AN14693

How to Connect Asynchronized Audio Source and Sink with NXP CM7 Software ASRC

Rev. 1.0 — 5 June 2025 Application note

Document information

Information	Content
Keywords	AN14693, RT1060, ASRC,SRC, Audio, Voice, USB audio
Abstract	This document introduces NXP CM7 Asynchronized Sample Rate Converter (ASRC) and demonstrates how it works on NXP RT1060 platform.

1 Introduction

The *How to Connect Asynchronized Audio Source and Sink with RT1170 ASRC* (document AN14631) describes a solution for matching an asynchronized audio source and sink by the hardware Asynchronized Sample Rate Converter (ASRC) block of i.MX RT1170. To handle the same problem on the processors that do not have a hardware ASRC, we must use a software ASRC (when local Audio PLL real time adjusting is not capable enough).

This document introduces NXP CM7 software ASRC and demonstrates how it works on the NXP i.MX RT1060 platform.

For basic concepts and ideas of synchronized or asynchronized audio source and sink, see the *How to Connect Asynchronized Audio Source and Sink with RT1170 ASRC* (document AN14631).

2 NXP CM7 ASRC overview

The NXP CM7 Asynchronized Sample Rate Converter (NXP CM7 ASRC) is designed for high-quality sample rate conversions for audio and speech applications. Each one instance of the NXP CM7 ASRC supports 1, 2, 3, or 4 channels. It provides four levels of converting quality, and the option for using linear interpolation or cubic interpolation. The converting consumes a reasonable amount of MCPS (MIPS), and the THD+N of the converting is as low as -80 dBFs.

The NXP CM7 ASRC converts an input signal of Fs In to an output signal of Fs Out. The ratio of Fs Out/Fs In can be any value between 1.0-R ~ 1.0+R. And the ratio value can be adjusted while the NXP CM7 ASRC is running. There are two use cases.

2.1 ASRC for synchronization only

In this use case, Fs In is nominally the same as Fs Out. Since the audio source and sink are driven by different clocks, the true value of Fs Out/Fs In varies within a small range around 1.0. Therefore, we have -0.01 < R < 0.01 (or even -0.0001 < R < 0.0001) in practical. When -0.05 < R < 0.05, the frequency aliasing created by NXP ASRC is small and can be ignored. The NXP ASRC can work standalone and no post- or pre-low pass filter is needed (If the local audio PLL run-time adjusting is allowed, it could be another approach for nominally same Fs synchronization).

2.2 ASRC for converting Fs and synchronization

In this use case, Fs In is different from FsIn. For example: Fs In = 33 kHz, Fs Out = 48 kHz, and they are asynchronized. The true value of Fs Out/Fs In varies within a small range around 1.4545. **R** is in the range of 0.45 < R < 0.46. The NXP ASRC (any pure sampling converting) creates frequency aliasing above 16.5 kHz (Nyquist frequency of Fs In). After the signal is converted, filter the aliasing with a low-pass filter. The low-pass filter is an FIR filter with fc = 16.5 kHz and at least AStop < -80 dB.

In either cases, an input buffer to the NXP ASRC is used and monitored. The **R** value is calculated from the Amount Of Data of the input buffer. Update **R** at real time while the NXP ASRC and the whole system is running, so that the audio signal gets sampling frequency converted and bit rate synchronized.

3 RT1060 ASRC testing project

The demo project of this application note is `Rt1060_SoftwareAsrcTest_McuPrg`. In this demo project, we use 48 kHz USB down streaming audio as the signal source, and select one of the following methods to synchronize/convert to local I^2S either at 48 kHz or at 44.1 kHz:

- Synchronized by RT1060 Feedback End Point.
- Synchronized by RT1060 audio PLL adjusting.
- Synchronized by RT1060 NXP CM7 software ASRC.

Note:

*The NXP ASRC provided in the RT1060 demo project is a library file and an API header file. This library is **ONLY** allowed to run on NXP RT 4-digit processors.*

The IDE for building the `Rt1060_SoftwareAsrcTest_McuPrg` project is MCUXpresso IDE v24.12.

The hardware for running this demo is MIMXRT1060-EVK or MIMXRT1060-EVKB.

3.1 NXP CM7 ASRC API functions

The provided NXP ASRC library file is: *libNxpCm7ASRC.a*. The API header file is: *NxpCm7ASRC.h*. Here are the main APIs:

- `extern int InitAsrcProcessor(T_ASRCProcessor *Asrc, int FsIn, float Drift_FsOutDivFsIn, int FrmSizeInSamples, enum AsrcQualityLevel QualityLvlIdx, int UseCubic, int ChNum)`
- `extern void DeInitAsrcProcessor(T_ASRCProcessor *Asrc)`
- `extern int AsrcConvertOneFrame(T_ASRCProcessor *Asrc,float *DstPtr, float *SrcPtr)`
- `extern void UpdateAsrcRatio(T_ASRCProcessor *Asrc, double r)`
- `extern void UpdateAsrcInterp(T_ASRCProcessor *Asrc, int ToUseCubic)`

The names of the API arguments tell clearly what the arguments are. The procedure of using the NXP ASRC is ordinary. Initialize the NXP ASRC and then call the main processing function after the audio streaming is started. The drifting value and the interpolation method can be adjusted at run time. The de-initialization function releases the memory that is applied from the heap.

Note:

- *Normally, Fs In used for the initialization must be within the range of 0.5 × Fs Local < Fs In < 1.5 × Fs Local.*
- *In run time, the drifting ratio must be between 0.5 and 1.5.*
- *Higher-quality level selected needs more MIPS. MIPS in high quality (ASRC_QualityHigh) is counted and illustrated in Section 4.7.*

3.2 RT1060 system signal flow chart

The RT1060 demo project configures and enables I^2S audio and USB audio interface. To test with varieties of signal source and different working modes, a single-character USB Com command set is defined and implemented. Figure 1 shows the system internal flow chart.

