Application Note

ADC Oversampling

Susmitha Bumadi

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 were tested on the TMDSCNCD28P65X device for Software oversampling and Hardware oversampling for the device initialization.

Table of Contents

1 Introduction....2
2 Theory....2
3 Software Oversampling....3
4 Hardware Oversampling....7
5 Results....12
6 Summary....18
7 References....18
8 Revision History....18

List of Figures

Figure 3-1. SOC Flow Diagram for Software Oversampling....5
Figure 3-2. Timings for Sampling a Signal in Software Oversampling....6
Figure 3-3. Incorrect Timings for Sampling a Signal in Software Oversampling....6
Figure 4-1. SOC Flow Diagram for Hardware Oversampling....9
Figure 4-2. Timings for Sampling a Signal in Hardware Oversampling....10
Figure 4-3. Incorrect Timings for Sampling a Signal in Hardware Oversampling....10
Figure 4-4. Overall Hardware Setup....11
Figure 4-5. Wiring Setup....11
Figure 5-1. Baseline Software Sampling FFT Plot....12
Figure 5-2. 2 × Software Oversampling FFT Plot....13
Figure 5-3. 4 × Software Oversampling FFT Plot....13
Figure 5-4. 8 × Software Oversampling FFT Plot....14
Figure 5-5. 16 × Software Oversampling FFT Plot....14
Figure 5-6. Baseline Hardware Sampling FFT Plot....15
Figure 5-7. 2 × Hardware Oversampling FFT Plot....16
Figure 5-8. 4 × Hardware Oversampling FFT Plot....16
Figure 5-9. 8 × Hardware Oversampling FFT Plot....17
Figure 5-10. 16 × Hardware Oversampling FFT Plot....17

List of Tables

Table 2-1. Relationship Between Oversampling Factor, SNR, and Extra Bits of Resolution....3
Table 3-1. Oversampling Time....7
Table 5-1. ADC Software Oversampling Results....12
Table 5-2. ADC Hardware Oversampling Results....15

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 TMDSCNCD28P65X device for Software oversampling and Hardware oversampling, with a 12-bit ADC.

2 Theory

The goal of oversampling is to increase ENOB by reducing the noise observed in the signal. Oversampling performs multiple conversions on the same input signal and accumulates the digital values to attain an ENOB higher than the inherent ENOB of the ADC. The precision of the result increases, depending on how much oversampling takes place. This accuracy can be demonstrated by measuring a varying input signal to determine the major frequency of the signal. 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 16 × 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 4.

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. See *General Oversampling of MSP ADCs for Higher Resolution* for more data on the theory behind the numbers in this table.

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 16 × is tested using a 5kHz 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 frequency of the input signal. The more diminished these are relative to the fundamental signal amplitude, the higher the resulting ENOB.

3 Software Oversampling

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, set up the ADC 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.

The following code 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 8 × 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 are stored, or else the ADC can continue to convert the analog signal. Figure 3-1 shows the basic flow of using an ePWM to trigger the burst conversion for oversampling.

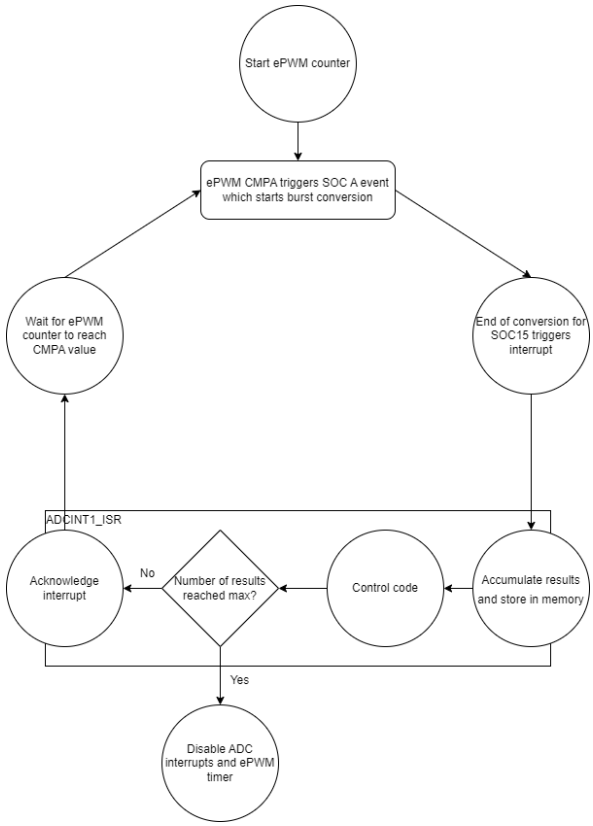

Figure 3-1. SOC Flow Diagram for Software Oversampling

Depending on the control loop for the specific application, more time can be required than what the maximum 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 3-2 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 3-3 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 requires extending the ePWM time base to move the trigger further.

