

Bologna Master Degree in Electrical and Computer Engineering

Digital Signal Processing Systems

2022/2023 - 2nd semester, P4

Introduction to the Digital Signal Processing Starter Kit DSK TMDX5515eZDSP & Numerically Controlled Oscillator (NCO)

Group 4:

Diogo Ferreira | 96189

Duarte Marques | 96523

Fábio Dias | 96201

May 21st 2023

Contents

1	Introduction	2
2	Demonstration Projects	2
2.1	Sine	2
2.2	Loop	4
3	Project #1 – Numerically controlled oscillator (NCO)	6
Task 1	7
Task 2	7
Task 3	8
Task 4	9
Task 5	10
Task 6	11
Task 7	12
Task 8	14
Task 9	16
4	Conclusion	17

1 Introduction

This report corresponds to the intermediate submission regarding the project to conduct on digital signal processing systems. The main objective of this work is to introduce the Digital Signal Processing Starter Kit (DSK) TMDX5515eZDSP from Texas Instruments and conduct the first step of the final project, which corresponds to creating a Numerically Controlled Oscillator (NCO) - sinusoidal quadrature oscillator and frequency synthesizer. To program this DSK, the software development tool Code Composer Studio (CCS) was used.

This document follows the steps of the laboratory guide, answering its questions sequentially while evaluating the experimental data and commenting on the achieved results. The analyses of the first two demonstration projects will be performed, followed by the development and study of the NCO.

2 Demonstration Projects

Two demonstration projects are present in the following subsections to serve as examples to understand the working principles and characteristics of the **software**. The main objectives are to check the file structure, as well as build, debug and run simple programs to serve as base examples from which the main project will be developed.

2.1 Sine

Listing 1 presents part of the code for the initial demonstration project. After defining and declaring the necessary variables, the code enters an infinite loop denoted by `while(1){[...]}`. This loop ensures that the enclosed code will run continuously. Inside this loop, there is the reading of data from both left and right ADCs, working at the sampling frequency of 16 kHz, followed by the output of a value from the previously declared `SineBuffer[]`, indexed by the variable `index`, which as a looping characteristic changing its values between 0 and 15.

```
void main( void ){[...]  
    Int16 DataInLeft, DataInRight, DataOutLeft, DataOutRight;  
    Int16 SineWave[]={0,383,707,924,1000,924,707,383,0,-383,-707,-924,-1000,-924,  
        -707,-383,0};  
    Int16 index, gain=32;  
    Int16 SineBuffer[16];  
  
    while(1) {  
        /* Read Digital audio */  
        while((Rcv & I2S0_IR) == 0); // Wait for interrupt pending flag  
        DataInLeft = I2S0_W0_MSW_R;  
        DataInRight = I2S0_W1_MSW_R; // data  
        /* Write Digital audio */  
        while((Xmit & I2S0_IR) == 0); // Wait for interrupt pending flag  
        I2S0_W0_MSW_W = DataOutLeft;
```

```


        I2S0_W1_MSW_W = DataOutRight; // data

        SineBuffer[index]=SineWave[index]*gain; // gain=32 or gain=33
        DataOutLeft=SineBuffer[index];
        DataOutRight=SineBuffer[index];
        index=(index+1)&15;
    }
}

```

Listing 1: Main part of the code implemented in the source file ‘main.c’, included in project ‘Sine’.

Upon analyzing the provided code, several immediate observations can be made. Firstly, all the variables used are integers due to the fixed-point system being utilized. In this system, all operations are performed using integers, and the programmer is responsible for managing the decimal positions.

Secondly, examining the `SineBuffer[]` vector reveals the presence of values representing a full period of a sine wave. It can be argued that representing only half or even a quarter of the period would suffice, thus reducing memory usage, but increasing programming complexity. This topic will be further discussed when constructing the frequency-controlled sine wave in Section 3. Moreover, the full resolution of the 16-bit capability is not utilized. Since the sine wave values range between -1 and 1, they could be represented in Q_{15} format (1 sign bit, 0 integer bits and 15 decimal bits), using values from -32768 to the maximum of 32767. However, in this example, each sine wave value is multiplied by a default gain of 32. As a result, the sine wave approaches  tary amplitude, with values ranging from -32000 to 32000.

Additionally, there is the issue of the gain being set to 33. If the vector values were multiplied by 33, the resultant values would range between -33000 and 33000, which exceeds the range of possible 16-bit represented values $[-32768, 32767]$. This results in an overflow, bringing the values from the most positive to the most negative, followed by a return to positive values, as illustrated in Figs. 1 and 2. This happens because the numbers are represented in two’s complement. Something similar occurs when there is an underflow. In order to reproduce the signal shown in Fig. 2 for a gain of 33, a wide range of frequencies would be necessary, thus the discrepancy observed between this plot and the respective results in the oscilloscope (Fig. 1).

The frequency of the output signal can be controlled by changing the index increments. In case the index variable is only changed by 1 in each iteration of the `while` loop, as it’s done in Listing 1, the sinusoid will be reached by using the 16 points from `SineBuffer[]` at the sampling frequency of $f_s = 16 \text{ kHz}$, resulting in a frequency in the output signal of $f = f_s/16 = 1 \text{ kHz}$. By incrementing the index two steps at a time, it is possible to double the frequency to 2 kHz, as the number of points used from `SineBuffer[]` will be half of its total. The same logic can continue to be applied to achieve higher frequencies, with a limit of incrementing the index eight steps at a time. This is, however, a theoretical limit situation: only by changing the table values so that the values used from the `SineBuffer[]` are not both zeros and by using the interpolation and filtering properties on the output signal would it be possible to get a 8 kHz final frequency. In practice, it cannot be achievable.

On the other hand, lower frequencies are impossible to produce without increasing the size of the vector `SineBuffer[]` - which would lead to having more points to use at the same conversion frequency.

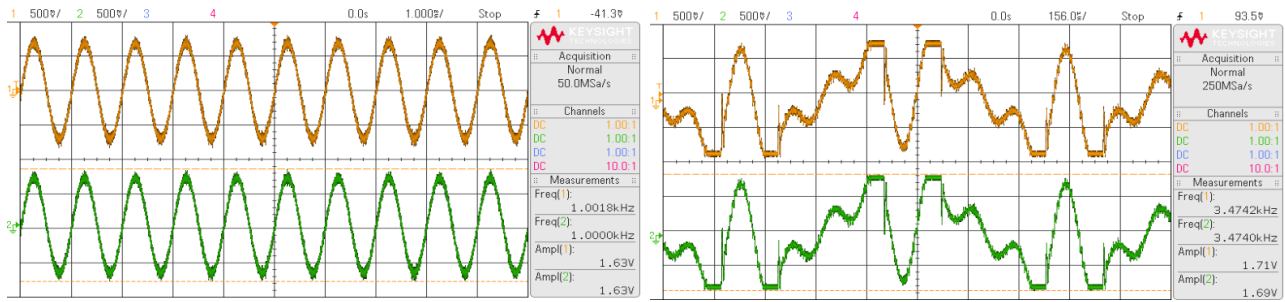


Figure 1: Oscilloscope traces showing the output signals (left and right channels) when the gain is set to 32 and 33, respectively.

