UM1995
User manual

Clock drift compensation library software expansion for STM32Cube

Introduction

The clock drift compensation (CDC) library user manual describes the software interface and its requirements. It describes how to integrate the module into a main program, like the Audio STM32Cube expansion software. It also provides a basic understanding of the underlying algorithm.

The CDC library is used to adapt the data rate by smoothly adding or removing one sample on demand. This module is needed and can be used when two asynchronous clocking mechanisms are used in the same SW chain like:

- Audio chain with a slave input (IIS for instance) connected to a master IIS output
- USB streaming synchronization
- One clock locked to a network while the system clock is free running

The CDC library is part of X-CUBE-AUDIO firmware package.

Contents

List of tables

List of figures

1 Module overview

1.1 Algorithm function

The CDC module provides functions to smoothly add or remove one sample as soon as a drift is detected between input and output streams. It is independent from input sampling rate.

1.2 Module configuration

The CDC module supports mono and stereo interleaved 16-bit or 32-bit I/O data, with a minimum input frame size of 282 stereo samples for the High Quality version (HQ) and a minimum of 218 stereo samples for the standard quality version.

Several versions of the module are available depending on the I/O format, the quality level, the Cortex Core and the used tool chain:

- CDC_CM4_IAR.a / CDC_CM4_GCC.a / CDC_CM4_Keil.lib:
  Standard version for low-MIPS and good quality requirements with 16 bits input/output buffers. It runs on any STM32 microcontroller featuring an Arm® core with Cortex®-M4 instruction set.
- CDCHQ_CM4_IAR.a / CDCHQ_CM4_GCC.a / CDCHQ_CM4_Keil.lib:
  High Quality version with 16 bits input/output buffers. It runs on any STM32 microcontroller featuring an Arm® core with Cortex®-M4 instruction set.
- CDC_32b_CM4_IAR.a / CDC_32b_CM4_GCC.a / CDC_32b_CM4_Keil.lib:
  Standard version for low-MIPS and good quality requirements with 32 bits input/output buffers. It runs on any STM32 microcontroller featuring an Arm® core with Cortex®-M4 instruction set.
- CDCHQ_32b_CM4_IAR.a / CDCHQ_32b_CM4_GCC.a / CDCHQ_32b_CM4_Keil.lib:
  High Quality version with 32 bits input/output buffers. It runs on any STM32 microcontroller featuring an Arm® core with Cortex®-M4 instruction set.
- CDC_CM7_IAR.a / CDC_CM7_GCC.a / CDC_CM7_Keil.lib:
  Standard version for low-MIPS and good quality requirements with 16 bits input/output buffers. It runs on any STM32 microcontroller featuring an Arm® core with Cortex®-M7 instruction set.
- CDCHQ_CM7_IAR.a / CDCHQ_CM7_GCC.a / CDCHQ_CM7_Keil.lib:
  High Quality version with 16 bits input/output buffers. It runs on any STM32 microcontroller featuring an Arm® core with Cortex®-M7 instruction set.
- CDC_32b_CM7_IAR.a / CDC_32b_CM7_GCC.a / CDC_32b_CM7_Keil.lib:
  Standard version for low-MIPS and good quality requirements with 32 bits input/output buffers. It runs on any STM32 microcontroller featuring an Arm® core with Cortex®-M7 instruction set.
- CDCHQ_32b_CM7_IAR.a / CDCHQ_32b_CM7_GCC.a / CDCHQ_32b_CM7_Keil.lib:
  High Quality version with 32 bits input/output buffers. It runs on any STM32 microcontroller featuring an Arm® core with Cortex®-M7 instruction set.

1.3 Resource summary

Table 1 contains the module requirements for Flash memory, Stack, RAM and frequency (MHz). All these requirements are independent from input sampling frequency.

