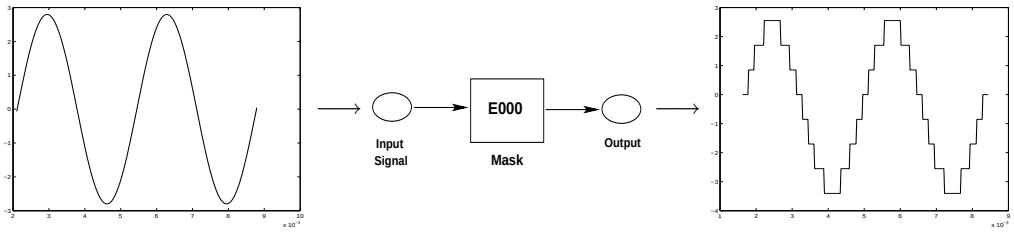
Experiment # 5

Pulse Code Modulation



1 Purpose

The purpose of this experiment is to introduce Pulse Code Modulation (PCM) by approaching this technique from two fronts: *sampling* and *quantization*. PCM is a technique widely used in communication systems, in particular in the conversion of analog signals into their digital representation.

PCM will be explored in this experiment in three manners:

- First, the validity of the sampling theorem will be verified through a simulation in `Simulink`.
- Second, you will verify the results of this simulation on the DSP target hardware. You will vary the sampling rate applied to the input signal by *downsampling* it. In this part of the experiment, you will observe *aliasing* occurring in practice, on the output signal.
- Third, you will force a change on the number of quantization levels utilized to convert the input signal. This will be done by applying a bit *mask* on the incoming samples. You will verify the consequences of this limitation on the signal displayed on the oscilloscope and also by retrieving data from the hardware into `Matlab`.

2 Background Reading and Preparation

It is fair to say that every textbook in digital signal processing starts by describing how an analog signal is converted into a digital one prior to being processed. Sampling and quantization are described in detail in many references. Therefore, rather than looking for a single reference for background reading, you can refer to any of the textbooks cited in the bibliography presented at

the end of this outline. Keep in mind, however, that these should be seen as introductory reading, since they only touch on PCM as applied to A/D conversion. The textbook for this course ([1]) looks at PCM from the point of view of communication systems going well beyond A/D conversion. Another good reference is [2]. You should review your notes on aliasing and downsampling from introduction to signals and systems, and bitwise operations from microprocessor theory.

Before coming to the lab, complete the **lab preparation** and hand it to the T.A.. If you *still* are not comfortable with `Matlab` and `Simulink`, come to the lab and practice on the workstations.

3 Equipment

Hardware:

1. One Signal Generator;
2. One Two-Channel Oscilloscope;
3. One Spectrum Analyzer
4. One TMS320C6711DSK Module (with audio daughtercard) 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

4 Experiment

The experiment is divided in three parts: the simulation of a system presenting aliasing; the modification of sampling rates to verify the occurence of aliasing on real signals; and the verification of the relation between signal integrity and word length in the quantization process. Differently than the previous experiments where you verified the concepts by designing a "whole" system, this experiment will touch individually on the two main parts of PCM, namely: sampling and quantization. The reason for taking this approach is primarily due to the fact that the target hardware already provides a CODEC which utilizes PCM to convert the signal from analog to digital. In this experiment you will modify the data generated by the existing CODEC and observe the consequences of the modifications on the output signal.

All results are to be reported in the spaces provided in this outline. Make sure that the T.A. verifies every result you record.

4.1 Sampling a Signal At The Wrong Sampling Rate

Since in every other experiment you started out trying to design the *right thing*, in this experiment you will be required to put in an extra effort to start by designing the *wrong thing*. It should come with no surprise that it is simpler than you might expect. Departing from a system working as it should, you will modify the sampling rate of the incoming signal to observe how **not** to sample it. This will be done with the use of the **downsample** block, found on the ECE416 blockset under `Simulink`. Downsampling is also known as **decimation**. The textbook representation of decimation in block diagrams is a block containing an arrow pointing down followed by a number as the **decimation factor**. As the name implies, when you decimate or downsample a discrete signal by a factor, you *reduce* the sampling rate of the signal by that factor. This is to say that you will discard a number of samples in the process. For instance, if you have a 1KHz signal sampled at 48KHz, you will have 48 samples per period. When you decimate this signal by a factor of 2, you will discard every other sample and end up with 24 samples per period. This is equivalent to sampling the 1KHz signal at a 24KHz sampling rate.