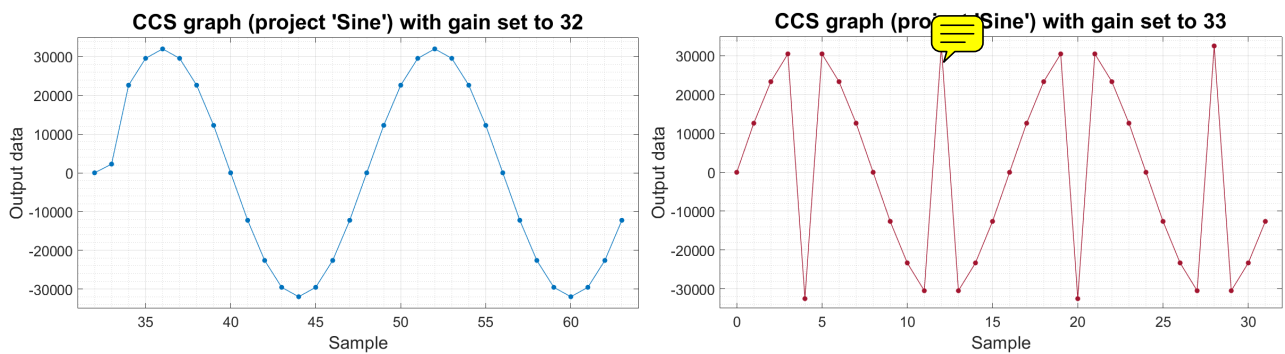


Figure 2: CCS graphs showing the output signals (left and right channels) when the gain is set to 32 and 33, respectively.

2.2 Loop

A similar approach is followed in the second demonstration project, whose code is very similar to the one presented for project ‘Sine’, with the main difference being shown in Listing 2. After the initial definitions and variable declarations, the code enters the `while(1){...}` loop. Within this loop, the code reads data from the left and right ADCs, which operate at a sampling frequency of 16 kHz. Subsequently, the received input is copied and outputted as a signal.

```
while(1) {
    DataOutLeft = DataInLeft; // loop left channel samples
    DataOutRight = DataInRight; // loop right channel samples
}
```

Listing 2: Main distinction included in the source file ‘main.c’ of project ‘Loop’.

In the case of a sinusoidal wave with an amplitude of 1.2 V and a frequency of 1 kHz (as shown in Fig. 3), it can be observed that the output closely follows the input, with only a slight delay caused by the signal processing time and the phase difference resulting from the **transfer function** of the microcontroller. An analogous delay is also noticeable in Figs. 4 to 6.

It is important to note that the peak-to-peak amplitude was chosen to be 1.2 V (values in $[-0.6 \text{ V}, 0.6 \text{ V}]$) in order to utilize the full range of the ADCs conversion capability, which has a maximum reading voltage of around 1.4 V. However, it should be clarified that the ADC operates with a voltage reference of 3.3 V, and decoupling capacitors set the analog ground voltage to 0.9 V. As a result, the ADC can accept voltage swings of $\pm 0.7 \text{ V}$ within the range of 0.2 V to 1.6 V. When a signal with a peak-to-peak voltage of 1.2 V is applied, the ADC reads voltages within the $[0.9 - 0.6, 0.9 + 0.6] = [0.3 \text{ V}, 1.5 \text{ V}]$ range, with a 0.1 V safety margin accounting for errors and internal imbalances.

When the input frequency varies, the effects of frequency filtering become evident. A bandpass-like frequency filtering behavior is observed when transitioning from a signal close to DC (e.g., 1 Hz) to the maximum frequency that can be sampled with the 16 kHz conversion rate (8 kHz, the Nyquist frequency). This effect is illustrated in Figs. 4 to 6, where the amplitude is attenuated for very low frequencies (such as 1 Hz) or frequencies equal to or larger than 8 kHz.

These effects are even more pronounced in the case of a square wave input, whose Fourier expansion contains infinite frequencies above the fundamental frequency. For example, when a square wave with a frequency of 1 kHz is applied, the output signal will consist of a sum of signals with frequencies of 1 kHz, 3 kHz, 5 kHz, and 7 kHz, as frequencies above 8 kHz are highly attenuated. Similarly, when the square wave frequency is changed to 3 kHz, the output will predominantly contain a sinusoidal wave with only 3 kHz, as the next harmonic at 9 kHz will experience significant attenuation.

Similar conclusions can be drawn for square waves with different input frequencies. With a 7 kHz input signal, the output will consist primarily of a 7 kHz sinusoid. For a 10 kHz input signal, even the fundamental frequency is attenuated, resulting in a DC signal.

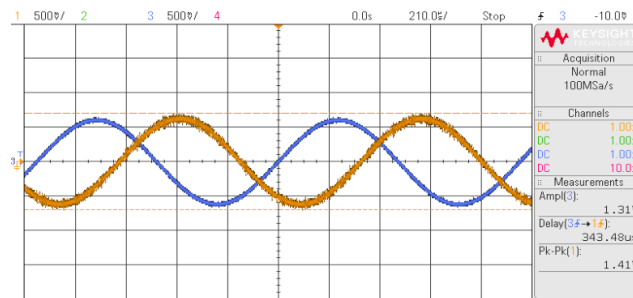
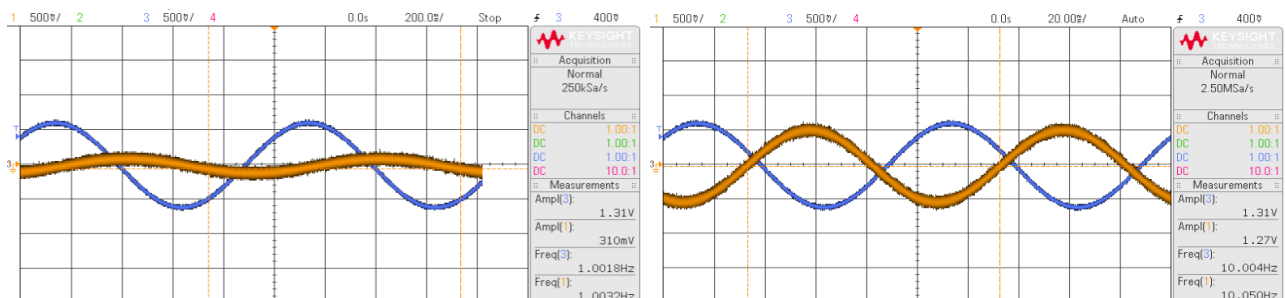


Figure 3: Oscilloscope trace showing the input sinusoidal signal (Channel 3) and the output signal (Channel 1) for 1 kHz.