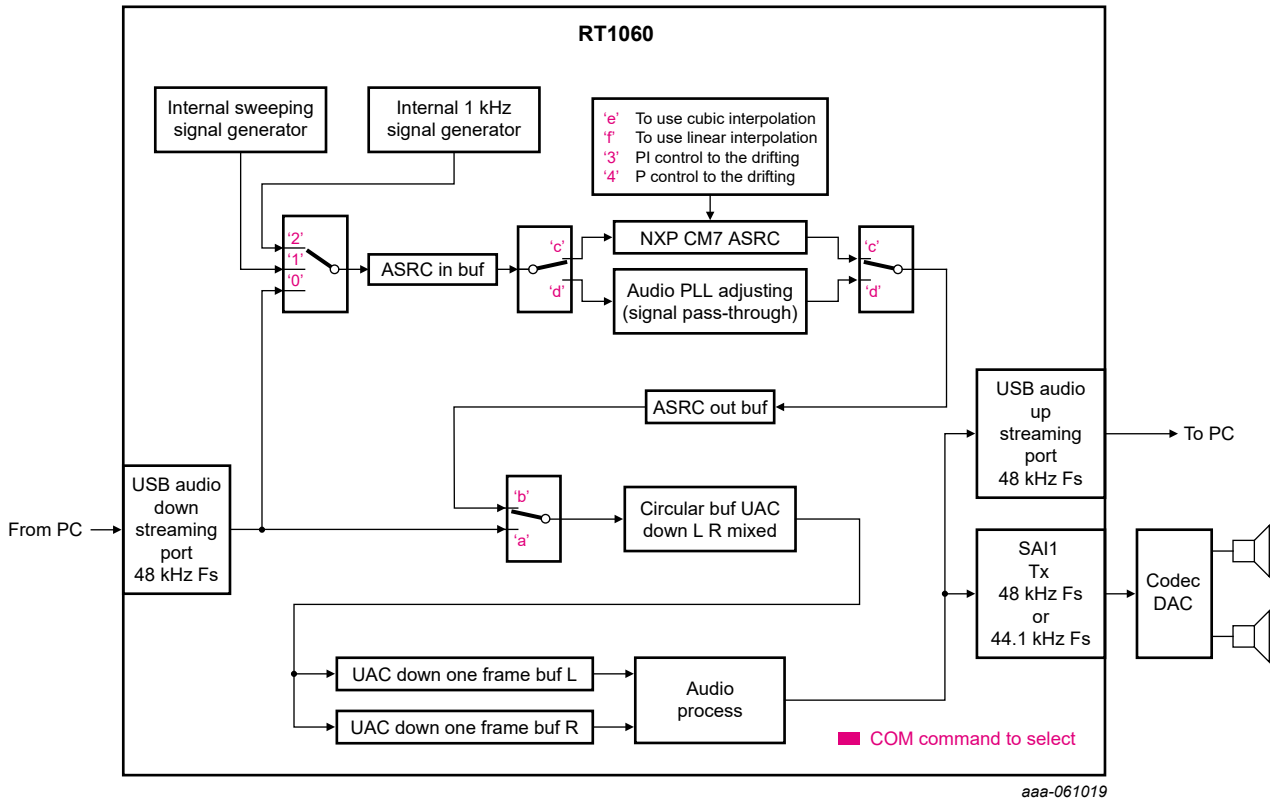

Figure 1. RT1060 internal signal flowchart

3.3 I²S audio DMA and USB audio configuration

The RT1060 demo project configures the I²S port SAI1 TX to: I²S master, DMA TX frame size = 6 samples, Fs = 48 kHz/44.1 kHz, 32-bit sample size. The USB audio interface is configured as 48 kHz, 32-bit sample size, two channels. Each USB audio down streaming packet has six sample pairs, too. A circular input buffer before the NXP ASRC and for holding the USB down streaming audio is defined, and its size is 5 ms (240 samples per channel).

The reason for I²S DMA frame size being 6 is to achieve lower DMA buffering size, lower signal latency, and easier calculation for the drifting according to the AOD of the input buffer.

3.4 USB com interface

The RT1060 project realizes both the USB Audio Class 2.0 and the USB CDC Com port：

To test with varieties of signal source and different working modes, a simple single character USB Com command set is defined and implemented in the RT1060 project. Here is the command list:

- Type and send **a/A** to select USB audio down streaming synchronized by FeedbackEP.
- Type and send **b/B** to select USB audio down streaming synchronized by Nxp CM7 ASRC or AudioPLL adjusting.
- Type and send **c/C** to select using Nxp CM7 ASRC for Usb audio down streaming synchronization.
- Type and send **d/D** to select using AudioPLL adjusting for Usb audio down streaming synchronization.
- Type and send **e/E** to select cubic interpolation.
- Type and send **f/F** to select linear interpolation.
- Type and send **0** to stop the internally generated signal overwriting the received Usb audio.

- Type and send **1** to let the internally generated sweeping tone overwrite the received Usv audio.
- Type and send **2** to let the internally generated 1 kHz tone overwrite the received Usv audio.
- Type and send **3** to select the PI control for the drift value calculation.
- Type and send **4** to select the P control for the drift value calculation.
- Type and send **PID: 123 1234 12345 123456** to set Kp, Ki, Kd, and ErrAccMax.

The RT1060 demo project also prints some watch values in the USB COM port RX window. By default, they are the PID Err accumulated value, the drifting value * 1000000, the AOD of the circular buffer and the CM7 cycle counts for calling the NXP ASRC main processing function. These values are copied to the excel table so that we can analyze the PID converge and NXP CM7 ASRC MIPS (MCPS).

3.5 USB audio playing/recording and I^{2}S audio capturing

When the RT1060 demo project is running, the RT1060-EVK board is a USB audio device. Plug the USB cable to EVK board J9 and connect it to the PC, and ensure that the recording audio enhancement and playing audio enhancement are turned off. For details, see **Section 4** in the *How to Connect Asynchronized Audio Source and Sink with RT1170 ASRC* (document AN14631). A *wav* file editor APP is required for playing and recording. We recommend OcenAudio, Audacity, or Adobe Audition.