Table 1. Resource summary

Version	User Case @ 48 KHz, 10 ms framing	Core	Flash code (.text)	Flash data (.rodata)	Stack	Persistent RAM	Scratch RAM(1)	Peak Frequency (MHz)
Standard	Mono	M4	2266 Bytes	8 Bytes	100 Bytes	12 Bytes	504 Bytes	3.1
		M7	2222 Bytes					1.7
	Stereo	M4	2266 Bytes					6.6
		M7	2222 Bytes					3.9
High quality	Mono	M4	2638 Bytes	8 Bytes	100 Bytes	12 Bytes	672 Bytes	4.7
		M7	2590 Bytes					2.5
	Stereo	M4	2638 Bytes					10.2
		M7	2590 Bytes					5.6
Standard 32-bit I/O	Mono	M4	2392 Bytes	8 Bytes	100 Bytes	12 Bytes	1008 Bytes	3.7
		M7	2348 Bytes					1.9
	Stereo	M4	2392 Bytes					7.2
		M7	2348 Bytes					4.3
High quality 32-bit I/O	Mono	M4	2728 Bytes	8 Bytes	100 Bytes	12 Bytes	1344 Bytes	5.6
		M7	2728 Bytes					2.9
	Stereo	M4	2776 Bytes					12
		M7	2728 Bytes					6.2

1. Scratch RAM is the memory that can be shared with other process running on the same priority level. This memory is not used from one frame to another by CDC routines.

Note: The footprints are measured on board, using IAR Embedded Workbench for ARM v7.40 (IAR Embedded Workbench common components v7.2).

The $Average_{MHz}$ is given by the following formula:

$$Average_{MHz} = \text{peak frequency} \times \text{frame size sec} * \frac{\text{ppm drift} \times \text{sampling freq}}{1000000}$$

For instance, on ST32F4, using High Quality version with a 10ms framing @ 48 kHz and a drift to compensate 300 ppm, we obtain:

- Peak Frequency (during one frame) = 10.2 MHz
- Average Frequency = 1.5 MHz

Maximum theoretical ppm drift (to be compensated) is given by the following formula:

$$\text{ppm drift max} = \frac{1000000}{\text{frame size sec} \times \text{sampling freq}}$$

With the same example as above, we obtain:

- ppm_drift_max = 2083 ppm

2 Module Interfaces

Two files are needed to integrate the CDC module: CDC_xxx_CMy_zzz.a/.lib and the *cdc_glo.h* header file. They contain all definitions and structures to be exported to the framework.

Note: The audio_fw_glo.h file is a generic header file common to all audio modules. It must be included in the audio framework.

2.1 API

Six generic functions have a software interface to the main program:

- cdc_reset
- cdc_setParam
- cdc_getParam
- cdc_setConfig
- cdc_getConfig
- cdc_process

2.1.1 cdc_reset function

This procedure initializes the persistent memory of the CDC module and static and dynamic parameters with default values.

API description:

```
int32_t cdc_reset(void *persistent_mem_ptr, void *scratch_mem_ptr);
```

Table 2. cdc_reset

I/O	Name	Type	Description
Input	persistent_mem_ptr	void *	Pointer to internal persistent memory
Input	scratch_mem_ptr	void *	Pointer to internal scratch memory
Returned value	-	int32_t	Error value

This routine must be called at least once at initialization time, when the real time processing has not started.

2.1.2 cdc_setParam function

This procedure writes module static parameters from the main framework to the module's internal memory. It can be called after the reset routine and before the start of the real time processing. It handles the static parameters, i.e. the parameters with values which cannot be changed during the module processing.

API description:

```
int32_t cdc_setParam(cdc_static_param_t *input_static_param_ptr,
void*persistent_mem_ptr);
```

Table 3. cdc_setParam

I/O	Name	Type	Description
Input	input_static_param_ptr	cdc_static_param_t*	Pointer to static parameters structure
Input	persistent_mem_ptr	void *	Pointer to internal persistent memory
Returned value	-	int32_t	Error value

Note: There is currently no static parameter, so no reason to call this routine in this module version.

2.1.3 cdc_getParam function

This procedure gets the module static parameters from the module internal memory to the main framework. It can be called after the reset routine and before the start of the real time processing. It handles the static parameters, i.e. the parameters with values which cannot be changed during the module processing.

