Aliasing and the Nyquist Theorem in Digital Signal Processing

Juan Camilo Garcia Perez. Department of Electronic Engineering, Santo Tómas University, Bogotá, Colombia. (e-mail: juancgarciap@usantotomas.edu.co)

Abstract: Aliasing is a phenomenon that occurs in digital signal processing when signals are sampled at a frequency lower than twice the highest frequency component in the signal. This results in an inaccurate representation of the original signal, which can lead to errors and distortion. The Nyquist theorem is a fundamental concept in digital signal processing that states that in order to accurately capture a signal, the sampling rate must be at least twice the highest frequency component in the signal. In this paper, we explore aliasing and the Nyquist theorem in the context of digital signal processing using a Digital Signal Processor (DSP) and analog-to-digital converter (ADC) and digital-to-analog converter (DAC) components. We present an implementation of the Nyquist theorem in the code and demonstrate the effects of aliasing in signal processing.

Introduction

Digital signal processing (DSP) is a key technology used in a wide range of applications, including audio processing, image processing, and telecommunications. One of the fundamental concepts in DSP is sampling, which is the process of converting continuous analog signals into discrete digital signals. This process involves sampling the analog signal at regular intervals and converting each sample into a digital value using an analog-to-digital converter (ADC). In order to accurately capture the original analog signal, the sampling frequency must be high enough to prevent aliasing.

Objectives

Main Objective

To determine the sampling rate required to avoid aliasing in the signal acquisition system, in accordance with the Nyquist theorem.

Specific Objectives

1. To explain the concept of aliasing and how it occurs in digital signal processing.
2. To introduce the Nyquist theorem and its importance in digital signal processing.
3. To demonstrate the effects of aliasing in signal processing using a DSP and ADC.

Procedure

To start this sampling process, It's taken the following equention:

$$x(n) = A\,\mathrm{Sin}\left(2\,\pi\left(\frac{f_a}{f_s}\right)n\right)$$

(1)

Where f_a is the Analog frequency, f_s is the sampling frequency, $A$is the amplitude and finally n is the sampling vector. In this scenario, we are looking for the following relation:

$$\frac{f_a}{f_s} = \frac{1}{N}$$

(2)

And N is the critical period, for this challenge $N = [2\ ,\ 10,\ 50,\ 512]$ and $f_a = 100\mathrm{Hz}$, using those values we obtaing the sampling values for each scenario.

$$F_{s_1} = 200\mathrm{Hz}$$
$$F_{s_2} = 1000\mathrm{Hz}$$
$$F_{s_3} = 5000\mathrm{Hz}$$
$$F_{s_4} = 51200\mathrm{Hz}$$

There is a relation between f_a and f_s , Nyquist said:

$$f_s \geq 2f_a$$

(3)

If you have the objective of rebuild a signal, the best way to achive it is using this theorem because if you don't follow it you will rebuild the same wave form but not the same frequency, In other words you will have the same signal but with two different frequencies. Therefore, in order to cause alising, we are using this relation:

$$\frac{f'_a}{f_s} = \frac{f_a + f_s}{f_s}$$

(4)

The code implements the Nyquist theorem by setting the ADC and DAC sampling rates to values that satisfy the Nyquist criterion. The ADC samples the analog signal at a high enough frequency to prevent aliasing, and the DAC converts the digital signal back into an analog signal. The code also includes an interrupt service routine that stores the ADC results in an array and updates the DAC output with the latest sample.

1. Configuring the ADC and DAC peripherals:

The ADC and DAC peripherals of the DSP are configured using the driverlib and device header files. The ADC is configured for single-ended mode with 12-bit resolution

```
ADC_setMode(ADCA_BASE, ADC_RESOLUTION_12BIT, ADC_MODE_SINGLE_ENDED);
```

While the DAC is configured for system clock load mode with ADC reference voltage.