Table 3-1 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. Use the following function 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)
```

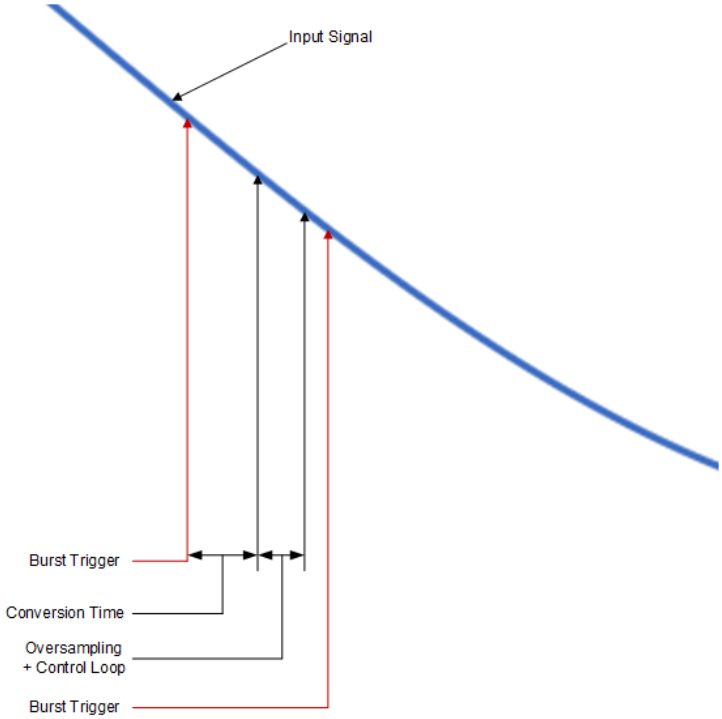

Figure 3-2. Timings for Sampling a Signal in Software Oversampling

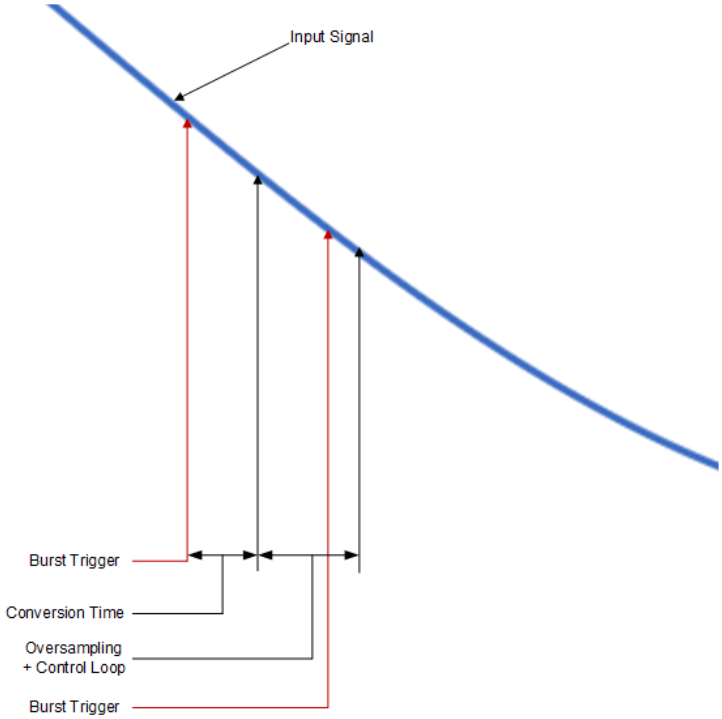

Figure 3-3. Incorrect Timings for Sampling a Signal in Software Oversampling

Table 3-1. Oversampling Time

Oversampling Factor	Oversample Time (clock cycles)
1 ×	9
2 ×	52
4 ×	127
8 ×	272
16 ×	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 16 × oversampling with ISR uses 851 cycles. If the sine wave is 5kHz, based on the Nyquist theorem, the minimum sampling rate is at least 10kSPS (kilo-samples per second). The table in the *ADC Timing Diagrams* chapter of the TMS320F28P65X 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 200MHz by default for the TMDSCNCD28P65X. For an ADC clock of 57MHz on the TMDSCNCD28P65X, the clock prescaler divides SYSCLK by 3.5, and the value for t_{LAT} is 39 SYSCLK cycles. F_{Sample} is the rate in samples per second required for a specific application, which is 10kSPS 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 19960. 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 is determined by the input network. See the *Choosing an Acquisition Window Duration* section within the *ADC* chapter in *TMS320F28P65x Real-Time Microcontrollers Technical Reference Manual* for more information on calculating ACQPS values.

4 Hardware Oversampling

For the purpose of testing ADC oversampling, a TMDSCNCD28P65X controlCARD was used to convert the input sine wave into digital values. See the *TMDSCNCD28P65X controlCARD Information Guide* to configure the reference voltage VREF and JTAG for the controlCARD. To keep the setup simple while reducing possible sources of error, the internal 1.65V reference was used. If the external VREF is used, extra steps must be taken. See the *Voltage Reference* chapter in the *ADC* chapter of the *TMS320F28P650DK Real-Time Microcontrollers Technical Reference Manual* for more information regarding VREF.

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. See *ADC Input Evaluation for C2000™ MCUs* for details on input conditioning. See the *Hardware Design Guide for F2800x C2000™ Real-Time MCU Series* application note for PCB layout design recommendations.

The example used for Hardware oversampling has the main code that configures the ADC, ePWM, and other peripherals.

For Hardware oversampling, the SOC is configured with trigger repeater which generates a number of repeat pulses as desired. In this example, SOC is triggered with EPWM pulse and results are accumulated in accumulator without missing a value when oversampling. Oversampling interrupt is set up to trigger once

the repeater count is reached. The interrupt runs the corresponding ISR, which stores the ADC result and accumulate the results if oversampling is enabled.

Oversampling Example With Trigger Repeater shows the example code for oversampling with a trigger repeater.

Oversampling Example With Trigger Repeater

```
// adcA1ISR - ADC A Interrupt 1 ISR
//
__interrupt void adcA1ISR(void)
{
    //
    // Store the results for A0
    //
    // myADC0Result = ADC_readResult(ADCARESULT_BASE, ADC_SOC_NUMBER0);

    //
    // Store the 4 oversampled A0 results together
    //

    lv_results[nloops++] = (uint16_t)ADC_readPPBSum(ADCARESULT_BASE, ADC_PPB_NUMBER1);


    // Clear the interrupt flag
    //
    ADC_clearInterruptStatus(myADC0_BASE, ADC_INT_NUMBER1);

    //
    // Check if overflow has occurred
    //
    if(true == ADC_getInterruptOverflowStatus(myADC0_BASE, ADC_INT_NUMBER1))
    {
        ADC_clearInterruptOverflowStatus(myADC0_BASE, ADC_INT_NUMBER1);
        ADC_clearInterruptStatus(myADC0_BASE, ADC_INT_NUMBER1);
    }

