UG10087

Asynchronous Sample Rate Converter

Rev. 1.0 — 10 January 2024 User guide

Document information

Information	Content
Keywords	Asynchronous Sample Rate Converter, ASRC, UG10087
Abstract	The Asynchronous Sample Rate Converter (ASRC) software module compensates the drift between two mono audio signals. This is not a frequency converter and so the nominal signal frequency is the same before and after the ASRC.

1 Introduction

The Asynchronous Sample Rate Converter (ASRC) software module compensates the drift between two mono audio signals. This is not a frequency converter and so the nominal signal frequency is the same before and after the ASRC.

The system works asynchronously. It compensates the difference between the input and output sampling rates with different clock domains.

2 ASRC overview

The ASRC overview describes the problem and the solution.

2.1 Problem

For the use case with two different clock domains. Due to clock drift, the audio sample rates are not 100 % identical. It is a result of a data buffering error. Therefore, at a particular moment, there is either too much data or not enough data. For audio use cases, it results of an audio artifact.

Example:

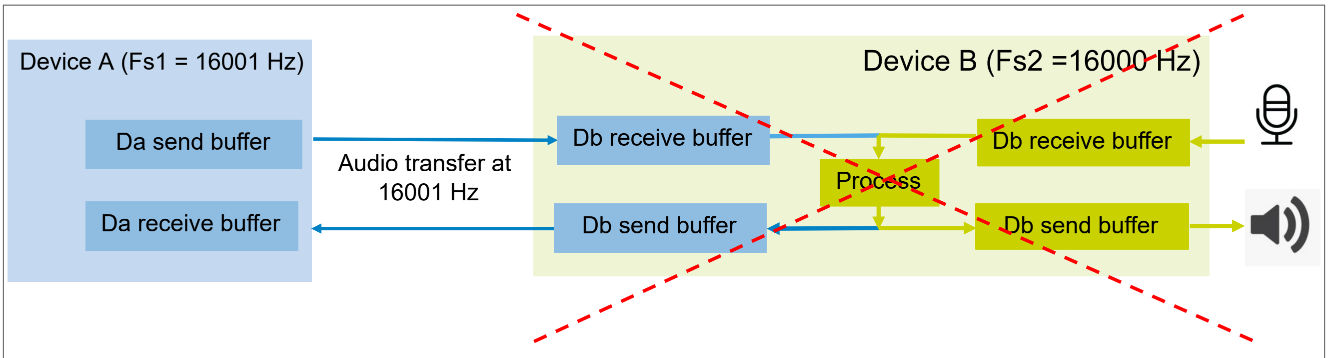

Figure 1. ASRC overview problematic

- Audio samples captured at $fs1$= 16001 Hz by device B due to audio link transfer
- The device B consumed the audio sample at $fs2$= 16000 Hz

At a moment:

- There are too many samples in the device B receive buffer.
- There are no more samples in the device B send buffer.
- It generates an audio artifact.

2.2 Solution

The solution is to use an ASRC) block to estimate the sample rate ratio and correct it. The block helps adjust the number of audio data consumed and generated by the device.

Example:

The ASRC block estimates the **sample rate ratio (** $\gamma = \frac{fs1}{fs2}$ **)** between frequency 1 and frequency 2**.**

It helps adjust the number of audio data consumed and generated by the device B process and adjust it to its own frequency.

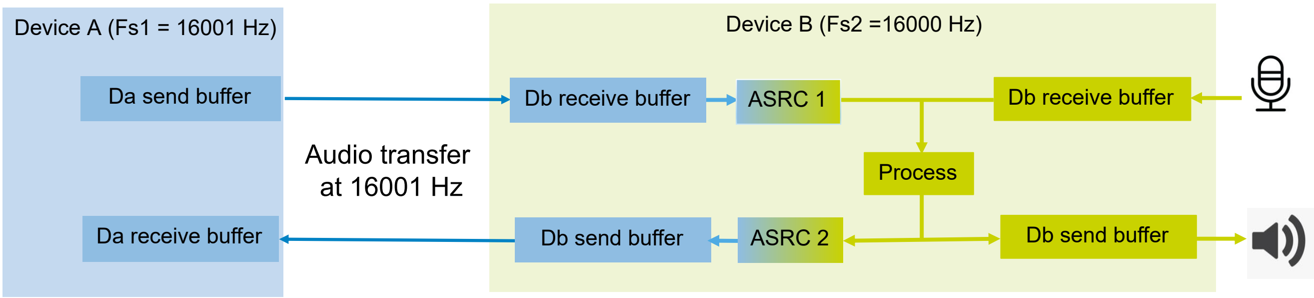

Figure 2. ASRC overview solution

3 Application programmer interface (API)

All public structure, function, and definition are available in the **ASRC.h** header file.

