Application Note
AN-AVS-0017-v1.14
April 2018

Multi-Room Music Hardware Pre-Qualification Test Suite

ABSTRACT

Amazon Multi-Room Music (MRM) technology imposes very specific hardware and software requirements on participating devices. The MRM Pre-Qualification Test Suite enables device makers to check the compatibility of their devices with the MRM technology. Devices wishing to implement MRM must pass all tests in the MRM Pre-Qualification Test Suite before the device maker will be provided with the MRM SDK.

TABLE OF CONTENTS

1 OVERVIEW

This document details the Pre-Qualification Test Suite used to pre-qualify a hardware platform to run the Amazon Multi-Room Music SDK (MRM SDK). In order to provide a good Multi-Room Music (MRM) experience for Amazon customers, MRM requires that the device hardware and software have the following capabilities:

1. A high-resolution timer (HRT), that is readable from user-space in less than 1 µs, and is unaffected by power state changes
2. An ALSA audio driver implementation that has a sample-accurate (< 50 µs) `snd_pcm_delay()` call, or some equivalent
3. A host processor and network chipset capable of sustaining 2 Mbps throughput under normal network conditions
4. An audio codec/DAC that uses a clock that is synchronous to the host processor clock

Please see the MRM Preliminary Requirements app note (AN-AVS-0006) for a more complete list and explanation.

In addition, to implement the tests described in this document, a development version of the device must expose a 3.3V or 5.0V GPIO line on a header outside the device that can be toggled with a delay of < 1µs.

Once each test runs with passing results on the target device, the device maker should to be able to write the interface layer to get the full MRM library running on the device-under-test (DUT).

The tests are as follows:

1. Audio Pipeline Drift
2. High-Resolution Timer
3. GPIO Validation
4. Time Synchronization
5. Audio Placement
6. Audio Distribution

The device must pass all six tests before it will be considered ready for an MRM implementation.

The default code presumes that the target device is Linux-based using ALSA audio drivers. If the target device does not conform to those expectations, please confer with your Amazon technical contact for next steps.

2 REQUIREMENTS

Required Equipment

- Two Devices Under Test (DUT) with GPIO lines available on external leads
- Two Echo Dot Gen2
- C++ Development Toolchain for DUT
- USB audio digitizer (Focusrite 18i8 or equivalent with a sampling rate of 48KHz)
- Logic analyzer (Saleae Logic 8 or any model capable of recording 2 digital GPIO lines)
- Amazon-provided MRM Line Out Test Board (used to level shift the GPIO line to audio levels) or equivalent circuit to voltage divide the GPIO signal to be usable at audio line levels. Please refer to the Recommended Equipment section of this document for a sample circuit.
- Laptop or desktop Macintosh computer
- Wi-Fi router, including admin access to see IP address of connected devices

Required Techniques and Software

- Audacity audio processing software (to run on Macintosh computer)
- Rscript interpreter (to run on Macintosh computer)
- Rstudio Desktop, https://www.rstudio.com/products/rstudio/download/ (to run on Macintosh computer)
- Rscript packages
    - dplyr
    - ggplot2
    - readr
    - tuneR
    - lubridate
    - optparse
    - scales
    - padr
    - stringr
    - reshape2
- Sox, http://sox.sourceforge.net (to run on Macintosh computer)
- DUT command to play a WAV file
- C++ code to initialize and access the device HRT (if `clock_gettime(CLOCK_MONOTONIC_RAW)` is not sufficient)
- C++ code to toggle GPIO

3 BUILDING THE PRE-QUALIFICATION TEST SUITE

The Pre-Qualification Test Suite is written to take advantage of the same class structure that is used by the MRM SDK. As such, implementing the classes required for the Pre-Qualification Test Suite will allow rapid integration of the MRM SDK.