When the demo runs in 44.1 kHz (with `#define LocalI2SFsIs44p1KHz 1`), the USB up streaming actually does not work properly. This is within expectation. Because in this demo, we only add the NXP ASRC to the down streaming path. There is no algorithm inserted in the up streaming path, while the audio up streaming does have the 44.1 kHz to 48 kHz problem, too. In this case, to analyze the NXP ASRC converted signal, use a logic analyzer to capture the I^{2}S signal waveforms, and convert to a *wav* file. The I^{2}S signal probing points are: R98, R99, R101 on RT1060-EVK or J36, J35, J37 on RT1060-EVKB. For details, see **Section 4** in the *How to Connect Asynchronized Audio Source and Sink with RT1170 ASRC* (document AN14631).

3.6 ASRC drifting calculation: P control and PI control

In the demo project of this application note, we put the NXP ASRC generated samples to a circular buffer, and monitor the Amount Of Data of this circular buffer. If the AOD is becoming more and trends to over flow, decrease the ASRC drifting value a little bit. If the AOD is becoming less and trends to under flow, increase the ASRC drifting value a little bit. We implemented the PID feedback control logic for the drifting value calculation.

PID means: Proportional Integral and Derivative. It is a feedback control logic that calculates the control value from the error value, to have the error value as close to zero as possible. In this demo, the control value is the ASRC drifting value. The error value is **the AOD - half of the circular buffer**, which means that the feedback control purpose is to keep the AOD at the half (center) of the circular buffer. In the demo, when local I^{2}S is set to 48 kHz, use P control. When local I^{2}S is set to 44.1 kHz, use PI control.

The P control calculation for the ASRC drifting value is:

$$\mathrm{AsrcDriftingValueTarget} = \mathrm{Kp} \times \mathrm{Err}$$

The PI control calculation for the ASRC drifting value is:

$$\mathrm{AsrcDriftingValueTarget} = \mathrm{Kp} \times \mathrm{Err} + \mathrm{Ki} \times \mathrm{PI\_Control\_SampleAOD\_ErrAccumulated}$$

The PID control calculation for the ASRC drifting value is:

$$\mathrm{AsrcDriftingValueTarget} = \mathrm{Kp} \times \mathrm{Err} + \mathrm{Ki} \times \mathrm{PI\_Control\_SampleAOD\_ErrAccumulated} + \mathrm{Kd} \times (\mathrm{Err} - \mathrm{PreErr})$$

Note: *The derivation part of the PID control is not used in this demo, because the Err value doesn't really change too much. Derivation part is only helpful in a system that may have a sudden Err change.*

In the above formulas:

- Kp, Ki, and Kd are the PID coefficients.

- Err is the difference between the current AOD and the target AOD (The target AOD is the half of the total circular buffer size).
- PreErr is the previous Err. It is used for getting the derivation of Err.
- `PI_Control_SampleAOD_ErrAccumulated` is the accumulated Err, that is, the integration of Err. It must be limited to a proper range.
- `AsrcDriftingValueTarget` is the calculation result used to update the Cadence ASRC drifting value

When the local I^2S Fs is set to 48 kHz, we know that the input USB audio Fs 48 kHz is very close to the local 48 kHz Fs. For most of the time, AOD is exactly the half of the circular buffer size, and Err is 0. In about every 20 ~ 60 seconds, a Non-Zero Err is seen, and we need a quick converge (Err going back to 0). This is why we perform the P control for USB audio down-streaming ASRC.

When the local I^2S Fs is set to 44.1 kHz, we select PI control, compromise the converge time, and ensure a stable drifting value, which gives lower THD+N.

The RT1060 demo of this document has already configured the good Kp, Ki coefficient values:

When the local I^2S Fs is set to 48 kHz, P control is:

$$K_p = 1/(3000 \times 50)$$

When the local I^2S Fs is set to 44.1 kHz, PI control is:

$$K_p = 1/(39500 \times 2)$$

$$K_i = 1/(1750000 \times 10)$$

Note: *The above PID coefficients are used for the analyzing of converting quality in Section 4.2 and Section 4.5.*

Additionally, to evaluate other PID coefficients, type and send **PID: 123 1234 12345 123456** in the USB COM window to set Kp, Ki, Kd, and ErrAccMax values in the demo project. In this example, new Kp = 1/123, new Ki = 1/1234, new Kd = 1/12345, new ErrAccMax = 123456.

4 Performance evaluation, ASRC converting quality

To evaluate the NXP ASRC converting quality, we record the output signal from the NXP ASRC either through USB audio up streaming or capturing/converting the local I^2S output signal, and analyze the spectrum of the recorded signal. This section evaluates the converted signal quality, the PID converge time, and the NXP ASRC MIPS (MCPS). All the following tests must use COM commands to switch to the RT1060 internal signal generator, which generates perfect sweeping tone or fixed frequency tone. This is because the USB down-streaming signal may not be the best and cleanest tone. While the RT1060 internal generator overwrites the received USB down-streaming audio, it still keeps the same data rate as if the USB host is playing the cleanest tone to the RT1060. So, with the perfect cleanest tone as the signal source, all the following analyzing of the spectrum and THD+N make sense.

4.1 Local I^2S 48 kHz - THD+N

In the RT1060 demo project, set **#define** LocalI2SFsIs48KHz 1, Set **#define** SetToneGen2To17KHz 1 or **#define** SetToneGen2To1KHz 1 to either get the internal 17 kHz tone generated or 1 kHz tone generated. Build and launch the project so that the RT1060-EVK runs with local I^2S Fs = 48 kHz.

Start playing the USB down-streaming audio and start recording the USB up-streaming audio.

