ENGG*3390- Signal Processing

Lab 2

Bilal Ayyache- 0988616

Palak Sood - 0986802

Kathlene Titus – 0954584

October 25th, 2018

Introduction:

To eliminate specific selected frequencies or artifacts in a signal, filtering should be performed. This could be achieved using a device or a process such as the TMS320C5505 eZdsp USB stick. The purpose of this lab is to program the Texas Instrument's TMS320C5505 eZdsp USB stick to perform some simple filtering using the impulse response. This process suppresses part of the signal input to reduce the background noise. Filtering was performed on a signal through three different discrete time systems that performs a type of finite impulse response (FIR) filter whose filter is an impulse response of a finite duration and requires no feedback. The three different systems used are a High Pass Filter (HPF), Low Pass Filter (LPF), and Band Pass Filter (BPF). High Pass Filters allow high frequency to pass and filter out lower frequencies. This is done by filtering out changes in the signal that occur over a prolonged period. Low Pass Filters allow lower frequencies to pass and filters out higher frequencies, which are portions of signal that change rapidly. Band Pass Filters allow is mixture of both high and low pass filters.

Materials:

During the lab, the following tools and equipment we used:

1. Function generator and oscilloscope
2. Input audio source, e.g. PC workstation, MP3 player, headphones, etc.
3. TI TMS320C5505 eZdsp USB Stick with cable, audio patch cords
1. Lab 2 project files as posted on CourseLink
4. TI Code Composer Studio (CCS) SDK

Procedure:

For each system, the following was done:

First, the coefficients of the system were multiplied by 2^{15} due to the 16-bit DSP architecture. The resulting integer values was then compared to the ones in MATLAB. Second, the two outputs were connected to two oscilloscope channels using the impulse signal as the trigger. The "impulse signal" was the output on the audio left channel and the "impulse response" was the output on the audio right channel. The waveform on the oscilloscope was compared to the impulse calculator on MATLAB. Then, the impulse response on the oscilloscope was compared to the impulse response on MATLAB. All results and observations were noted. Thirdly, the left channel audio input was combined with the impulse response to yield a result on the left audio channel output. The original right channel input is passed straight through to the right audio channel output. A function generator was used to input a 200 Hz square wave to left channel audio input, and the oscilloscope was used to measure the resulting step response from the left audio channel output. Lastly, the function generator was disconnected, and the music source was connected to the audio input of the Breakout Board. Any difference between the original and processed audio was noted.

Results/ Discussion:

To understand the fixed-point 16-bit DSP architecture, the floating-point values obtained for the impulse response were multiplied by 2^{15} using MATLAB for each system. These numbers were compared to the values in the FIR_filters.h file. As seen in Table 1-3, both the values were almost exactly equal. This is justified because the floating-point values need to be multiplied by 2^{15} to be used in a fixed-point system. Figures 1-3 show the MATLAB code to get the floating-point values.

MATLAB Floating Point Values	FIR_filters.h values
6.9796	7
51.1010	51
182.2681	182

Table 1: A sample of the first 3 values for system 1

MATLAB Floating Point Values	FIR_filters.h values
5.0084E03	5008
-1.6771E04	-16771
1.6017E04	16017

Table 2: A sample of the first 3 values for system 2

MATLAB Floating Point Values	FIR_filters.h values
854.5239	855
3702.5025	3702
5269.5427	5269

Table 3: A sample of the first 3 values for system 3

Figures 11-13 show the impulse as the input and the impulse response as the output for systems 1-3 respectively. All three systems are graphed in MATLAB as well. Although in MATLAB the impulse response is modelled as a discrete function (Figure 5), it can still be observed that the MATLAB plots, Figures 6-8, are very similar to the oscilloscope signals for all systems.

The impulse response was convolved with a 200 Hz square wave and then an audio input. As shown in Figure 14, the yellow signal is the input of 200 Hz square wave and the blue signal is the output of the convolution of the impulse response with the square wave. To determine which type of filter each system was, an audio input was used. A Low Pass Filter (LPF) is a filter that passes signals with a frequency lower than a selected cutoff frequency and attenuates signals with frequencies higher than the cutoff frequency. This can be noted in system one, which resulted in low frequencies being passed through the audio output. A Band Pass Filter (BPF) passes frequencies within a certain range and rejects frequencies outside that range. This was recognized in system 2 where the audio seemed to be more muffled. A High Pass Filter (HPF) is a filter that passes signals with a frequency higher than a certain cutoff frequency and attenuates signals with frequencies lower than cutoff frequency. This was prominent for system 3 since only higher frequency audio was heard through the audio output.

