ENGG*3390- Signal Processing

Lab 1

Bilal Ayyache- 0988616

Palak Sood - 0986802

Kathlene Titus - 0954584

October 4th, 2018

Introduction:

The purpose of this lab was to setup and program the Texas Instrument's TMS320C5505 eZdsp USB stick using Code Composer Studio (CSS) to perform simple sampling from continuous-time signals to discrete-time signals. Using 3 sampling rate at a frequency of 48 kHz, 24 kHz, and 8 kHz, the quality of audio from a music source was examined.

Materials:

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

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

Procedure:

The hardware used in the lab included a PC workstation, a DSP, a breakout board, an oscilloscope, a signal generator, and a pair of headphones. The workstation delivered the music to the breakout board through an auxiliary cord, which was then sent to the DSP to sample the signal of the music. The newly sampled signal was then sent back to the breakout board, which sent the new signal to the headphone. The other configuration of the lab was to connect a signal generator to the breakout board which would then send the signal to the oscilloscope and the DSP. The DSP would then sample from the source signal and send the sampled signal back to the breakout board, which sent the sampled signal to the oscilloscope. The oscilloscope would then display the signal of the signal generator and the sampled signal from the DSP for comparison.

The procedure began by importing a CCS project from CourseLink. The main.c file was examined, and it was noted that the program was set to 48 kHz sample rate, giving the best rendition of the audio. The program was run, and music was played from the workstation. The quality of the music was noted for reference. This quality of audio was compared to other sampling rates of 24 kHz and 8kHz by setting the value of SAMPLES_PER_SECOND to 24000 and 8000 respectively. A lower sampling frequency was simulated by passing through every nth sample, where n was 48 kHz. The main.c file was modified to zero every second sample as well as every 5 of 6 samples. Screenshots of the oscilloscope were taken for all the three frequencies. Likewise, a lower sampling frequency was simulated again by using a zero-order hold to interpolate the lost samples. Instead of zeroing every n-1 samples, the last valid sample was repeated every n-1 times for n=2 and n=6. Screenshots of the oscilloscope were taken for all the three frequencies.

Results/Discussion:

Figures 1, 2, and 3 show the waveforms of 3 different DSP sampling rates at 48 kHz, 24 kHz, and 8 kHz. As seen in Figure 3, at a frequency of 20 KHz, the best rendition of the audio

was noticed. In comparison as seen in figure 1, the audio quality created by the 5 kHz frequency was poor. The data in Table 1 shows the ratio between the input and output peak-to-peak voltages from the lab code in figure 16. The low frequency results in a lower quality of the output.

Three different frequencies of sine wave were inputted into the board via the signal generator. As the input frequency increased, the output frequency also increased. This is shown in figures 4, 5, and 6. The board has a gain of approximately 3 and is reflected by the amplitude of the output signal. As the input frequency increased, the ratio of output to input voltage decreased. Decimation process was used to create the following results. Decimation is the process of reducing storage and computation requirements. Decimation creates a new sequence by selecting every nth sample resulting in a time compression. To increase the sampling rate, the process of interpolation is applied.

During this part, decimation was achieved by sampling at the original frequency of 48 kHz and stimulating a lower sampling frequency f_s by passing through every nth sample and zeroing out every n-1 samples, where n = 48 kHz /fs. The code in Figure 16 shows the modified main.c to zero every 2^{nd} sample. The modulus operator was used to make the sample of every even number equal to 0. This was done by checking if the remainder was 0 when every sample was divisible by 2.

To create figure 7, 8, and 9, a program was coded to simulate different sampling rates 5 kHz, 10 kHz, and 20 kHz. The code in Figure 17 shows the modified main.c used to zero out 5 of 6 samples. This was done using the modulus operator to simulate a lower frequency at every 6^{th} sample, making the rest 0. The output was set to the input received every 6^{th} sample. It was found that as the sampling rate decreased the output became less accurate. The output also became more discrete. This makes sense since less samples were collected and displayed as an output voltage.