3.1 ASRC enums and structures

This topic lists the following ASRC enums and structures.

- Section 3.1.1 "ASRC_LibInfo_st"
- Section 3.1.2 "ASRC_ReturnStatus_en"
- Section 3.1.3 "ASRC_OperatingMode_en"
- Section 3.1.4 "ASRC_Fs_en"
- Section 3.1.5 "ASRC_InstanceParams_st"
- Section 3.1.6 "ASRC_ControlParams_st"

3.1.1 ASRC_LibInfo_st

```
/* ASRC_Lib_Info_st structure */
typedef struct
{
    PL_UINT8                    ASRC_LIB_Release_MAJOR;
    PL_UINT8                    ASRC_LIB_Release_MEDIUM;
    PL_UINT8                    ASRC_LIB_Release_MINOR;

} ASRC_LibInfo_st;
```

Figure 3. ASRC library version

3.1.2 ASRC_ReturnStatus_en

```
// Error type
typedef enum
{
    ASRC_SUCCESS,                               /// Successful return from a routine
    // invalid silicon
    ASRC_INVALID_SILICON_CHECK,                 /// NXP board check not valid
    // invalid parameters
    ASRC_INVALID_BUFFER_MEMORY_ALIGNMENT,       /// Memory alignment error
    ASRC_INVALID_NULL_ADDRESS,                   /// Memory alignment error
    // Error in given ASRC_InstanceParams_st
    ASRC_INVALID_INPUT_FRAME_SIZE,              /// InputSamplesPerFrame parameter not supported
    ASRC_INVALID_OUTPUT_FRAME_SIZE,             /// OutputSamplesPerFrame parameter not supported
    ASRC_INVALID_FRAME_SIZE,
    ASRC_INVALID_TARGET_FREQUENCY,              /// targetFs not supported
    ASRC_INVALID_PI_CONTROLLER_ON,              /// PIControllerON parameter is not boolean value
    ASRC_INVALID_ASRC_ENABLE,                   /// ASRC_Enable parameter is not boolean value
    ASRC_INVALID_ALPHA,                         /// alpha parameter not supported
    ASRC_INVALID_BETA,                          /// beta parameter not supported
    ASRC_INVALID_LAMBDA,                        /// lambda parameter not supported
    ASRC_INVALID_OPERATING_MODE,                /// OperatingMode parameter not supported
    ASRC_INVALID_SYSTEM_CLOCK,                  /// systemClock parameter not supported
    // circular buffer error cases
    ASRC_CIRCULAR_BUFFER_OVERFLOW,              /// read and write pointer equals
    ASRC_CIRCULAR_BUFFER_FULL,                  /// not space to write  in circular buffer
    ASRC_CIRCULAR_BUFFER_EMPTY,                 /// not enough data to read  in circular buffer
    ASRC_CIRCULAR_BUFFER_ERROR,
    /// general errors
    ASRC_ERROR_UNDEFINED,                       /// undefined error
    ASRC_SYSTEM_ERROR,                          /// Unknow error
    ASRC_NB_ERROR                               /// number of error

} ASRC_ReturnStatus_en;
```

Figure 4. ASRC error type

3.1.3 ASRC_OperatingMode_en

Helps select between the two modes:

- **Master**: ASRC computes the ratio based on input and output ASRC internal circular buffer level
- **Slave**: ASRC used a given external ratio

```
/* ASRC Operating Mode
 */
typedef enum
{
    ASRC_DISABLE        ,                       ///  module deactivated
    ASRC_MASTER_MODE    ,                       ///  evaluate and fix gamma value within ASRC
    ASRC_SLAVE_MODE     ,                        ///  user fix gamma value
    ASRC_NUMBER_OF_OPERATING_MODE

} ASRC_OperatingMode_en;
```

