应用报告
ZHCA505A–June 2012–Revised November 2012

双 SPI 仿真 I²S，此仿真在 Stellaris® LM4F MCU 上实现

Michael Risley
Steve Sager

Stellaris® Microcontrollers
Digital Field Applications

摘要

这份应用报告给出了一个使用两个串行外设接口 (SPI) 来仿真一个集成音频接口芯片 (I²S) 外设，此外设被集成在 Stellaris® LM4F232 微控制器上。 通过采用音频 API，存储在板载安全数据 (SD) 卡上的 .wav 音频文件由伪 I²S 播放至一个外部德州仪器的TLV320AIC3107编解码器内。 这个通用软件平台可被定制成将音频功能性添加到嵌入式系统中。

内容

图片列表

图表列表

1 简介

集成音频接口芯片 (I²S) 是一款广泛应用于数字音频传输的电气串行总线接口。 很多嵌入式应用将 I²S 用于音频回放，而大多数现代微控制器通常提供 I²S 支持。 如果这个外设在微控制器不可用，或者需要额外的 I²S，可使用现有资源来仿真此外设。 这份应用报告使用两个串行外设接口 (SPI) 来检查一个伪 I²S 接口的执行，这两个 SPI 接口使用户能够克服硬件限制或解决缺少专用外设的问题。 这份应用报告的基础是一个演示，在这演示中，Stellaris LM4F232H5QD 微控制器使用这个伪 I²S 接口将音频播放至 TITLV320AIC3107EVM-K。 这个演示是开源代码的并可从12 节，参考中的连接中下载。 这份文档对此演示的特定的设置进行了说明，如何在一个普通 Stellaris 微控制器上创建伪 I²S，以及音频驱动程序的重要特性的配置。

2 总体概述，双 SPI 到 I²S

I²S 接口通常由一个具有至少三个信号的总线组成：位时钟 (BCLK)，数据输入 (DIN) 和字时钟 (WCLK)。 这三个信号被用在图 1内的连接图中所示的双 SPI 执行中。 当一个外部 I²S 器件被配置为主控时，SPI 模块由 I²S 接口的位和字时钟控制。 通过使用字时钟信号来启用或禁用微控制器的受控 SPI 端口，可创建一个伪 I²S 接口。

注: SPI 数据帧选择线路被倒置，这使得数据传输在两个 SPI 端口间切换。

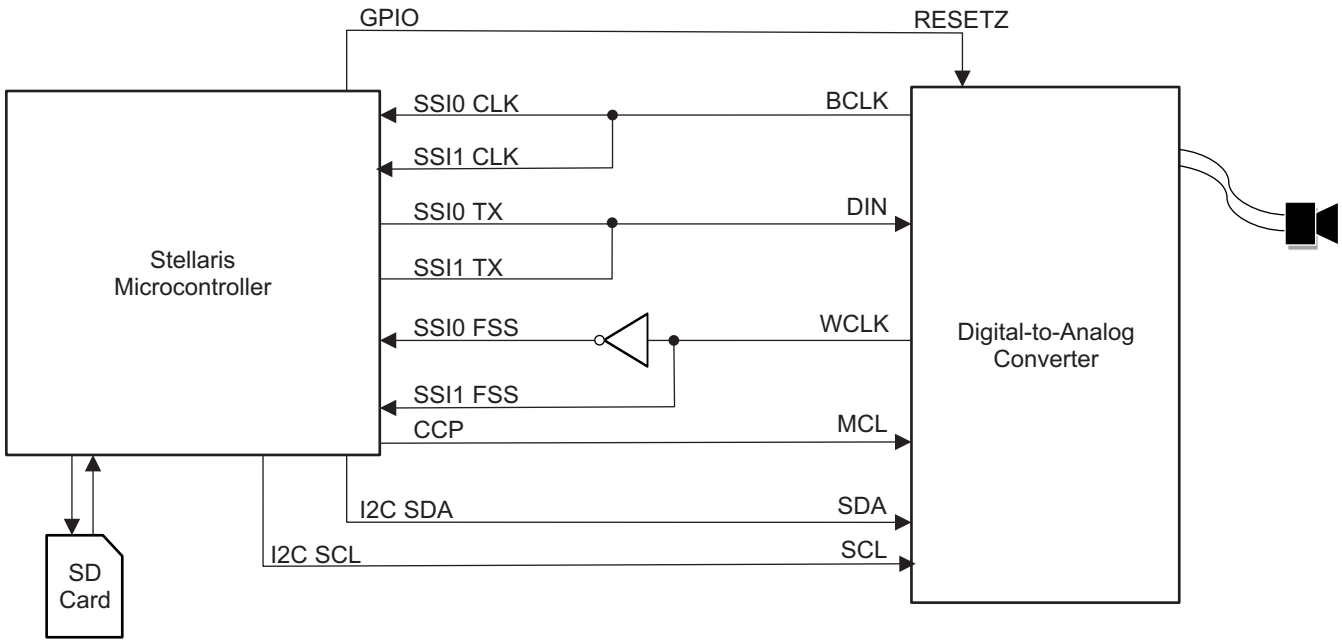


图 1. 普通系统连接

一个 Stellaris 微控制器的捕获和比较输出 (CCP) 被用来提供编解码器主控时钟 (MCLK)。 在这执行中，此编解码器与一个 25MHz MCLK 一同提供。 然后，这个编解码器内部锁相环路 (PLL) 根据所需的音频采样频率生成需要的 BCLK 和 WCLK 信号。

图 1顶部的 GPIO 输出被用来将编解码器 RESETZ 线路的电平拉低来复位编解码器。 这个 GPIO 在任一数据被传送到编解码器之前被切换以触发一个复位，从而确保正常功能。

为了配置数据路径、计时、线路输出和另外的项目，内部集成电路 (I²C) 信号，SDA 和 SCL，被用来写入到内部编解码器寄存器。

图 1中显示的 SD 卡被用来保存脉冲编码调制 (PCM) .wav 音频数据。 这个 .wav 文件可被存储在其它位置；然而，这个示例从 SD 卡中读取数据来演示一个更加广泛的且容量更大的存储介质。

3 硬件执行，LM4F232 到 TLV320AIC3107

双 SPI 音频示例由一个连接到外部 TLV320AIC3107EVM-K 的 Stellaris LM4F232H5QD 微控制器建立并进行验证。图 2显示了这些主板的外部走线。 这个部分对建立这个配置所需的硬件连接进行了说明。

