Application Note

ADC Oversampling

Omer Amir and Joseph Casuga

ABSTRACT

C2000™ real-time microcontrollers offer analog-to-digital converters (ADCs) that are widely used across numerous applications from controlling motor to reading sensors. There are times when a customer design demands a resolution higher than the ADC of the selected device. This application note describes how an oversampling method can be incorporated to increase ADC resolution past the currently available number of bits. This can help reduce the cost in building a system by utilizing lower resolution ADCs to oversample a signal and obtain a higher resolution result. Detailed instructions are provided and have been tested on the TMDSCNCD280039C device, using SysConfig for the device initialization.

Table of Contents

List of Figures

List of Tables

Trademarks

C2000™ is a trademark of Texas Instruments.
All trademarks are the property of their respective owners.

1 Introduction

Analog-to-Digital Converter (ADC) modules have a discrete number of bits available to digitize an analog signal, or resolution. An ideal ADC faithfully reproduces the digitized signal to within the specified resolution. However, in the real world, various electrical imperfections and noise factors contribute to reduce the realized signal resolution below the specified value. The realized signal resolution when these imperfections are considered is referred to as the effective number of bits, or ENOB.

ADC signal oversampling is a technique that can overcome these inherent imperfections, and achieve a higher ENOB than is nominally possible at the baseline for the device. This application report discusses the purpose behind oversampling, and provides the following details of an oversampling example: the theory, the hardware and software setup, and the measured results. The example provided in this application note uses a TMDSCNCD280039C device, with a 12-bit ADC.

2 Theory

The goal of software oversampling is to increase ENOB by reducing the noise observed in the signal. Software oversampling performs multiple conversions on the same input signal and accumulates the digital values to attain an ENOB higher than the ADC's inherent ENOB. The result's precision increases, depending on how much oversampling takes place. This accuracy can be demonstrated by measuring a varying input signal to determine the signal's major frequency. The amount of oversampling possible is theoretically limited to the data width of the variable used to store the conversion result. For instance, a 16-bit result word limits you to 16X oversampling on a 12-bit ADC, with a maximum accumulated value of 65535.

In addition to data size constraints, the amount of oversampling is limited by the relationship between the throughput of the ADC and the fundamental frequency of the input signal, as the number of oversampled conversions per second cannot fall below the Nyquist rate. This also means that the oversampling factor is limited by the control loop frequency needed to achieve the system performance requirements.

The size limit occurs because oversampling accumulates the results, which invariably requires more memory than the original result because there can be an overflow from the addition. The accumulated values are not averaged since this effectively removes the additional precision that is obtained. As such, averaging maintains the size of the stored result and the reduced noise, but this does not affect the observed ENOB of the result to any significant degree.

Oversampling with accumulation improves noise reduction in the final value obtained, but the ENOB does not increase as much if there is significant noise affecting the signal. There are several board layout guidelines that, if followed, can help to minimize significant sources of noise in analog signals for ADC conversion. These include:

- Verifying no signal crossing between analog and digital signals
- Having separate layers for analog and digital signals
- Having a dedicated return ground for analog signals that are not shared with digital
- Isolating the analog region from the digital region

For more details about good hardware design for C2000 ADCs, see Section 3.

A Fast Fourier Transform (FFT) is used in this document to process the oversampled ADC results stored in memory. The FFT plot gives us a view of the signal noise and harmonic distortions that affect the observed major frequency, and as such diminish the ENOB. These values are quantified from the FFT data and used to compute an approximate ENOB value. For the purpose of testing, the FFT was computed on ADC data exported from RAM. Before the ADC results have an FFT performed on them, windowing is required on data stored in memory to avoid creating artifacts in the signal. This is because the start and end points do not always line up to form a complete waveform. The windowing function used in this application note is the 7-term Blackman-Harris function. The FPU DSP library also has the capability of performing fast Fourier transforms on data in stored memory using windowing. The different windowing functions available can be viewed in the FFT module within SysConfig, or within the directory C2000Ware_X_XX_XX_XX\libraries\dsp\FPU\c28\include\fpu32 as files labeled fpu_fft_<name>.h.

The magnitude of noise present in a signal can be expressed using the Signal-to-Noise Ratio (SNR), and the harmonics observed in the signal can be expressed using Total Harmonic Distortion (THD). The noise and harmonics present in the sampled signal reduce the ENOB of the result. Table 2-1 shows the theoretical ENOB increase and SNR improvement possible with various oversampling factors. For more data on the theory behind the numbers in this table, see *General Oversampling of MSP ADCs for Higher Resolution*.