In the USB COM port window:

- Type and send **b** to select USB audio down-streaming be synchronized by NXP ASRC + P control.
- Type and send **2** to let the RT1060 internal tone generator overwrite the received USB audio with 17 kHz tone or 1 kHz tone. This makes it easier to observe the recorded signal spectrum.

The recording may last for two minutes or more. From the recorded audio, and also from the COM printing window, every 20 ~ 40 seconds, the PID control logic sees the Err value change to non-zero and pushes the Err value back to **0**. While the PID is doing this, the recorded converted audio quality degrades a little bit. This is because the ASRC causes a little bit of distortion.

To quantize the distortion, in the `wav` APP (Adobe audition), we apply a notch filter to the recorded single frequency tone. In this test case, we set the Fc of the notch filter to 17 kHz or 1 kHz. Ideally, if there is no harmonic distortion, after the filtering, the signal level is close to zero, practically smaller than -130 dB ~ -110 dB. Because the converted signal does have some distortion, the level of the notch filtered signal is a larger value, between -80 dB and -110 dB. This is the method how we evaluate the THD+N of the converted signal.

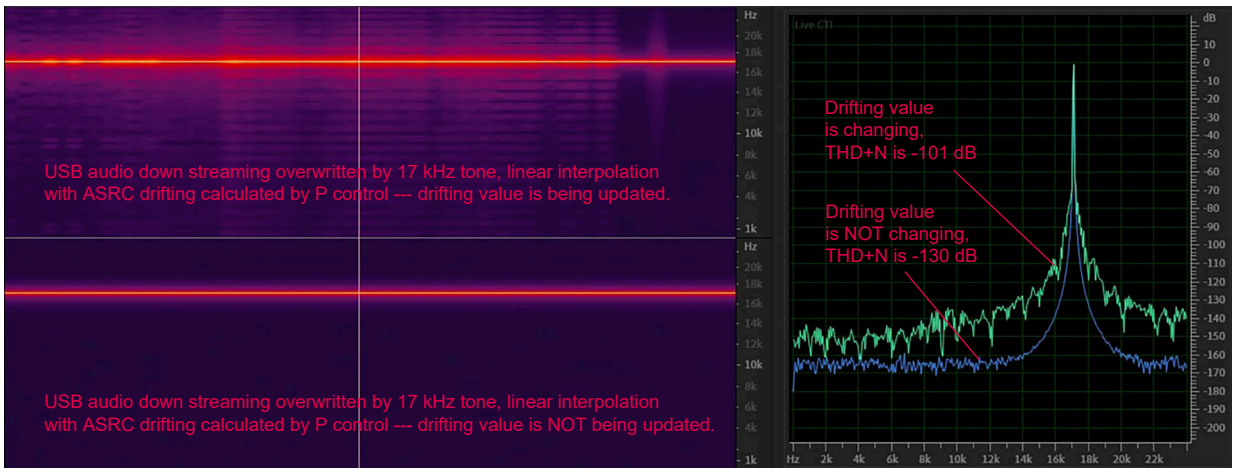

Figure 2. Converted signal THD+N at 17 kHz - 48 kHz local I^2S

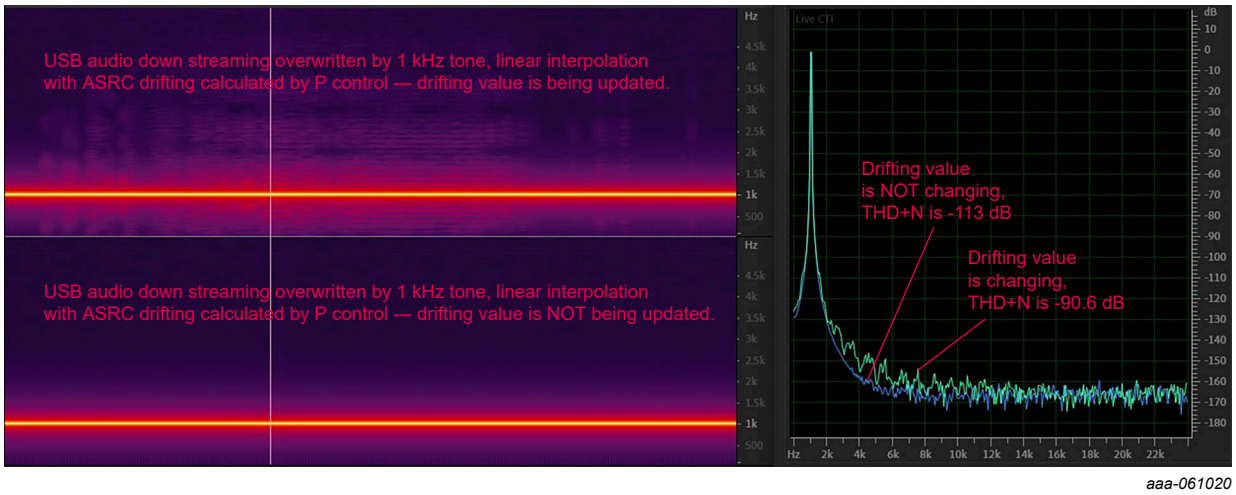

Figure 3. Converted signal THD+N at 1 kHz - 48 kHz local I²S

4.2 Local I²S 48 kHz - Converge time

Follow the setup in Section 4.1. Start the USB audio playing and recording. Copy the values in the COM printing window to an excel table, and we get the converge curve. Once the Err value is changed to a non-zero value, the P control pushes it back to 0 within about nine seconds, as shown in Figure 4.

