

# **Instituto Superior Técnico**

## **Signal Processing Electronic Systems (MEE)**

3<sup>rd</sup> Quarter 2022/2023

Gonçalo Tavares

Introduction to the Digital Signal Processing

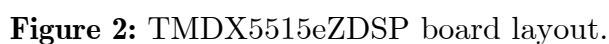
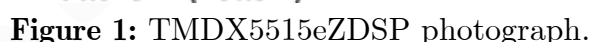
Starter Kit DSK TMDX5515eZDSP

&

Numerically Controlled Oscillator (NCO)

(Revised February 2023)

The main objective of this work is to become familiar with the software developing systems and digital signal processing starter kit DSK TMS320C5515 from *Texas Instruments*. In addition, a sinusoidal quadrature oscillator and frequency synthesizer will be developed and programmed. The TMS320C5515 is a 16 bit, fixed point digital signal processor with 240 MIPS (mega instructions per second) performance and 320kB of on-chip internal memory. The TMDX5515eZDSP is a starter kit featuring the TMS320C5515 DSP, a 16 bit stereo audio codec, a secure digital card interface and a USB 2.0 dedicated interface. The DSK photograph and board layout are represented in Figures 1 and 2 respectively.

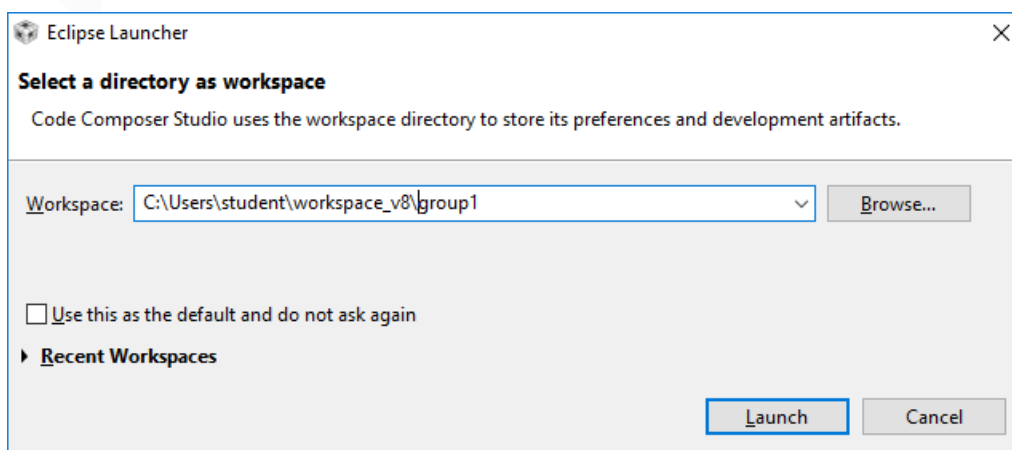


The software development tool used to program this DSK is the software suite *Code Composer Studio* (CCS), version 8.3.1. This program provides an integrated development environment (IDE) which is based on the Eclipse development suite and includes all required applications such as the assembler, compiler, linker and debugger. It allows real-time code debugging and analysis using RTDX (*Real Time Data Exchange*) between the DSK and the host PC using a USB interface. A JTAG interface allows the DSP to be assessed by the PC. After completing this laboratory, the student is expected to:

