XR872 AC107 Codec User Guide

Revision 1.0

Oct 11, 2019

Declaration

THIS DOCUMENTATION IS THE ORIGINAL WORK AND COPYRIGHTED PROPERTY OF XRADIO TECHNOLOGY ("XRADIO"). REPRODUCTION IN WHOLE OR IN PART MUST OBTAIN THE WRITTEN APPROVAL OF XRADIO AND GIVE CLEAR ACKNOWLEDGEMENT TO THE COPYRIGHT OWNER.

THE PURCHASED PRODUCTS, SERVICES AND FEATURES ARE STIPULATED BY THE CONTRACT MADE BETWEEN XRADIO AND THE CUSTOMER. PLEASE READ THE TERMS AND CONDITIONS OF THE CONTRACT AND RELEVANT INSTRUCTIONS CAREFULLY BEFORE USING, AND FOLLOW THE INSTRUCTIONS IN THIS DOCUMENTATION STRICTLY. XRADIO ASSUMES NO RESPONSIBILITY FOR THE CONSEQUENCES OF IMPROPER USE (INCLUDING BUT NOT LIMITED TO OVERVOLTAGE, OVERCLOCK, OR EXCESSIVE TEMPERATURE).

THE INFORMATION FURNISHED BY XRADIO IS PROVIDED JUST AS A REFERENCE OR TYPICAL APPLICATIONS, ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS DOCUMENT DO NOT CONSTITUTE A WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. XRADIO RESERVES THE RIGHT TO MAKE CHANGES IN CIRCUIT DESIGN AND/OR SPECIFICATIONS AT ANY TIME WITHOUT NOTICE.

NOR FOR ANY INFRINGEMENTS OF PATENTS OR OTHER RIGHTS OF THE THIRD PARTIES WHICH MAY RESULT FROM ITS USE. NO LICENSE IS GRANTED BY IMPLICATION OR OTHERWISE UNDER ANY PATENT OR PATENT RIGHTS OF XRADIO.THIRD PARTY LICENCES MAY BE REQUIRED TO IMPLEMENT THE SOLUTION/PRODUCT. CUSTOMERS SHALL BE SOLELY RESPONSIBLE TO OBTAIN ALL APPROPRIATELY REQUIRED THIRD PARTY LICENCES. XRADIO SHALL NOT BE LIABLE FOR ANY LICENCE FEE OR ROYALTY DUE IN RESPECT OF ANY REQUIRED THIRD PARTY LICENCE. XRADIO SHALL HAVE NO WARRANTY, INDEMNITY OR OTHER OBLIGATIONS WITH RESPECT TO MATTERS COVERED UNDER ANY REQUIRED THIRD PARTY LICENCE.

Revision History

Version	Date	Summary of Changes
1.0	2019-10-11	Initial Version

Table 1- 1 Revision History

Contents

Figures

1 宏定义说明

注：凡是使能类型的宏定义，默认 0 为不使能，非 0 为使能（建议非 0 时定义为 1）

1.1 调试宏定义说明

1.1.1 AC107_DBG_EN

驱动 debug 调试打印使能，使能后驱动中所有调用到 AC107_DBG 宏定义的调试信息在 codec 使用过程中会被打印出来，方便调试，如下图 1-1 所示为录音时的调试打印：

```
$
$ audio cap 48000 2 /test.pcm
<ACK> 200 OK
[WRN] switch to high speed error, use DS: 25 MHz
[FS INF] mmc init
[FS INF] mount success
[cmd] CMD:drv audio cap (samplerate)48000 (channel)2 (file)/test.pcm
[AC107_CODEC] --->ac107_dai_set_sysclk
[AC107_CODEC] SYSCLK source select MCLK
[AC107_CODEC] --->ac107_dai_set_fmt
[AC107_CODEC] AC107 set to work as Slave
[AC107_CODEC] AC107 config I2S format
[AC107_CODEC] AC107 config BCLK&LRCK polarity: BCLK_normal,LRCK_normal
[AC107_CODEC] --->ac107_dai_hw_params
[AC107_CODEC] AC107 set BCLK_DIV_[4]
[AC107_CODEC] Sample rate-[48000], channel numbers-[2], sample resolution-[16]
[AC107_CODEC] --->ac107_dai_set_volume
[AC107_CODEC] AMIC set volume Gain-[0]
[AC107_CODEC] Route(cap): amic Enable
[AC107_CODEC] --->ac107_codec_open
[cmd] Capture run.
$ audio end
[cmd] audio task end
<ACK> 200 OK
[AC107_CODEC] --->ac107_codec_close
[AC107_CODEC] --->ac107_dai_hw_free
[AC107_CODEC] AC107 reset all register to their default value

[cmd] Capture end.
$
```

