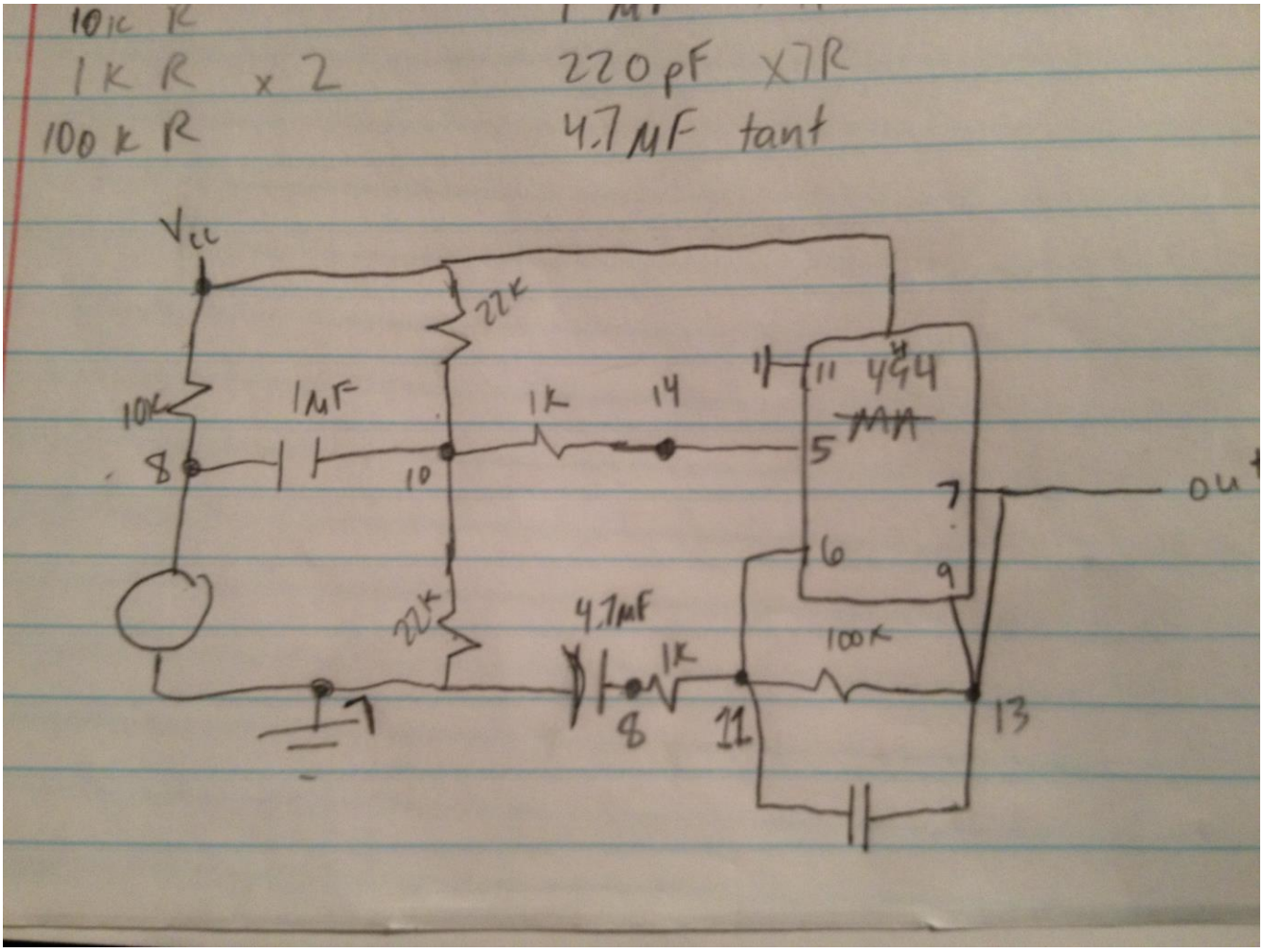
Sam Caldwell
Rohith Prakash

EE445M Lab 4 Report

**OBJECTIVES**

The objective of this lab was to design and implement various analog and digital filters to interface a microphone to record sound and display voltage and frequency information in real-time. This lab required a high pass, low pass, and finite impulse response filter. The HPF and LPF were analog filters attached to the circuit which interfaced the microphone with the ADC. The FIR filter was digital (done in software), and was bandpass, attenuating signals whose frequencies were outside of the specified range. The last part of this lab was motivated by the desire to measure and display voltage vs. time data as well as the spectral analysis of input signals. Real-time data in the frequency and time domains are displayed on the OLED, calculated from the ADC inputs. All of these functions were implemented on top of our OS which used priority scheduling and blocking semaphores.

**HARDWARE DESIGN**



**SOFTWARE DESIGN**

```
unsigned long X[FILTER_LEN] = {0,};
unsigned long Y[FILTER_LEN] = {0,};
static const long H[FILTER_LEN] = {0,0,-1,-1,0,1,-1,-6,-8,-2,5,
    3,-11,-22,-12,11,17,-10,-46,-41,16,68,28,-125,-309,
    888,-309,-125,28,68,16,-41,-46,-10,17,11,-12,-22,-11,
    3,5,-2,-8,-6,-1,1,0,-1,-1,0,0};

long buffers[3][BUFF_LEN] = { {0,},};
long *inputBuff, *outputBuff, *unusedBuff;

void FIR_Filter(void) {
  static int index = 0;
  unsigned long sample;
  long *temp;

  inputBuff = buffers[0];
  outputBuff = buffers[1];
  unusedBuff = buffers[2];
  RIT128x96x4PlotClear(0, 1023, 0, 1, 2, 3);

  while(1) {
    int i;
    for(i = 0; i < BUFF_LEN; i++) {
      sample = OS_Fifo_Get();
      if(FilterEn) {
        int i, sum;
        X[index] = sample;
        sum = 0;
        for(i = 0; i < FILTER_LEN; i++)
          sum += H[i]*X[(index + FILTER_LEN - i)%FILTER_LEN];
        Y[index] = sum/256 + 512;
        sample = Y[index];
        index = (index + 1)%FILTER_LEN;
      }
      inputBuff[i] = sample;
                  }
    // rotate buffers - possible race condition if the input can fill two buffers faster
    // than the output can go through one (unlikely)
    temp = unusedBuff;
    outputBuff = inputBuff;
    unusedBuff = outputBuff;
    inputBuff = temp;
    // Scope prints from output buffer
    OS_AddThread(&Scope, 64, 0);
  }
}
```

```
long FFT_in[64], FFT_out[64];
void Scope(void) {
  unsigned long FFT_output;
  // make sure their is only one thread printing at a time
  OS_bWait(&PrintSemaphore);
  // clear points on plot
  RIT128x96x4PlotReClear();
  if(ScopeMode == PLOT_TIME) {
    // plot voltage versus time
    // plot two Y values for every X value for a more connected look
    int i;
    for(i = 0; i < BUFF_LEN; i++) {
      if(VLindex < BUFF_LEN) {
        VoltageLog[VLindex++] = outputBuff[i];
      }
      RIT128x96x4PlotPoint(outputBuff[i]);
      if(i % 2) {
        RIT128x96x4PlotNext();
      }
    }
  }
  else {
    // plot voltage versus frequency
    // calculate using every other sample from outputBuff (throw away half the data)
    int i;
    for(i = 0; i < 64; i++)
    {
      FFT_in[i] = outputBuff[i];
    }
    cr4_fft_64_stm32(FFT_out, FFT_in, 64);
    for(i = 0; i < 64; i++)
    {
      long real, complex;
      real = FFT_out[i]&0xFFFF;
      complex = (FFT_out[i]&0xFFFF0000) >> 16;
      FFT_output = sqrt(real*real + complex*complex);
      if(FFT_LogIndex < 64) {
        FFT_Log[FFT_LogIndex++] = FFT_output;
      }
      plotBar(FFT_output);
//       RIT128x96x4PlotPoint(FFT_out[i]);
//       RIT128x96x4PlotNext();
    }
  }
  RIT128x96x4ShowPlot();
  OS_bSignal(&PrintSemaphore);
  OS_Kill();
}
```

```
void plotBar(long point) {
  int i;
  point *= 10;
  if(point < 0) {
    point = 0;
  }
  else if(point > 1023) {
    point = 1023;
  }
  for(i = 0; i <= point; i++) {
    RIT128x96x4PlotPoint(i);
  }
  RIT128x96x4PlotNext();
}
```

**MEASUREMENT DATA**

For all oscilloscope shots, 1 is the input and 2 is the output.