A lower sampling frequency was stimulated by now using a “zero-order” hold to interpolate the lost samples. Instead, of zeroing every n - 1 samples, the last valid sample is repeated n-1 times. All observations for the third part can be found in figures 10-15. As the sampling rate decreased, the output became more step like and less accurate. This makes sense because values were repeated which leads to horizontal “steps”. The code in Figure 18 shows the modified main.c that repeats the last valid sample if the remainder was 0 when every sample was divisible by 2. Similarly, the code in Figure 19 shows the modified main.c that repeats the last valid sample if the remainder was 0 when every sample was divisible by 6.

Conclusions:

In conclusion, the quality of audio was determined using 3 sampling rates at frequency of 48 kHz, 24 kHz, and 8 kHz. As the input frequency increased, the output frequency also increased. Thus, the lower the frequency, the worse the quality of the audio. To increase the sampling rate, the process of interpolation was applied. During this part, decimation was achieved by sampling at the original frequency of 48 kHz and stimulating a lower sampling frequency f_s by passing through every nth sample and zeroing out every n-1 samples. It was

found that as the sampling rate decreased by lowering the frequency, the output became less accurate and discrete. A “zero-order” hold was used to interpolate the lost samples. As the sampling rate decreased, the output became more step like and less accurate.

References:

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

Appendix

Frequency (kHz)	Sampling Rate (kHz)	Input	Output
20	48	1.08 V	560 mV
10	24	960 mV	680 mV
5	8	1.04 V	120 mV

Table 1: Ratio between the input and output peak-to-peak voltages from original lab code