Table 2-1. Relationship Between Oversampling Factor, SNR, and Extra Bits of Resolution

Oversampling Factor	SNR Improvement (dB)	Extra Bit of Resolution
2	3	0.5
4	6	1
8	9	1.5
16	12	2
32	16	2.5
64	18	3
128	21	3.5
256	24	4
512	27	4.5
1024	30	5
2048	33	5.5
4096	36	6

In the example shown in this application note, each oversampling factor from baseline to 16X is tested using a 10 kHz sine wave input signal. An FFT plot is used to display the results here because the plot visualizes the signal to noise ratio, harmonic distortion, and accuracy of the sampling. The noise frequencies present in the signal can be observed as minor peaks, which are far below the peak of the DC and signal frequency. Excluding the peak at 0, which is the DC component of the signal, the highest peak is the closest to the input signal's frequency. The more diminished these are relative to the fundamental signal amplitude, the higher the resulting ENOB.

3 Hardware

For the purpose of testing ADC oversampling, a TMDSCNCD280039C controlCARD was used to convert the input sine wave into digital values. To configure the reference voltage VREF and JTAG for the controlCARD, see the *TMS320F280039C controlCARD Information Guide*. To keep the setup simple while reducing possible sources of error, the internal 2.5 V reference was used. If the external VREF is used, extra steps must be taken. For more information regarding VREF, see the *Voltage Reference* chapter in the *ADC* chapter of the *TMS320F28003x Real-Time Microcontrollers Technical Reference Manual*.

The hardware for ADC sampling can reduce environmental and signal noise when configured properly. In the context of evaluating oversampling performance, equipment can be a source of noise. Use signal sources with a high resolution and follow practices for reducing noise in the system for a validation setup. For this application note, the Agilent AG33522A Arbitrary Waveform Generator (AWG) was used as the signal source. In general, a signal source with a higher resolution than the ADC produces the best results. To reduce possible deviations in obtained ENOB values, follow the best layout practices for analog circuits. ADC input conditioning also plays a role in improving the accuracy of ADC itself. For details on input conditioning, see *ADC Input Evaluation for C2000™ MCUs*. For PCB layout design recommendations, see *Hardware Design Guide for F2800x C2000™ Real-Time MCU Series*.