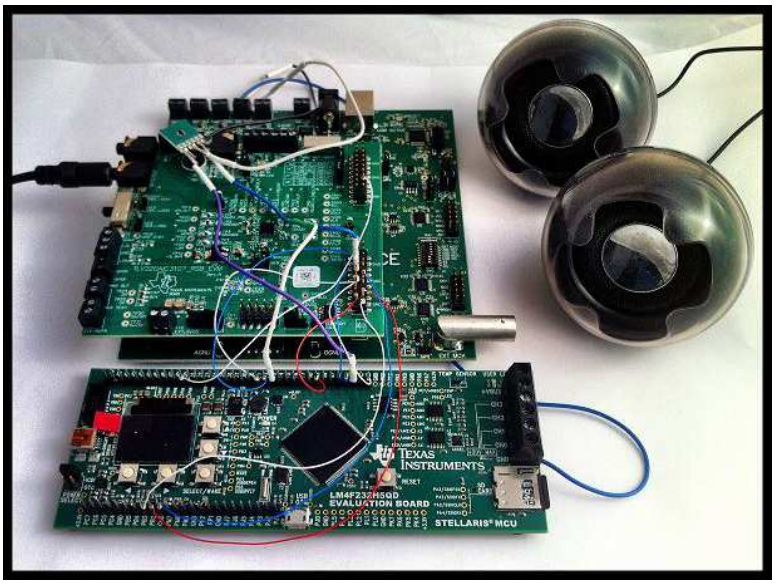


图 2. 完整系统设置

表 1显示了针对这个配置的引脚至引脚走线。 例如，第三个 SPI 外设 (SSI3Tx) 的 Stellaris 传输线路必须被路由至编解码器评估模块 (EVM) 的 I^2S 数据输入 (DIN)。 将被连接的相应引脚为 PH3 至 P11（在 J5 排针上）。

表 1. LM4F 至 TLV320AIC3107 EVM 连接

Stellaris LM4F232H5QD 引脚（外设）	TI TLV320AIC2107 编解码器 EVM 引脚（外设）
PF2 (SSI1Clk)	J5/P3 (BCLK)
PF3 (SSI1Fss)	J5/P7 (WCLK)
PF1 (SSI1Tx)	J5/P11 (DIN)
PH0 (SSI3Clk)	J5/P3 (BCLK)
PH1 (SSI3Fss)	J5/P7 (WCLK)
PH3 (SSI3Tx)	J5/P11 (DIN)
PB2 (I2C0SCL)	J5/P16 (SCL)
PB3 (I2C0SDA)	J5/P20 (SDA)
PD0 (GPIO)	J4/P14 (RESETZ)
PG3 (T5CCP1)	J5/P17 (MCLK)

编解码器和 USB 基板可针对多重操作进行配置。 为了确保正常的功能性，必须正确设置开关和跳线。表 2和表 3显示了针对编解码器 EVM 和 USB 基板的所需配置。 对于开关 (SWn)，**1**表示*开*状态，而**0**表示关。对于跳线 (Jn)，**N**表示*未组装*而**P**表示*已组装*。

表 2. 编解码器 USB-MODEVM 硬件配置

说明	开关或跳线名称	状态
稳压器使能	SW1（2 个开关）	3.3VD EN 和 1.8VD EN
USB 配置	SW2（8 个开关）	EXT MCK，USB RST，USB SPI，USB MCK，USB I2S，OFF，OFF，OFF
IOVDD 选择	SW3（8 个开关）	3.3V，OFF，OFF，OFF，OFF，OFF，OFF，OFF
外部 I2C	J6（2 个跳线）	N，N
外部音频数据	J14（6 个跳线）	N，N，N，N，N，N
外部 SPI	J15（6 个跳线）	N，N，N，N，N，N
SDA 上拉电阻	JMP3	P
SCL 上拉电阻	JMP4	P
CNTL 或 FSX 选择	JMP5	FSX
MCLKI	JMP7	2–3
外部 MCLK	JMP8	N

表 3. TLV320AIC3107 EVM 硬件配置

说明	开关或跳线名称	状态
单端或差分输入	SW1	SE
电容或无电容	SW2	CAP
VDD 选择	SW3	+5VA
麦克风偏置电压选择	W9	1–2
板载麦克风	W10	P
板载麦克风	W11	P
耳机	W12	N
耳机	W13	N
IOVDD 选择	W14	1–2
GPIO1 输入	W15	P
RESET 输入	W16	P
EEPROM 选择	W17	N
SPOM 选择	W18	N
SPOP 选择	W19	N

注: 要实现 I²C 的正常运行，组装 USB 基板的 JMP3 和 JMP4 以提供一个 2.7kΩ 上拉电阻值。 再 R1 和 R2 过孔至 +V 之间并联一个 1kΩ 电阻器。 这个配置保证了一个大约为 720Ω 的弱上拉电阻值。特定的受控器件也许会造成足够的负载，以至于无声或噪声水平使得上拉电阻器很难提高 SCL 信号。

4 软件编解码器设置

在这执行中，德州仪器 TLV320AIC3107EVM-K 被用于外部编解码器。 这个评估模块是一个具有几个输入和输出、扩展音频线路、混音和特效功能的完整立体声音频编解码器。 如图 1中的连接图所示，共用七条独立线路与编解码器通信：RESETZ，BCLK，WCLK，MCLK，DIN，SDA 和 SCL。 此外，完全差分立体声扬声器应该被连接到 J10 上的 HP_OUTPUT。 具有寄存器的编解码器功能方框图可在TLV320AIC3107 数据表中找到。

软件配置的数据路径用红色在编解码器功能方框图，图 3中标出。 与微控制器进行通信的七条线路通信可在这个图的顶部和底部找到。 到右侧，HPLOUT 和 HPROUT 的两个输出进入扬声器。

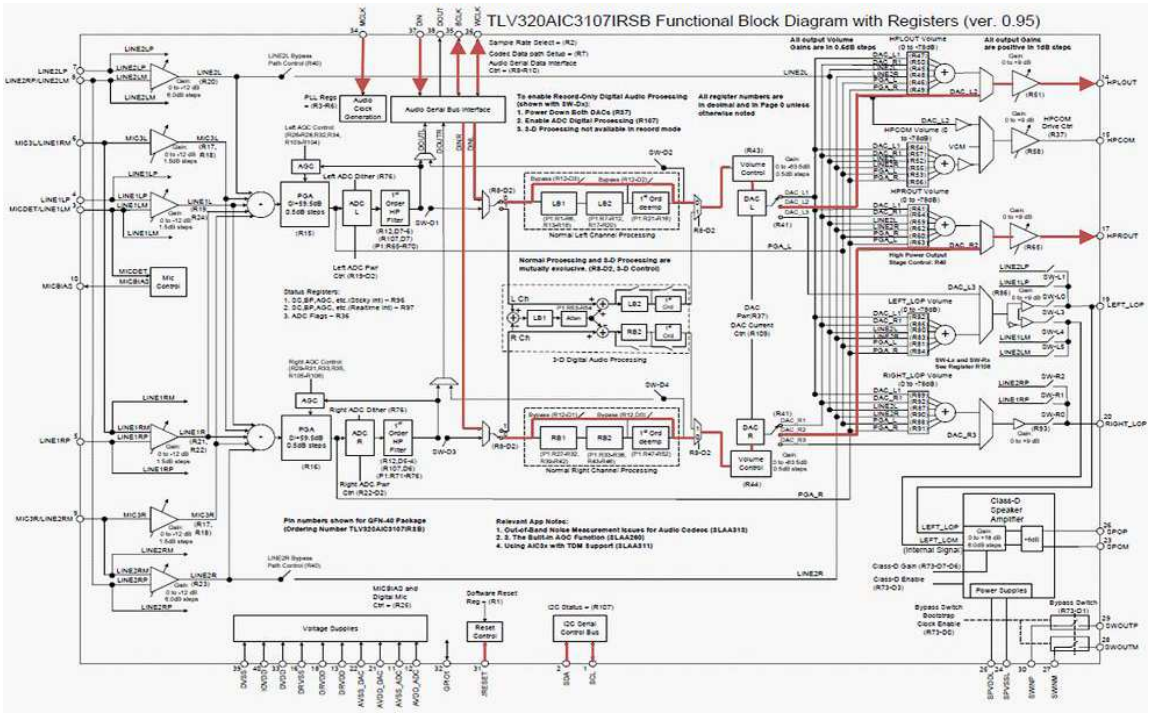


图 3. TLV320AIC3107 数据路径

编解码器初始化期间，图 3中的数据路径由写入不同的编解码器控制寄存器进行配置。 为了获得最佳效果，按照特定的顺序来设置编解码器。 Stellaris 微控制器使用 I²C 来执行这些寄存器写入。