图 1-1 Xradio AC107 Codec 录音调试信息打印

如上图所示，其中带有“[AC107_CODEC]”前缀的打印为 XR872 AC107 Codec 驱动的调试信息打印。建议调

试过程中使能此宏，调试完成后可关闭。

配置建议：此宏可根据调试需要来打开或者关闭；默认关闭；

1.1.2 AC107_ERR_EN

驱动 error 调试打印使能，使能后驱动中所有调用到 AC107_ERR 宏定义的调试信息在 codec 使用过程中会被打印出来，方便驱动出错时调试查找问题。

另外，驱动中还有一个 AC107_ALWAYS 宏定义，使用此宏定义的调试信息始终会打印出来，不受使能控制，如配置音量、路径时的打印。

配置建议：此宏可根据调试需要来打开或者关闭；默认打开；

1.2 功能配置宏定义说明

1.2.1 AC107_DEFAULT_LRCK_PERIOD

AC107 默认 LRCK_PERIOD 宏定义，用于 AC107 Codec 注册时初始化私有数据结构体的 lrck_period 变量，此变量在后续 Codec 使用过程中还可以通过 Ioctl 命令配置。此变量的值最终会配置到 AC107 的 0x32 和 0x33 寄存器中，具体含义请查阅 AC107 User Manual 关于此寄存器的说明。

配置建议：采用默认值即可，后续需要修改可通过 Ioctl 命令配置；默认值为 32；

1.2.2 AC107_DEFAULT_RECORD_GAIN

默认录音增益，声卡注册时会初始化录音音量为 VOLUME_INVALID，此时如果应用层未设置过录音音量，则录音时会使用此处定义的默认录音音量，应用层设置过录音音量后则会使用应用设置的录音音量。

配置建议：采用默认值即可；默认值为 VOLUME_GAIN_0dB；

1.2.3 AC107_MALLOC

AC107 Codec 驱动中申请内存宏定义接口。

配置建议：采用默认值即可；默认值为 HAL_Malloc；

1.2.4 AC107_FREE

AC107 Codec 驱动中释放内存宏定义接口。

配置建议：采用默认值即可；默认值为 HAL_Free；

1.2.5 AC107_MEMCPY

AC107 Codec 驱动中 memcpy 宏定义接口。

配置建议：采用默认值即可；默认值为 HAL_Memcpy；

1.2.6 AC107_MEMSET

AC107 Codec 驱动中 memset 宏定义接口。

配置建议：采用默认值即可；默认值为 HAL_Memset；

1.2.7 AC107_I2C_READ

AC107 Codec 驱动中 I2C 读取宏定义接口；

配置建议：采用默认值即可；默认值为 HAL_I2C_Master_Receive_Mem_IT；

1.2.8 AC107_I2C_WRITE

AC107 Codec 驱动中 I2C 写入宏定义接口；

配置建议：采用默认值即可；默认值为 HAL_I2C_Master_Transmit_Mem_IT；

2 基本接口

2.1 ac107_codec_priv 结构体

struct ac107_codec_priv 结构体为 AC107 Codec 驱动中使用的私有数据结构，其对应定义的全局指针变量为 ac107_priv，在 Codec 注册时会为其申请内存，具体定义如下代码段所示：

```
//AC107 codec priv struct
struct ac107_i2c_config {
    I2C_ID i2c_id;
    uint8_t i2c_addr;
};

struct ac107_codec_priv {
    //hw config
    bool pdm_en;
    bool encoding_en;
    bool encoding_fmt;
    uint8_t  encoding_nums;
    uint8_t  adc_pattern;
    uint16_t lrck_period;

    //I2C config
    uint8_t chip_nums;
    struct ac107_i2c_config *ac107_i2c_cfg;
};

struct ac107_codec_priv *ac107_priv;
```

