Experiment # 2
Filters

Purpose

The purpose of this experiment is to demonstrate some of the effects introduced by a radio channel on a binary signal travelling through it. The binary signal carrying the information will be represented by a square wave, and the band-limited channel by a digital filter. In this experiment, you will vary the bandwidth (the pass band) of the channel by utilizing a low-pass filter a band-pass filter, and will verify the effects of the bandwidth variation on signal integrity. This will be done by looking at the signal in both time domain and in frequency domain.

A review on Fourier Series, Fourier Transform, convolution and a basic understanding of digital filtering (FIR) is recommended. In order to perform this experiment effectively, a good understanding of `Simulink` and `Matlab` is required. Make sure you refresh your knowledge of `Simulink` and `Matlab` before you start the experiment.

This experiment is to be conducted in three main steps: a) the design and simulation of a filter in `Matlab`/`Simulink`, b) the generation and testing of the simulated system on a DSP platform, and c) the modification of that system in real-time. These three steps should provide you with an understanding of the concepts seen in theory, as well as the general practical issues faced in the actual implementation of a digital filter.

At the end of this experiment, you should have a clearer understanding of:

- The role of the channel in a communication system;
- The effect of band limitation on binary transmission;
- How a digital filter is implemented.

Equipment

Hardware:

1. One Signal Generator;
2. One Oscilloscope;
3. One Spectrum Analyzer;
4. One TMS320C6711 board attached to a workstation;
5. Three Coaxial cables.

Software:

1. `Matlab` Release 12;
2. `Simulink` with ECE416 Toolbox;
3. Code Composer Studio, v.2.1.

Reviewing the Theory

The theoretical background needed for this experiment is well documented, and it can be found in both basic and advanced texts in signals and systems and Digital Signal Processing. The textbook [1] provides a review of Fourier Series. For the practical filtering implementation, you are encouraged to look at [2], [3] and [4] as a preparation to this laboratory. The topics of interest are the represeintation of signals in the time and frequency domains, the Fourier Series, the Fourier Transform and a basic understanding of digital filter design and implementation. It is sufficient for you to understand that the filtering process can be described, in a simplified manner, as a result of the convolution between the input (sampled) signal and the coefficients (also called "parameters", or "taps") of the digital filter, which in a direct implementation represent the impulse response of the filter.

In previous courses (such as Signals and Systems), the synthesis of a square wave was demonstrated by the addition of its fundamental and its odd harmonics, as described by a Fourier Series. At that time, the Gibbs phenomenon, which appears in the synthesis of periodic signals with discontinuities (i.e., square wave, saw-tooth wave, etc.), was also introduced. In this experiment, a somewhat reverse process to that will be explored. Rather than building up a signal from its harmonics, you will see what happens to a signal when some of its components are subtracted from it. It is important, therefore, that you review the concepts introduced in previous courses.

Experiment

The experiment is divided in three parts: the design and simulation of a digital filter, the generation of a working model on a DSP-based platform, and the modification of that model in real-time.

1. **Designing and Simulating a Digital Filter**

   There are numerous software packages commercially available that will assist the Engineer in designing Digital Filters. In this experiment, the Digital Filter `Simulink` block will make use of `Matlab` capabilities to generate the desired filter coefficients. Throughout this experiment, you will use a *Finite Impulse Response* filter, or FIR. This is the same type of filter you have studied during your lab preparation.

   The first step is to build a model for simulation. By making use of the blocks found in the ECE416 toolbox, connect input and output to the digital filter block. You may use different input signals to simulate your model; do not be restricted by the straight-forward solution.

The digital filter block is actually a filter design tool as well, so that by double-clicking on it, one will find a digital filter design GUI.