Procedure 2:

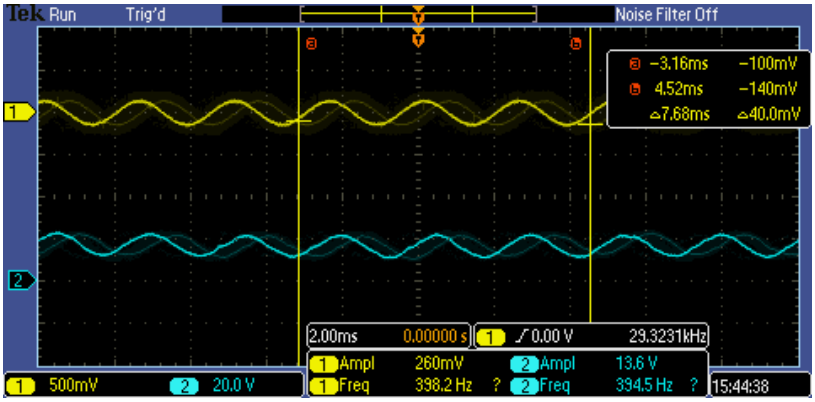



Figure 1. Input and Output at 400 Hz

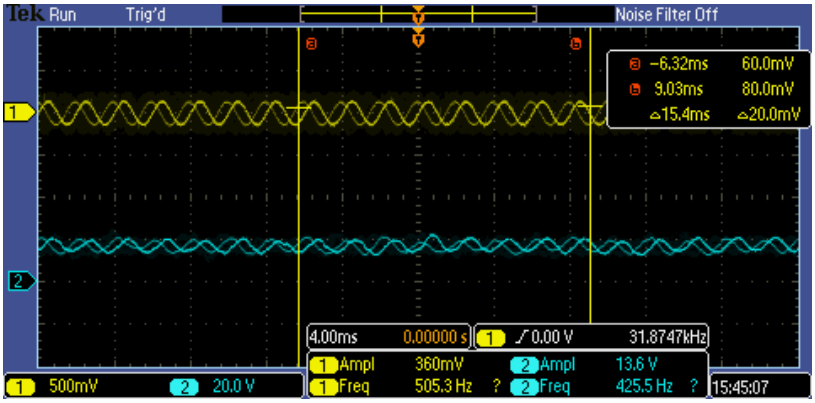



Figure 2. Input and Output at 500 Hz

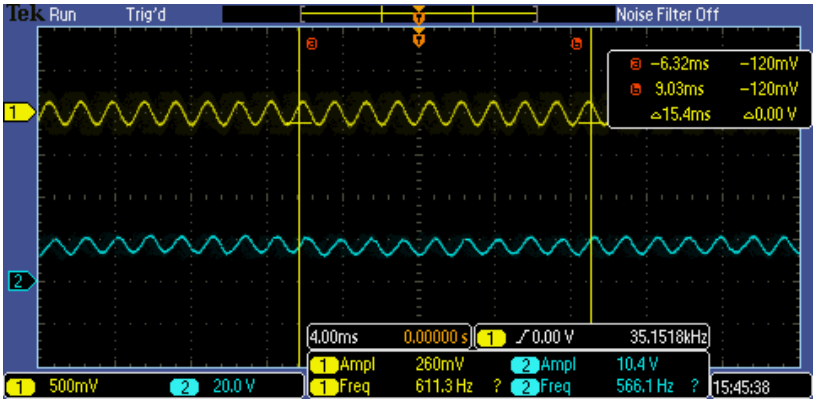



Figure 3. Input and Output at 600 Hz

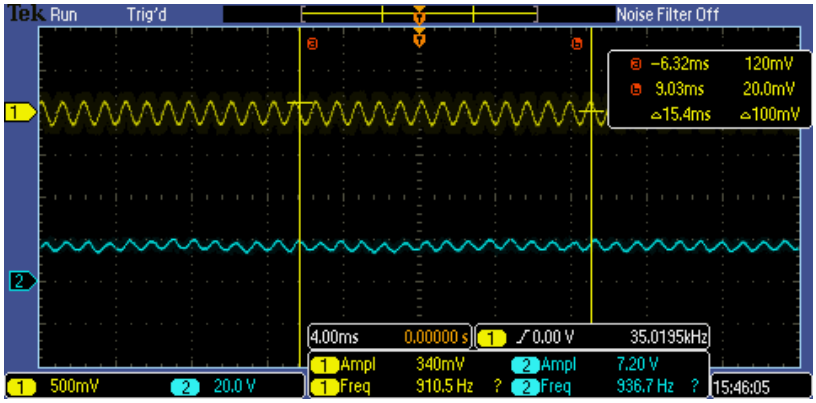



Figure 4. Input and Output at 900 Hz

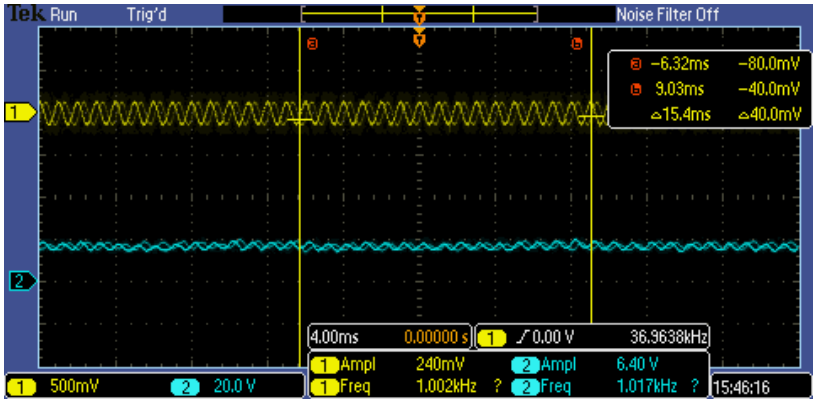



Figure 5. Input and Output at 1 kHz

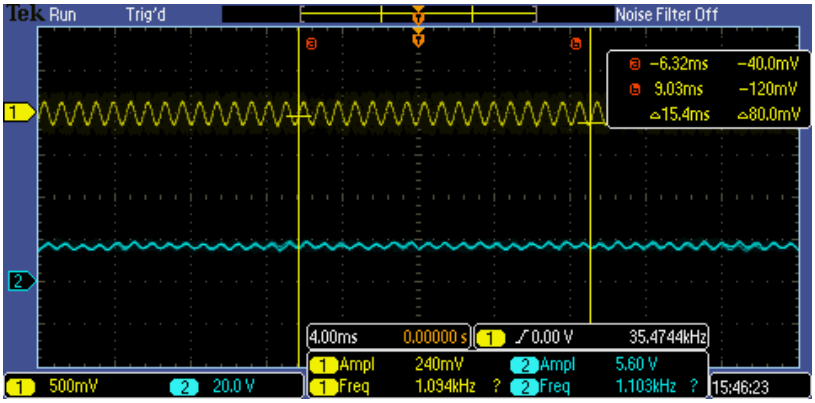


Figure 6. Input and Output at 1.1 kHz

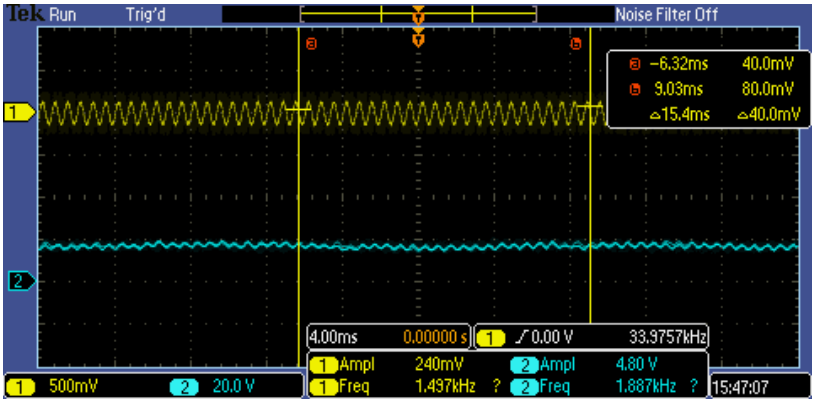


Figure 7. Input and Output at 1.5 kHz