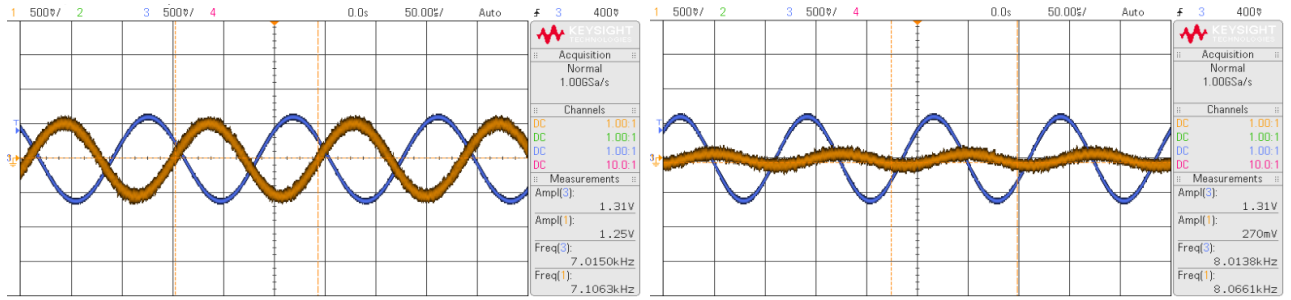


Figure 4: Oscilloscope traces showing the input sinusoidal signal (Channel 3) and the output signal (Channel 1) for frequencies of 1 Hz, 10 Hz, 7 kHz and 8 kHz, respectively.

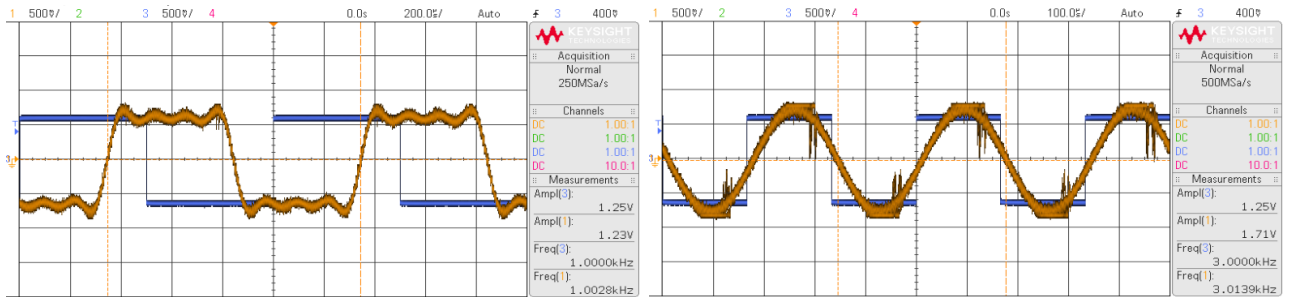


Figure 5: Oscilloscope traces showing the input square-wave signal (Channel 3) and the output signal (Channel 1) for frequencies of 1 kHz and 3 kHz, respectively.

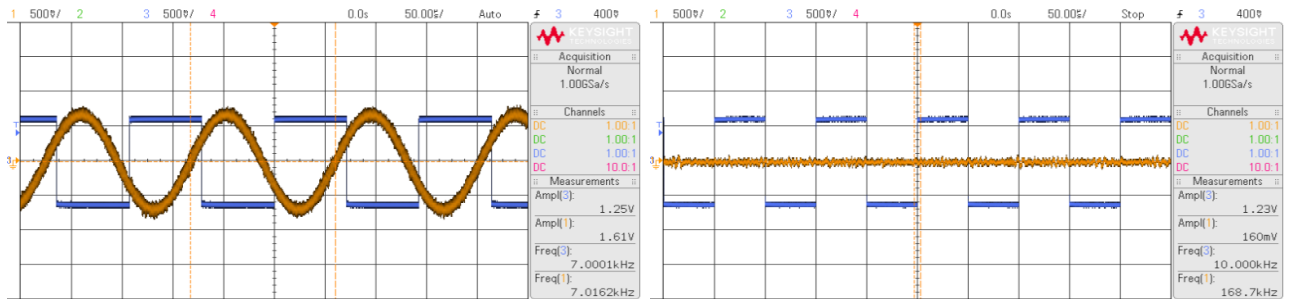


Figure 6: Oscilloscope traces showing the input square-wave signal (Channel 3) and the output signal (Channel 1) for frequencies of 7 kHz and 10 kHz, respectively.

3 Project #1 – Numerically controlled oscillator (NCO)

This section focuses on the initial phase of the final project submission, which involves creating a numerically controlled oscillator (NCO) following the steps outlined in the guide. The main objectives include constructing the required code, evaluating its implementation, and commenting on the experimental results. The following tasks are organized according to the enumerated steps in the guide. This relaxation oscillator should have a minimum frequency of $f_{\min} = 2 \text{ kHz}$ and a maximum frequency of $f_{\max} = 6 \text{ kHz}$. Once again, a pre-programmed sampling frequency of $f_s = 16 \text{ kHz}$ is used.

Task 1

The implementation of an oscillator is straightforward. It involves having a variable that exhibits cyclic behavior and can be controlled in terms of its periodicity. The circular nature of the two's complement representation enables the utilization of a 16-bit signed integer (`Int16 r`) that is incremented by a certain value (Δ). As a result, when the variable `r` reaches its maximum value, an overflow occurs, causing it to reset back to a value equal to or close to (and above) the minimum possible value. This behavior restarts the cycle of incrementation, leading to the effect illustrated in Fig. 7.

The relaxation oscillator (implemented in C) can be seen in Listing 3. It is important to note that the `Buffer[]` array is used solely for visualization purposes in the CCS graphs. This array is a 32-element `int` array and, as such, it requires repeated indexing within the range $[0, 31]$ to accommodate its circular behavior.

```
Int16 Buffer[32]; // Array for CCS graphs
Int16 r=0, Delta, index=0; // Declare variables
[...]
while(1) {[...]
    r+=Delta; // Increment ramp
    Buffer[index]=r;
    index=(index+1)&31;
    DataOutRight=r;
}]
```

Listing 3: Relaxation oscillator (ramp integrator) using a 16-bit signed integer variable `r`, incremented by a value `Delta` every sampling interval.

