



ADCTest User Guide
0.2, version 235
2018-02-05

Contact information

AVP | <https://www.WeAreAVP.com> | info@WeAreAvp.com

GitHub repository

<https://github.com/WeAreAVP/ADCTest>

Download pages for application

Windows

<https://www.weareavp.com/wp-content/uploads/2019/09/ADCTestSetup.zip>

Note

Please help refine ADCTest further by reporting all bugs to
<https://github.com/WeAreAVP/ADCTest/issues>

Change Log	2
Definitions	2
Creating a New Project	3
Performing Calibration	3
Performing a Test Suite	5
Opening an Existing Project	10
Inside an .avt File	10
Configuration File	15
Test Results File	16
Copyright and Licensing	17
Appendix A: Background and Context	18

Change Log

0.2 Version 235

- Changed from two-channel default operation to single-channel default operation. The previous version of ADCTest generated and captured two-channel signals/files but only analyzed the channel number associated with the test. This version generates and captures a single-channel signal/file, providing smaller files and more flexibility.
- Incorporated channel routing. This version of ADCTest adds the ability to select which channel of the DAC you would like the signal to be generated from and which channel of the ADC you would like the signal to be received on. This was enabled by transitioning to a single-channel default operation as well.
- Added test mode, response file naming, and audio routing to the main user interface to make these edits easier. Previously this could only be done by editing the appropriate XML file.
- Added the concept of projects. To begin, a user creates a new project and identifies the directory to store all of the project files including signal files, response files, graphic files, and the .avt file containing all of the project test parameters.
- When creating a new project the user is prompted to input the project name, the operator name, the audio DAC name/identifier, and the audio ADC name/identifier, after the project is created the user can also enter and save notes related with the project. All of this information is saved as part of the project enabling more effective management of test results over time.
- Changed the XML file extension for a project to .avt from XML.
- Ability to open past test results. Previously without the concept of a project, users could only open past test results by re-analyzing the response files. Now when opening a project ADCTests will read the graphic .xml files for displaying the results of that test and avoid the time consuming task of re-analyzing.
- Improved stability and reliability of audio driver performance. On certain setups the last version of ADCTest would experience audio driver instability. This has been significantly improved, but not solved completely.
- Added “retest” result. Because the audio driver stability issue was not completely solved, and because these types of errors will always exist in various circumstances as evidenced by our work on exploration and analysis of interstitial errors, this version of ADCTest has error detection. If an error is detected in the test and measurement process, ADCTest will give a test result of “retest”. This single test can then be performed again.

Definitions

Signal File: The .wav file that is generated by ADCTest to send to the output of the DAC and into the input of the ADC.

Response File: The .wav file that is generated as a result of capturing the test signal audio that is run through the ADC for testing.

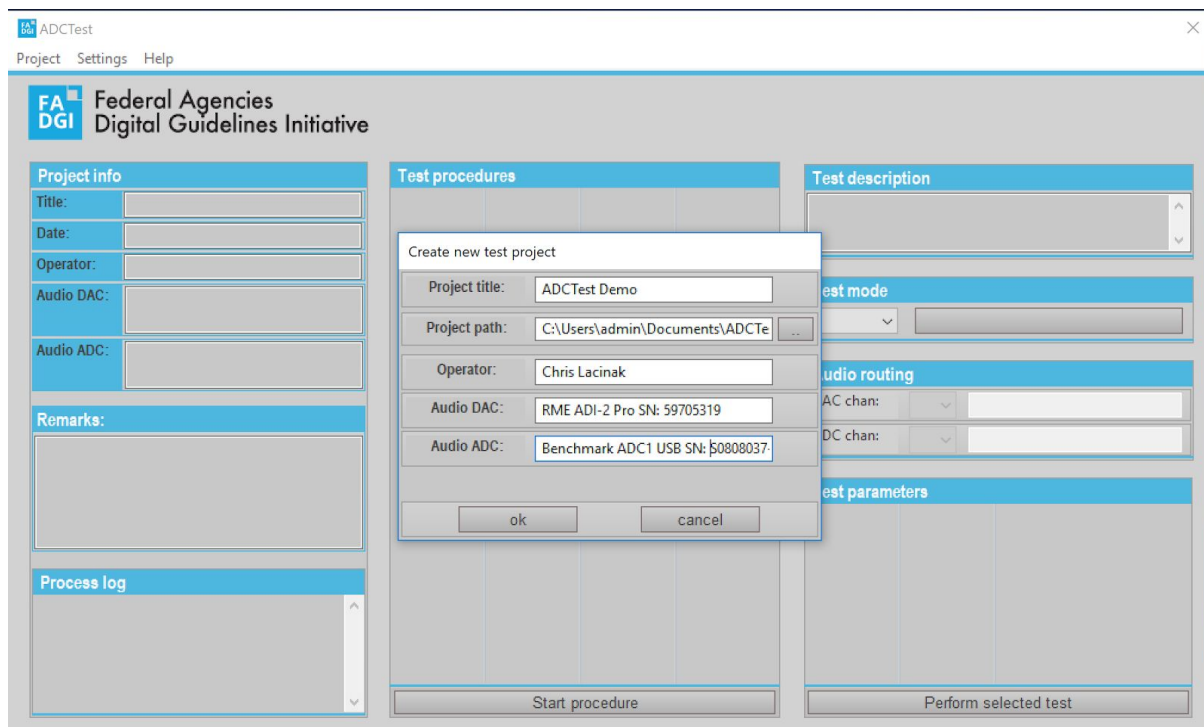
Results File: The .xml file that contains the results of analyzing the response file.

Project File: The .avt file containing the project, configuration and test parameter information for a particular project. The .avt extension is made up for ADCTest. This file contains XML.

