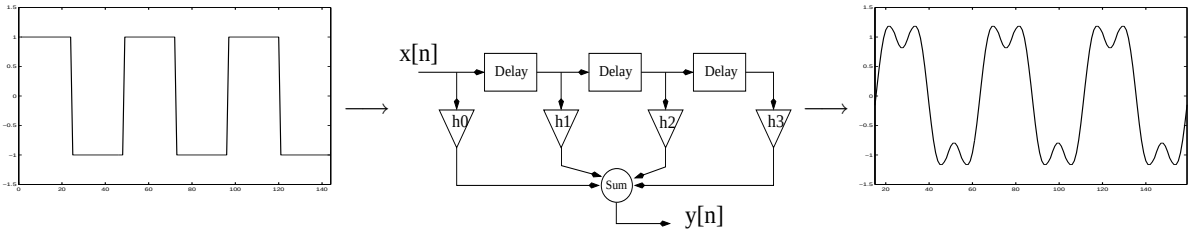
Experiment # 2

Filters



1 Purpose

The purpose of this experiment is to provide you with the basic understanding of 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 and a band-pass filter, and will verify the effects of the bandwidth variation on signal integrity. This will be accomplished by looking at the signal in both time domain and frequency domain.

A review on Fourier Series, Fourier Transform, convolution and the basics 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 involved in this practice, 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 basic 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.

2 Equipment

Hardware:

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

Software:

1. `Matlab` Release 13;
2. `Simulink` with ECE416 Toolbox;
3. Code Composer Studio, v.2.1.

3 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 can refer to [2], [3] and [4], if you want to learn more. The topics of interest are the representation 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 frequency and its odd harmonics, as described by a Fourier Series. At that time, the Gibbs phenomenon was also introduced. Such feature appears in the synthesis of periodic signals with discontinuities (i.e., square wave, saw-tooth wave, etc.). 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.

4 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 (called FDA Tool) will make use of `Matlab` capabilities to generate the desired filter coefficients. Throughout the 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. Keep in mind that your design should be simulating a system which operates at a sampling frequency of 48KHz.

Build a model for simulation, with input source, a filter (FDA Tool) and appropriate output displaying blocks. 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 Experiment # 1. You may be required to resolve pending problems related to signal shape, as it is handled by `Simulink`. 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. The trade-offs involved in designing digital filters are a topic for a full course. Your simulation will introduce you to some of them.

The questions below are related to the filter design tool that you are utilizing.

- *Present below the impulse response and the frequency response of the two filters you have designed. What values represent the coefficients of your filter?*

- *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)*

The questions below are related to the simulation of your system. If any of these steps is not clear to you, do not hesitate to ask the TA or the Lab Engineer.

- *This exercise is for you to verify the frequency response of your filters. Input a 1KHz sinusoid to the low-pass filter that you have designed. Report the values at 500Hz, 1KHz, 3KHz, 5KHz, 10KHz and 15KHz. Repeat for the bandpass filter. Your best option is to create a three-column table (Hz, $V_{pp}LowPass$, $V_{pp}BandPass$. Indicate if the results are what you expected.*

- *Now use a 1KHz square wave as input to the low pass filter, and sketch the time domain and frequency domain results. Explain the results.*

- *Repeat the procedure for 2KHz. Explain your results.*

2. **Building and Running a Model**

In this part, you will insert the filter block (FDA Tool) into the "template" system that you built in Experiment#1. You will find the sytem with all the proper parameters pre-set in the `work/template/direct.mdl`. Run the model (i.e., "play") 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. Press the `build` button on your model window.

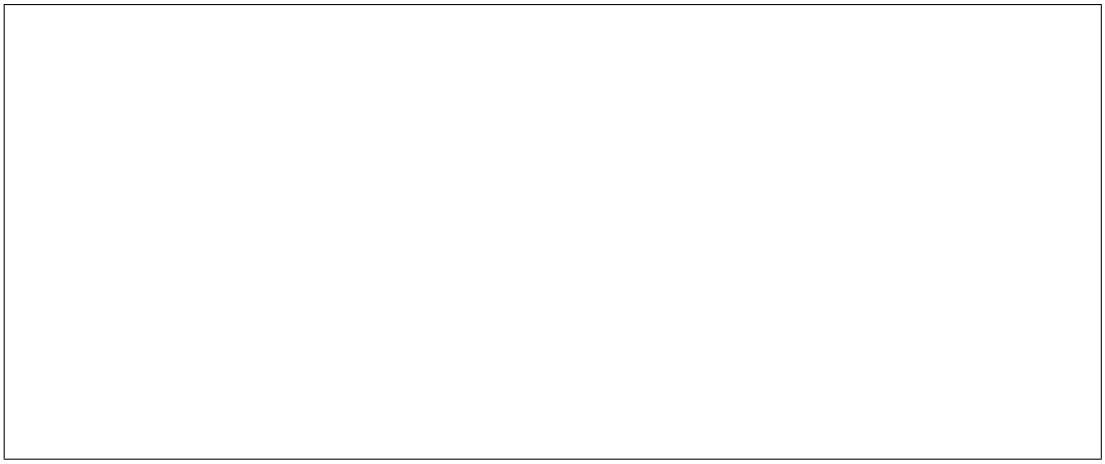
After the code for your model has been generated, `Matlab` will create a project and send it into Code Composer Studio. You will have to compile (by pressing the "build all" button in CCS) and load the executable file into the target hardware. Note that there are two **build** processes in question: one is for `Matlab` to build a project (i.e., generate C files) based on your block diagram and send it into Code Composer Studio; the other is the building of an executable file in Code Composer Studio, by compiling, linking and assembling the files generated from `Matlab`. The latter is done for a specific target hardware, which in your case is the TMS320C6711 DSK.