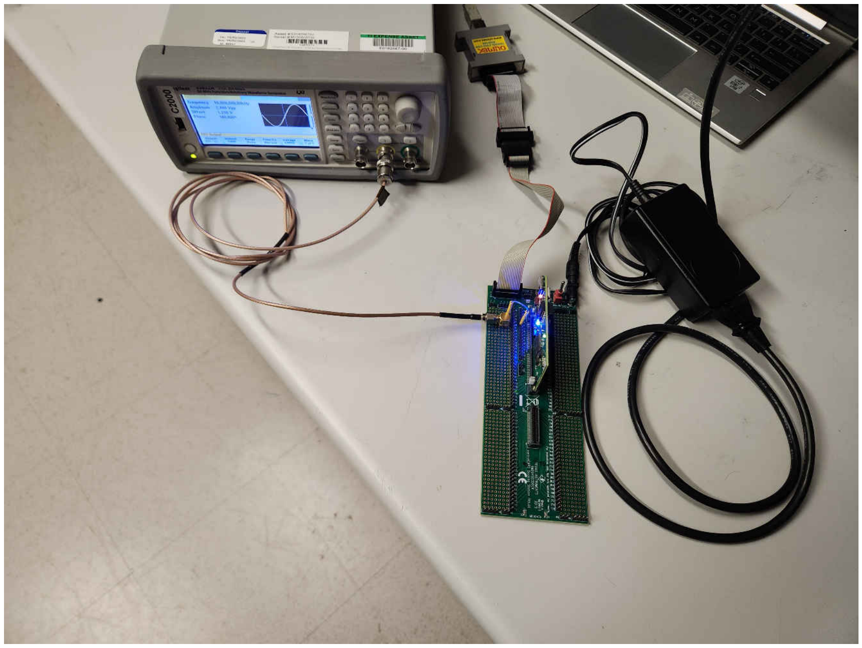

Figure 3-1. Overall Hardware Setup

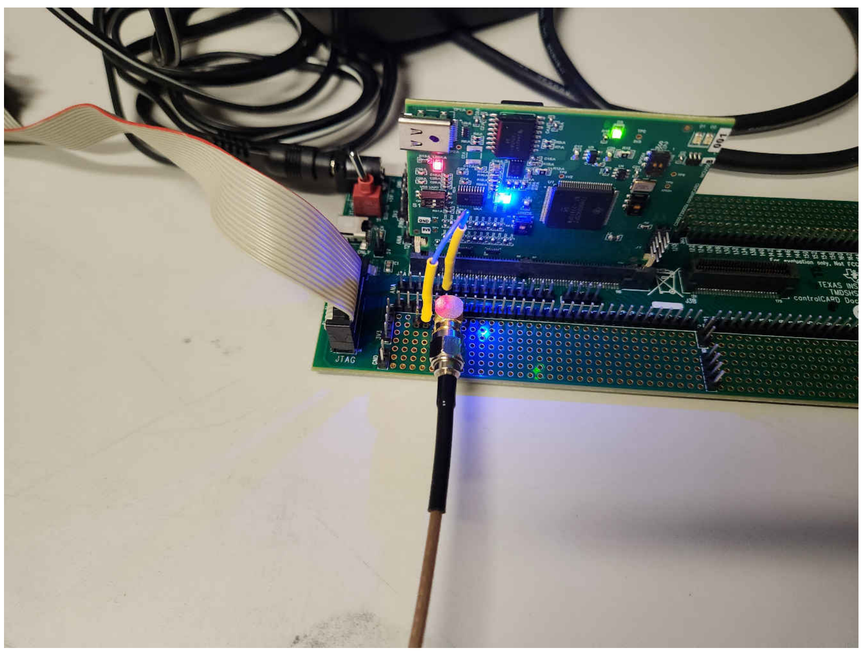

Figure 3-2. Wiring Setup

4 Software

The example used for this application note utilizes SysConfig with driverlib for the configuration of the ADC, ePWM, and other peripherals. Within the program code, the ADC should be set up to minimize overhead such that more time can be used between conversions to do a control loop. For the example used in this application note, the SOCs are configured in burst mode with round-robin priority, so that the SOCs are triggered together and accumulated without missing a value when oversampling. An interrupt is set up to trigger once the last SOC, SOC15 for F28003x, reaches the end of conversion. The interrupt runs the corresponding ISR, which stores the ADC result and accumulate multiple SOC results if oversampling is enabled.

The ePWM triggers the SOCs here, however software triggers and CPU timer triggers are also available. Take care when choosing the period of the trigger to maintain uniform sampling of the SOCs, and appropriate conversion time with respect to the rest of the control loop. Once the last SOC in the burst sequence issues an end-of-conversion signal, the ISR executes the control loop. In this example, the control loop consists of a simple accumulation function for oversampling and storing the results. Avoid averaging the values because this effectively reduces measurement precision by discarding information contained in the lower bits of the result. The final result is stored in memory before the next burst is triggered.

Below is an example of baseline sampling with the burst ISR setup:

```
__interrupt void adcA1ISR(void)
{
    //
    // Clear the interrupt flag
    //
    ADC_clearInterruptStatus(ADCA_BASE, ADC_INT_NUMBER1);

    //
    // 1X Oversampling
    //
    lv_results[nloops++] = ADC_readResult(myADC0_RESULT_BASE, ADC_SOC_NUMBER0);

    //
    // Check if overflow has occurred
    //
    if(true == ADC_getInterruptOverflowStatus(ADCA_BASE, ADC_INT_NUMBER1))
    {
        ADC_clearInterruptOverflowStatus(ADCA_BASE, ADC_INT_NUMBER1);
        ADC_clearInterruptStatus(ADCA_BASE, ADC_INT_NUMBER1);
    }

    //
    // Check if all results are stored
    //
    if(nloops >= numBins)
    {
        //
        // Disable ADC interrupt
        //
        ADC_disableInterrupt(myADC0_BASE, ADC_INT_NUMBER1);
        ESTOP0;
    }

    //
    // Acknowledge the interrupt
    //
    Interrupt_clearACKGroup(INTERRUPT_ACK_GROUP1);
}
```

An example of oversampling a signal at 8X with ISRs is as follows:

```
__interrupt void adcA1ISR(void)
{
    //
    // Clear the interrupt flag
    //
    ADC_clearInterruptStatus(ADCA_BASE, ADC_INT_NUMBER1);

    //
    // Accumulate 8 ADC results to oversample 8X
    //
    lv_results[nloops++] = (ADC_readResult(myADC0_RESULT_BASE, ADC_SOC_NUMBER0) +
ADC_readResult(myADC0_RESULT_BASE, ADC_SOC_NUMBER1) +
            ADC_readResult(myADC0_RESULT_BASE, ADC_SOC_NUMBER2) +
ADC_readResult(myADC0_RESULT_BASE, ADC_SOC_NUMBER3) +
            ADC_readResult(myADC0_RESULT_BASE, ADC_SOC_NUMBER4) +
ADC_readResult(myADC0_RESULT_BASE, ADC_SOC_NUMBER5) +
            ADC_readResult(myADC0_RESULT_BASE, ADC_SOC_NUMBER6) +
ADC_readResult(myADC0_RESULT_BASE, ADC_SOC_NUMBER7));

    //
    // Check if overflow has occurred
    //
    if(true == ADC_getInterruptOverflowStatus(ADCA_BASE, ADC_INT_NUMBER1))
    {
```

```
        ADC_clearInterruptOverflowStatus(ADCA_BASE, ADC_INT_NUMBER1);
        ADC_clearInterruptStatus(ADCA_BASE, ADC_INT_NUMBER1);
    }

    //
    // Acknowledge the interrupt
    //
    Interrupt_clearACKGroup(INTERRUPT_ACK_GROUP1);
}
```