2.   Setting the timebase period: The timebase period of the ePWM module is set based on the required sampling rate. The timebase period is calculated based on the sample rate divider and the maximum timebase period supported by the ePWM module.

```
while(1)
    {
        // If the sample rate divider has changed
        if (Old_N != N)
        {
            // Update the sample rate divider and EPWM time-base period based on the new value
            Old_N = N;
            switch(N)
            {
                case 2:
                    TBPRD = 62500;
                    break;
                case 10:
                    TBPRD = 12500;
                    break;
                case 50:
                    TBPRD = 2500;
                    break;
                default:
                    N = 512;
                    TBPRD = 244;
                    break;
            }
            EPWM_setTimeBasePeriod(EPWM1_BASE, TBPRD);
        }
    }
```

4.   Triggering the ADC conversion: The ADC conversion is triggered using the ePWM event. The ADC results are stored in a buffer and the DAC shadow value is updated with the latest result.

```
    __interrupt void adcA1ISR(void)
{
    adcAResults[index] = ADC_readResult(ADCARESULT_BASE, ADC_SOC_NUMBER0);

    DAC_setShadowValue(DACA_BASE, adcAResults[index++]);

    if(RESULTS_BUFFER_SIZE <= index)
    {
        index = 0;
    }

    ADC_clearInterruptStatus(ADCA_BASE, ADC_INT_NUMBER1);

    if(true == ADC_getInterruptOverflowStatus(ADCA_BASE, ADC_INT_NUMBER1))
    {
        ADC_clearInterruptOverflowStatus(ADCA_BASE, ADC_INT_NUMBER1);
        ADC_clearInterruptStatus(ADCA_BASE, ADC_INT_NUMBER1);
    }

    Interrupt_clearACKGroup(INTERRUPT_ACK_GROUP1);
}
```

The function performs the following operations:

1. It reads the result of the ADC conversion from the ADC result register and stores it in an array called adcAResults at the index position determined by the variable index.
2. It sets the output of the DAC to the value just read from the ADC result register using the DAC_setShadowValue() function.
3. It checks if the index has reached the maximum size of the array. If so, it resets the index to zero to overwrite the oldest value in the buffer.
4. It clears the ADC interrupt flag by calling ADC_clearInterruptStatus().
5. It checks for ADC interrupt overflow and clears any flags associated with it.
6. Finally, it clears the interrupt flag by calling Interrupt_clearACKGroup().

Results

The aliasing is demonstrated by presenting the results of the sampling process. The results include the sampled signal waveform and the ADC and DAC output waveforms.

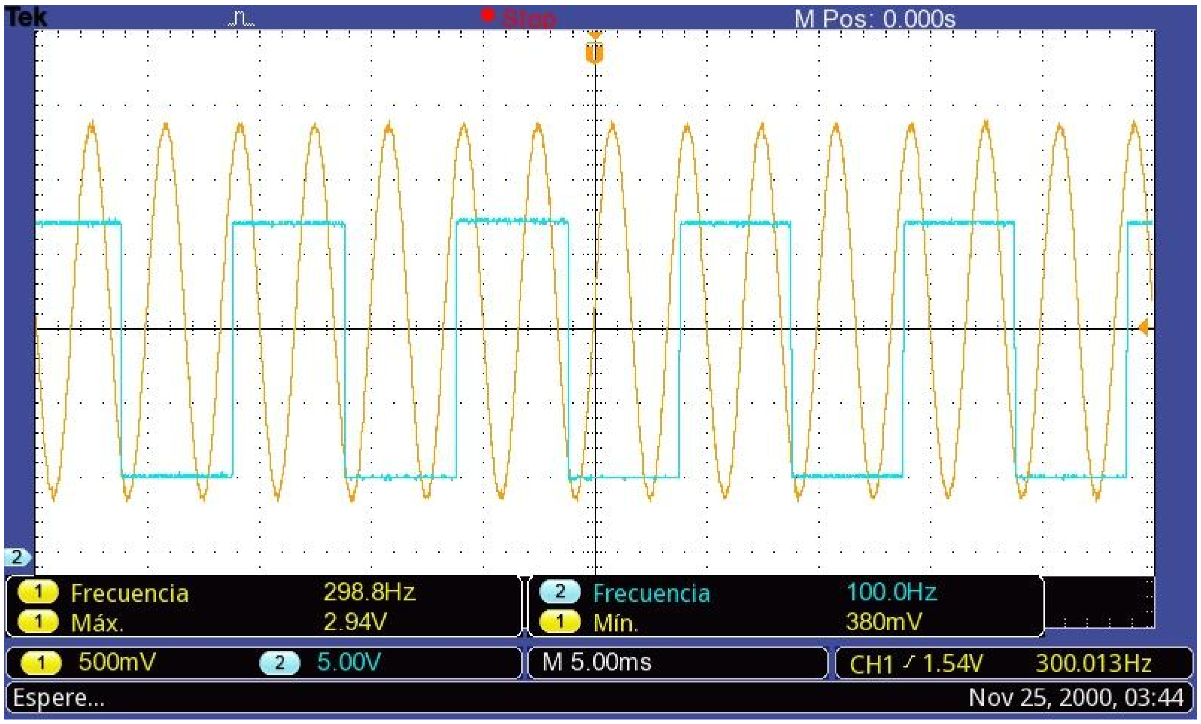

Fig. 1. Aliasing when $N = 2$ and $f'_a = 300$.