2.1.1 硬件配置变量

- **pdm_en**：PDM 接口使能控制，可通过 Ioctl 命令配置；
- **encoding_en**：I2S Encoding 模式使能控制，可通过 Ioctl 命令配置；
- **encoding_fmt**：I2S Encoding 开始通道编号格式配置，可通过 Ioctl 命令配置；
- **encoding_nums**：I2S Encoding 模式实际录音通道数配置，可通过 Ioctl 命令配置；
- **adc_pattern**：ADC 发送数据源选择，可通过 Ioctl 命令配置；
- **lrck_period**：I2S 接口 LRCK PERIOD 配置，可通过 Ioctl 命令配置；

2.1.2 I2C 配置变量

- **chip_nums**：自动检测到硬件外挂 AC107 芯片个数；
- **ac107_i2c_cfg**：AC107 I2C 配置 struct ac107_i2c_config 结构体类型指针，在 Codec 注册时会为其申请内存并根据自动检测到的 I2C 总线号和 I2C 地址对其进行初始化；struct ac107_i2c_config 结构体包含两个

变量：i2c_id 和 i2c_addr，分别对应 I2C 总线号和 AC107 的 I2C 地址；

2.2 常量数组

2.2.1 ac107_sample_rate[]

AC107 Codec 不同采样率与具体的寄存器配置值映射关系数组，数组元素为 struct real_val_to_reg_val 结构体常量；如下代码段所示：

```
//const array define
struct real_val_to_reg_val {
    uint32_t real_val;
    uint32_t reg_val;
};
static const struct real_val_to_reg_val ac107_sample_rate[] = {
    {8000,  0},
    {11025, 1},
    {12000, 2},
    {16000, 3},
    {22050, 4},
    {24000, 5},
    {32000, 6},
    {44100, 7},
    {48000, 8},
};
```

struct real_val_to_reg_val 结构体各变量定义如下：

- real_val：实际值，此处为实际要配置的采样率；
- reg_val：寄存器值，此处为对应采样率的寄存器值；

2.2.2 ac107_bclk_div[]

AC107 Codec 不同 BCLK 分频比与寄存器配置值映射关系数组，数组元素为 struct real_val_to_reg_val 结构体常量；如下代码段所示：

```
static const struct real_val_to_reg_val ac107_bclk_div[] = {
    {0,  0},
    {1,  1},
    {2,  2},
    {4,  3},
    {6,  4},
    {8,  5},
    {12, 6},
    {16, 7},
    {24, 8},
    {32, 9},
    {48, 10},
    {64, 11},
    {96, 12},
    {128,13},
```

```
    {176,14},
    {192,15},
};
```

2.2.3 ac107_pga_gain[]

AC107 Codec 不同 PGA 增益与寄存器配置值映射关系数组，数组元素为 struct real_val_to_reg_val 结构体常量；如下代码段所示：

```
static const struct real_val_to_reg_val ac107_pga_gain[] = {
    {VOLUME_GAIN_MINUS_6dB, 0},
    {VOLUME_GAIN_0dB,       1},
    {VOLUME_GAIN_3dB,       4},
    {VOLUME_GAIN_4dB,       5},
    {VOLUME_GAIN_5dB,       6},
    {VOLUME_GAIN_6dB,       7},
    {VOLUME_GAIN_7dB,       8},
    {VOLUME_GAIN_8dB,       9},
    {VOLUME_GAIN_9dB,       10},
    {VOLUME_GAIN_10dB,      11},
    {VOLUME_GAIN_11dB,      12},
    {VOLUME_GAIN_12dB,      13},
    {VOLUME_GAIN_13dB,      14},
    {VOLUME_GAIN_14dB,      15},
    {VOLUME_GAIN_15dB,      16},
    {VOLUME_GAIN_16dB,      17},
    {VOLUME_GAIN_17dB,      18},
    {VOLUME_GAIN_18dB,      19},
    {VOLUME_GAIN_19dB,      20},
    {VOLUME_GAIN_20dB,      21},
    {VOLUME_GAIN_21dB,      22},
    {VOLUME_GAIN_22dB,      23},
    {VOLUME_GAIN_23dB,      24},
    {VOLUME_GAIN_24dB,      25},
    {VOLUME_GAIN_25dB,      26},
    {VOLUME_GAIN_26dB,      27},
    {VOLUME_GAIN_27dB,      28},
    {VOLUME_GAIN_28dB,      29},
    {VOLUME_GAIN_29dB,      30},
    {VOLUME_GAIN_30dB,      31},
};
```