Conclusion:

For the implementation on a fixed-point, 16-bit DSP architecture, the floating-point values were required to be multiplied by 2^{15}. It was observed that the MATLAB plots was similar to the signals from the oscilloscope for each system. To determine which type of filter each system was, an audio input was used. A Low Pass Filter was noted in system one, which resulted in low frequencies being passed through the audio output. A Band Pass Filter (BPF) was recognized in system 2 where the audio seemed to be more muffled. A High Pass Filter (HPF) was prominent for system 3 since only higher frequency audio was heard through the audio output.

References:

1. ENGG*3390 Signal Processing Lab 1 Manual (F18)
2. ENGG*3390 Signal Processing Lab-guide (F18)

Appendix:

Editor - H:\engg3390\sys1.m

sys1.m | sys2.m | sys3.m

```
%system 1
y = [1 -3.316808 4.174246 -2.357426 0.503375];
x = [0.000213 0.000853 0.001279 0.000853 0.000213];
impResp = impz(x,y,64);
new = (2^15)* impResp;
plot(impResp);
```

Figure 1: System 1 code for implementation on a fixed point 16-bit DSP architecture

Editor - H:\engg3390\sys2.m

sys1.m | sys2.m | sys3.m

```
%system 2
y = [1 -0.651472 0.620472 -0.14737 0.026169];
x = [0.152843 -0.611373 0.917059 -0.611373 0.152843];
impResp = impz(x,y,64);
new = (2^15)* impResp;
plot(impResp);

```

Figure 2: System 2 code for implementation on a fixed point 16-bit DSP architecture

Editor - H:\engg3390\sys3.m

sys1.m | sys2.m | sys3.m

```
%system 3
y = [1 -4.332825 8.606785 -10.513306 8.785253 -5.116547 1.997132 -0.477997 0.055764];
x = [0.026078 -0.104311 0.156466 -0.104311 0.026078];
impResp = impz(x,y,64);
new = (2^15)* impResp;
plot(impResp);
```

Figure 3: System 3 code for implementation on a fixed point 16-bit DSP architecture

Editor - H:\engg3390\ImpluseMatlab.m

sys1.m | sys2.m | sys3.m | ImpluseMatlab.m

```
t = -2:2;
y = zeros(1,5);
y(1,3) = 1;
stem(t,y);
```