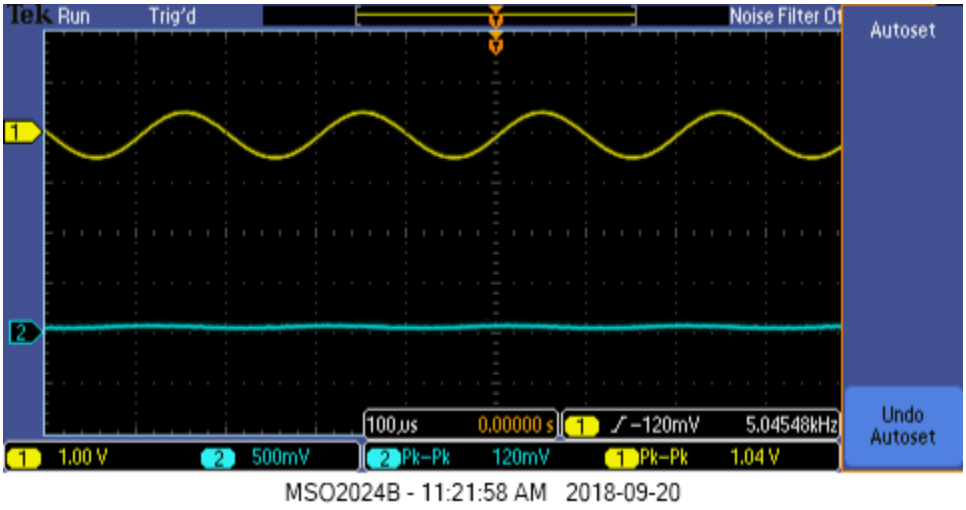

Figure 1: Sampling rate of f_s =8kHz at frequency of 5kHz

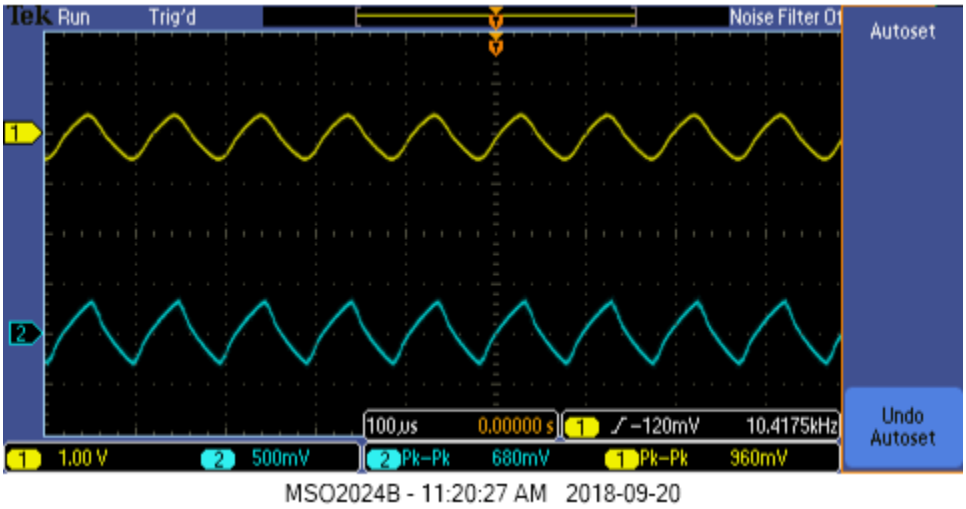

Figure 2: Sampling rate of f_s =24kHz at frequency of 10kHz

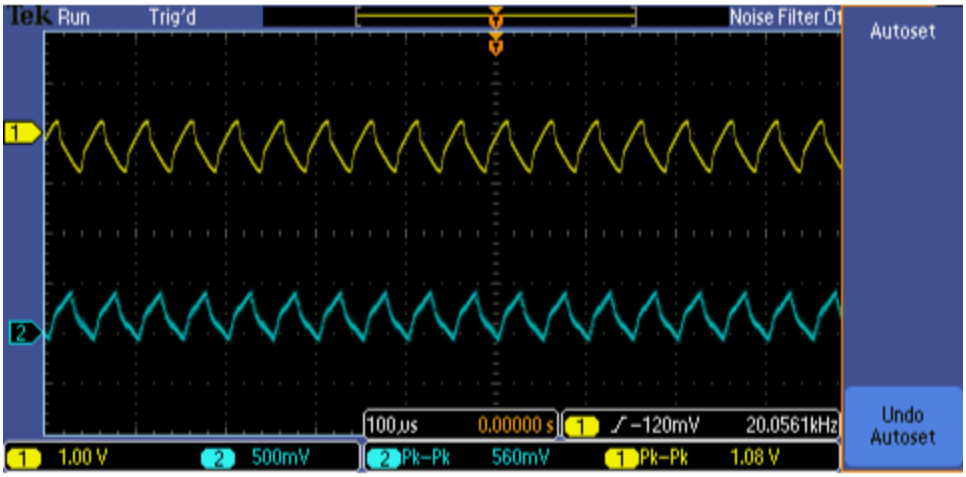

Figure 3: Sampling rate of f_s =48kHz at frequency of 20kHz

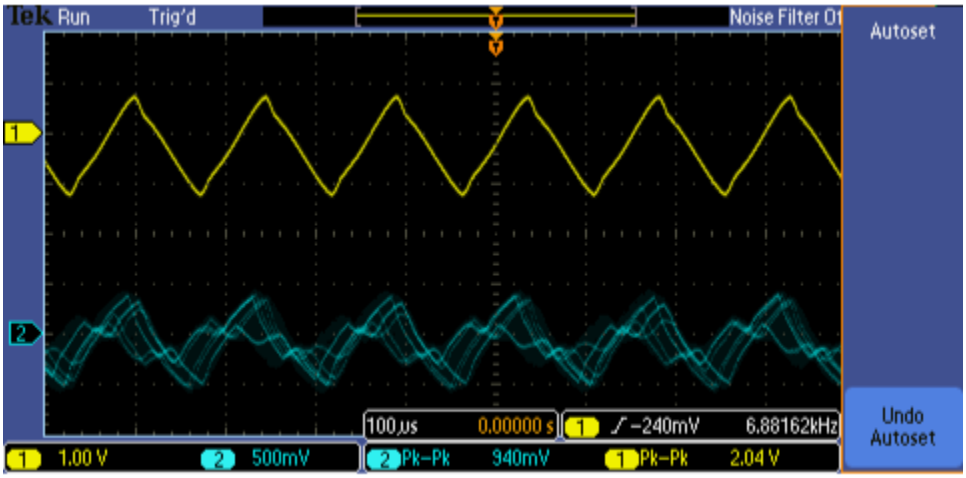

Figure 4: Sampling rate of f_s =24kHz at frequency of 5kHz (n=2)