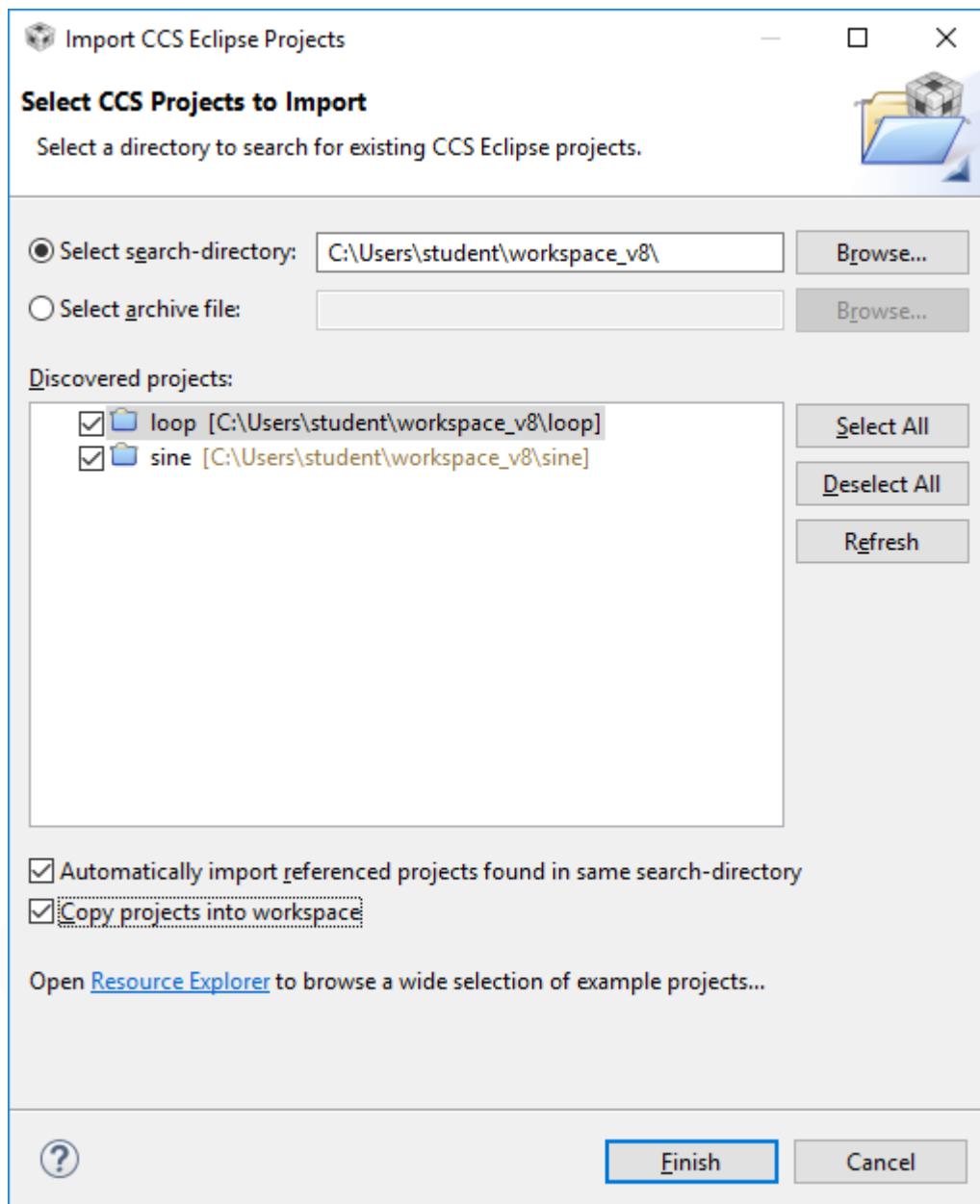
1. Be able to create a CCS development project.
2. Have understood the program control mechanism via software interrupts.
3. Have understood the communication protocol between the DSP and the audio CODEC which is responsible for the digitalization of analog input signals (ADC function) and for converting digital words into analog signals (DAC function).
4. Be able to develop simple programs using only fixed-point arithmetic.
5. Have developed a numerically controlled sinusoidal oscillator (NCO) which will be used in the next part of the laboratory project.

## Launching CCS and importing projects

The CCS software suite is already installed in the laboratory computers and shortcuts for it are available in the desktop. The CCS main directory is `c:\ti\` but all projects should be created in the workspace directory `c:\Users\student\workspace_v8`. To start your work launch the CCS application (a shortcut named `CCS_8.3.1` is available at the computer desktop). After the application launches change the workspace as follows: goto *File* → *Switch Workspace*. A window will pop up indicating the directory `c:\...\workspace_v8\`. Append the string “\group#” where # is your group number and click OK.