音频驱动程序的初始化期间，dac.c 中的 DACInit() 函数被调用。 这个函数每次通过调用 DACWriteRegister() 函数来写入编解码器内的一系列寄存器。 除了在图 3中为控制路径设置 MUX，还要进行如下配置：

- PLL 分压器，用来定义编解码器采样速率 (8kHz，11.025kHz，16kHz …)
- Fsref (44.1kHz，48kHz)
- 头戴式耳机检测
- 左右输出音量
- 爆音降噪
- 体积

如果使用一个编解码器而非 TLV320AIC3107，DACInit() 函数内的寄存器写入调用必须进行相应地修改。 DACInit() 函数完全体现了寄存器的名称、寄存器编号、写入结果和写入值。 以下三个编解码器示例显示了这些写入：

```
//
// DAC Output Switching Control Register
// ----------------------
// Left DAC output selects DAC_L2 path to left high power output drivers.
// Right DAC output selects DAC_R2 path to right high power output
// drivers.
// ----------------------
// D[7:0] = 1010 0000
//
DACWriteRegister(41, 0xA0);
```

来自左侧和右侧编解码器的多个输出路径。 使用 L2 和 R2 绕过模拟音量控制和混音网络。 这个输出使用减少的功耗提供最高质的编解码器回放性能，但是只有当编解码器输出未被同时路由至多个输出驱动器以及不要求编解码器输出与其它模拟信号的混音的时候才可用。 如果应用需要混音，应该使用 L1 和 R1。 然而，对于差分扬声器，这份应用报告中的执行是足够的。 选择 DAC_L2 和 DAC_R2 来选中图 3中用红色显示的路径。 上面的示例代码正在执行一个 I^2C 传输来将值 0xA0 写入到第 41 个寄存器，*编解码器输出开关控制寄存器*。 正如 TLV320AIC3107 数据表的控制寄存器部分中所见，这个写入选择左侧的编解码器路由至 DAC_L2，而将右侧编解码器路由至 DAC_R2。 这个配置也可将每个通道设置成具有一个单独的音量控制。

Stellaris 微控制器将 MCLK 驱动为 25MHz。 数据转换器基于在部件内部使用的 Fsref 速率的概念，并且 Fsref 率通过一系列比率与实际转换采样率相关。 对于典型采样率，Fsref 为 44.1kHz 或 48kHz。 要获得所需的音频采样频率，MCLK 必须被分频以实现一个 44.1kHz 或 48kHz 的 Fsref。

```
//
// PLL Programming Register A
// ----------------------
// PLL Disabled (Enable Later)
// Q = 2
// P = 2
// ----------------------
// D[7:0] = 0001 0010
//
DACWriteRegister(3, 0x12);
```

上面的代码正在执行一个 I^2C 传输来将值 0x12 写入到第三寄存器，PLL 编程寄存器 A。当 PLL 被启用时，Fsref = (PLLCLK_IN x K x R) / (2048 x P)。 与这个时钟生成相关的额外细节可在 TLV320AIC3107 数据表的音频时钟生成部分中找到。 要获得所需的 44.1kHz 的 Fsref：Q = 2，P = 2，J = 7 并且 K = 2253。 这些值都在 PLL 编程寄存器 A-D 中设置。 上面的代码设定 Q =2 和 P = 2。 PLL 被保持禁用，直到所有 PLL 编程寄存器被正确配置。

```
//
// Codec Sample Rate Select Register
// ----------------------
// DAC Fs = Fsref/4.
// ----------------------
// D[7:0] = 0000 0110
//
DACWriteRegister(2, 0x06);
```

编解码器的采样率可被设定为 Fsref/NDAC 或 2xFsref/NDAC，其中 NDAC 为 1，1.5，2，2.5，3，3.5，4，4.5，5，5.5 或 6。例如，在 Fsref = 44.1kHz 时，通过使用 NDAC = 1 的值，采样率可被设定为 44.1kHz。 在上面的代码中，通过将值 0x06 写入到第二寄存器，*代码采样率选择寄存器*，编解码器采样频率被设定为 11.025kHz (44.1/4)。

由于将被播放的音频的采样频率在编解码器被初始化时未知，音频 API 中的 SoundSetFormat() 函数通过对特定采样频率所需的 Fsref，K，J 和 D 值进行硬编码来处理对于多采样频率的支持。 为了支持一个采样频率，必须添加一个事例来检查 ulSampleRate 变量的值。 设定 Fsref，以及 PLL，K，J 和 D 值。 启用编解码器双速率模式（如果需要的话）并设定 NDAC 值。 TLV320AIC3107 数据表可被用来确定合适值；然而，TLV320AIC3107EVM-K 图形化用户接口软件也许是最简便的方法。 使用 Clocks 标签页进入设置并点击，**Search for Ideal PLL Settings**（查找最佳 **PLL** 设置）。表 4列出了被预先编入这个软件示例的所支持的采样频率。

表 **4.** 所支持的音频格式

所支持的每采样位	所支持的通道	所支持的采样频率 **(kHz)**
16	单声道 (1)	8.000
	立体声 (2)	11.025
		16.000
		32.000
		48.000
		64.000

5 软件模式概述

双 SPI 软件示例使用多个驱动程序。 图 4中显示了一个针对 dual_spi.c 应用的简化软件堆栈。

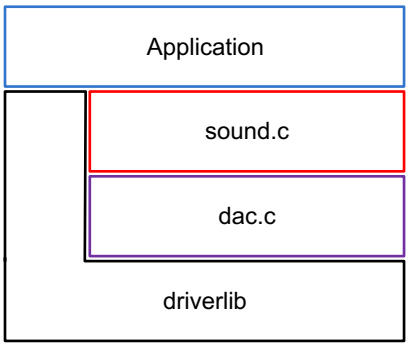


图 **4.** 双 **SPI** 应用软件堆栈

在用户级，由 dual_spi.c 主应用程序初始化音频驱动器，图形，SD 卡 I/O 和用户按钮轮询。 Dual_spi.c 取决于很多 StellarisWare® 资源，其中包括 driverlib，utils，grlib 和 第三方 fat fs。 这个应用只通过 sound.c 与音频交互。 当应用启动时，可在 SD 卡上找到一个装有 .wav 文件的列表。 然后，屏幕显示第一个音频片段的名称。 这个 GUI 的示例请参考图 5，名为 DEMO16'1.WAV 的 12 秒音频文件内的 8 秒，播放音量为 50%。