Creating a New Project

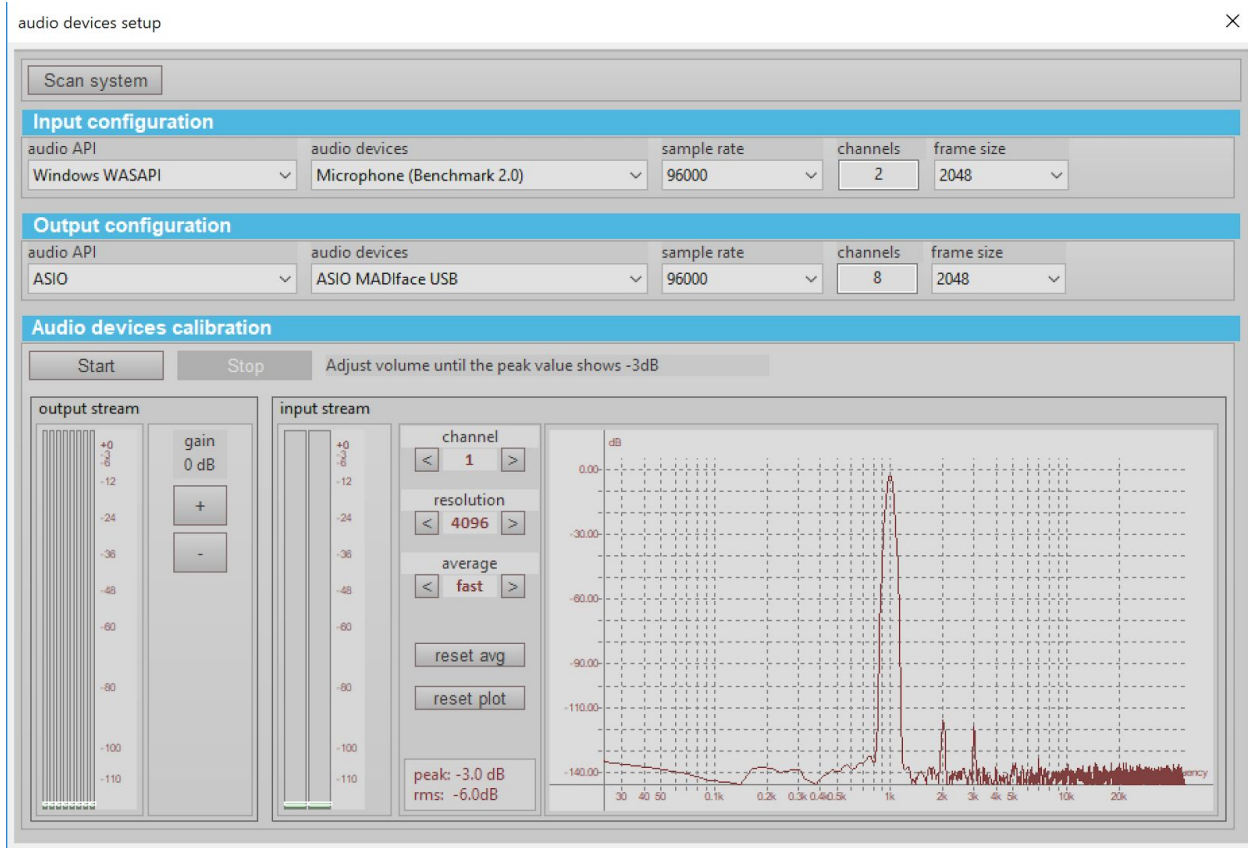
Upon opening, the main ADCTest window will be blank. Select “New Project” from the the “Project” menu. This will bring up a dialog window, prompting the user to fill out the:

- Project title which will create a project folder with that name
- Project file path where you would like to save the project folder
- Operator performing the test
- Audio DAC being used in the test
- Audio ADC being used in the test



Performing Calibration

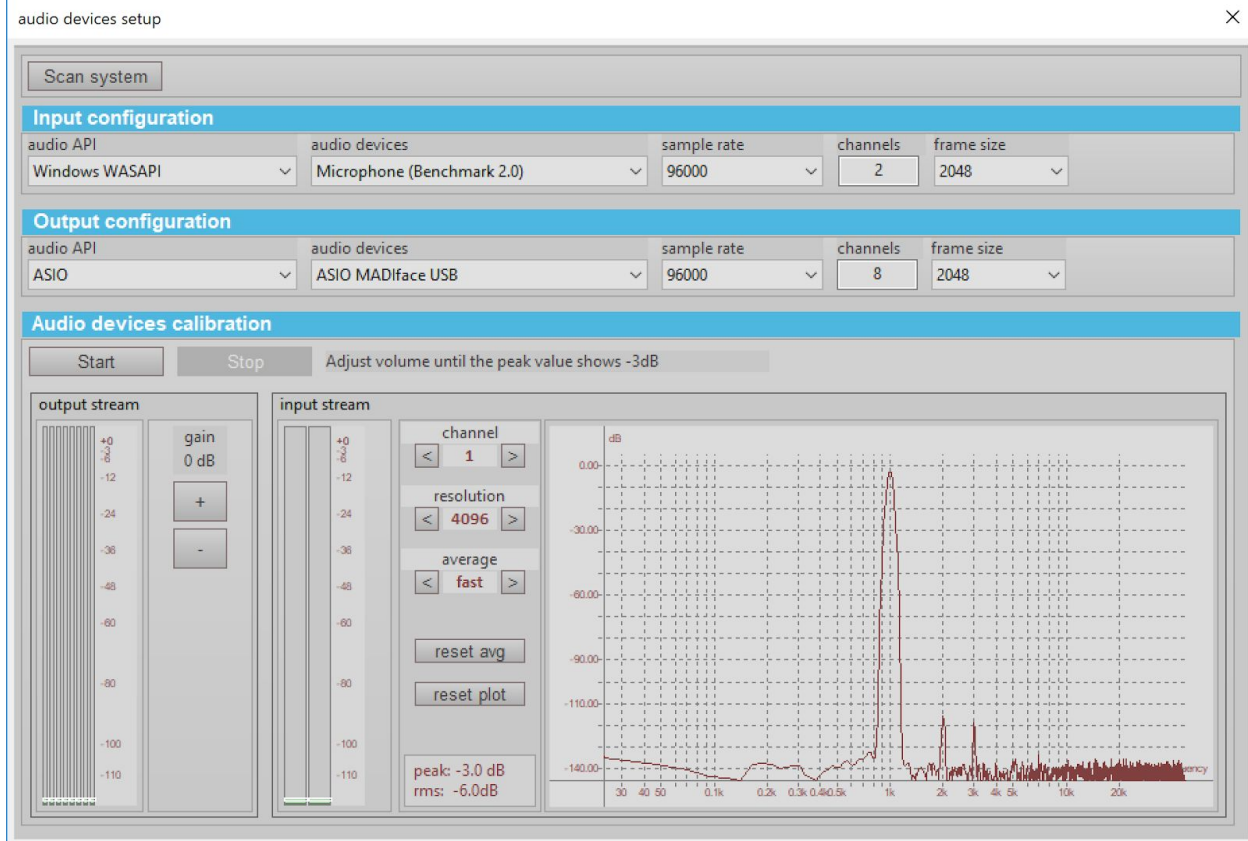
To perform calibration, select “Devices” from the “Settings” menu within the application.