    //
    // Acknowledge the interrupt
    //
    Interrupt_clearACKGroup(INT_myADC0_1_INTERRUPT_ACK_GROUP);
    if(nloops >= numBins)
           {
               //
               // Disable ADC interrupt
               //
               ADC_disableInterrupt(myADC0_BASE, ADC_INT_NUMBER1);
               ESTOP0;
           }

}
```

The appropriate interrupts can be disabled once the intended number of results are stored, or else the ADC can continue to convert the analog signal. Figure 4-1 shows the basic flow of using an ePWM to trigger the repeater for oversampling.

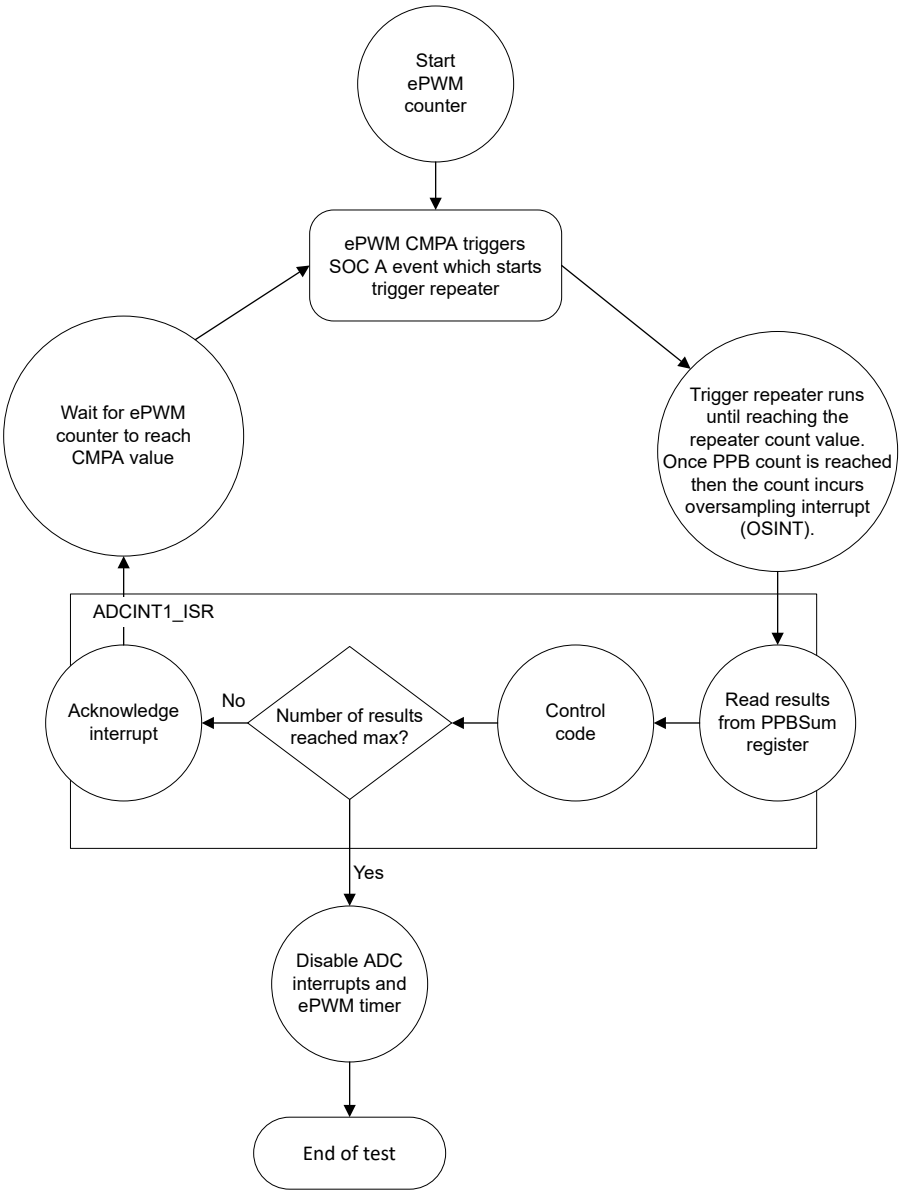

Figure 4-1. SOC Flow Diagram for Hardware Oversampling