2.2.4 ac107_pll_div[]

AC107 Codec 不同 PLL 输入和输出频率下 PLL 参数配置数组，数组元素为 struct pll_div 结构体常量，如下代码段所示：

```
struct pll_div {
    uint32_t freq_in;
    uint32_t freq_out;
    uint32_t m1;
    uint32_t m2;
    uint32_t n;
```

```
    uint32_t k1;
    uint32_t k2;
};

static const struct pll_div ac107_pll_div[] = {
    {400000,   AUDIO_CLK_12M, 0,  0, 983,  15, 1},  //<out: 12.2875M>
    {512000,   AUDIO_CLK_12M, 0,  0, 960,  19, 1},  //24576000/48
    {768000,   AUDIO_CLK_12M, 0,  0, 640,  19, 1},  //24576000/32
    {800000,   AUDIO_CLK_12M, 0,  0, 768,  24, 1},
    {1024000,  AUDIO_CLK_12M, 0,  0, 480,  19, 1},  //24576000/24
    {1600000,  AUDIO_CLK_12M, 0,  0, 384,  24, 1},
    {2048000,  AUDIO_CLK_12M, 0,  0, 240,  19, 1},  //24576000/12
    {3072000,  AUDIO_CLK_12M, 0,  0, 160,  19, 1},  //24576000/8
    {4096000,  AUDIO_CLK_12M, 0,  0, 120,  19, 1},  //24576000/6
    {6000000,  AUDIO_CLK_12M, 4,  0, 512,  24, 1},
    {6144000,  AUDIO_CLK_12M, 1,  0, 160,  19, 1},  //24576000/4
    {12000000, AUDIO_CLK_12M, 9,  0, 512,  24, 1},
    {13000000, AUDIO_CLK_12M, 12, 0, 639,  25, 1}, //<out: 12.2885M>
    {15360000, AUDIO_CLK_12M, 9,  0, 320,  19, 1},
    {16000000, AUDIO_CLK_12M, 9,  0, 384,  24, 1},
    {19200000, AUDIO_CLK_12M, 11, 0, 384,  24, 1},
    {19680000, AUDIO_CLK_12M, 15, 1, 999,  24, 1}, //<out: 12.2877M>
    {24000000, AUDIO_CLK_12M, 9,  0, 256,  24, 1},

    {400000,   AUDIO_CLK_11M, 0,  0, 1016, 17, 1},  //<out: 11.2889M>
    {512000,   AUDIO_CLK_11M, 0,  0, 882,  19, 1},
    {768000,   AUDIO_CLK_11M, 0,  0, 588,  19, 1},
    {800000,   AUDIO_CLK_11M, 0,  0, 508,  17, 1},  //<out: 11.2889M>
    {1024000,  AUDIO_CLK_11M, 0,  0, 441,  19, 1},
    {1600000,  AUDIO_CLK_11M, 0,  0, 254,  17, 1},  //<out: 11.2889M>
    {2048000,  AUDIO_CLK_11M, 1,  0, 441,  19, 1},
    {3072000,  AUDIO_CLK_11M, 0,  0, 147,  19, 1},
    {4096000,  AUDIO_CLK_11M, 3,  0, 441,  19, 1},
    {6000000,  AUDIO_CLK_11M, 1,  0, 143,  18, 1},  //<out: 11.2895M>
    {6144000,  AUDIO_CLK_11M, 1,  0, 147,  19, 1},
    {12000000, AUDIO_CLK_11M, 3,  0, 143,  18, 1},  //<out: 11.2895M>
    {13000000, AUDIO_CLK_11M, 12, 0, 429,  18, 1},  //<out: 11.2895M>
    {15360000, AUDIO_CLK_11M, 14, 0, 441,  19, 1},
    {16000000, AUDIO_CLK_11M, 24, 0, 882,  24, 1},
    {19200000, AUDIO_CLK_11M, 4,  0, 147,  24, 1},
    {19680000, AUDIO_CLK_11M, 13, 1, 771,  23, 1}, //<out: 11.28964M>
    {24000000, AUDIO_CLK_11M, 24, 0, 588,  24, 1},


    {12288000, AUDIO_CLK_12M, 9,  0, 400,  19, 1},     //24576000/2
    {11289600, AUDIO_CLK_11M, 9,  0, 400,  19, 1},     //22579200/2

    {24576000/1,   AUDIO_CLK_12M, 9, 0, 200, 19, 1},   //24576000
    {24576000/16,  AUDIO_CLK_12M, 0, 0, 320, 19, 1},   //1536000
    {24576000/64,  AUDIO_CLK_12M, 0, 0, 640, 9,  1},   //384000
    {24576000/96,  AUDIO_CLK_12M, 0, 0, 960, 9,  1},   //256000
    {24576000/128, AUDIO_CLK_12M, 0, 0, 512, 3,  1},   //192000
    {24576000/176, AUDIO_CLK_12M, 0, 0, 880, 4,  1},   //140000
    {24576000/192, AUDIO_CLK_12M, 0, 0, 960, 4,  1},   //128000

    {22579200/1,   AUDIO_CLK_11M, 9, 0, 200, 19, 1},   //22579200
    {22579200/4,   AUDIO_CLK_11M, 4, 0, 400, 19, 1},   //5644800
```

```
    {22579200/16,  AUDIO_CLK_11M, 0, 0, 320, 19, 1},   //1411200
    {22579200/64,  AUDIO_CLK_11M, 0, 0, 640, 9,  1},   //352800
    {22579200/96,  AUDIO_CLK_11M, 0, 0, 960, 9,  1},   //235200
    {22579200/128, AUDIO_CLK_11M, 0, 0, 512, 3,  1},   //176400
    {22579200/176, AUDIO_CLK_11M, 0, 0, 880, 4,  1},   //128290
    {22579200/192, AUDIO_CLK_11M, 0, 0, 960, 4,  1},   //117600

    {22579200/6,   AUDIO_CLK_11M, 2, 0, 360, 19, 1},   //3763200
    {22579200/8,   AUDIO_CLK_11M, 0, 0, 160, 19, 1},   //2822400
    {22579200/12,  AUDIO_CLK_11M, 0, 0, 240, 19, 1},   //1881600
    {22579200/24,  AUDIO_CLK_11M, 0, 0, 480, 19, 1},   //940800
    {22579200/32,  AUDIO_CLK_11M, 0, 0, 640, 19, 1},   //705600
    {22579200/48,  AUDIO_CLK_11M, 0, 0, 960, 19, 1},   //470400
}
```