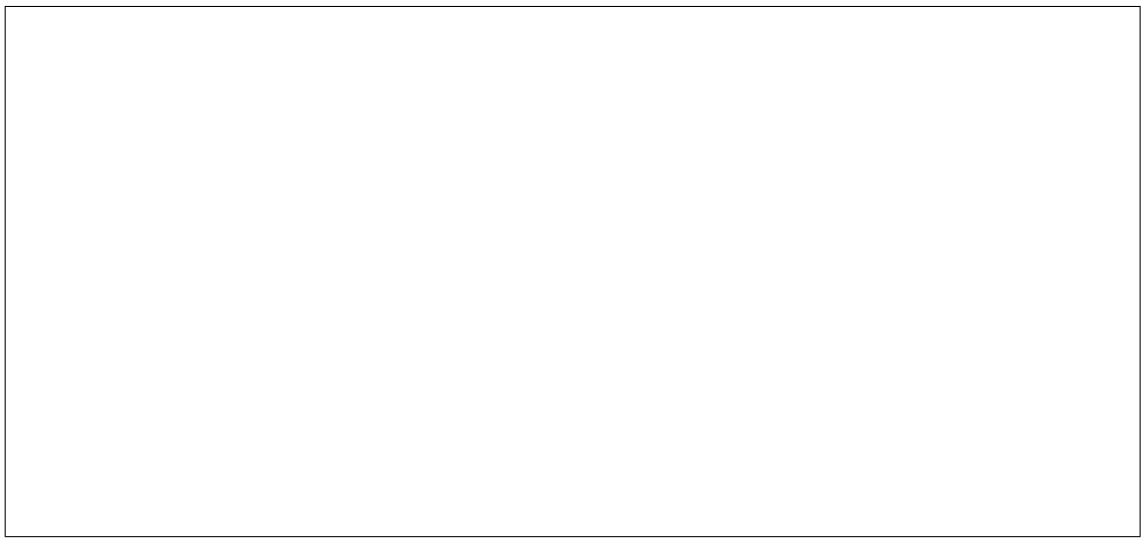
Before you start, remember that:

- All blocks that you will use to build your system are in the `ECE416` blockset on `Simulink`.
- The input to an FFT-based scope must be buffered, with a buffer size that is an integer multiple of the FFT size (could be the same size). It makes no sense to attempt to perform an FFT on a single sample;
- A greater FFT length means a more accurate frequency domain representation of the signal (take 1024 points as a reasonable length). At this point you should be able to tell when the results are producing relevant information or insufficient information;

Start by opening a new model on `Simulink` with a DSP Sine wave generator attached to one time scope and one FFT scope. Adjust your sine wave to be $2V_{pp}$ and $22KHz$, using a $48KHz$ sampling rate. Set yor Spectrum Scope to display frequencies between 0 and f_s. Make sure that this runs smooth. Now place the decimation block between your sine wave generator and the scopes. Set the decimation factor to be 2 and select "force single rate". Run your simulation.

Be prepared to answer questions asked by the T.A.

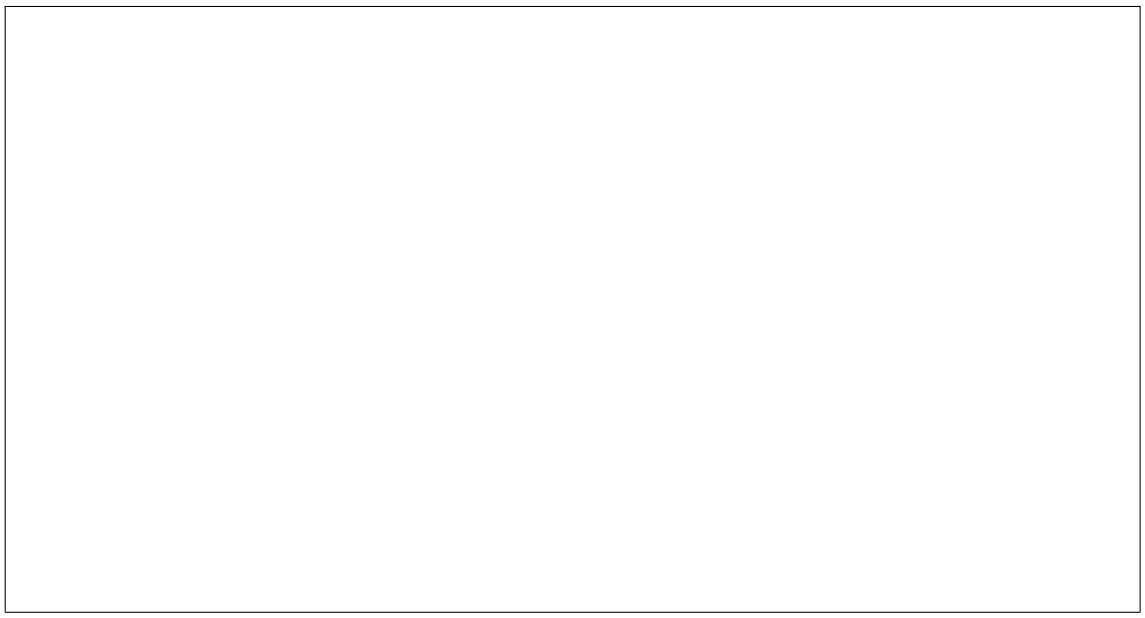
- *In the space below, sketch the output signal observed in the time domain and frequency domain after the decimation. Point the frequencies in which the components are found. If you cannot read them, calculate them.*



- *Increase the decimation factor to 6. What is the new sampling rate? Sketch the output signal observed in the time domain and frequency domain. Show the values of the frequency components. Is it appropriate to sample a 22KHz signal at this rate? Why?*

Now you will observe the same effect on a real signal. Insert the **downwsample** block on one of the signal paths on the model found in `work/template/direct.mdl`. Your objective is to have the signal travel without modification on one of the channels and have the signal decimated on the other channel with a decimation factor of 6. Save your model and download it to the DSP target hardware. Compile the code generated, load the executable code and run your system. Use a $22KHz$, $2V_{pp}$ sine wave as input to your system.

