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.

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.)

3/20/19

(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.

3/20/19

What is k in digitrev_index

radix=2 nbits=bitsN)

radix2 = radix >> 1 = 1
nbot = nbits >> radix2
ndiff = nbits & radix2
ntop = nbot + diff
n2 = 1 << ntop

$n = 2^k$ $n = N = 1024$

General flow

```
Interrupt(){

left = Input Sample()
  right = 0
  X[i] = left
  X[i+1] = right
  i += 2
  if(i = 2N){                    ∈ i = 0
      FFT(x)
      Bit reverse(x)
      for(i = 0; i < N-1; i += 2){
          X(i) = X[i]^2 + X[i+1]^2     squared magnitude
          if(X(i) > max) max = X(i)
          X(i+N) = X(i)                copy over
      }
      if(max > 2^16)
          scalar = 2^16 / max
  }

  left = X(i) * scalar
  right = ((i%2N) > 200) ? 1 : 0       square wave

  output Sample(left, right)
}
```

3/20/19

38

had issues with interrupt in assembly. Occured once only. Found out that the GIE bit is set to 0 when a interrupt is called. Set GIE to 1 at end of interrupt

Figure 1 FFT of 3 kHz sine wave. (Top) Spectrum analyzer FFT, (Bottom) DPS chip FFT output.

I have apt pulse at ~ 1kHz for some reason. My FFT outputs an impulse at the sinusoid frequency and due to the DAC, that impulse becomes a sine function. There is also some noise at the edges of the FFT (0, 4kHz)

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?

3/29/19

Figure 2 Sinusoid sweep between 1 Hz and 4 kHz over 5 seconds. Top is the output at roughly 2 seconds. Bottom is the output at roughly 4 seconds.

This is a snaphot of a frequency sweep between 0Hz and 4kHz over 5 seconds to show how the fft updates. The magnitude is less than in Figure (1) and instead of an impulse, the frequencies are more spread to show how the freqeuency is increasing.

My estimate would be that the integral over the activity in the fft for figures (1, 2) would be the same

3/29/19

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

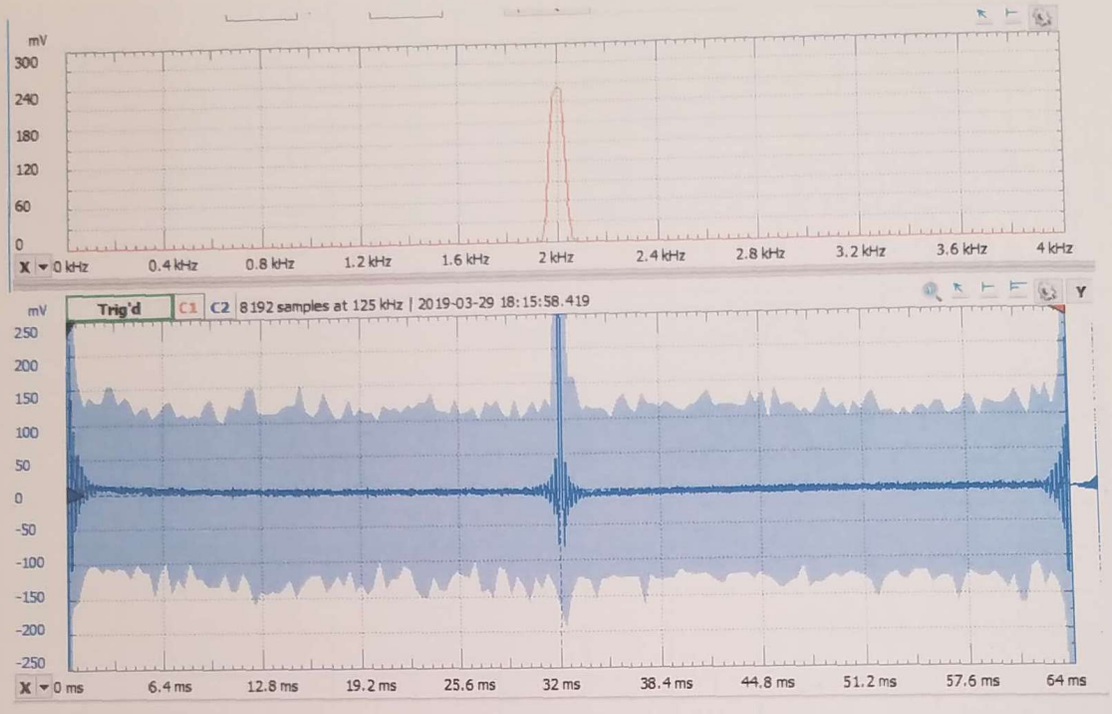


Figure 3 Square sinusoid signals at 2 kHz. (Top) Spectrum analyzer FFT of the square wave and (Bottom) DSP chip FFT output of the square wave.

