr2()", and "bit_rev_2()". Right click the functions and go to their definition to see their descriptions and usages. Explain to the TA what the functions do and why you want to use them here.

E. Compile and run the code. Observe the result on oscilloscope.

F. Record the CPU time.

vi. Compare the CPU time of the two FFT's, and comment on their difference.

REPORT: The report should contain

- **ALL** the `MATLAB` plots and the oscilloscope screenshots **WITH BRIEF DESCRIPTIONS**.
- Answer the questions 1(d), 1(e), 2(d), 2(e) and 3(b-vi) with some discussions.

You **do not** need to include the `MATLAB` code and the C code in the report. However, you have to submit the code separately.