Electrical and Computer Engineering 434

Final Project: SONAR PROCESSING SYSTEM

This assignment is a group exercise with 3 due dates:	
Lesson 35 COB:	Stage 1-5 Due
Lesson 39 BOC:	Stage 6-11 Due
Lesson 40:	Team Presentations

1. Introduction

For this exercise, your instructor will form you into teams of 3 to 4 students that will simulate a small DSP company that is competing for a design contract. You will use your judgment and the knowledge you've gained so far in ECE434 to design the DSP processing stages of a sonar system. You can employ any basic DPS techniques you wish, and use any design tools available in Matlab to meet the project specifications. Any specifications you are unable to meet or any engineering trade-offs you must decide between must be fully justified. Your company must find a way to distinguish your sonar system from your competitors. You must decide if your goal is to optimize your system for high quality sonar images or to display fast real-time sonar images. You can also make your system unique by offering features not found in your competitor's sonar systems. Your professor and a panel of reviewers will attend your company's final presentation. The panel will review your design, evaluate your system based on the stated objectives, and assess your response to any questions. The panel will determine the "winning" company. All members of a given company will share the same basic group score for this exercise, weighted by their individual contribution (discussed later). The winning company normally receives a grade in the "A" range. The other companies normally receive grades between a 0 and an 90.

2. Background

On the ECE434 website schedule for lesson 32 you will find the background information about the sonar system. **Sonar_Processing_v6.pdf** describes the 11 basic stages of the sonar system and references other readings (such as Dr Musselman's *Radar in the Electrical Engineering Major*, **radar.pdf**).

You will be coding your sonar system in MATLAB and processing real data obtained from our 4 channel microphone phased array stored in test data files on the ECE434 website. You do not have to design a 14 stage sonar system from scratch. We have provided you a top level MATLAB program called **cpx3_sonar_v4.m** which initializes all the parameters for the system, calls all the 14 stages, measures the time each stage takes, and plots many figures (like 28 !!!) in both the time and frequency domain so you can see the effect of each stage. This top-level program calls each stage as a subroutine which has been compiled as pcode (so you cannot see the code inside). Your job will be to write your own Matlab subroutines to replace these pcode (.p) versions. [Note: For Matlab to run your function in place of the pcode version, it must have the same name, same parameters, and the .p file should be deleted from the directory] All this code is on the website inside

cpx3_sonar_v4.zip. Matlab templates for your functions can be found in **cpx3_sonar_Matlab_Headers.zip**.

3. CompEx Requirements

For all your MATLAB subroutine design and implementations, your subroutines and overall system will be evaluated in terms of correctness of approach, achievement of the design objectives, and the quantitative and qualitative performance of your system (i.e., test results).

Note: Ask your instructors permission before using any "canned" functions in MATLAB. For example, for the upsampling stage, you cannot use the MATLAB functions *interp* or *upsample*. A list of MATLAB functions you can use are given in Appendix A.

To help your team pace yourself, you will have two different milestones. For each milestone, each team must implement at least the number of stages corresponding to their number of team members (e.g., 3 stages for 3-person teams). Each person on the team is responsible for implementing a stage (note in your report who implemented each stage, and if your team does an extra stage, which stage is the "bonus" stage). Implementing extra stages can give you a competitive advantage over other teams.

- By **COB Lesson 35**, stages 1 to 5 (from Calibration to Beamformer) are due. The Beamformer and the Noise Removal Filter are mandatory stages that must be implemented by the team. If the Noise Removal Filter is done inside another stage, then this meets the requirement, however, person doing filter now needs to find another function to do.
- By **BOC Lesson 39**, stages 6 to 11 (from Demodulator to Tracking) are due. The Demodulator is the only mandatory stage that must be implemented by the team.

What is in the Milestone Report?

For each milestone you will turn in via email the MATLAB code for the subroutines and a clear can concise report on the stages. The report's introduction should discuss your overall system's design goals (e.g. fast or high quality; extra features planned), discuss each subroutine implemented, and have an overall conclusion for the milestone. The discussion for each subroutine must include the subroutine purpose (*what it does)*, the subroutine design goals (*justify why you selected the design parameters)*, implementation (*how it was implemented and its specifications*), and the test/analysis results. The testing should prove the system meets specs (I really like to see figures and tables, time domain and frequency domain), and any claims that it is better than the original .p provided must have evidence to support the claim. Also list the tasks performed by each team member during this milestone phase. Grading criteria is posted on the class website.

If you had to modify **cpx3_sonar_v4.m** for your subroutines to run, also turn this in. You are discouraged from modifying **cpx3_sonar_v4.m** unless it is for a change that enhances your system.

Important note: When your MATLAB code is turned in on at the final milestone on lesson 39, your company's system design is then frozen. By BOC lesson 39 each company must supply

all the other competing companies in their ECE434 section their MATLAB code via email (must include the filter coefficients and matlab code to simulate the system, and any test files). You must CC your instructor on this email. This simulates a bit of industrial espionage. Each company has a chance to find any weak areas of flaws in their competitor's designs to exploit in their own presentation to the panel. Failure to provide this data on time in a usable manner will result in disqualification of the company from the competition and a very poor grade for all the members of the company.