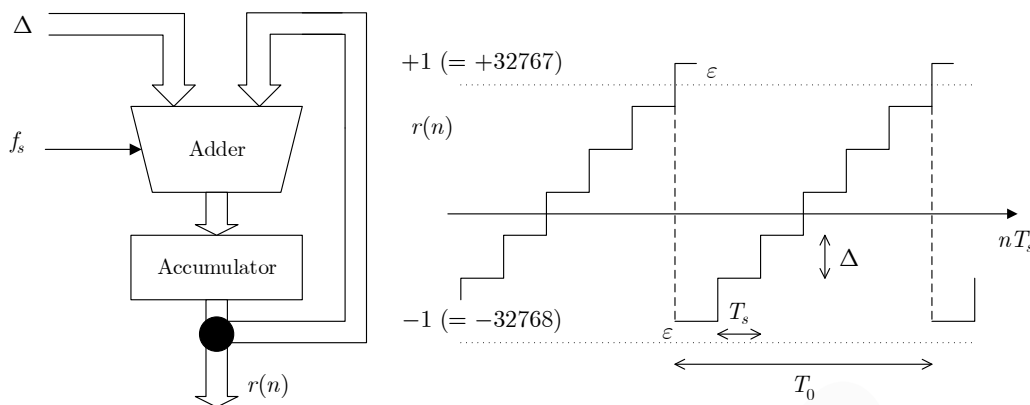


Figure 7: Relaxation oscillator principle and waveform.

Task 2

To determine the necessary value of Δ to achieve a specific frequency, (1) provides the appropriate correspondence. This equation relates the desired output wave's period (T) and frequency (f) to the sampling period (T_s) and frequency (f_s). Additionally, the equation considers using 2^{16} due to the 16-bit format used and can be easily inferred from the waveform shown in Fig. 7.

$$\frac{T}{T_s} = \frac{2^{16}}{\Delta} \iff \frac{f_s}{2^{16}} \cdot \Delta \iff \Delta = \frac{f}{f_s} \cdot 2^{16} \quad (1)$$

By applying the aforementioned formulas, the values of Δ can be determined for the central frequency (Δ_0), minimum frequency (Δ_{\min}), and maximum frequency (Δ_{\max}):

- $f_0 = 4\text{kHz} \implies \Delta_0 = 16384$;
- $f_{\min} = 2\text{kHz} \implies \Delta_{\min} = 8192$;
- $f_{\max} = 6\text{kHz} \implies \Delta_{\max} = 24576$.

```
Int16 Delta_0=16384, Delta_min=8192, Delta_max=24576;
//f0=4kHz, f_min=2kHz, f_max=6kHz
```

Listing 4: Code for the variables Δ_{Δ} , which control the oscillator frequency.

Task 3

The oscilloscope was then utilized to observe the oscillating signal through the output channel of the board. The signal representation when Δ is set to its central value (Δ_0) is illustrated in Fig. 8, where the right-hand image shows the respective CCS graph, in which successive periodic increments on the value of r can be verified. The outputted signal captured and displayed on the oscilloscope is shown in the left-hand image. It is important to note a clear discrepancy between these two images. The increase in the waveform value in the CCS graph is constant from point to point and the transitions from maximum to minimum values upon overflow exhibit a sudden change, with an infinitely sharp transition. An interpolation-like phenomenon arises on the oscilloscope, limiting high-frequency variations in the output signal and contributing to the differences observed on the device. This discrepancy, coupled with the presence of an inverter in the board (which makes it so that the ramp signal in the oscilloscope successively decreases), leads to noticeable changes in the output signal.

By changing the value of Δ to Δ_{\min} , a decrease in frequency can be observed, as the increments between successive points become smaller. When Δ is set to Δ_{\max} , another phenomenon occurs. Since Δ_{\max} is not a sub-multiple of the full range (2^{16}), the subsequent value after an overflow is not always the same. Consequently, lower frequency harmonics are present in addition to the main frequency resulting from the factors mentioned above. The images captured from the oscilloscope are shown in Fig. 9, whereas Fig. 10 displays both CSS graphs.

In Fig. 9, it can be seen in the oscilloscope trace that the output signal has a frequency of $f \approx 2\text{kHz}$ when $\Delta = \Delta_{\max}$. In fact, by using (1), it can be concluded that $T_{\max} = (8/3)T_s$, which means that the period of the output signal is $3 \times T_{\max}$ (after three cycles, an integer multiple of T_s is obtained). Consequently, the frequency of the output signal is, in fact, $f = f_{\max}/3 = 2\text{kHz}$.

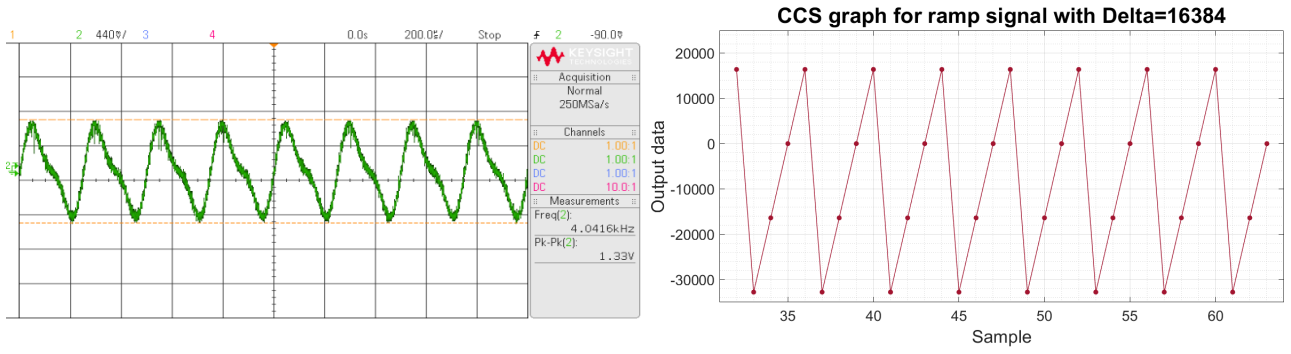


Figure 8: Oscilloscope trace (on the left) and CCS graph (on the right) of the right output channel showing the state variable $r(n)$ for $\Delta = \Delta_0 = 16384$.

