AD9082/AD9081/AD9988/AD9986 MATLAB Application User Guide

Table of Contents

HARDWARE NEEDED

- **ADS9 FPGA Board**
- **AD9XXX + EVALUATION BOARD**

SOFTWARE NEEDED

- **Analysis, control, evaluate (ACE) software**
- **DPGDownloaderLite software (Included in ACE installation)**
- **MATLAB**

DOCUMENTATIONS

AD9XXX.DPG_Lite_Example.m is a functional MATLAB automation example that free user from repetitively using ACE and DPGLite. This example automates generating a tone and sends it using DPGLite.

AD9XXX.m file is used to configure and control ACE (Software that allows the user to set up the MxFE® in various modes and to capture analog-to-digital converter (ADC) data for analysis).

NDPG9XXX.m file is used to configure and control DPGDownloaderLite (Software that generates and transmits vectors to the digital-to-analog converters (DACs), which can then be sent to a spectrum analyzer for further analysis).

Auotmation.m file used to run the automation with the same configuration and set-up repeatedly.

Plot.m file is used to plot a fast fourier transform (FFT) from the Excel file generated by the data gathered from the automation.

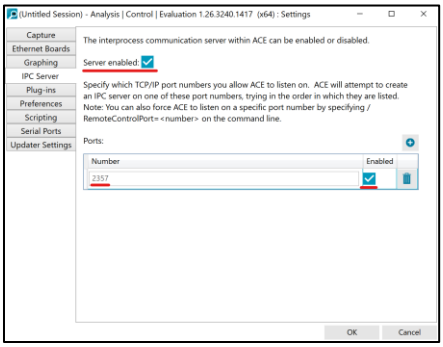
IMPORTANT

Make sure to open ACE and apply configuration to the hardware before running the automation. ACE should be up the whole time for the automation to successfully function.

Basic Setup

Enable and Match IPC Port

1. Open **setting** on the bottom left corner of ACE.
2. Select **IPC Server** tab and check the **Server enabled** box.
3. In the **Ports** section of **IPC Server** tab, confirm the number displayed and make sure the enabled box is checked.



4. Open up *AD9XXX.DPG_Lite_Example.m*, in line 26, confirm that the **IPC_Port** number match with the number displayed on ACE setting, if not, change the number in *AD9XXX.DPG_Lite_Example.m* file manually.

```
% AD9xxx(IPC_Port, Subsystem_#, DAC_CLOCK, ADC_CLOCK)
mxfe = AD9xxx(2357,1,dacclock,adcclock);
```

MATLAB File Setup

AD9XXX.m

a. In line 41 and 42, replace the AD9xxxs on the end of each command with the name of user's AD9xxx.

```
obj.chip_cont=['\system\Subsystem_' num2str(subsys) '\AD9082-FMCA-EBZ\AD9082'];
obj.brd_cont=['\system\Subsystem_' num2str(subsys) '\AD9082-FMCA-EBZ'];

obj.ads9_cont=['\system\Subsystem_' num2str(subsys) '\ADS9-V2'];
```

Make sure this matches what you see in ACE:

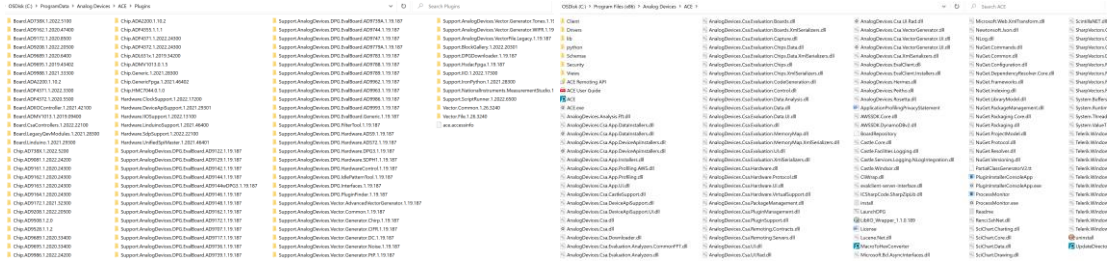
b. In line 608, replace `C:\Users\***\AppData\Local\Analog Devices\ACE\ExportData\AD9082\.acesamples` with the path user desires, keep *.acesamples* at the end of the path.

```
obj.client.ExportStoreDataToFile("C:\Users\***\AppData\Local\Analog Devices\ACE\ExportData\AD9082\.acesamples", "acesamples", "", "");
```