Another important note: The individual's grade on this group work exercise is partially determined by a confidential "peer eval." By **COB lesson 40**, *each student* must complete the on line "peer eval" in which you evaluate yourself and your team mates efforts. Also, For each report, 75% of your grade is based on your team grade and 25% is based on your individual grade for the report.

4. Presentation Requirements

The last part of this exercise is your group presentation, attempting to convince the panel that your company has the best design and implementation with the best features. Your presentation of your testing and analysis results will be the key component for your company to win.

The presentation should be very concise (no longer than 10 minutes), professional, and involve *every* member of the company. Your presentation will be evaluated based on correctness, completeness, organization, evidence of insight into DSP topics, communication skills, and how your presentation compares to the presentations of the other companies. As a minimum, your presentation should include the following

- Justification for your design approach/goals
- Frequency and Time domain plots for the given test files showing the performance of the stages implemented.
- Demonstration of additional features.

You should supply a hard copy of your presentation slides to your professor and the panel members. The presentation grading criteria is posted on the website, under lesson 40.

5. Additional Comments on Design Specifications

Some of the processing stages have very straight forward design goals (like calibration's "remove DC Bias"), but other stages, such as Noise Removal LPF are more complicated and the specifications should be justified. For the stages that require filters, you have already had design experience (in CPX2) and should justify your choice for specs such as Cutoff Frequency, Transition Bandwidth, Maximum Passband Ripple, Minimum Stopband Attenuation, Group Delay, Filter Order, and issues such as whether Linear Phase is important or not.

Your company's decision whether to implement high quality sonar images (at the cost of speed) or a fast real-time system (at the cost of quality) will drive many of your specifications. Some companies may find a way to have both a high quality and fast system.

Here are some ideas on higher quality sonar images:

- Increase filter order
- Increase number of beams (thus angular resolution). This would require higher upsampling
- Remove ringing of transmitted pulse. One way to do this is to experimentally measure the impulse response of the speakers. Then deconvolve the desired pulse by this impulse response to create a new pulse waveform. When this new waveform hits the speakers, it will come out as a clean sinusoid without ringing (*almost)*. **[due to hardware issues, this may not be an option this year]**

Here are some ideas on faster implementations:

- Reduce filter order without impacting quality too much
- Implement your subroutine in C language, assuming you can figure out how to call C functions from Matlab.
- Delete unnecessary stages. (You must turn in two versions: one with all the individual subroutines and one with the deleted stages)
- Precompute intensive calculations known aprior into "look-up-tables", so they do not have to be computed in real-time. One example is the polar-to-rectangular conversion required in Scan Conversion is known a head of time, and can be precomputed.
- Use vectorized Matlab instead of "for loops."

Remember that the panel of experts judging your sonar system will decide if your design decisions "enhanced" the system or made it worse.

6. Additional Comments on Testing/Analysis

cpx3_sonar_v4.zip comes with a tests file containing one frame of real sonar data called **test_data**. There is also a second test file, **test_data3**, that includes multiple frames over time, so you can see how persistence works (for instructions on accessing/using the file see the **test_data3.doc** file on the website). This file contains both a stationary and a moving object. You may also generate your own "synthetic" test files.

The real test files provide a great way to explore the *qualitative* performance of your subroutine, however, may be difficult to use to "measure" the performance of your subroutines. You may find you need to create your own test file to test the performance of an individual subroutine to measure its *quantitative* performance. For example, for a filter you will want to measure how many dB the signal is reduced in the stopband, ensure the signal stays within the allowable ripple in the passband, and ensure the 3 dB cutoff frequency is in

the correct location. *How could you create an input signal to test this for your filter?* For simpler functions like *Time Gain Compensation (TGC)*, you can create a simple input signal such as a constant voltage (DC) signal, and the output should be properly amplified with respect to time as you specified for your *TGC* function.

How to measure time? Measuring time for a function is not as easy as it may seem. **cpx3_sonar_v4.m** already has the code in place to measure the time. To more accurately measure the time, set `continuous = 1.0;` and use **test_data3**

Appendix A: MATLAB functions you may use inside your Subroutines

Contact your instructor for permission to use other MATLAB functions.

Standard Loops (For, While), If/Then/Else, standard math operators * / + -
Vectorized math operations.

Filter(): for filter's you must export your filters such that the filter() can be used as:

y = filter(filt1.tf.num, filt1.tf.den, x)

Conv()
FFT()
IFFT()
Plot()
Length()
Size()
Sqrt()
Mean()
Abs()
RMS()
Max()
Min()
Log(), Log10()
Exp()
Window()
Remez()
Zeros()
Sin(), cos(), tan()
Asin(), acos(), atan(), atan2()
Load()

Find() may be used for window/level (stage 9), but not for calibration (stage 1)

Histeq() and Imadjust() only may be used if you also implement Window/Level contrast enhancement, then compare their results to pick the best one.

For Edge Enhancement, you may use any Matlab edge or contour finding command, but if you do, you must mathematically explain how they work, and compare their results, and pick the one you think best for your system.

Don’t wait until the last minute to do this! Have fun...