Figure 4: Code to get an impulse

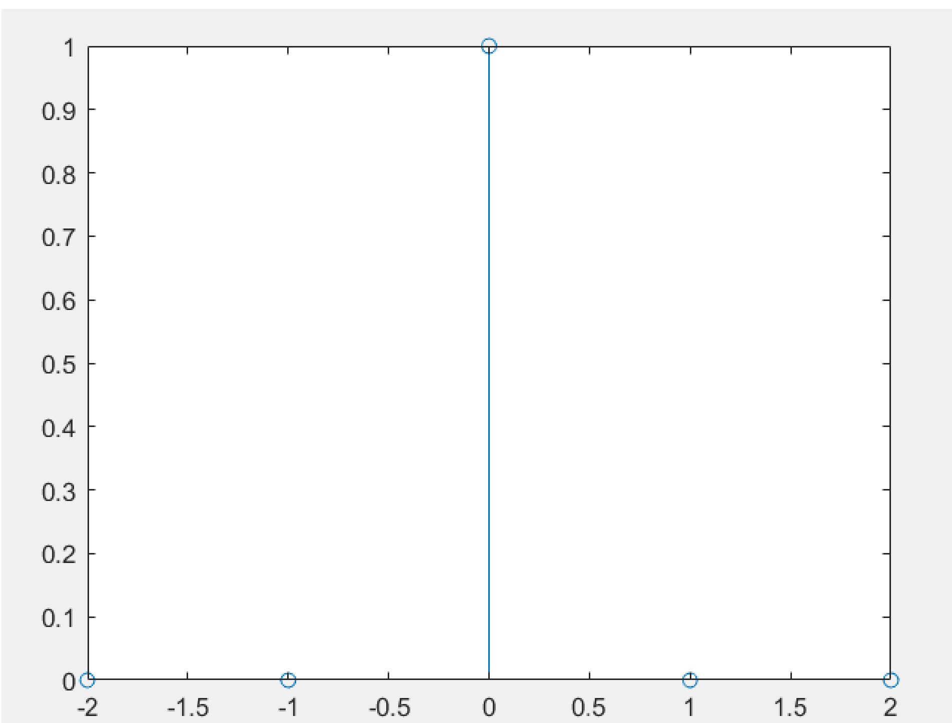

Figure 5: Impulse function

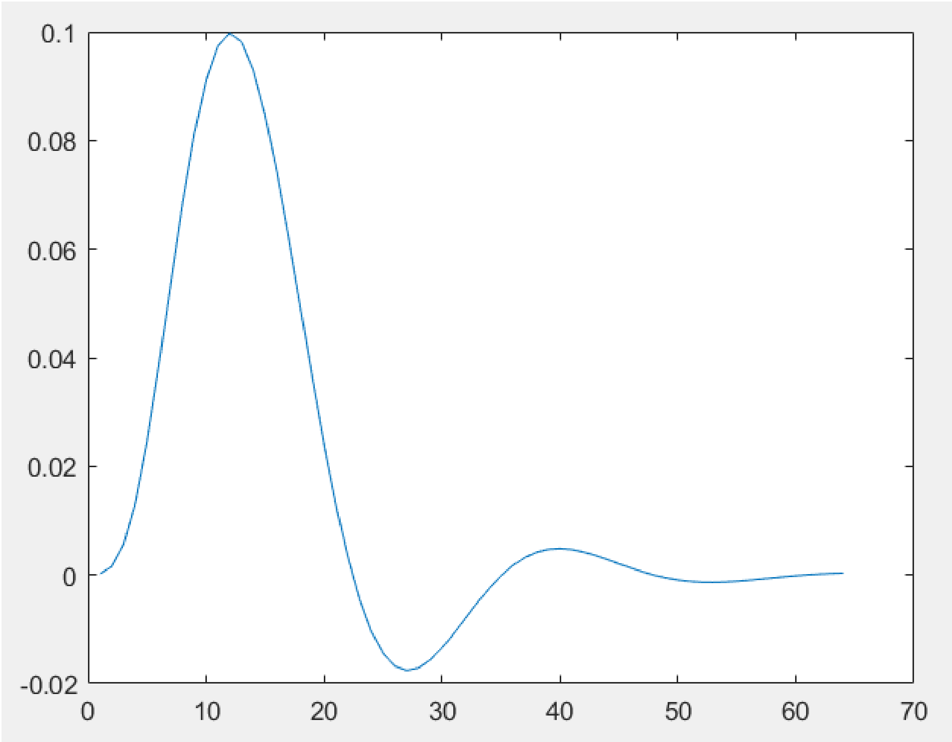

Figure 6: Plot of the impulse response for System 1

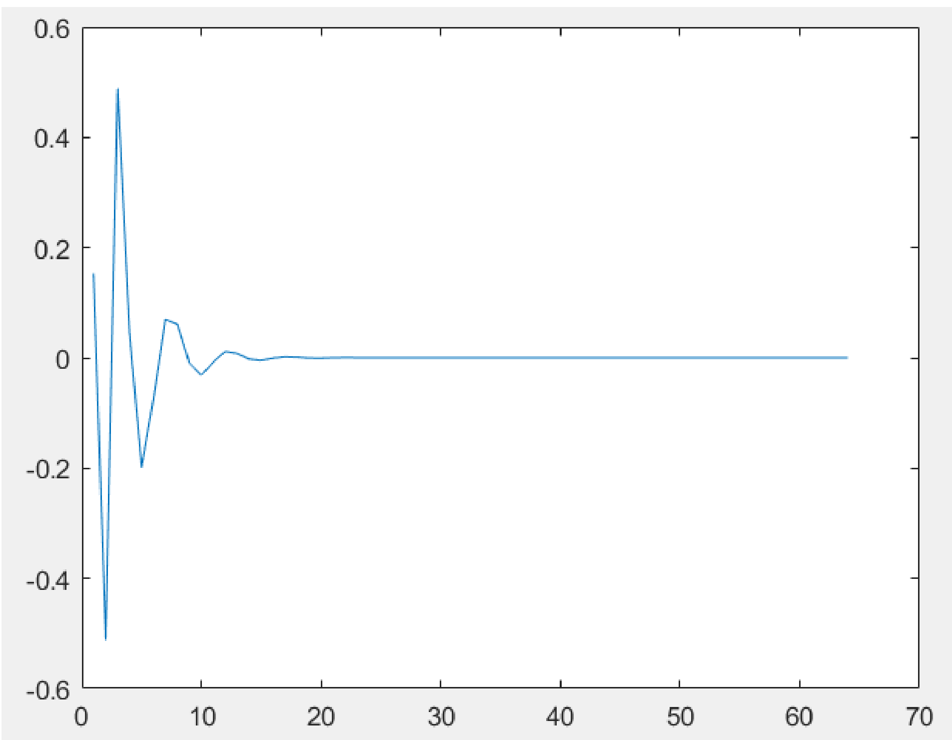

Figure 7: Plot of the impulse response for System 2

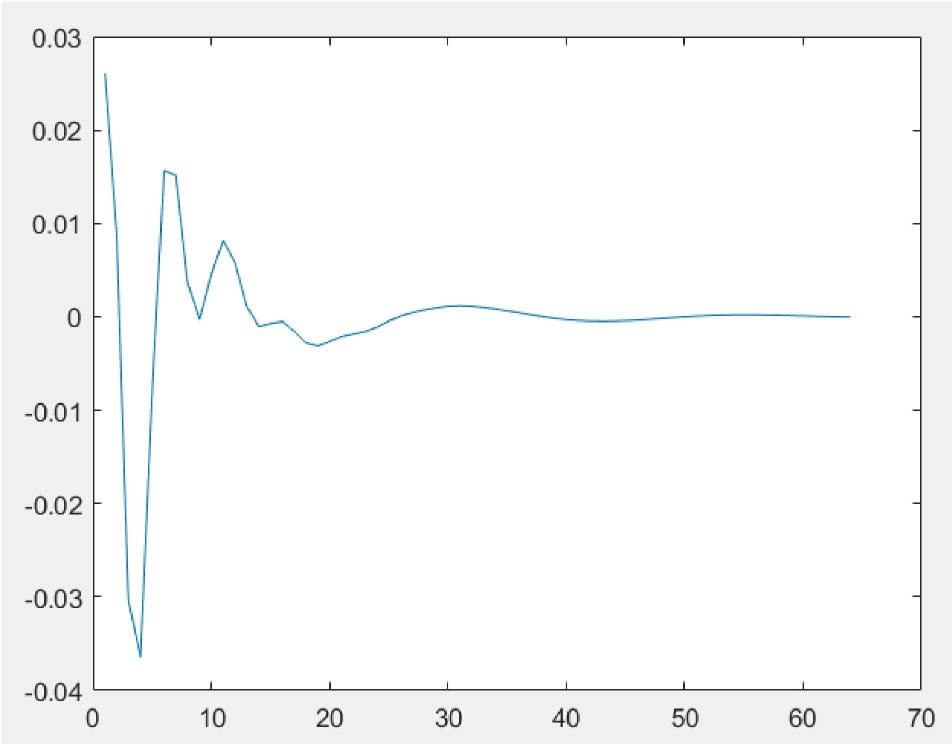

Figure 8: Plot of the impulse response for System 3

```c
#include "stdio.h"
#include "usbstk5505.h"
#include "aic3204.h"
#include "PLL.h"
#include "FIR_filters_asm.h"
#include "FIR_filters.h"

#define SAMPLES_PER_SECOND 48000
#define GAIN_IN_dB   0


Int16 left_input;
Int16 right_input;
Int16 left_output;
Int16 right_output;

unsigned long int i = 0;

/* ------------------------------------------------------------------------
 *
 *  main( )
 *
 * ------------------------------------------------------------------------
void main( void )
{

  /* Initialize BSL */
  USBSTK5505_init( );

  /* Initialize the Phase Locked Loop in EEPROM */
  pll_frequency_setup(100);

  /* Initialise hardware interface and I2C for code */
  aic3204_hardware_init();

