Signal Sampling Using ADC and DAC in a TMS320F2837xD DSP

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

Abstract: This paper presents a procedure for sampling analog signals using the analog-to-digital converter (ADC) and digital-to-analog converter (DAC) of a digital signal processor (DSP). The procedure involves configuring the ADC and DAC peripherals, initializing the ePWM module, setting the timebase period, and triggering the ADC conversion using the ePWM event. The paper also discusses the implementation details and presents the results of the sampling procedure. The procedure can be used in a wide range of applications, including data acquisition, signal processing, and control systems.

Introduction

Sampling analog signals is a crucial step in many applications, such as data acquisition, signal processing, and control systems. The accuracy and speed of the sampling process depend on the performance of the analog-to-digital converter (ADC) and digital-to-analog converter (DAC) used in the system. In this paper, we present a procedure for sampling analog signals using the ADC and DAC peripherals of a digital signal processor (DSP). The procedure involves configuring the ADC and DAC, initializing the ePWM module, setting the timebase period, and triggering the ADC conversion using the ePWM event. The paper also presents the main objective and three specific objectives of the procedure.

Objectives

Main Objective

To present a procedure for sampling analog signals using the ADC and DAC peripherals of a DSP.

Specific Objectives

1. To configure the ADC and DAC peripherals of the DSP for signal sampling.
2. To initialize the ePWM module and set the timebase period for triggering the ADC conversion.
3. To demonstrate the effectiveness of the sampling procedure by presenting the results of the sampling process.

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 $f_a = 100\text{Hz}$, using those values we obtaing the sampling values for each scenario.

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

The following step it's to configure the signal frequency, in the following way of setting up an ePWM module a sin wave signal is generated. Knowing the frequency of the signal the PWM peripheral needs to set up its signal frequency: *Time base period (TBPRD)* as its shown in eq(3).

$$\text{TBPRD} = \frac{f_{\text{DSP}}}{2 f_s D_1 D_2}$$

(3)

Eq(1) shows us the divider value 2, because it's used the up and down ePWM counter mode, also shows two dividers, which can be modified, $CLKDIV = D1 = [1, 2, 4, 8, 16, 32, 64, 128]$ and $HSPCLKDIV = D2 = [1, 2, 4, 6, 8, 10, 12, 14]$, Just to recall the maximum TBPRD value is 6553565535 if that value is greter than or equal to it you must modify the dividers values, for this configuration eq(1) is set:

$$\text{TBPRD} = \frac{10\text{MHz}}{8 f_s}$$

(4)

This value is going to be necessary in all the next PWM configurations because we will need the following TBPRD:

$$\text{TBPRD}_{f_{s_1}} = 62.500$$
$$\text{TBPRD}_{f_{s_2}} = 12.500$$
$$\text{TBPRD}_{f_{s_3}} = 2.500$$
$$\text{TBPRD}_{f_{s_4}} = 244$$

The procedure for sampling analog signals using the ADC and DAC of the DSP involves the following steps:

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

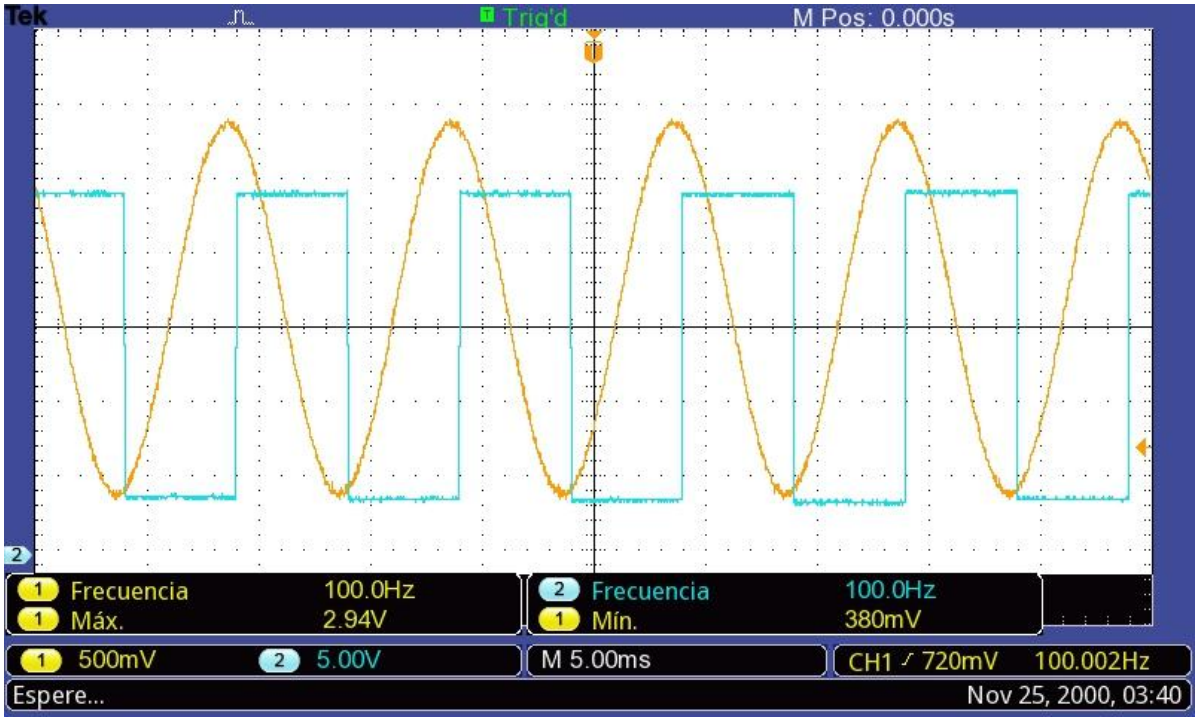

Fig. 1. Sampling when $N = 2$.