Both signals have the same amplitude, but the DSP FFT is a sinc function and the spectrum Analyzer is more of a hump. Looks almost identical for square vs sin functions

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?

Using a loop to compute the FFT multiple times, I could only run the loop once. If further optimization was made, I could probably fit 2 FFTs in 1 sample time

The FFT size could potentially be increased. As the FFT was computed in 1 sample time, there can be overlap in samples where the FFT could be computed every N/2 (+512) samples

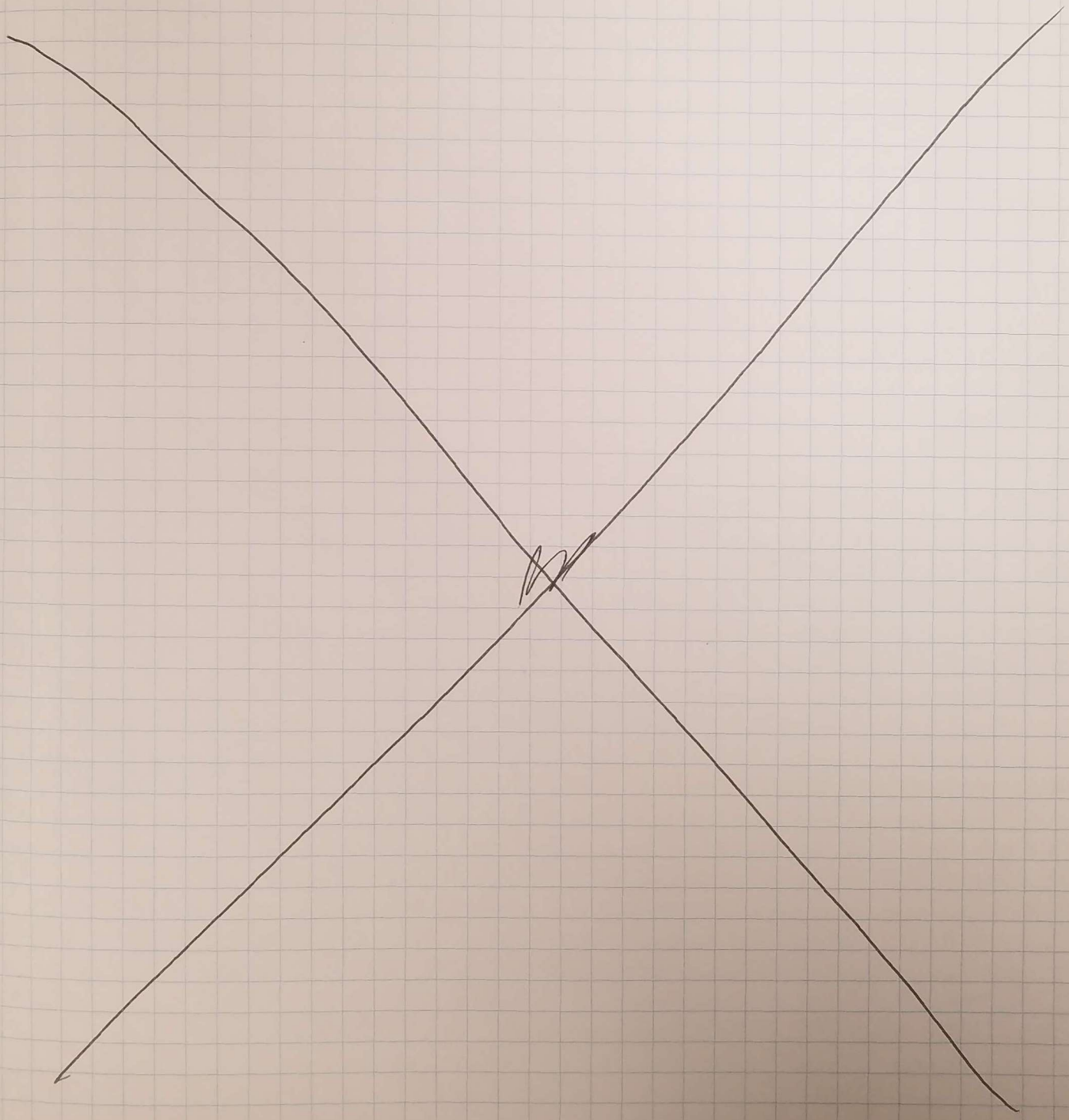
2. Did your output spectrum look like you expected it to look? Describe your results.