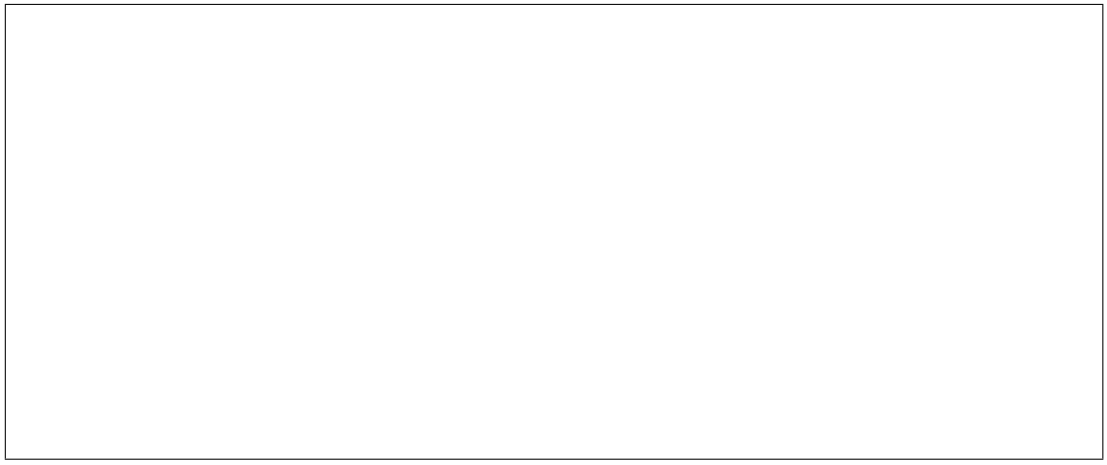
The objective of this part of the experiment is to simulate a low pass filter with cut-off frequency of 4KHz, and a band-pass filter with cut-off frequencies of 1KHz and 10KHz. Design both filters with the **same order** (which also means the same number of coefficients). There is no need to choose an order higher than 30. Also, to avoid complications in the second part of the experiment, avoid the "minimum order" option. Choose the "window" method for FIR design, and use a Kaiser window. If time allows at the end of the experiment, go back into the filter design tool and explore different options. Digital filte design is not the topic of interest here, it is to be explored in other courses.

- *Present below the impulse response and the frequency response of the two filters you have designed. How is the order chosen related to the length of the impulse response? Where do the "coefficients" come from?*

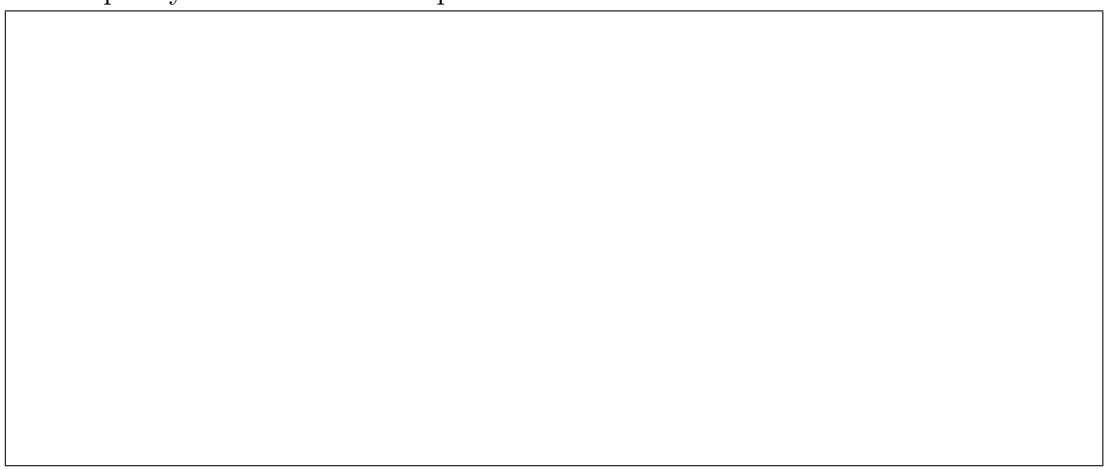
- *Changing the order of your filter to 60, you will notice that the frequency response is improved. At what expense is this improvement achieved? (look at the impulse response and at the FIR block diagram on your lab preparation)*

Having chosen all parameters, now choose the input and output. For a signal source, use initially the `DSP Sine Wave` block, and at a second run a `Pulse Generator` with a 1/2 period wide pulse to emulate a square wave. Starting with a sinusoid, use an amplitude of $1V_p$. You can visualize the results for the input signal in time domain and in frequency domain by utilizing the `Simulink` blocks you used in the previous experiment (Experiment # 1). You may be required to resolve pending problems related to signal shape, as it is handled by `Simulink`.