Figure 5. ASRC operating mode

3.1.4 ASRC_Fs_en

The list of supported frequency sample rate. The current library was tested only for Fs equal to 16,000 Hz. To support different sample rates, contact the NXP support.

```
typedef enum
{
    ASRC_FS_16000 = 0 , // 16kHz sampling rate
    ASRC_FS_INVALID

} ASRC_Fs_en;
```

Figure 6. Frequency sample rate

3.1.5 ASRC_InstanceParams_st

```
typedef struct
{
    ASRC_OperatingMode_en   OperatingMode;          /// see enum
    PL_UINT16               InputSamplesPerFrame;   /// [| 41 - 120 |]     Number of input samples per frame
    PL_UINT16               OutputSamplesPerFrame;  /// [| 41 - 120 |]     Number of output samples per frame
    ASRC_Fs_en              targetFs;               /// { 16000 }          Estimate frequency of input and output
    PL_UINT32               systemClock;            /// [| 50MHz - 2GHz |] System Platform clock rate

} ASRC_InstanceParams_st;
```

Figure 7. Used to create an ASRC instance.

3.1.6 ASRC_ControlParams_st

```
/* Control Parameter structure */
typedef struct
{
    PL_BOOL        PIControllerON; /// { true, false } Enable analysis part of ASRC
    PL_BOOL        ASRC_Enable;    /// { true, false } Enable ASRC PIController and synthesis : if false ASRC is transparent

    PL_FLOAT       alpha_copy;     /// [ -0.01 - 0.01]  Copy of the Proportional gain of the PI Controller
                                   /// Default :  -0.00015, if increased, faster convergence but larger oscillation

    PL_FLOAT       beta_copy;      /// [ -0.01 - 0.01] Copy of the Integral gain of the PI Controller
                                   /// Default : -0.00000005, if increased, the history get more importance in the calculation but can cause overshoot

    PL_FLOAT       lambda_copy;    /// [ 0 - 1]  Copy of the smoothing of parameter for AOD_corrected computation
                                   /// Default :  0.08, if increased, faster convergence but larger oscillation
} ASRC_ControlParams_st;
```

Figure 8. Used to control the ASRC instance

Default value: optimum configuration for 10 Hz drift:

- λ = 0.08
- α= -0.00015
- β = -0.00000005

See code comments for tuning the controller parameters.

3.2 Functions

This topic lists the following ASRC functions.

- Section 3.2.1 "ASRC_GetLibInfo"
- Section 3.2.2 "ASRC_GetMemoryTable"
- Section 3.2.3 "ASRC_GetInstanceHandle"
- Section 3.2.4 "ASRC_SetControlParameters"
- Section 3.2.5 "ASRC_GetRatio"
- Section 3.2.6 "ASRC_GetControlParameters"
- Section 3.2.7 "ASRC_Push"
- Section 3.2.8 "ASRC_Pull"
- Section 3.2.9 "ASRC_Process"

3.2.1 ASRC_GetLibInfo

```
/**
 * @brief
 *
 * @param pLib_Info
 * @return ASRC_ReturnStatus_en
 */
ASRC_ReturnStatus_en ASRC_GetLibInfo(ASRC_LibInfo_st    *pLib_Info);
```

Figure 9. Get the ASRC library information

3.2.2 ASRC_GetMemoryTable

Get the amount of memory required by the ASRC library.

```
/**
 * @brief
 *
 * @param hInstance
 * @param pMemoryTable
 * @param pInstanceParams
 * @return ASRC_ReturnStatus_en
 */
ASRC_ReturnStatus_en ASRC_GetMemoryTable(ASRC_Handle_t         hInstance,
                                        PL_MemoryTable_st      *pMemoryTable,
                                        ASRC_InstanceParams_st  *pInstanceParams);
```

Figure 10. Get the amount of memory required

3.2.3 ASRC_GetInstanceHandle

```
/**
 * @brief
 *
 * @param phInstance
 * @param pMemoryTable
 * @param pInstanceParams
 * @return ASRC_ReturnStatus_en
 */
ASRC_ReturnStatus_en ASRC_GetInstanceHandle(  ASRC_Handle_t              *phInstance,
                                              PL_MemoryTable_st         *pMemoryTable,
                                              ASRC_InstanceParams_st     *pInstanceParams);
```