The Pre-Qualification Test Suite you receive will have already been built on the toolchain for your target device. However, the underlying classes needed to successfully implement the tests have been stubbed out or are implemented using a generic set of calls that work on a Raspberry Pi 3B. The first task is to implement the following class methods in `Delegates3P.cpp`:

- `DeviceInfoDelegate->getIPAddress()`
- `GPIODelegate->GPIODelegate()`
- `GPIODelegate->setGPIO()`
- `GPIODelegate->clearGPIO()`

If you determined during the MRM requirements questionnaire that `clock_gettime(CLOCK_MONOTONIC_RAW)` was sufficiently fast for your timer requirement, then you do not need to implement a custom version of `TimeDelegate->getLocalTimeNs()`. However, if your device needs to use custom timer code, please implement it within that class method.

After implementing the new delegates, update `Makefile` to use your delegate source code and to use your compiler settings.

Devices that do not use ALSA for their audio pipeline would need to implement their own class to replace the `ALSA` class defined in `ALSA.cpp` and change `preQualTest.cpp` to instantiate the new audio player class and use the equivalent methods. If your device falls into that category, please confer with your Amazon technical contact for next steps.

Once the delegate classes are implemented specific for the device, use `make` to compile the test. If you are cross-compiling, load the compiled test onto the DUT. Then run each test according to the directions in the associated test description in sections below.

4 TEST 1: AUDIO PIPELINE DRIFT

The Amazon MRM library requires the host processor clock and the audio output clock to be synchronous. Specifically, if the host processor is sending a 48000 Hz audio signal, it is important that the DAC actually generate a 48000 Hz signal with a high degree of accuracy. Passing this test does not guarantee that the device will be able to pass end to end synchronization, but it is a pre-requisite to that goal.

This test is accomplished without custom software that accompanies the MRM Pre-Qualification Test Suite. It is a manual test of playing audio and digitizing it to confirm the accuracy of the audio output clock.

4.1 Equipment required

- DUT
- Computer running Audacity
- USB audio digitizer connected to computer

4.2 Procedure

Play and capture the audio

Copy the WAV file `480Hz_2ch_30s.wav` to the DUT. A 480Hz sine wave is used to create an easy to measure multiple of the 48000 Hz sample rate since each wave period should correspond to exactly 100 samples.

Connect the DUT output to the USB digitizer. If the device has line-out, use that. Alternatively, you can use either a filter board that takes the internal speaker's Class D output and low-pass filters it into the audio range, or you can use a microphone that is placed in close proximity to the DUT speaker.

On the device, run a command to play the audio file. A typical example might be:

```
aplay -D hw:0,0  -c 2  -r 48000  -f S32_LE  480Hz_2ch_30s.wav
```

Record the DUT output using Audacity on the computer, with a sample rate of 48000 samples/sec.

Examine the captured audio

Find an upward-going zero crossing near the start of the recording, and put the cursor there.

Record the approximate time T1 (in seconds) at that zero crossing, and the exact 48000 Hz sample number (S1).

Approximately 0.5-1.0 seconds later in the recording, find another upward-going zero crossing.

Record the approximate time T2 (in seconds) at this zero crossing, and the exact 48000 Hz sample number (S2). S2-S1 should be an exact multiple of 100 samples (or very close to it), if the audio pipeline drift is zero.

Calculate the drift

wavFileFrequency = 480
sampleRate = 48000
samplesPerPeriod = sampleRate / wavFileFrequency = 100
error_samples = (S2-S1) – samplesPerPeriod * round((S2-S1)/100)
deltaT = T2-T1
error_PPM = 1.0E6 * (error_samples / sample_rate) / deltaT

4.3 Expected results

To pass the audio pipeline drift test, the error_PPM should be less than the tolerance of the host processor crystal plus the tolerance of the crystal in the USB digitizer.

The tolerance of the crystal in the USB digitizer is typically < 20 ppm. According MRM requirements, the tolerance of the host processor crystal should be between 2 and 30 ppm. Please refer to your host processor crystal datasheet to confirm the actual value.

If you get a number that is non-zero, check the numerator and denominator used to generate the audio pipeline PLL and adjust, if possible, to make the error = 0.0 ppm.

In many cases, failing this audio pipeline drift test is caused by some combination of having the audio output stage (DAC or DAC + DSP) being the I2S master instead of the host and/or an interposed DSP not adjusting the DAC clock for drift caused by a separate DSP crystal.

