基于 STM32 I2S 的音频应用开发介绍

前言

在音频开发中，I2S（Inter-IC Sound）接口被广泛采用。大部分 STM32 集成了 I2S 接口。本文主要为了让 STM32 使用者了解 I2S 音频接口，及快速实现 I2S 接口的音频应用开发。 首先，对 STM32 的 I2S 接口进行简单介绍，然后描述了几种常见 I2S 音频应用架构及每种架构音频部分的电路图，最后围绕每种架构给出实现例，以便读者进行参考理解。其中，实现例会围绕 STM32CubeMX 展开，以便开发者能够参考并快速、简便地实现软件开发。除此之外，在 Cube 软件包中有 I2S 外设应用例程，提供了更完善的实现参考。

一 STM32 I2S 接口简介

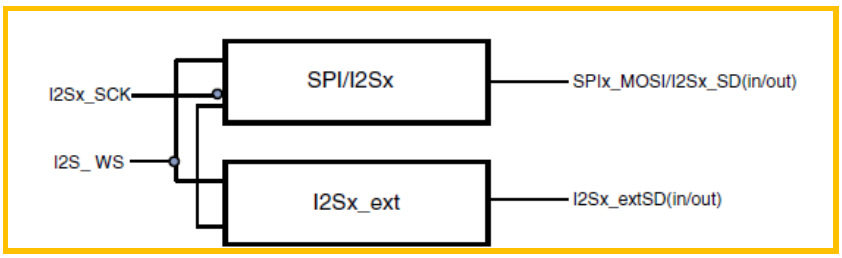
I2S(Inter-IC Sound)是飞利浦公司针对数字音频设备之间的音频数据传输，制定的一种总线标准。

STM32 I2S 接口信号线构成如下表：

信号线名	功能	描述	主模式	从模式
MCK	主时钟（系统时钟）	I2S 工作于主模式时，可用于为外部设备提供系统时钟。工作于从模式时，不可用。	可选（建议采用）	不可用
CK	串行时钟	位时钟，对于数字音频的每一位数据。	必须	必须
WS	字段选择	用于切换左右通道，或者作为帧时钟	必须	必须
SD	串行数据	发送或者接收数据	必须	必须
SD_Ext	串行数据	接收或者发送数据 （仅全双工 I2S 外设支持）	可选（全双工时必须）	可选（全双工时必须）

其中，SD 和 SD_Ext 信号线可分别配置为发送或者接收。在 Cube 驱动库中已对其进行封装，例如当配置 SD 信号线为发送端时，SD_Ext 自动被配置为接收端；配置 SD 为接收端时，SD_Ext 自动被配置为发送端。

全双工 I2S 是由两个 I2S 外设组成，如下图所示。



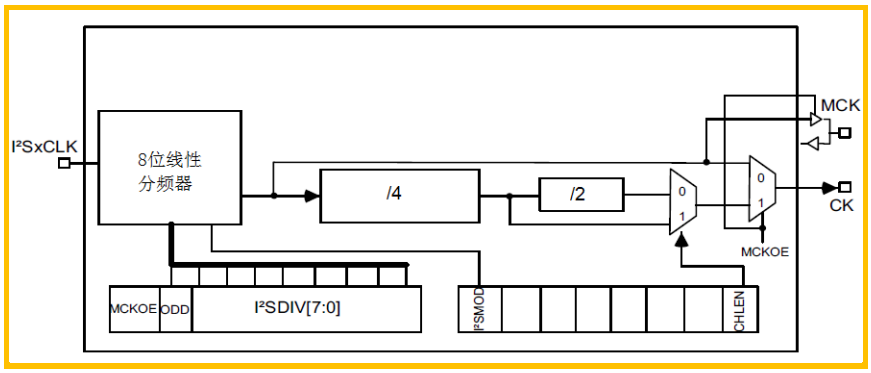
对于构成全双工 I2S 的每个 I2S 外设，都具有单独的寄存器组，如下表所示（以 STM32F413xG/H 为例）。在 Cube 驱动库中，全双工下的两个 I2S 外设操作已经被封装，用户只需像配置一个全双工 SPI 一样，对一个全双工 I2S 的 API 进行调用即可。后续会以实例形式进行描述。

外设	寄存器地址空间
I2S3ext	0x4000 4000 - 0x4000 43FF
I2S3/SPI3	0x4000 3C00 - 0x4000 3FFF
I2S2/SPI2	0x4000 3800 - 0x4000 3BFF
I2S2ext	0x4000 3400 - 0x4000 37FF

STM32 I2S 支持四种接口标准和数据格式，如下表。更多内容请参考对应型号 STM32 的参考手册。

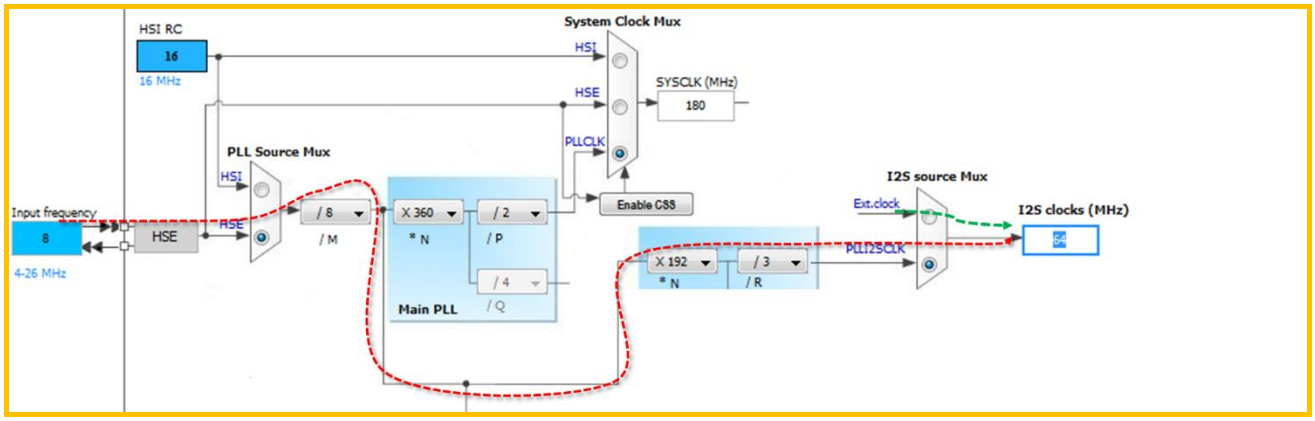
接口标准	数据格式
I2S 标准（Philips 标准）、左对齐标准、右对齐标准、PCM 标准	16 位数据 @16 位帧、16 位数据 @32 位帧、24 位数据 @32 位帧、32 位数据 @32 位帧