API description:

```
int32_t cdc_getParam(cdc_static_param_t *input_static_param_ptr, void
*persistent_mem_ptr);
```

Table 4. cdc_getParam

I/O	Name	Type	Description
Input	input_static_param_ptr	cdc_static_param_t *	Pointer to static parameters structure
Input	persistent_mem_ptr	void *	Pointer to internal persistent memory
Returned value	-	int32_t	Error value

Note: There is currently no static parameter, so no reason to call this routine in this module version.

2.1.4 cdc_setConfig function

This procedure sets the module dynamic parameters from the main framework to the module internal memory. It can be called at any time during processing (after cdc_reset() routines).

API description:

```
int32_t cdc_setConfig(cdc_dynamic_param_t *input_dynamic_param_ptr, void
*persistent_mem_ptr);
```

Table 5. cdc_setConfig

I/O	Name	Type	Description
Input	input_dynamic_param_ptr	cdc_dynamic_param_t *	Pointer to dynamic parameters structure
Input	persistent_mem_ptr	void *	Pointer to internal persistent memory
Returned value	-	int32_t	Error value

2.1.5 cdc_getConfig function

This procedure gets module dynamic parameters from the internal persistent memory to the main framework. It can be called at any time during processing (after reset and setParam routines).

API description:

```
int32_t cdc_getConfig(cdc_dynamic_param_t *input_dynamic_param_ptr, void
*persistent_mem_ptr);
```

Table 6. cdc_getConfig

I/O	Name	Type	Description
Input	input_dynamic_param_ptr	cdc_dynamic_param_t *	Pointer to dynamic parameters structure
Input	persistent_mem_ptr	void *	Pointer to internal persistent memory
Returned value	-	int32_t	Error value

2.1.6 cdc_process function

This procedure is the module's main processing routine. It should be called at any time, to process each frame.

```
int32_t cdc_process(buffer_t *input_buffer, buffer_t *output_buffer, void
*persistent_mem_ptr);
```

Table 7. cdc_process

I/O	Name	Type	Description
Input	input_buffer	buffer_t *	Pointer to input buffer structure
Output	output_buffer	buffer_t *	Pointer to output buffer structure
Input	persistent_mem_ptr	void *	Pointer to internal static memory
Returned value	-	int32_t	Error value

This process routine cannot run in place; the input_buffer data is modified during processing, thus it cannot be used as it is after any call to the cdc_process() routine.

2.2 External definitions and types

2.2.1 Input and output buffers

The CDC library uses extended I/O buffers which contain, in addition to the samples, some useful information on the stream such as the number of channels, the number of bytes per sample and the interleaving mode.

An I/O buffer structure type, like the one described below, must be used each time, before calling processing routine; otherwise an error will be returned:

```
typedef struct {
    int32_t    nb_channels;
    int32_t    nb_bytes_per_Sample;
```

```
    void        *data_ptr;
    int32_t     buffer_size;
    int32_t     mode;
} buffer_t;
```

Table 8. Input and output buffers

Name	Type	Description
nb_channels	int32_t	Number of channels in data: 1 for mono, 2 for stereo.
nb_bytes_per_Sample	int32_t	Dynamic of data in number of bytes (2 for 16-bits data, 4 for 32 bits data)
data_ptr	void *	Pointer to data buffer (must be allocated by the main framework)
buffer_size	int32_t	Number of samples per channel in the data buffer
mode	int32_t	Buffer mode: 0 = not interleaved, 1 = interleaved.