The appropriate interrupts can be disabled once the intended number of results have been stored, or else the ADC can continue to convert the analog signal. The basic flow of using an ePWM to trigger the burst conversion for oversampling is shown in Figure 4-1.

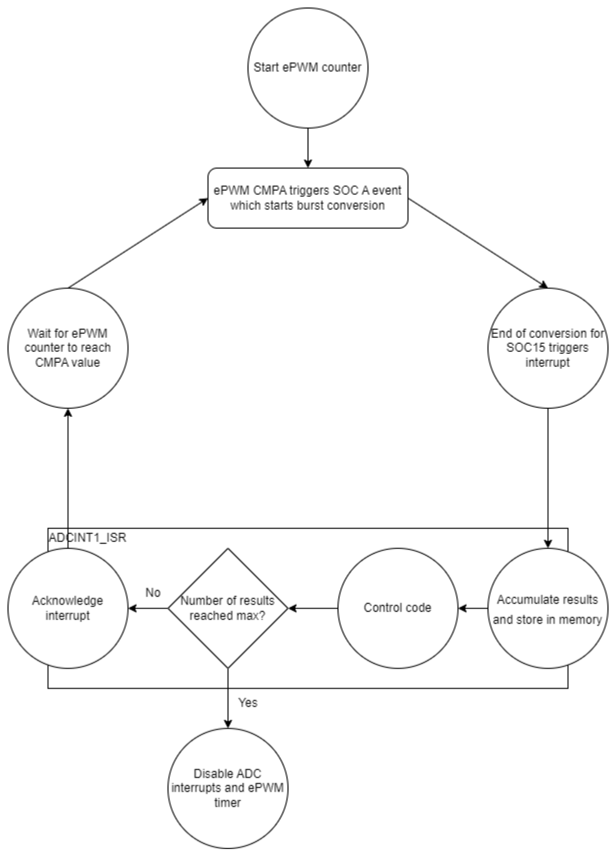

Figure 4-1. SoC Flow Diagram for Oversampling

Depending on the control loop for the specific application, more time may be required than what the max sampling rate of the ADC allows. To solve this, increase the ePWM time base to allow a longer conversion time, giving the control loop more time to complete. This reduces the maximum frequency that can be properly measured, since the ADC does not trigger as often.

The input frequency affects the oversampling factor that can be used. For signals that are at a higher frequency or need to be sampled at a higher rate, a lower oversampling factor is necessary because of the software overhead required. To determine the maximum input frequency where data is not likely to be missed, the number of cycles needed for the control loop and oversampling are needed. The control loop cycle count includes any user-related operations such as ISR handling or processing that need to happen every time new samples are obtained. Figure 4-2 shows where these timings come into play when sampling a signal. In this image, the oversampling and control loop time includes system clock cycles for interrupt latency and ISR execution. Notice that there is some buffer time between the end of the control loop and the arrival of the next ADC trigger, so that processing does not prevent a trigger from occurring and data is not missed. Figure 4-3 shows that when the total time for conversions, oversampling, and the control loop exceeds the burst trigger period, data is missed. The solution for this is to extend the period, which in this example would require extending the ePWM time base to move the trigger further.

Table 4-1 shows the timings of the oversampling used in this application note, which includes time for reading results from the ADC register, accumulating values if necessary, and storing the result in RAM. The timings for this table were taken with only the --opt_for_speed = 5 for optimization, so the timings are not necessarily the minimum achievable values. For more details on how to improve the speed of a program, see the C2000 C28x Optimization Guide.