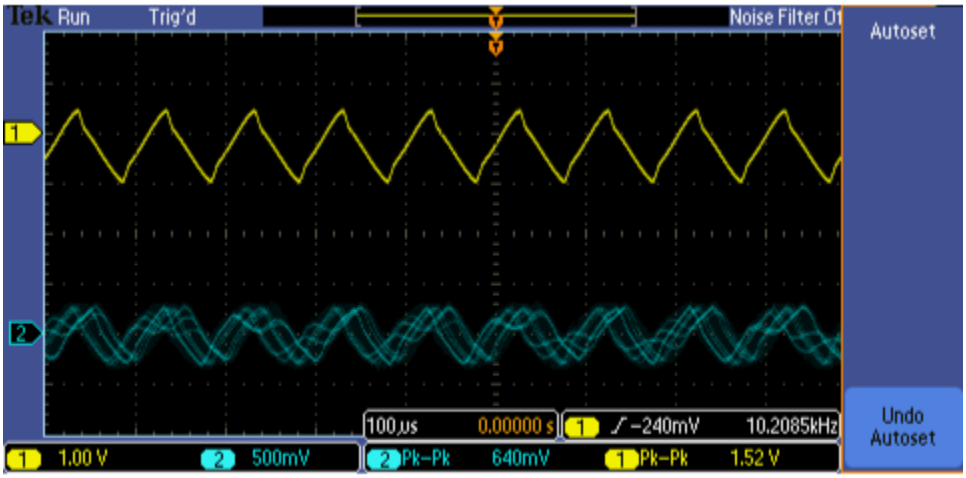

Figure 5: Sampling rate of f_s =24kHz at frequency of 10kHz (n=2)

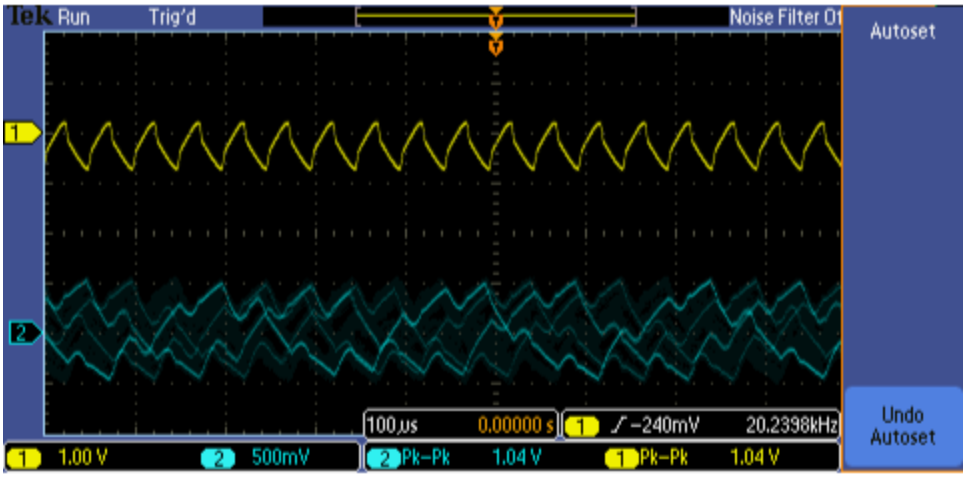

Figure 6: Sampling rate of f_s =24kHz at frequency of 20kHz (n=2)