The ouput waveforms were located at the correct frequencies, but the pulse shape is different. The Discovery FFT is more of a hump, the DSP FFT is a sinc

3/22/19
MV

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

The spectrum can be used to identify the magnitudes of various frequencies in a signal.
My implementation outputs the signal as squared magnitude, while other ffts seem to default to a logrithmic scale with dB
Another disadvantage is the system can only run at 8kHz so higher frequencies will Alias.
This system is good for frequencies below 4kHz



3/29/19

```
        .title "interruptFFT.sa"
        .def _interruptFFT
        .sect ".interrupts"
        .global _input_sample
        .global _input_right_sample
        .global _output_sample
        .global _cfftr2_dit
        .global _bitrev
        .global _xBuff
        .global _w
        .global _index
        .global _i
        .bss _scalar,4,4                                        ;assembly global

_xBuff  .usect "arraySect", 8192, 4             ;allocate space for xBuff[1024*2]
_w              .usect "arraySect", 4096, 4             ;extern float w[1024];
_index  .usect "arraySect", 4096, 4             ;extern short index[1024];
_interruptFFT  .proc B3
        .reg inputLR, sample, outputLR, xBuff, w, index, p_i, i, j, N, boolVal, mask, max, cnt
        .reg real, imag, max16, scalar, p_scalar, constants, temp, leftChan, rightChan
        mvkl _xBuff, xBuff
        mvkh _xBuff, xBuff
        mvkl _w, w
        mvkh _w, w
        mvkl _index, index
        mvkh _index, index
        mvkl _i, p_i
        mvkh _i, p_i
        mvkl _scalar, p_scalar
        mvkh _scalar, p_scalar
        mvk 1024, N

        ldw *p_i,i
        .call inputLR = _input_sample()                 ;obtain 2 int_16

        mvkl 0x0000FFFF, mask
        mvkh 0x0000FFFF, mask                           ;mast upper 16 bits
        AND mask, inputLR, rightChan
        intsp rightChan, sample                         ;right channel (data)
        stw sample, *+xBuff[i]
        add i,1,i

        shru inputLR, 16, leftChan
        intsp leftChan, sample                          ;shift upper 16 bits
        mvk 0, sample
        stw sample, *+xBuff[i]
        add i,1,i                                               ;left channel (0)

        mvk 0x800, temp
        cmpeq temp,i, boolVal
[!boolVal] b Skip_FFTLogic
        mvk 0, i                                        ;branch if i!=N
        mvk 0x1, mask
        NOT mask, mask
        mvc CSR, temp
        AND temp, mask, temp
        mvc temp, CSR                                   ;diable GIE
        mvk 1, cnt                                              ;may not be needed
LoopFFT:
```

3/29/19

```
        .call _cfftr2_dit(xBuff, w, N)                  ;compute FFT
        sub cnt, 1, cnt
[cnt] b LoopFFT
        .call _bitrev(xBuff, index, N)                  ;bit reverse the output

        mvk 0, max
        mvk 512, cnt
for1: .trip 1024
        ldw *xBuff, real                                ;***could have used lddw
        ldw *+xBuff[1], imag
        mpysp real, real, real
        mpysp imag, imag, imag                          ;compute the square magnitude
        addsp imag, real, real                          ;real^2+imag^2
        mvk 512, temp
        stw real, *+xBuff[N]                            ;store magnitude into x
        stw real, *xBuff++[2]                           ;store into 2nd half of x
        cmpltsp max, real, boolVal
[boolVal] mv real, max

                                                ;if(max<xBuff[j]) max=xBuff[j]
[cnt]   sub cnt, 1, cnt
[cnt]   b for1
        mpy N,2,temp
        subaw xBuff, N, xBuff                           ;set back to xBuff[0]

        mvk 1, scalar
        intsp scalar, scalar ;1.0f
        mvkh 0x47800000, max16                          ;constant for 2^16
        cmpgtsp max, max16, boolVal
[!boolVal] b skipScalar
        mvkl 0x47000000, constants                      ;float 0.5*2^16
        mvkh 0x47000000, constants
        rcpsp max, max                                  ;1/max
        mpysp constants, max, scalar                    ;compute scalar
skipScalar:
        stw scalar, *p_scalar                           ;store for later interrupts

Skip_FFTLogic:

        ldw *+xBuff[i], leftChan                        ;get the magnitude
        ldw *p_scalar, scalar                           ;get the scalar
        mpysp leftChan, scalar, leftChan                ;scale output
        spint leftChan, leftChan                        ;float to int

        mvk 0x3FF, mask
        AND i, mask, temp                                       ;i%1024
        mvk 200, constants
        cmplt temp, constants, boolVal                  ;if(i%1024>200)
[boolVal] mvk 25000, rightChan
||[!boolVal] mvk 0, rightChan
        pack2 leftChan,rightChan, sample        ;lower 16 bits get combined into 21 bits
        .call _output_sample(sample)                    ;output sample

        stw i, *p_i                                             ;store i
        mvk 0x1, mask                                   ;enable interrupt
        mvc CSR, temp
        OR temp, mask, temp
        mvc temp, CSR
        .endproc B3
        b B3
        nop 5
```

3/29/19
MV