If for some reason you are unable to bring the drift into a passing value, it is possible to modify the implementation of the `TimeDelegate` to adjust for the drift. Please see the implementation notes in the `TimeDelegate` sample code for instructions on where to put the measured ppm value. A positive number for the ppm value means that the audio pipeline clock is slower than

the host processor clock. The `TimeDelegate` class will use this information to correct the values before handing off audio to the pipeline.

5 TEST 2: HIGH-RESOLUTION TIMER

This test confirms that the high-resolution timer (HRT) is functioning with sufficient accuracy. Typically, `clock_gettime(CLOCK_MONOTONIC_RAW)` will be sufficient for this purpose. If it is not sufficient, the HRT can be implemented either using one of the performance counters (e.g. on ARM, using the CCR = Cycle Count Register), or using a ~1 MHz counter. In both cases, the counter must be readable from user space, and readable in a very short amount of time (typically <1 µs). The counter must not be affected by power state changes.

It is acceptable if the counter counts faster than 1MHz (e.g. 1.5Mhz). This test ensures that the HRT chosen can be read from user space with a minimum latency, and that the values returned are increasing and approximately correct.

5.1 Equipment required

DUT

- DUT with preQualTest compiled

5.2 Procedure

On the DUT, run: `preQualTest –gtest_filter="HRT.*"`

5.3 Expected Results

The test output will show `PASSED` if the test has passed.

5.4 Sample output

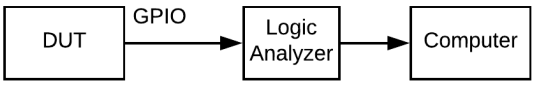
```
[          ] PreQualification Tests for 'XXXXXX', run: 2017-11-15T21:47:57Z
[          ]
Note: Google Test filter = HRT.*
[==========] Running 3 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 3 tests from HRT
[ RUN      ] HRT.GranularityTest
[          ]    Ensures that the HRT resolution is less than 1us
[       OK ] HRT.GranularityTest (0 ms)
[ RUN      ] HRT.SlowAccessTest
[          ]    Ensures that HRT increments at about the right rate (~1E9 ns/sec)
[          ]
[          ]            HRT1_ns,HRT2_ns,delta_ns,result
[          ]            1938561812666,1939561868666,1000056000,PASS
…
[          ]            1942562474666,1943562528666,1000054000,PASS
[       OK ] HRT.SlowAccessTest (5001 ms)
[ RUN      ] HRT.FastAccessTest
```

```
[          ]    Ensures that the HRT can be accessed quickly (<2us, one failure allowed)
[          ]
[          ]            HRT1_ns,HRT2_ns,delta_ns,result
[          ]            1943562911333,1943562912666,1333,PASS
…
[          ]            1943563724666,1943563725333,667,PASS
[       OK ] HRT.FastAccessTest (1 ms)
[----------] 3 tests from HRT (5003 ms total)
[----------] Global test environment tear-down
[==========] 3 tests from 1 test case ran. (5003 ms total)
[  PASSED  ] 3 tests.
```

6 TEST 3: GPIO VALIDATION

One GPIO line is brought out of the DUT to get high-accuracy timing reference pulses to measure the HRT and time synchronization. This test confirms that the GPIO functions at the required speed.

6.1 Equipment required



- DUT with GPIO line exposed and preQualTest compiled
- Logic analyzer connected to computer

6.2 Procedure

Connect the GPIO line and ground to the logic analyzer. Setup the logic analyzer to capture with a resolution of at least 1µs per sample.

On the DUT, run `preQualTest -gtest_filter="GPIO.*"`

Start logic analyzer capture, with 1 µs resolution, for 30 seconds. Immediately start the executable.

Wait 30 seconds for the capture to complete.

6.3 Expected Results

Look at the results manually.

Ensure that the slow pulses are approximately 1 second high, and 1 second low, each.

At the end of the long pulses, there will many short pulses. Because the sampling rate is much slower than the GPIO toggle rate, you will probably not see all the toggles. Verify that each of the short pulses are <1 µs high and <1 µs low.