N=2

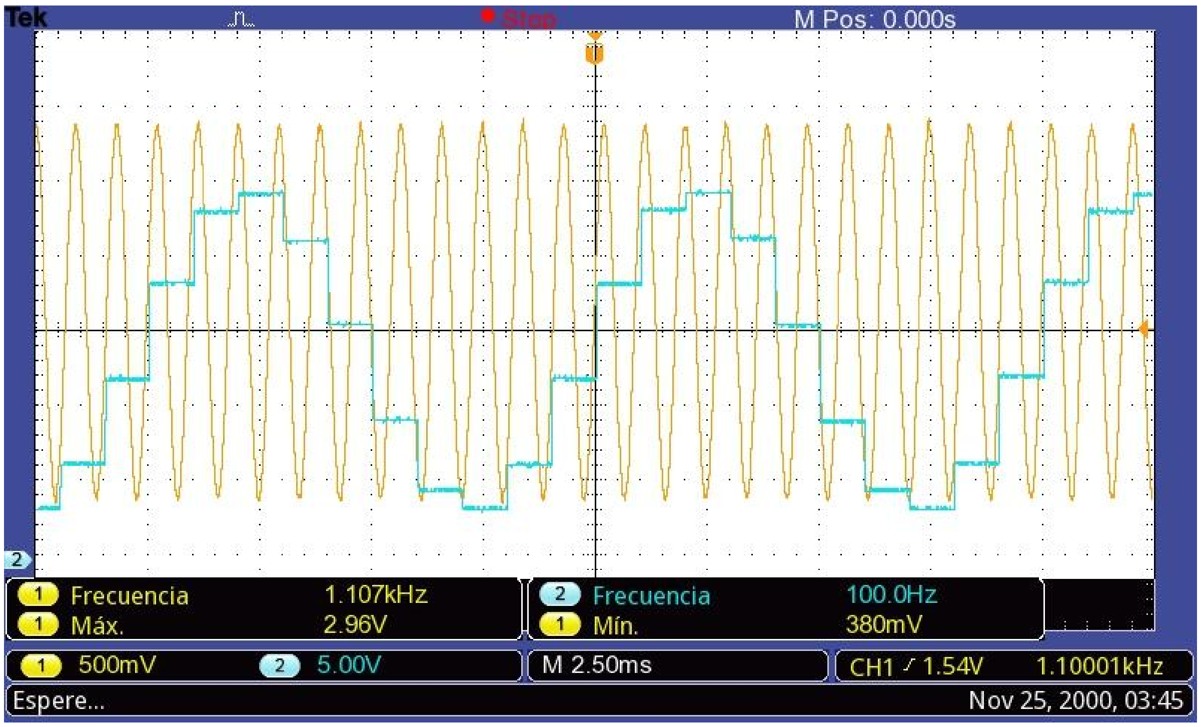

Fig. 2. Aliasing when $N = 10$ and $f'_a = 1100$.