Figure 11. Create an ASRC library instance

3.2.4 ASRC_SetControlParameters

Set the ASRC control parameters. It is used to update the control parameter after an ASRC instance is created. It can be used when the use case is running.

```
/**
 * @brief
 *
 * @param phInstance
 * @param pNewParams
 * @return ASRC_ReturnStatus_en
 */
ASRC_ReturnStatus_en ASRC_SetControlParameters( ASRC_Handle_t               phInstance,
                                                ASRC_ControlParams_st   *const pNewParams);
```

Figure 12. Set the ASRC control parameters

3.2.5 ASRC_GetRatio

```
/**
 * @brief
 *
 * @param asrcHandler
 * @param gamma
 * @return ASRC_ReturnStatus_en
 */

ASRC_ReturnStatus_en ASRC_GetRatio(ASRC_Handle_t asrcHandler,
                                   PL_FLOAT *gamma);
```

Figure 13. Get the current ASRC sampling rate frequency ratio

3.2.6 ASRC_GetControlParameters

```
/**
 * @brief
 *
 * @param phInstance
 * @param pControlParams
 * @return ASRC_ReturnStatus_en
 */
ASRC_ReturnStatus_en ASRC_GetControlParameters( ASRC_Handle_t          phInstance,
                                                ASRC_ControlParams_st  *pControlParams);
```

Figure 14. Read the current ASRC control parameters

3.2.7 ASRC_Push

It helps add a frame to the ASRC instance for further process. This frame is at the initial sample rate.

```
/**
 * @brief push input frame
 *
 * @param asrcHandler
 * @param inputFrame
 * @return ASRC_ReturnStatus_en
 */
ASRC_ReturnStatus_en ASRC_Push(ASRC_Handle_t asrcHandler,
                               const PL_FLOAT *inputFrame);
```

Figure 15. Add a frame to the ASRC instance

3.2.8 ASRC_Pull

It helps read a frame for the ASRC instance. This frame is at the final sample rate. Final sample rate = Initial sample rate -/+ ratio.

```
/**
 * @brief
 *
 * @param asrcHandler
 * @param outputFrame
 * @param framesize
 * @param timestamp
 * @return ASRC_ReturnStatus_en
 */
ASRC_ReturnStatus_en ASRC_Pull(ASRC_Handle_t asrcHandler,
                                const PL_FLOAT *outputFrame,
                                PL_UINT16 framesize,
                                PL_UINT32 timestamp);          // according to systemClock
```

Figure 16. Read a frame for the ASRC instance

3.2.9 ASRC_Process

It launches the ASRC instance process. The ASRC process consumes a frame from the ASRC input buffer and writes a frame to the ASRC output buffer. The number of samples read and write are different according to the ratio between the two frequencies. This process helps compensate a clock drift and does not help convert the sample rate (this is not SRC).

```
/**
 * @brief Interpolate output sample if delay line ready
 *
 * @param asrcHandler
 * @param timestamp
 * @param userGammaRatio
 * @return ASRC_ReturnStatus_en
 */
ASRC_ReturnStatus_en ASRC_Process(ASRC_Handle_t asrcHandler,
                                   PL_UINT32 timestamp,   // according to systemClock
                                   PL_FLOAT userGammaRatio);
```

Figure 17. Launches the ASRC instance process

3.3 Sequence

This topic describes the Section "Create an ASRC instance" and the Section "Run the ASRC in the process loop".

3.3.1 Create an ASRC instance

To create and configure an ASRC instance, follow the sequence in the application code.

The ASRC library does not support the grey part in Figure 18. There, make sure to code the sequence in the application source code.