Note: *ErrValue equals to buffer AOD - 120 (half of the buffer). The drifting value in the converging is smaller than 0.004 %. The nine-second converging is not audible.*

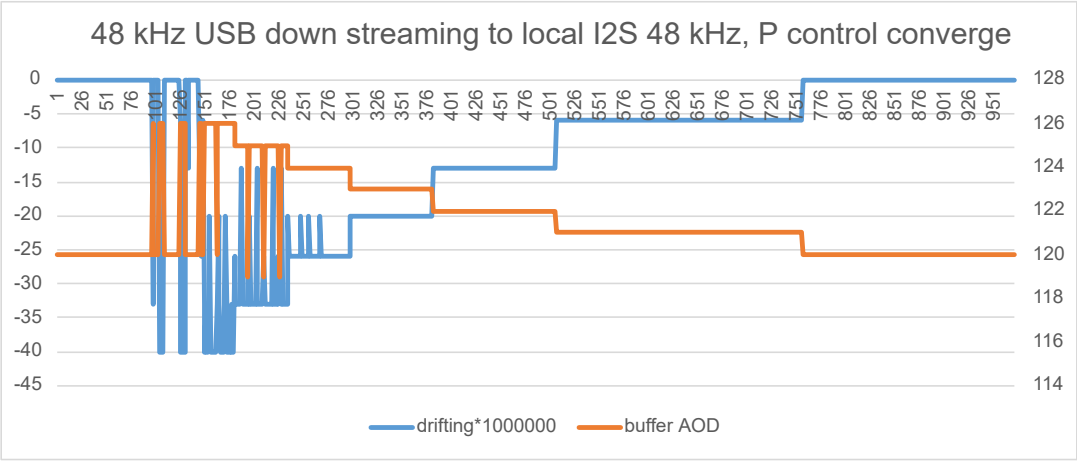

Figure 4. P control converge, Drifting, and AOD - 48 kHz local I²S

4.3 Local I²S 44.1 kHz - Sweeping

In the RT1060 demo project, set **#define** LocalI2SFsIs44p1KHz 1. Build and launch the project so that the RT1060-EVK runs with local I²S Fs = 44.1 kHz.

Start playing the USB down streaming audio and start recording the USB up streaming audio.

In the USB COM port window:

- Type and send **b** to select USB audio down streaming to be synchronized by NXP ASRC + PI control.

- Type and send **1** to let the RT1060 overwrite the received USB audio with the internally generated sweeping tone.
- Type and send **f** to select linear interpolation, or **e** to select cubic interpolation.

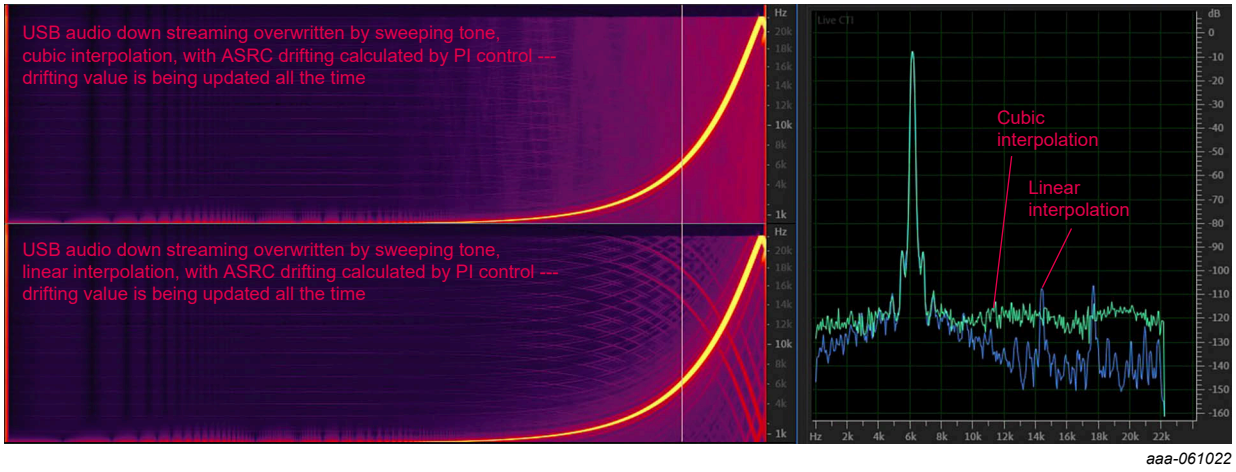

Figure 5. Converted sweeping signal, Linear VS Cubic - 44.1 kHz local I^{2}S

As shown in Figure 5, when converting audio from 48 kHz to 44.1 kHz, the cubic interpolation gives a higher spectrum noise floor, while the linear interpolation gives higher spectrum at two certain frequency points. Section 4.4 describes the THD+N at certain frequencies.

4.4 Local I^{2}S 44.1 kHz - THD+N

In the RT1060 demo project, set **#define** LocalI2SFsIs44p1KHz 1. Set **#define** SetToneGen2To17KHz 1 or **#define** SetToneGen2To1KHz 1 to either get the internal 17 kHz tone generated or 1 kHz tone generated. Build and launch the project so that the RT1060-EVK runs with local I^{2}S Fs = 44.1 kHz.

Start playing the USB down streaming audio and start recording the USB up streaming audio.

In the USB COM port window:

- Type and send **b** to select USB audio down streaming to be synchronized by NXP ASRC + PI control.
- Type and send **1** to let the RT1060 overwrite the received USB audio with the internally generated 1 kHz/17 kHz tone.
- Type and send **f** to select linear interpolation, or **e** to select cubic interpolation.