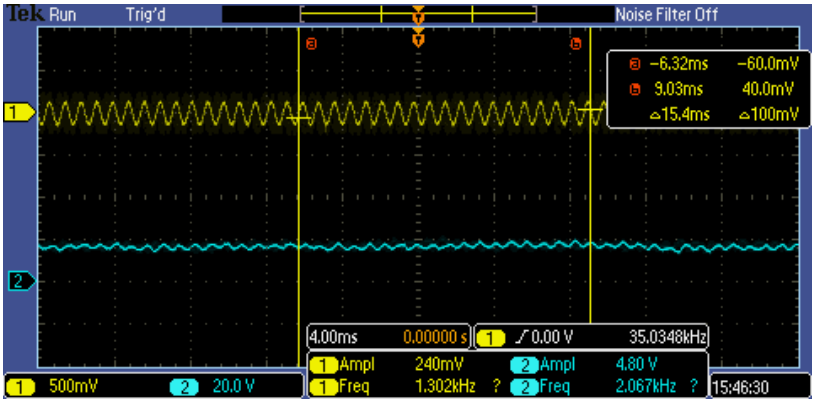


Figure 8. Input and Output at 1.9 kHz

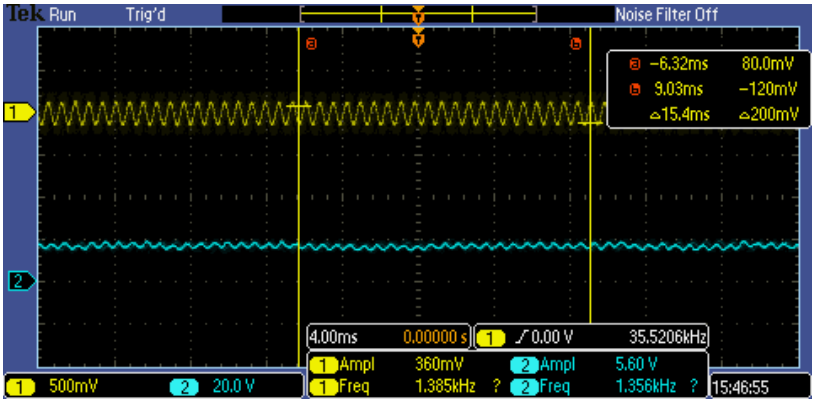



Figure 9. Input and Output at 2 kHz

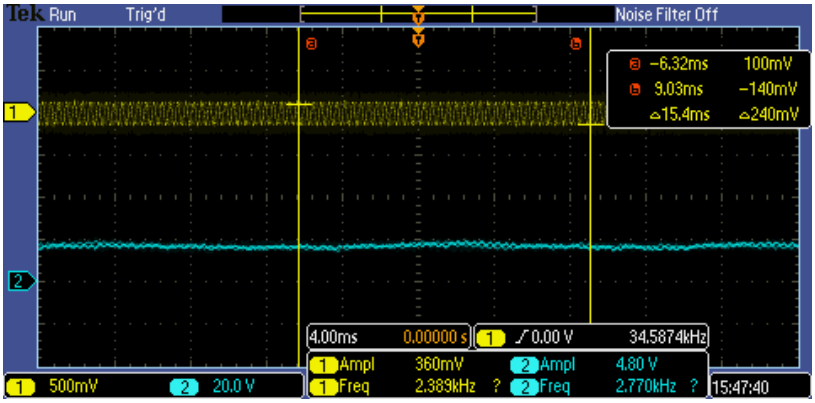



Figure 10. Input and Output at 3 kHz

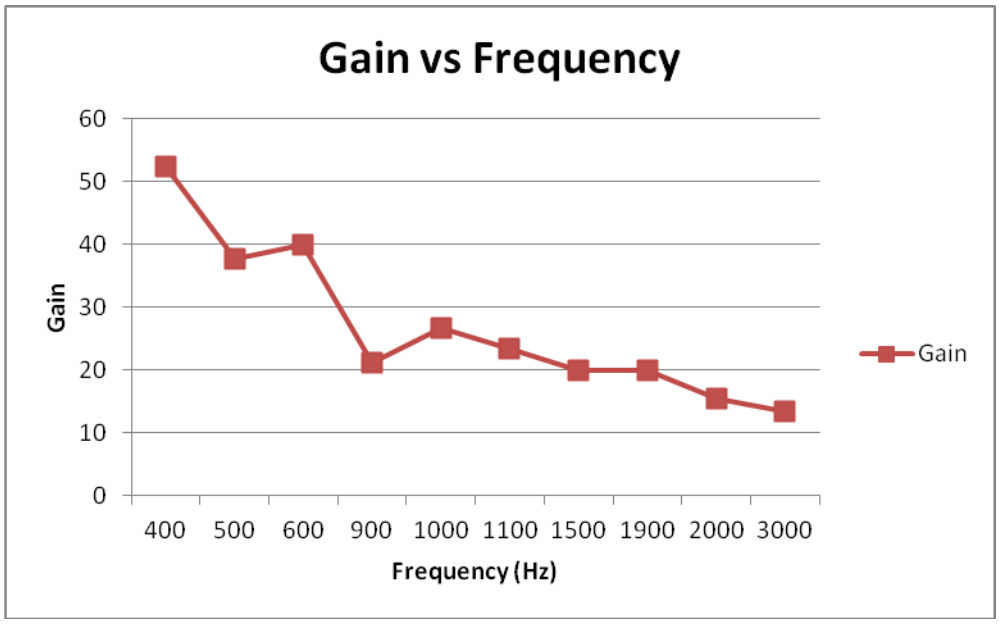



Figure 11. Gain vs. Frequency

Procedure 3:

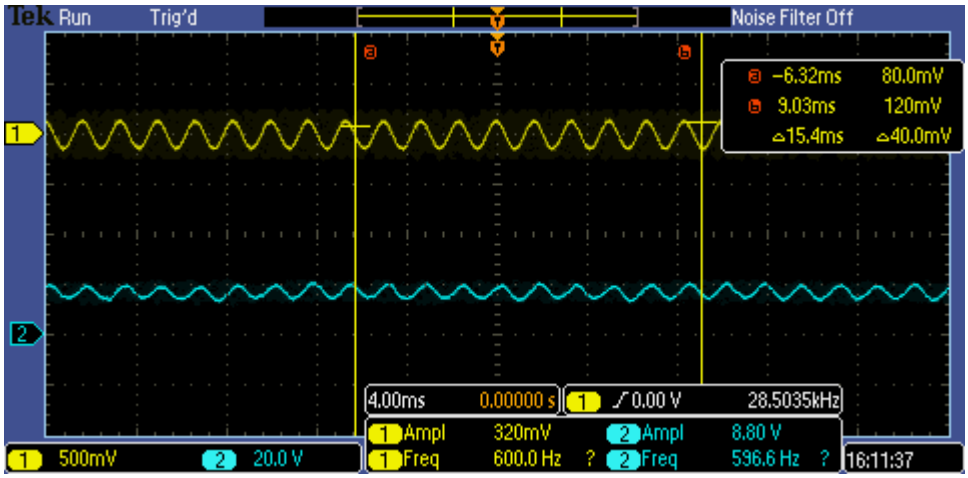


Figure 12. Input and Output at 600 Hz (.1 Fs)

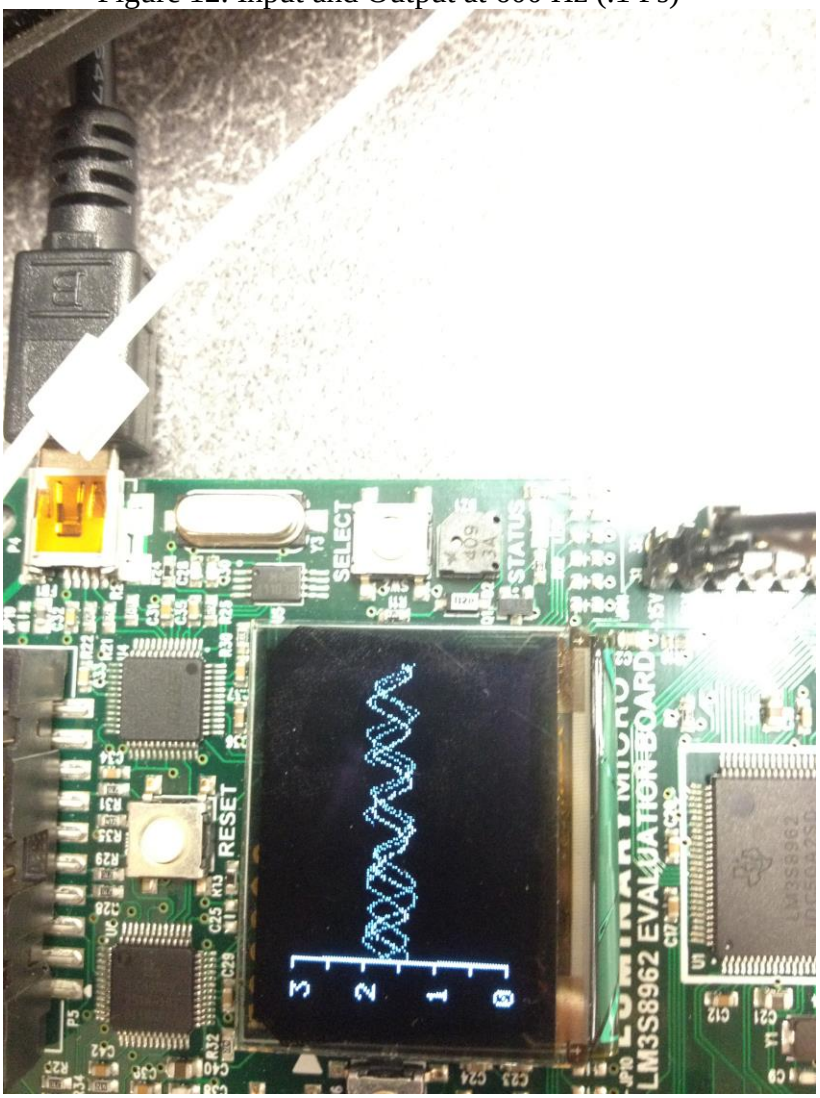


Figure 13. Scope at 600 Hz

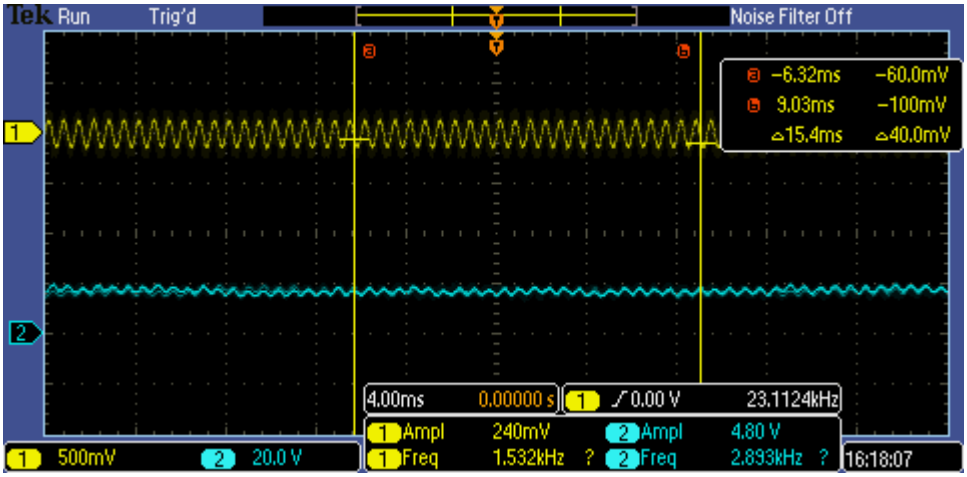


Figure 14. Input and Output at 1.5 kHz (.25 Fs)