使用 GUI 来对音频片段进行如下控制：

- 按下**LEFT**或**RIGHT**按钮来循环可用的音频片段。 每个被选中显示的片段的名称。
- 通过按下**SELECT/WAKE**按钮来选择一个音频片段。
- 回放时，通过按下**SELECT/WAKE**按钮来暂停或取消暂停片段。
- 要停止片段并返回主菜单，请按下**LEFT**按钮。
- **UP**和**DOWN**按钮可相应地将音频音量增加和减少 10%。 音量的设置用屏幕右侧的垂直音量条显示。
- 回放期间，请检查屏幕底部的时间戳和水平时间条来查看音频文件进度。

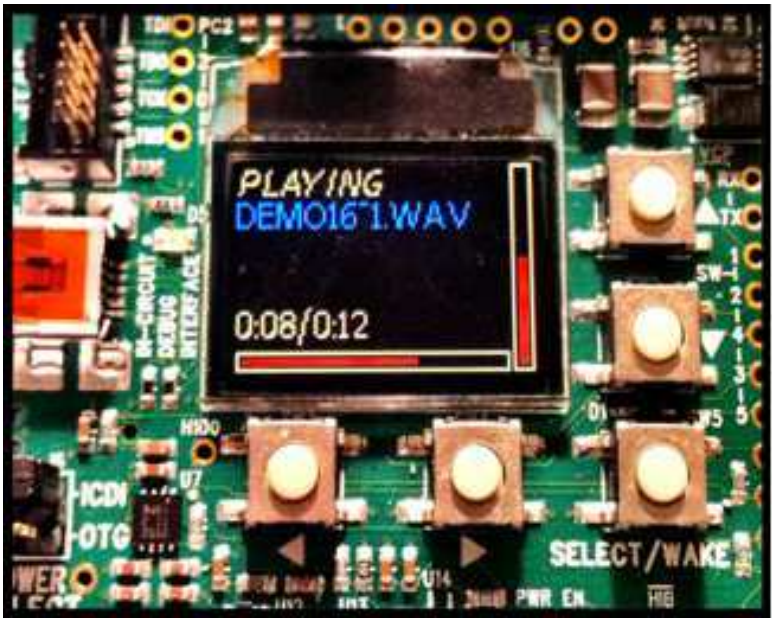


图 **5.** 双 **SPI** 软件示例 **GUI**

主用于配置所需的音频架构并通过 sound.c 内的音频 API 来开始回放。 在任何其它 API 调用前，必须调用 SoundInit() 函数。 在音频驱动器被初始化后，使用 SoundOpen() 函数来打开一个 .wav 文件。 最后，要开始已打开的音频文件的回放，请调用 SoundPlay() 函数。 使用以下附加函数来修改或接收数据反馈，其中包括：

- 设定并获得音量
- 获得一个时间串
- 获得歌曲长度
- 获得回放状态
- 获得采样率
- 调高或调低音量

此音频 API 通过 dac.c 内的编解码器 API 初始化并配置 TLV320AIC3107。 SoundInit() 函数通过调用 DACInit() 函数来初始化编解码器。 在对编解码器寄存器进行任何写入前必须调用这个函数。

6 音频 API

需要用来初始化和配置音频驱动程序的 API 包括以下三个函数。

注: 这些 API 必须按照显示的顺序进行调用。

```
void SoundInit(void);
tBoolean SoundOpen(const char *pcFileName,
                   tWaveHeader *pWaveHeader);
void SoundPlay(void);
```

SoundInit() 函数初始化音频回放所需的 Stellaris 微控制器的外设，其中包括两个 SPI 模块，一个周期定时器中断，µDMA 和所有编解码器初始化。

SoundOpen() 函数打开一个 .wav 文件并对它的头文件信息进行语法分析。 这个函数填入一个具有文件音频特性的调用程序提供的头文件结构（格式、通道数、采样率、平均字节率、每采样位数）。 在进行前，音频文件的有效性被确认。

SoundPlay() 函数开始回放由 SoundOpen() 函数打开的音频文件。 SoundPlay() 函数设定所需的标志，启用一个周期中断（基于采样频率）并针对音频数据缓冲启用下一个 uDMA 传输。 SoundPlay() 函数继续播放 SoundOpen() 函数打开的音频文件。

剩余的函数在音频驱动程序中组成可选的 API。

```
void SoundStop(void);
void SoundPause(void);
void SoundVolumeSet(unsigned long ulPercent);
void SoundVolumeDown(unsigned long ulPercent);
void SoundVolumeUp(unsigned long ulPercent);
unsigned char SoundVolumeGet(void);
unsigned long SoundGetTime(tWaveHeader *pWaveHeader, char *pcTime,
                           unsigned long ulSize);
unsigned long SoundSampleRateGet(void);
unsigned long SoundGetLength(void);
tBoolean SoundPlaybackStatus(void);
```

SoundStop() 函数将回放的状态改变为*Stopped*，这样，音频片段停止播放。 这个函数将回放位置返回为 0 并禁用所有相关的中断。

SoundPause() 函数将回放的状态改为*Paused*并暂停音频片段，直到 SoundPlay() 函数被再次调用，回放从已暂停的状态中恢复。

SoundVolume() 函数将音频音量设定为用户指定的百分比。 所提供的水平表示为 0%（静音）和 100%（满音量）之间的一个百分比，增量为 10%。

SoundVolumeDown() 函数将音量减少一个用户指定的百分比。 已调节的音量不能低于 0%。

SoundVolume() 函数将音量增加一个用户指定的百分比。 已调节的音量不会高于 100%。

SoundVolumeGet() 函数返回当前音量设置，此设置被规定为 0% 至 100% 之间的一个百分比（包括 0% 和 100%）。

SoundGetTime() 函数格式化一个文本串以表示已打开音频文件的已播放时间和总的播放时间。

SoundSampleRateGet() 函数返回已打开音频文件的采样率。

SoundGetLength() 函数返回以秒表示的已打开音频文件的总长度。

SoundPlaybackStatus() 函数返回已打开音频文件的当前回放状态：正在播放或没在播放。

7 编解码器 API

以下五个函数包含编解码器初始化和配置 API。

注: 必须首先调用 DACInit() 函数。

```
tBoolean DACInit(void);
void DACVolumeSet(unsigned long ulVolume);
void  DACClassDEn(void);
void  DACClassDDis(void);
tBoolean DACWriteRegister(unsigned char ucRegister,
                          unsigned long ulData);
```

DACInit() 函数初始化 I^2C 接口并写入适当的编解码器寄存器来建立图 3中所示的路径。 这个函数必须在编解码器模块中的任何其它 API 之前被调用。 针对 TLV320AIC3107 进行专门配置，寄存器写入的序列可被改变为与不同的编解码器进行成功对接。