If the control loop timing is not known, a simple GPIO toggle is accurate enough to determine the period of this loop. The function below can be used to route the SOC A event trigger to the corresponding external pin. This can be used to verify the event is triggering properly, and that the ISR has sufficient time to run before the burst gets triggered again.

```
SysCtl_enableExtADCSOCSource(SYSCTL_ADCSOC_SRC_PWM1SOCA)
```

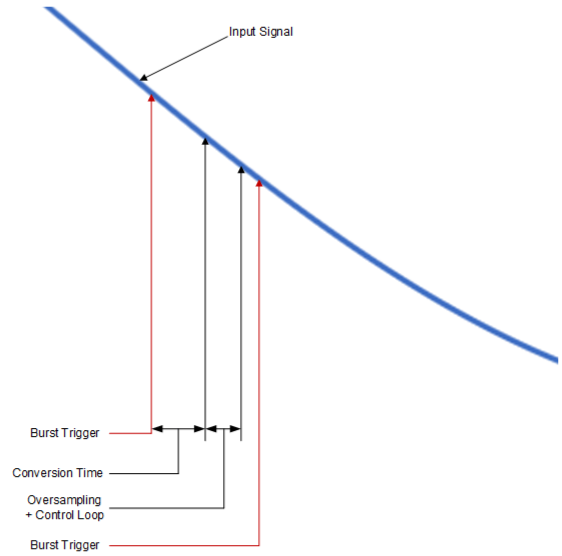

Figure 4-2. Timings for Sampling a Signal

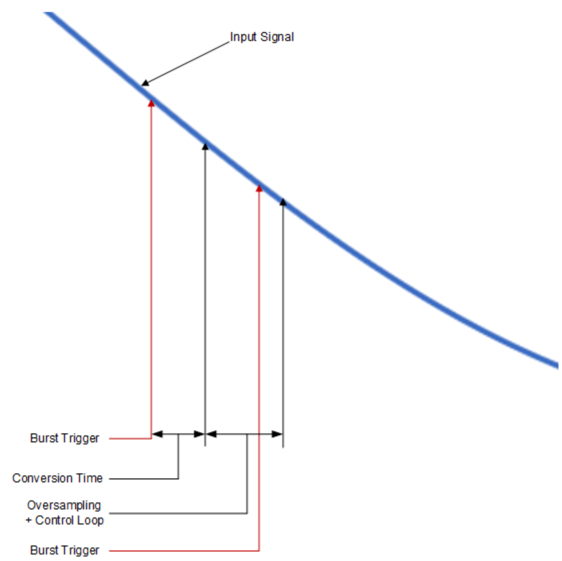

Figure 4-3. Incorrect Timings for Sampling a Signal

Table 4-1. Oversampling Time

Oversampling Factor	Oversample Time (clock cycles)
1X	9
2X	52
4X	127
8X	272
16X	551

Take for example a control loop within an interrupt service routine (ISR) that takes about 300 cycles to run. Measuring a sine wave at 16X oversampling with ISR uses 851 cycles. If the sine wave is 10 kHz, based on the Nyquist theorem, the minimum sampling rate is at least 20 kSPS (kilo-samples per second). The table in the *ADC Timing Diagrams* chapter of the *TMS320F28003x Real-Time Microcontrollers Technical Reference Manual* shows how t_{LAT} increases with larger ADC clock prescale values. SYSCLK is the system clock frequency. This is 120 MHz by default for the TMDSCNCD280039C. For an ADC clock of 60 MHz on the TMDSCNCD280039C, the clock prescaler divides SYSCLK by 2, and the value for t_{LAT} is 23 SYSCLK cycles. F_{Sample} is the rate in samples per second required for a specific application, which is 20 kSPS here.

$$ACQPS_{MAX} = \frac{SYSCLK}{F_{Sample}} - t_{LAT} - 1 \quad (1)$$

$$Cycles_{Sample} = t_{LAT} + ACQPS + 1 \quad (2)$$

$$Maximum\ Input\ Frequency = \frac{SYSCLK}{2 \times \left(Cycles_{Sample} + Cycles_{Control\ Loop} + Cycles_{Oversample}\right)} \quad (3)$$

In this example the maximum acquisition window size (ACQPS), based on the above formula, is 5,976. This value is very large only because the sampling rate does not have a very high requirement. Having a maximum ACQPS value is important so that there is sufficient time to sample an input without missing significant data points, as the ACQPS itself is determined by the input network. For more information on calculating ACQPS values, see the *Choosing an Acquisition Window Duration* section within the *ADC* chapter in *TMS320F28003x Real-Time Microcontrollers Technical Reference Manual*. The maximum input frequency measurable with this example setup is about 67 kHz, given the Nyquist rate. For comparison, the data collected in this application note was sampled at about 3 MSPS, which can only be achieved by using an ACQPS value of 16 or less. The maximum input frequency measurable using this sample rate was about 74 kHz, given that the ISR time was about 211 cycles.

5 Results

The results of ADC oversampling are shown in Table 5-1. The baseline ENOB for this data is noticeably lower than the value provided in the device data manual because the setup used for this example was simplified, without buffering the input to the ADC, using a reliable external voltage reference, or otherwise optimizing to reduce noise in the system. Overall, the ENOB increased by approximately 1.56, which is close to the theoretical amount of increase in resolution. This shows that the accuracy of an ADC can be increased without changing hardware and adding extra cost to the bill of materials. With an increase in oversampling, the amount of time required to sample the ADC also increases. This can have diminishing returns, depending on how time-sensitive activities within the rest of the system are. For more details on this, see Table 4-1.