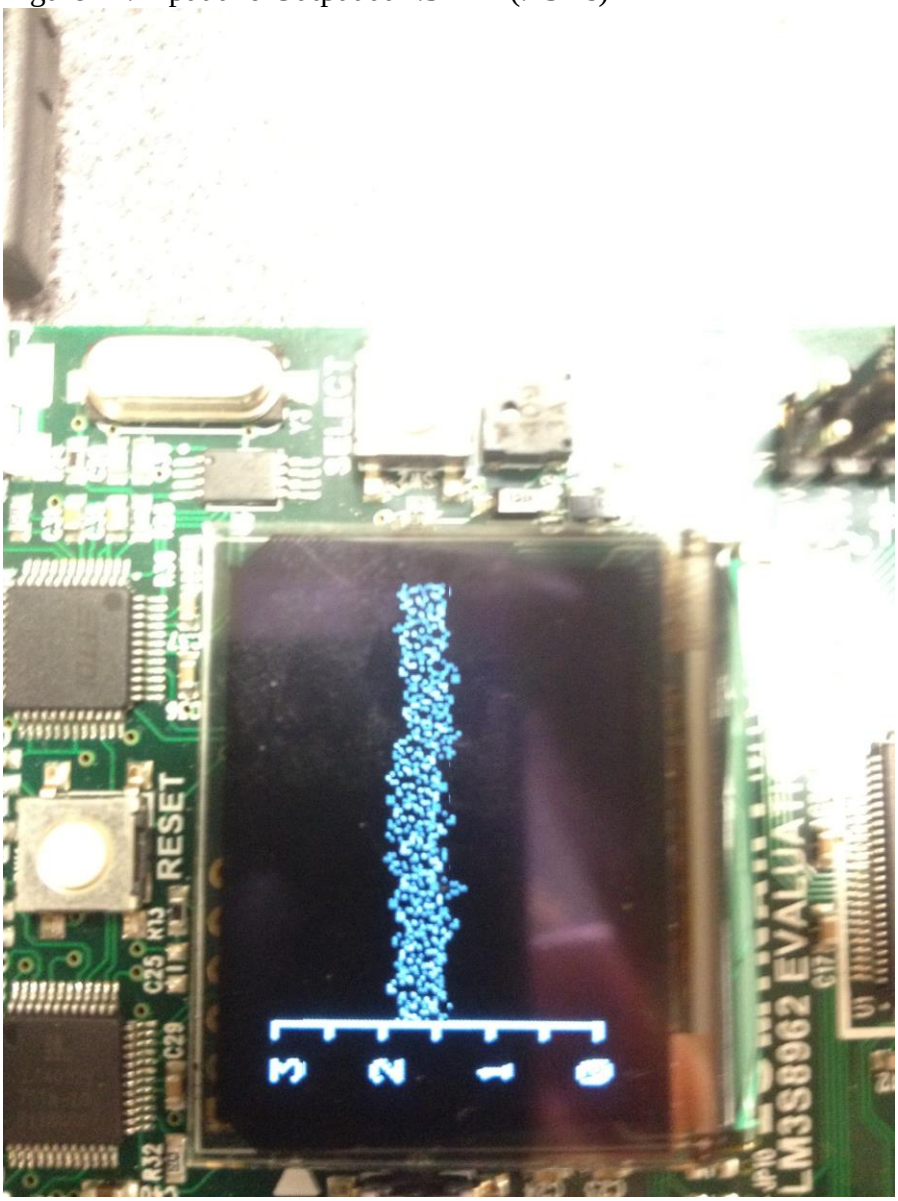


Figure 15. Scope at 1.5 kHz

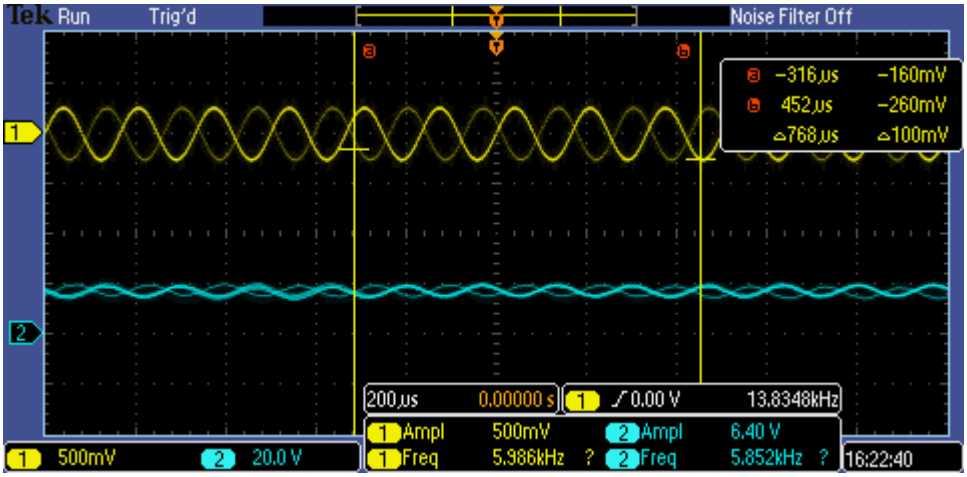


Figure 16. Input and Output at 6 kHz (Fs)