N=10

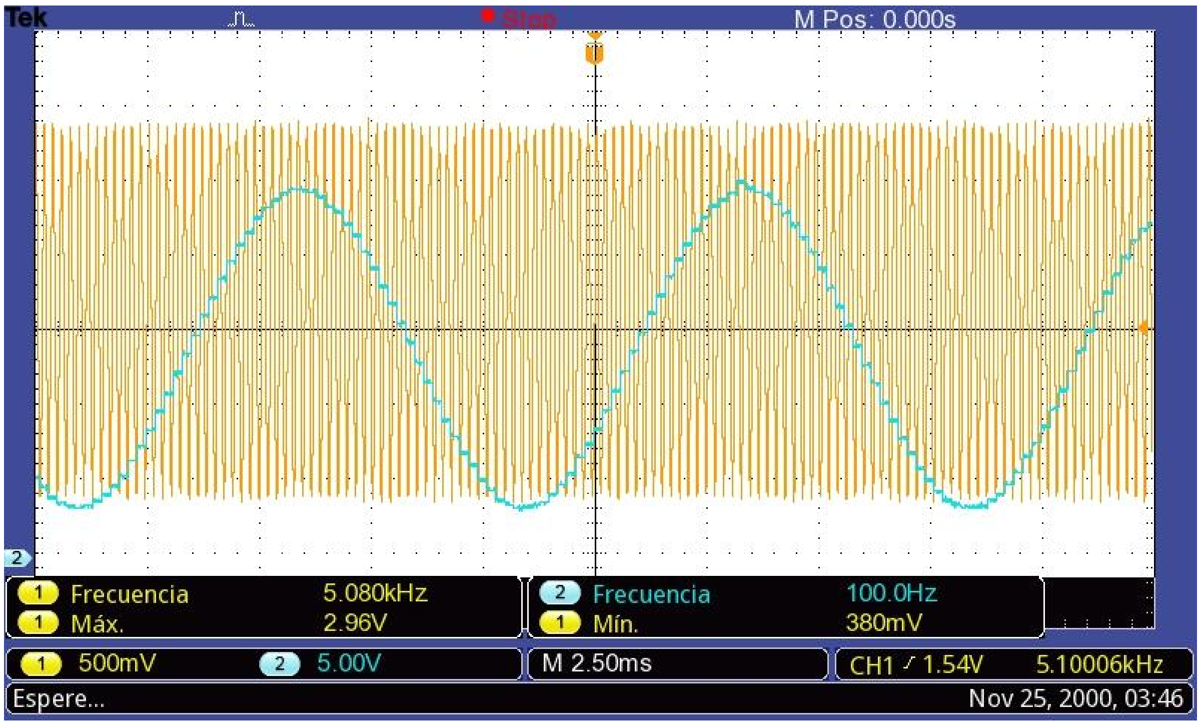

Fig. 3. Aliasing when $N = 50$ and $f'_a = 5100$.

N=50

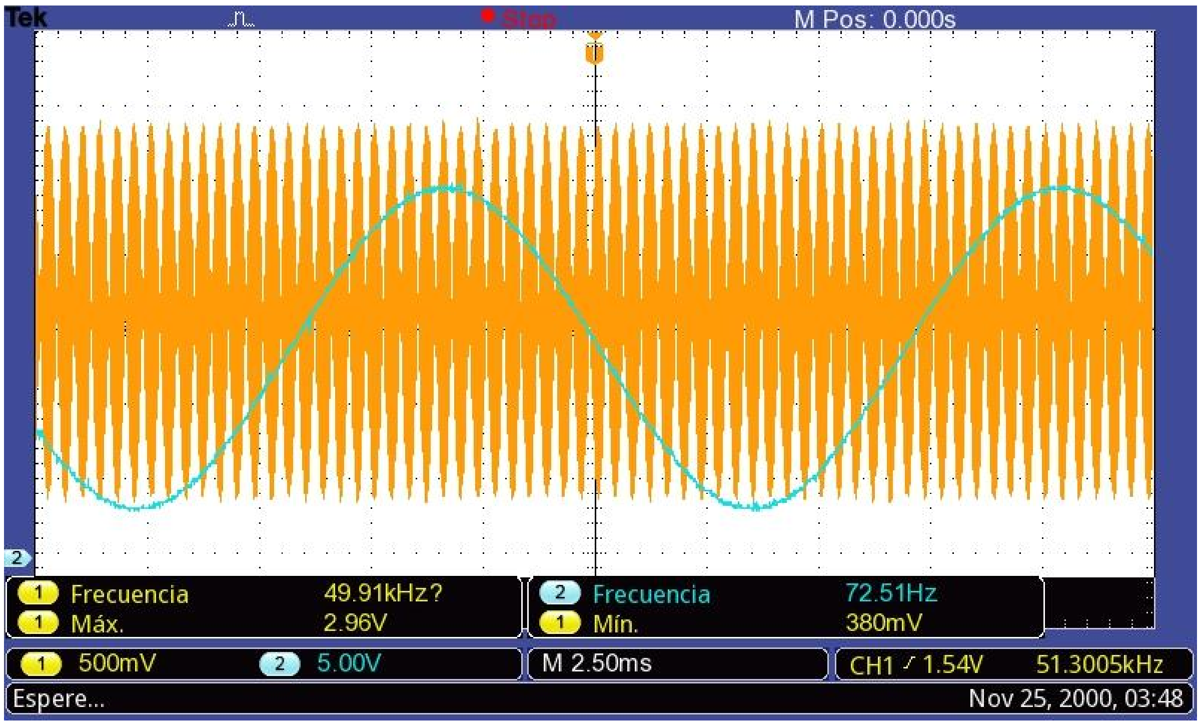

Fig. 4. Aliasing when $N = 512$ and $f'_a = 51300$.