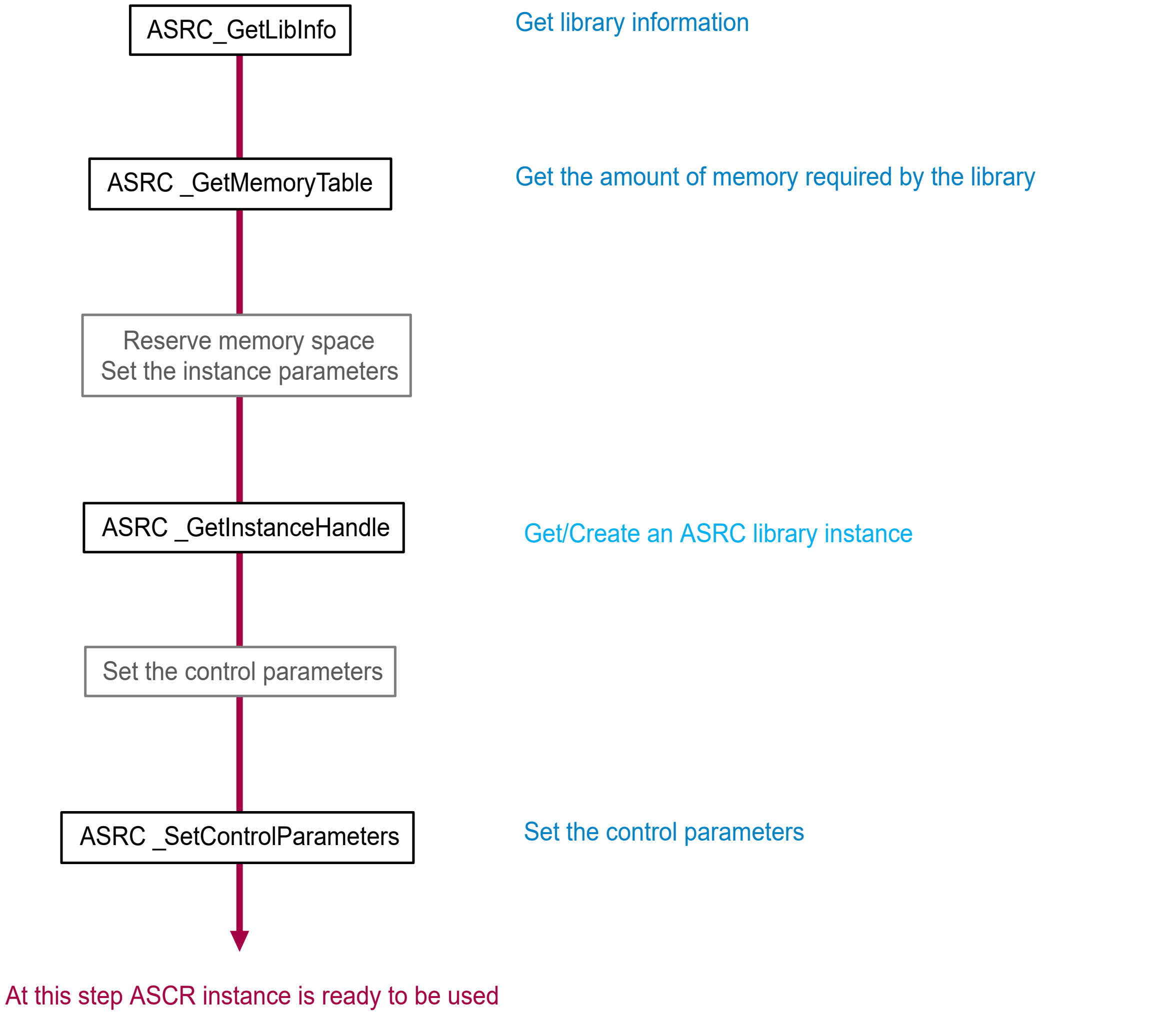

Figure 18. ASRC create instance sequence

3.3.2 Run the ASRC in the process loop

To explain the use of ASRC, let us consider an audio flow transmission between a Device A and a Device B.