NDPG9XXX.m

a. Open the directories as mentioned in line 28 and 29 (C:\ProgramData\Analog Devices\ACE\Plugins & C:\Program Files (x86)\Analog Devices\ACE).

b. Confirm and match the highlighted paths from line 32-41 with directories opened. Each path should have a corresponding **.dll** file in the folders within the directories.



c. If the highlighted paths do not match, change the paths in the *NDPG9XXX.m file*.

```
methods
    function self = NDPG9xxx()
        PluginDir='C:\ProgramData\Analog Devices\ACE\Plugins';
        ACEDir='C:\Program Files (x86)\Analog Devices\ACE';

        try
            NET.addAssembly([PluginDir '\Hardware.SdpSupport.1.2022.22100\lib\sdpApi1.dll']);
            NET.addAssembly([ACEDir '\AnalogDevices.Csa.dll']);
            NET.addAssembly([ACEDir '\AnalogDevices.Csa.UI.dll']);
            NET.addAssembly([PluginDir '\Support.AnalogDevices.DPG.HardwareControl.1.19.187\lib\net452\AnalogDevices.DPG.HardwareControl.dll']);
            NET.addAssembly([ACEDir '\AnalogDevices.Csa.Hardware.dll']);
            NET.addAssembly([PluginDir '\Support.AnalogDevices.Vector.Common.1.19.187\lib\net452\AnalogDevices.Vector.Common.dll']);
            NET.addAssembly([PluginDir '\Support.AnalogDevices.DPG.Interfaces.1.19.187\lib\net452\AnalogDevices.DPG.Interfaces.dll']);
            NET.addAssembly([PluginDir '\Support.AnalogDevices.DPG.EvalBoard.AD9986.1.19.187\lib\net452\AnalogDevices.DPG.EvalBoard.AD9986.dll']);
            NET.addAssembly([PluginDir '\Support.AnalogDevices.DPG.Hardware.ADS9.1.19.187\lib\net452\AnalogDevices.DPG.Hardware.ADS9.dll']);
            NET.addAssembly([PluginDir '\Support.AnalogDevices.DPG.Hardware.ADS9.1.19.187\lib\net452\AnalogDevices.DPG.Hardware.ADS9.dll']);

        %catch ex
            %ex.ExceptionObject.LoaderExceptions.Get(0).Message
        end
```

Basic setup complete, the user may now run the main file (*AD9XXX.DPG_Lite_Example.m*).

Import Functions

There are functions created in *AD9XXX.m* and *NDPG9XXX.m* files to help customize the automation.

- To call functions in *AD9XXX.m*, add a line of command "mxfe.*function name*()" to *AD9XXX.DPG_Lite_Example.m file*. This would import the corresponding function in *AD9XXX.m* file into the automation.

```
% dummy read. improves stability
mxfe.captureAllChannels();
```

- To call functions in *NDPG9XXX.m*, add a line of command "dpg.*function name*()" to *AD9XXX.DPG_Lite_Example.m file*. This would import the corresponding function in *NDPG9XXX.m* file into the automation.

```
%Stop Vector from playing
dpg.stop();
```

- Some functions might require the user to input values for variables manually.

```
%setNCOfreq(Freq, 'nco3, nco2, nco1, nco0')
%Freq  - NCO frequency
%mask  - Which NCOs to set their frequency.
%        '1111' will write to all ncos
%        '1010' will write to NCO1 and NCO3
%        '0101' will write to NCO0 and NCO2
mxfe.ADCsetNCOfreq(500e6, '1111')
```

Function List

To use a function, user must call it in the *AD9XXX.DPG_Lite_Example.m file.*

Mask variable is created for user to adjust each subcomponent individually. User will need to enter a number sequence to decide which subcomponent they will adjust.

For instance, **mxfe.ADCsetFineNCOfreq(0, '11111111')** will adjust all 8 NCO. **'01010101'** will adjust NCO 1, 3, 5, and 7, leave NCO 0, 2, 4, and 6 unchanged.

The digit of the number sequence corresponds with the number label given to each subcomponent.

Some components require an 8 digit mask, while some requires only 4 digit, check the comments in *AD9xxx.m* file for requirements.

mxfe.ADCsetFineNCOfreq(*freq, mask*)

This function configures the Fine NCO frequency of ADC.