Table 5-1. ADC Oversampling Results

Oversampling Factor	ENOB	THD	SNR	FFT Result
1X	10.81	-84.02	66.91	Figure 5-1
2X	11.18	-83.99	69.24	Figure 5-2
4X	11.51	-84.41	71.26	Figure 5-3
8X	11.91	-86.29	73.70	Figure 5-4
16X	12.37	-86.51	76.63	Figure 5-5

Figure 5-1 through Figure 5-5 show the FFT plots for each oversampling factor.

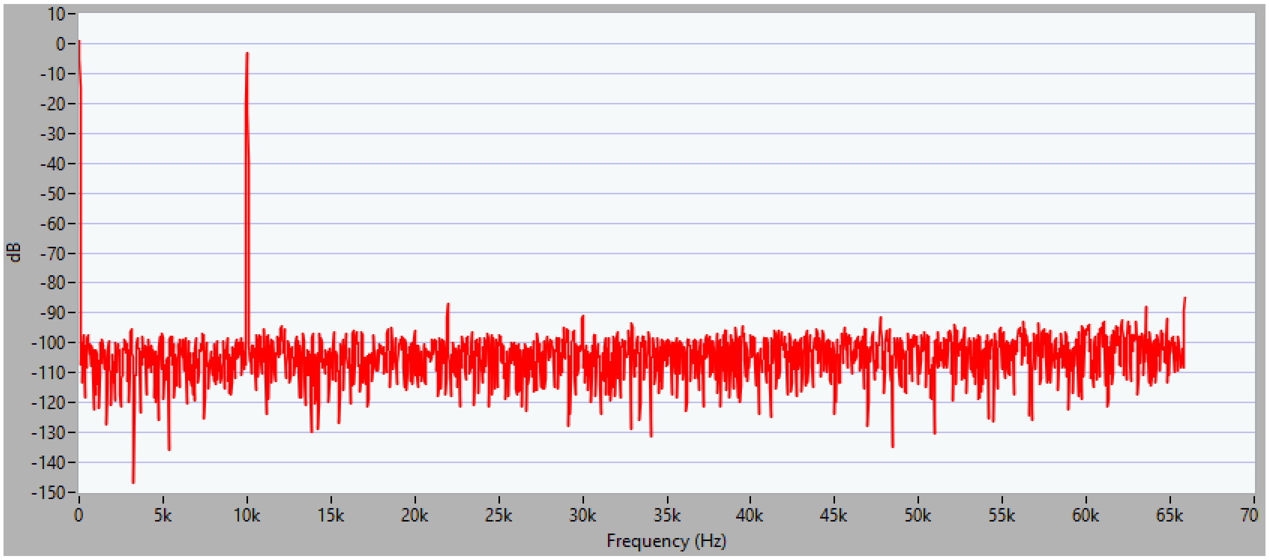

Figure 5-1. Baseline Sampling FFT Plot

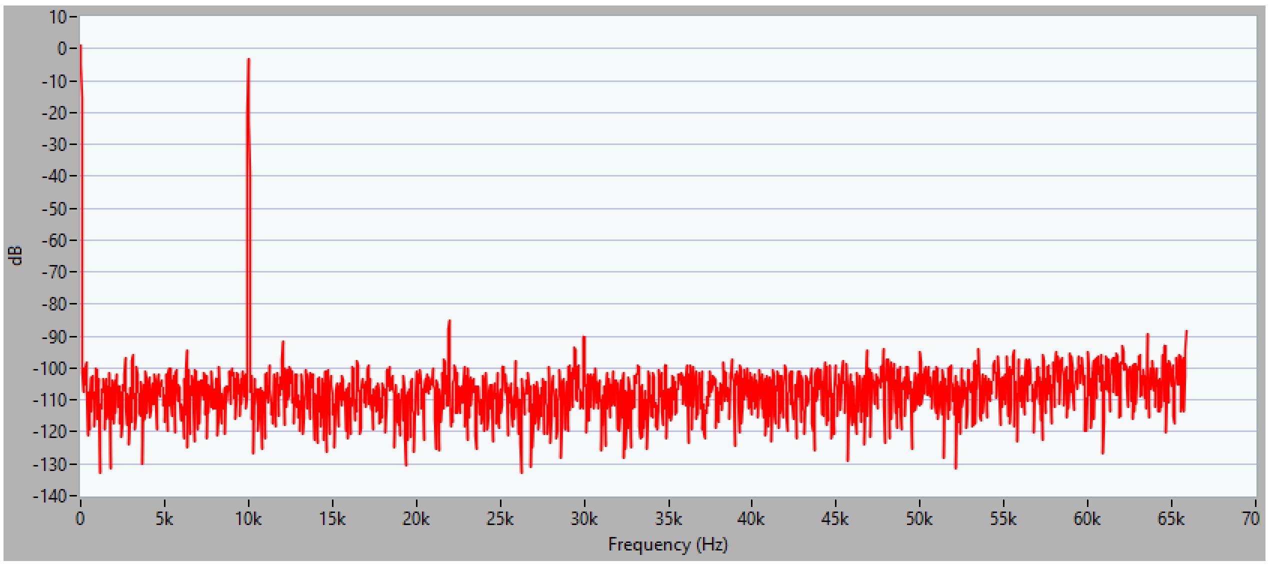

Figure 5-2. 2X Oversampling FFT Plot

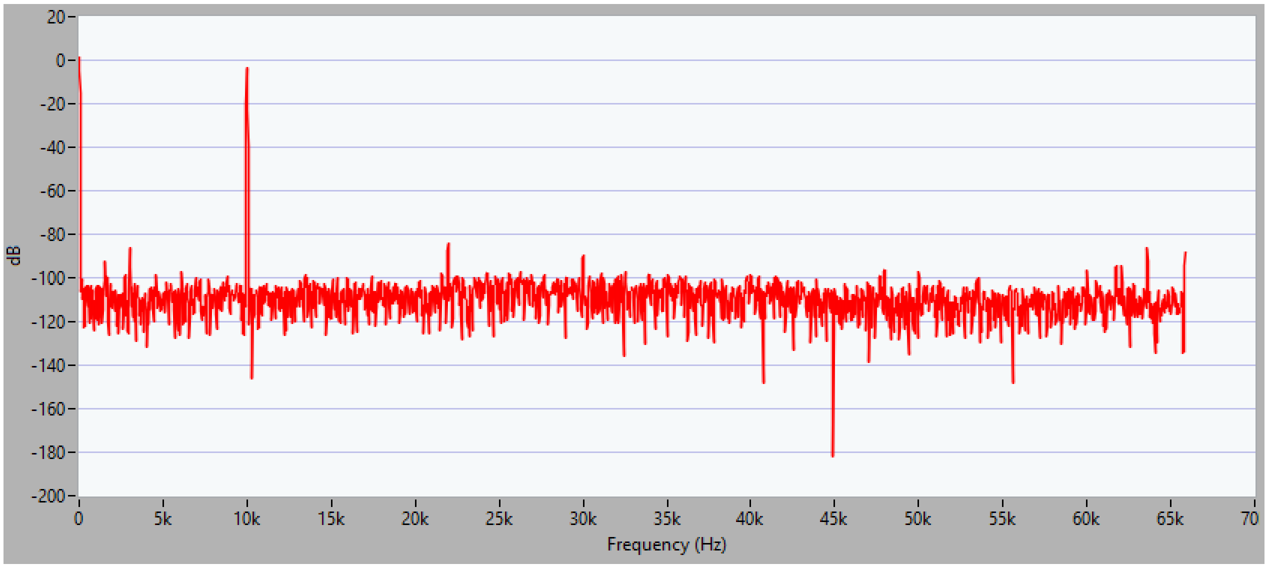

Figure 5-3. 4X Oversampling FFT Plot

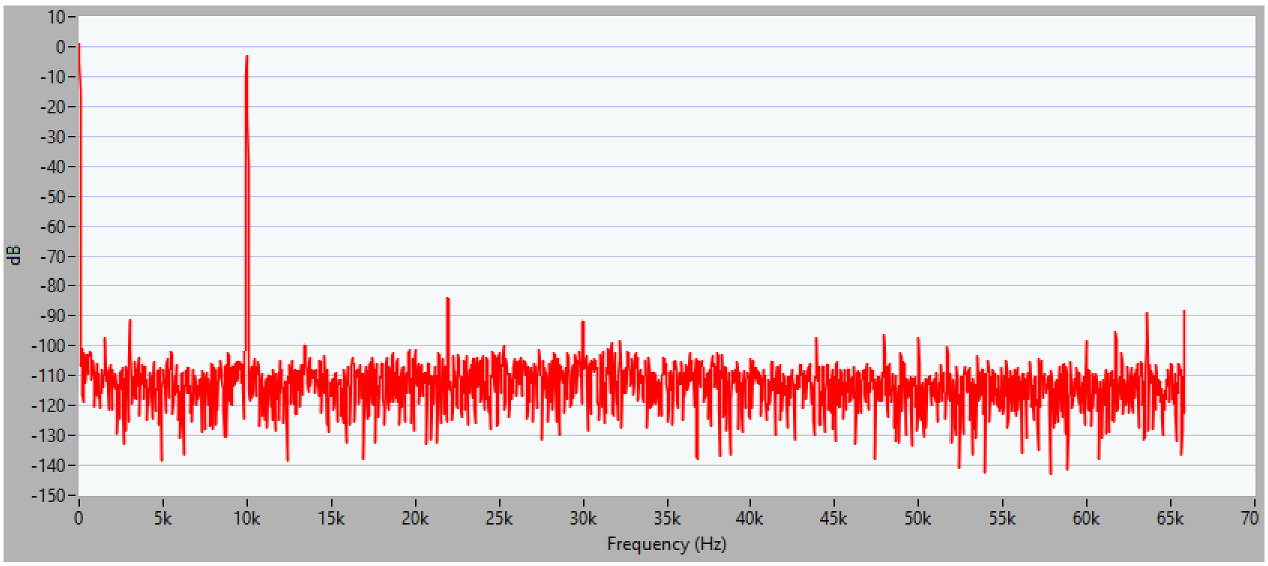