由表可看出，STM32 I2S 支持音频分辨率可为 16,24 和 32 位。I2S 时钟配置及数据格式选择决定了音频采样率，时钟产生架构如下图所示。不同系列 STM32 I2S 接口能够支持的最大音频采样率有差异，更多采样率支持情况请参考对应型号 STM32 的参考手册。



图中 MCK、CK 分别对应 I2S 总线上的主时钟和总线时钟。其中 I²SxCLK 获取路径如下图所示（对应于右侧的 I2S clocks）。红色线路或者绿色线路可选，本文中以红色线路为例，利用 PLL 时钟源获取 I²SxCLK 时钟。

注：下图是 STM32F429 时钟配置图的部分。不同型号 STM32 的时钟树存在差异，具体以实际采用型号的时钟树为准。



在遵循 I2S 标准的实现方案中，采样率公式如下 （注：Fs 为采样率，得益于 Cube 驱动库中的良好 API 实现，可以直接设置采样率，使用者不需要按照下述公式进行 I2SDIV 和 IDD 的计算及配置。）：

主时钟	帧位宽（位）	采样率
使能	16	$\mathrm{Fs} = \frac{I^2SxCLK}{[(16*2)*((2*\mathrm{I2SDIV})+\mathrm{ODD})*8]}$
	32	$\mathrm{Fs} = \frac{I^2SxCLK}{[(32*2)*((2*\mathrm{I2SDIV})+\mathrm{ODD})*4]}$
失能	16	$\mathrm{Fs} = \frac{I^2SxCLK}{[(16*2)*((2*\mathrm{I2SDIV})+\mathrm{ODD})]}$
	32	$\mathrm{Fs} = \frac{I^2SxCLK}{[(32*2)*((2*\mathrm{I2SDIV})+\mathrm{ODD})]}$

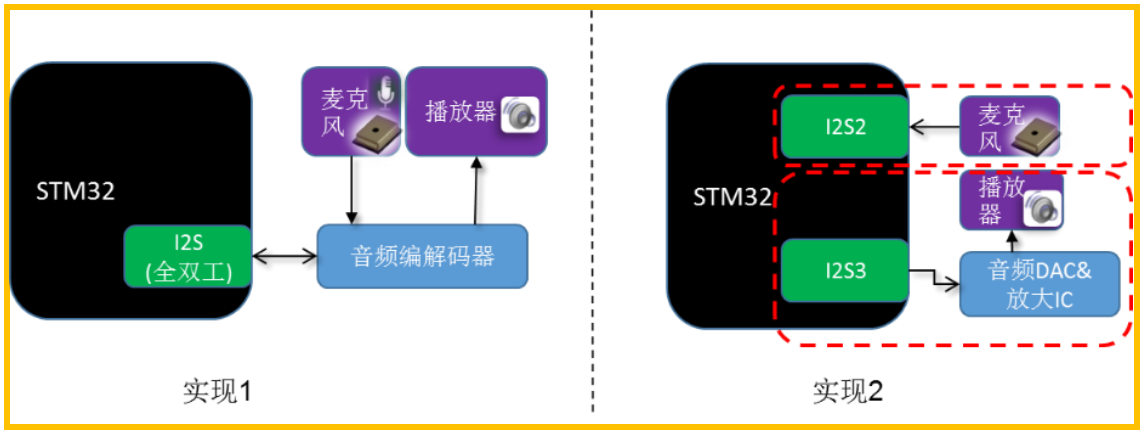
上述采样率公式不能直接用于 PDM 输出的 MEMS 麦克风，通过后一节中介绍可知，PDM 麦克风访问只是利用了 I2S 的数据和时钟线，并且在采集到麦克风位流数据后，需要经过降频操作（PDM 转 PCM，ST 提供了 PDM 转 PCM 库支持，更多介绍可参考 AN3998），从而获得 PCM 数据。所以，在这种情况下，主时钟配置为失能，数据位宽需要与帧位宽相同。折算后的采样率为：

主时钟	帧位宽（位）	采样率	
		单麦克风	双麦克风
失能	16	$\frac{I^2SxCLK}{[(2*\mathrm{I2SDIV})+\mathrm{ODD}]*DIV} = \frac{(16*2)}{DIV}Fs$	$\frac{I^2SxCLK}{[(2*\mathrm{I2SDIV})+\mathrm{ODD}]*DIV*2} = \frac{16}{DIV}Fs$
	32	$\frac{I^2SxCLK}{[(2*\mathrm{I2SDIV})+\mathrm{ODD}]*DIV} = \frac{(32*2)}{DIV}Fs$	$\frac{I^2SxCLK}{[(2*\mathrm{I2SDIV})+\mathrm{ODD}]*DIV*2} = \frac{32}{DIV}Fs$