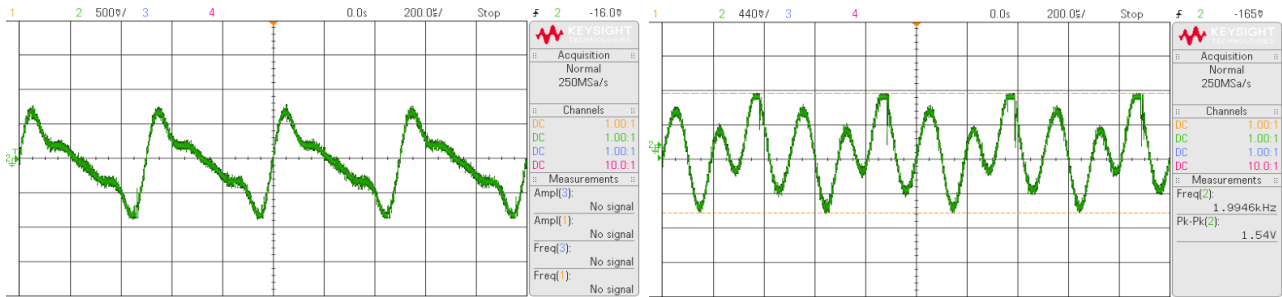


Figure 9: Oscilloscope traces of the right output channel showing the state variable $r(n)$ for $\Delta = \Delta_{\min} = 8192$ and $\Delta = \Delta_{\max} = 24576$, respectively.

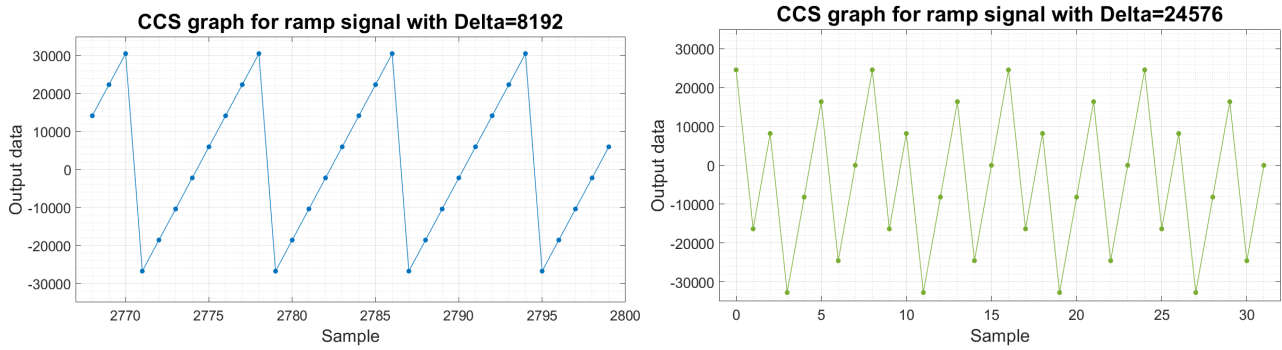


Figure 10: CCS graphs of the right output channel showing the state variable $r(n)$ for $\Delta = \Delta_{\min} = 8192$ and $\Delta = \Delta_{\max} = 24576$, respectively.

Task 4

The vector shown in Listing 5 is produced using the following:

$$y(m) = \text{round} \left(32767 \times \sin \left(\frac{\pi}{32} m \right) \right), \quad m = 0, 1, \dots, 31 \quad (2)$$

```
Int16 lut[32]={0,3212,6393,9512,12539,15446,18204,20787,23170,25329,27245,28898,
30273,31356,32137,32609,32767,32609,32137,31356,30273,28898,27245,25329,23170,
20787,18204,15446,12539,9512,6393,3212};
```

Listing 5: Look-up-table (LUT) obtained using (2), which corresponds to 32 positive values of half a cycle of the sine function using the most accurate representation format (Q_{15}).

Task 5

Using the oscillator variable r and extracting the 5 most significant bits (excluding the sign bit) through a logical AND operation with the appropriate bit mask ($011111_2 = 31_{10}$), the vector created in the previous section (Task 4) is indexed to generate a sine wave.

Since the vector represents only the positive values of a sinusoid (half cycle), the sign of the output wave is managed based on the sign of the r variable - positive values of the ramp correspond to positive values of the sine wave and negative values of the ramp correspond to the symmetrical values of the look-up-table.

The code for generating the sine wave is presented in Listing 6 and Fig. 11 illustrates the traces of the oscillator r and the resulting sine wave.

```
Int16 y , i;
[...]
while(1) {[...]
    i = (r>>10)&(31); // Extract i to adress the LUT
    y = lut[i];
    if (r<0){
        y=-y;
    }
    DataOutLeft = y; // Since wave
    DataOutRight = r; // Ramp
}]
```

Listing 6: Using the oscillator state variable r to index the LUT and obtain the sine values.

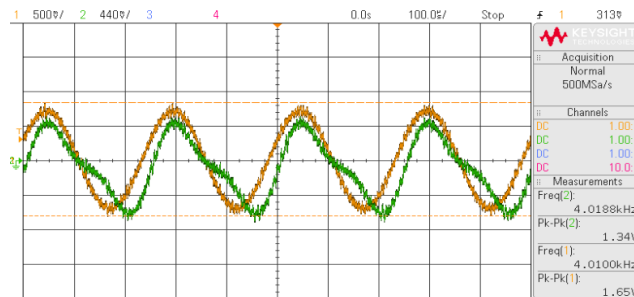


Figure 11: Oscilloscope trace showing the oscillator signals - ramp signal (Channel 2) and sine wave $y(n)$ (Channel 1).

Task 6

By introducing a variable to control the gain of the sine wave, it is possible to adjust the sinusoid's amplitude by multiplying the values obtained from the `lut` vector. It is crucial to ensure that the multiplication factor is less than one, as the vector already utilizes the entire range of a 16-bit word. Using a larger value could lead to an overflow and produce incorrect results. This approach allows for adjusting and reducing the output sine wave amplitude.