Connect the signal generator to the inputs to the board, and the scope to the outputs.

Now you must load the executable file into the target hardware by selecting `File/Load Program`. After this is done, you press on "Run". You may not see any output on the first time you do it. In case this happens, just load the program again and 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. See if you can find your filter coefficients, for instance.

- *Using a sinusoid as your input signal ($1V_{pp}$), determine the cutoff frequency of your filter. Draw a frequency response graph with a minimum of 5 points.*

- *Set your input signal to a 1KHz, $1V_{pp}$, square wave. Draw the time domain output and explain what it represents.*



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 to answer questions. The data retrieved will be the same data sent to the DAC, which eventually will be seen in the time domain on the (real) oscilloscope. 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. After the data is acquired, 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. Sometimes you may observe a "discontinuity" on the signal retrieved. That is due to `Matlab` capturing the data while the buffer (the space allocated in memory) is being written with new data. If that happens, just capture the data once again. The length of 1024 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**. You should be aware of the fact that you have this signal input to two channels, and only **one** of the channels will actually have a filter in it.

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:

```
close all, cc=CCS_Obj;
x=read(cc,address(cc,'channel_a'),'double',1024);  % channel A
xx=read(cc,address(cc,'channel_b'),'double',1024);  % channel B
z=fft(x,1024);
zz=fft(xx,1024);
figure
subplot(2,1,1), plot(abs(z)), axis([0 512 0 300])  % using 1Vpp input
subplot(2,1,2), plot(abs(zz)),axis([0 512 0 300])
figure
subplot(2,1,1), plot(x), axis([0 150 -3 3])
subplot(2,1,2), plot(xx), axis([0 150 -3 3])
```

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

This program reads directly from the memory on the target hardware. All samples that are produced on two separate channels are stored in two buffers prior to being sent to the digital-to-analog converter (DAC). These buffers are called 'channel_a' and 'channel_b'.

In general lines, the program

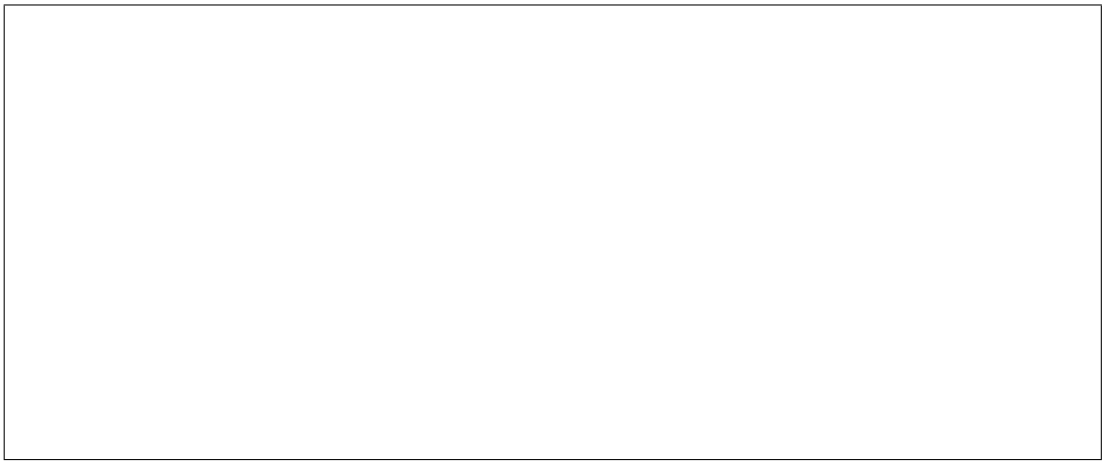
- closes all previous graphs to avoid multiple plots on your workspace;
- assigns all hardware communication parameters to variable cc; it reads 1024 samples of format 'double' from address 'channel_a' on the target defined by variable cc and assigns these values to variable x;
- does the same for 'channel_b' and assigns to variable xx;
- takes a 1024 point FFT on the 1024-sample long variable x, and assigns the results to variable z;
- does the same for variable xx, assigning to variable zz;
- opens a new figure (a window which will hold the plot);
- creates the first (of two) plot, which will display the absolute value of the first 512 values of z (as defined by the axis command);
- creates the second plot, which will display the absolute value of the first 512 values of zz;

- opens another figure
- creates the first plot, to display the first 150 values of variable x (as defined in the 'axis' command);
- creates the second plot, which will display the first 150 values of variable xx;

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.

Use the code given above (modifying it when necessary) to respond to the following items:

- *Using a 1000Hz, $1V_{pp}$ square wave, draw the output in both time domain and frequency domain. Indicate the frequency components on the frequency domain plot.*



- *Compare the filtered output with the "direct" output. Explain what you see. Modify the input frequency to 2KHz and compare the outputs again. Explain what you see.*

5 Extra-class activity

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 (refer to your preparation).

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.

6 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