struct pll_div 结构体各变量定义如下：

- freq_in：PLL 输入频率；
- freq_out：PLL 输出频率，对 AC107 为 12.288M 或 11.2896M；
- m1,m2,n,k1,k2：PLL 倍频配置参数，PLL 输入与输出频率计算公式如下所示：

FOUT =(FIN*N)/[(M1+1)*(M2+1)*(K1+1)*(K2+1)] ;

2.3 基本读写寄存器接口

2.3.1 ac107_read

- **原型：**static int ac107_read(uint8_t reg, uint8_t *rt_value, struct ac107_i2c_config *i2c_cfg)；
- **传参：**reg：寄存器地址；rt_value：读取返回值变量指针；i2c_cfg：struct ac107_i2c_config 结构体指针；
- **功能：**读取 codec 寄存器值；
- **返回值：**1：读取成功；0：读取失败

2.3.2 ac107_write

- **原型：**static int ac107_write(uint8_t reg, uint8_t value, struct ac107_i2c_config *i2c_cfg)；
- **传参：**reg：寄存器地址；value：要写入的寄存器值；i2c_cfg：struct ac107_i2c_config 结构体指针；
- **功能：**写入 codec 寄存器值；
- **返回值：**1：写入成功；0：写入失败；

2.3.3 ac107_update_bits

- **原型：**static int ac107_update_bits(uint8_t reg, uint8_t mask, uint8_t value, struct ac107_i2c_config *i2c_cfg)

- **传参：**reg：寄存器地址；mask：需要更新的 bit mask；value：需要更新的 bit mask 对应值；i2c_cfg：struct ac107_i2c_config 结构体指针；
- **功能：**更新 codec 寄存器的某几 bit；
- **返回值：**0：更新成功；