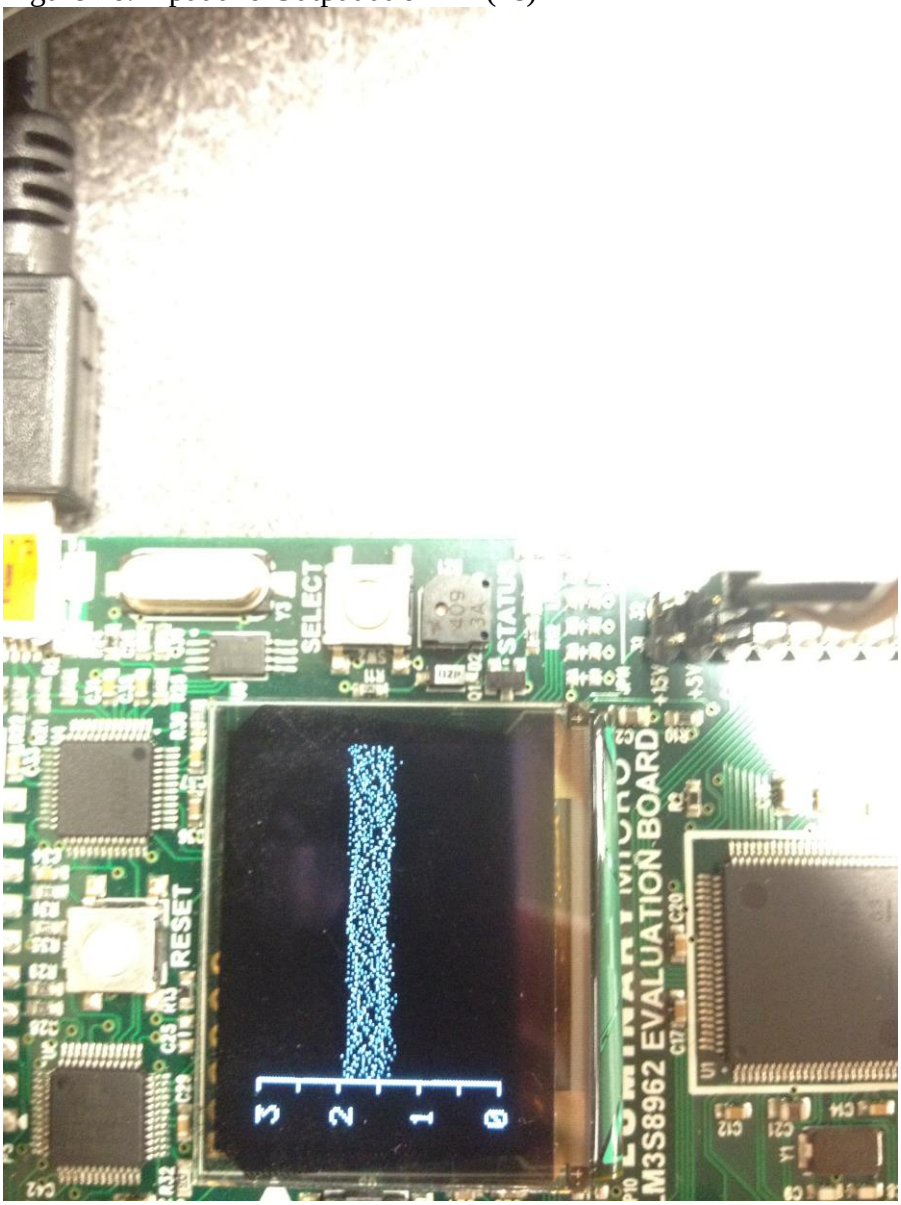


Figure 17. Scope at 6 kHz

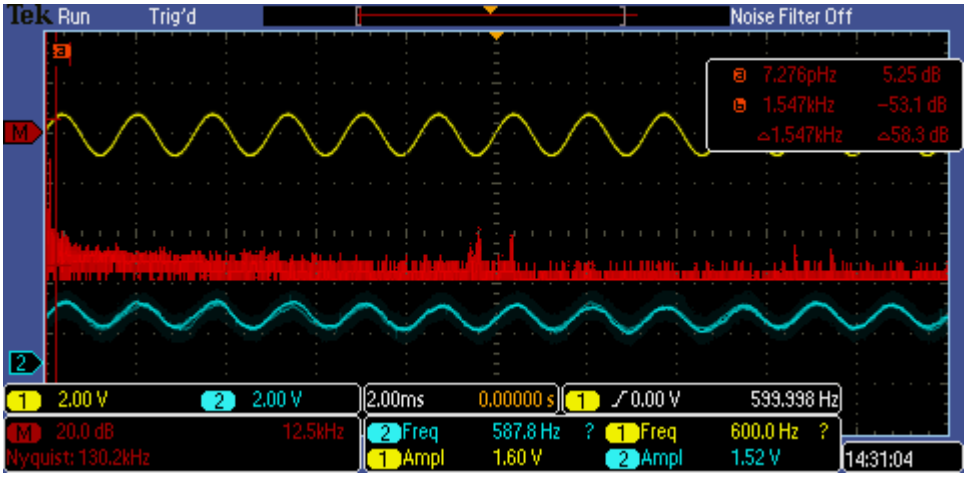


Figure 18. Input, Output, and FFT at 600 Hz (.1 Fs)

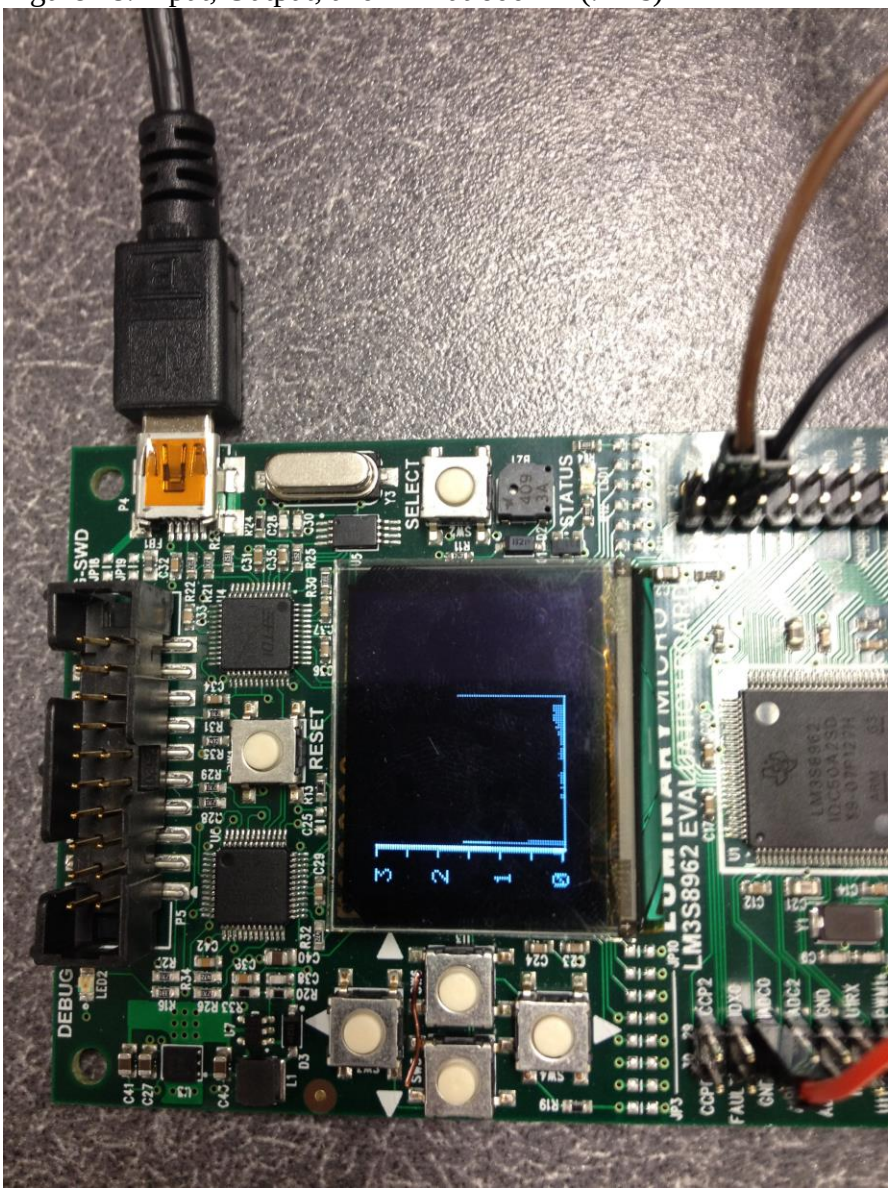


Figure 19. FFT output at 600 Hz

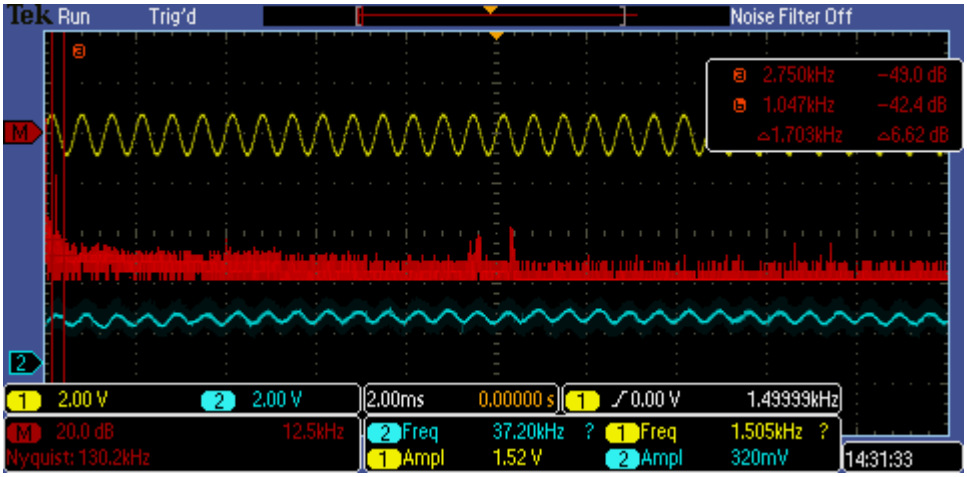


Figure 20. Input, Output, and FFT at 1.5 kHz (.25 Fs)