其中，DIV 为 PDM 转 PCM 的降频因子，由调用的 API 决定。

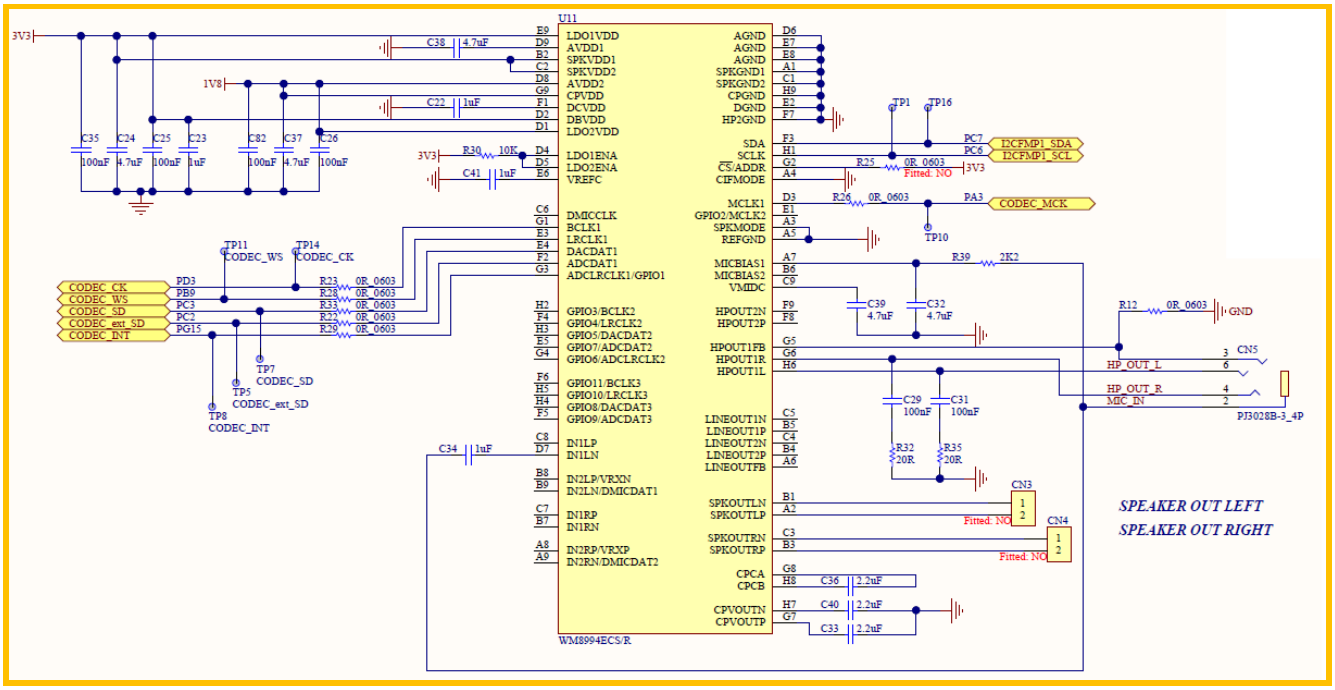
二 常见 I2S 接口音频应用实现

I2S 接口应用相对固定，整理两种音频支持结构如下。

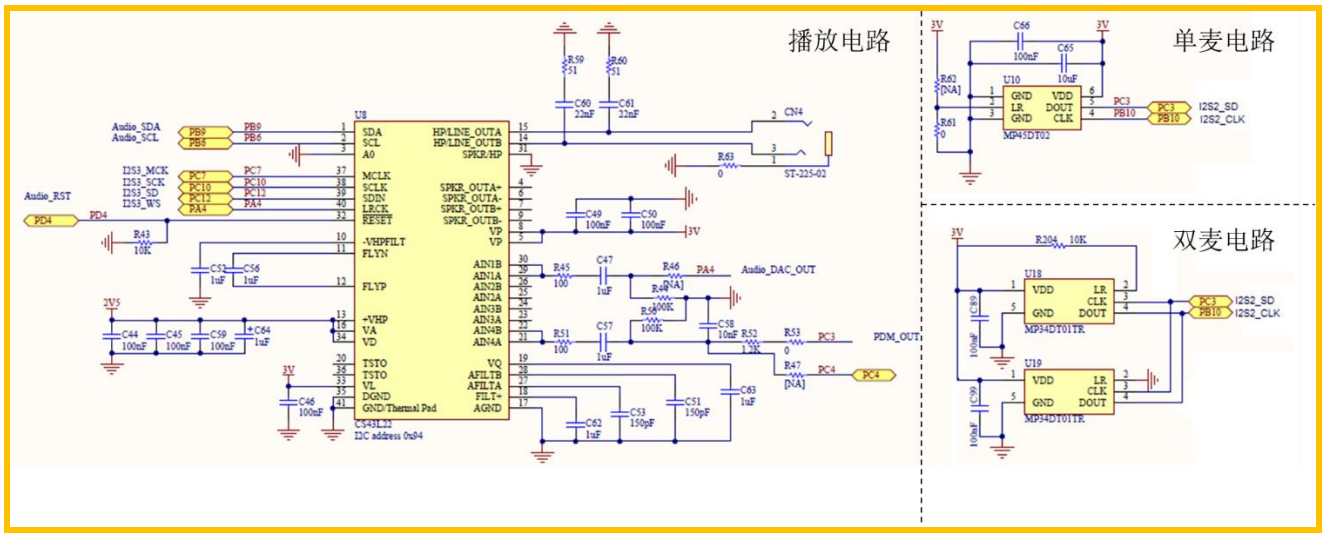


其中，麦克风与播放器功能的实现相互独立。可根据实现需要决定采用的实现架构。

实现 1 参考电路如下图。原理图摘自 STM32F413H-DISCO 板，可在 ST 官网获取完整的原理图及 BOM 表等资源。其中 CODEC_MCK、CODEC_CK、CODEC_WS、CODEC_SD、CODEC_ext_SD 分别对应 I2S 的 MCK、CLK、WS、SD 和 ext_SD 信号线。

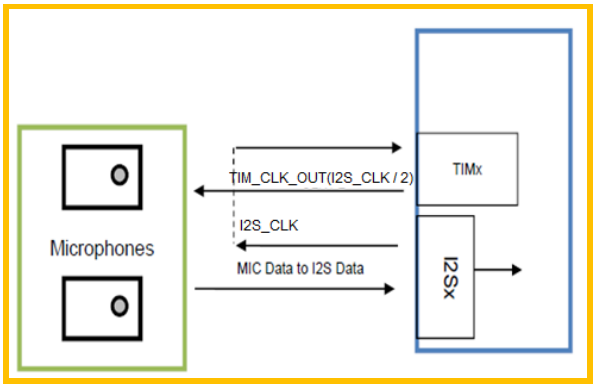


实现 2 参考电路如下图。其中单麦电路和双麦电路同时存在仅为读者参考理解，实际开发时可根据应用需要选择单麦或者双麦实现。原理图摘自 STM32F411E-DISCO 和 STM32429I-EVAL 板，可在 ST 官网获取完整的原理图及 BOM 表等资源。