Listing 7 provides the code snippet responsible for performing the multiplication. It is worth noting the casting of one of the multiplication operands. When multiplying two 16-bit operands, the **ALU (Arithmetic Logic Unit)** uses a 16-bit register, resulting in the least significant bits being preserved and potentially rendering the multiplication value invalid. One of the operands is cast to a 32-bit word to address this issue, forcing the ALU to utilize a 32-bit register and preserve the entire result of the operation. After the multiplication, a right shift by 16 bits is performed to replace the least significant bits with the most significant ones. Consequently, when storing the final result in the 16-bit variable `y`, the correct value is obtained. Additionally, since the multiplication will produce a word with 2 sign bits, an initial left shift operation is performed to discard this unnecessary bit and increase the resolution of the value.

```
Int16 Gain=16384; // 1/2->16384; 1/4->8102
[...]
while(1) {[...]
    i = (r>>10)&(31);

    y=((long)lut[i]*Gain)<<1>>>16;

    if (r<0){
        y=-y;
    }
    DataOutLeft = y;
    DataOutRight = r;
}
```

Listing 7: Creating a new variable (**Gain**) to control the amplitude of the sinusoidal signal.

In the images shown in Fig. 12, the oscillator waveforms are depicted for $f_0 = 4$ kHz. The output signal is presented for two different gain settings: for a half-scale amplitude (which, in Q_{15} format, is given by a gain of $\text{round}(0.5 \times 2^{15}) = 16384$) and for a gain equal to $1/4$ ($\text{round}(0.25 \times 2^{15}) = 8102$ in Q_{15} format).

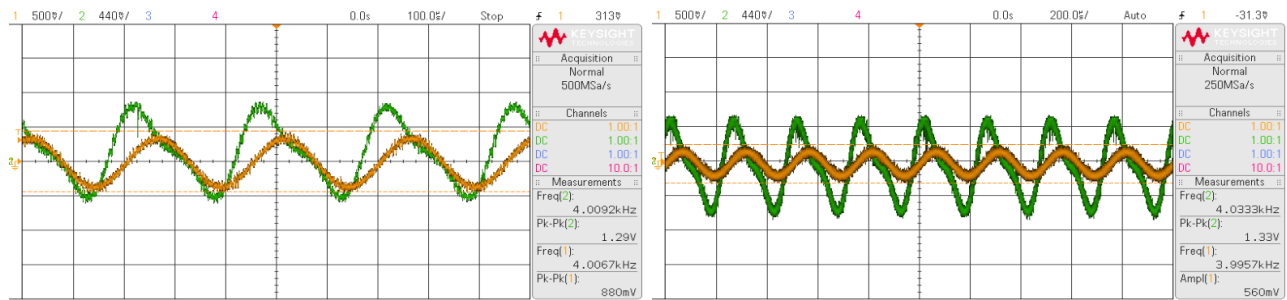


Figure 12: Oscilloscope traces showing the oscillator signals - ramp signal (Channel 2) and sine wave $y(n)$ (Channel 1), for a frequency of $f = f_0 = 4$ kHz and gains equal to $1/2$ and $1/4$, respectively.

Task 7

The frequency of the output signal, and consequently the value of Δ , should vary with the input voltage. The expression in (3) represents the relation between the input voltage $\epsilon(n)$ and the oscillator instantaneous frequency, which should vary linearly with the input signal amplitude. By utilizing this equation, it is possible to calculate the required value for the variable k using the expression given in (4) since, when $\epsilon(n) = 1$ (+32767), the output frequency should be $f_{\max} = 6$ kHz and, when $\epsilon(n) = -1$ (-32768), it should be $f_{\min} = 2$ kHz. Since $k = 2^{-2}$ (8192 in Q_{15} format), the value of Δ obtained by (3) could simply be computed by shifting $\epsilon(n)$ two bits to the right.

$$\Delta = \Delta_0 + k \times \epsilon(n) \quad (3)$$

$$k = \frac{\Delta_{\max} - \Delta_0}{\epsilon(n)_{\max} - \epsilon(n)_{\min}} = \frac{\Delta_{\min} - \Delta_0}{\epsilon(n)_{\min} - \epsilon(n)_{\max}} = 0.25 \quad (4)$$

```
[...]
while(1) {[...]
    Delta = Delta_0 + (((long)8192*DataInLeft)<<1)>>16); // k=0.25->8192 in Q15

    r+=Delta;
    [...]
    y = lut[i];
    if (r<0){
        y=-y;
    }
    DataOutLeft = y;
    DataOutRight = DataInLeft;
}]
```

Listing 8: Controlling the oscillator frequency with the input signal, by varying Δ linearly with the amplitude.

The images shown below demonstrate the response of the output signal to variations in the input voltage. Applying a square wave input signal ranging from -0.6 to 0.6 , the output sinusoid undergoes a frequency change from around f_{\min} to nearly f_{\max} . The trace in Fig. 13 provides an overview of the frequency transition, while Figs. 14 and Fig. 15 present zoomed-in views of each frequency waveform. In this case, the highest value of $\epsilon(n)$ corresponds to the lowest frequency in the output signal (and vice-versa), as opposed to what was to be expected by the relation in (3), due to the inverter included in the DSK board. The values of these frequencies are close to the minimum and maximum frequencies of 2 kHz and 6 kHz, respectively, even though slight variations from these values can be seen in the oscilloscope traces due to imperfections in the externally applied input signal.

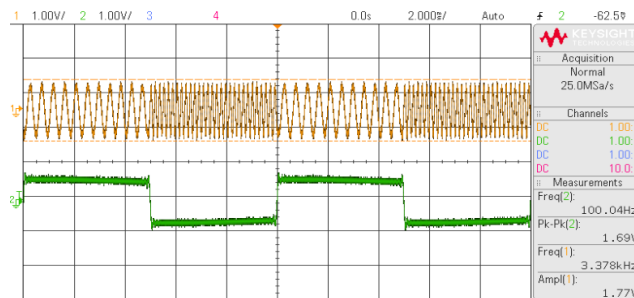


Figure 13: Oscilloscope trace showing the input signal after sampling $\epsilon(n)$ (Channel 2), and the frequency-modulated output signal $y(n)$ (Channel 1).

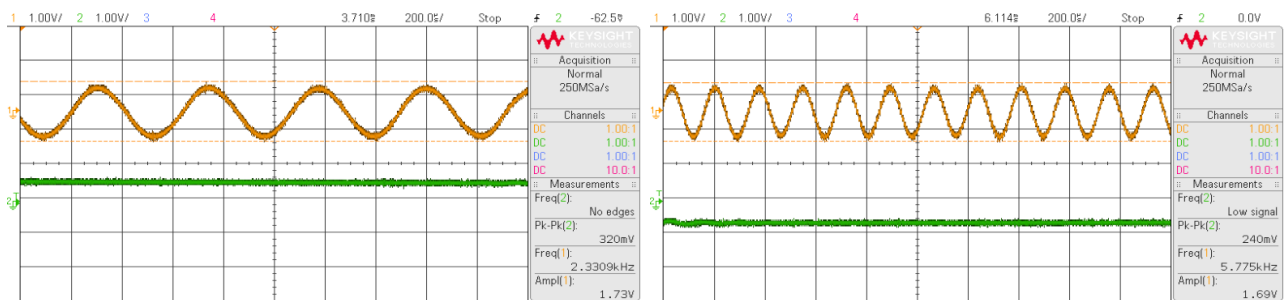


Figure 14: Zoomed in oscilloscope traces showing the input signal after sampling $\epsilon(n)$ (Channel 2), and the frequency-modulated output signal $y(n)$ (Channel 1), for the two different frequencies obtained in $\epsilon(n)$ (shown in Fig. 13).

