MT793X IoT SDK for

Tinypcm Library API Introduction

Version: 0.1

Release date: 2020-10-29

Version History

Version	Date	Author	Description
0.1	2020-10-29	Xuan Xu	Create tinypcm API document

Table of Contents

List of Figures

List of Tables

1 Tinypcm Library API

This chapter introduces some commonly-used tinypcm API that can help you to get started quickly.
The header files and libaudio.a are provided by MediaTek for using the tinypcm API.

1.1 Tinypcm Header File Introduction

- **Header Files**: The header files supported by tinypcm. Table 1-1 lists the header files.

Table 1-1. Tinypcm Header Files.

No.	Header file	Note
1	tinypcm.h	The application using tinypcm must include this header file
2	asound.h	The application using tinypcm must include this header file

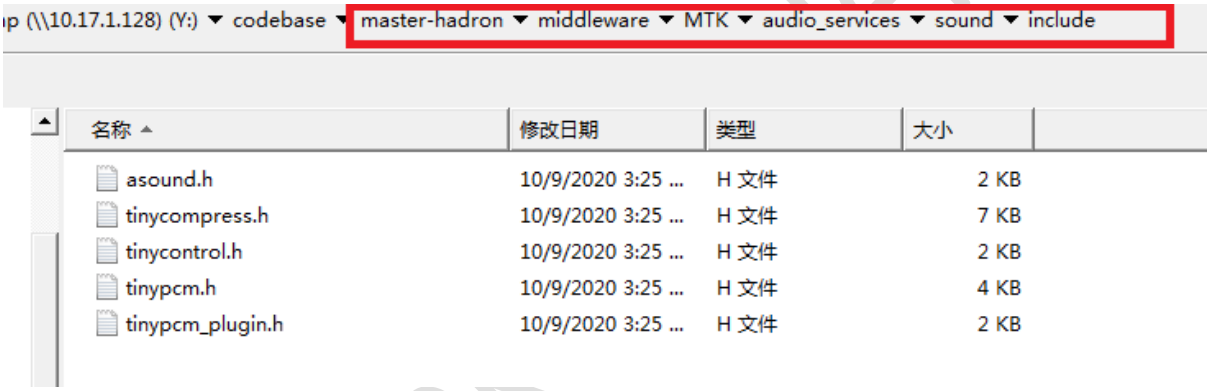

Figure 1-1. Header File Path.

1.2 Tinypcm Library

MediaTek provides the header files and libaudio.a as indicated in Figure 1-2.

- Tinypcm lib is an audio driver framework and used to write/read with audio hardware.
- The purpose of tinypcm lib is to play some applications and write data to alsa node.

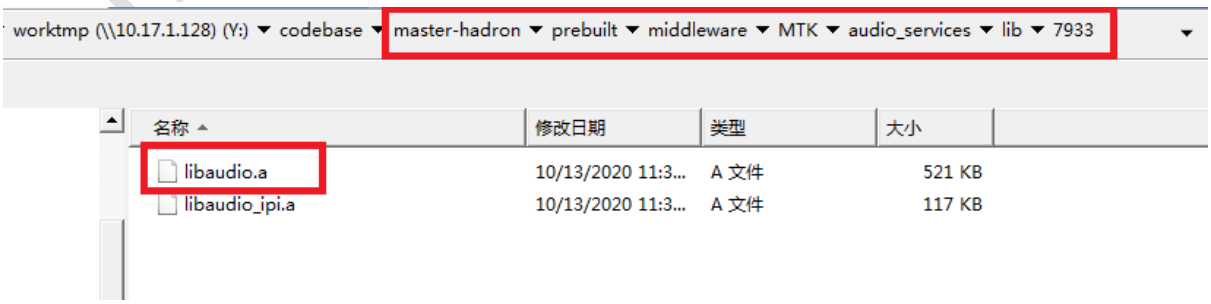

Figure 1-2. Libaudio.a Path.