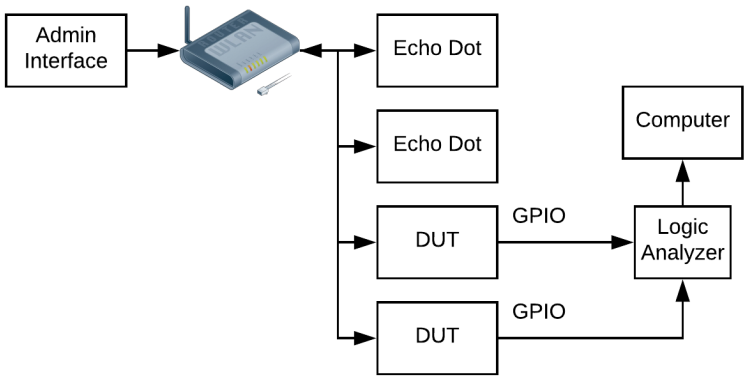
Verify that the total duration of the fast toggle section is <20 µs if there are 20 toggles and <100 µs if the version you are using sends 100 toggles.

7 TEST 4: TIME SYNCHRONIZATION

This test uses an instance of the Amazon MRM Time Synchronizer class to check for correct time sync. Packets are exchanged across the Wi-Fi interface and a GPIO line is toggled based on the "common time" established by the Time Synchronizer. The synchronization between a test device and the DUT is checked by recording the digital GPIO outputs with the logic analyzer. The results are processed by a script.

NOTE: The library to run the Time Synchronizer code is provided only in binary form, rather than source code.

7.1 Equipment required



- Two DUTs with GPIO line exposed and preQualTest compiled
- Logic analyzer connected to computer
- Two Echo Dots
- RStudio software running on computer

7.2 Procedure

First, you need to find the IP address of a time master. To do this, do the following:

- On your Alexa companion app, log in as the user configured on the two Echo Dots
- Under Settings, tap on the entry for the first Echo Dot, scroll down until you see the "Serial number," write down that serial number
- Do the same thing for the second Echo Dot and write down the serial number
- Determine which Echo Dot has the "highest" serial number and mark it physically
- Under Settings->Multi-Room Music, create group "Everywhere" and add the two Echo Dots
- Open up the admin interface of the Wi-Fi router. Turn off all devices except the one marked as "highest."
- Document the IP address of the device that is still connected. This is the `Time_Test_Target`
- Power back on the other Echo Dot

Next, attach both DUTs to the same Wi-Fi network as the two Echo Dots.

Connect pins 0 and 1 of the logic analyzer to the GPIO pins of the first DUT and the second DUT respectively.

Run the test on both of the DUTs: `preQualTest -t 1800 --gtest_filter="TimeSync.*" <Time_Test_Target>` This tells the test to run for 30 minutes on each device. Be sure to specify the `Time_Test_Target` address on both.

After waiting at least 10 minutes for the clocks to fully converge both skew and drift, check the console output of DUTs to see that the drift values demonstrate convergence (i.e. they report stable drift values). It is possible that convergence could take up to 20 minutes of wait time on congested networks. Once they have converged, then start measuring using the logic analyzer.

Capture data for 5 minutes on channels 0 and 1 using the logic analyzer app with a resolution of 1M samples/second.

Export the data from the logic analyzer to a file (one line per repeated condition) into a file named `test_<date>.csv`. In the Data Export settings of the Logic application, select "All time" for the Samples to Export and "Output one row per change" for the CSV Settings. Make sure that there is a header line in the file of the format `sample,device1,device2`.

Run the R script and provide the name of the CSV file along with a label for the DUT to analyze the data:
`Rscript --vanilla mrm-timesync-analysis.R test_<date>.csv <DUT>`

7.3 Expected results

The drift report in the DUT console should converge to a stable value.

Examine the plot: `hist_<DUT>_skew.png`. There should be a clear peak near 0 µs, and the bulk of the histogram (TP95 spread) should be within 150 µs of the median (TP50) line.

NOTE: In the final evaluation of DUT compatibility with MRM, this error is added to the error in the audio placement test (Test 5) below, so it is important that this error be minimized.