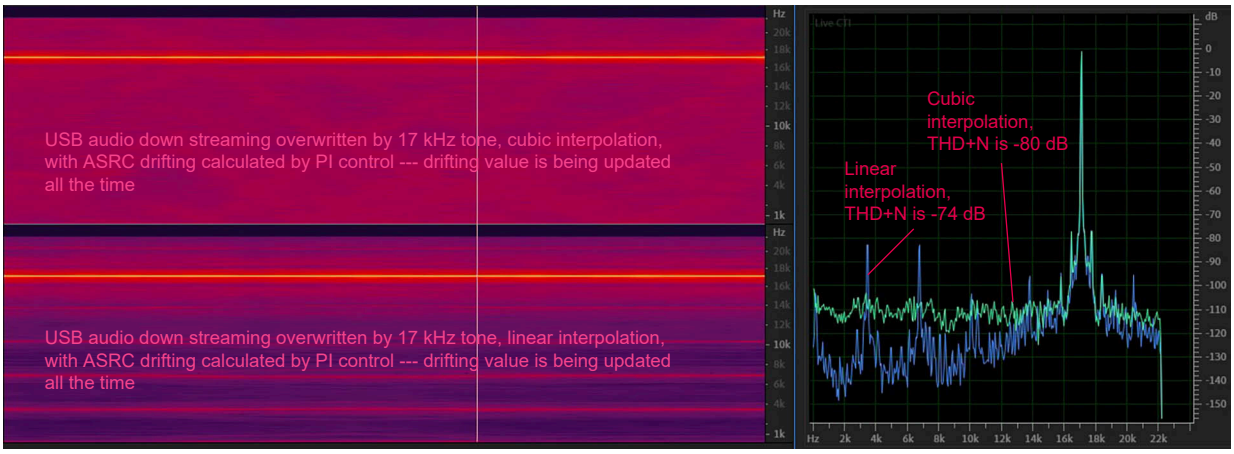

Figure 6. Converted 17 kHz tone THD + N, Cubic VS Linear - 44.1 kHz local I²S

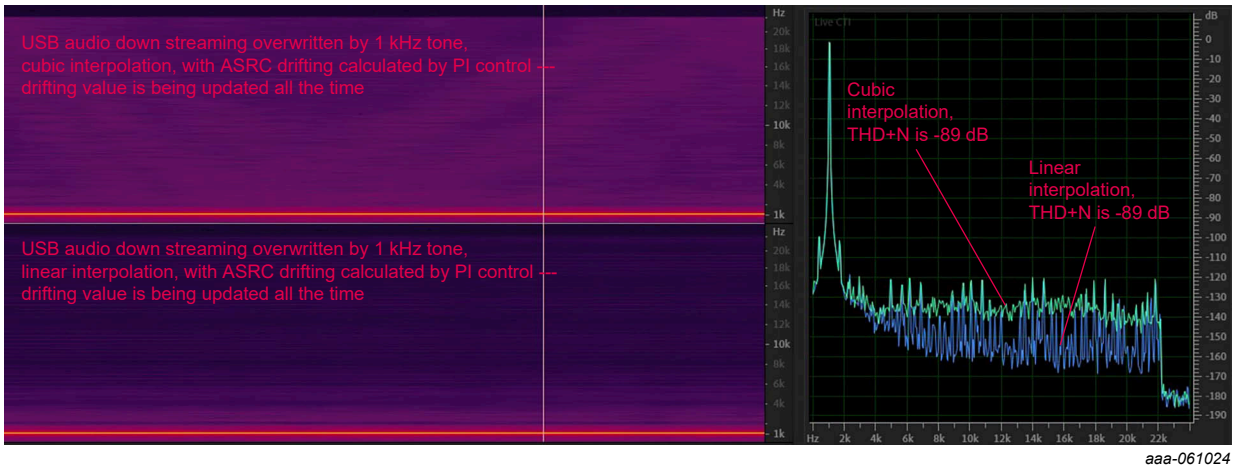

Figure 7. Converted 1 kHz tone THD + N, Cubic VS Linear - 44.1 kHz local I²S

We can see, when converting 48 kHz to local 44.1 kHz and with PI control, cubic interpolation gives lower THD + N (-80 dB) at 17 kHz. The THD + N at 1 kHz is the same (-89 dB).

4.5 Local I²S 44.1 kHz - Converge time

Follow the setup in Section 4.4.

Type and send **3** to select NXP ASRC + PI control. Switching from P control to PI control resets the PI control calculation, and you can see the converge.

Type and send **4** to select NXP ASRC + P control. Switching from PI control to P control resets the P control calculation, and you can see the converge.