Increasing the ePWM time base to allow a longer conversion time is a good design practice, 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 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 ePWM period, data is missed. The solution for this is to extend the period, which in this example requires extending the ePWM time base to move the trigger further.

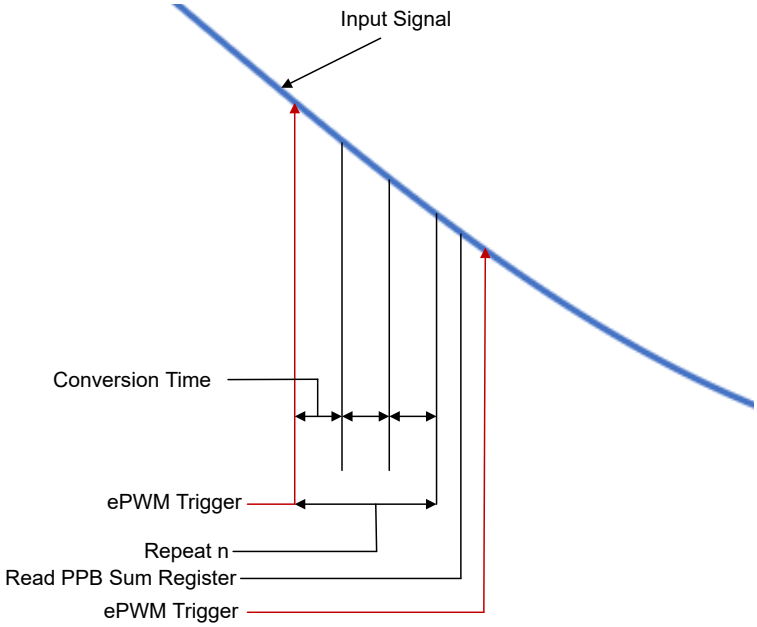

Figure 4-2. Timings for Sampling a Signal in Hardware Oversampling

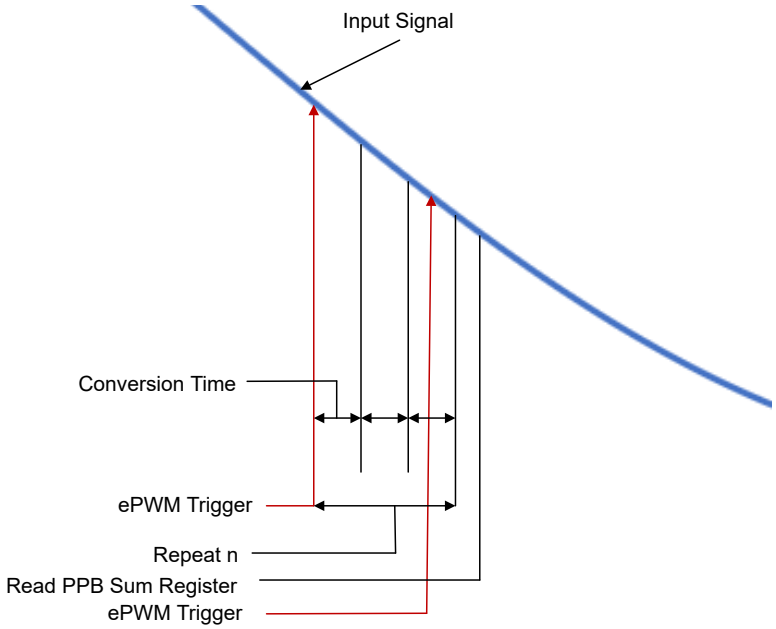

Figure 4-3. Incorrect Timings for Sampling a Signal in Hardware Oversampling

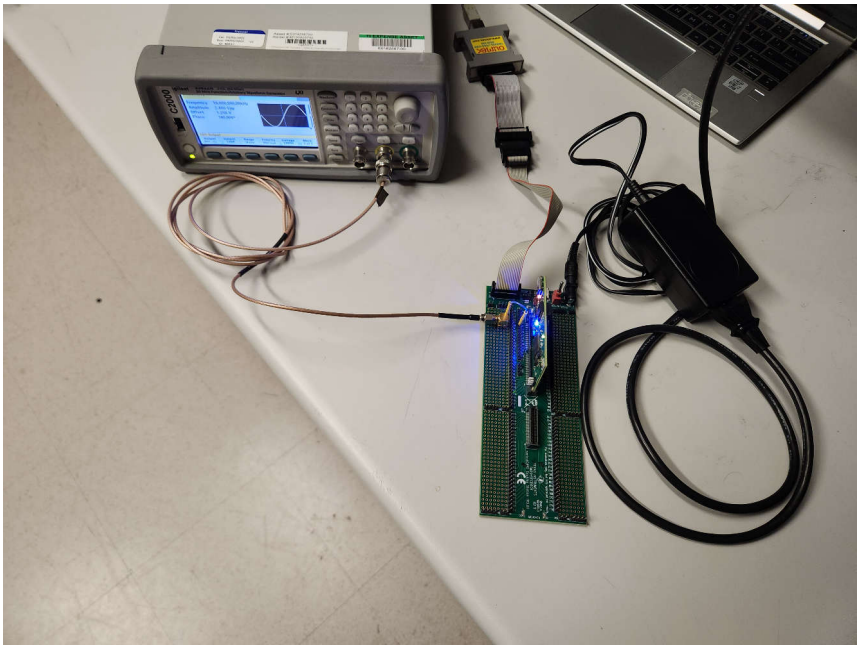

Figure 4-4. Overall Hardware Setup

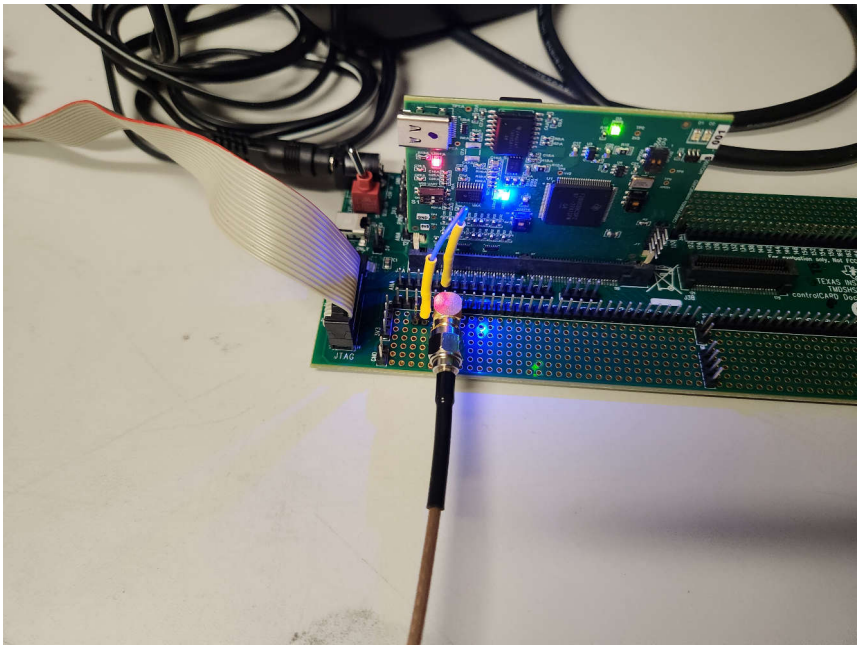

Figure 4-5. Wiring Setup

5 Results