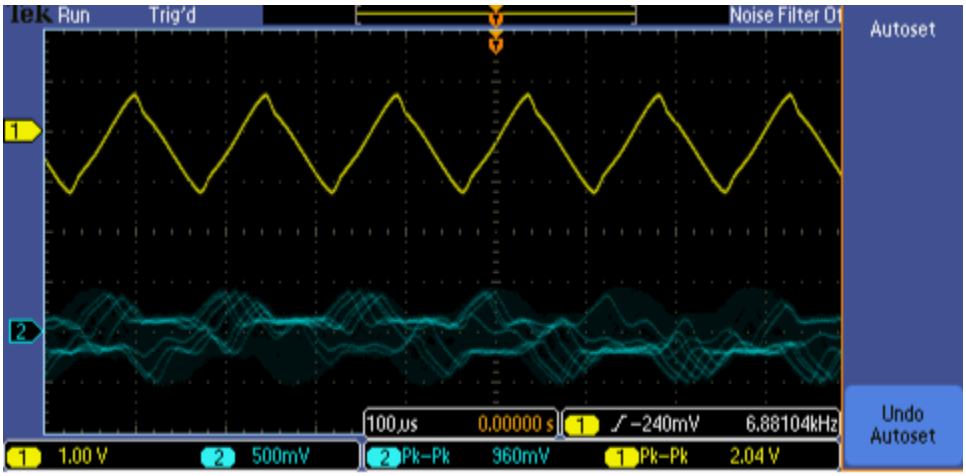

Figure 7: Sampling rate of f_s =8kHz at frequency of 5kHz (n=6)

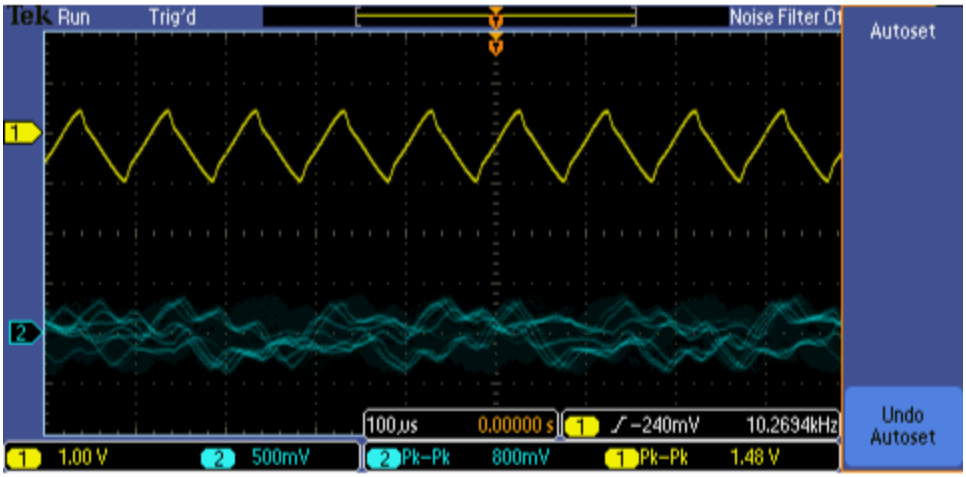

Figure 8: Sampling rate of f_s =8kHz at frequency of 10kHz (n=6)

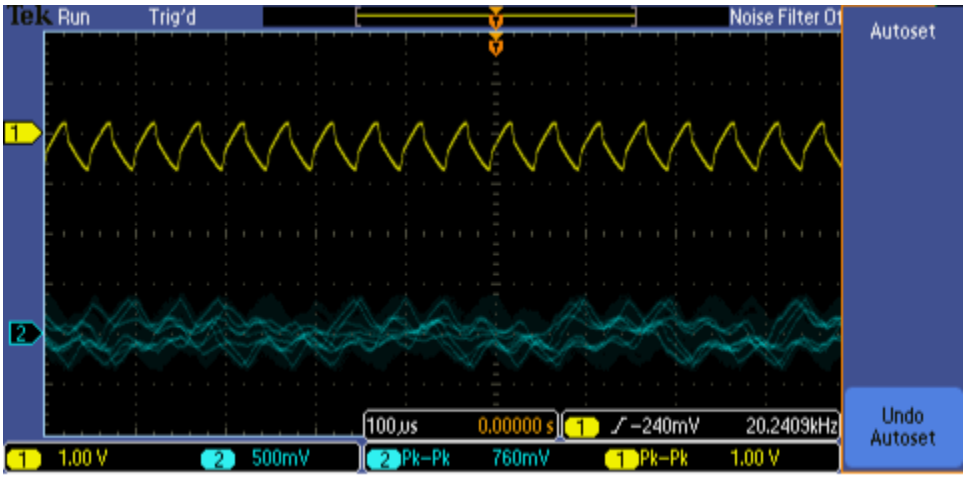

Figure 9: Sampling rate of f_s =8kHz at frequency of 20kHz (n=6)