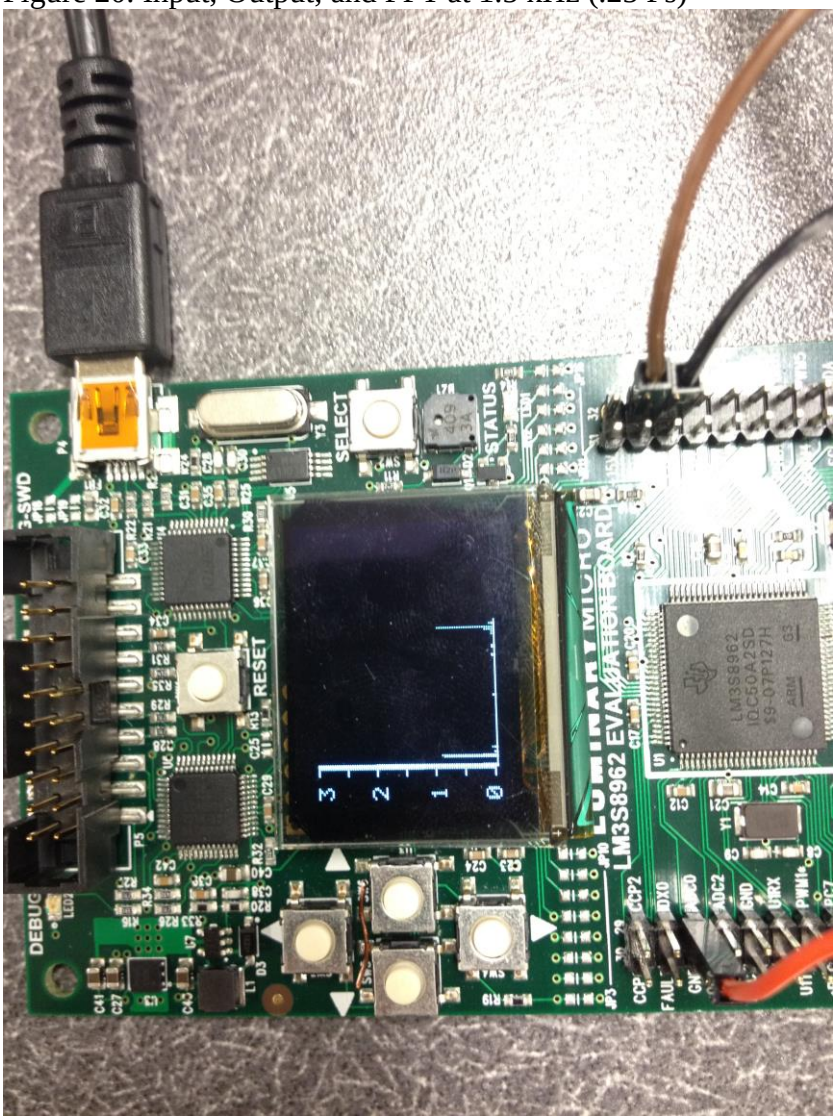


Figure 21. FFT output at 1.5 kHz

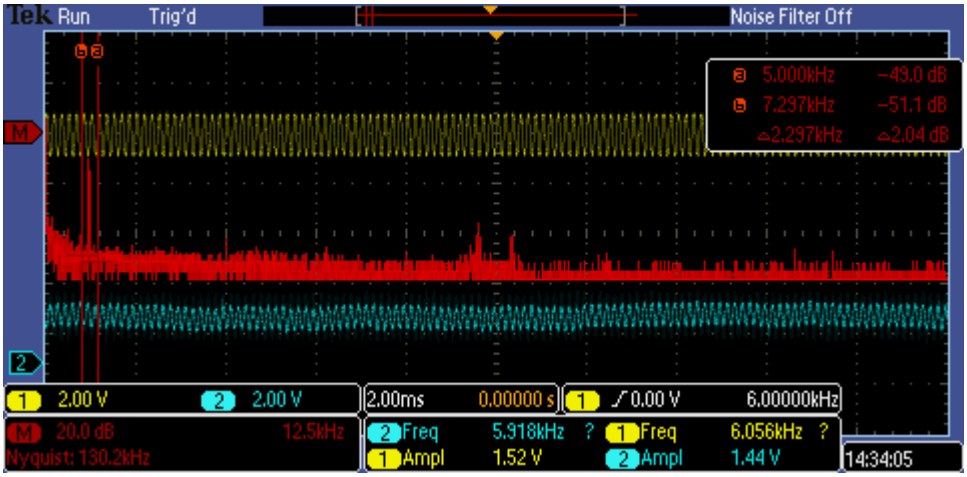


Figure 22. Input, Output, and FFT at 6 kHz (Fs)

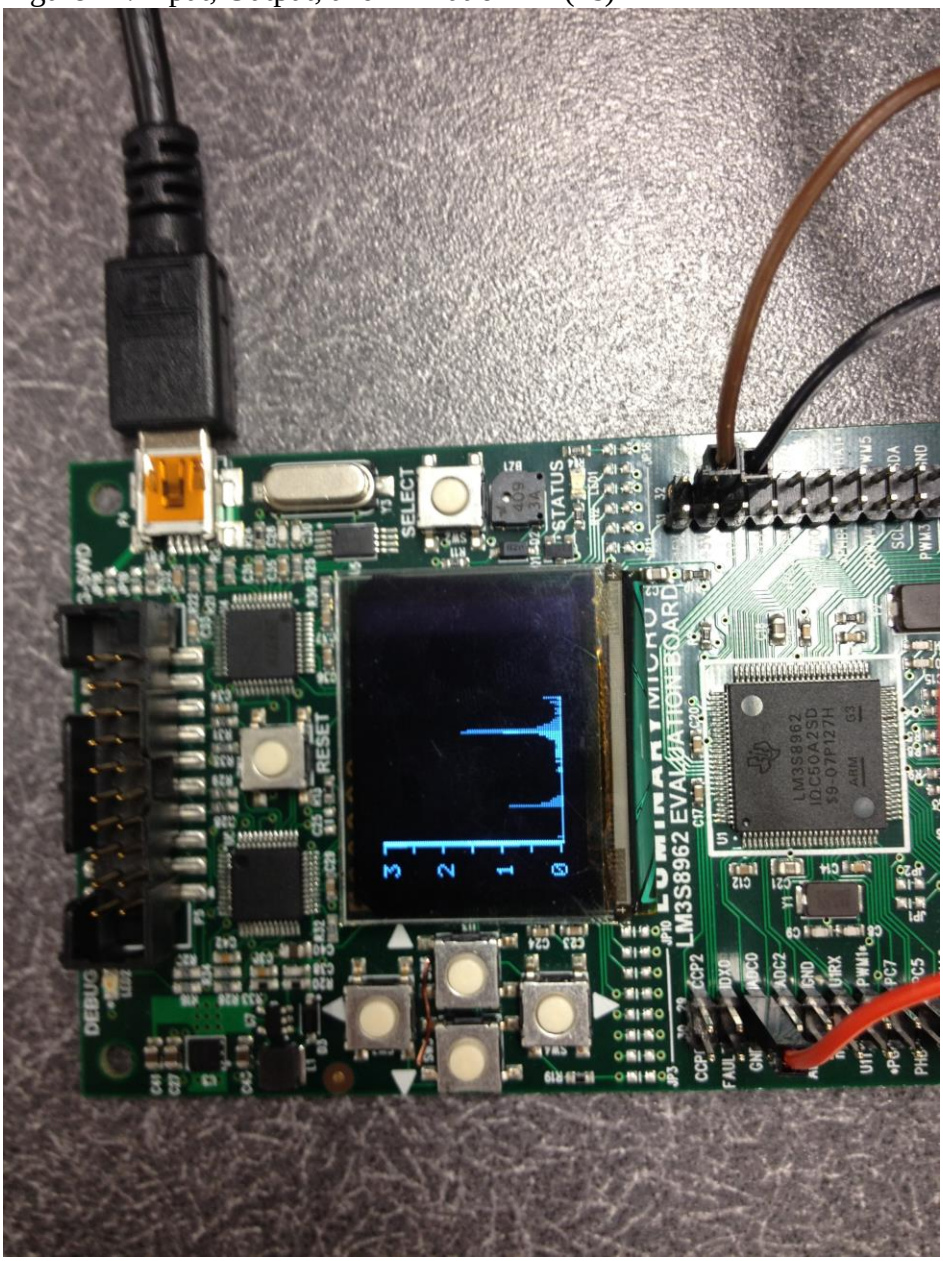
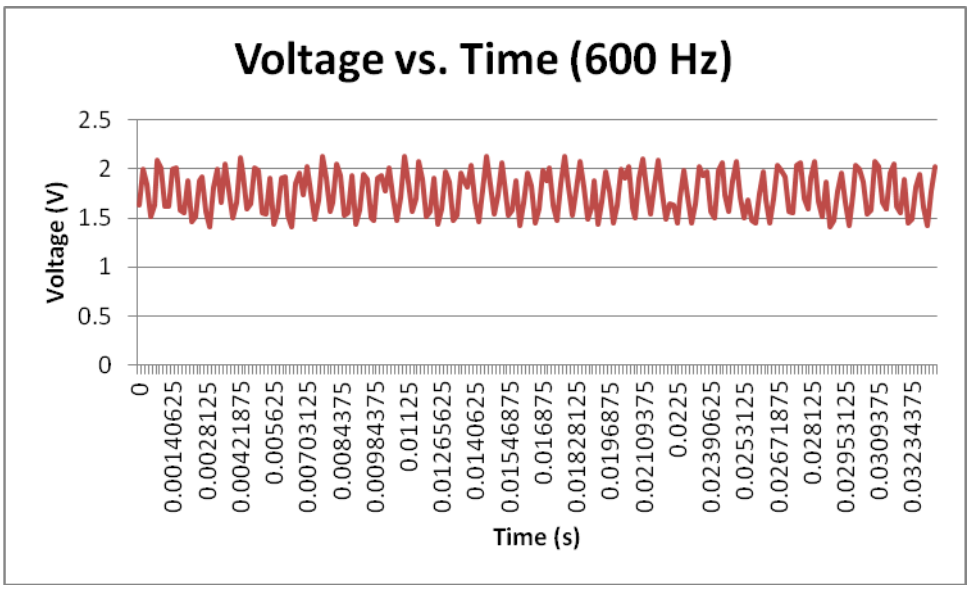
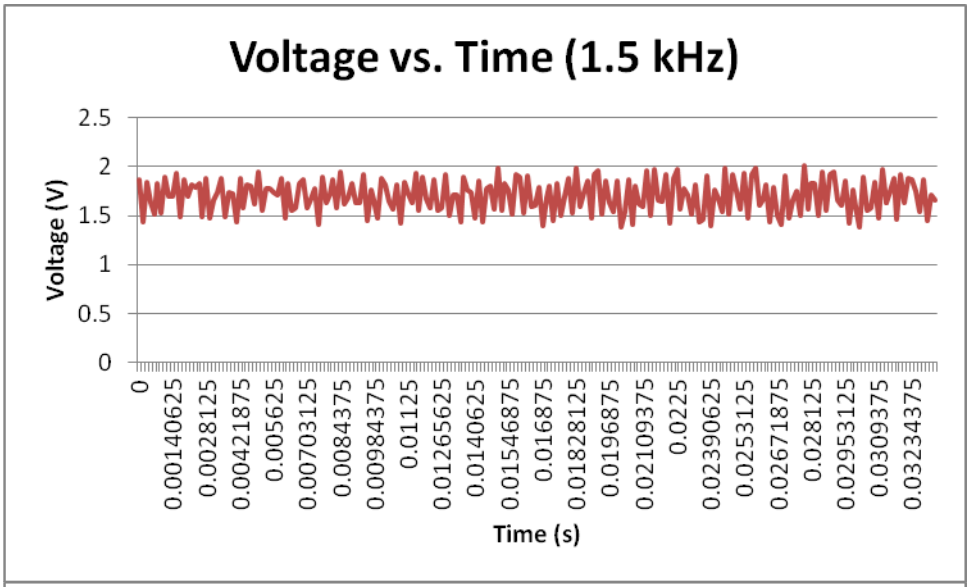
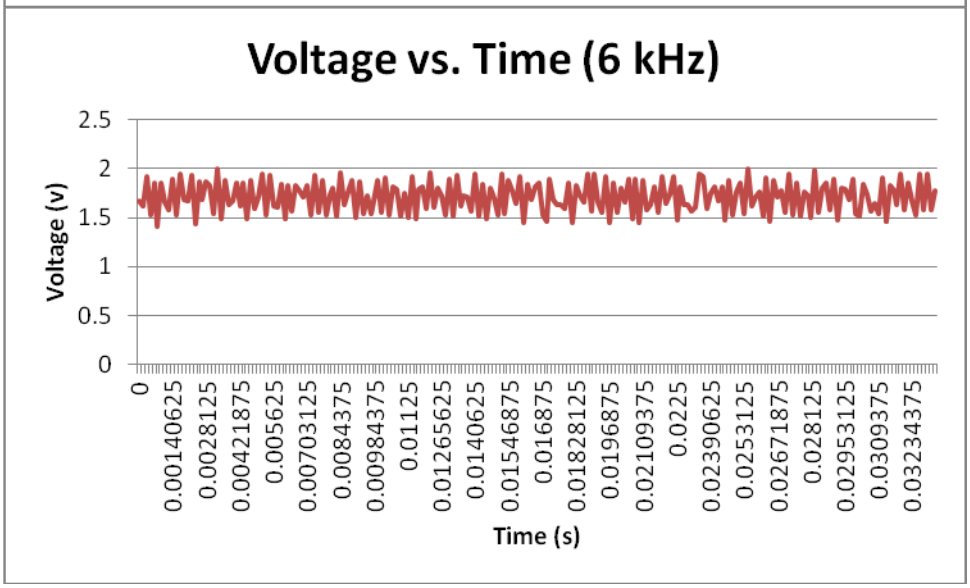
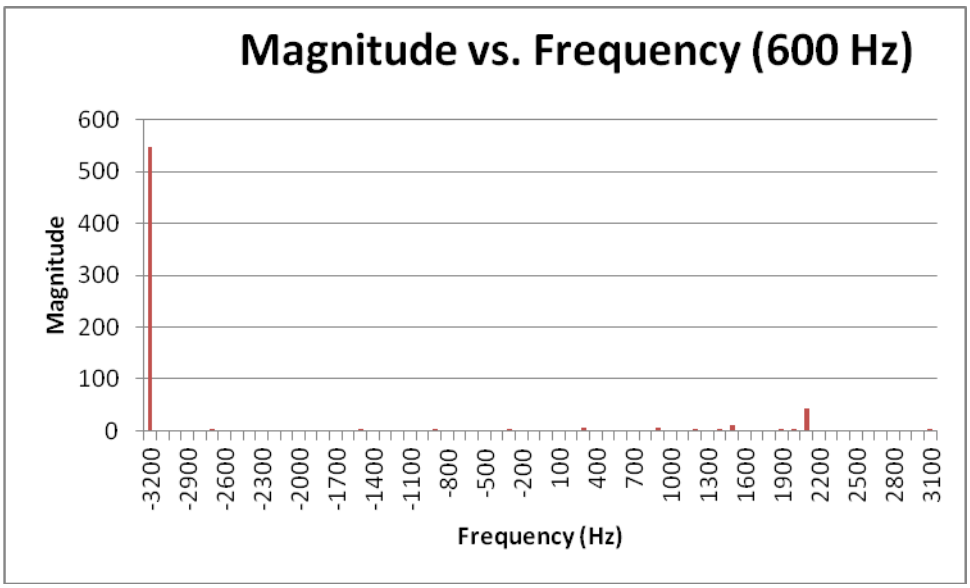