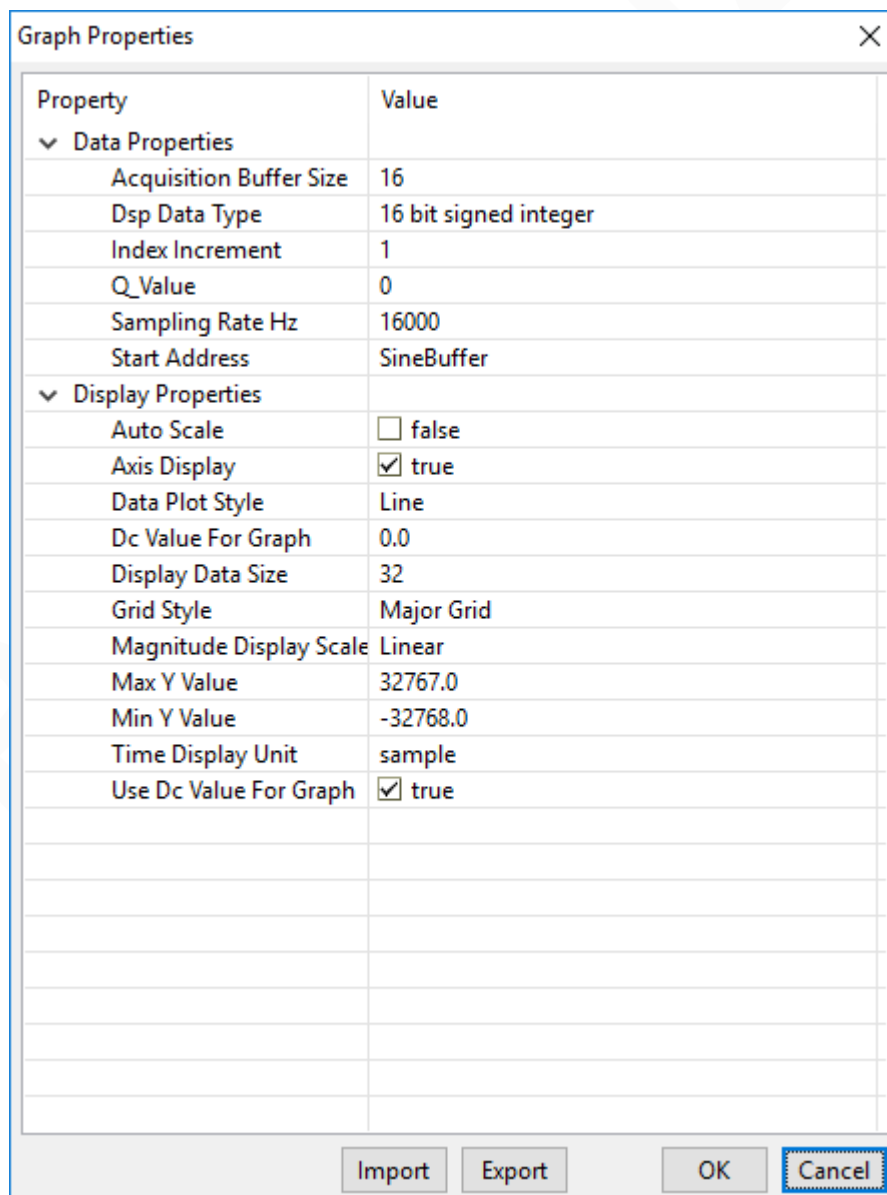
The CCS program will restart and will also create the directory `c:\..\workspace_v8\group#`. This will be your working directory during this course. The first time you start CCS (in each new laboratory session) be sure to change the workspace so CCS will read all your projects. Next you need to add the two available projects to your workspace. Go to *Project* → *Import Existing CCS Eclipse Project*. Fill the pop-up windows as shown in the next image. Click Finish.



At this point, the two projects should appear in the *Project Explorer* window of CCS. Your laboratory instructor will give you further indications on how to proceed.

## Demonstration Project: Sine

Open the project Sine. Check the project file structure and select the source file `main.c`. Study this program in detail. Note that only integer-type variables are used (the appendix contains all available data types and their characteristics). Build the project (*Project* → *Build Project*) and check that there are no errors. Click *Run* → *Debug* to enter debug mode and run the program. Connect the left and the right channel outputs to the oscilloscope and observe the signals. Stop program execution. Place a breakpoint in one line of the program and watch some variables, in particular the variable `gain`. Run the program, checking that it stops at the breakpoint. Check the variable values in the watch window. Measure the output frequency and explain its value. In the CCS main menu, go to *Tools* → *Graph* → *Single Time* and fill the parameters as shown below:



The image shows a 'Graph Properties' dialog box with a table of properties. The table is divided into two sections: 'Data Properties' and 'Display Properties'. The 'Data Properties' section includes 'Acquisition Buffer Size' (16), 'Dsp Data Type' (16 bit signed integer), 'Index Increment' (1), 'Q\_Value' (0), 'Sampling Rate Hz' (16000), and 'Start Address' (SineBuffer). The 'Display Properties' section includes 'Auto Scale' (false), 'Axis Display' (true), 'Data Plot Style' (Line), 'Dc Value For Graph' (0.0), 'Display Data Size' (32), 'Grid Style' (Major Grid), 'Magnitude Display Scale' (Linear), 'Max Y Value' (32767.0), 'Min Y Value' (-32768.0), 'Time Display Unit' (sample), and 'Use Dc Value For Graph' (true). At the bottom of the dialog are buttons for 'Import', 'Export', 'OK', and 'Cancel'.

| Property                | Value                                    |
|-------------------------|--|
| ▼ Data Properties       |  |
| Acquisition Buffer Size | 16                                       |
| Dsp Data Type           | 16 bit signed integer                    |
| Index Increment         | 1  |
| Q_Value                 | 0  |
| Sampling Rate Hz        | 16000                                    |
| Start Address           | SineBuffer                               |
| ▼ Display Properties    |  |
| Auto Scale              | <input type="checkbox"/> false           |
| Axis Display            | <input checked="" type="checkbox"/> true |
| Data Plot Style         | Line                                     |
| Dc Value For Graph      | 0.0                                      |
| Display Data Size       | 32                                       |
| Grid Style              | Major Grid                               |
| Magnitude Display Scale | Linear                                   |
| Max Y Value             | 32767.0                                  |
| Min Y Value             | -32768.0                                 |
| Time Display Unit       | sample                                   |
| Use Dc Value For Graph  | <input checked="" type="checkbox"/> true |

This will open a docked Graph windows showing the contents of `SineBuffer[]`. Compare this graph with the oscilloscope trace. Change the value of the variable gain from 32 to 33 and watch again the graph and the resulting output signal. Can you explain what happens? Explain why the sine table in this project is not well designed.

**Report:** CCS graphs and oscilloscope traces showing the output signal when the gain is set to 32 and to 33.

## Demonstration Project: loop

This project is very important because it will serve as a template upon which your own projects will be based. Open the project loop and select the source file `main.c`. Study this program in detail, particularly the way program control is accomplished by using the interrupts generated by the serial port which communicates with the AIC3204 codec. Use another set of signal cables to connect an external signal generator to the line-in DSK input. Generate a sinusoidal signal with no more than 1.5V peak-to-peak amplitude and frequency 1000Hz. Observe the signal at the line-out output. Change the signal frequency from DC to 8 KHz and see what happens. Change the signal waveform to a square wave and vary its frequency. Explain the behavior of the output signal as the frequency changes.

**Report:** Oscilloscope trace showing the input square-wave signal and the output signal for two frequencies:  $f_0 = 1$  kHz and  $f_0 = 3$  kHz.

## How to create a new project?

You may create a new project going to *Project → New CCS Project*. However, there is a lot of work involved in this process because you will need to enter a significant amount of information on the processor type and supporting files. An expedite way to create a new project based on project loop is as follows:

- 1) If your current workspace (`c:\..\workspace_v8\group#`) is active, select the project loop, right-click and select the *Rename* option. Enter the desired name for the project in the pop-up window. A new directory with the project name will be created in the workspace and the new project will appear in the Project Explorer window.
- 2) Select the new project, right-click and select *Properties*. Under *C5500 Linker → Basic Options* replace the `.out` and `.map` old file name with the new project name.

### Important note

In this and all subsequent projects, numbered guidelines followed by a letter **(T)**, **(L)**, **(S)**, (or a combination of these) are given to the students. The meaning of this classification is:

**(T), (S):** Questions must be studied and solved **before** the laboratory class.

**(L):** Work to be done during the laboratory class.

## Project #1 – Numerically controlled oscillator (NCO)

**Note:** This project will be used in the second project, Digital Tracking

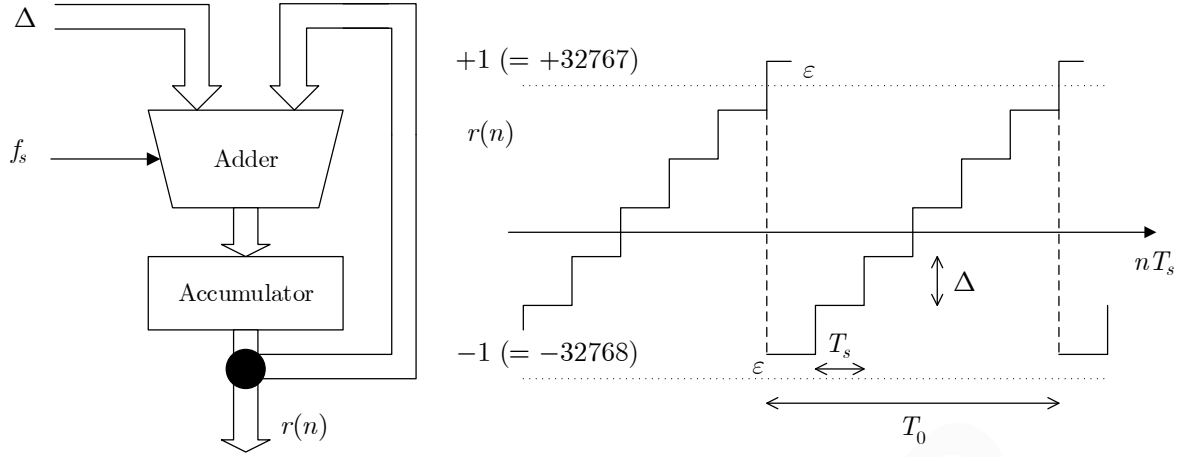
Based on the project loop, create a new project to implement a numerically controlled oscillator (NCO). This project should only use `short` and `integer` variables (signed or unsigned). The NCO characteristics are listed in Table 1.

**Table 1:** Relaxation oscillator specifications

| Parameter            | Symbol          | Value  | Remarks  |
|----------------------|-----------------|--------|--|
| Sampling frequency   | $f_s = 1 / T_s$ | 16 kHz | Pre-programmed                                       |
| Minimum frequency    | $f_{\min}$      | 2 kHz  | Frequency when the input signal amplitude is minimum |
| Maximum frequency    | $f_{\max}$      | 6 kHz  | Frequency when the input signal amplitude is maximum |
| Right output channel |                 |        | Oscillator output, $r(n)$                            |
| Left output channel  |                 |        | Oscillator sine output, $y(n)$                       |

The development sequence should be as follows:

1. **(L)** Develop a relaxation oscillator (ramp integrator) using a 16-bit signed integer variable (short type). This state variable,  $r(n)$ , should be incremented by a value  $\Delta$  (another short variable) every sampling interval. Due to the circularity of the two's complement representation, the state variable will eventually overflow and take a value equal or close (and above) to the minimum possible, see Figure 3.

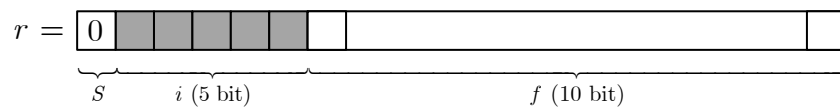


**Figure 3:** Relaxation oscillator principle and waveform.

2. **(T)** As can be seen from this figure, the variable  $\Delta$  controls the oscillator frequency and should be declared in your program. Determine the expression for the output frequency  $f_0 = 1 / T_0$  of  $r(n)$  as a function of  $\Delta$  and  $f_s = 1 / T_s$ . Use this expression to determine the value of  $\Delta$  required to have  $f_0 = 4$  kHz (central frequency) which will be named  $\Delta_0$ . Initialize  $\Delta = \Delta_0$  in your program. Compute the two other values of  $\Delta$  required to generate  $f_{\min}$  and  $f_{\max}$ .
3. **(L)** Send  $r(n)$  to the right output channel and observe it. Explain the differences between what is observed and the waveform in Figure 3. Change  $\Delta$  to the other two values and measure the output frequency.
4. **(T)** Set up a look-up-table (LUT) in data memory with 32 positive values of half a cycle of the sine function using the most accurate representation format  $Q_{15}$ :

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

5. **(L)** Use the oscillator state variable  $r$  to index the LUT and obtain the sine values. The procedure is indicated in Figure 4. From the 16-bit state variable ( $r$  in this figure), extract the required 5 bits (integer part, variable  $i$ ) to address the LUT and get the sine value  $y(n) = \text{LUT}(i)$ .