DACVolumeSet() 函数根据指定的百分比来设定编解码器的音量。 这个函数使用一个查找表格来将 0-100% 刻度转换为 MUTE-0dB。

DACClassDEn() 函数在编解码器中启用 D 类放大器。

注: 这个示例不采用 D 类放大器，如图 3中所示，此放大器被绕过。

DACClassDDis() 函数在编解码器中禁用 D 类放大器。

注: 这个示例不采用 D 类放大器，如图 3中所示，此放大器被绕过。

DACWriteRegister() 函数将指定数据写入一个指定的编解码器寄存器。

8 软件示例入门

按照这些步骤来在 CCS 中打开双 SPI 音频演示并将示例载入到 Stellaris LM4F232H5QD 微控制器中。

1. 将已压缩的 dual_spi_audio.zip 文件夹移动到*../StellarisWare/boards/ek-lm4f232/*内的 LM4F 主板示例文件夹。
2. 右键点击 dual_spi_audio.zip 文件夹并选择**Extract All**（解压缩所有内容）。
3. 点击 Next > Next > Finish 来提取包含 C 源文件（此文件在5 节，*软件模式概述*中被检查）的 CCS 项目。
4. 打开 Code Composer Studio™ 并选择您希望使用的工作区。
5. 导入 dual_spi_audio CCS 项目。
6. 选择 Project > Import Existing CCS/CCE Eclipse Project。
7. 点击**Browse**，并在*../StellarisWare/boards/ek-lm4f232/*中选择已解压缩的***dual_spi_audio***文件夹。
8. 点击**Finish**。
9. 建立项目。
10. 选择 Project > Build Active Project。 除了一个警告之外，此项目被成功建立。

    请注意：必须在建立这个项目前建立 driverlib/ccs-cm4f 和 grlib/ccs-cm4f。
11. 要将软件载入到目标中，请确保 EVM 被连接，驱动程序被安装。 选择 Target > Launch TI Debugger。

9 CPU 使用率

这个部分描述了 CPU 使用率计算的过程和结果。 针对每个采样频率和相应的 CPU 使用百分比来显示结果。

注: 在计算中，负责在回放期间更新图形的定时器中断处理程序被省略以移除不必要的开销。

要计算 CPU 的使用率，当在主循环中空闲时，此应用被重复置于睡眠状态。 当所有外设睡眠模式被启用时，一个中断可在中断处理程序运行期间将微控制器从睡眠中唤醒，然后返回睡眠模式。 每周期 (T_p) 时间和这个周期内所有中断持续时间总和 (T_i) 之间的比率，生成 CPU 被使用带宽的百分比，在这里，*N*是每周期耗时 T_i的中断数量的平均数，这是因为，某些周期中断也许会被唤醒而不进行任何操作。

$[(N \times T_i) / T_p] \times 100$

为了捕获睡眠和中断时间的持续长度，GPIO 被切换并使用一个示波器进行查看。 有一个专用 GPIO 来查看睡眠时间并在调用睡眠函数前被切换为高电平。 然后，同一个 GPIO 在任何中断处理程序启动前被切换为低电平。 一个单独的 GPIO 被用于每个中断，然后在进入和退出中断处理程序时被切换。 下面是一个此操作的简单伪代码示例：

```
INTERRUPTHANDLER()
{
     // WRITE GPIO A LOW.
     // WRITE GPIO B HIGH.
     ...
     // WRITE GPIO B LOW.
}

MAIN()
{
     SOUNDINIT();
     SOUNDOPEN(...);
     SOUNDPLAY();
     WHILE(1)
     {
          // WRITE GPIO A HIGH.
          SYSCTLSLEEP();
     }
}
     }
```

针对一个 16 位，8kHz 立体声音频采样的中断序列由图 6中显示的波形进行描述。 波形 1（黄色）显示了微控制器进入睡眠模式的频率和持续时间。 当为高电平时，微控制器处于睡眠模式。 低脉冲表示微控制器在中断例程的持续时间内唤醒。 波形 2（蓝色）显示了 sound.c 中用来定期调用 SoundPlayContinue() 函数的 Timer2AIntHandler() 函数的频率和持续时间。 当脉冲为高电平时，中断被激活。 这个中断被建立以在需要时重新填充缓冲器并执行任何需要的看管操作来保持编解码器的音频数据供给。 这个中断没 55ms 发生一次一适应 8kHz 采样频率。 如图 6所示，这个中断在两个垂直光标（-126ms 至 318ms，一个 444ms 窗口）内发生了 8 次。 当第一个中断发生时（因为太小，在这里无法看到），缓冲器无需处理，这意味着中断无需操作并立即返回睡眠状态。 波形 3（粉色）显示了 SoundIntHandler() 函数在由于 SPI 外设中断而被处理器调用时（因为太小，在这里无法看到）的频率和持续时间。 由于这些中断的极短的持续时间，这部分时间被排除在计算之外。

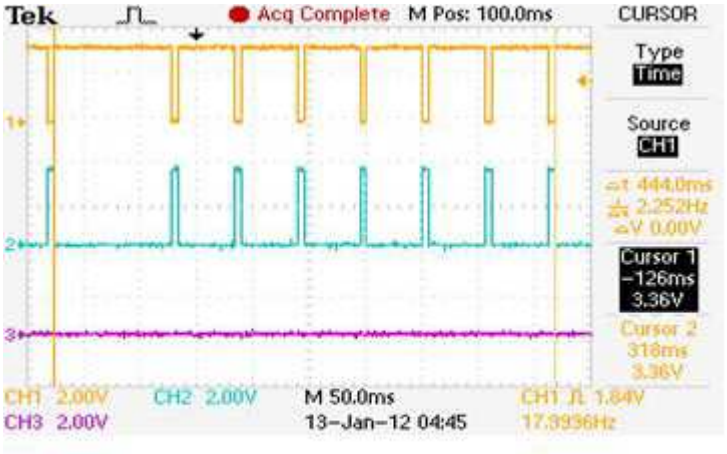


图 6. 8kHz 中断波形

如垂直光标测量的那样，这个周期的持续时间为 444ms。 发现每个定时器中断的持续时间为 5.2ms。 在每个周期中，这些中断中的 8 个中断发生，其中一个需要很短的时间。 因此，CPU 使用率计算如下：

[(7 × 5.2) / 444] × 100 = 8.2%

针对所有采样频率使用这个方法，可确定相应的 CPU 使用率。 请注意，当微控制器运行频率为 50MHz，在 16 位立体声音频回放期间，这些点被采用。 CPU 使用率与采样频率线性叠加，正如从 SD 卡的数据检索增加。 CPU 使用率的绝大部分用来从 SD 卡中读取数据。