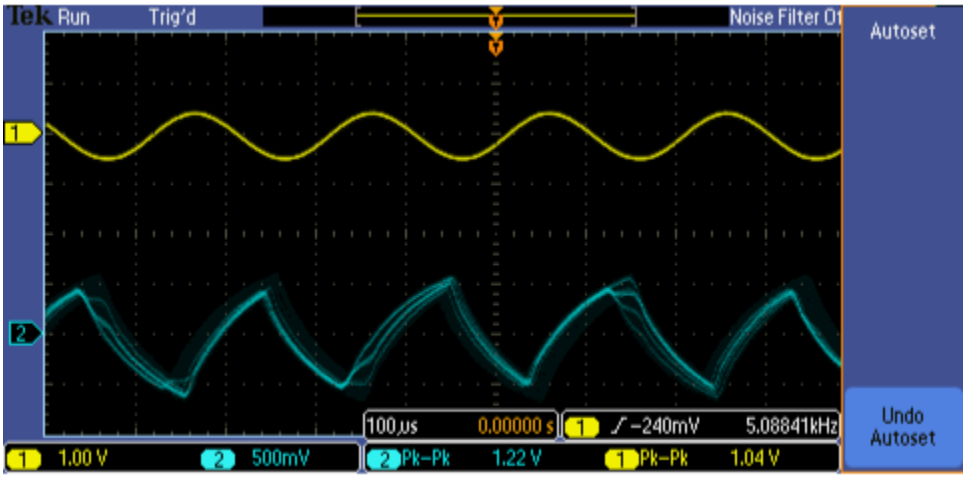

Figure 10: Sampling rate of f_s =24kHz at frequency of 5kHz, last sample repeated n-1 times (n=2)

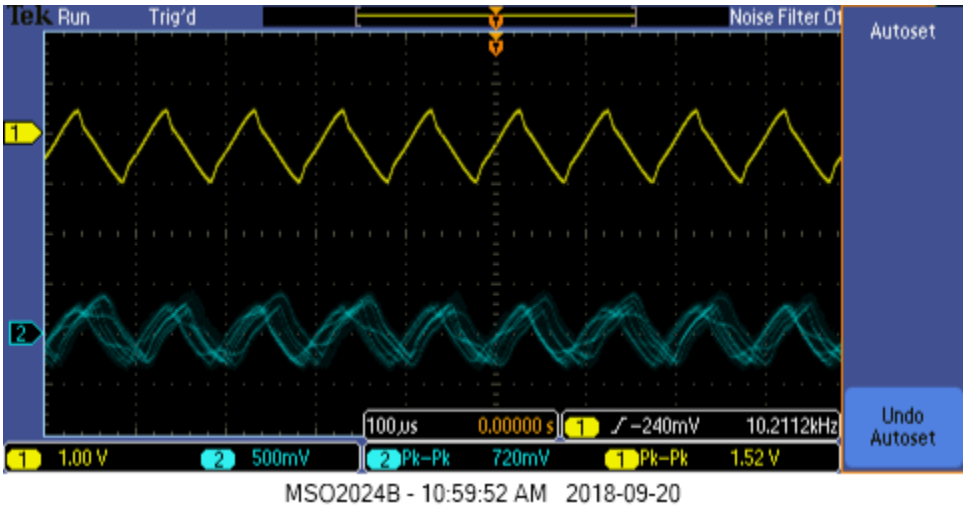

Figure 11: Sampling rate of f_s =24kHz at frequency of 10kHz, last sample repeated n-1 times (n=2)