2.3.4 ac107_multi_chips_read

- **原型：**static int ac107_multi_chips_read(uint8_t reg, uint8_t *rt_value)；
- **传参：**reg：寄存器地址；rt_value：读取返回值数组指针；
- **功能：**读取多个 codec 同一个寄存器的值；codec 个数由 ac107_priv->chip_nums 决定；
- **返回值：**0：读取成功；

2.3.5 ac107_multi_chips_write

- **原型：**static int ac107_multi_chips_write(uint8_t reg, uint8_t value)；
- **传参：**reg：寄存器地址；value：要写入的寄存器值；
- **功能：**往多个 codec 同一个寄存器写入相同值；codec 个数由 ac107_priv->chip_nums 决定；
- **返回值：**0：写入成功；

2.3.6 ac107_multi_chips_update_bits

- **原型：**static int ac107_multi_chips_update_bits(uint8_t reg, uint8_t mask, uint8_t value)；
- **传参：**reg：寄存器地址；mask：需要更新的 bit mask；value：需要更新的 bit mask 对应值；
- **功能：**更新多个 codec 的同一个寄存器的相同几 bit 的值；codec 个数由 ac107_priv->chip_nums 决定；
- **返回值：**0：更新成功；

2.4 基本硬件配置接口

2.4.1 ac107_hw_init

- ✓ **原型：**static void ac107_hw_init(struct ac107_i2c_config *i2c_cfg)；
- ✓ **传参：**i2c_cfg：struct ac107_i2c_config 结构体指针；
- ✓ **功能：**AC107 Codec 基本硬件初始化；包括系统时钟使能、Module 复位和时钟 gating 释放等；
- ✓ **返回值：**无；

2.4.2 ac107_set_pll

- ✓ **原型：**static int ac107_set_pll(Codec_Pllclk_Src pllclk_src, uint32_t freq_in, uint32_t freq_out)
- ✓ **传参：**pllclk_src：PLL 输入时钟源；freq_in：PLL 输入频率；freq_out：PLL 输出频率；
- ✓ **功能：**配置 PLL 倍频参数、使能 PLL 模块输出等；
- ✓ **返回值：**HAL_OK：配置成功；other：配置失败；

2.5 基本通路配置接口

2.5.1 ac107_set_amic

- ✧ **原型：**static void ac107_set_amic(bool enable)；
- ✧ **传参：**enable：AC107 Codec 2 个 AMIC 通路 enable/disable 配置参数；
- ✧ **功能：**enable/disable AC107 2 个 AMIC 通路；
- ✧ **返回值：**无；

2.5.2 ac107_set_dmic

- ✧ **原型：**static void ac107_set_dmic(bool enable)
- ✧ **传参：**enable：DMIC enable/disable 配置参数；
- ✧ **功能：**enable/disable DMIC 通路；
- ✧ **返回值：**无；

3 codec_dai_ops 接口

AC107 Codec 的 codec_dai_ops 接口封装如下代码段所示：

```
/*** codec dai ops ****/
static const struct codec_dai_ops ac107_codec_dai_ops = {
    .set_sysclk = ac107_dai_set_sysclk,
    .set_fmt    = ac107_dai_set_fmt,
    .set_volume = ac107_dai_set_volume,
    .set_route  = ac107_dai_set_route,
    .hw_params  = ac107_dai_hw_params,
    .hw_free    = ac107_dai_hw_free,
};
```

3.1 ac107_dai_set_sysclk

- **原型：** static int ac107_dai_set_sysclk(Codec_Sysclk_Src sysclk_src, Codec_Pllclk_Src pllclk_src, uint32_t pll_freq_in, uint32_t sample_rate)；
- **传参：**sysclk_src：系统时钟源选择；pllclk_src：PLL 时钟源选择；pll_freq_in：PLL 输入频率；sample_rate：采样率；
- **功能：**配置 AC107 Codec 系统时钟；
- **返回值：**HAL_OK：配置成功；other：配置失败；

3.2 ac107_dai_set_fmt

- **原型：**static int ac107_dai_set_fmt(uint32_t fmt)；
- **传参：**fmt：包括主从角色配置、I2S/LJ/RJ/PCM 格式配置、BCLK/LRCK 极性配置，fmt 为 bit mask 组合配置；
- **功能：**配置 AC107 Codec I2S 接口通信格式；
- **返回值：**HAL_OK：配置成功；other：配置失败；