Examine the plot: `VsTime_<DUT>_skew.png`. There should be a smooth curve that settles around 0.0 µs skew after a few minutes. Any outlier points should be investigated.

NOTE: It is normal to see some outliers, when the time sync algorithm is still converging, as new estimates of the skew/drift are made.

NOTE: If the device passed the previous GPIO and HRT tests but this test fails to show the TP95 spread being within 150 µs of the median (TP50) line, let the time sync run longer and then take the measurement again. In congested environments it is possible that the devices could take upwards of 20 minutes for both the clock skew and drift to fully converge.

7.4 Sample Output – Console

```
===== Time Sync Test =====
Time Master: 192.168.0.103
Starting output on GPIO line...
Starting TimeSyncClient...

***** timeSynchronized: FALSE
INFO:TimeSyncClient:clientThread:reason=starting,server=TM:
INFO:TimeSyncClient:clientThread:reason=clientStarted,remote=TM: :55444
INFO:TimeSyncClient:clientThread:reason=skewOrDriftChange,master=TM:
,time=2373.544408,skewX=2371.300479,skewY=163412.532201,drift=24.436914,reportedDrift=0.000000,numExchanges=100
***** onNewSkewAndDriftAt: time=165786077738000, skew=163412532255449.687500, drift=0.000000
<HDMI_EP>get cec msg cec_event=37
INFO:TimeSyncClient:clientThread:reason=skewOrDriftChange,master=TM:
,time=2378.575389,skewX=2376.568327,skewY=163412.532201,drift=20.097458,reportedDrift=0.000000,numExchanges=200
```

Check for convergence of drift values:

```
INFO:TimeSyncClient:clientThread:reason=skewOrDriftChange,master=TM:
,time=2868.986923,skewX=2669.451574,skewY=163412.535836,drift=12.682828,reportedDrift=0.000000,numExchanges=9900
***** onNewSkewAndDriftAt: time=166281531385000, skew=163412538366197.937500, drift=0.000000
<HDMI_EP>get cec msg cec_event=37
***** onNewSkewAndDriftAt: time=166289238814076, skew=163412538463481.250000, drift=0.000000
INFO:TimeSyncClient:clientThread:stats:sent=10046,recvd=10000
INFO:TimeSyncClient:clientThread:reason=skewOrDriftChange,master=TM:
,time=2881.968767,skewX=2676.946450,skewY=163412.535940,drift=12.726485,reportedDrift=0.000000,numExchanges=1010
0
***** onNewSkewAndDriftAt: time=166294520529692, skew=163412538549457.000000, drift=0.000000
***** onNewSkewAndDriftAt: time=166298742831000, skew=163412538603251.218750, drift=0.000000
***** onNewSkewAndDriftAt: time=166304605634153, skew=163412538677627.343750, drift=0.000000
<HDMI_EP>get cec msg cec_event=37
***** onNewSkewAndDriftAt: time=166309597631769, skew=163412538741430.562500, drift=0.000000
INFO:TimeSyncClient:clientThread:reason=skewOrDriftChange,master=TM:
,time=2901.515933,skewX=2889.463743,skewY=163412.538641,drift=12.710580,reportedDrift=0.000000,numExchanges=1050
0
***** onNewSkewAndDriftAt: time=166314092194000, skew=163412538794651.843750, drift=0.000000
INFO:TimeSyncClient:clientThread:reason=skewOrDriftChange,master=TM:
,time=2908.674211,skewX=2901.223617,skewY=163412.538797,drift=12.751425,reportedDrift=0.000000,numExchanges=1060
0
***
```