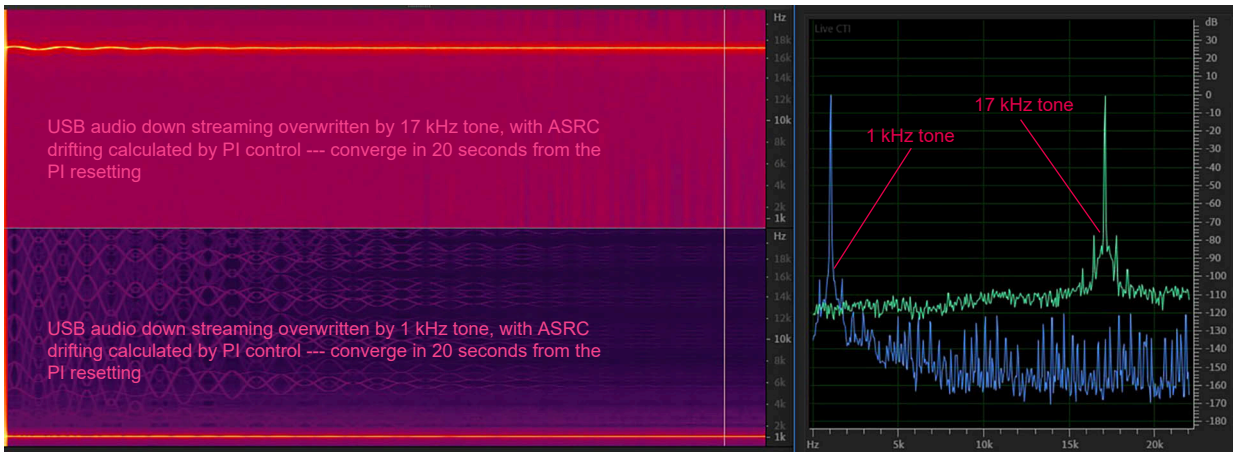

Figure 8. PI control converge, 17 kHz and 1 kHz - 44.1 kHz local I²S

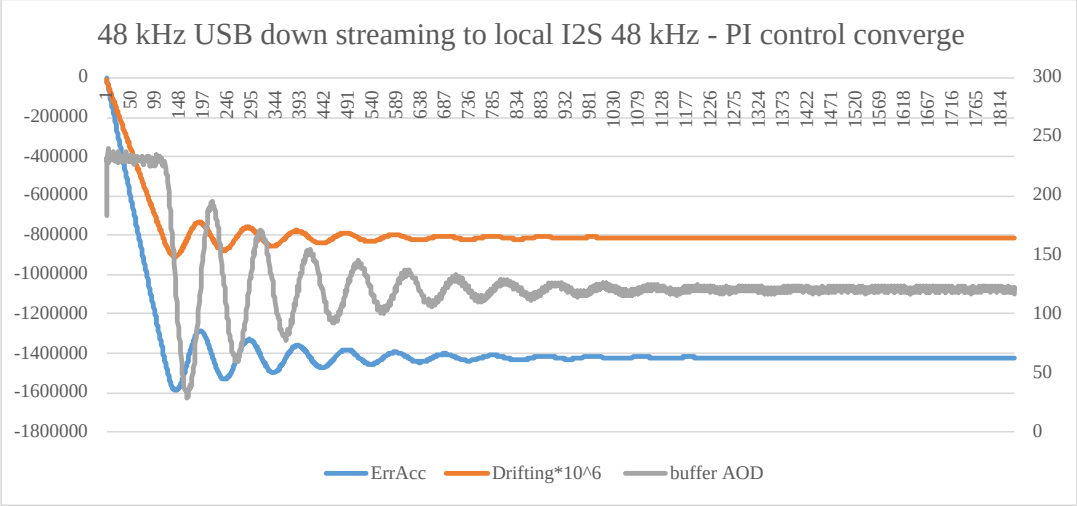

Figure 9. PI control converge, ErrAcc, Drifting, and AOD - 44.1 kHz local I²S

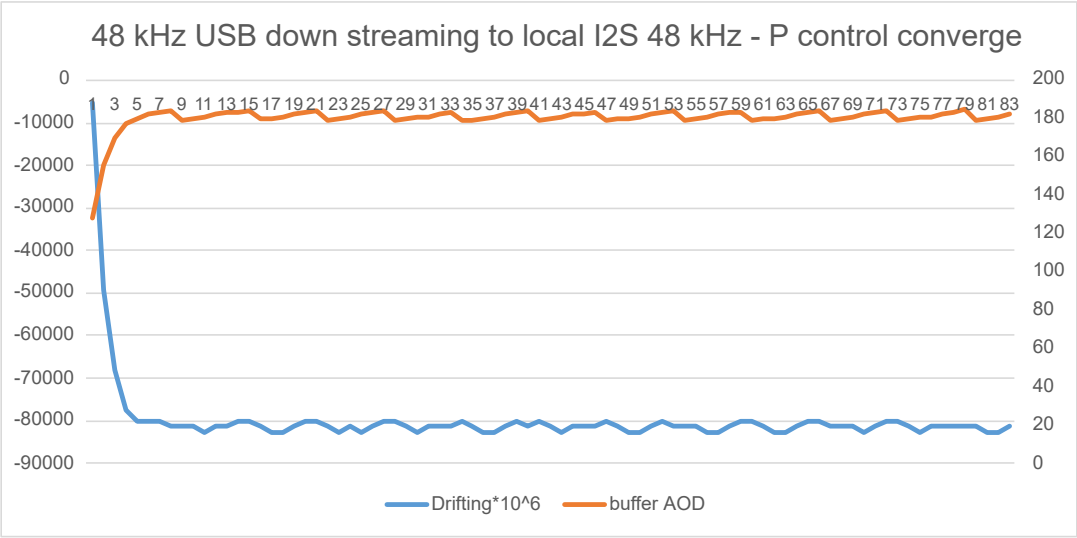

Figure 10. P control converge, Drifting, and AOD - 44.1 kHz local I²S

As shown in Figure 9, the converge time of PI control is long: 20 seconds (X-axis unit is 18 ms). But only the first three seconds may give audible pitch shifting. Once it is fully converged, the drifting value keeps in a small range: -8.129 % ~ -8.123 %.

As shown in Figure 10, the converge time of P control is as short as 0.1 seconds (X-axis unit is 18 ms), but we get a larger final converged drifting value range: -8.266 ~ -8.000 %. It is 37 times bigger than the PI control. This is why PI control gives lower THD+N (only -50 dB) compared with P control in this case.

4.6 NXP ASRC latency

In the RT1060 demo project, set **#define** EnableAsrcLatencyTest 1. Build and run the demo. Use the single-character command “b” to select to use NXP ASRC for the USB audio down streaming synchronization. Use the single-character command **2** to enable the internal 1 kHz tone generator to overwrite the received audio.

In this scenario, make a sudden amplitude decrease once every 1 second (or similar) in the left channel in the internal tone-sweeping generator. And when the internal tone-sweeping generator decides to change the amplitude, the USB up streaming right channel amplitude is also decreased immediately. So, the USB audio up-streaming right channel reflects the signal amplitude change immediately, while the left channel amplitude change is delayed by the ASRC converting.

Observing the recorded USB up-streaming audio, we can exactly measure the latency in samples. As shown in Figure 11, the latency = 139 samples, which consist of two parts: the USB audio buffer (ASRC input buffer) latency 120 samples and the actual ASRC latency. So, the actual NXP ASRC latency is 139 - 120 = 19 samples (0.4 ms at 48 kHz Fs).

Figure 11. NXP ASRC latency - USB 48 kHz down streaming to local I^2S 48 kHz

4.7 NXP ASRC MIPS

This demo sets ASRC quality to `ASRC_QualityHigh`. In this section, we measure and calculate the MIPS in `ASRC_QualityHigh`.