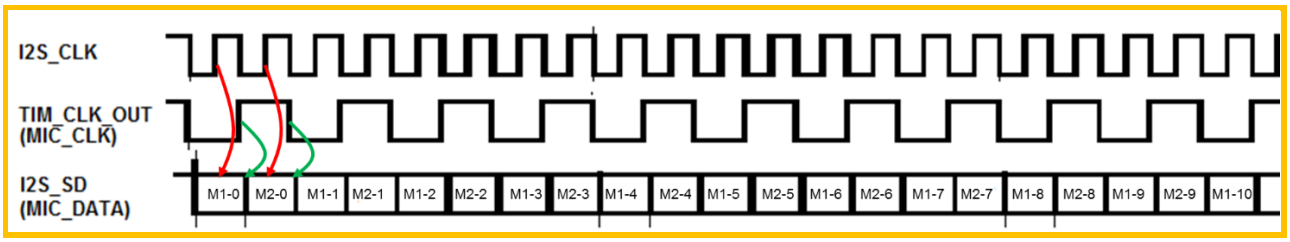


在实现 2 中，直接采集麦克风数据。市面上 MEMS 麦克风有 PCM 输出和 PDM 输出之分，其中 PDM 麦克风由于内部结构相对简单，成本更低，被大量采用。图中 MP45DT02 和 MP34DT01TR 都为 PDM 输出的 MEMS 麦克风。PDM 数据不能直接使用，需要经过滤波，降频等操作获得 PCM 数据。

另外，I2S 对双麦克风的支持需要结合定时器及 2 个 IO 复用引脚。实现框架如下图。



通过定时器对 I2S_CLK 信号进行两分频输出，然后将获得的信号提供给 MEMS 麦克风的数据线。实现时序图如下所示。依据 I2S 标准（Pilips 标准、左对齐标准和右对齐标准）时序， I2S_CLK 的上升沿获取数据。而对于文中提及的两种 MEMS 麦克风，输入时钟（TIM_CLK_OUT）的下降沿使得左通道麦克风（LR 引脚下拉）输出有效数据，右通道麦克风（LR 引脚上拉）数据线进入高阻态；输入时钟的上升沿，左通道麦克风数据线进入高阻态，右通道麦克风输出有效数据。从而实现双麦克风采集。



三 应用实现例

本节围绕上述介绍的两种典型实现架构，结合 ST 评估板，介绍 I2S 接口应用在 STM32CubeMX 工具上配置实现，以及在生成工程后的 API 调用，最终实现基于 I2S 接口的音频数据传输。利用 STM32CubeMX，能够更快的实现针对自定义 STM32 平台的开发。实现流程如下。

4.1 前期准备

实现例	STM32 板	USB 线	STM32CubeMX	Cube 软件包	IDE
实现 1	STM32F413H-DISCO * 1 **链接：** http://www.st.com/content/st_com/en/products/evaluation-tools/product-evaluation-tools/mcu-eval-tools/stm32-mcu-eval-tools/stm32-mcu-discovery-kits/32f413hdiscovery.html	USB 线 (Type A ←→ Micro B)	**链接：** http://www.st.com/content/st_com/en/products/development-tools/software-development-tools/stm32-software-development-tools/stm32-configurators-and-code-generators/stm32cubemx.html	STM32CubeF4 **说明：**在安装 STM32CubeMX 后，在其“菜单栏\Help\Install New Libraries”中安装 STM32CubeF4.本文实现例中采用的是 STM32Cube_FW_F4_V1.16.0	IAR **链接：** https://www.iar.com/ **说明：**本文中以 IAR 为例。除 IAR 外，CubeMX 还支持 MDK、TrueStudio、SW4STM32
实现 2	STM32F4-Discovery **链接：** http://www.st.com/content/st_com/en/products/evaluation-tools/product-evaluation-tools/mcu-eval-tools/stm32-mcu-eval-tools/stm32-mcu-discovery-kits/stm32f4discovery.html%20	USB 线 (Type A ←→ Mini B)			