**Figure 4:** Decomposition of the variable  $r = (-1)^S \times i.f$  into sign  $S$  bit, integer part,  $i$  bits, and fractional part,  $f$  bits.



Note that the table only has *half* cycle of the sinewave so you must manage separately the sign of  $y(n)$ . Send the sinewave  $y(n)$  to the left output channel and observe it.

6. **(L)** Create a new variable (gain) to control the amplitude of the sinusoidal signal by multiplying it by the sine sample  $y(n)$ . Recall that the multiplication of two 16-bit integer variable gives a result with 32 bit which needs to be truncated down to a 16-bit word. Vary the amplitude variable and watch the output signal.

**Report:** Oscilloscope trace showing the oscillator signals, ramp signal and sinewave, for a frequency  $f_0 = 4$  kHz and a half-scale amplitude (i.e., gain equal to  $\frac{1}{2}$ ).

At this point you have developed a simple NCO which produces a sinusoidal signal with an instantaneous frequency proportional to the variable  $\Delta$ . This oscillator must now be improved in three ways: 1) its frequency must be programmed by a modulation signal represented by  $\varepsilon(n)$  and 2) its output quality should be improved by means of linear interpolation. To accomplish this, follow these guidelines:

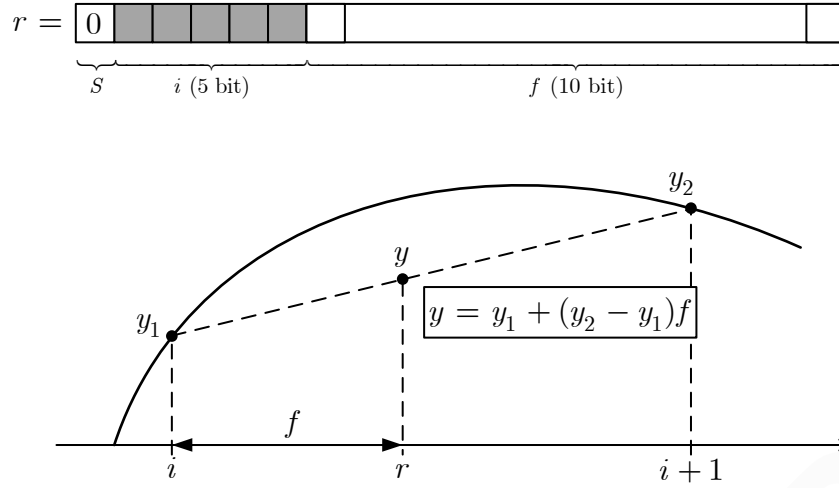
7. **(T)+(L)** Recall that the variable  $\Delta$  controls the NCO frequency. Remove the amplitude control implemented in 6) or set the amplitude to the maximum possible. Now modify the program so that the oscillator frequency is controlled by the input signal  $\varepsilon(n)$ : the variable which controls the oscillator *instantaneous* frequency, should vary linearly with the input signal amplitude, that is

$$\Delta = \Delta_0 + k \cdot \varepsilon(n). \quad (2)$$

In this equation  $k$  is a constant which should be determined for the NCO to have the required frequency range: when  $\varepsilon(n) = 1 = (+32767)$  the output frequency should be  $f_0 = 6$  kHz and when  $\varepsilon(n) = -1$  it should be  $f_0 = 2$  kHz. Note that this NCO is in fact a *frequency* modulator. Test the oscillator by injecting a low-frequency square-wave in the right input channel of the DSK (this will be  $\varepsilon(n)$  in your program) and observing the frequency-modulated output on  $y(n)$ . Use the function generator to produce the analog square-wave signal  $\varepsilon(t)$ .

**Report:** Oscilloscope trace showing the input signal after sampling (i.e., returned from the DSP to the codec in one channel),  $\varepsilon(n)$ , and the frequency-modulated output signal  $y(n)$ .

8. **(L)** Improve the quality of the sinusoidal oscillator by using linear interpolation as shown in Figure 5. Use the remaining 10 bit from the ramp  $r$  to form the remainder (or fraction)  $f$ , and represent it in  $Q_{15}$  format. Read two consecutive values  $y_1 = \text{LUT}(i)$ ,  $y_2 = \text{LUT}(i+1)$  and then compute the sinusoidal interpolated value as  $y = y_1 + (y_2 - y_1)f$  in  $Q_{15}$  format.