2.2.2 Returned error values

Table 9 contains the possible returned error values:

Table 9. Returned error values

Definition	Value	Description
CDC_ERROR_NONE	0	OK - no error detected
CDC_IOBUFFERS_TOO_SMALL	-1	Input frame size is too small
CDC_UNSUPPORTED_NUM_CHANNEL	-2	Input data is neither mono nor stereo
CDC_WRONG_NBBYTES_PER_SAMPLES	-3	Input data are neither 16 nor 32-bit values
CDC_INCONSISTENT_BUFFERSIZE_WITH_MODE	-4	Output frame size is not aligned with input frame size and current dynamic used mode. It should be input size +/- 1 sample
CDC_UNSUPPORTED_MODE	-5	Only UPSAMPLING_MODE and DOWNSAMPLING_MODE are supported
CDC_UNSUPPORTED_INPLACE_PROCESSING	-6	Input and output buffers must not be the same
CDC_BAD_HW	-7	The library is not used with the right hardware.

2.3 Static parameters structure

There is no static parameter to be set before calling process routine. The static parameter structure contains a dummy field, for the compatibility with other structures.

```
struct cdc_static_param {
    int32_t empty;
}
typedef struct cdc_static_param cdc_static_param_t;
```

Table 10. Static parameters structure

Name	Type	Description
empty	int32_t	Dummy field - just required to have a non-empty structure

2.4 Dynamic parameters structure

There is one dynamic parameter to be used.

```
struct cdc_dynamic_param {
    uint32_t cdc_mode;
}
typedef struct cdc_dynamic_param cdc_dynamic_param_t;
```

Table 11. Dynamic parameters structure

Name	Type	Description
cdc_mode	uint32_t	This corresponds to drift direction to compensate: #define DOWNSAMPLING_MODE 0 // removes one sample #define UPSAMPLING_MODE 1 // add one sample

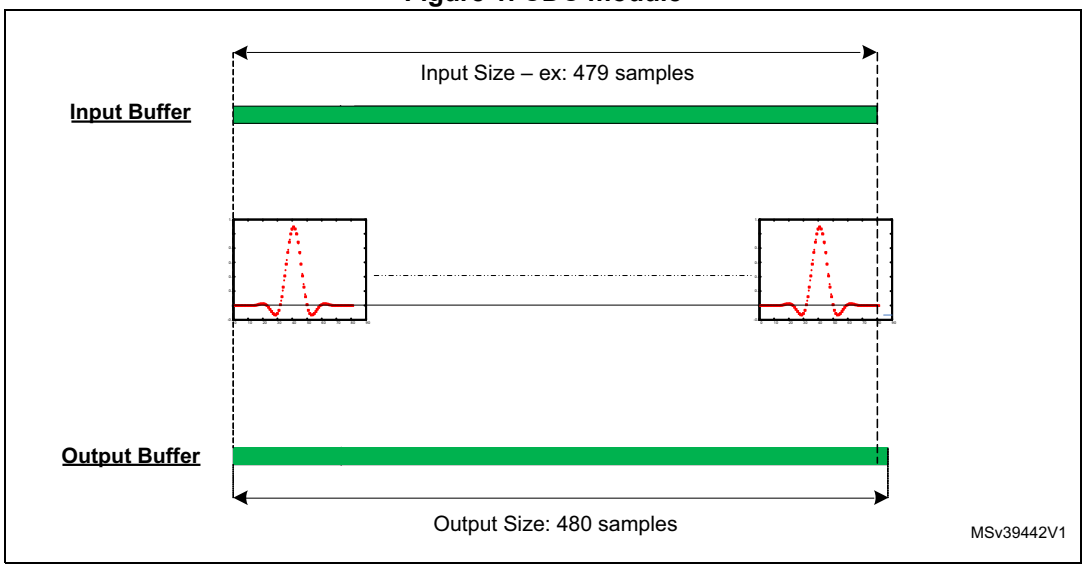
3 Algorithm description

3.1 Processing steps

The CDC module is a module based on re-sampling techniques using two-stage poly-phase filter. This implementation has been MIPS optimized, for Cortex® M4 and M7 cores, by using SIMD instructions set. Audio quality is obtained using fine-tuned ratio and poly-phase filters.

Figure 1 shows an example of drift compensation by smoothly adding one sample to generate a 10ms frame at 48 kHz.

Figure 1. CDC module



3.2 Data formats

The module supports fixed point data, in Q15 or Q31 format, with a mono or stereo interleaved pattern.