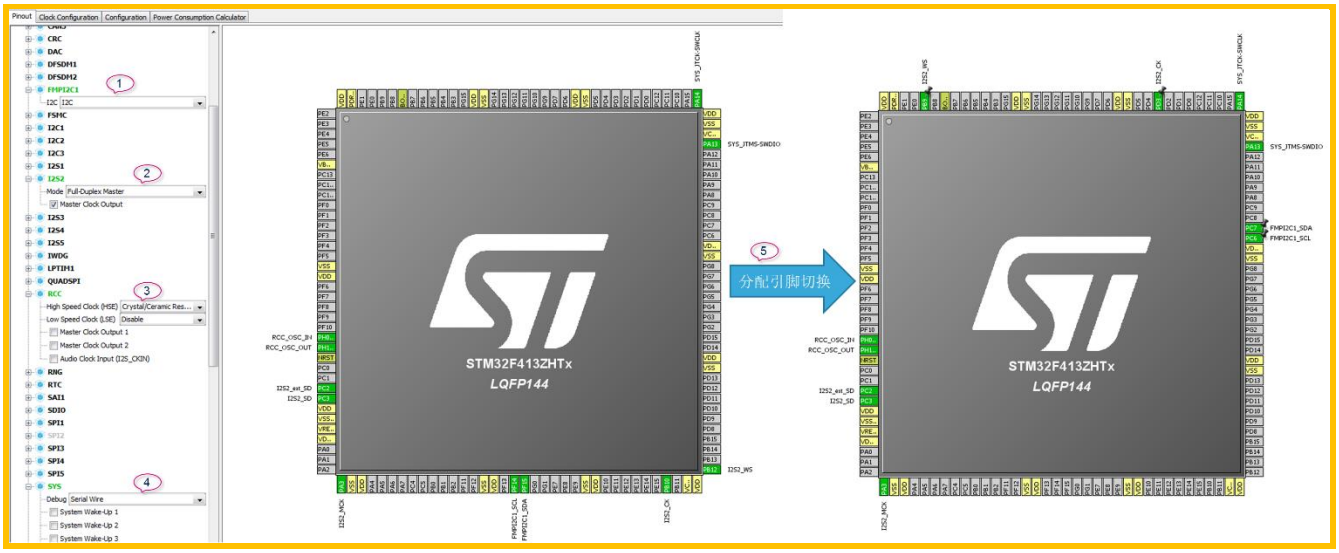
4.2 应用实现

4.2.1 实现 1

结合 STM32CubeMX 的软件开发流程如下图。

接下来一步一步呈现实现过程。

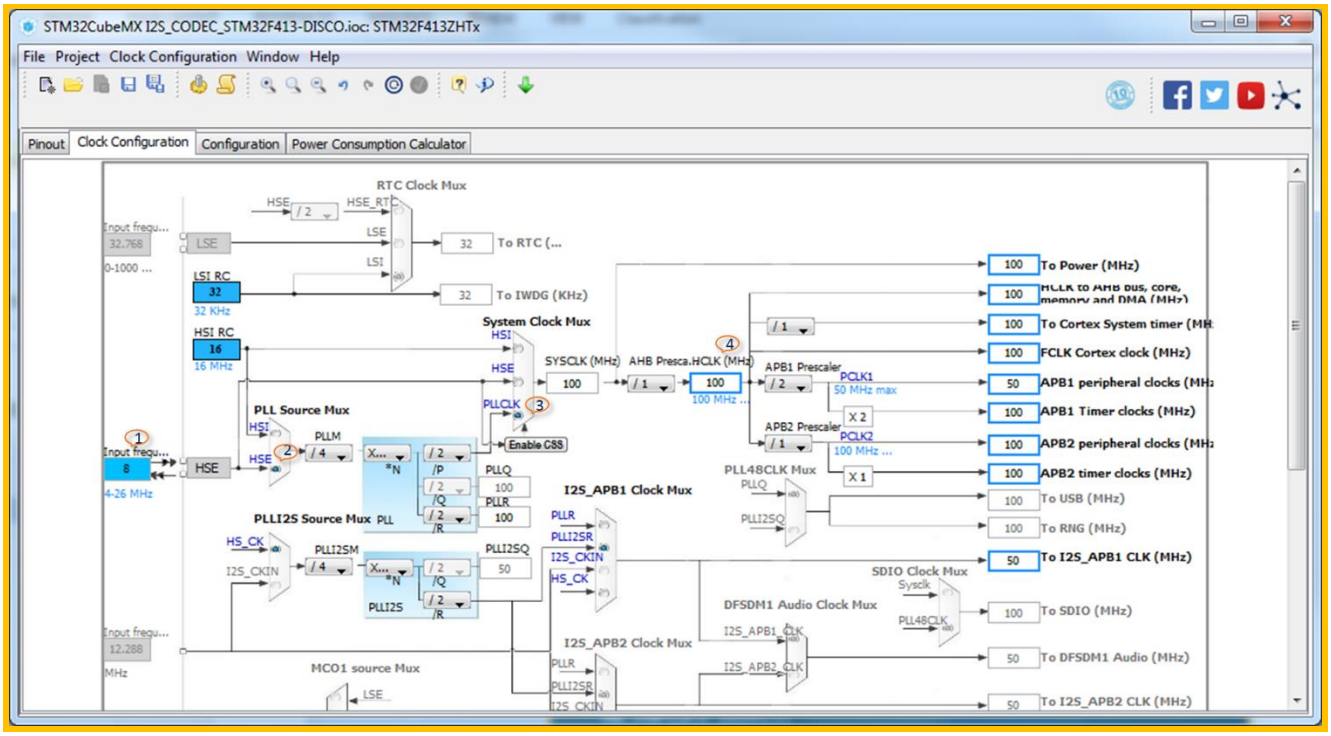
步骤 1：在 STM32CubeMX 中根据硬件选择 STM32F413ZHTx、外部时钟、调试接口、I2C 通道和 I2S 通道（利用 I2C 接口配置和控制编解码器），如下截图。硬件电路原理图可以在上节的链接网址中获取。其中，I2S 工作于主模式。



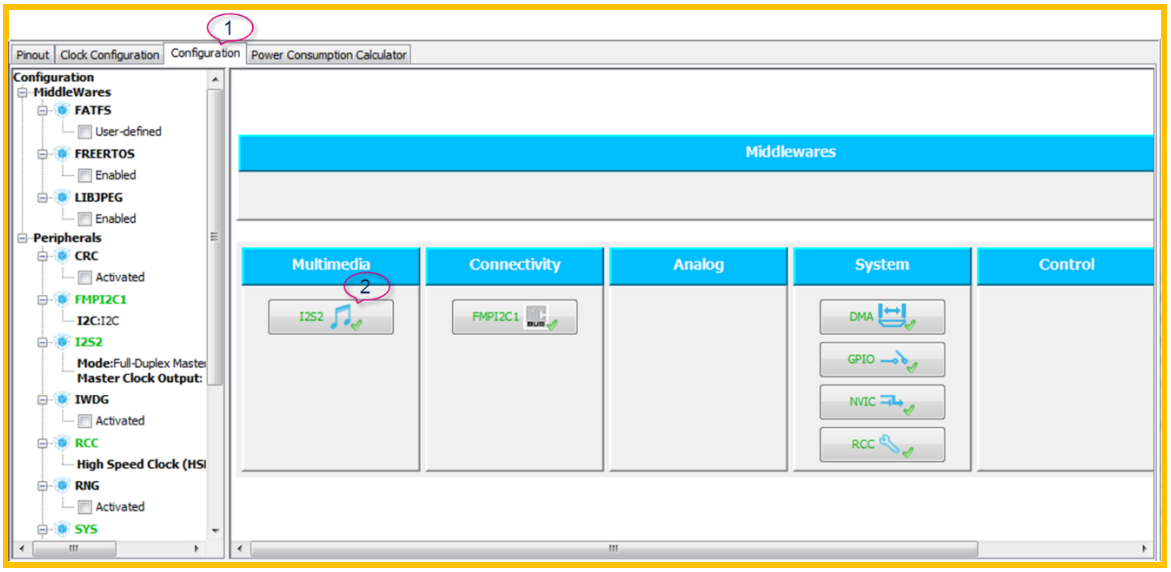
选择各外设后，由于外设功能可关联到不同的引脚，自动分配的引脚可能与硬件连接的引脚不一致。此时，可以在需要重新关联的引脚上按住 Ctrl 键+鼠标左键按下，出现支持相同功能的全部复用引脚，将其拖动到与硬件设计一致的引脚上。如上图步骤 5 所示。

步骤 2：时钟配置。时钟配置涉及环节较多，STM32CubeMX 提供了便捷的时钟配置实现，如下图，只需简单的几步，即可获得最高主频运行的时钟配置。需要注意“ Input Frequency ”值，应保持与外部高速时钟一致。