- Replace *freq* with the frequency user desires.
- Replace *mask* with an 8 digit number sequences (Remember to enclose the sequence in ' ').

mxfe.DACsetFineNCOfreq(*freq, mask*)

This function configures the Fine NCO frequency of DAC.

- Replace *freq* with the frequency user desires.
- Replace *mask* with an 8 digit number sequences (Remember to enclose the sequence in ' ').

mxfe.DACsetDCOffsetGain(*gain, mask*)

This function configures the DDC Offset gain of DAC.

- Replace *gain* with the decibel user desires.
- Replace *mask* with an 8 digit number sequences (Remember to enclose the sequence in ' ').

mxfe.ADCFineDDCGain(*ToF, mask*)

This function configures the Fine DDC gain of ADC.

- Replace *ToF* with the 0 or 1 (0 represent off[0dB], 1 represent on[6dB]).
- Replace *mask* with an 8 digit number sequences (Remember to enclose the sequence in ' ').

mxfe.ADCCoarseDDCGain(*ToF, mask*)

This function configures the Coarse DDC gain of ADC.

- Replace *ToF* with the 0 or 1 (0 represent off[0dB], 1 represent on[6dB]).
- Replace *mask* with an 4 digit number sequences (Remember to enclose the sequence in ‘ ’).

mxfe.DACFSCSetting(*mamp, mask*)

This function configures the Full-Scale current of DAC.

- Replace *mamp* with the microampere user desires (1000000 microampere = 1 ampere).
- Replace *mask* with an 4 digit number sequences (Remember to enclose the sequence in ‘ ’).

mxfe.SampleCount(*count*)

This function configures the number of samples that the MxFE will capture.

- Replace *count* with the number of samples user desires (number must be 2^x, where $0 \leq x \leq 20$).

Repeat Automation

To run the *AD9XXX.DPG_Lite_Example.m* file multiple times without changing the desired configuration, the user may run the *Automation.m* file to gather data.

```
x = 10;
y = 0;

while y < x
    run("AD9081_DPG_Lite_Example.m")
    y = y + 1;
    pause(3)

    display(y)
end
```

Pull up *Automation.m* file.

In line 4, set **x** = (# of times user wants the automation to run).

Replace the **'filename'** in line 8 if it does not match with user's *AD9XXX.DPG_Lite_Example.m*.

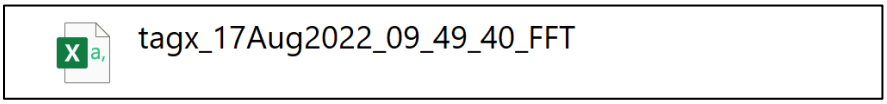
User may now run the *Automation.m* file, the gathered data sets will appear in the path user presets in *AD9xxx.m*.

Plotting FFT from Excel

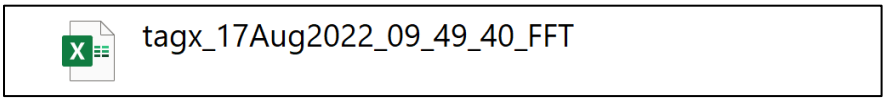
User may also plot FFT from Excel data gathered by the automation in MATLAB.

MATLAB only plots .xlsx excel files, but our automation gathers .csv excel files. Thus, user need to change the file format.

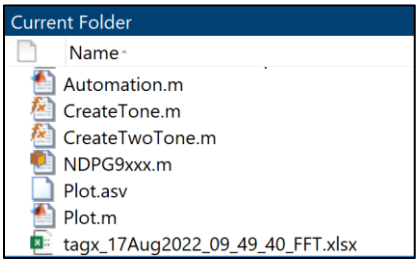
To start, find the FFT.csv file in the path user presets in *AD9xxx.m*.



Open the file in Excel and save as Excel Workbook (*.xlsx) file.



Move the new FFT.xlsx file to the same directory as *Plot.m* file (User may double click on the Current Folder space and select "Open Current Folder in Explorer to find the directory).



Pull up *Plot.m* file.

```
rawTable = readtable('tagx_17Aug2022_09_49_40_FFT.xlsx');

%Replace the line after rawTable. with the header of the column you want to
%plot
x = rawTable.Frequency;
y = rawTable.Amplitude;
```

Replace the **'filename'** in the parenthesis following "readtable" in line 3 with user's FFT.xlsx.

User may also change the line after rawTable to match with the input/output they want to use. Simply replace the line with the header of the column user want to use as input/output.

User may now run the *Plot.m* file, a MATLAB window should pop up with the FFT figure.