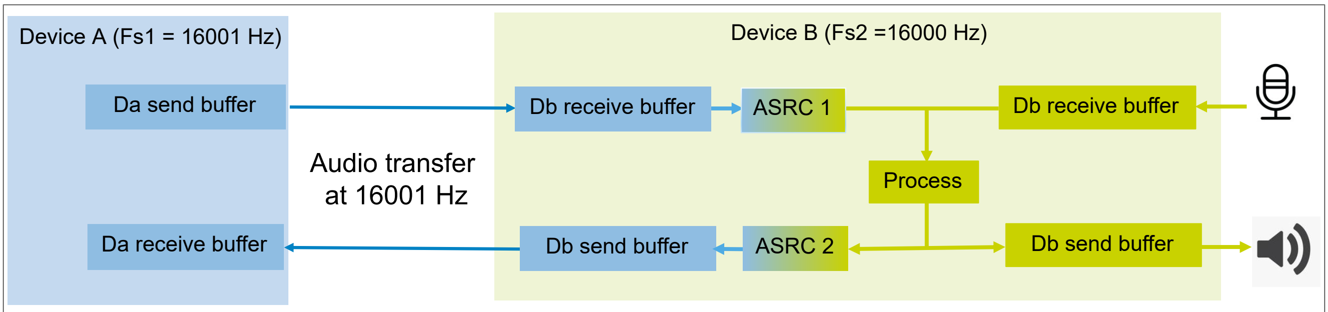

Figure 19. ASRC audio use case example

- Device A (Fs1 frequency domain) has an audio link with device B (Fs2 frequency domain). Device A is a master.
- 2 ASRC instances are required in device B:
  – ASRC1 in master mode: convert audio stream from blue to green frequency domain.

– ASRC2 in slave mode: convert audio stream from green to blue frequency domain.

Device A does not have ASRC. It is device B, which handle the ASRCs instance.

Device B source code has got:

- Frame-based audio process loop. This loop is based on hardware dependency based on Fs2 sample rate.
- Hardware resource interrupt handler (Speaker and Microphone in our example) based on Fs2 sample rate.
- Audio transfer interrupt handler to handle hardware resource.

Linktext-Figure_ schematics show how to insert the ASRC process in the loop and in the different interrupt handler.