**Figure 5:** Look-up table interpolation scheme.

To test the effect of the interpolation, disable the frequency modulation by setting  $k = 0$  in equation (2) temporarily. Use the oscilloscope FFT feature to watch and compare the spectrum of the output signal from the two sinusoidal oscillators developed (with and without interpolation) for  $f_0 = 4$  kHz. Change the frequency to  $f_0 = 3.9991$  kHz (recompute  $\Delta_0$  for this new frequency) and watch again. Explain the spectrum differences in these two situations.

9. (T) Explain why the previous interpolation equation  $y = y_1 + (y_2 - y_1)f$  can in fact be computed using  $Q_{15}$  format.

**Report:** Oscilloscope trace showing the spectrum of the generated sinewave with and without interpolation for a frequency  $f_0 = 3.9991$  kHz.

## Appendix I – TMS320C5515 DSK ADC and DAC voltages

The TMS320C5515 DSK has a stereo AIC3204 codec which provides two 16-bit DACs (outputs) and two 16 bit ADC (inputs) in two's complement. The DSK is powered from 3.3 V and the analog ground voltage (GND) is internally set to 0.9 V with a maximum excursion of  $\pm 0.7$  V (swing from 0.2 to 1.6 V). Because there are series decoupling capacitors both at the ADC inputs and DAC outputs, the voltages at the BNC connectors are as listed in Table 2, depending on the 16 bit word written to the DACs or read from the ADCs.

**Table 2:** DSK input and output approximate voltages (at BNC connectors)

| Word read/written (hex) | Input voltage (V) | Output voltage (V) |
|-------------------------|-------------------|--------------------|
| 0x7FFF                  | 0.7 (maximum)     | -0.7 (minimum)     |
| 0x0000                  | 0.0 (GND)         | 0.0 (GND)          |
| 0x8001                  | -0.7 (minimum)    | 0.7 (maximum)      |

At the inputs the maximum voltages of  $\pm 0.7$  V (1.4 V<sub>pp</sub>) should not be exceeded, otherwise the ADCs will saturate and may suffer permanent damage. In addition, the DAC outputs go through inverting amplifiers, which is why the values in the DAC column are inverted. Note that this means that the signals at the DAC outputs appear **inverted** in the oscilloscope.

Also note that because of the decoupling capacitors (which form a highpass filter), the input/output user frequencies below 5 Hz (approximately), including DC values cannot read or observed.

## Appendix II – Data types

| Type   | Size    | Representation | Range                          |                            |
|--|---------|----------------|--------------------------------|----------------------------|
|  |         |                | Minimum                        | Maximum                    |
| char, signed char                                      | 8 bits  | ASCII          | −128                           | 127                        |
| unsigned char  | 8 bits  | ASCII          | 0                              | 255                        |
| short  | 16 bits | 2s complement  | −32 768                        | 32 767                     |
| unsigned short   | 16 bits | Binary         | 0                              | 65 535                     |
| int, signed int  | 32 bits | 2s complement  | −2 147 483 648                 | 2 147 483 647              |
| unsigned int   | 32 bits | Binary         | 0                              | 4 294 967 295              |
| long, signed long                                      | 40 bits | 2s complement  | −549 755 813 888               | 549 755 813 887            |
| unsigned long  | 40 bits | Binary         | 0                              | 1 099 511 627 775          |
| long long, signed long long                            | 64 bits | 2s complement  | −9 223 372 036 854 775 808     | 9 223 372 036 854 775 807  |
| unsigned long long                                     | 64 bits | Binary         | 0                              | 18 446 744 073 709 551 615 |
| enum   | 32 bits | 2s complement  | −2 147 483 648                 | 2 147 483 647              |
| float  | 32 bits | IEEE 32-bit    | 1.175 494e−38 <sup>†</sup>     | 3.40 282 346e+38           |
| double   | 64 bits | IEEE 64-bit    | 2.22 507 385e−308 <sup>†</sup> | 1.79 769 313e+308          |
| long double  | 64 bits | IEEE 64-bit    | 2.22 507 385e−308 <sup>†</sup> | 1.79 769 313e+308          |
| pointers,<br>references,<br>pointer to data<br>members | 32 bits | Binary         | 0                              | 0xFFFFFFFF                 |

<sup>†</sup> Figures are minimum precision.