```

Figure 9: main.c part 1

```c
    /* Initialise the AIC3204 codec */
    aic3204_init();

    printf("\n\nRunning Lab 2 code Impulse Response using main.c\n");

    /* Set sampling frequency in Hz and ADC gain in dB */
    set_sampling_frequency_and_gain(SAMPLES_PER_SECOND, GAIN_IN_dB);


    asm(" bclr XF");


    // One minute: left output is impulse response, right output is impulse
    printf("Part A: impulse response (60 s)\n");
    printf("  Left output: impulse response\n");
    printf("  Right output: impulse function\n");
    FIR_filters_asm_init();
    for ( i = 0  ; i < SAMPLES_PER_SECOND * 600L  ;i++  )
      {
       aic3204_codec_read(&left_input, &right_input);

       // Create impulse delta[0]=2^15-1 every 256 samples (Why not delta[0]=1?)
       if ((i % 256) == 0)
      left_input=32767;
       else
      left_input=0;

       left_output = FIR_filter_asm(System3, left_input);
       right_output = left_input;

       aic3204_codec_write(left_output, right_output);
      }

```

Figure 9: main.c part 2

```c
    // One minute: output is processed input
    printf("Part B: processed audio (60 s)\n");
    printf( "  Audio Stereo Line IN (L) ->[filter]-> (L) HP/Lineout\n" );
    printf( "                       (R) -----------> (R)           \n" );
    FIR_filters_asm_init();
    for ( i = 0  ; i < SAMPLES_PER_SECOND * 600L  ;i++  )
      {
       aic3204_codec_read(&left_input, &right_input);

       left_output = FIR_filter_asm(System3, left_input);
       right_output = right_input;

       aic3204_codec_write(left_output, right_output);
      }

    /* Disable I2S and put codec into reset */
    aic3204_disable();

    printf( "\n***Program has Terminated***\n" );
    SW_BREAKPOINT;
}
```