Select the audio drivers you would like to use for the ADC and DAC using the dropdown menu under “audio API”. Note that different audio drivers have different performance levels and this choice may impact your results.

Once the audio drivers are selected, the user may select the ADC and DAC devices under “audio devices” as well as the sample rate under the “sample rate” dropdown. “channels” will automatically populate based on the audio device selection. “frame size” defaults to 2048 which should be left alone unless there are audio driver and performance issues related to buffering. If you do not see your audio device, select the “Scan system” button at the top of the window to re-register audio drivers and devices.

After the audio drivers, ADC, DAC and sample rate are selected, select the “Start” button under “Audio devices calibration”. This will send a 1 kHz signal out of the output of the DAC and to the input of the ADC. Activity should be seen at the “output stream” level meters, “input stream” level meters, and on the FFT graph.



The user must adjust each channel to -3dB using some combination of the level meters on the DAC, ADC and gain within ADCTest. The graph and reporting of peak levels default to channel 1. When levels are adjusted so that channel 1 is set to somewhere between -2.9dB and -3.1dB the peak level reporting will turn green. After channel 1 is set, adjust the other channels in the same manner. Be sure not to overload the output of the DAC or the input of the ADC when performing calibration. The meters and FFT graph should provide visual feedback to help with performing calibration successfully. The resolution and average parameters in the input stream pane are simply for controlling the FFT display and do not impact the calibration.

Once all channels are set at -3dB select “Stop” under “Audio devices calibration” and close the audio devices setup window using the “x” in the top right corner of the window. At this point all calibration settings are saved.

Performing a Test Suite

Overview

A test suite is a series of individual tests performed in sequence. The default test suite that is loaded upon launching the application is the FADGI test suite. The default settings are stored in default.avt which is located next to the executable file for ADCTest. Note that to alter this file it must be stored in a location where the user has permissions to write data. The settings for a given project are stored in [project_name].avt file within the project directory.

Clicking on each test within the suite will bring up the associated information in the panels on the right side of the main window. Altering the information will change the information for the selected test only.

Enable/Disable

The user may enable or disable tests by right/control clicking on the middle column within the “Test procedures” window and selecting “enable” or “disable”.

Test mode

There are two test modes under the “Test mode” panel dropdown.

“Normal” test mode means that the signal will be generated, the response files will be created, and the response files will be analyzed all at once. In “Normal” mode, selecting the button to the right of the dropdown menu will allow the user to enter an alternate name for the response file.

“Offline” test mode means that the response files exist already and will only be analyzed. In “Offline” mode, selecting the button to the right of the dropdown menu will allow the user to select the file that they want to analyze. Otherwise the application will expect the default file name within the project directory.

For more on the use-cases associated with these two modes, read the Use-Cases and Application Architecture section of the Appendix.

Audio routing

Within the “Audio routing” panel there is one dropdown menu to select the channel that the signal should be generated from on the DAC and one for the channel the signal should be received on on the ADC. This is only functional when using “Normal” test mode since “Offline” only analyzes existing files.

The dropdown menu will contain an option for each channel on the selected device. The device that is shown is dictated by the device selected within the “Devices” window. 0 is the first channel on the device and the dropdown will contain a number for each channel on the device.

Editing a test suite

If users want to change the default test settings or the project test settings beyond what is available through the user interface, they may do so by editing the default.avt or [project_name].avt respectively.

User actions

ADCTest allows for pauses in test suites to enable user actions to be performed. The default test suite contains two of these in order to perform the user action required for the cross-talk test. See the Cross-Talk test section for more information on the required action for this test. For the performance of offline test suites these user actions can either be disabled or deleted.

Running a test suite

Once everything is configured properly and the project is saved, begin the testing process by selecting the “Start procedure” button at the bottom of the “Test procedures” panel. All enabled tests in the suite will be performed in sequence from top to bottom in an automated fashion. The “Process log” panel on the bottom left of the main window will report on progress of the testing.

Once the test process is complete the user is prompted with a success message.

Performing a Single Test

Sometimes users may want to run a single test instead of a full test suite. To do this the user must select the test they want to perform and then select “Perform selected test”.