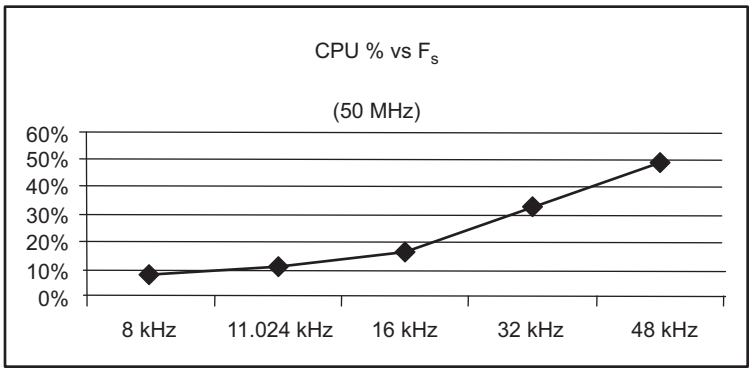
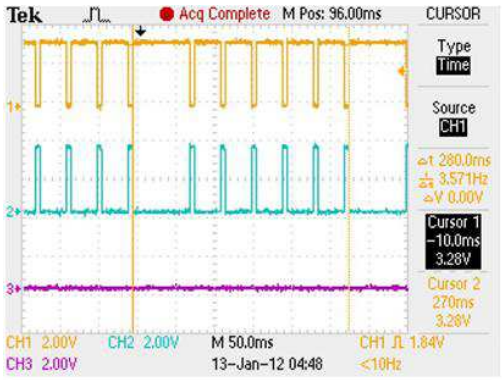
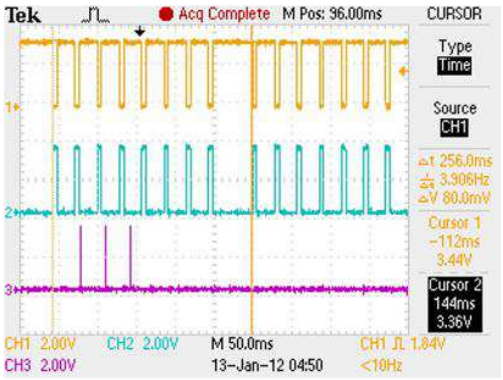


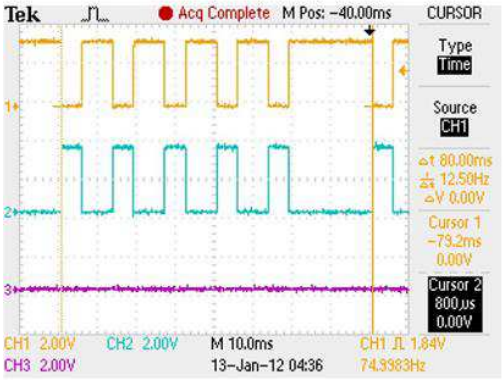
图 **7.** 基于音频采样频率的 **CPU** 使用率图



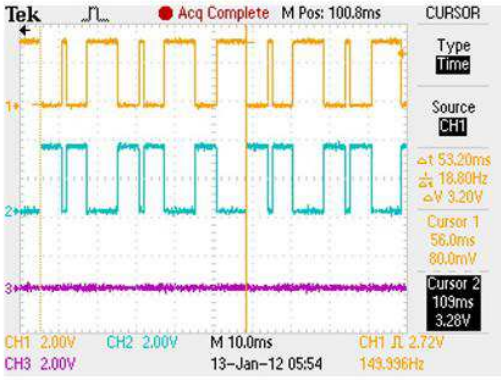
11.025-kHz Interrupt Waveforms



16-kHz Interrupt Waveforms



32-kHz Interrupt Waveforms



48-kHz Interrupt Waveforms

图 **8.** 中断波形

10 修改

双 SPI 软件示例的用途是引入音频 API 的使用和功能。 dual_spi.c 驱动程序只代表音频 API 的简单使用，并作为一个参考。 需要对音频 API 进行修改以满足最终用户应用的需要，是否是一个 Stellaris LM4F232 I^2S 替代器件，或者是到不同 Stellaris 微控制器的附加 I^2S 外设。 这个部分讨论了可由用户添加或修改的几个特性，以及进行这些修改所需的步骤。

10.1 添加针对额外采样频率的支持

音频 API 支持在缺省情况下支持六个不同的采样频率：8，11.025，16，32，48 和 64kHz。 如果需要一个额外的采样频率，需要在 sound.c 内编辑 SoundSetFormat() 函数。 缺省情况下，一个音频文件的采样频率被读取并被作为 ulSampleRate 保存在 pWaveHeader 中。 然后，这个值被传递到 SoundSetFormat() 函数中。 一个 if 语句被用来选择当前的采样率，并将所需的数据写入编解码器寄存器。 写入这些寄存器的更多细节请见4 节，*软件编解码器设置*。

例如，如果需要支持 22.05kHz 的采样频率，在这情况下添加一个额外的 else-if 块。 为了支持 22.05kHz，将编解码器 Fsref 设定为 44.1kHz 并按比例缩减 2 倍。使用图 9中所示的 TLV320AIC3107EVM 软件评估工具，所需的 Fsref 和 PLLCLK_IN 可被配置成计算最佳的 K，J 和 D 值。 将这些值与值为 2 的 NDAC 一同使用，编解码器的采样率达到 22050。

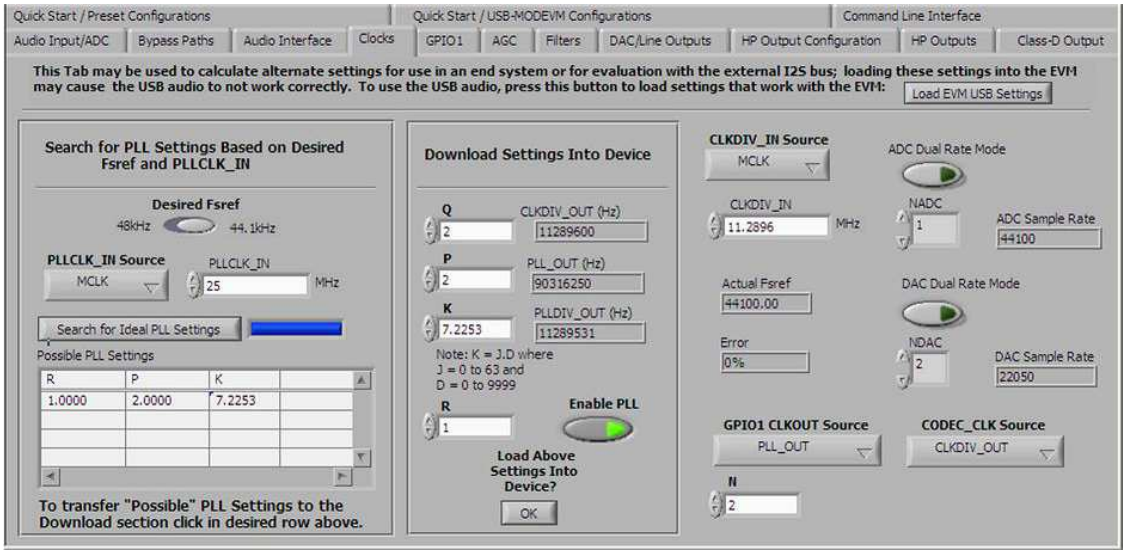


图 **9. TLV320AIC3107EVM-K GUI** 软件

10.2 增加 8 或 32 位音频支持

缺省情况下，音频 API 支持最直观的 16 位音频，这是因为 SPI 传输 FIFO 为 16 位宽。 要实现 8 位或 32 位音频支持，需要对音频 API 进行多次修改。

在 sound.c 中的 SoundSetFormat() 函数的开头，SPI 协议、操作模式、位速率和数据宽度被配置。 将 SSIConfigSetExpClk() 函数的 ulDataWidth 参数从 16 改为 8 来支持 8 位运行。 这个参数定义了数据传输宽度，可以是一个 4 至 16 之间的值（包含 4 和 16）。 对于一个 32 位模式，16 位宽度已足够。 对于立体声 32 位，左右通道可被分成偶数和奇数数据块。 对于单声道 32 位，每个数据帧被一分为二。