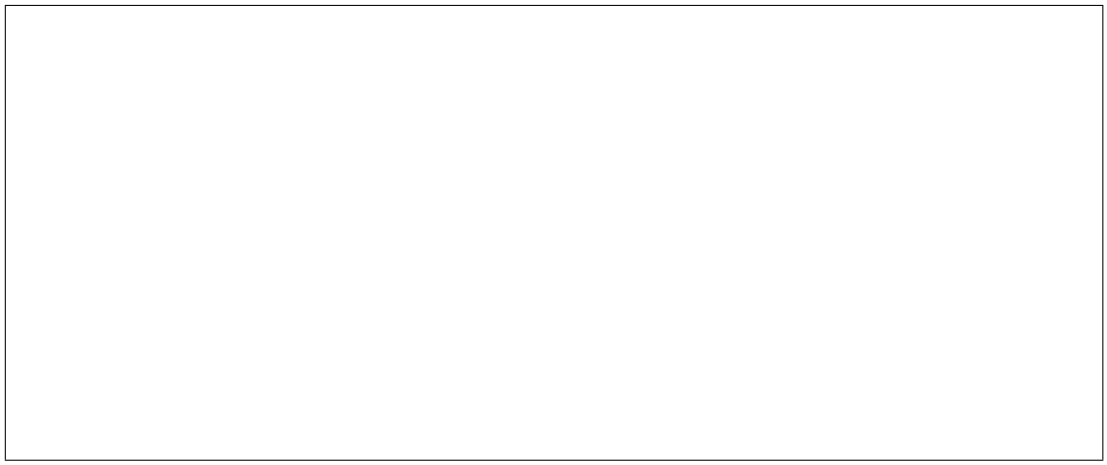
- *I*nput a 500Hz sinusoid to the low-pass filter that you have designed. Report the values at 1KHz, 3KHz, 5KHz, 10KHz and 15KHz. Repeat for the bandpass filter (create a three column table). Explain your results.



- Now use a 1000Hz square wave as input to the low pass filter, and sketch the time domain and frequency domain results. Explain the results.

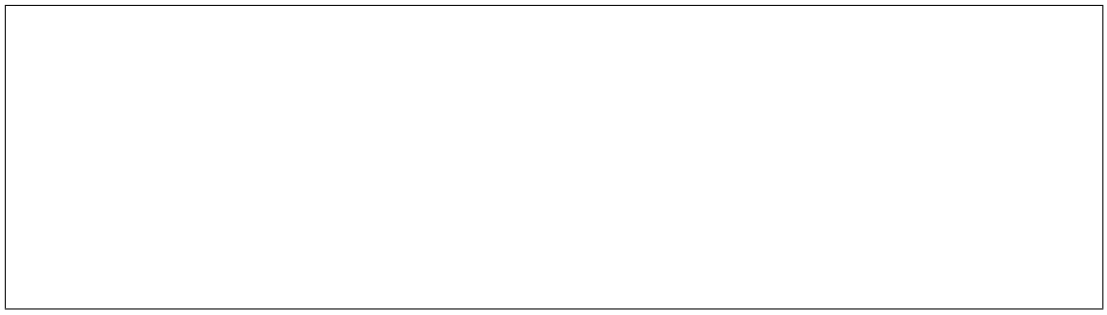


- Repeat the procedure with the band-pass filter, for 200Hz, 1KHz and 5KHz. Explain your results.



- Explain what you believe would be the consequence of a band-limited channel on a

binary signal.



2. **Building and Running a Model**

   In this part, substitute the input and output blocks on your model, respectively, by the ADC and DAC blocks found on the ECE416 Toolbox. Make sure the design parameters used in your filter design are compatible to the parameters set in these two "new" blocks. Run the model under `Simulink` to check for any signal shape incompatibility (don't expect to see any output; this is just a check for signal shape compatibility). In the Digital Filter block, choose initially the same low-pass filter design you chose for the simulation part. Now the model is ready to be build and downloaded to the DSP platform.

   Make sure you modify the `Simulink` build option parameters to make your model compatible with the target hardware. The parameters to be set are found under tools/real-time workshop/options, found on the command bar of your `Simulink` model. By selecting "options", a window will appear to you. Follow this procedure:

   - Select the "Solver" tab. Click on "Solver Options" and select type "Fixed-Step";
   - Next, select the "Real-Time Workshop" tab. Click on the "Browse" button on the Configuration area. A new window will appear, with a list of "*.tlc*" files. Select the "*ti_c*6000*.tlc*" file and click on the OK button.
   - Under the same "Real-Time Worshop" tab, go under Category and look under "TI C6000 Target Selection". You should have the C6711DSK selected, and must not change any other selections.
   - Click on OK and you are ready to download your model onto the DSP platform.