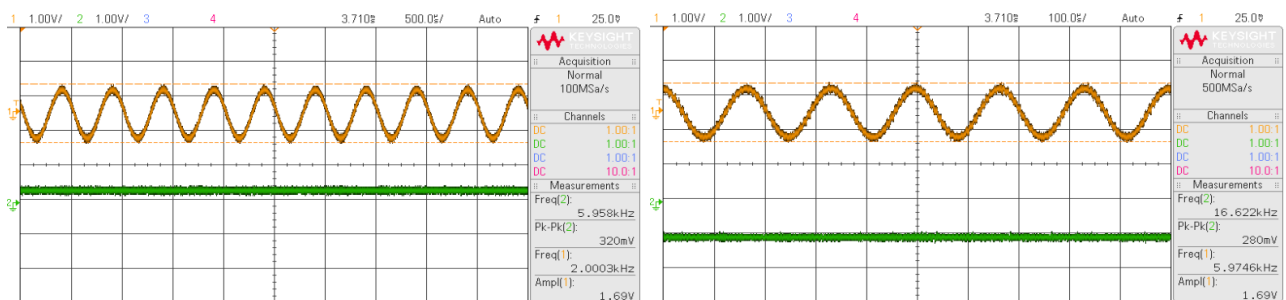


Figure 15: Oscilloscope traces showing the input signal after sampling $\epsilon(n)$ (Channel 2), and the frequency-modulated output signal $y(n)$ (Channel 1), for $\epsilon(n) = -1$ (-32767) and $\epsilon(n) = +1$ ($+32767$), respectively.



Task 8

The effects of the change in frequency can be observed by conducting a Fourier analysis. In the left-hand side of Fig. 16, a signal with a frequency of 4 kHz is displayed. As discussed earlier, this frequency is a sub-multiple of the sampling frequency, and the value of Δ is a sub-multiple of the full range. Consequently, the Fourier analysis exhibits a clean frequency spectrum, visible in both examples in Fig. 17.

However, when the value of Δ is slightly reduced, corresponding to a frequency of 3.9991 kHz, the Fourier analysis shows a spread of spectral components around the central frequency, as seen in the left-hand side image of Fig. 18.

Signal interpolation can be employed to mitigate or reduce these effects. The differences between successive points are smoothed out by applying interpolation, concentrating the signal energy around the central frequency. This can be observed in the right-hand side of Fig. 18.

The code used to perform the interpolation is provided in Listing 9. Furthermore, it is worth mentioning that in order to achieve good frequency resolution, a decrease in time resolution is necessary - in the oscilloscope traces of Fig. 16, a resolution of $100\mu\text{s}$ is used, while this value corresponds to 200ms in the FFT spectrums.

```
[...]
while(1) {[...]
    r+=Delta;
    i = (r>>10)&(31);

    f=(r&1023)<<5; // fractional part
    y1=lut[i]; // Q15
    y2=lut[(i+1)&31]; // Q15

    if (r<0){
        y1=-y1;
        y2=-y2;
    }

    y=y1+((((long)(y2-y1)*f)<<1)>>16); // Interpolation

    DataOutLeft = y; // With interpolation
    DataOutRight = y1; // Without interpolation
}
```

Listing 9: Linear interpolation using the remaining 10 bits from the ramp r to form the remainder f .

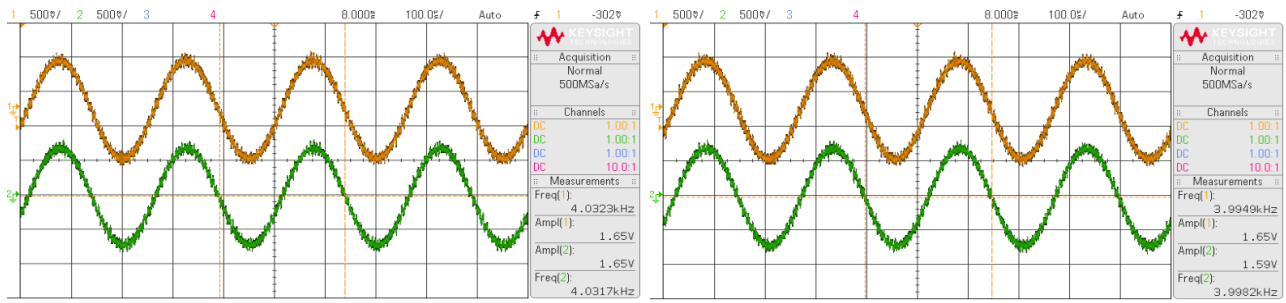


Figure 16: Oscilloscope traces showing the generated sine waves with and without interpolation (Channels 2 and 1, respectively), for $f_0 = 4 \text{ kHz}$ and $f_0 = 3.9991 \text{ kHz}$, respectively.

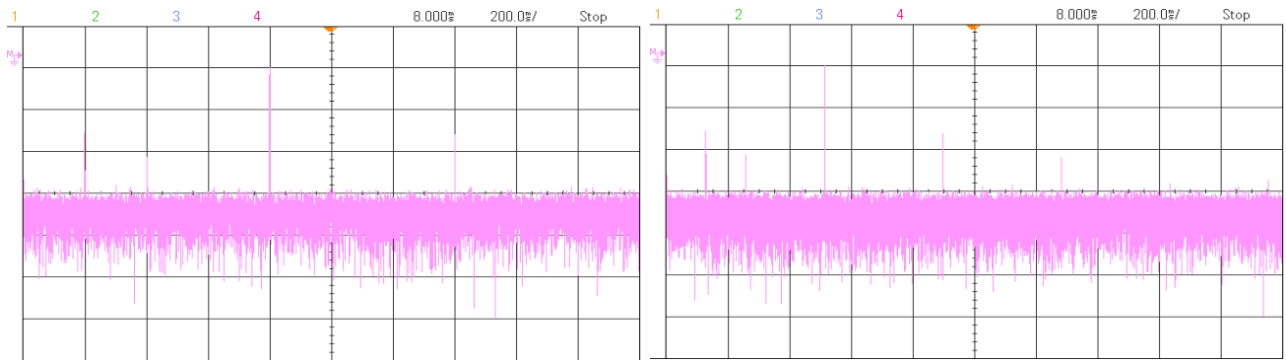


Figure 17: Oscilloscope traces showing the spectrum of the generated sine wave without interpolation (on the left) and with interpolation (on the right) for $f_0 = 4 \text{ kHz}$, using the oscilloscope FFT feature with the signals on Channels 1 and 2, respectively. To obtain these images, spans of 10 kHz (centered around 5 kHz) and 15.6 kHz (centered around 7.8 kHz) were respectively selected, therefore the largest peak (which occurs around 4 kHz) **does not appear in the same location in the two images.**