Figure 10: main.c part 3

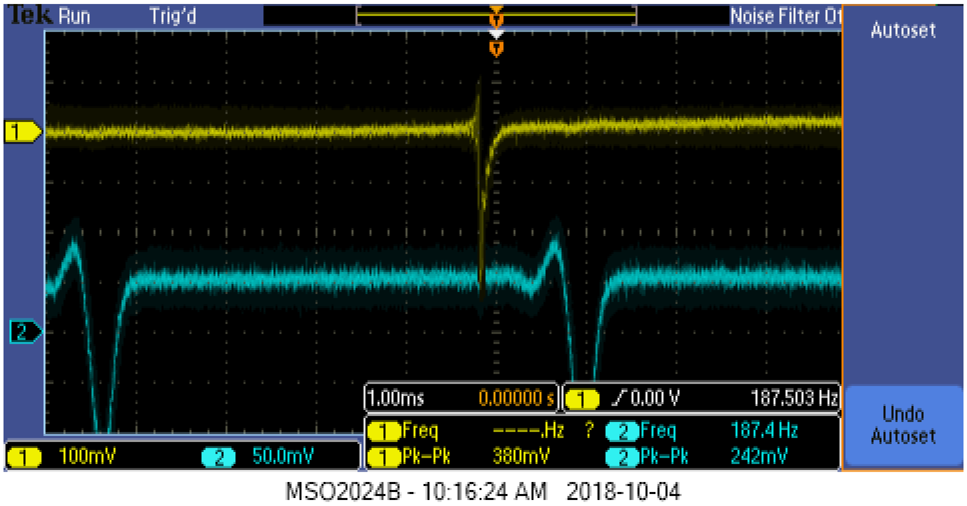

Figure 11: Part 2, System 1

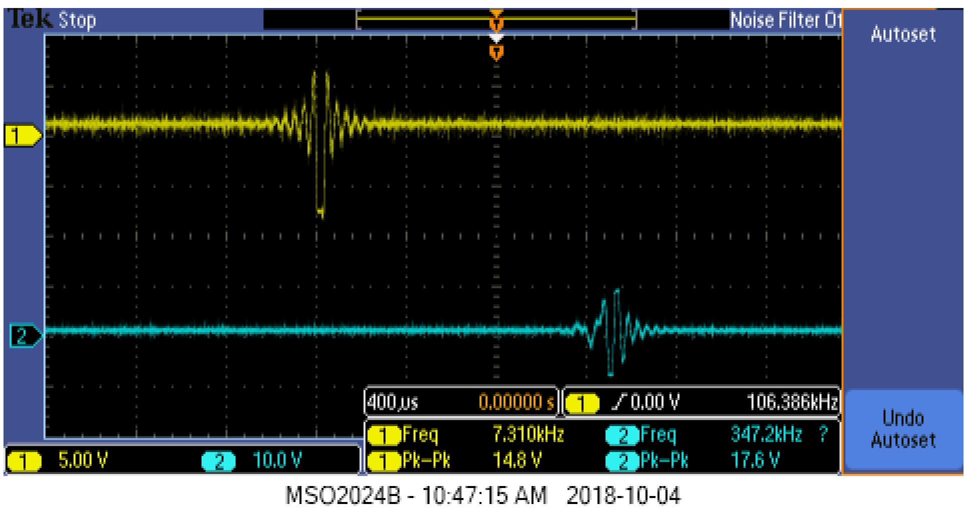

Figure 12: Part 2, System 2

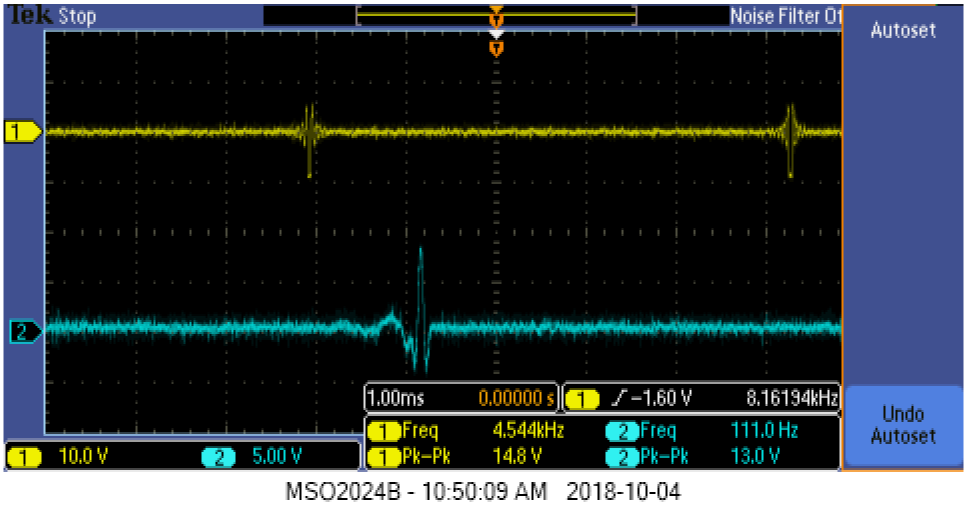

Figure 13: Part 2, System 3