In the figs. shown before we are able to notice what happens when the Nyquist theorem is not used, the sampled signal has a different frequency value than the analog signal, each value of f'_a was obtained using eq. (4).

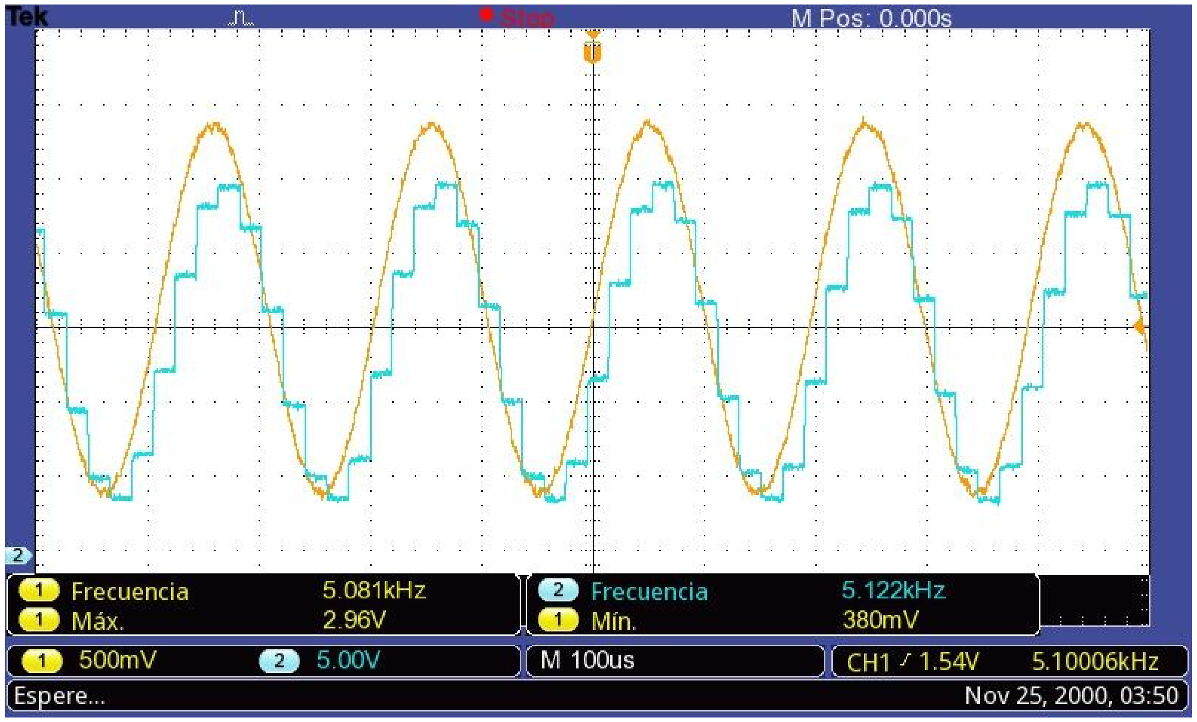

Fig. 5. sampled signal when $N = 512$ and $f'_a = 5100$.

But we can observe that if the value of N increments using the last f_a, we almost obtained a proper sampled signal.

Conclusions

We have demonstrated how aliasing can lead to errors and distortion in signal processing and how the Nyquist theorem can be used to prevent these issues.

By implementing the Nyquist theorem in the code, we have shown how a DSP and ADC can be used to accurately capture and process analog signals.

The Nyquist theorem is a fundamental concept in digital signal processing and is essential for ensuring the accuracy of signal processing applications.

Aliasing can be resume as a situation that happens when a signal is trying to be sampled but it presents a different frequency value between f_a and f_s.

References:

[1] Nyquist, H. (1928). Certain topics in telegraph transmission theory. Transactions of the American Institute of Electrical Engineers, 47(2), 617-644.

[2] Proakis, J. G., & Manolakis, D. G. (2006). Digital signal processing: principles, algorithms, and applications. Pearson Education.

[3] TMS320F2837xD Dual-Core Microcontrollers Technical Reference Manual.

[4] F2837xD Firmware Development Package. USER’S GUIDE.

[5] DSP2839D Plans. Texas Instruments.