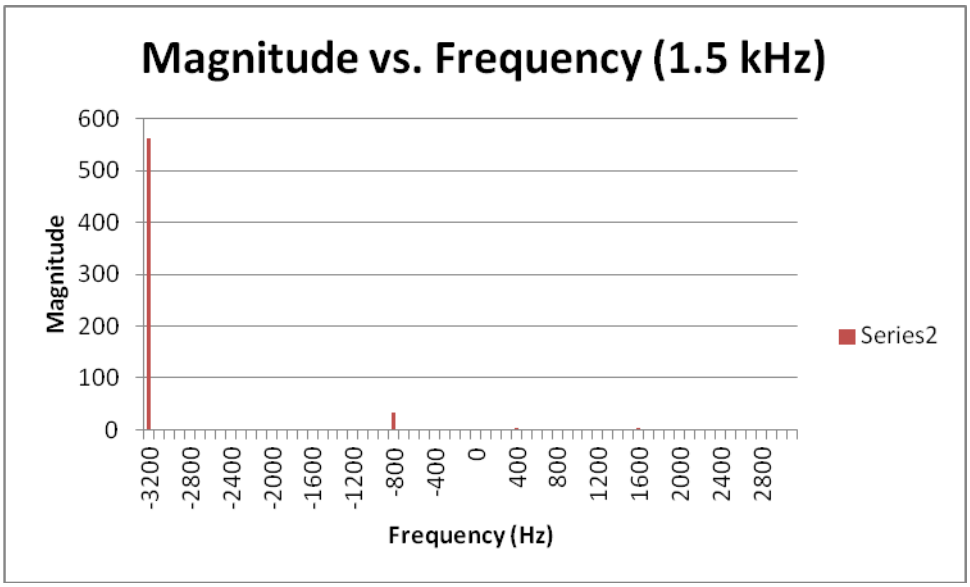
Figure 23. FFT output at 6 kHz

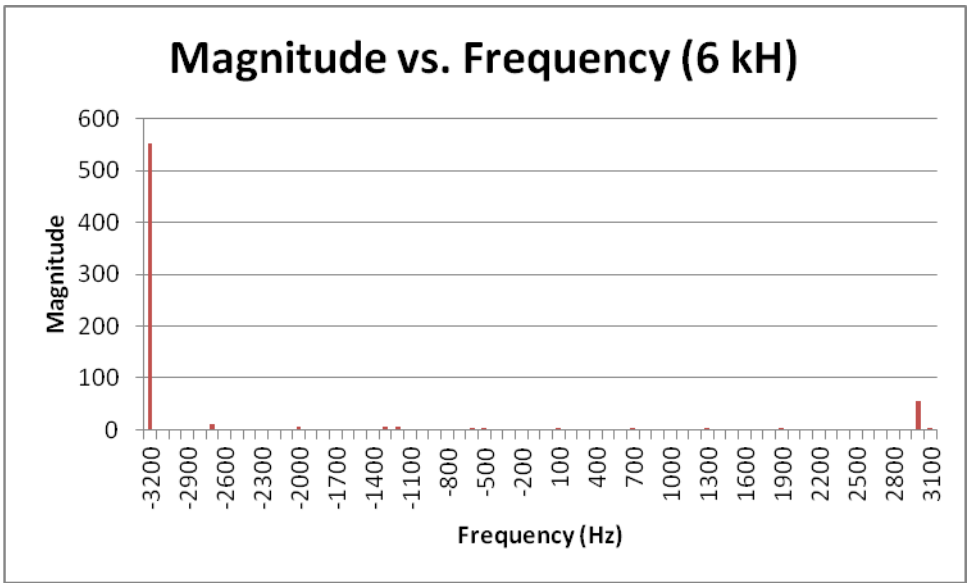

Voltage vs. Time (600 Hz)
Voltage (V)
2.5
2
1.5
1
0.5
0
0
0.00140625
0.0028125
0.00421875
0.005625
0.00703125
0.0084375
0.00984375
0.01125
0.01265625
0.0140625
0.01546875
0.016875
0.01828125
0.019625
0.02109375
0.0225
0.02390625
0.025312
0.02671875
0.028125
0.02953125
0.030937
0.03234375
Time (s)


Voltage vs. Time (1.5 kHz)
Voltage (V)
2.5
2
1.5
1
0.5
0
0
0.00140625
0.0028125
0.00421875
0.005625
0.00703125
0.0084375
0.00984375
0.01125
0.01265625
0.0140625
0.01546875
0.016875
0.01828125
0.019625
0.02109375
0.0225
0.02390625
0.025312
0.02671875
0.028125
0.02953125
0.030937
0.03234375
Time (s)


Voltage vs. Time (6 kHz)
Voltage (v)
2.5
2
1.5
1
0.5
0
0
0.00140625
0.0028125
0.00421875
0.005625
0.00703125
0.0084375
0.00984375
0.01125
0.01265625
0.0140625
0.01546875
0.016875
0.01828125
0.019625
0.02109375
0.0225
0.02390625
0.025312
0.02671875
0.028125
0.02953125
0.030937
0.03234375
Time (s)