3.3 ac107_dai_set_volume

- **原型：**static int ac107_dai_set_volume(Audio_Device device, uint16_t volume)；
- **传参：**device：AUDIO_Device 类型音频设备；volume：配置音量；
- **功能：**配置 AC107 Codec 对应通道的音量增益；
- **返回值：**HAL_OK：配置成功；other：配置失败；

3.4 ac107_dai_set_route

- **原型：**static int ac107_dai_set_route(Audio_Device device, Audio_Dev_State state)；
- **传参：**device：AUDIO_Device 类型音频设备；state：Audio_Dev_State 类型设备状态；
- **功能：**打开/关闭 AC107 Codec 对应通道；
- **返回值：**HAL_OK：配置成功；other：配置失败；

3.5 ac107_dai_hw_params

- **原型：**static int ac107_dai_hw_params(Audio_Stream_Dir dir, struct pcm_config *pcm_cfg)；
- **传参：**dir：Audio_Stream_Dir 类型音频流方向；pcm_cfg：struct pcm_config 结构体类型指针；
- **功能：**配置 AC107 Codec 的采样率、通道数、采样精度、I2S 接口 slot 宽度以及调用 ac107_hw_init 接口进行 common 硬件初始化配置等；
- **返回值：**HAL_OK：配置成功；other：配置失败；

3.6 ac107_dai_hw_free

- **原型：**static int ac107_dai_hw_free(Audio_Stream_Dir dir)；
- **传参：**dir：Audio_Stream_Dir 类型音频流方向；
- **功能：**停止录音时对 AC107 Codec 进行软复位操作，所有寄存器恢复上电默认值状态；
- **返回值：**HAL_OK：配置成功；other：配置失败；

4 codec_ops 接口

AC107 Codec 的 codec_ops 接口封装如下代码段所示：

```
/*** codec ops ****/
static const struct codec_ops ac107_codec_ops = {
    .open  = ac107_codec_open,
    .close = ac107_codec_close,

    .reg_read  = ac107_codec_reg_read,
    .reg_write = ac107_codec_reg_write,

    .ioctl = ac107_codec_ioctl,
};
```

4.1 ac107_codec_open

- **原型：**static int ac107_codec_open(Audio_Stream_Dir dir)；
- **传参：**dir：Audio_Stream_Dir 类型音频流方向；
- **功能：**Codec open 时进行相关配置操作；未使用，此接口实现为空；
- **返回值：**HAL_OK：配置成功；other：配置失败；

4.2 ac107_codec_close

- **原型：**static int ac107_codec_close(Audio_Stream_Dir dir)；
- **传参：**dir：Audio_Stream_Dir 类型音频流方向；
- **功能：**Codec close 时进行相关关闭操作；未使用，此接口实现为空；
- **返回值：**HAL_OK：配置成功；other：配置失败；

4.3 ac107_codec_reg_read

- **原型：**static int ac107_codec_reg_read(uint32_t reg)
- **传参：**reg：寄存器地址；
- **功能：**读取自动检测时检测到的第一颗 AC107 Codec 寄存器的值；
- **返回值：**大于等于 0：读取成功，返回读取到的寄存器值；小于 0：读取失败；

4.4 ac107_codec_reg_write

- **原型：**static int ac107_codec_reg_write(uint32_t reg, uint32_t val)；
- **传参：**reg：寄存器地址；value：要写入的寄存器值；
- **功能：**往多个 codec 同一个寄存器写入相同值；codec 个数由 ac107_priv->chip_nums 决定；
- **返回值：**HAL_OK：写入成功；other：写入失败；

4.5 ac107_codec_ioctl

- **原型：**static int ac107_codec_ioctl(uint32_t cmd, uint32_t cmd_param[], uint32_t cmd_param_len)；
- **传参：**cmd：Codec_Ioctl_Cmd 类型命令；cmd_param[]：命令参数数组指针；cmd_param_len：命令参数数组长度；
- **功能：**对 AC107 Codec 进行相关 ioctl 命令操作，如配置硬件参数；
- **返回值：**HAL_INVALID：无效 ioctl 命令；other：相应命令的执行结果；