3.3 Performance Assessment

There is no objective measurement available for the CDC module; performances are based on subjective assessment, using tones and frequency sweeps.

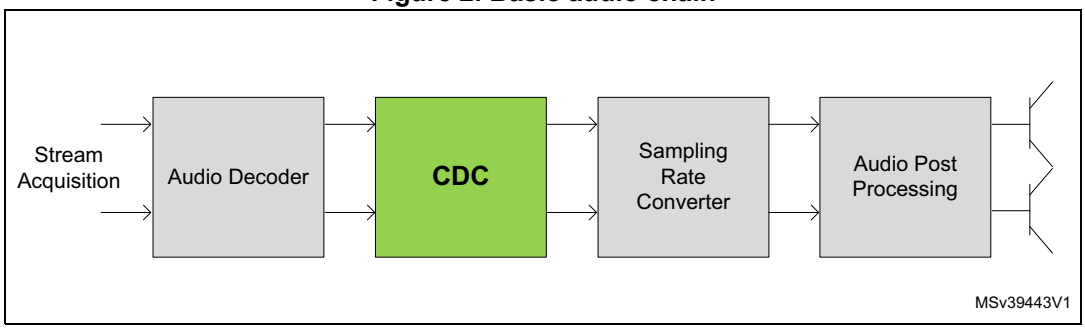
4 System requirements and hardware setup

CDC libraries are built to run either on a Cortex® M4 or on a Cortex® M7 core, without FPU usage. They can be integrated and run on corresponding STM32F4/STM32L4 or STM32F7 family devices. There is no other hardware dependency.

4.1 Recommendations for optimal setup

The clock drift compensation algorithm could be placed in the first part of the audio chain, between the audio decoder and the sampling rate converter for instance. If needed, after this module, streams can be mixed, or post processing can be applied. Samples are then played on the audio DAC. Refer to *Figure 2: Basic audio chain*.