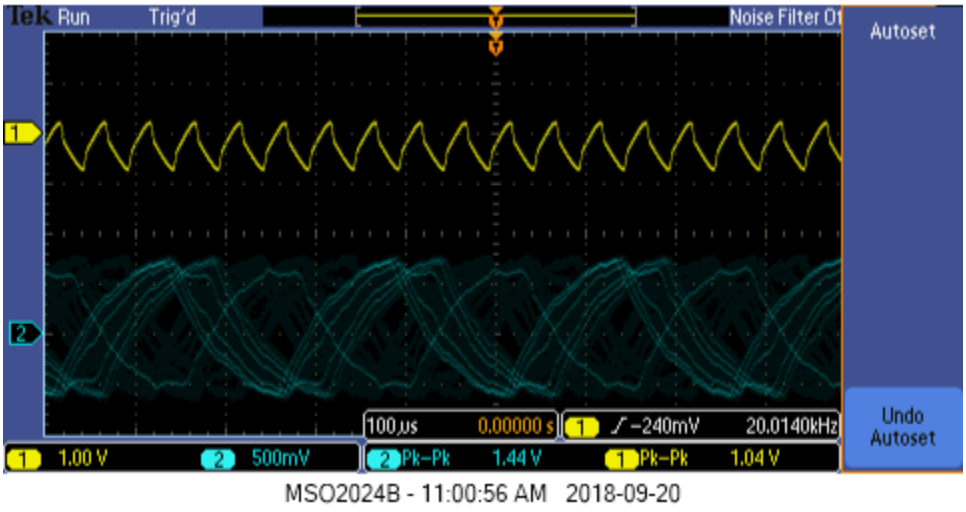

Figure 12: Sampling rate of f_s =24kHz at frequency of 20kHz, last sample repeated n-1 times (n=2)

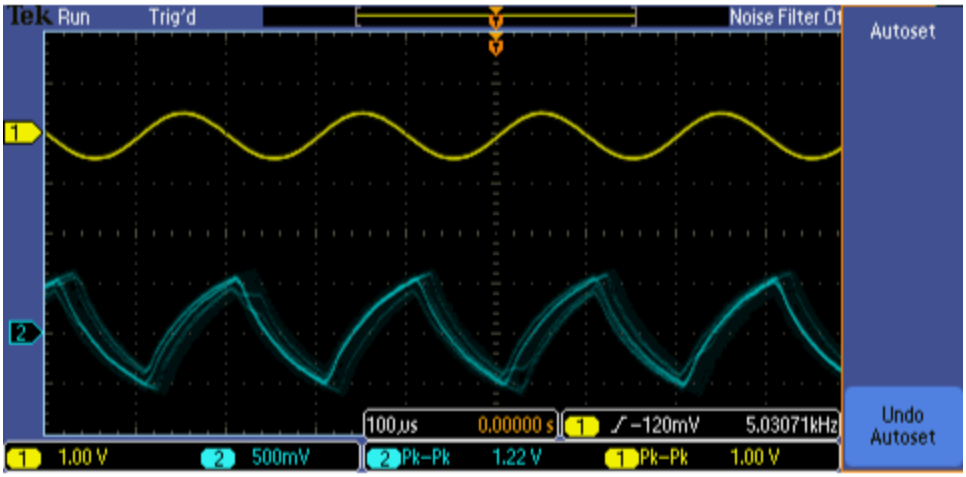

Figure 13: Sampling rate of f_s =8kHz at frequency of 5kHz, last sample repeated n-1 times (n=6)

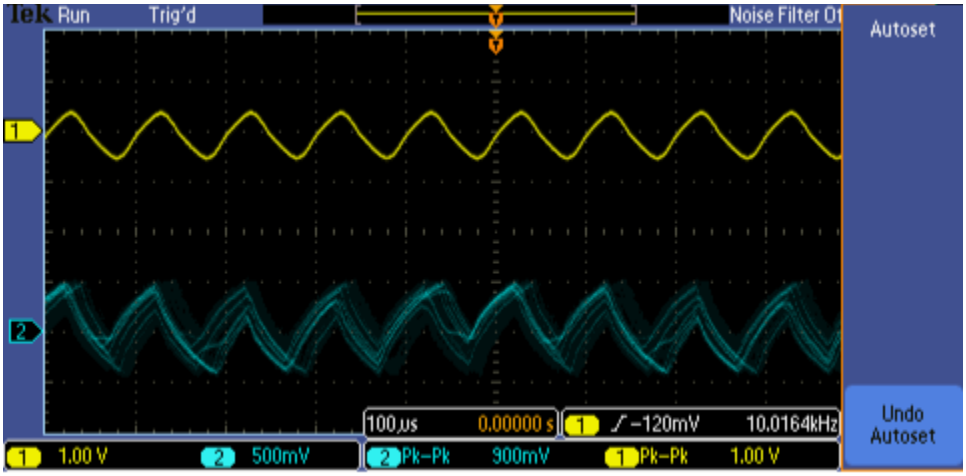

Figure 14: Sampling rate of f_s =8kHz at frequency of 10kHz, last sample repeated n-1 times (n=6)