The last column of the data received in the USB COM RX port window is the CM7 cycle counts of each time calling the NXP ASRC main processing function(two channels 48 kHz to 48 kHz/44.1 kHz audio processing).

Table 1. MIPS measured from the cycle counts printed in the COM port - Two channels configured

	Cycles used in 6 samples period (averaged)		Equal MCPS/MIPS (averaged)	
	Cubic interpolation	Linear interpolation	Cubic interpolation	Linear interpolation
Local I^2S: 44.1 kHz	13387	7786	98	57
Local I^2S: 48 kHz	14472	8390	116	67

The cycle counts are measured when the NXP ASRC is configured to 1/2/3/4 channels. FsIn is set to **48000**, and the drifting ratio is set to **1.01**.

Table 2. MIPS measured for channel number set to 1, 2, 3, 4

K cycles for processing 6 sample input - 48 kHz input Fs		
Ch Num	Cubic interpolation	Linear interpolation
1	7.7	4.7
2	14.6	8.7
3	21.3	12.7
4	27.9	16.6
MIPS - 48 kHz input Fs		
Ch Num	Cubic interpolation	Linear interpolation
1	61.6	37.6
2	116.8	69.6
3	170.4	101.6
4	223.2	132.8

The memory needed for running the NXP ASRC is small: 5 K ~ 10 K bytes depends on the channel number.

5 Synchronization by RT1060 audio PLL

Besides using NXP ASRC to handle the USB audio synchronization, another approach is to adjust the RT1060 Audio PLL. The frequency of the Audio PLL is set by `CCM_ANALOG_PLL_AUDIO_NUM`, `CCM_ANALOG_PLL_AUDIO_DENOM`, and the `DIV_SELECT` field in `CCM_ANALOG_PLL_AUDIO`. The Audio PLL output frequency equals to:

$$\text{FreqClkSource} \times \left(\text{DIV\_SELECT} + \text{AUDIO\_NUM} / \text{AUDIO\_DENOM} \right)$$

With the following constrains:

- The input clk source must be 24 MHz.
- The fractional part `AUDIO_NUM`/`AUDIO_DENOM` must be within -1.0 to +1.0.
- The Audio PLL output frequency must be within 650 MHz to 1300 MHz.

By monitoring the AOD of the USB down streaming buffer, and with the PID logic (PI or P control), we calculate and know the drifting value, which means how much the speed of consuming the received USB audio samples should be increased or decreased. Instead of updating the drifting ratio value of the NXP ASRC and calling the NXP ASRC, change the numerator value in the `CCM_ANALOG_PLL_AUDIO_NUM` register.

In this RT1060 demo, the audio PLL fOut is configured as:

$$f_{Out} = 24\ \text{MHz} \times \left(33 + \left(-23200\right) / 100000\right) / 1 = 24 \times 32.768 = 786.432\ \text{MHz}$$

Then the audio PLL fOut is to be divided by 64 and we get the commonly used 12.288 MHz MCLK for the local I^2S 48 kHz.

If you want the I^2S output streaming to consume samples a little faster, increase the NUMERATOR value up to 99999. So, we can get audio PLL fOut (24 MHz × 33.99999) increased by 3.75 %. On the contrary, decrease the NUMERATOR value down to -99999 to get fOut (24 MHz × 32.00001) decreased by 2.3 %.

When the demo works at 48 kHz local I^2S, type and send **d** to select adjusting the audio PLL and by-passing the NXP ASRC, while the PID feedback control logic keeps unchanged. In this scenario, no additional audio samples are generated or removed, and all the audio samples received are to be streamed out without any modification. So, when performing the USB up streaming audio recording, no distortion and any additional noise floor can be seen in the spectrum. But, the I^2S clock is jittering while the PID control logic is taking effect. Distortion of the audio signal in the analog domain can be measured by dedicated equipment (AP, for example). Analog signal quality measuring and analyzing is not in the scope of this AN.

When the demo works at 44.1 kHz local I^2S, increase the audio PLL fOut by 8.84 % (48/44.1=1.0884). This is out of the possible range of -2.3 % ~ 3.75 %. Use NXP ASRC in this case.

So, it is clear that when the expected frequency drifting ratio is within the range of -2.3 % ~ 3.75 %, use audio PLL adjusting for the synchronization, because it is MIPS free. If the frequency drifting range is outside -2.3 % ~ 3.75 %, use the NXP ASRC, with consuming a reasonable amount of MIPS.

6 Conclusion

- When you want to connect the asynchronized audio source and sink and there is no hardware ASRC, use software ASRC or audio PLL adjusting.
- NXP ASRC is a highly efficient library that can convert asynchronized audio with drifting change at real-time and the converted signal THD+N is lower than -80 dB.
- Set a circular buffer with the proper buffer size for as low latency as possible and to support PID feedback control to the drifting value.
- The ASRC drifting is controlled by PID feedback control. Tune the PID coefficients to get the PID converged as quick as possible. The Err has as small jittering as possible.
- If the expected frequency drifting range is within -2.3 % and 3 .75 %, use audio PLL adjusting for the synchronization, because it is MIPS free.
- NXP ASRC has two interpolation options: linear and cubic. Cubic interpolation gives lower THD+N but almost double the MIPS compared with linear interpolation. Make the decision according to the MIPS consumption in the APP.

7 Acronyms

Table 3. Acronyms

Acronyms	Description
AOD	Amount Of Data
SSRC	Synchronized Sample Rate Converting
ASRC	Asynchronized Sample Rate Converting
P control	Proportional control
PI control	Proportional Integral control
PID control	Proportional Integral Derivative control
MIPS	Million Instructions Per Second
MCPS	Million Cycles Per Second

8 Revision history

Table 4 summarizes the revisions to this document.

Table 4. Revision history

Document ID	Release date	Description
AN14693 v1.0	05 June 2025	Initial public release

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Contents