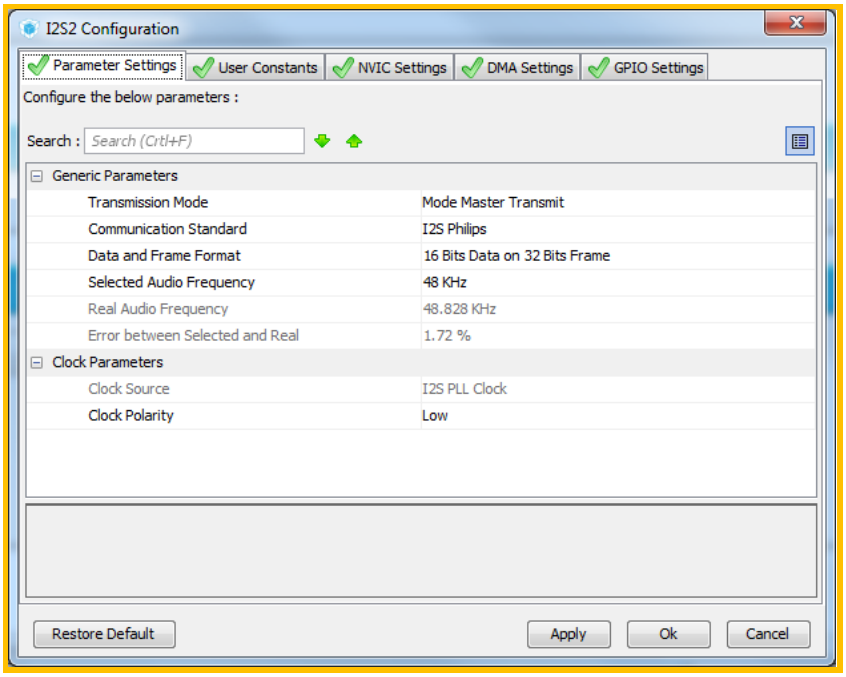
尽管在上述 I2S 接口介绍中， I2S 采样率与时钟配置有关联，但在 HAL 库实现中会根据 I2S 中的采样率参数，自动完成时钟参数配置。



步骤 3：I2S 配置。点击切换到 Configuration 标签页，按照如下步骤进入 I2S 配置界面。



I2S 参数配置。配置后截图如下。



Transmission Mode: 传输模式。决定 SD 数据线传输方向（SD_Ext 方向相反）。根据硬件设计， I2Sx_SD 向编解码器输出数据，所以选择发送模式。

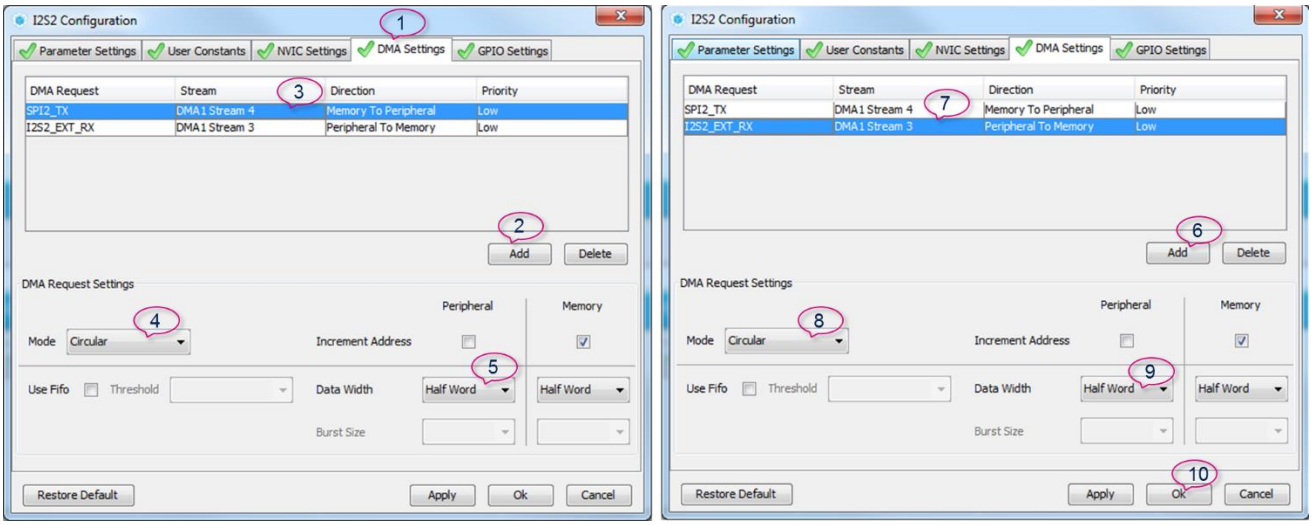
Communication Standard: 传输标准。本文中采用 I2S Philips 标准（需要利用 I2C 发送命令配置 WM8994 工作于相同标准）。

Data and Frame Format: 数据位宽和帧位宽。如同“传输标准”的配置，保持与编解码器配置一致。

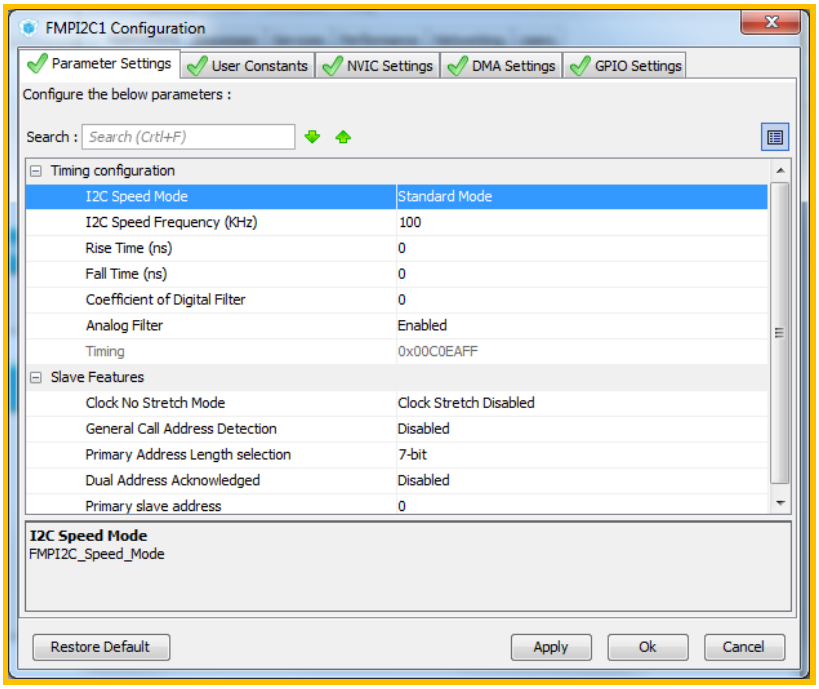
Selected Audio Frequency: 音频频率。可选频率 8KHz、11KHz、16KHz、22KHz、32KHz、44KHz、48KHz、96KHz、192KHz，这里选择 48KHz。如同“传输标准”的配置，保持与编解码器配置一致。