Test results

Tests that have passed will be green and the right column within the “Tests procedures” panel will be labeled “pass”.

Tests that have failed will be red and will be labeled “fail”.

If the test result is yellow and the status is “retest” this means that there was a glitch in the signal file or response file and the test should be selected and run again. This can be done by selecting the test and then selecting the “Perform selected test” button on the bottom right of the main window.

If the test result is yellow and the status is “error: file” this means that the file that ADCTest expects to find in the project/designated folder is not present.

If the test result is yellow and the status is “error: sig” this means that ADCTest did not see an input audio signal on the ADC channel specified for the test. This may indicate that there is an issue with DAC/ADC configuration, cables unplugged or misplugged, or audio driver issues.

ADCTest

Project Settings Help

FA DGI Federal Agencies Digital Guidelines Initiative

Project info
Title: ADCTest Demo
Date: 14/9/2018 at 17:38
Operator: Chris Lacinak
Audio DAC: RME ADI-2 Pro SN: 59705319
Audio ADC: Benchmark ADC1 USB SN: S0808037-0
Remarks:

Process log
Test run: saving to file finished
Test run: response analysis
response analysis 16 of 1
Test run: error: sig
error: sig 16 of 1
Test run: Test procedures finished

Test procedures

thdn_6597_m1_right_ch	enabled	pass
thdn_997_m10_right_ch	enabled	pass
thdn_997_m20_right_ch	enabled	pass
thdn_997_m40_right_ch	enabled	pass
snr_left_ch	enabled	pass
snr_right_ch	enabled	error: sig
tfmd_left_ch	enabled	pass
tfmd_right_ch	enabled	pass
tfmd_left_ch	enabled	pass
tfmd_right_ch	enabled	error: file
spis_left_ch	enabled	pass
spis_right_ch	enabled	retest
user_action	enabled	paused
xtalk_20_left_ch	enabled	fail
xtalk_997_left_ch	enabled	pass
xtalk_20k_left_ch	enabled	pass
user_action	enabled	paused
xtalk_20_right_ch	enabled	fail
xtalk_997_right_ch	enabled	pass
xtalk_20k_right_ch	enabled	fail

Start procedure

Test description
right channel dynamic range

Test mode
Normal snr_right_ch_resp.wav

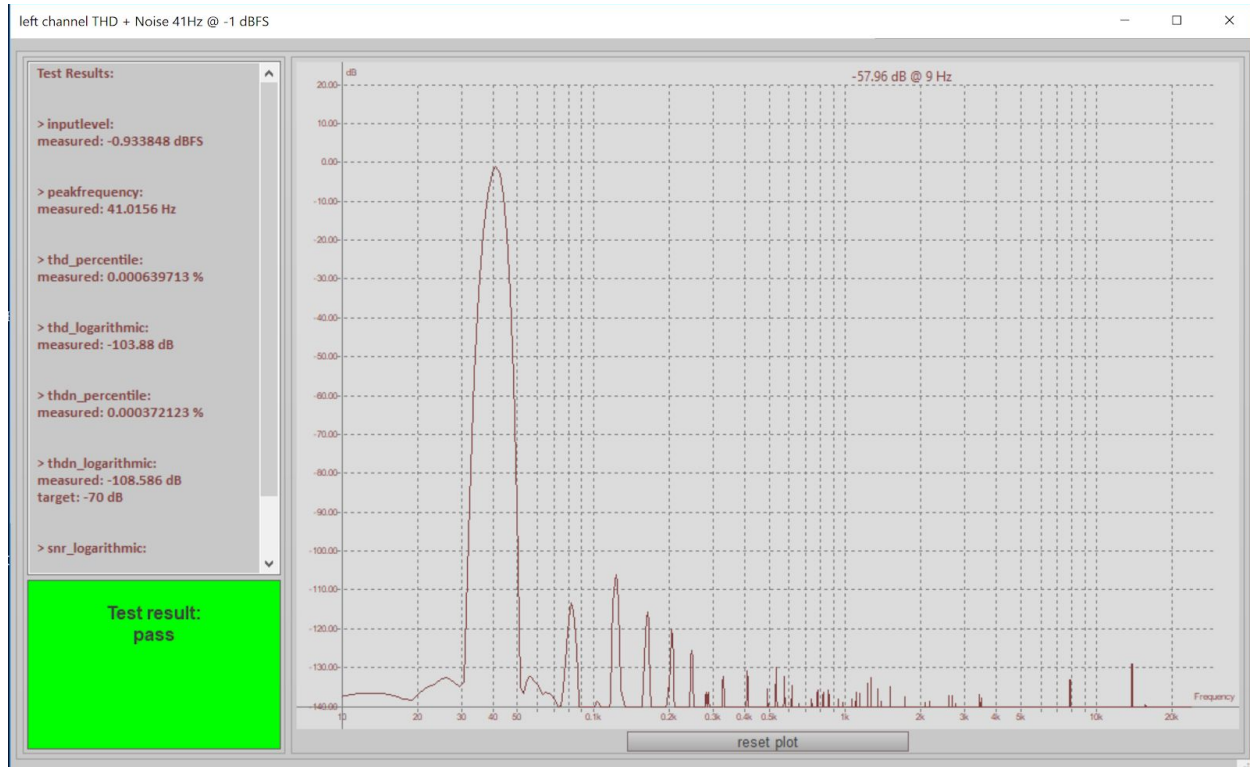
Audio routing
DAC chan: 1 ASIO MADiface USB
ADC chan: 1 Microphone (Benchmark 2.0)

Test parameters

signal module	singlesine
analysis module	thdn
tone frequency	997
tone level	-40
signal detection threshold	-70
fft size	32768
number of fft averages	16
execution time	successful

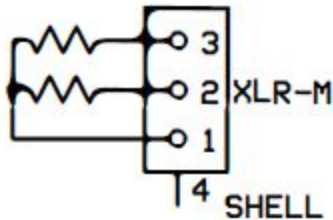
Perform selected test

Double clicking on any test will open a window that contains the relevant performance metrics, results and a pass-fail outcome in one pane and a graph of the results in another pane. The user may place the cursor over the graph to see frequency and level information for a given location on the graph. The user may also select an area of the graph to zoom in on. Selecting “reset plot” will fit the results in the pane.