7.5 Sample Output – RStudio Analysis

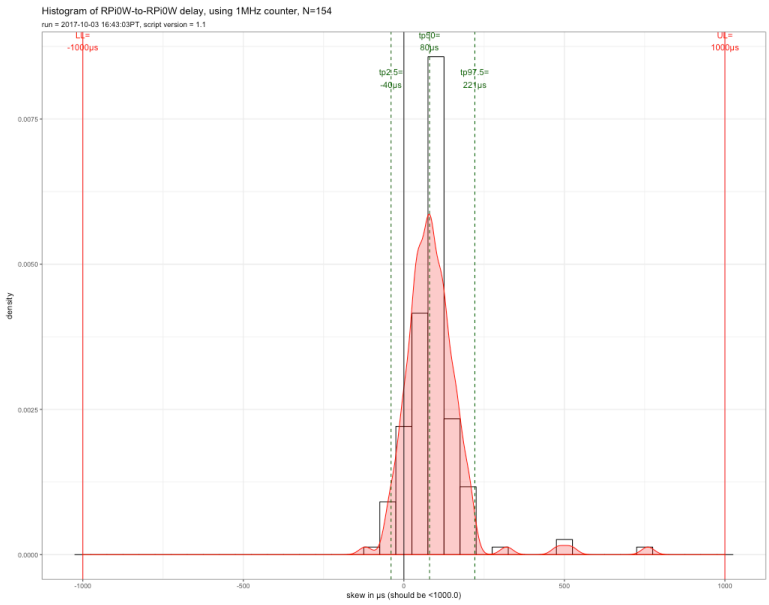


Figure 1. Sample plot with two RPi0s – device-to-device Time Sync delay

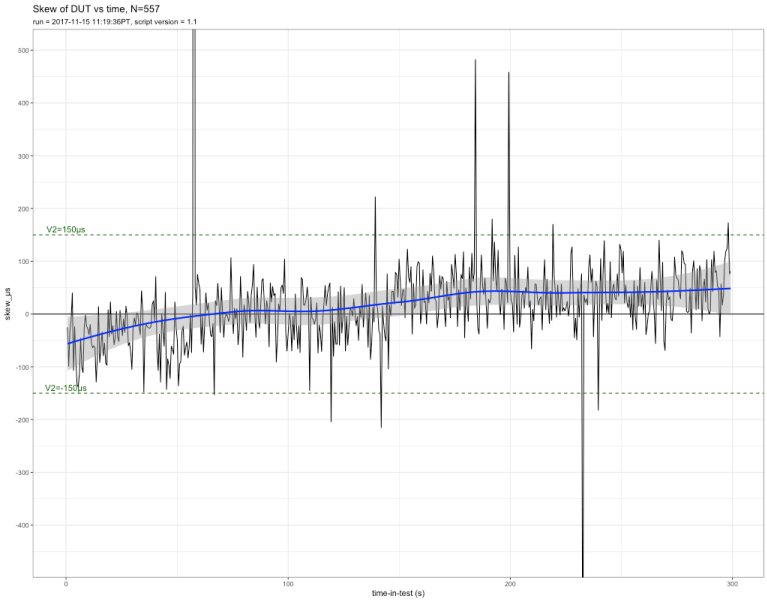


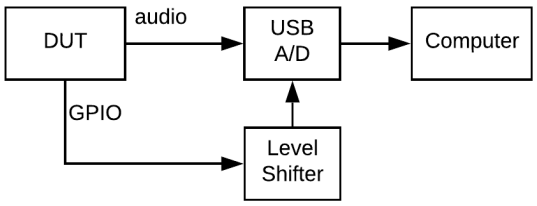
Figure 2. Sample plot with two RPi0s – device-to-device Time Sync skew vs time

8 TEST 5: AUDIO PLACEMENT

Audio placement is the process of "placing" audio in time, such that the first sample of the audio appears at exactly the specified time. This test uses a GPIO to indicate a specific point in time (as measured by the HRT) and plays audio at exactly 1 second in the future. Audio is played repeatedly, so that many samples of audio placement can be measured in a single test run.

The core effect of this test is to confirm that the ALSA `snd_pcm_delay()` call returns a correct and consistent answer.