The results of ADC Software 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 3-1.

Table 5-1. ADC Software Oversampling Results

Oversampling Factor	ENOB	THD	SNR	FFT Result
1 ×	11.17	–79.17	69.02	Figure 5-1
2 ×	11.58	–78.65	71.52	Figure 5-2
4 ×	11.96	–79.19	73.76	Figure 5-3
8 ×	12.34	–78.87	76.04	Figure 5-4
16 ×	12.77	–79.78	78.65	Figure 5-5

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

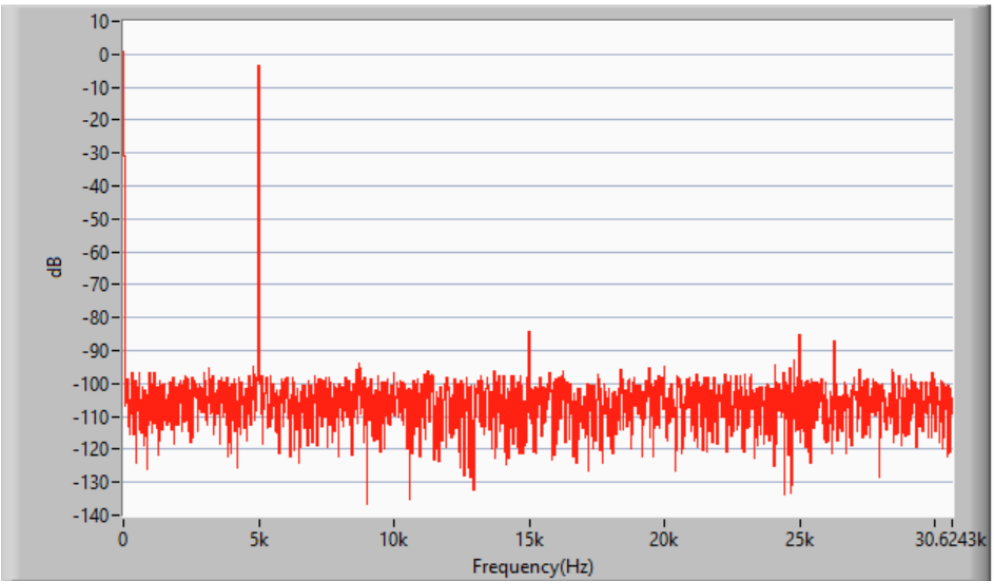

Figure 5-1. Baseline Software Sampling FFT Plot

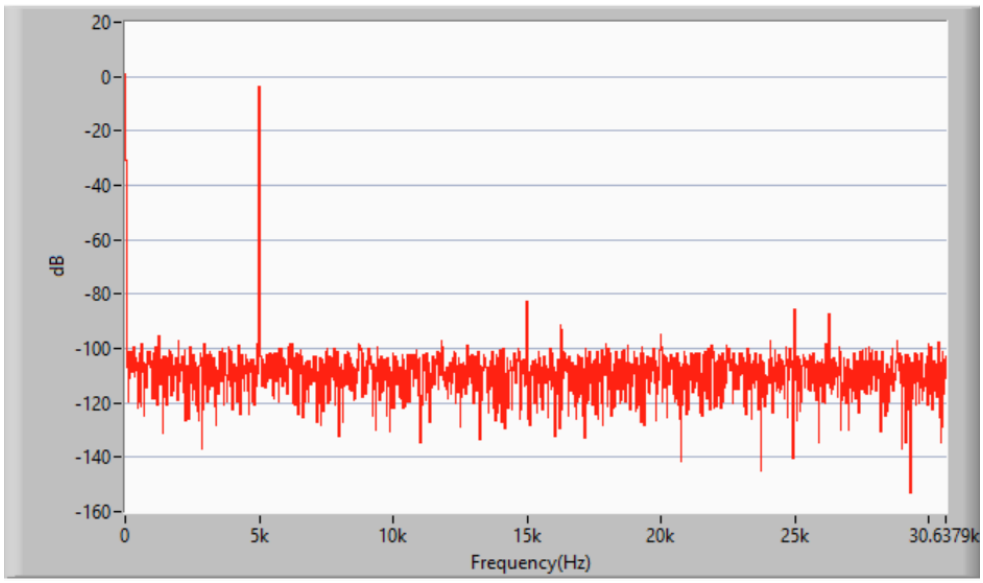

Figure 5-2. 2 × Software Oversampling FFT Plot

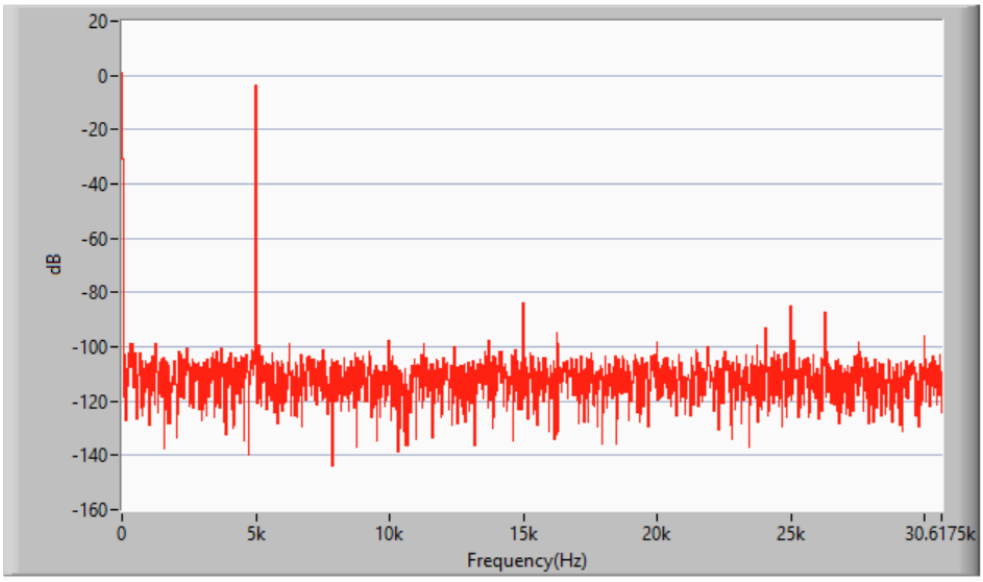

Figure 5-3. 4 × Software Oversampling FFT Plot

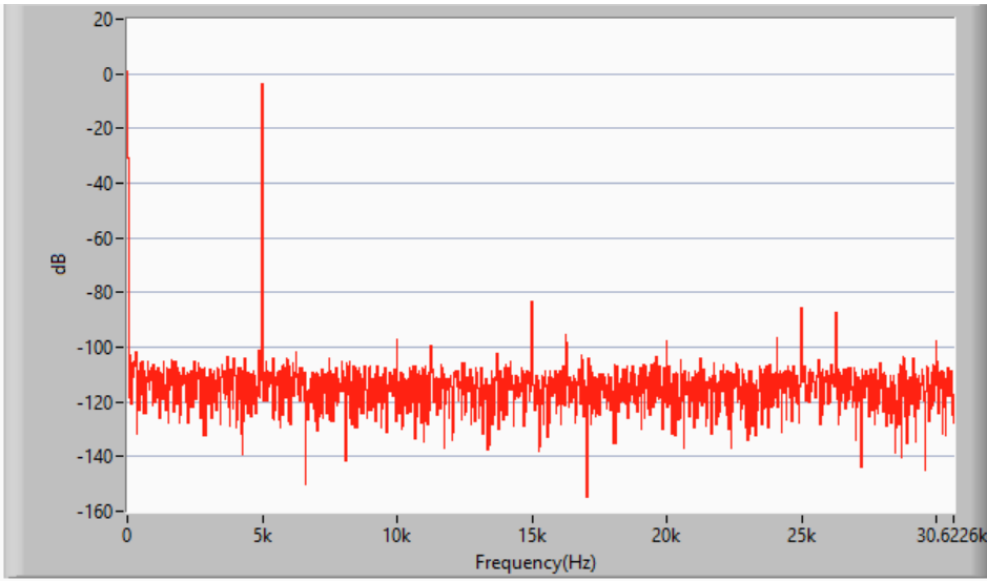