在 SoundSetFormat() 函数顶部针对单声道和立体声格式完成 SPI 配置后，需要在这个函数接近底部的位置进行其它修改，在这里 DMA 设置针对所使用的通道进行配置。 ulDMASetting 是五个值的逻辑与操作后的结果：源地址增量、目标地址增量、仲裁尺寸和使用突发标志。 为了包含针对单声道的 8 位功能性，请进行如下设置：

```
//
// The transfer size is 8 bits from the TX buffer to the TX FIFO.
//
ulDMASetting = UDMA_SIZE_8 | UDMA_SRC_INC_8 |
               UDMA_DST_INC_NONE | UDMA_ARB_2;
```

要实现 8 位立体声，DMA 可被配置成保存左右数据块，从而得到 16 位尺寸：

```
//
// The transfer size is 16 bits(stereo 8 bits) from the TX buffer
// to the TX FIFO.
//
ulDMASetting = UDMA_SIZE_16 | UDMA_SRC_INC_16 |
               UDMA_DST_INC_NONE | UDMA_ARB_2;
```

在针对 µDMA 通道控制结构的传输参数被设定的地方，需要进行最终的修改。 在启用 µDMA 通道前，使用 sound.c 内的 SoundBufferPlay() 函数内的 uDMAChannelTransferSet() 函数来完成这些修改。 当前执行针对 16 位数据尺寸（具有 32 的源地址增量）对 uDMA 通道进行配置，从而在调用 uDMAChannelTransferSet() 函数时可将左右数据分开。 头 16 位（左通道）被传输到第一个 SPI 通道：

```
ROM_uDMAChannelTransferSet(ulChannel,
                UDMA_MODE_PINGPONG,
                (unsigned short *)g_sOutBuffers[g_ulPlaying].pulData,
                (void *)(I2S_CHAN0_BASE | SSI_O_DR),
                g_sOutBuffers[g_ulPlaying].ulSize);
```

下一个 16 位（右通道）被传输到第二个 SPI 通道：

```
ROM_uDMAChannelTransferSet(ulChannel2,
                UDMA_MODE_PINGPONG,
                (unsigned short *)g_sOutBuffers[g_ulPlaying].pulData+1,
                (void *)(I2S_CHAN1_BASE | SSI_O_DR),
                g_sOutBuffers[g_ulPlaying].ulSize );
```

请注意第三和第四个自变量间的差异。 由之前源增量尺寸设定的 32 位数据块被分成 16 个块并被发出各自的 SPI 基地址。

10.3 使用编解码器板载 D 类放大器

TLV320AIC3107 配备有一个本音频 API 不使用的板载 D 类扬声器放大器。 如果需要更高的增益，将编解码器重新配置成在数据路径中包含这个放大器，从而生成高达 +18dB 的增益。

这个示例使用图 3中所示的 TLV320AIC3107 功能方框图中的数据路径。 控制 DAC L 和 DAC R 的寄存器 41，必须被改变为选择 DAC_L3 和 DAC_R3。 然后，必须通过寄存器 73 来配置和启用 D 类放大器。 这些寄存器的技术规格请见 TLV320AIC3107 数据表。 在 DACInit() 函数中对驱动程序，dac.c 进行这些修改。 在4 节，*软件编解码器设置*中解释了与写入编解码器寄存器的函数相关的附加信息。

10.4 使用一个 TLV320AIC3107 之外的编解码器

在双 SPI 执行中使用的音频 API 针对 TLV320AIC3107 而进行了专门修改。 然而，在本文档中显示的硬件和软件配置之后，这些 API 可被转换成在不同的编解码器上正常工作。

针对编解码器所需的硬件接口和针对外部 MCLK，外部复位，I^2C 和 I^2S 的所需支持请见图 1。 硬件接口简单易用并有可能只包含几个轻微改变。 大部分修改是对 dac.c 驱动程序的修改。 这个驱动程序建立必要的架构来将值写入编解码器寄存器并从中读取数据。 DACInit() 函数配置这些外设，然后针对 TLV320AIC3107 进行多个 DACWriteRegister() 调用。 每个写入被完全注释为写入编解码器的说明、功能性和二进制数据。 如果迁移到不同的编解码器，相应的功能性必须被重新执行。

除 DACInit() 函数之外，有几个寄存器存在于音频 API 中（例如，当根据音频文件设置采样频率时）。 由于编解码器的 MCLK 的频率为 25MHz，SoundSetFormat() 函数对必要的 PLL，Fsref 和需要用来改变编解码器采样频率的 NDAC 值进行硬编码，这样音频在所需的速率上从 Stellaris FIFO 中采样。 当不使用 TLV320AIC3107，每个 DACWriteRegister() 调用必须被修改为相等的寄存器或数据来实现同样的功能性。 特定寄存器信息请见替代编解码器数据表和 TLV320AIC3107 数据表。

10.5 改变 CPU 频率或编解码器主时钟

Stellaris 微控制器为 TLV320AIC3107 生成主时钟 (MCLK)，一个由 16 位定时器驱动的 25MHz 时钟信号。 这个定时器由一个单一负载值进行简单配置，这意味着此定时器将系统时钟一分为二。 由于编解码器计时（PLL，Fsref，NDAC 等）基于这个预先设定的 25MHz MCLK 进行配置，主系统时钟的调整使 MCLK 发生偏离并因此影响了编解码器采样率。 例如，如果 Stellaris 系统时钟从 50MHz 被改为 40MHz，新的 MCLK 将为 40/2=20MHz。 由于这个频率更慢，编解码器以更低的速率从 Stellaris SPI FIFO 中采样数据，从而导致更慢的数据输出。

恢复回放速率所需的修改在 SoundSetFormat() 函数内。 针对每个采样频率的 if 语句内的每次编解码器寄存器写入必须针对新的 MCLK 进行评估。 通过使用 TLV320AIC3107 EVM 软件评估工具，PLLCLK_IN 字段可被改变为新的 MCLK 值。 新计算得出的 PLL 设置取代被写入 SoundSetFormat() 函数内的值。 当改变 CPU 频率或编解码器系统时钟时，这个方法可被用来适应已改变的 MCLK。

10.6 使用一个 LM4F232 以外的微控制器

虽然这个示例的主要用途是使得 Stellaris LM4F 系列微控制器具有音频功能性，但是这个示例也可被用来在不同的平台上引入额外的 I^2S 外设。

在这个情况下，在 dac.c 和 sound.c 内的硬件映射的定义必须针对替代微控制器进行修改。 驱动程序 dac.c 包含针对用来与外部编解码器通信的 I^2C 和复位引脚的 *#defines*。 驱动程序包含针对用在音频驱动器的所有伪 I^2S 引脚的*#defines*。 这些定义必须被改变，将替代的微控制器可用引脚和外设与现有的一致。 针对 Stellaris 微控制器的端口和引脚定义可在 StellarisWare DriverLib 的 pin_map.h 内找到。

10.7 使用闪存而非 SD 卡来存储音频

双 SPI 音频 API 从一个板载 SD 卡中读取音频 API。 这些读取时间组成了 CPU 使用的绝大部分并可通过将音频保存在闪存存储器中来减少此类时间。 然而，这个方法占用了大量的存储器，不建议将这个方法应用于长度大于几秒的音频片段。