Cross-Talk Test

The cross-talk test requires special consideration and an accessory. The test method for cross-talk testing requires removing the cable from the channel under test and terminating the input of the ADC with a shorting plug. Below is an example schematic for an XLR shorting plug. The resistor values should be matched to the output impedance of the DAC/signal generator and matched within 5% of each other. The necessary pause in the test suite to perform this activity requires a pause function in the application. If you scroll down the list of tests in the default.xml you will note that there is a test labeled “user action”. This pauses the test suite and prompts the user to hit ok when the shorting plug is in place.



Schematic drawn by Phillip Sztenderowicz.

Opening an Existing Project

Existing projects can be opened by selecting “Open Project” from the “Project” menu in the main window of ADCTest and navigating to and selecting the appropriate *[project_name].avt* file. ADCTest will load the settings and configuration of the .avt project file. If there are test result .xml files in the same directory ADCTest will present those results. For simply viewing past results, ADCTest does not require there to be any audio files in the directory.

To re-analyze existing response files the test mode needs to be in offline mode for all tests and the response files need to be located in the project directory and named according to the file names identified in the .avt file.

Inside an .avt File

An .avt file contains XML that specifies the configuration parameters and values for a project. All of the parameters and values are displayed in the ADCTest user interface, but only some are able to be modified in the user interface.

In order to keep the user interface simple, yet offer flexibility and expandability, we decided to utilize an XML schema to identify and document all of the parameters and associated settings for test signal generation and analysis. The default configuration conforms to the FADGI low cost guideline, but users may also setup alternate configurations as they wish by altering the XML file and saving it.

The main node, named “FADGIProject” contains basic parameters such as the project title and the date and time that the project was created.

The child node “procedures” contains the description and operational parameters of each test.

The following properties are defined for each “test” node:

1) *id*

The test identification number; This must be incremental.

2) *name*

The test short-hand name.

3) *alias*

A human-readable brief description of the test procedure.

4) *enabled*

Specifies whether the specific test procedure is enabled. If set to false, the test will not be run. This parameter can be changed using the application user interface or within the XML.

The following properties are defined for each “audiorouting” node:

1) signal_ch_idx

Specifies the output channel for the DAC to be used for the test. This parameter can be changed using the application user interface or within the XML.

2) response_ch_idx

Specifies the input channel for the ADC to be used for the test. This parameter can be changed using the application user interface or within the XML.

The following properties are defined for each "fileio" node:

1) signalfile

Specifies the file name that will be used for the .wav signal file generated for the test.

2) responsefile

Specifies the file name that will be used for the .wav response file generated for the test. This parameter can be changed using the application user interface or within the XML.

3) resultsfile

Specifies the file name that will be used for the .xml results file generated for the test.

4) workfolder

Specifies the file path where the files associated with the test will be located.

5) testtype

Specifies whether the "Normal" or "Offline" test mode will be used for the test. This parameter can be changed using the application user interface or within the XML.

The "config" child node enumerates individual operational parameters for each test. A number of parameters are common to all tests, in particular:

The "parameters" child node enumerates individual operational parameters for each test. A number of parameters are common to all tests, in particular:

1. signal

Specifies the type of stimulus signal used by the test procedure. Currently the signal types are: "octsine", or octave stepped sine for the frequency response tests; "dualsine", or dual sine for the intermodulation distortion tests, and "singlesine", or single sine for all other tests.

2. analyser

Specifies which analysis module will be used for the test. Currently the analyser types are: "stepfreq" for the frequency response tests; "thdn" for the THD + noise and dynamic range tests; "lfimd" for both of the frequency intermodulation distortion tests; "spis" for spurious inharmonic tests, and "xtalk" for cross talk tests.

3. *detectionlevel*
Specifies the level threshold at which a signal is determined to be valid to begin analysis.
4. *inttime*
Specifies the minimum length in mS of the stimulus signal.
5. *transtime*
Specifies a “guarding period” in mS that will be added at the beginning and end of the stimulus signal. This period will be discarded by the analysis module in order to avoid the effect of sudden transients.
6. *bursttime*
If the test procedure requires multiple burst signals, this parameter specifies the pause period in mS between bursts.
7. *outputfreqresponse*
Whenever the specific test generates a frequency plot during the performance metrics evaluation, if this parameter is enabled, the plot will be saved in the results file.
8. *workfolder*
Defines the location where the files associated with a test should be saved. As a default this parameter inherits the value for “datafolder” parameter in the main node.
9. *signalfile*
Specifies the name of the audio file containing the stimulus signal.
10. *responsefile*
Specifies the name of the audio file containing the audio stream recorded from the ADC under test.
11. *resultsfile*
Specifies the name of the XML file containing calculated performance and pass/fail result.
12. *maxsigdev*
Specifies the the maximum allowable signal deviation threshold for triggering a “retest” result.
13. One final parameter that is hard coded into the application for all tests is the window type that it is used. The only window specified within the FADGI guideline currently is Kaiser 7. Therefore Kaiser 7 is hard coded into all tests. The application has the ability to turn this into a parameter, allowing users to select alternate windows, but Kaiser 7 utilizes a good trade-off between main and side lobes, making it a good candidate for general use in test and measurement. It is also worth noting that level deviations caused by windowing are compensated for, so the level of a harmonic component is accurate, regardless of the type of window used.