- *Use the* `Matlab` *program to display the output data for the two channels in the time and frequency domain. Sketch the results below. Report the necessary values. If you would like to double-check on your values, you may use the FFT capability on your oscilloscope.*



4.2 Uniform Quantization And Number Of Bits

This part of the experiment will be done without simulation. You will build a system and download it to the DSP target hardware, and modify the number of bits utilized to pass the incoming data to the DAC. You have already verified in the first experiment of this course that if you build a "straight-through" system, the sinusoid obtained at the output is a faithful representation of the sinusoid you input to the system. This means that the quatization done by the CODEC is done in enough levels to provide for a very good representation of the incoming signal, as it was sampled at 48KHz. In this part of the experiment, you will limit the number of bits used in this quantization, and observe the results on the oscilloscope. As you have already reviewed, when you reduce the number of bits on the quantization, you increase the error introduced in the process by reducing the number of quantization levels available. The question to be asked is: how much error can you tolerate for your application?

The technique you will utilize to modify the word length of the incoming signal is known as "masking", and you likely have already utilized it in a Microprocessors course. It consists of performing a logical AND operation between the data word passed on to the DAC and a "mask" that you select. The CODEC utilized by your DSP target hardware operates with a 32bit word, representing a two-channel (stereo) signal. Each channel is represented by half of the word (16 bits). For example, if you are looking to select one channel of this 32-bit word, you should apply a mask defined by the hexadecimal number **0000FFFF**.Keep in mind, of course, that the size and format of the data word depends on the hardware being utilized. The values for masks that you will use here are specific for your DSP target hardware, but the concept is the same across platforms.

From the model you have just utilized to verify decimation, you will substitute the downsampling block by the **masking** block, found on your `ECE416` blockset. This block performs the operations necessary to select the 16 bits representing one channel. The block comes with a pre-defined mask of **FFFF**. After you insert the masking block in your model, save the model and start the downloading process.

While you go through the steps defined below, you will be required to modify the mask applied to the channel. In order to make it faster, you will modify the mask directly on the C code generated. The fastest way to make modifications and get your system running again is this:

- From the project tree, double click on the "**.c**" file named after your model. It should be found under "source";
- Search for the word "mask". This is done by typing the word on the blank field beside the "binoculars" icon found at the top toolbar in Code Composer Studio. If you click on the icon showing a folder beside a pair of binoculars, a new window will appear for a more specific search. After you do the search, the line of code that you will modify is the one immediately following the comment containing the word "mask";
- Now go to the menu under option/customize and click on "program load options". Check the box for "load program after build". This will cause the executable to be loaded automatically every time you compile your program;
- One feature you might also want to try this time is the "incremental build", rather than the "build all". The incremental build will only compile the modified files and link them to the previously compiled versions of the unchanged files. This is done by clicking the blue button to the left of the build all button that you normally use.

Use an input signal of around $3.5V_{pp}$ and $100Hz$ to answer the questions below. The reason why you are using $3.5V_{pp}$ is because you want to use the full range of the CODEC. Keep in mind that some of the signal generators need to be set to half of that value due to impedance mismatch. If you do not have the full range of the signal going into the CODEC, your masking experiment will not give you the right results.

- *Set your mask to be* **00008000**. *Sketch your result and explain what is the mask selecting.*

- *Change your mask to reduce the data word to allow for 32 quantization levels. What is the mask to be utilized?*

- *Sketch the output produced if a 2 bit data word is utilized to quantize the input signal. Now reduce the input voltage level until you jump to the lower quantization step. What is the value of that step in volts? It may be a good idea to overlap the direct signal with the quantized one, and see how that happens.*

- *Suppose you could hear the signal coded with 3 bits, uniform PCM. Do you expect to hear any difference in the signal?* ***Should*** *you hear any difference in the signal? You are to base your answer on objective measurements. However, assessments such as this in the industry are often done subjectively by using the responses from a group of people.*

5 Going The Extra Mile

No extra mile. That's enough.

6 Conclusion

In this experiment, you acquired a better understanding of Pulse Code Modulation by further probing into sampling and quantization. You verified the introduction of aliasing by sampling a signal at a rate lower than that dictated by the sampling theorem. You also verified the consequence of reducing the quantization levels by limiting the number of bits utilized to represent a signal. The concepts of sampling and quantization are fundamental to a multitude of areas in engineering.

Obs: The books cited in this outline contain relevant material for the student to better understand the topics pertaining to the experiment. The student should note that by reading the course textbook only, one should be able to successfully perform the experiment. This is to say that it is not necessary for the student to purchase all the references.

References

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

[2] S. Haykin *Communication Systems*, 4th Edition. Toronto: John Wiley & Sons, Inc., 2001.

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

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

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