   Resolve any pending incompatibilities detected by `Simulink` and build the model, as you did in the previous lab experiment. You can do this either by going under Tools/Real-Time Workshop/Build or by pressing Ctrl-B. At this stage, progress messages will appear on the `Matlab` Command Window. If any error occurs, you will be showed a new window with the specific details of the error. Try to resolve it, and if you cannot proceed, ask for assistance from your Laboratory T.A..

   After the code for your model has been generated, `Matlab` will load it onto Code Composer Studio. This program will be opened in a new window, and progress messages will appear indicating that your model has been turned into a "Project" and that the compiled and assembled project is being loaded onto the DSP platform. The program will run automatically. A window will appear within CCS, which is the "Disassembled" code that is running on the platform.

Now that the project is visible in Code Composer Studio, explore the files avaliable on the project tree (left-hand side). Take a careful look at the code, in both C files and H files. From this brief survey, you should have a picture of how the project was generated. Now that you are ready to run your first digital filter, run it. You can run or halt the program as you wish. Try to identify the routines in which the many parts of your model are generated in software.

Connect the signal generator to one of the inputs to the board, and the scope to the matching output. Set the signal generator to a reasonable frequency and amplitude, according to the experience you have acquired in the previous experiment.

For your report,

- Using initially a sinusoid and the low-pass filter, vary the frequency and observe if the output signal changes as expected.
- Now change the input signal to a square wave and observe the ouput signal.
- Verify if the signal produces what was expected from the model you first built and simulated.
- Report on the output observed for both signals passing through the low-pass filter, and whether there was any difference from what you expected, based on the previous simulation.

In order for you to see the output signal in the frequency domain, you must allow for the DSP platform to exchange data with `Matlab`. Only then you will be able to visualize the frequency domain representation of your output signal. You have two options to visualize the signal in the frequency domain: one is to attach a spectrum analyzer to the output of the target board; the second is to use the digital signal processor to send data back into the `Matlab` workspace. *F*rom this point on, even though you may have the spectrum analyzer attached to your target board, you will save the data in `Matlab` and manipulate it to generate plots for your report. This data will be the same data sent to the DAC, which eventually will be seen in the time domain on the oscilloscope used before. Now you will request that data be read from the DSP memory, and have that data written into a `Matlab` variable.

Since the data comes from a limited space in memory, the `Matlab` variable will be a vector of limited length. In your case, the length is pre-determined to be 1024 samples. When that is done, you can manipulate it mathematically to visualize it. You can also repeat the procedure as many times as you wish, to visualize 1024 points of data at a time. This number is conveniently chosen for you to utilize a 1024-point FFT.

From this point on to the end of this section, utilize a **square wave input**.

The procedure to be followed is presented below (you are to modify it to produce meaningful plots for the report):

- Create a *".m"* file with the following contents:

```
cc=CCS_Obj;
```

```
x=read(cc,address(cc,'storage_array'),'int32',1024);
y=double(x);
z=fft(y,1024);
subplot(2,1,1)
semilogx(abs(z))
axis([0 512 0 5e11])
subplot(2,1,2)
plot(y)
axis([0 150 -1e9 1e9])
```

- Run the file
- Modify the file and run again as you wish, to visualize the data in the frequency domain.

You may notice that the magnitude of the numbers read is very large. Remember that they are being read from memory, and not only are a result of some computations, but must be scaled properly to be sent to the D/A converter. The numbers seen obviously do not represent voltage; they must be scaled down. You can find the scaling factor by looking at them in the time domain (in `Matlab`) and comparing the numbers seen with those read on the oscilloscope (that is, after passing through the D/A converter). You can then correct the code given above to represent the actual voltage values. Note also that the display for the FFT (that's variable "z" on the code) goes up to the "number" 512. This **does not** represent frequency; it represents the 512th point of the resulting FFT. From theory, you should know that if you are performing an 1024 point FFT, the corresponding frequency at the 1024th point is actually the sampling frequency. Remember from the study of the sampling theorem that in the frequency domain, the spectrum of the sampled signal is replicated at every integer multiple of the sampling frequency.

For your report,

- Include a `Matlab` plot of a square wave signal similar to the one you are using as input to your system, and a frequency domain plot of that signal (using a 1024-point FFT);
- Include a time domain plot of the output (low-pass filtered) signal as collected from the target board, and a corresponding frequency domain plot.
- Collect data for different frequencies, and make sure to point out the distinctive characteristics of the filtered signal for every change in frequency you make.