8.1 Equipment required



- DUT with GPIO line exposed and preQualTest compiled
- USB audio digitizer
- Sox software running on computer
- Rscript running on computer
- Amazon-provided MRM test board or equivalent (used to level shift the GPIO line to audio levels)

8.2 Procedure

Connect DUT GPIO to the USB audio digitizer on channel 0, via the MRM test board

Connect DUT audio line out to the USB audio digitizer on channel 1

On the DUT, run `preQualTest -n "hw:0,0" --gtest_filter="AudioPipeline.PlacementTest"`

Verify by Using Audacity, verify that the test is running correctly by confirming that `preQualTest --gtest_filter="AudioPipeline.PlacementTest"` generates a pulse on audio channel 0, followed by a sinewave approximately 1 second after each pulse on channel 1. Set levels to about ¾ full scale, for both GPIO and audio channels. Stop the `preQualTest --gtest_filter="AudioPipeline.PlacementTest"`.

Modify the `testDir` variable in `measurePlacement.sh` to point at the target directory for the reports and to point at the Sox executable directory.

Run `measurePlacement.sh` on the Macintosh computer.

After `measurePlacement.sh` has been running on the computer for 5 seconds, on the DUT run `preQualTest -n "hw:0,0" -p 100 --gtest_filter="AudioPipeline.PlacementTest"`. This command will run the audio placement test over 100 iterations.

When `measurePlacement.sh` is done capturing audio, it will automatically run the sox and Rscript to process results. Please note that the capture time set in the `measurePlacement.sh` script it set to expect around 100 iterations.

If the device also includes a mixer in the audio pipeline, repeat the test using the ALSA mixer target in the -n parameter. For example: `preQualTest -n "plug:ossmix" -p 100 --gtest_filter="AudioPipeline.PlacementTest"`.

The DUT must pass placement on both direct hardware and through the mixer in order to be considered passing.

8.3 Expected results

Examine the histogram to ensure that a single peak appears, with a TP95 width of no more than ±50 µs. Although the peak should ideally be centered at zero, the MRM Engine will add/subtract a constant value from the Play At time to correct this offset (NOTE: the offset must be constant over time for this method to work).

NOTE: the performance of the system as it relates to MRM is this error is added to the error from time synchronization test above (Test 4.) The sum of the TP90 errors in Test 4 and the error in Test 5 should not exceed 5000 µs for room to room-level MRM synchronization or 150 µs for stereo pair MRM synchronization.