Clock polarity:时钟极性（非激活态时）。

I2S DMA 设置。切换到 DMA Settings 标签页，按照下图步骤设置。（图为设置完成后截图。）



步骤 4：I2C 配置。点击 FMPI2C1 图标进入 FMPI2C 配置界面，参数配置如下图。参数介绍请参考对应型号的参考手册。

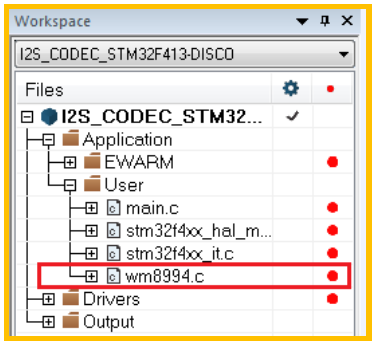


步骤 5：生成 IAR 工程文件。在菜单栏 \ Project \ Settings 打开工程设置界面，设置工程名、工程存放位置以及对应的 IDE 工具（本文中采用 IAR EWARM）。其他保持默认，更多参数介绍请参考 UM1718。

点击菜单栏 \ Project \ Generate code 生成工程。STM32CubeMX 生成工程中包含了时钟、外设等初始化。开发者可以在此基础上增加函数调用实现应用开发。

步骤 6：利用 IAR EWARM 打开工程，添加 API 调用，实现音频数据传输，具体步骤如下。

1. 添加编解码器 wm8994 的驱动函数。为简化操作，直接采用 Cube 软件包中提供的 wm8994 驱动文件 wm8994.c，wm8994.h 和 audio.h（文件位于 STM32Cube_FW_F4_V1.16.0\Drivers\BSP\Components\wm8994 和 STM32Cube_FW_F4_V1.16.0\Drivers\BSP\Components\Common）。其中 wm8994.c 复制到 src 文件夹中，wm8994.h 和 audio.h 复制到 inc 文件夹中，并添加到工程中，如下图。



2. 按照下表增添应用代码。实现如下音频数据流向。为简便起见，直接将 I2S 接收和发送关联到同一个存储空间，并同时执行，在实际应用中，应加以优化完善，避免读写位置交错引起的错误。

   另外，由于硬件上仅有一路麦克风通道输入，尽管采用双麦克风通道配置，但有一路麦克风通道无有效数据，体现在耳机输出上，有一路无有效输出。

3. 编译生成执行文件，下载运行。

注：下表中省略号表示，之间有其他未列出的代码部分。

文件名	函数名	操作

main.h		/* 增加如下代码 */
		#include <stdint.h>
		/**
		* @brief Audio I2C Slave address
		*/
		#define AUDIO_I2C_ADDRESS ((uint16_t)0x34)
		/* Codec output DEVICE */
		#define OUTPUT_DEVICE_SPEAKER ((uint16_t)0x0001)
		#define OUTPUT_DEVICE_HEADPHONE ((uint16_t)0x0002)
		#define OUTPUT_DEVICE_BOTH ((uint16_t)0x0003)
		#define OUTPUT_DEVICE_AUTO ((uint16_t)0x0004)
		#define INPUT_DEVICE_DIGITAL_MICROPHONE_1 ((uint16_t)0x0100)
		#define INPUT_DEVICE_DIGITAL_MICROPHONE_2 ((uint16_t)0x0200)
		#define INPUT_DEVICE_INPUT_LINE_1 ((uint16_t)0x0300)
		#define INPUT_DEVICE_INPUT_LINE_2 ((uint16_t)0x0400)
		#define INPUT_DEVICE_DIGITAL_MIC1_MIC2 ((uint16_t)0x0800)
		
		#define AUDIO_VOLUME 80
		
		void AUDIO_IO_Write(uint8_t Addr, uint16_t Reg, uint16_t Value);
		uint16_t AUDIO_IO_Read(uint8_t Addr, uint16_t Reg);
		uint8_t WM8994_Config(void);

main.c		/* 添加 wm8994 驱动头文件 */ /* USER CODE BEGIN Includes */ #include "wm8994.h"

```
……
/* 增加音频数据流暂存空间 */
/* USER CODE BEGIN PV */
/* Private variables ---------------------------------------------
--------------*/
uint8_t AudioBuf[2*2*48] = {0};    //2 CH*16bit*48KHz
```

```
/* USER CODE BEGIN 4 */
/**
  * @brief  Writes a single data.
  * @param  Addr: I2C address
  * @param  Reg: Reg address
  * @param  Value: Data to be written
  * @retval None
  */
void AUDIO_IO_Write(uint8_t Addr, uint16_t Reg, uint16_t Value)
{
  uint16_t tmp = Value;
  Value = ((uint16_t)(tmp >> 8) & 0x00FF);
  Value |= ((uint16_t)(tmp << 8)& 0xFF00);
  /* Check the communication status */
  if(HAL_FMPI2C_Mem_Write(&hfmpi2c1, Addr, (uint16_t)Reg,
FMPI2C_MEMADD_SIZE_16BIT, (uint8_t*)&Value, 2, 1000) != HAL_OK)
  {
    /* De-initialize the I2C communication bus */
    HAL_FMPI2C_DeInit(&hfmpi2c1);
    /* Re-Initialize the I2C communication bus */
    MX_FMPI2C1_Init();
  }
}
/**
  * @brief  Reads a single data.
  * @param  Addr: I2C address
  * @param  Reg: Reg address
  * @retval Data to be read
  */
uint16_t AUDIO_IO_Read(uint8_t Addr, uint16_t Reg)
{
  uint16_t read_value = 0, tmp = 0;
  if(HAL_FMPI2C_Mem_Read(&hfmpi2c1, Addr, (uint16_t)Reg,
FMPI2C_MEMADD_SIZE_16BIT, (uint8_t*)&read_value, 2,
1000) !=HAL_OK)
  {
    /* De-initialize the I2C communication bus */
    HAL_FMPI2C_DeInit(&hfmpi2c1);
    /* Re-Initialize the I2C communication bus */
    MX_FMPI2C1_Init();
  }
```

```
  tmp = ((uint16_t)(read_value >> 8) & 0x00FF);
  tmp |= ((uint16_t)(read_value << 8)& 0xFF00);
  read_value = tmp;
  return read_value;
}
/**
  * @brief  Initializes Audio low level.
  */
void AUDIO_IO_Init(void)
{
    //Already defined in MX_FMPI2C1_Init().
}
void    AUDIO_IO_Delay(uint32_t Delay)
{
  HAL_Delay(Delay);
}
uint8_t WM8994_Config(void)
{
  uint8_t ret = HAL_ERROR;
  uint32_t deviceid = 0x00;
  /* wm8994 codec initialization */
  deviceid = wm8994_ReadID(AUDIO_I2C_ADDRESS);
  if((deviceid) == WM8994_ID)
  {
    /* Reset the Codec Registers */
    wm8994_Reset(AUDIO_I2C_ADDRESS);
    ret = HAL_OK;
  }
  else
  {
    ret = HAL_ERROR;
  }
  if(ret == HAL_OK)
  {
    /* Initialize the codec internal registers */
    wm8994_Init(AUDIO_I2C_ADDRESS,
INPUT_DEVICE_INPUT_LINE_1|OUTPUT_DEVICE_AUTO, AUDIO_VOLUME,
I2S_AUDIOFREQ_48K);
  }
  if(wm8994_SetMute(AUDIO_I2C_ADDRESS, AUDIO_MUTE_OFF) != 0)
  {
    ret = HAL_ERROR;
  }
  return ret;
}
/* USER CODE END 4 */
```