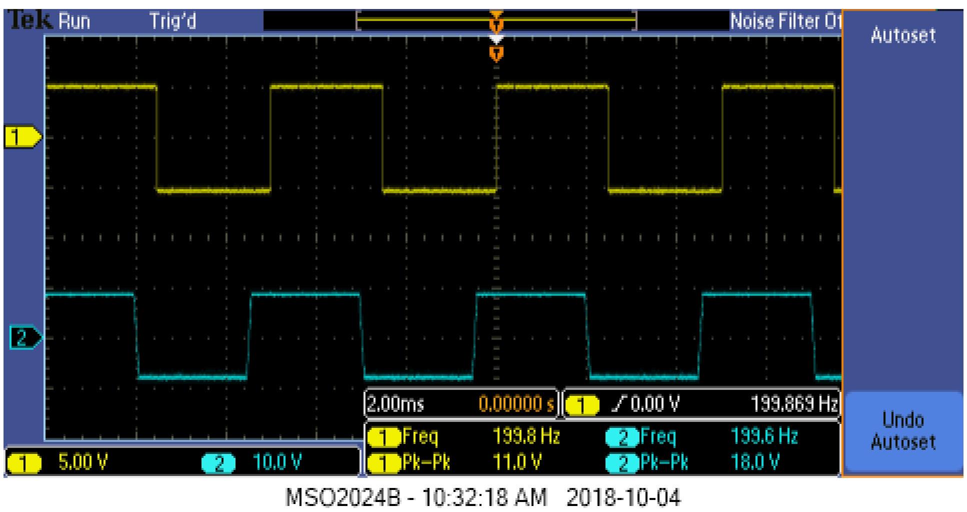

Figure 14: Part 3, System 1

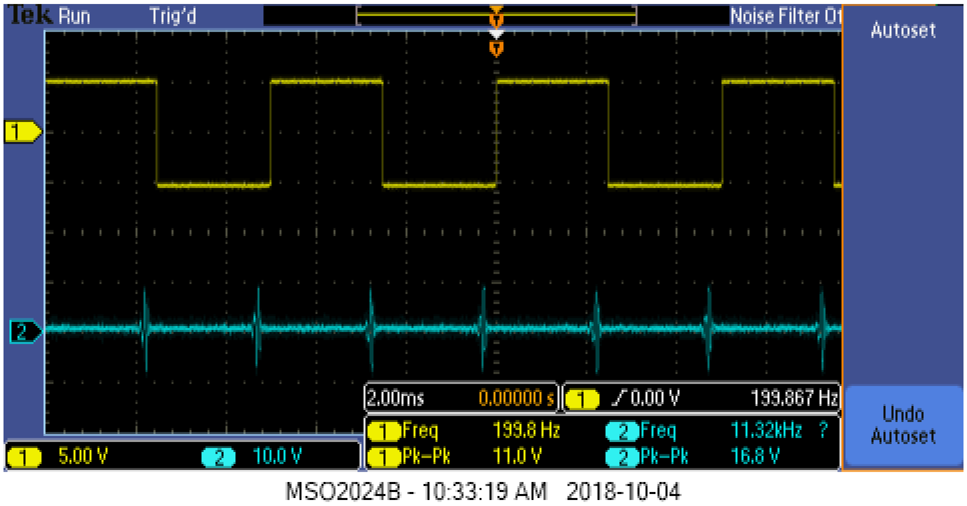

Figure 15: Part 3, System 2

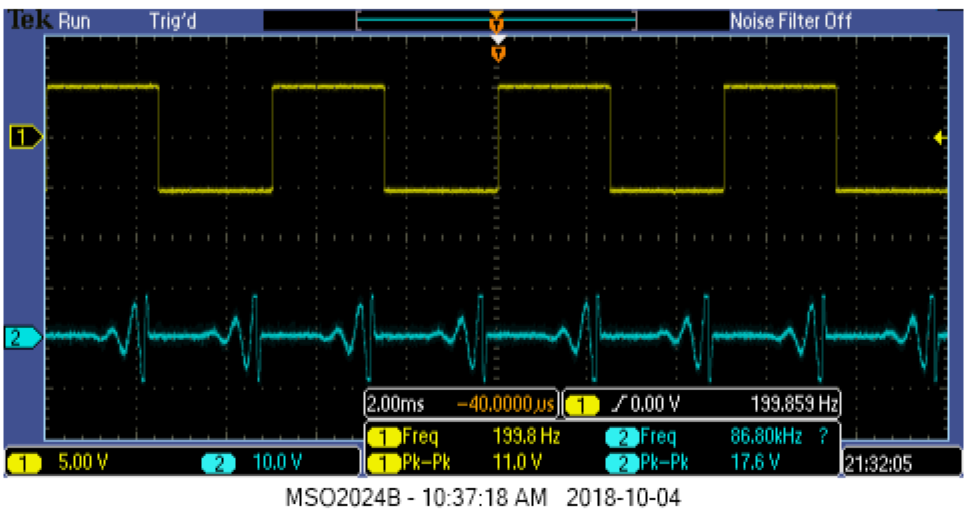

Figure 16: Part 3, System 3