Figure 5-4. 8 × Software Oversampling FFT Plot

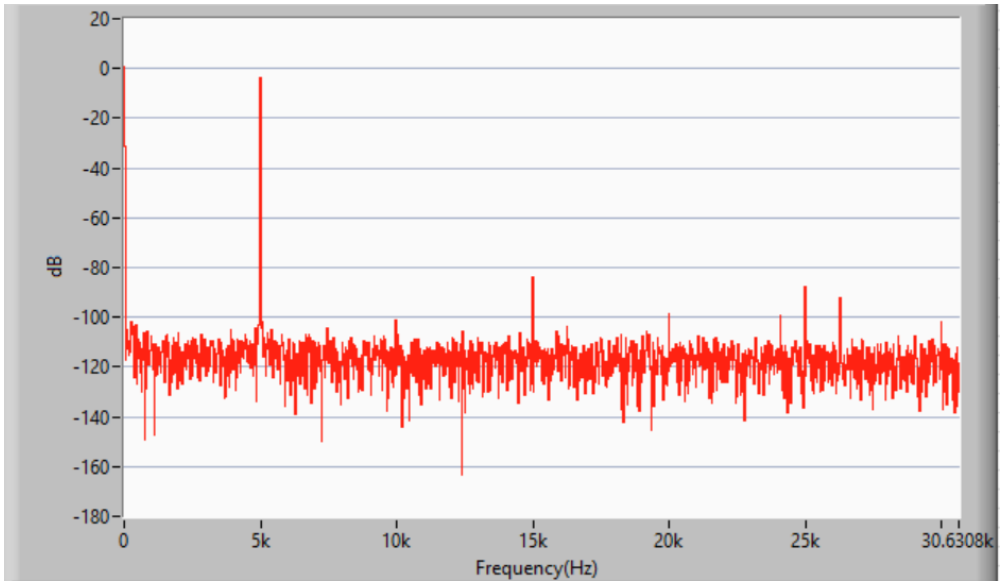

Figure 5-5. 16 × Software Oversampling FFT Plot

Table 5-2 shows the results of ADC hardware oversampling.

Table 5-2. ADC Hardware Oversampling Results

Oversampling Factor	ENOB	THD	SNR	FFT Result
1 ×	11.21	–86.79	69.23	Figure 5-6
2 ×	11.60	–87.52	71.58	Figure 5-7
4 ×	12.12	–89.99	74.72	Figure 5-8
8 ×	12.49	–89.37	76.97	Figure 5-9
16 ×	12.85	–89.50	79.10	Figure 5-10

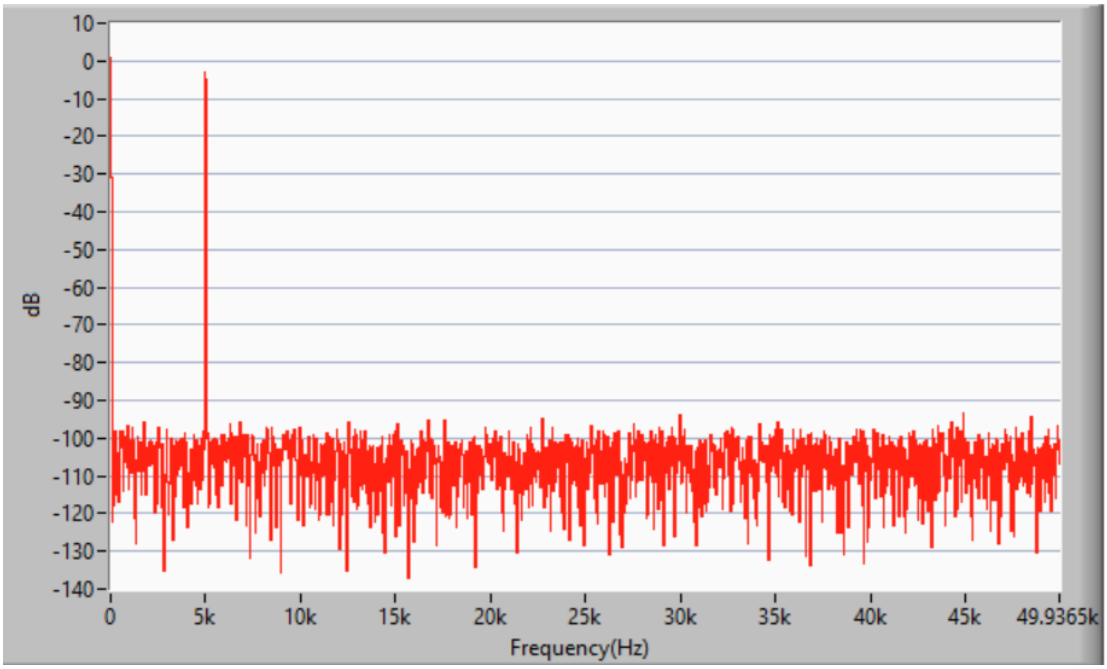

Figure 5-6. Baseline Hardware Sampling FFT Plot

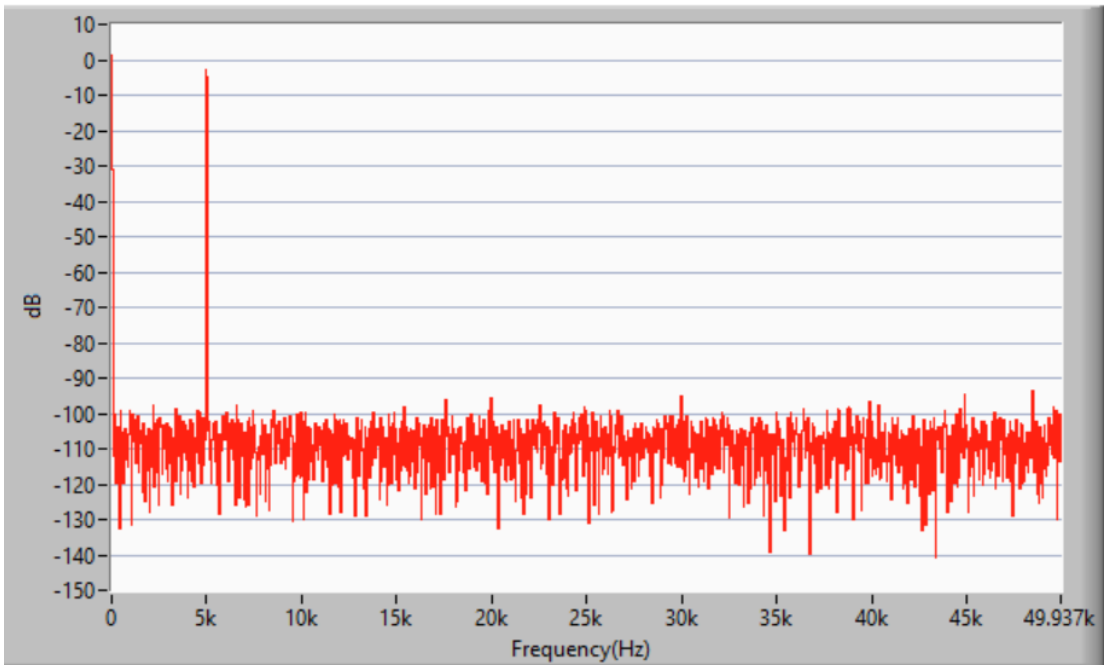

Figure 5-7. 2 × Hardware Oversampling FFT Plot

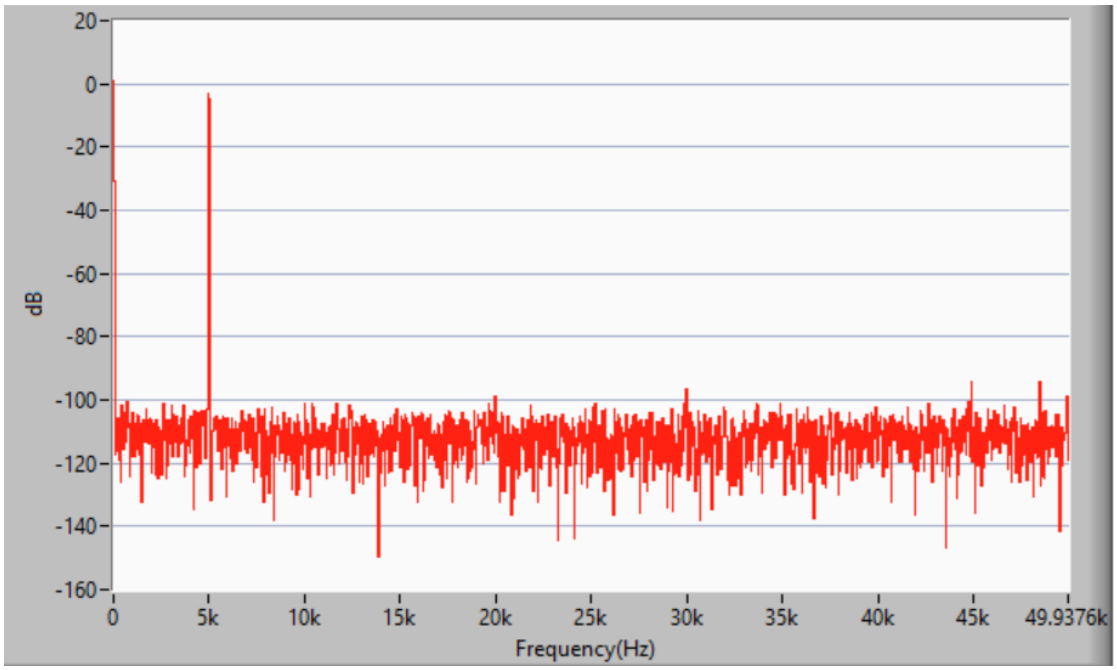

Figure 5-8. 4 × Hardware Oversampling FFT Plot