1.3 Tinypcm API List

This section introduces the functions for tinypcm connection. Note that all functions are from tinypcm. Table 1-2 shows the detailed information.

Table 1-2. Tinypcm API Files.

No.	API	Note
1	snd_pcm_open	open a PCM
2	snd_pcm_writei	write interleaved frames to a PCM
3	snd_pcm_readi	read interleaved frames from a PCM
4	snd_pcm_hw_params	install one PCM hardware configuration
5	snd_pcm_sw_params	install PCM software configuration defined by params
6	snd_pcm_start	start a PCM
7	snd_pcm_prepare	prepare PCM for use
8	snd_pcm_hw_free	remove PCM hardware configuration and free associated resources
9	snd_pcm_drop	stop a PCM dropping pending frames
10	snd_pcm_drain	stop a PCM preserving pending frames
11	snd_pcm_close	close PCM handle
12	snd_pcm_avail	get the available space for playback/capture

1.4 API Detailed Information

This section introduces the tinypcm API interface definition and function in tinypcm.

1.4.1 snd_pcm_open Function

1. API Definition

   - **int snd_pcm_open(sound_t **psnd, const char *name, int stream, int mode);**
     - This API is used to open a PCM.

2. Parameters

   - **psnd:** It is a PCM handler.
     - If the return value is OK, this value is valuable. This API must be used when you call other APIs of the tinypcm.
   - **name:** ASCII identifier of the PCM handle.
   - **stream:** Wanted stream
   - **mode:** Open mode

```
enum msd_open_mode {
    MSD_BLOCK = 0,
    MSD_NONBLOCK,
};
```

3. Return Value

   - **0** Ok
   - **Others** Fail

NOTE: For details about the error types, please refer to the LINUX standard error type definition.

1.4.2 snd_pcm_writei Function

1. API Definition

   - **int snd_pcm_writei(sound_t *snd, void *buf, unsigned int size);**
     - This API is used to write interleaved frames to a PCM.
     - If the blocking behaviour is selected and it is running, then the routine waits until all requested frames are played or put to the playback ring buffer. The returned number of frames can be less only if a signal or underrun occurs.
     - If the non-blocking behaviour is selected, then the routine does not wait at all.

2. Parameters

   - **snd:** the PCM handler obtained from snd_pcm_open.

- **buf:** buffer that contains the frames.
- **size:** frames to be written.

3. Return Value

- **positive** Frames actually written
- **Others** Fail

Note: For details about the error types, please refer to the LINUX error type definition.

1.4.3 snd_pcm_readi Function

1. API Definition

- **int snd_pcm_readi(sound_t *snd, void *buf, unsigned int size)**
  - This API is used to read interleaved frames from a PCM.
  - If the blocking behaviour is selected and it is running, then the routine waits until all requested frames are filled. The returned number of frames can be less only if a signal or underrun occurs.
  - If the non-blocking behaviour is selected, then the routine does not wait at all.

2. Parameters

- **snd:** the PCM handler obtained from snd_pcm_open
- **buf:** buffer that contains the frames
- **size:** frames to be read

3. Return Value

- positive Frames actually read
- Others Fail

Note: For details about the error types, please refer to the LINUX error type definition.

1.4.4 snd_pcm_hw_params Function

1. API Definition

- **int snd_pcm_hw_params(sound_t *snd, struct msd_hw_params *params);**
  - This API is used to install one PCM hardware configuration.
  - The hardware parameters cannot be changed when the stream is running (active).

2. Parameters

- **snd:** the PCM handler obtained from snd_pcm_open
- **params:** space definition configuration container

3. Return Value

- **0** OK
- **Others** Fail

Note: For details about the error types, please refer to the LINUX error type definition.

1.4.5 snd_pcm_sw_params Function

1. API Definition

- **int snd_pcm_sw_params(sound_t *snd, struct msd_sw_params *params)**
  - This API is used to install PCM software configuration defined by params.
  - The software parameters can be changed at any time.

2. Parameters

- **snd:** the PCM handler obtained from snd_pcm_open
- **params:** configuration container.