Figure 2. Basic audio chain

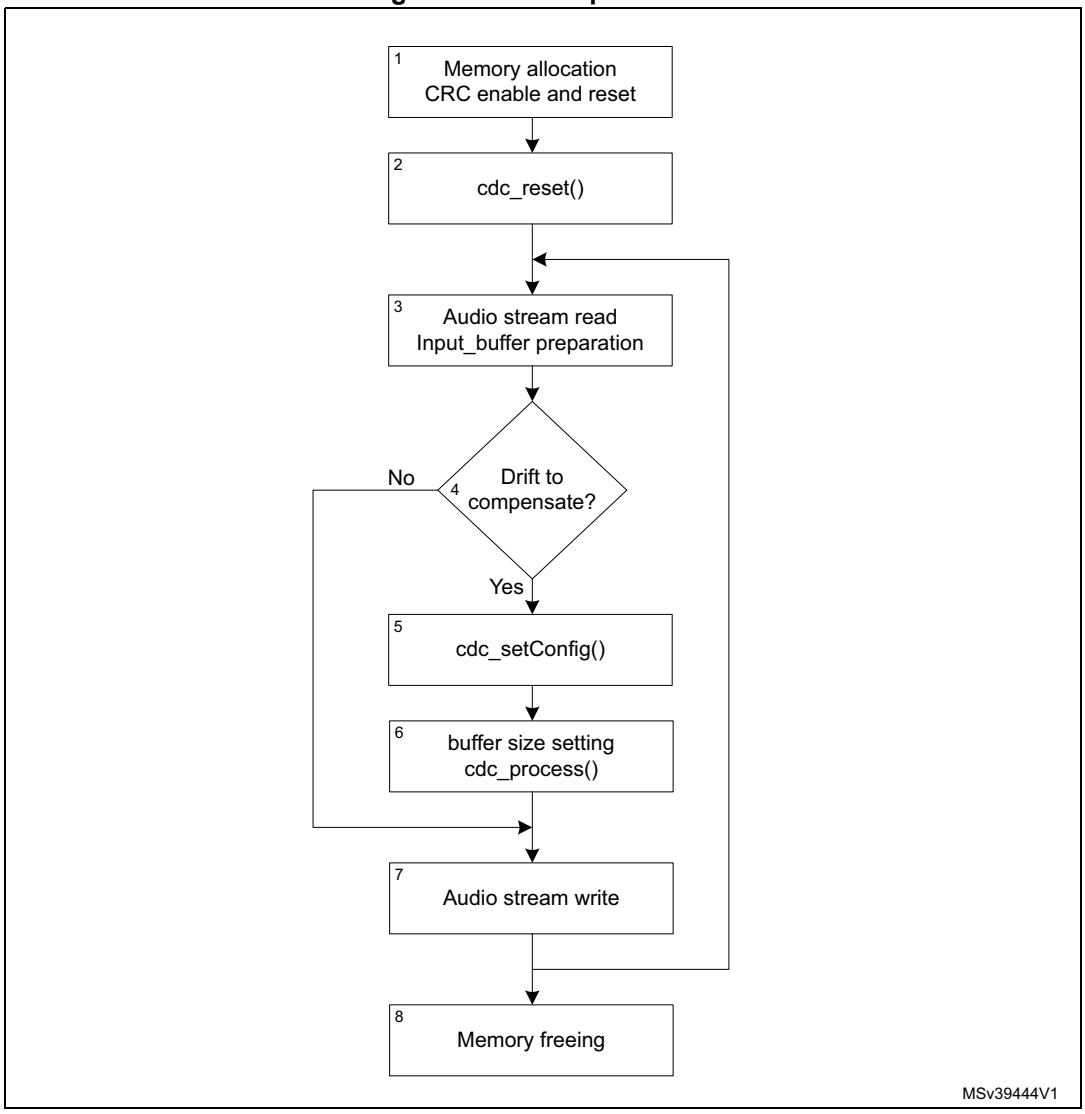


4.1.1 Module integration example

Cube expansion CDC integration examples are provided on STM32746G-Discovery and STM32469I_Discovery boards. Refer to provided integration code for more details.

4.1.2 Module integration summary

Figure 3. API call procedure



1. As explained above, the module persistent and scratch memories have to be allocated, as well as the input and output buffer. Also, CDC library must run on STM32 devices so CRC HW block must be enable and reset.
2. Once the memory has been allocated, the call to cdc_reset() function will initialize the internal variables.
3. The audio stream is read from the proper interface and the input_buffer structure has to be filled in according to the stream characteristics (number of channels, sample rate, interleaving and data pointer). The output buffer structure has to be set as well.
4. At this step the output of a drift detector is caught. Drift direction information is extracted here as well. If some drift compensation is needed, CDC will be called, else CDC processing is bypassed.
5. The dynamic parameters are updated and cdc_setConfig() routine is called to send the dynamic parameters from the audio framework to the module.
6. Depending on drift direction and framework, update input or output buffer size (+/- 1 sample compared to frame size) and call the processing main routine to apply the CDC compensation.
7. The output audio stream can now be written in the proper interface.
8. Once the processing loop is over, the allocated memory has to be freed.

5 How to run and tune the application

CDC library does not manage drift detection but only drift compensation.

The drift detection output is the only dynamic parameter used to add, to remove a sample or to keep the stream as it is. For this reason there is no tuning available for CDC module.

The only available choice is to link the CDC_xxx_CMy_zzz.a/.lib library version associated to cdc_glo.h header file.

In the integration example, samples are get from a file I/O, and one sample every 70 ms is added to simulate a ~ 300 ppm drift at 48 kHz. Signal modification cannot be heard even by playing tones or sweeps.

6 Revision history

Table 12. Document revision history

Date	Revision	Changes
20-Jan-2016	1	Initial release.
21-Mar-2017	2	Updated: – *Section 1.2: Module configuration* – *Table 1: Resource summary* – *Section 2.1: API* – *Section 4.1.1: Module integration example* – *Section 5: How to run and tune the application*
09-Jan-2018	3	Updated: – *Introduction*