Utah State University
Real-Time Digital Signal Processing Laboratory
ECE 5640

Lab 5: Real-time Spectral Analysis Using the FFT

Objective

One purpose of this lab is to use the FFT to implement a real-time spectrum analyzer. This requires the use of block processing of input data, a common type of real-time processing.

A second purpose to to learn how to incorporate library functions in your code, which have been written by someone else. You must study and understand the documentation for the code in order to successfully use it in your program.

References

1. *OMAP-L138 Datasheet*, Texas Instruments.
2. *OMAP L138 Technical Reference Manual*, Texas Instruments.
3. *TMS320C674x DSP CPU and Instruction Set Reference Guide*, Texas Instruments.
4. On-line documentation for *Code Composer Studio*.

Required Equipment

The following equipment is necessary to complete the lab:

1. The Texas Instruments OMAP-L138 Experimenter Kit.
2. A Analog Discovery soft spectrum analyzer and signal generator.
3. An oscilloscope.
4. Cables for connecting the system to the spectrum analyzer/generator and oscilloscope.

Required Software

The following software is necessary to complete the lab:

1. The 'C67 *Code Composer Studio* compiler and debugger
2. The OMAP-L138 Experimenter board support library (BSL): `evmomapl138_bsl.lib`.
3. The `.zip` archive file `template2.zip`.
4. The `.asm` radix-2 FFT routine file `67-cfftr2.asm`.
5. The `.asm` bit reversing routine file `bitrevf.asm`.

Background

One of the most commonly used algorithms in digital signal processing is the FFT. It can be used to provide an estimate of the spectrum of an input signal, pattern recognition, speech synthesis, etc. It is often necessary to use the FFT for real-time signal analysis, which requires efficient computation of the algorithm and proper data handling.

Procedure

1. Implement a real-time spectrum analyzer using the FFT. As in previous labs, be sure to write the entire program (except the initialization of the twiddle factor tables and codec) in assembly code. The ISR should be coded *entirely* in linear assembly. The program should be able to take an input sampled at 8 kHz and provide an analog output of the signal spectrum which can be displayed on an oscilloscope. The FFT should be 1024 points long with no overlap of input points, and the output of the program should be the *squared-magnitude* of the first half of the FFT.

   Keep in mind the following points:

   (a) Compute the FFT using the radix-2 code supplied by TI (`67-cfftr2.asm`). Please read the comments in the code to understand how the algorithm is implemented. If you want, you can modify the code to remove unwanted sections or initializations to speed up execution. Remember that a call to a function from a linear assembly function is performed using the `.call` statement as in

   ```
   .call  _cfftr2_dit(arg1, arg2, ...)
   ```

   After the FFT is computed, you must "bit reverse" the data to order the output data in proper order. This can be done with the TI-supplied assembly function `bitrev()` which is in the file `bitrevf.asm`. A bit reversing indexing table can be created during the call to the function `digitrev_index()` (Refer to the `bitrevf.asm` file for details.) You should generate the table *once* in C code as part of the initialization of the program.

   The twiddle factors used in the FFT must be bit reverse indexed. The `cfftr2_dit()` routine uses N/2 complex twiddle factors, and the bit reversed values are based on N/2 indices.

   The (complex) result of the FFT will be stored in the same location as the original input data. You must use the first 512 (complex) points of the FFT to compute the squared-magnitude result and write it back to the buffer. Remember the 512 outputs of each FFT need to be written out twice for each block of 1024 input samples that are processed. (It may be easiest to just copy the 512 output samples to the second half of the buffer.)

   (b) Due to the optimized pipelining in `cfftr2_dit()`, you can not interrupt the FFT while it is processing the data. Assume you have stored a block of input samples. After the FFT is completed on this data, you must implement a method to store incoming samples and write out outgoing samples at the same time for each block of samples that follow. Once all of the output samples in the buffer are written

to the D/A, (and the buffer is filled with recent input samples), the FFT can be performed on the buffer. This is then repeated for each block of input samples.

Remember that if you aren't able to complete the FFT in a sample time, it might be difficult to have the chip recognize the next interrupt, and it might quit responding to input samples. This means you must be efficient in handling the input and output data so your data handling and an FFT can be performed in one sample time.

(c) Remember that a sync pulse must be output in one of the two output channels together with the spectrum so that the oscilloscope can be synchronized to the output spectrum. A simple way to do this is to generate a square wave with two rising edges per block; one at the beginning of the buffer and the other at the beginning of the second half (start of second spectrum) of the buffer.

(d) To use the Analog Discovery soft spectrum analyzer to view the output spectrum by triggering on the generated square wave:

i. Open the scope module in the waveforms software.

ii. Set the time base to 6.4ms/div.

iii. Set the source to your square wave channel (1 or 2).

iv. Drag the horizontal cursor at the top of the scope display all the way to the left of the display.

v. Properly scale the V/div for each channel to view the output spectrum.

(e) Input samples are captured in the `int_16` format. However, the FFT procedure expects the input samples in float format. Use the `intsp` instruction to cast from an int to float. Also note that the FFT procedure expects each sample to have a real and imaginary part (which is zero in our case).

(f) Be careful of overflow. If the squared magnitude of the floating-point FFT bins is greater than a short int can represent, converting these to short ints will generate garbage values. you must scale or otherwise prepare the values for conversion to short ints.

(g) You can check for proper operation of the FFT routine by using an initialized known input and graphing the FFT output after stopping the processor during debugging. The graphing capability of `Code Composer Studio` is available under the `View->Graph` drop-down menu.

(h) Remember to scale the output before you convert to integer so that the output samples will not saturate the D/A converter.

2. Start your spectrum analyzer and use a sinusoid at 3 kHz as an input. Observe the output on the oscilloscope by syncing the oscilloscope to the output signal. (Sync to a pulse generated by your program on the second channel of the output.)

   Since the FFT is 1024 points long and the sampling rate is 8 kHz, continuous (real-time) block processing of all input samples requires that you output a block of samples in

$$\frac{1024 \text{ samples}}{8,000 \text{ samples/s}} = 128 \text{ ms}.$$

(Remember that you will output half of the FFT *twice* per block of samples.) You should therefore be setting the timebase to trace across the screen twice within a block time. Since the oscilloscope screen has 10 divisions, this means that it should be set to trace at about 6 ms/division.

Connect the input in parallel to the Analog Discovery spectrum analyzer. Does the output of your spectrum analyzer match the output of the Analog Discovery spectrum analyzer? Why or why not?

3. Change the input to a square wave at 2 kHz. How does your output compare to the spectrum analyzer?

Questions

1. Approximately how many 1024 real FFTs can you compute during each block of 1024 samples acquired at 8 kHz using your program? Do you think you could increase the FFT size or overlap points and still complete the processing in time?

2. Did your output spectrum look like you expected it to look? Describe your results.

3. Discuss how useful your spectrum analyzer would be for understanding the spectrum of a signal. State any advantages or limitations of your implementation.