3. Return Value

- **0** OK
- **Others** Fail

Note: For details about the error types, please refer to the LINUX error type definition.

1.4.6 snd_pcm_start Function

1. API Definition

- **int snd_pcm_start(sound_t *snd);**
  - This API is used to start a PCM.

2. Parameters

- **handle:** the PCM handler obtained from snd_pcm_open

3. Return Value

- **0** OK
- **Others** Fail

Note: For details about the error types, please refer to the LINUX error type definition.

1.4.7 snd_pcm_prepare Function

1. API Define

- **int snd_pcm_prepare(sound_t *snd);**

- This API is used to prepare PCM for use.

2. Parameters
   - **handle:** the PCM handler obtained from snd_pcm_open
3. Return Value
   - **0** OK
   - **Others** Fail

Note: For details about the error types, please refer to the LINUX error type definition.

1.4.8 snd_pcm_hw_free Function

1. API Definition
   - **int snd_pcm_hw_free(sound_t *snd);**
     - This API is used to remove PCM hardware configuration and free associated resources.
2. Parameters
   - **handle:** the PCM handler obtained from snd_pcm_open.
3. Return Value
   - **0** OK
   - **Others** Fail

   Note: For details about the error types, please refer to the LINUX error type definition.

1.4.9 snd_pcm_drop Function

1. API Definition
   - **int snd_pcm_drop(sound_t *snd);**
     - This API is used to stop a PCM dropping pending frames.
     - This function stops the PCM immediately.
     - The pending samples on the buffer are ignored.
     - For processing all pending samples, use ::snd_pcm_drain() instead.
2. Parameters
   - **handle:** the PCM handler obtained from snd_pcm_open
3. Return Value
   - **0** OK
   - **Others** Fail

   Note: For details about the error types, please refer to the LINUX error type definition.

1.4.10 snd_pcm_drain Function

1. API Definition

   - **int snd_pcm_drain(sound_t *snd);**
     - This API is used to stop a PCM preserving pending frames.
     - For playback, wait for all pending frames to be played and then stop the PCM.
     - For capture, stop PCM permitting to retrieve residual frames.
     - For stopping the PCM stream immediately, use ::snd_pcm_drop() instead.

2. Parameters

   - **handle**: the PCM handler obtained from snd_pcm_open.

3. Return Value

   - **0** OK
   - **Others** Fail

   Note: For details about the error types, please refer to the LINUX error type definition.

1.4.11 snd_pcm_close Function

1. API Definition

   - **int snd_pcm_close(sound_t *snd)**;
     - This API is used to close PCM handle.
     - Closes the specified PCM handle and frees all associated resources.

2. Parameters

   - **handle**: the PCM handler obtained from snd_pcm_open.

3. Return Value

   - **0** OK
   - **Others** Fail

1.4.12 snd_pcm_avail Function

1. API Define

   - **int snd_pcm_avail(sound_t *snd);**
     - This API is used to get the available (writable) space for playback and get the available (readable) space for capture.
     - On capture, this API takes all the actions needed to transport all the ready frames across underlying layers to the application level.
     - The position corresponds to the hardware (driver) position in the sound ring buffer in this functions.

2. Parameters

   - **handle:** the PCM handler obtained from snd_pcm_open.