5 codec_driver 接口

AC107 Codec 的 codec_driver 接口封装如下代码段所示：

```
/*** codec driver ****/
static struct codec_driver ac107_codec_drv = {
    .name = AC107_CODEC_NAME,
    .codec_attr = XRADIO_CODEC_AC107,

    .init   = ac107_codec_init,
    .deinit = ac107_codec_deinit,

    .dai_ops   = &ac107_codec_dai_ops,
    .codec_ops = &ac107_codec_ops,
};
```

5.1 ac107_codec_init

- **原型：**static int ac107_codec_init(void)；
- **传参：**无；
- **功能：**AC107 Codec 模块全局硬件初始化；未使用，此接口实现为空；
- **返回值：**HAL_OK：配置成功；

5.2 ac107_codec_deinit

- **原型：**static void ac107_codec_deinit(void)；
- **传参：**无；
- **功能：**AC107 Codec 模块全局硬件反初始化；未使用，此接口实现为空；
- **返回值：**无；

6 注册接口

6.1 ac107_codec_register

- **原型：**HAL_Status ac107_codec_register(void)；
- **传参：**无
- **功能：**AC107 Codec 注册接口，包括 AC107 Codec 自动检测、申请私有数据内存空间、插入 Codec 链表等；
- **返回值：**HAL_OK：注册成功；HAL_ERROR：注册失败；

此接口不会在外部直接调用，会先在 HAL_SndCard 层封装后再统一给到外部调用，封装后的接口如下代码段所示：

```
HAL_Status HAL_SndCard_CodecRegisterAc107(void)
{
    return ac107_codec_register();
}
```

接口调用如下代码段所示：

```
__weak HAL_Status board_soundcard_init(void)
{
    ......

#if PRJCONF_AC107_SOUNDCARD_EN
    HAL_SndCard_CodecRegisterAc107();
#endif

    ......

    return HAL_OK;
}
```

6.2 ac107_codec_unregister

- **原型：**HAL_Status ac107_codec_unregister(void)；
- **传参：**无；
- **功能：**AC107 Codec 反注册接口，包括删除 Codec 链表项、释放私有数据内存空间等；
- **返回值：**HAL_OK：反注册成功；

此接口不会在外部直接调用，会先在 HAL_SndCard 层封装后再统一给到外部调用，封装后的接口如下代码段所示：

```
HAL_Status HAL_SndCard_CodecUnregisterAc107(void)
{
```

```
    return ac107_codec_unregister();
}
```

接口调用如下代码段所示：

```
__weak HAL_Status board_soundcard_deinit(void)
{
    ......

#if PRJCONF_AC107_SOUNDCARD_EN
    HAL_SndCard_CodecUnregisterAc107();
#endif

    ......

    return HAL_OK;
}
```

7 PDM 接口

针对于 AC107 用作 PDM 接口连接到 XR872 Internal Codec 的 DMIC 接口的应用场景，AC107 Codec 驱动中单独封装了两个 PDM 初始化和反初始化接口供外部调用，此时 AC107 Codec 不会与 Platform 绑定形成声卡设备（但需要调用 ac107_codec_register 接口进行注册以便自动检测确定 I2C 通信参数），而只是当作一个 DMIC 元器件来使用，只是使用前需要通过调用 ac107_pdm_init 接口配置一下 AC107 的相关寄存器，使用完后通过调用 ac107_pdm_deinit 复位 AC107 寄存器到上电默认状态即可。

7.1 ac107_pdm_init

- **原型**：HAL_Status ac107_pdm_init(Audio_Device device, uint16_t volume, uint32_t sample_rate)；
- **传参**：device：AUDIO_Device 类型音频设备；volume：配置音量；sample_rate：采样率；
- **功能**：AC107 Codec PDM 接口初始化；
- **返回值**：HAL_OK：配置成功；other：配置失败；

7.2 ac107_pdm_deinit

- **原型**：HAL_Status ac107_pdm_deinit(void)；
- **传参**：无
- **功能**：AC107 Codec PDM 接口反初始化，通过调用 ac107_dai_hw_free 接口对 AC107 进行软复位；
- **返回值**：HAL_OK：配置成功；other：配置失败；