3. **Modifying the Digital Filter in Real-Time**

Your filter is running, the results produced are what you expected initially, but you would like to verify different sets of coefficients without having to repeat the entire process. The objective here is to verify the output of a signal travelling through two different channels, which will be represented by your low-pass filter (as you experimented above) and a band-pass filter (the same one you used in the simulation part). In this part of the experiment, keep your input signal as a 1.5KHz square wave, 1Vpp. This is the signal "carrying" your information. In this part of the experiment, you will attempt to pass it through two different channels.

The software packages in use here allow for the designer to access, read and write contents in memory while the processor runs. This is to say that if you have two different sets of coefficients (usually stored as a vector in `Matlab`), you can build one filter model and test different sets of coefficients for that same model, provided that you are using filters of the same order. The **order** of a digital filter. For this case, the order of your filter will be the number of the coefficients of the numerator plus one. In this part of the experiment, you are required to modify the coefficients of your filter directly on the memory of your DSP platform while it runs. Since you have a low pass filter already designed and running, you will substitute the existing coefficients by those of a new filter, which will be a band-pass filter. Notice that when you "export" coefficients from the filter design tool, you will be required to specify names for a *numerator* and for a *denominator*. The *numerator* will hold the coefficients that you are interested in. For an FIR filter, they represent the discrete-time impulse response of the filter you have designed,and the length of this *numerator* is the order you chose for your filter, plus one.

The tricky task for this part of your lab is to identify where the original coefficients are stored in memory, so that one can read from and write to that location. Having identified where is the location of the data you wish to modify, use the commands "read" and "write" from within `Matlab` to execute your task. First, however, re-design your band-pass filter according to the specifications given in the first section. Then, utilize the File/Export tool in the Digital Filter Design block to send the coefficients into a vector (a variable) in the `Matlab` workspace (call the *numerator* variable "bpf").

On the `Matlab` command window, type "who" to assure that your new coefficients are there. With the system running, read the coefficients that are residing in memory, by typing on the command window:

```
lpf=read(CCS_Obj, address(CCS_Obj,'rtP'),'double', length_of_filter_plus_one)
length(lpf)
```

By "length of filter plus one", it is meant that what you have stored on that location in memory is the coefficients of the *numerator* of the transfer function of your filter. Then, write the new coefficients into memory by typing:

```
write(CCS_Obj, address(CCS_Obj,'rtP'), double(bpf))
```

Your band-pass filter is running now. You can use the program provided in the previous section to verify the results in the frequency domain. Include on your report a plot and an explanation for the observed output based on the theory reviewed.

For your report,

- Include plots for the output in the time and frequency domains and an explanation for the observed output based on the previous simulation and the theory reviewed.

If you want to return to the original low-pass filter, you can type:

```
write(CCS_Obj, address(CCS_Obj,'rtP'), double(lpf))
```

Note that "lpf" was the name of the variable you created when you read the coefficients from memory.

Going The Extra Mile

As you know from the theory, the process of filtering in the digital domain is done through the convolution between the sampled signal coming into a filter, and the coefficients of that digital filter, calculated according to a set of desired parameters. Up to this point, we have placed three blocks together in `Simulink` (namely, the ADC, the digital filter, and the DAC) and upon building our model, `Simulink` magically implemented the convolution.

If you would like to experiment further, break down the filtering block, by building a digital filter using unit delays and gains, following FIR topologies easily found in literature.

Another extra adventure you may want to try is to modify the C code of the generated project and make your own project. This will help you to understand many details of writing code for real-time applications.

Conclusion

In this experiment, you verified how a signal can be distorted by variations of the communication channel. Two digital filters represented the communication channels, while the signal representing "data" was a square wave signal. A sine wave signal was also used throughout the experiment, in order to allow for a verification of the characteristics of the digital filters designed and implemented. You also became familiar with some of the key issues involved in designing, simulating and implementing a digital filter. It is expected that you achieved a practical understanding of the concepts studied not only in the theory, but also in other courses of the curriculum.

References

[1] B.P. Lathi *Modern Digital and Analog Communication Systems*, 3rd Edition, Oxford University Press, 1998

[2] Oppenheim, Willsky, with Young, *Signals and Systems*, 2nd Edition, Prentice Hall

[3] S. Orfanidis, *Introduction to Signal Processing*, Prentice Hall

[4] E. Ifeachor and B. Jervis, *Digital Signal Processing - A Practical Approach*, Addison Wesley