Magnitude vs. Frequency (600 Hz)
Magnitude
600
500
400
300
200
100
0
-3200
-2900
-2600
-2300
-2000
-1700
-1400
-1100
-800
-500
-200
100
400
700
1000
1300
1600
1900
2200
2500
2800
3100
Frequency (Hz)


Magnitude vs. Frequency (1.5 kHz)
Magnitude
600
500
400
300
200
100
0
-3200
-2800
-2400
-2000
-1600
-1200
-800
-400
0
400
800
1200
1600
2000
2400
2800
Frequency (Hz)
Series2


Magnitude vs. Frequency (6 kH)
Magnitude
600
500
400
300
200
100
0
-3200
-2900
-2600
-2300
-2000
-1700
-1400
-1100
-800
-500
-200
100
400
700
1000
1300
1600
1900
2200
2500
2800
3100
Frequency (Hz)

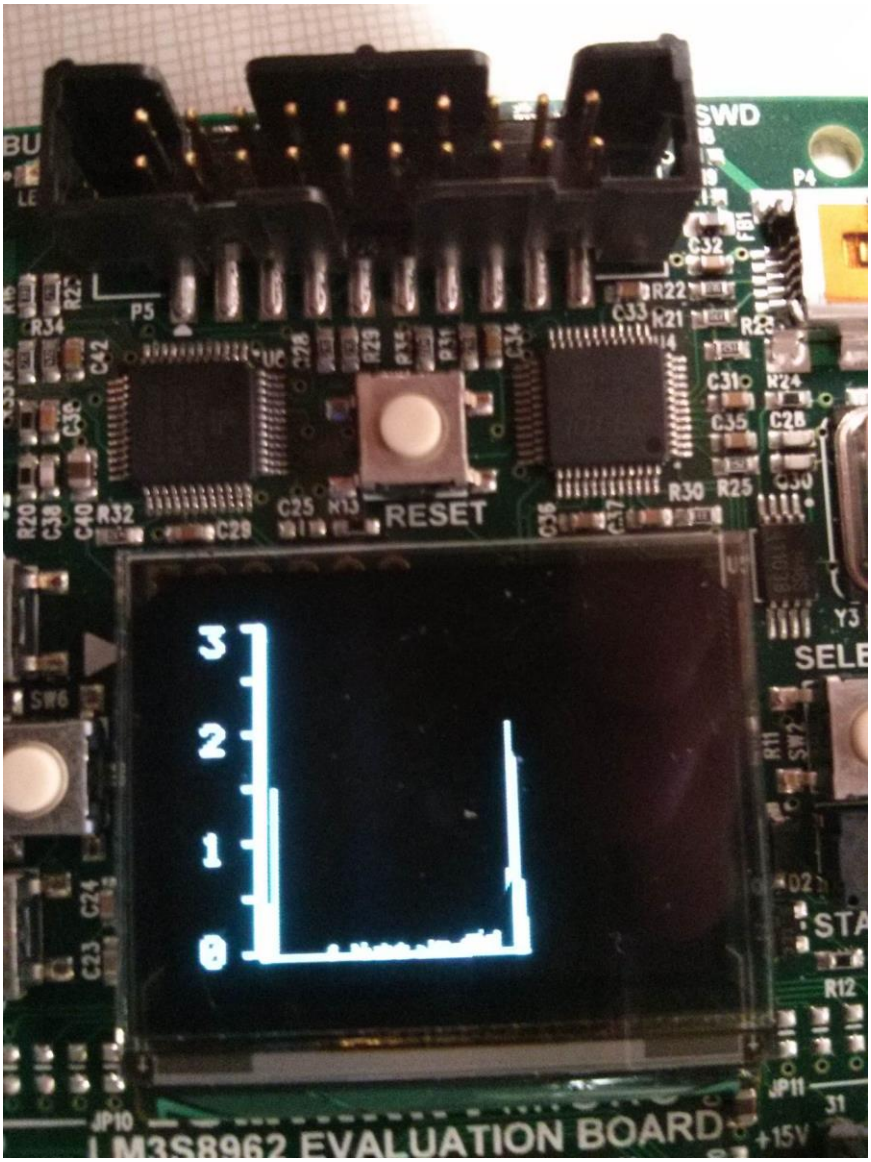


Figure 24. FFT output of filtered data (clean)

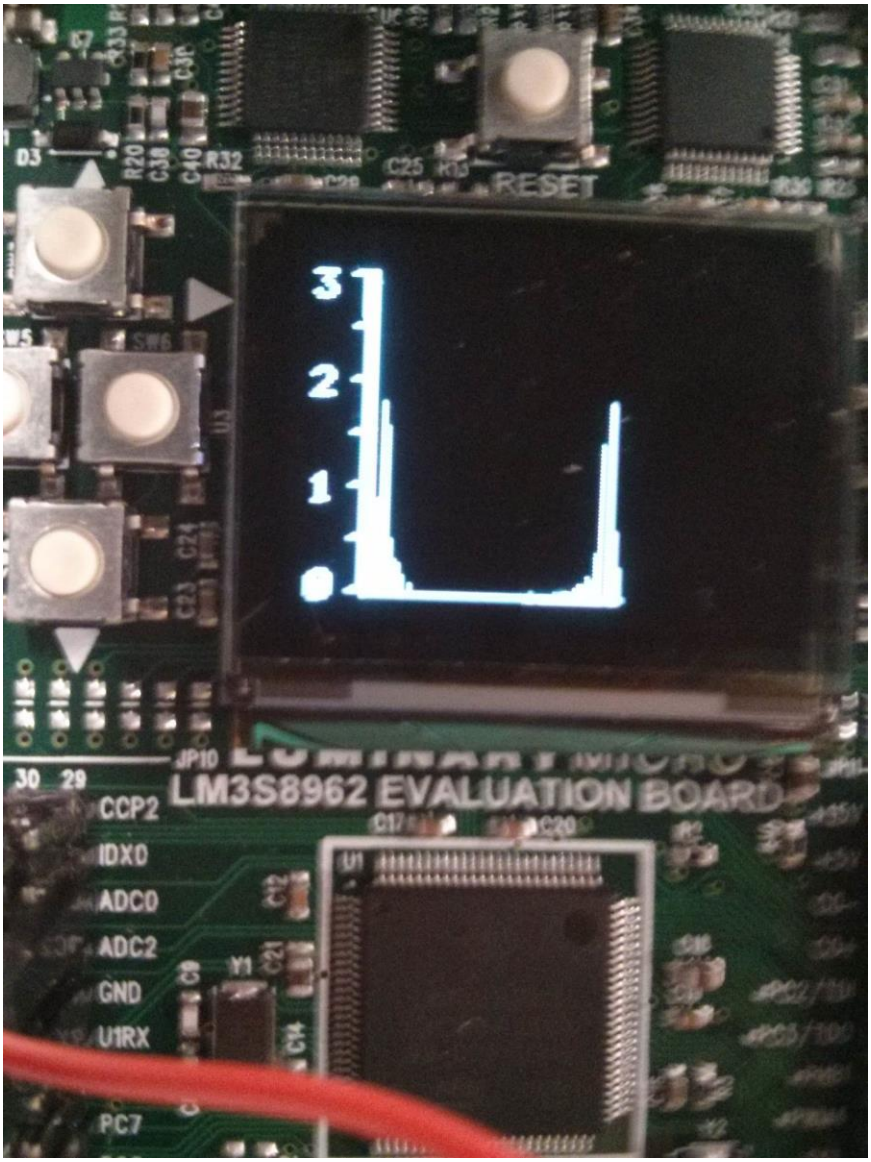


Figure 25. FFT output of unfiltered data (noisy).

**ANALYSIS & DISCUSSION**

1. The high-pass filter cutoff is 1/(2 * pi * .22uF * 11kOhm) = 66 Hz.
   The low-pass filter cutoff is 1/(2 * pi * 220pF * 100kOhm) = 7.2 kHz. It was difficult to tell from our measurements the impact of the filters.
2. We kept increasing the sampling rate until we lost too much data. The main factor affecting bandwidth how fast the consumer threads can retrieve and use the data.
3. Since the Fourier transform of a rectangular pulse is a sinc, the transform of a square wave would be the convolution of infinite sinc pulses.
4. This is thermal noise and noise from the power supply. There will always exist some type of noise in the system due to these factors.
5. The ADC samples are being triggered by a timer, which is controlled by the clock, which is controlled by the crystal oscillator, which has almost no jitter.
6. Our FIR filter had a length of 51. The ADC inputs had a maximum size of 10 bits, and the FIR coefficients also had a maximum size of 10 bits. Therefore, we see that when performing the convolution, we multiply these two to get 51 20-bit values. Then, we add these values. The maximum size required to store this value is ceil($\log_2 51$) + 20 = 26 bits. We store the value in a 32 bit integer, so we never get overflow.
7. Because of the symmetry of the filter coefficients, we can approximately halve the number of multiplications by adding the two "mirrored" x terms before multiplying, e.g. y[i] = (h[0] * (x[i] + x[i-50]) + ... + h[25] * (x[i-25] + x[i-26]))/256