3. Return Value

   - **positive** A positive number of frames that are ready
   - **negative** Fail

2 Sample Code

2.1 Playback Demo

```
#include “asound.h”
#include “tinypcm.h”

sound_t *w_snd;
int ret, index;
struct msd_hw_params params;
params.format = MSD_PCM_FMT_S32_LE;
params.channels = 2;
params.period_count = 4;
params.period_size = 640;
params.rate = 48000;
int bytes_per_frame = 32 * params.channels / 8;
int data_size = bytes_per_frame * params.period_size * 4;

void *data_src = malloc(data_size);
if (!data_src) {
	printf("%s: memory error\n", __FUNCTION__);
	return;
}
memset(data_src, 0, sizeof(data_src));

void *golden_src = afe_sgen_golden_table_32bits;
int golden_size = table_size;
for (index = 0; index < data_size / golden_size; index++) {
	memcpy(data_src + index * golden_size, golden_src, golden_size);
}
aud_msg("data_src = %p, data_size = 0x%x", data_src, data_size);

connect_route("track0", "INTDAC out", 1, CONNECT_FE_BE);
connect_route("I_22", "O_20", 1, CONNECT_IO_PORT);
connect_route("I_23", "O_21", 1, CONNECT_IO_PORT);

ret = snd_pcm_open(&w_snd, "track0", 0, 0);
if (ret)
	goto exit1;
ret = snd_pcm_hw_params(w_snd, &params);
if (ret)
	goto exit2;
ret = snd_pcm_prepare(w_snd);
```

```
if (ret)
	goto exit2;
for (index = 0; index < 100; index ++) {
	ret = snd_pcm_write(w_snd, data_src, data_size / bytes_per_frame);
	if (ret != data_size / bytes_per_frame)
		aud_msg("ret: %d", ret);
}

ret = snd_pcm_drop(w_snd);
if (ret)
	goto exit2;
ret = snd_pcm_hw_free(w_snd);
if (ret)
	goto exit2;

exit2:
snd_pcm_close(w_snd);

exit1:
free(data_src);
```

Exhibit 1 Terms and Conditions

Your access to and use of this document and the information contained herein (collectively this "Document") is subject to your (including the corporation or other legal entity you represent, collectively "You") acceptance of the terms and conditions set forth below ("T&C"). By using, accessing or downloading this Document, You are accepting the T&C and agree to be bound by the T&C. If You don't agree to the T&C, You may not use this Document and shall immediately destroy any copy thereof.

This Document contains information that is confidential and proprietary to MediaTek Inc. and/or its affiliates (collectively "MediaTek") or its licensors and is provided solely for Your internal use with MediaTek's chipset(s) described in this Document and shall not be used for any other purposes (including but not limited to identifying or providing evidence to support any potential patent infringement claim against MediaTek or any of MediaTek's suppliers and/or direct or indirect customers). Unauthorized use or disclosure of the information contained herein is prohibited. You agree to indemnify MediaTek for any loss or damages suffered by MediaTek for Your unauthorized use or disclosure of this Document, in whole or in part.

MediaTek and its licensors retain titles and all ownership rights in and to this Document and no license (express or implied, by estoppels or otherwise) to any intellectual propriety rights is granted hereunder. This Document is subject to change without further notification. MediaTek does not assume any responsibility arising out of or in connection with any use of, or reliance on, this Document, and specifically disclaims any and all liability, including, without limitation, consequential or incidental damages.

THIS DOCUMENT AND ANY OTHER MATERIALS OR TECHNICAL SUPPORT PROVIDED BY MEDIATEK IN CONNECTION WITH THIS DOCUMENT, IF ANY, ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE. MEDIATEK SPECIFICALLY DISCLAIMS ALL WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, COMPLETENESS OR ACCURACY AND ALL WARRANTIES ARISING OUT OF TRADE USAGE OR OUT OF A COURSE OF DEALING OR COURSE OF PERFORMANCE. MEDIATEK SHALL NOT BE RESPONSIBLE FOR ANY MEDIATEK DELIVERABLES MADE TO MEET YOUR SPECIFICATIONS OR TO CONFORM TO A PARTICULAR STANDARD OR OPEN FORUM.

Without limiting the generality of the foregoing, MediaTek makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does MediaTek assume any liability arising out of the application or use of any product, circuit or software. You agree that You are solely responsible for the designing, validating and testing Your product incorporating MediaTek's product and ensure such product meets applicable standards and any safety, security or other requirements.

The above T&C and all acts in connection with the T&C or this Document shall be governed, construed and interpreted in accordance with the laws of Taiwan, without giving effect to the principles of conflicts of law.