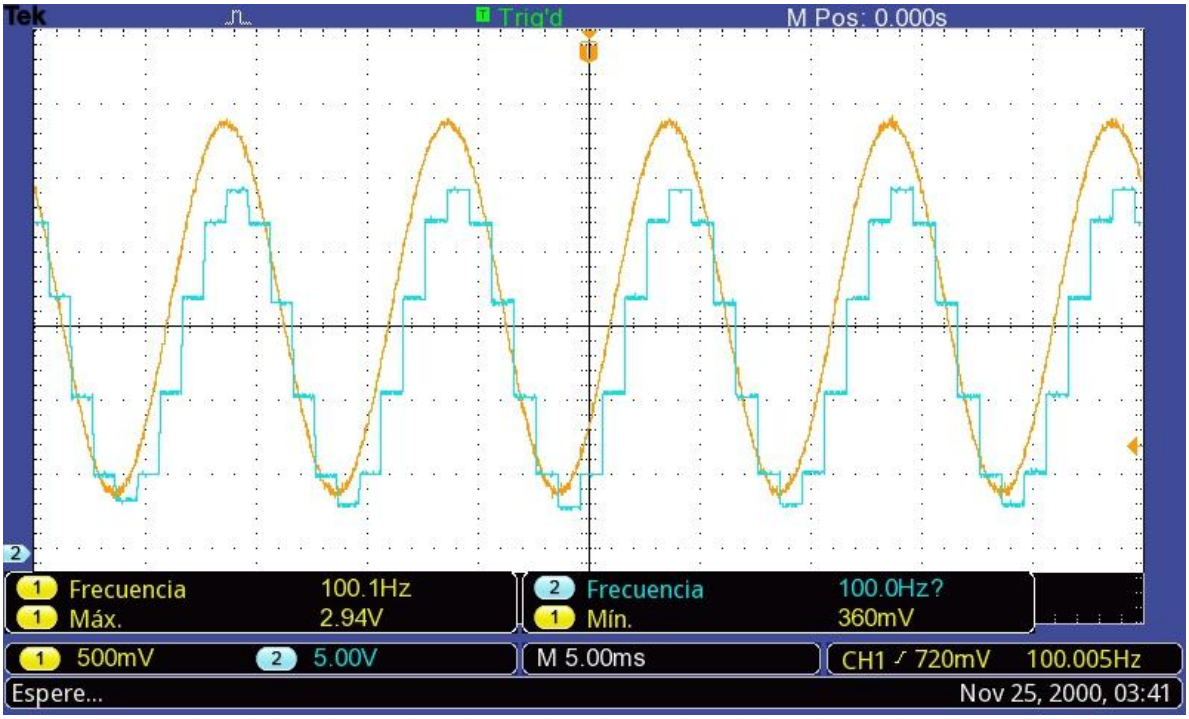

Fig. 2. Sampling when $N = 10$.