8.4 Sample output

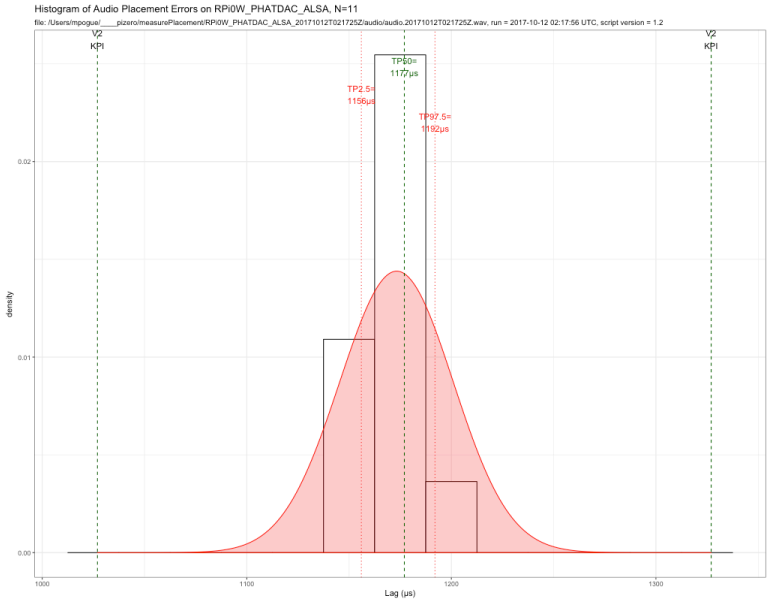


Figure 3. Sample plot - histogram of audio placement inaccuracy

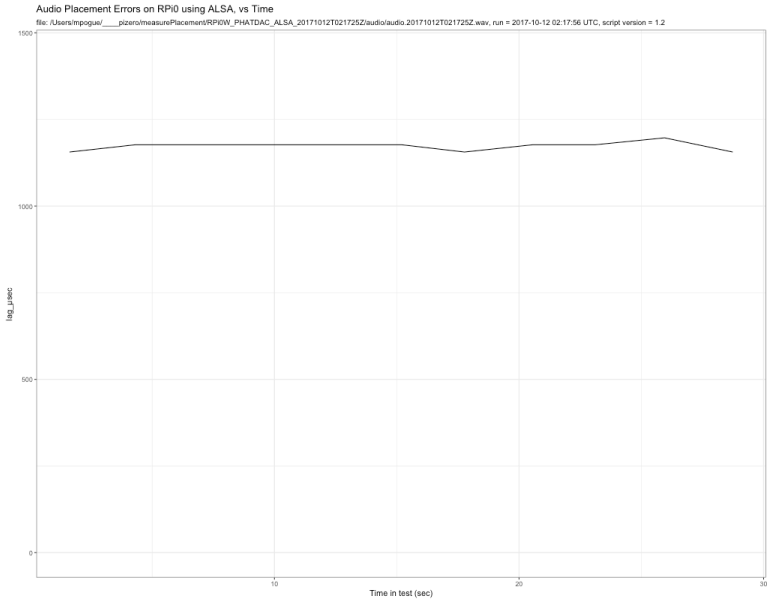


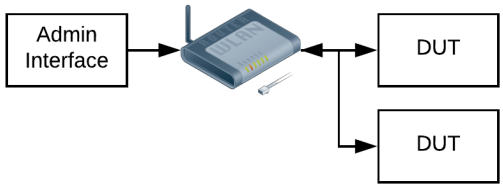
Figure 4. Sample plot - audio placement inaccuracy vs Time

9 TEST 6: AUDIO DISTRIBUTION

In order to play audio output to N devices without any audible interruptions or audio dropouts, each Audio Distribution Master must be capable of sustaining approximately 520Kbps to each Audio Distribution Slave. This test simulates audio output from one DUT to another under “normal office” conditions, to ensure that the host processor + Wi-Fi chipset are likely to be able to sustain this data rate.

NOTE: certification testing will do a more rigorous test, involving more devices, and under controlled conditions of congestion and simulated distance.

9.1 Equipment required



- 2 DUTs with preQualTest compiled

9.2 Procedure

Assign one DUT to be the receiver and determine its IP address. `<MRM_receiver_IP>`

Run `preQualTest --gtest_filter=AudioDistribution.Slave` on that receiver DUT.

Run `preQualTest --gtest_filter=AudioDistribution.Master <MRM_receiver_IP>` on the other DUT.

9.3 Expected results

The average bandwidth in a “normal” office environment should be >2 Mbps. This bandwidth result increases the likelihood that under more challenging conditions and with more devices in the group, that the network stack on the DUTs will be sufficient to support at least 4 devices in a MRM group.

9.4 Sample output

Master

```
UNICAST MASTER, version 2.14: compiled Oct 17 2017, 19:23:07
QoS disabled
KILL AFTER: 0 MBytes (0 = no kill)
BLOCKSIZE: 10 Kbytes
Trying to connect to Receiver at '192.168.27.11'...CONNECTED.
master: now connected to 192.168.27.11 on port 1234....
```

Slave

```
UNICAST SLAVE, version 2.14: Oct 17 2017, 20:21:39
slave: waiting for connections...
slave: got connection from 192.168.30.45
Each report below = ~1000000 bytes received.

timestamp,incrMbps,cumuMbps
2017-10-17T20:27:03.121743Z,1.281,1.281
2017-10-17T20:27:06.129685Z,2.679,1.735
2017-10-17T20:27:08.769002Z,3.013,2.019
2017-10-17T20:27:11.242973Z,3.252,2.231

…
```

10 RECOMMENDED EQUIPMENT

Below are references to equipment needed for the testing.

10.1 Saleae Logic Analyzer

https://www.amazon.com/Logic-Black-Saleae-8-Channel-Analyzer/dp/B0749G85W2

10.2 Focusrite 18i8 USB Audio Digitizer

https://www.amazon.com/Focusrite-Scarlett-Audio-Interface-Tools/dp/B01E6T547Y

10.3 Example Circuit for GPIO Level Shifter (3.3V to audio line level)