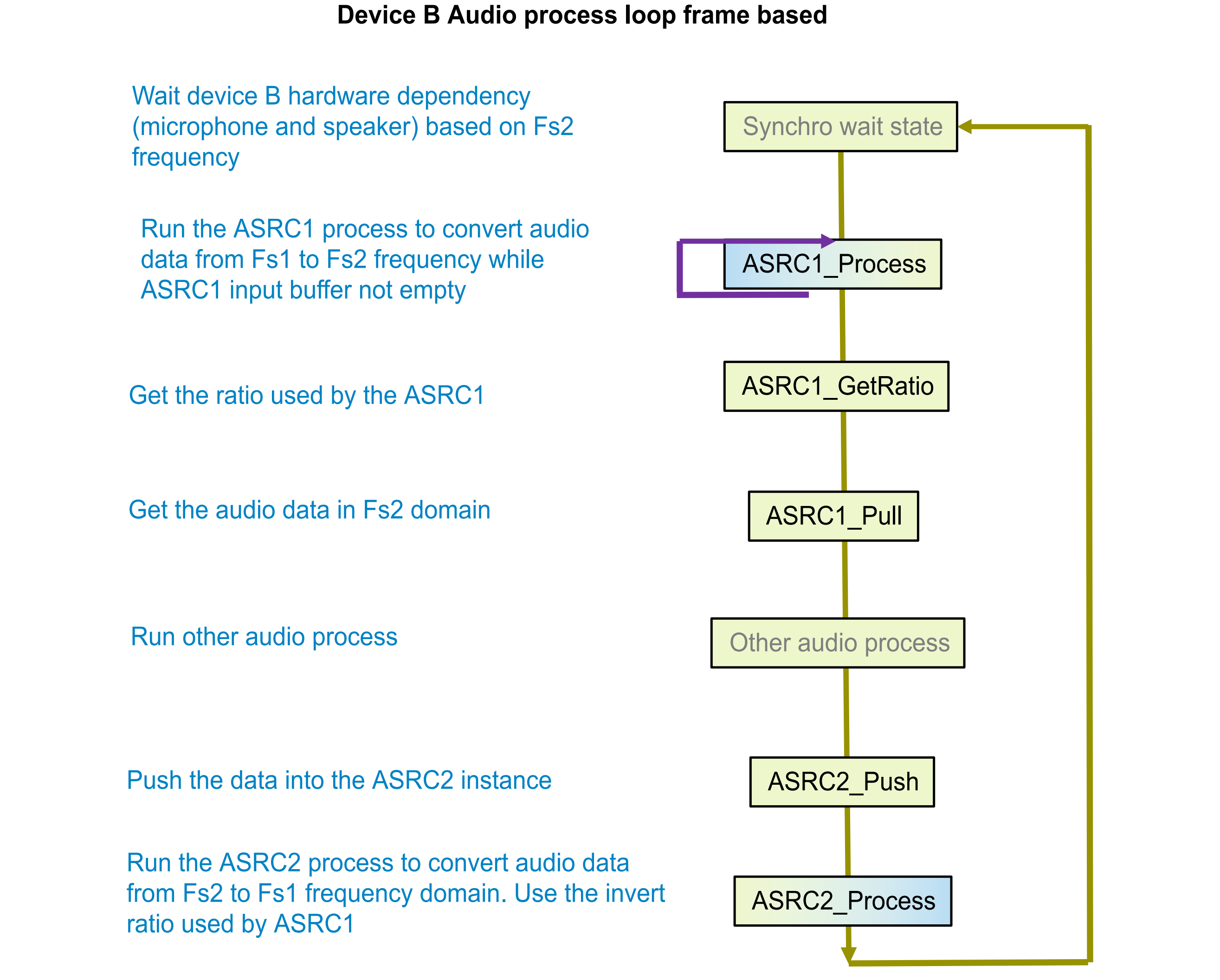

Figure 20. ASRC running loop integration

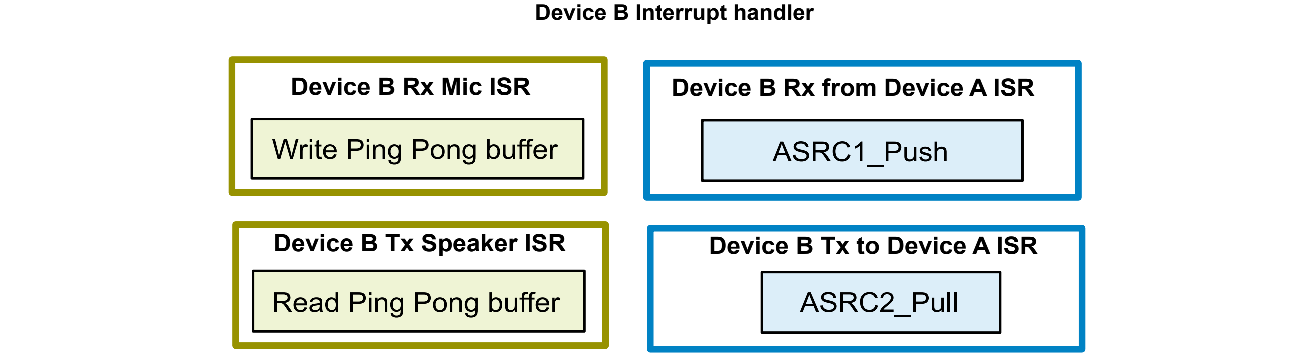

Figure 21. ASRC integration in ISR

4 Performance

This topic describes the Section 4.1 and the Section 4.2.

4.1 Memory consumption

4.1.1 Sample rate 16 kHz

ASRC memory can be repatriated in four different memories:

- SLOW_DATA does not impact MIPS consumption.
- FAST_DATA and FAST_COEFF must be placed in platform fast memory not to impact million instructions per second (MIPS) consumption.
- TEMPORARY_FAST must be placed in platform fast memory not to impact MIPS consumption. This amount of memory can be reused by another algorithm when the ASRC function is not under run. This can also be named scratch memory.

Memory type	Size (Bytes)
PL_PERSISTENT_SLOW_DATA	95
PL_PERSISTENT_FAST_DATA	2.7 K
PL_PERSISTENT_FAST_COEF	16.5 K
PL_TEMPORARY_FAST	15
TOTAL	**19.4 K**

Figure 22. ASRC memory consumption (Fs = 16 kHz)

4.2 MCPS consumption

4.2.1 Sample rate 16 kHz

These metrics are run on an NXP i.MX RT1170 platform. It can be slightly different according to memory access performance.

Function	Performance (MCPS)
ASRC_Pull	0.26
ASRC_Process	12
ASRC_Push	0.2
Total	15 MCPS

Figure 23. ASRC MCPS consumption for i.MX RT1170 - Fs = 16 kHz

5 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2024 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

6 Revision history

Table 1. Revision history

Document ID	Release date	Description
UG10087 v.1.0	10 January 2024	Initial version updated for MCUXpresso SDK v2.15.000.

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.

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.

i.MX — is a trademark of NXP B.V.

Tables

Figures

Contents

Date of release: 10 January 2024
Document identifier: UG10087