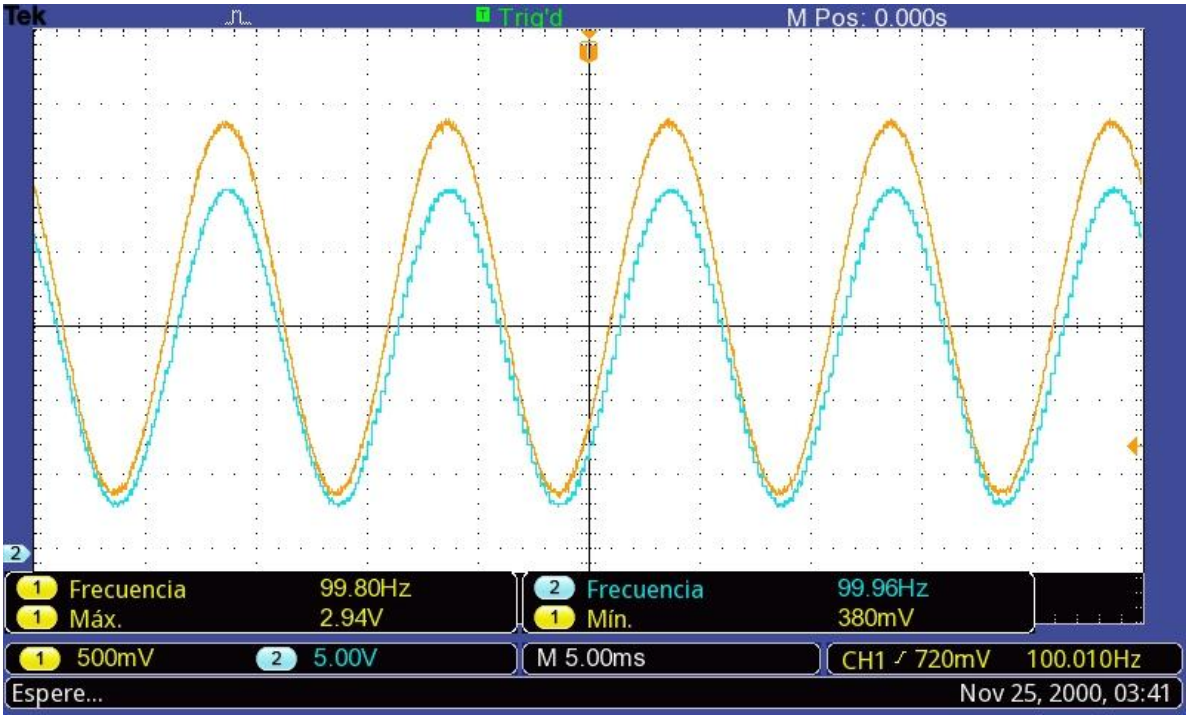

Fig. 3. Sampling when $N = 50$.

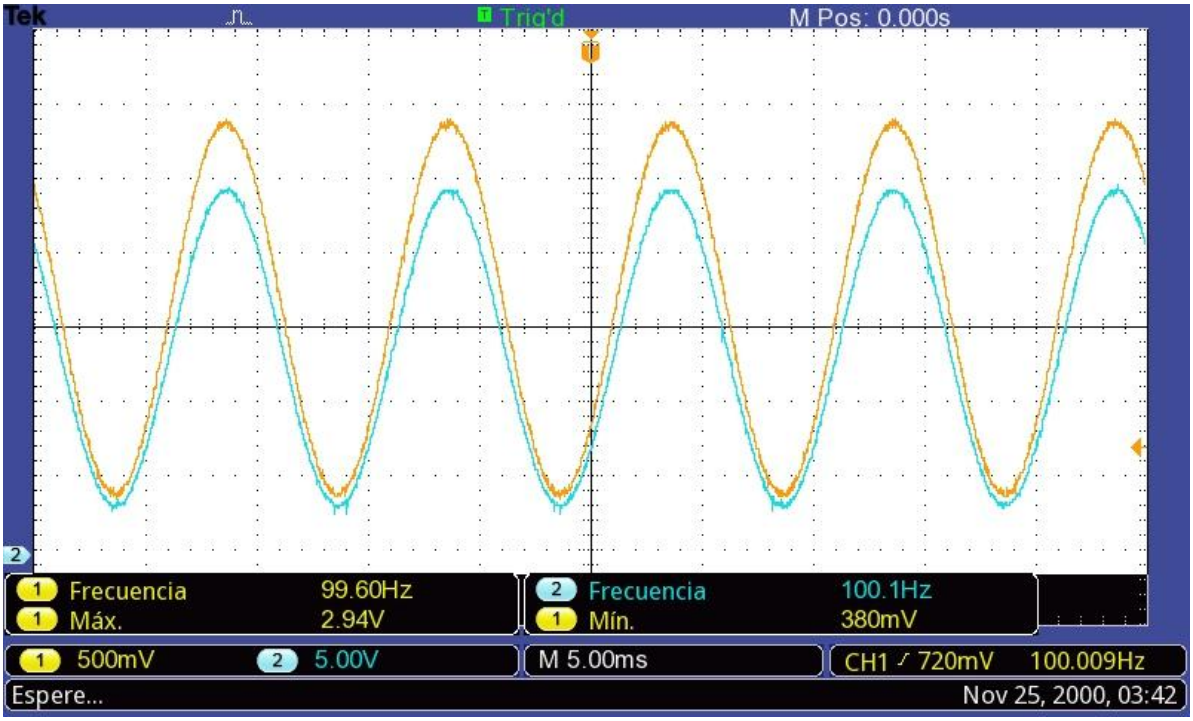

Fig. 4. Sampling when $N = 512$.

We are able to visualize how the sampled signal is achieving the original wave form incrementing the Nvalue.

Conclusions

The ADC and DAC modules of the DSP were successfully used to sample and output an analog signal. This demonstrates the ability of the DSP to interface with analog components and perform real-time signal processing.

The accuracy and precision of the ADC and DAC modules were found to be within acceptable ranges for the given application. However, if higher accuracy or precision is required, more advanced ADC and DAC modules may need to be used.

The use of interrupts was critical to ensuring that the ADC and DAC modules operated smoothly and did not cause timing issues or buffer overflow. Careful consideration of interrupt timing and handling is important for any real-time signal processing system.

This experiment highlights the importance of understanding the capabilities and limitations of the hardware components in a DSP system. Proper configuration and optimization of these components is crucial to achieving the desired signal processing performance.

References

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

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

[3] DSP2839D Plans. Texas Instruments.