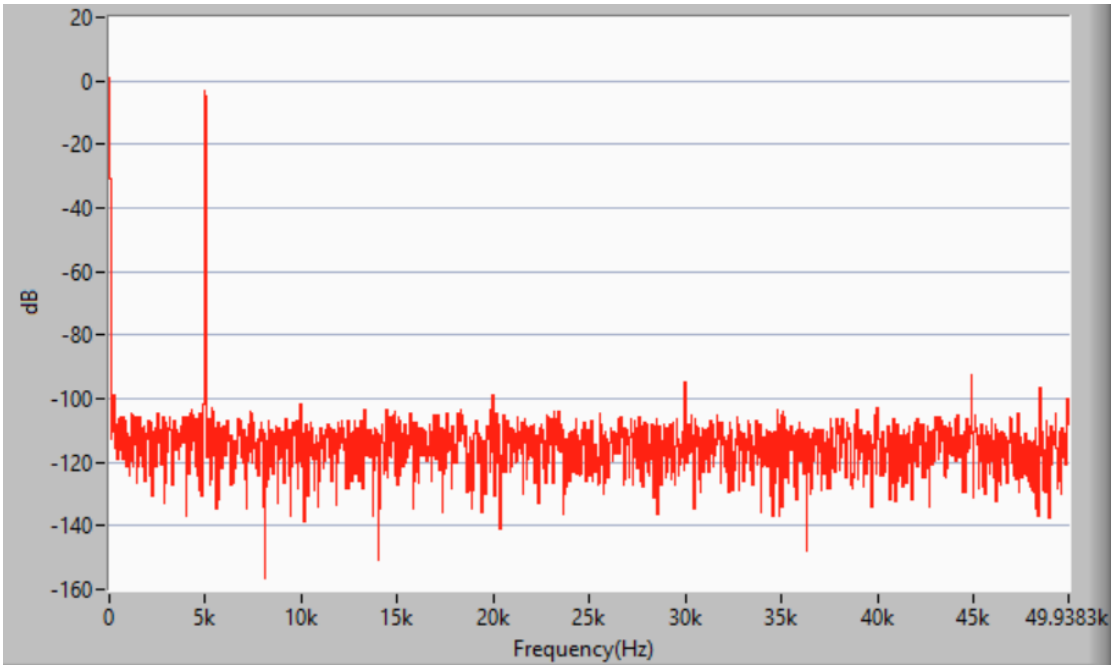

Figure 5-9. 8 × Hardware Oversampling FFT Plot

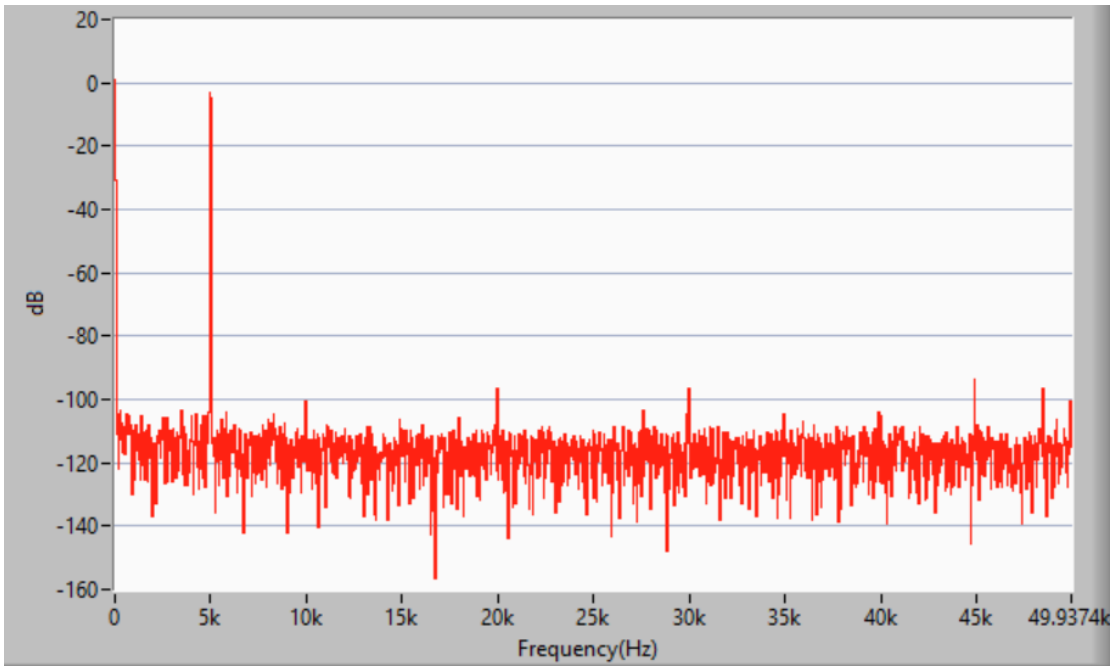

Figure 5-10. 16 × Hardware Oversampling FFT Plot

6 Summary

This application note covered the effects of oversampling with an ADC using software and hardware 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*
- *ADC Input Circuit Evaluation for C2000 MCUs (TINA-TI) Application Report*
- C2000 Academy - ADC
- PSpice for TI design and simulation tool
- *Real-Time Control Reference Guide*
  - Refer to the ADC section
- TI Precision Labs - ADCs
- TI Precision Labs: Driving the reference input on a SAR ADC (Video)
- TI Precision Labs: Introduction to analog-to-digital converters (ADCs) (Video)
- TI Precision Labs: SAR ADC input driver design (Video)
- TI e2e: Connecting VDDA to VREFHI
- TI e2e: Topologies for ADC Input Protection
- TI e2e: Why does the ADC Input Voltage drop with sampling?
  - Sampling a high impedance voltage divider with ADC
- *Understanding Data Converters Application Report*

8 Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Revision * (March 2023) to Revision A (August 2024)	Page
• Added flow chart, code example and additional information to Section 4.	7
• Changed the document to show the hardware oversampling data	12

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

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