可使用 StellarisWare 中的 makefsutility 来创建包含格式为 .wav 的 8 位音频片段的 C 数组。 这个数组可被保存在一个头文件中并由音频 API 的 SoundOpen() 函数而非一个文件项目引用。 请参见 Stellaris EVALBOT 演示示例。 要了解所做的修正，请见 StellarisWare/boards/ek-evalbot/sound_demo/sound_demo.c 中的源代码。

10.8 播放单声道音频与播放立体声音频间的关系

双 SPI 音频示例支持单声道和立体声格式，并且理解音频驱动程序是如何区分它们的十分重要。

sound.c 音频驱动程序支持采用立体声和单声道格式的多个 16 位采样频率。 这些属性从 .wav 文件中进行语法分析并被装入 SoundOpen() 函数内。 在这些信息被检索后，自动使用针对特定格式的配置。 从用户的角度来说，从 SD 卡中打开一个 16 位，单声道 8kHz .wav 文件与打开一个 16 位立体声 64kHz .wav 文件没有什么不同。 多种格式的音频文件可进行背靠背播放。 GoldWare freeware 软件可被用来输出和配置 PCM，N 位，NkHz 音频文件。

11 结论

双 SPI 音频示例为在不提供一个 I²S 模块的 Stellaris 微控制器上开发包含音频的应用提供了一个基础。 所提供的音频 API 可通过 I²S 实现将一个 Stellaris LM4F 微控制器对接到一个外部编解码器的简便执行。 音频 API 提供一个稳健耐用、易于操作的开发平台来满足最终用户的 LM4F 音频应用的需要。

12 参考

以下相关文档和软件可从Stellaris 网站www.ti.com/stellaris内下载：

- Stellaris LM4F232H5QD 数据表，文献编号SPMS319
- 针对本应用报告的相关文件，其中包括用于音频 API 和 dual_spi 示例的源代码：相关文件
- Stellaris LM4F232H5QD 微控制器产品文件夹：LM4F232H5QD

以下资源包含与本应用报告中使用的外部编解码器相关的额外信息：

- TLV320AIC3107 数据表，文献编号SLAU261
- TLV320AIC3107EVM-K GUI 软件：SLAC250
- TLV320AIC3107 产品文件夹：TLV320AIC3107

重要声明

德州仪器(TI) 及其下属子公司有权根据 JESD46 最新标准, 对所提供的产品和服务进行更正、修改、增强、改进或其它更改， 并有权根据 JESD48 最新标准中止提供任何产品和服务。客户在下订单前应获取最新的相关信息, 并验证这些信息是否完整且是最新的。所有产品的销售都遵循在订单确认时所提供的TI 销售条款与条件。

TI 保证其所销售的组件的性能符合产品销售时 TI 半导体产品销售条件与条款的适用规范。仅在 TI 保证的范围内，且 TI 认为 有必要时才会使用测试或其它质量控制技术。除非适用法律做出了硬性规定，否则没有必要对每种组件的所有参数进行测试。

TI 对应用帮助或客户产品设计不承担任何义务。客户应对其使用 TI 组件的产品和应用自行负责。为尽量减小与客户产品和应 用相关的风险，客户应提供充分的设计与操作安全措施。

TI 不对任何 TI 专利权、版权、屏蔽作品权或其它与使用了 TI 组件或服务的组合设备、机器或流程相关的 TI 知识产权中授予 的直接或隐含权限作出任何保证或解释。TI 所发布的与第三方产品或服务有关的信息，不能构成从 TI 获得使用这些产品或服 务的许可、授权、或认可。使用此类信息可能需要获得第三方的专利权或其它知识产权方面的许可，或是 TI 的专利权或其它 知识产权方面的许可。

对于 TI 的产品手册或数据表中 TI 信息的重要部分，仅在没有对内容进行任何篡改且带有相关授权、条件、限制和声明的情况 下才允许进行复制。TI 对此类篡改过的文件不承担任何责任或义务。复制第三方的信息可能需要服从额外的限制条件。

在转售 TI 组件或服务时，如果对该组件或服务参数的陈述与 TI 标明的参数相比存在差异或虚假成分，则会失去相关 TI 组件 或服务的所有明示或暗示授权，且这是不正当的、欺诈性商业行为。TI 对任何此类虚假陈述均不承担任何责任或义务。

客户认可并同意，尽管任何应用相关信息或支持仍可能由 TI 提供，但他们将独力负责满足与其产品及在其应用中使用 TI 产品 相关的所有法律、法规和安全相关要求。客户声明并同意，他们具备制定与实施安全措施所需的全部专业技术和知识，可预见 故障的危险后果、监测故障及其后果、降低有可能造成人身伤害的故障的发生机率并采取适当的补救措施。客户将全额赔偿因 在此类安全关键应用中使用任何 TI 组件而对 TI 及其代理造成的任何损失。

在某些场合中，为了推进安全相关应用有可能对 TI 组件进行特别的促销。TI 的目标是利用此类组件帮助客户设计和创立其特 有的可满足适用的功能安全性标准和要求的终端产品解决方案。尽管如此，此类组件仍然服从这些条款。

TI 组件未获得用于 FDA Class III（或类似的生命攸关医疗设备）的授权许可，除非各方授权官员已经达成了专门管控此类使 用的特别协议。

只有那些 TI 特别注明属于军用等级或“增强型塑料”的 TI 组件才是设计或专门用于军事/航空应用或环境的。购买者认可并同 意，对并非指定面向军事或航空航天用途的 TI 组件进行军事或航空航天方面的应用，其风险由客户单独承担，并且由客户独 力负责满足与此类使用相关的所有法律和法规要求。

TI 已明确指定符合 ISO/TS16949 要求的产品，这些产品主要用于汽车。在任何情况下，因使用非指定产品而无法达到 ISO/TS16949 要求，TI不承担任何责任。

	产品		应用
数字音频	www.ti.com.cn/audio	通信与电信	www.ti.com.cn/telecom
放大器和线性器件	www.ti.com.cn/amplifiers	计算机及周边	www.ti.com.cn/computer
数据转换器	www.ti.com.cn/dataconverters	消费电子	www.ti.com/consumer-apps
DLP® 产品	www.dlp.com	能源	www.ti.com/energy
DSP - 数字信号处理器	www.ti.com.cn/dsp	工业应用	www.ti.com.cn/industrial
时钟和计时器	www.ti.com.cn/clockandtimers	医疗电子	www.ti.com.cn/medical
接口	www.ti.com.cn/interface	安防应用	www.ti.com.cn/security
逻辑	www.ti.com.cn/logic	汽车电子	www.ti.com.cn/automotive
电源管理	www.ti.com.cn/power	视频和影像	www.ti.com.cn/video
微控制器 (MCU)	www.ti.com.cn/microcontrollers		
RFID 系统	www.ti.com.cn/rfidsys		
OMAP应用处理器	www.ti.com/omap		
无线连通性	www.ti.com.cn/wirelessconnectivity	德州仪器在线技术支持社区	www.deyisupport.com

邮寄地址： 上海市浦东新区世纪大道 1568 号，中建大厦 32 楼 邮政编码： 200122