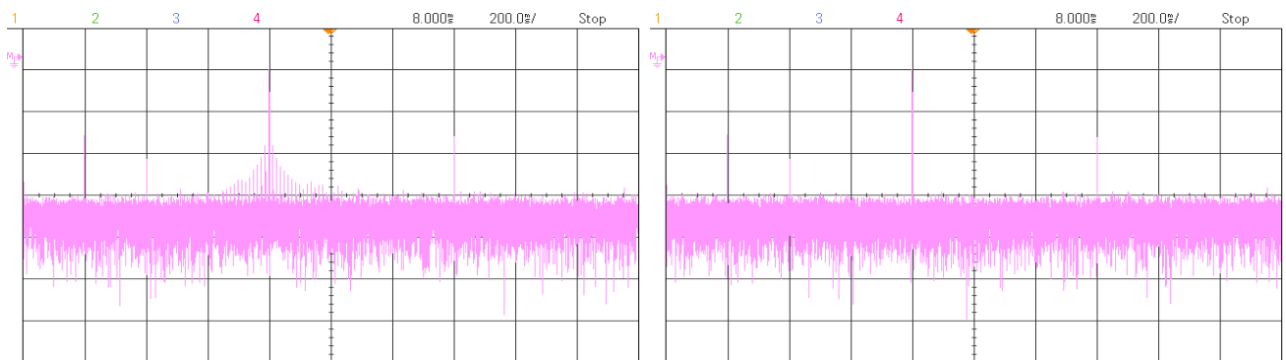


Figure 18: Oscilloscope traces showing the spectrum of the generated sine wave without interpolation (on the left) and with interpolation (on the right) for $f_0 = 3.9991 \text{ kHz}$, using the oscilloscope FFT feature with the signals on Channels 1 and 2, respectively. A span of 10 kHz (centered around 5 kHz) was selected to obtain these images.

Task 9

The interpolation concept demonstrated in Task 8 can be visualized in Fig. 19. In each iteration of the loop, instead of using the current vector value as the index, both the current value and the following value are utilized. A weighted average is calculated, where a value between the two numbers is obtained based on the remaining fractional part of the r variable. The 5 most significant bits represent the integer part of the index, while the remaining 10 bits serve as a weight for the subsequent vector value.

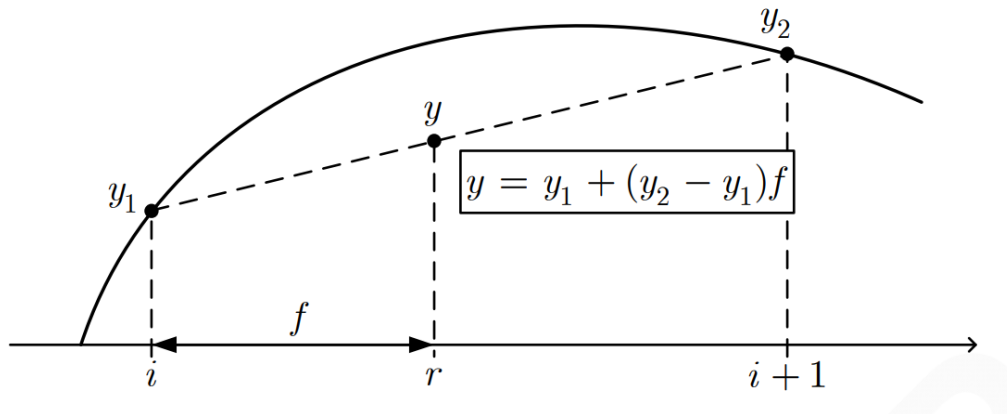


Figure 19: Look-up table interpolation scheme.

It is worth noting that the value following the current one may be outside the bounds of the vector, particularly if the current value is the last element. There are two approaches to address this issue: the first option involves increasing the vector size by repeating the first vector value at the end, allowing access to it while maintaining the correct value - this was the option taken on the demonstration projects (as shown in Listing 1). Although simple, this solution consumes more memory. The second approach, implemented in the provided code, involves checking the index variable using a logical AND operation with a mask, similar to the one used for circulating the index variable itself. This method increases the complexity of the code and processing, but not significantly. **As shown in Listing 9, a bit mask of $0111111111_2 = 1023_{10}$ was used.**

To perform the multiplication, as mentioned earlier, one of the 16-bit variables needs to be cast to 32-bit. A left shift is applied to discard the repeated sign bit, and a subsequent right shift of 16 bits is performed to restore the value to a 16-bit format and assign it correctly to the output variable y .

An evaluation of the data format can also be conducted. Both y_1 (the current vector value in the iteration) and y_2 (the next vector value) are in Q_{15} format, representing a sinusoid within the range of $[-1, 1]$. The variable f is also in Q_{15} format, representing the fractional part of the indexing value derived from the r oscillator. It can be understood that the subtraction $y_2 - y_1$ results in a value smaller than 1. **It can also be concluded that the sum of y_1 with the result of $(y_2 - y_1) \times f$ will also be less than 1, as the difference between subsequent vector**

values will always be smaller than the required value to exceed 1 when multiplied by f and summed to y_1 . In case y_1 were 1, the respective fractional part will be $f=0$; thus the result of the interpolation will also not exceed 1 - an analogous situation occurs in case y_1 were -1. This ensures that the resulting value remains below unity and can be represented in Q_{15} format. Even if intermediate calculations were to exceed the range of this format (which does not occur), the final value, being within the range, can and will be used to achieve higher resolution.

4 Conclusion

This introductory laboratory assignment provided valuable hands-on experience and learning opportunities. The successful implementation of the NCO on the DSK TMS320C5515 platform demonstrated a well-executed development process. Additionally, gaining knowledge in fixed-point arithmetic proved to be a valuable outcome. The insights and skills acquired from this assignment will be instrumental in advancing the subsequent phases of the project.