Figure 5-4. 8X Oversampling FFT Plot

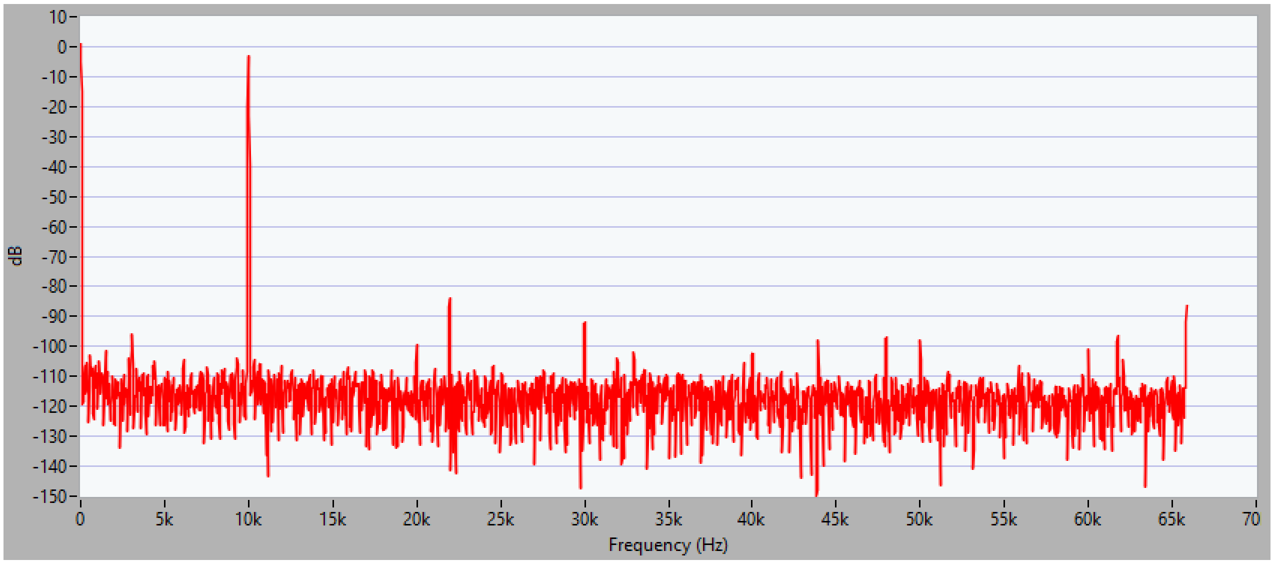

Figure 5-5. 16X Oversampling FFT Plot

6 Summary

This application note covered the effects of oversampling with an ADC using software methods, and how oversampling can increase the effective number of bits (ENOB). The theoretical ENOB increases by 0.5 every time the oversampling factor is doubled. The observed ENOB increase is slightly lower due to system noise limitations. This can be further improved by adding a buffer to the ADC input and using an external VREF source. Oversampling does decrease the rate at which individual samples are read, since the oversampling takes up additional time within the control loop. There is also an increase in time to collect results from the ADC, depending on the oversampling factor.

7 References

- Texas Instruments: *General Oversampling of MSP ADCs for Higher Resolution*
- Texas Instruments: *ADC Input Evaluation for C2000™ MCUs*
- Texas Instruments: *Hardware Design Guide for F2800x C2000™ Real-Time MCU Series*
- C2000 C28x Optimization Guide
- Texas Instruments: *TMS320F28003x Real-Time Microcontrollers Technical Reference Manual*

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265