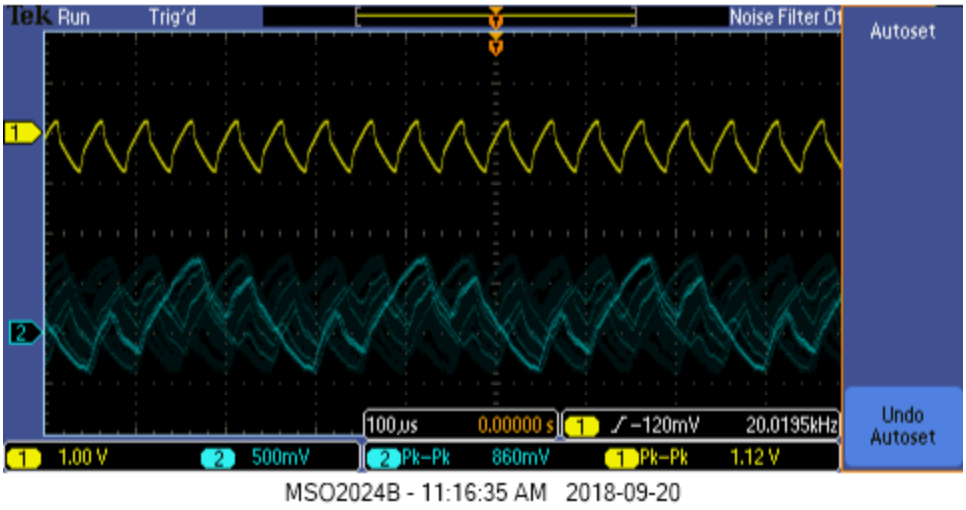

Figure 15: Sampling rate of f_s =8kHz at frequency of 20kHz, last sample repeated n-1 times (n=6)

```
asm(" bclr XF");

 for ( i = 0  ; i < SAMPLES_PER_SECOND * 600L  ;i++  )
  {
    aic3204_codec_read(&left_input, &right_input); // Configured for one interrupt per two channels.

    left_output =  left_input;           // Very simple processing. Replace with your own code!
    right_output = right_input;    // Directly connect inputs to outputs.

    if (i % 2 == 0){
        left_output = 0;
        right_output = 0;
    }
    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 16: main.c code to simulate a lower sampling frequency f_s by passing through every nth sample and zeroing the other n-1 samples(n=2)

```
 for ( i = 0  ; i < SAMPLES_PER_SECOND * 600L  ;i++  )
  {
    aic3204_codec_read(&left_input, &right_input); // Configured for one interrupt per two channels.


    if (i % 6 == 0){
        left_output =  left_input;           // Very simple processing. Replace with your own code!
        right_output = right_input;}// Directly connect inputs to outputs
    else{
        left_output = 0;
        right_output = 0;
    }
    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 17: main.c code to simulate a lower sampling frequency f_s by passing through every nth sample and zeroing the other n-1 samples(n=6)

```
    if (i % 2 == 0){
        left_output = templeft;
        right_output =tempright;
    }
    else{
        left_output=  left_input;          // Very simple processing. Replace with your own code!
             right_output = right_input;
             tempright= right_output;
             templeft=left_output;

    }
    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 18: main.c code to simulate a lower sampling frequency f_s by using a "zero-order" hold to interpolate the lost samples. The last valid sample repeated n-1 times (n=2)

```
    if (i % 6==0){
        left_output = templeft;
        right_output =tempright;
    }
    else{
        left_output=  left_input;          // Very simple processing. Replace with your own code!
             right_output = right_input;
             tempright= right_output;
             templeft=left_output;

    }
    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 19: main.c code to simulate a lower sampling frequency f_s by using a "zero-order" hold to interpolate the lost samples. The last valid sample repeated n-1 times (n=6)