The following parameters are specific to only certain tests as identified:

14. *freqstart* [frequency response]
The frequency at which the frequency sweep will begin.
15. *freqstop* [frequency response]
The frequency at which the frequency sweep will end.
16. *octsteps* [frequency response]
The number of steps per octave used in the frequency sweep.
17. *level* [frequency response]
The level at which the tones in the frequency sweep should be at the output of the ADC.
18. *tonefreq* [THD+N, spurious inharmonic, cross-talk]
The frequency of the stimulus signal.
19. *tonelevel* [THD+N, intermodulation distortion, spurious inharmonic, cross-talk]
The level at which the stimulus signal should be at the output of the ADC.
20. *fftlength* [THD+N, intermodulation distortion, spurious inharmonic, cross-talk]
The size, or number of points used, for FFT based analysis.
21. *fftavg* [THD+N, intermodulation distortion, spurious inharmonic, cross-talk]
The number of averages to use for the FFT in performing analysis.
22. *fftavgtype* [THD+N, intermodulation distortion, spurious inharmonic, cross-talk]
Specifies the type of averaging used when performing analysis. For instance, linear or exponential.
23. *notchbw* [THD+N, intermodulation distortion, spurious inharmonic]
Specifies the bandwidth of the notch used for the removal of a stimulus frequency in the analysis of a response file.
24. *harmsearchbw* [THD+N]
For each nth order harmonic, a search is made in the response spectrum for the highest value in the range: $(n \cdot F_c - \text{harmsearchbw} : n \cdot F_c + \text{harmsearchbw})$. In practice, due to the limited FFT resolution, the harmonic might fall in an adjacent frequency bin, using a search range guarantees that the harmonic is found.
25. *lowerlimit* [THD+N, intermodulation distortion, spurious inharmonic]
The lowest frequency at which a signal will be analyzed.
26. *tonefreq1* [intermodulation distortion]
The frequency of the first tone in a dual tone stimulus signal.
27. *tonefreq2* [intermodulation distortion]
The frequency of a second tone in a dual tone stimulus signal.

28. *levelratio* [intermodulation distortion]

The level ratio between the first and second tone in a dual tone stimulus signal.

29. *imdtype* [intermodulation distortion]

Specifies the intermodulation distortion analysis type. For instance, SMPTE or CCIT.

30. *higherlimit* [spurious inharmonic]

The highest frequency at which a signal will be analyzed.

The “performancespecs” node lists which performance metric will be used by the application to return a pass or fail result at the end of the specific test. The parameters are as follows:

1. *name*

The name given to the performance metric.

2. *type*

Variable type (either double or string), currently this specifier is not used by the application.

3. *value*

The pass-fail value for the performance metric.

4. *units*

The unit of measure being used for the performance metric. For instance dB or dBFS.

5. *criterion*

The relationship of the result compared to the pass-fail value. For instance, lessthan.

The following properties are defined for each “projectinfo” node:

1) *opsname*

Contains the information that is entered into the “Operator” field when creating a new project.

2) *dacname*

Contains the information that is entered into the “Audio DAC” field when creating a new project.

3) *adcname*

Contains the information that is entered into the “Audio ADC” field when creating a new project.

The “remarks” node contains the information that is entered into the “Remarks” panel in the main window of ADCTest.

Configuration File

ADCTest saves a configuration file in the “Public Documents” directory on the local machine, which stores configuration settings that are saved as part of the calibration procedure. The file is named “ADCTest.cfg”. The parameters stored within the configuration file are as follows:

1. NewPrefsInitialized
Version control managed internally by the application.
2. Version
Version control managed internally by the application.
3. DataDumpDir
No longer in use.
4. InputAudioSRate
Identifies the sample rate being used for the input side of the signal path.
5. InputFrameSize
Identifies the frame size being used for audio processing on the input side of the signal path.
6. OutputAudioSRate
Identifies the sample rate being used for the output side of the signal path.
7. OutputFrameSize
Identifies the frame size being used for audio processing on the output side of the signal path.
8. InputAudioHostName
Identifies the audio drivers being utilized on the input side of the signal path.
9. OutputAudioHostName
Identifies the audio drivers being utilized on the output side of the signal path.
10. InputDevName
Identifies the ADC being used.
11. InputDevChans
Identifies the number of channels on the ADC.
12. OutputDevName

Identifies the DAC being used.

13. OutputDevChans

Identifies the number of channels on the DAC.

14. OutputStreamGain

Identifies the level set within ADCTest as a reference point for all signal levels.

15. RTALength

Length, or resolution, of the FFT plot in the calibration panel, managed from the audio devices setup panel.

16. RTAWindow

Window type for the real time FFT in the calibration panel. This is no longer in use. The window is currently fixed to kaiser7.

17. RTAExAvg

Specifies the settings for the real time FFT averaging in the calibration panel, managed from the audio devices setup panel.

18. EnumerateDevicesToFile

No longer in use.

19. EnumerateDevicesToFilePath

No longer in use.

20. LogToFile

No longer in use.

21. LogToFilePath

No longer in use.

Test Results File

When a test is performed there are three files generated per test. These are the signal audio file, the response audio file, and the result XML file. The result XML file has the following structure.

FADGIResults is the parent node and has the title of the result file as well as the channel index of the audio channel being reported on. FADGIResults contains dataset, performancespecs, and testoutcome.

dataset has an ID and contains testmetrics and freqresponse. testmetrics contains a variable list of parameters. These parameters contain reporting on results for different relevant metrics for each test, providing the name of the metric, the result for that metric, and the unit being reported. Within the application these are the metrics which are reported within the results user interface on the left hand side of the graph. freqresponse contains the frequency and associated resulting level for each frequency measured.

performancespecs contains the specification name, data type, value, unit of measure, and criterion serving as the reference for pass/fail reporting.

testoutcome contains either a pass or fail value, reporting the outcome of the results compared to the performance specification.