main

```
/* USER CODE BEGIN 2 */
```

		HAL_I2SEx_TransmitReceive_DMA(&hi2s2,(uint16_t *)AudioBuf,(uint16_t *)AudioBuf,(sizeof(AudioBuf)/2));
		
		if(WM8994_Config() != HAL_OK)
		{
		Error_Handler();
		}

wm8994.h		/* 修改头文件路径 */
		//#include "../Common/audio.h"
		#include "audio.h"
		#include "main.h"
		……
		/* 修改函数声明 */
		//uint8_t AUDIO_IO_Read(uint8_t Addr, uint16_t Reg);
		uint16_t AUDIO_IO_Read(uint8_t Addr, uint16_t Reg);

wm8994.c	wm8994_Init	/* 修改参数，降低放大倍数*/
		case INPUT_DEVICE_INPUT_LINE_1 :
		……
		/* Disable mute on IN1L_TO_MIXINL and +30dB on IN1L PGA output */
		//counter += CODEC_IO_Write(DeviceAddr, 0x29, 0x0035);
		counter += CODEC_IO_Write(DeviceAddr, 0x29, 0x0033);
		
		/* Disable mute on IN1R_TO_MIXINL, Gain = +30dB */
		//counter += CODEC_IO_Write(DeviceAddr, 0x2A, 0x0035);
		counter += CODEC_IO_Write(DeviceAddr, 0x2A, 0x0033);
		……
		/* 修改参数，避免后续操作将麦克风偏置 1 关闭*/
		/* Enable bias generator, Enable VMID, Enable HPOUT1 (Left) and Enable HPOUT1 (Right) input stages */
		/* idem for Speaker */
		//power_mgnt_reg_1 |= 0x0303 | 0x3003;
		if(input_device > 0)
		power_mgnt_reg_1 |= 0x0303 | 0x3003 | 0x0010;
		else
		power_mgnt_reg_1 |= 0x0303 | 0x3003;

由上述添加及修改，可知在利用 STM32CubeMX 配置、生成工程后，I2S 数据接收和发送实现方便，只需要调用 HAL 库提供的 API 即可。工作较多集中在 STM32 的音频接口了解和编解码器功能配置方面。编解码器方面，一般编解码器厂商会有文档、配套工具或者配置例程提供。

4.2.2 实现 2

这种架构实现例，可参考 Cube 软件包中提供的例程，不再做展开介绍。例程路径如下：

1.STM32Cube_FW_F4_V1.16.0\Projects\STM32F401-Discovery\Applications\Audio\Audio_playback_and_record

2.STM32Cube_FW_F4_V1.16.0\Projects\STM32F411E-Discovery\Applications\Audio\Audio_playback_and_record

四 低功耗设计

不同功耗模式下，I2S 工作情况如下表。

模式 (1)	描述	批处理模式（BAM）
运行	激活	-
睡眠	激活。外设中断或者事件触发退出睡眠模式。	支持(2)
停止	冻结。外设寄存器保持。	-
待机	掉电。	-

注1. 不同系列 STM32，低功耗模式有差异，具体以参考手册为准。

注2. 批处理模式（BAM）并非所有 STM32 产品都支持，支持情况请以对应型号的 STM32 参考手册中描述为准。BAM 能够实现在睡眠模式下，批量获取数据，然后再退出睡眠模式进行运算处理等操作。能够进一步实现功耗的降低。更多关于 BAM 介绍可参考 RM0430。

五 小结

在 STM32 音频开发中，结合 STM32CubeMX 和 Cube 软件包中提供的例程，容易完成基于 STM32 的音频平台搭建。I2S 各协议时序清晰，开发过程中，出现异常时，开发者可以先利用示波器，判断波形对应时序是否正确，以此缩小问题范围，加快问题定位及解决。

参考文档

RM0430 STM32F413/423 advanced ARM®-based 32-bit MCUs

AN2739 How to use the high-density STM32F103xx microcontroller to play audio files with an external I²S audio codec

AN3998 PDM audio software decoding on STM32 microcontrollers

UM1718 STM32CubeMX for STM32 configuration and initialization C code generation

重要通知 – 请仔细阅读

意法半导体公司及其子公司（“ST”）保留随时对ST 产品和/ 或本文档进行变更、更正、增强、修改和改进的权利，恕不另行通知。买方在订货之前应获取关于ST 产品的最新信息。ST 产品的销售依照订单确认时的相关ST 销售条款。

买方自行负责对ST 产品的选择和使用， ST 概不承担与应用协助或买方产品设计相关的任何责任。

ST 不对任何知识产权进行任何明示或默示的授权或许可。

转售的ST 产品如有不同于此处提供的信息的规定，将导致ST 针对该产品授予的任何保证失效。

ST 和ST 徽标是ST 的商标。所有其他产品或服务名称均为其各自所有者的财产。

本文档中的信息取代本文档所有早期版本中提供的信息。