Copyright and Licensing

ADCTest is in the public domain and is licensed under a 3-Clause BSD license.

Appendix A: Background and Context

Background

In 2016 AVP created and submitted to the Library of Congress a performance testing guideline and a user guide for low cost testing of analog to digital converters (ADCs) used in the digitization of audio recordings for preservation. These documents and the work behind them were part of a larger effort and set of deliverables over the past few years focused on the performance testing of ADCs. The earlier work in this larger effort focused on a high level testing guideline and user guide, offering a stringent set of performance specifications and a series of sophisticated test methods requiring a high performing, highly calibrated and complex test system. The testing devices on the market which meet the requirements of the high level guideline are upwards of \$25,000. It is the case that these devices have dropped significantly in value since the original work was performed, and can be found on the market today for upwards of \$15,000, but even this more reasonable number is well beyond the means of many archives. The concern at the conclusion of the high level test guideline effort was that a majority of organizations would simply not be able to utilize it, failing to address a central aim of this effort, which was to provide a simple, low cost means of testing their ADCs.

With this realization, the Library of Congress was interested in exploring lower cost test options that may be used by organizations with less resources. It was clear from the start that the endeavor to lower the cost would entail a sacrifice in both sophistication and precision. Considering these trade offs the following considerations became central in the conversation:

- Many organizations are currently not testing their devices at all. This was an impetus for the project.
- Testing, even if elementary, is better than not testing at all. A major finding of our field tests, performed as part of the larger effort over the years, was that even the most basic testing revealed significant errors in the ADC or signal path that had not been caught previously due to lack of performing any testing at all.
- The more frequently that testing is performed, the better the chance of identifying issues.
- The complexity and learning curve involved with being able to properly operate and interpret the results of sophisticated test devices is a significant deterrent for many organizations. Thus, one of the original aims was to bring simplicity to this process.
- The historic lack of standards, best practices, or guidelines providing both the test methods and the performance metrics in one succinct document has created ambiguity on the need and protocols for testing.
- The significant cost of test devices has been a major deterrent for organizations. In a survey done as part of this work, an overwhelming majority of organizations reported that they would not spend more than \$1,500 for a test device. In the author's experience working with organizations over the past 17 years, many organizations are not even able to pay \$1,500 for their ADC much less a test device.

The high level test guideline and associated deliverables, including scripts for running the tests, solve most of the problems listed above with the one major exception of cost which impacts a large number of organizations. While the guideline may be used in specifications for outsourcing digitization without impact on internal capital expenditures, for organizations that have internal digitization operations and wish to use a test system to apply the guideline, it is likely that only a small percentage that have not performed testing in the past would be empowered to do so in the future with the publishing of the high level guideline and associated deliverables due to cost.

On the flip side, the low cost test guideline and associated deliverables solve most of the problems with a major caveat that the quantity of tests in the test suite and the quality of the results fall significantly short of the high level test guideline. Speaking generally, a low cost device will simply not be as sophisticated or as precise as a higher cost device. However, the low cost test enables a large percentage of organizations that have internal digitization operations and wish to use a test system to apply the guideline to do so, enabling many to begin routine testing for the first time. As mentioned above, the transition from not-testing-at-all to testing routinely, even if using less than ideal means is probably the biggest gain to be had. At minimum this can answer the question of whether or not the ADC/system is failing.

The observations that arose and the thinking that went into this led us to realize that there are really three primary questions that an organization aims to answer through use of a testing guideline:

1. Is my ADC/system failing?
2. How does my calibrated, well performing ADC performs relative to the guideline and other ADCs?
3. Is my ADC/system performing optimally relative to its own specifications?

A high level test device that meets the guideline can answer all of these questions. A low cost test device can likely answer question number one for any converter, but it can only answer questions number two and three for ADCs which have performance specifications that are significantly lower than the device itself. This leads to the belief that:

1. A low cost guideline is a worthwhile endeavor with meaningful and practical utility.
2. Perhaps those without sufficient budget for performing the high level test guideline might temporarily procure a test device or utilize a service provider at key points (e.g. upon purchasing an ADC, performing quarterly testing), while frequently using the low cost test guideline and device on an ongoing and routine basis.

Origins of ADCTest

In 2016, the test guideline and a user guide were drafted and submitted to the Library. Following this work, the Library contracted with AVP to develop a beta-version software application that would perform the test methods and report the results relative to the performance metrics in an automated or semi-automated way. AVP first attempted to work with the developers of existing test and measurement software applications, such as ARTA and RMAA, to work on a FADGI-specific application. These attempts were unsuccessful, leaving us with needing to start from scratch. The first thing we did was to draft a list of requirements. At a high level these requirements and features consisted of the following:

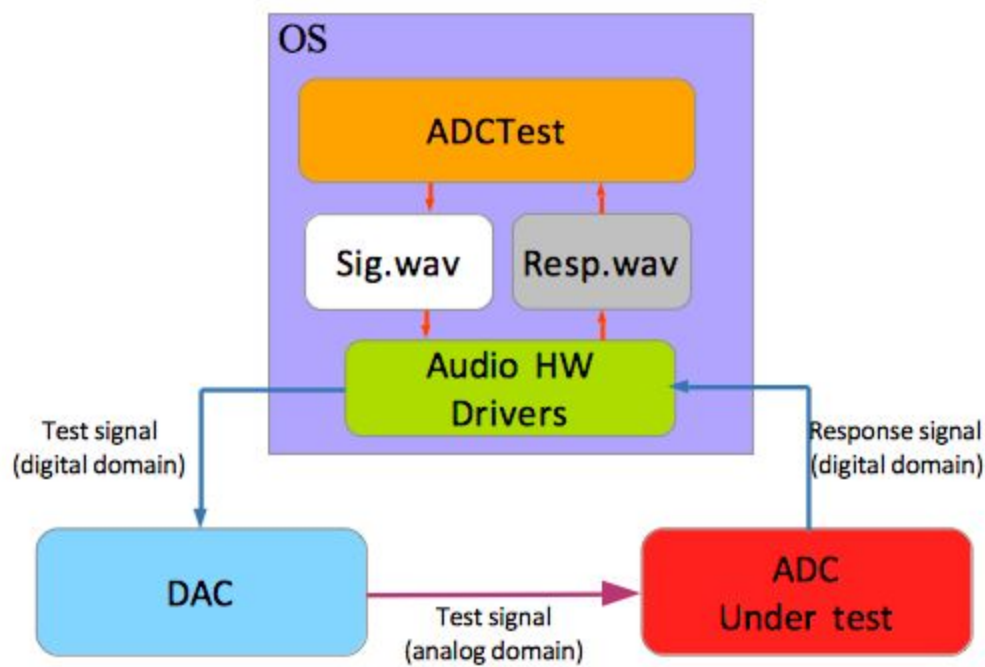
1. Perform all of the tests in the low cost guideline
2. Report results of tests according to the low cost guideline methods and metrics
3. Provide simple pass-fail reporting as well as more detailed results
4. Decouple the signal generation from the signal analysis to enable non-real time analysis and the use of alternative signal generators
5. Provide a simple calibration mechanism
6. Perform the tests using as much automation as possible
7. Provide ability to enable and disable individual tests
8. Support up to 24 bit, 96 kHz signals
9. Support USB, SPDIF, Firewire and AES interfaces

Following the drafting of these requirements AVP worked with application developer Christian Landone to develop an application referred to as ADCTest using C++ as the coding language.

Use-Cases and Application Architecture

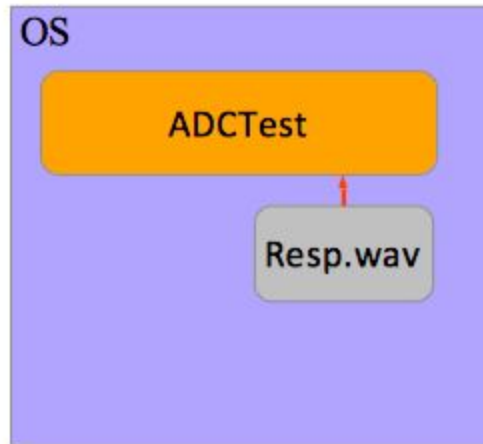
With the base requirements in mind we set out to design the application architecture and approach and came up with the name ADCTest. To begin, we established the basic test setup and protocol envisioned for use of ADCTest.

The first envisioned test setup and protocol considers a scenario in which an operator is using ADCTest for signal generation and analysis to test their own ADC. This test type is referred to as “normal” and uses the following setup and protocol:



1. The DAC is connected to one of the audio interfaces on the computer running ADCTest.
2. The ADC under test is connected to one of the audio interfaces on the computer running ADCTest.
3. ADCTest is started and upon opening ADCTest shows the list of tests in the guideline. The parameters for each test procedure are pre-defined, as specified in the guideline.
4. The operator selects both the DAC and ADC from a list of devices.
5. The operator performs calibration to establish levels to be used as a point of reference by ADCTest when performing testing.
6. The operator reviews the list of test procedures to be performed and has the option to enable or disable each test.
7. The operator initiates the test procedure with a simple button press.
8. The application automatically performs each test procedure in the list, pausing and prompting the operator for input at points where human intervention is required.
9. Once the end of the test procedures list is reached, results are saved to file both in machine and human-readable formats.
10. The operator is able to review simple pass-fail results as well as see details for each test result.

The second envisioned test setup and protocol considers a scenario where a client is analyzing and reporting on a set of response files received from a digitization vendor that has produced them using the guideline and is submitting them to the client for proof of ADC conformance. This test type is referred to as “offline” and uses the following setup and protocol:



1. The vendor captures response files that have been passed through the ADC being used for client work. The test signals used and the response files captured conform to the FADGI low cost guideline and ADCTest parameters.
2. The vendor delivers the response files to the client.
3. The client opens ADCTest and loads or points ADCTest to the response files. The computer that the client is running does not need to have any specialized audio hardware or software.
4. The client initiates the analysis procedure with a simple button press.
5. Once the end of the analysis procedure is reached, the client is able to review simple pass-fail results as well as see details for each test result in order to audit whether the vendor’s ADC conforms to the FADGI guideline performance metrics, and if interested, the extent to which it passed and/or failed.
6. Results are saved to file both in machine and human-readable formats.

Following the establishment of these scenarios we explored the application architecture and concluded the following:

1. The architecture should permit the introduction of additional classes that extend the type of waveforms that can be produced by the application’s internal signal generation engine, e.g. random noise (white/pink), chirps.

2. The architecture should permit the introduction of additional classes that extend the type of tests can be performed by the application's analysis engine, e.g. Impulse response of the system.
3. The architecture should use a fully modular approach, treating each individual test as its own module and allowing the source type, test signal characteristics, and measurement algorithm parameters to be configured for each newly created module.
4. The architecture should decouple the signal generation and signal analysis routines so that testing can be performed in real-time, non-real-time and use test signals from other generators.
 - a. ADCTest will generate the test signals and save them as WAVE files (signal files).
 - b. ADCTest will play the signal files through the DAC and ADC and save the resulting WAVE files (response files)
 - c. ADCTest will load the response files, analyze them, report the results and save the analysis results as XML files.