## 10.1 基于PDM-PCM的音频数据采集与存储

### 10.1.1概述

随着数字信号处理技术的发展，数字音频技术在电子产品中的应用越来越多。数字麦克风以其抗干扰能力强、低成本、体积小容易集成等优点广泛应用于便携式设备。在数字音频接口中，采用脉冲密度调制（PDM）接口的驻极体电容式（ECM，Electret Condenser Microphone）麦克风和微型机电系统（MEMS，Micro Electro Mechanical System）数字麦克风已被广泛应用。典型的应用如手机和便携式笔记本电脑中的数字麦克风。PSoC6 MCU内部集成了PDM-PCM模块，并提供底层控制程序库，可以实现PDM位流数据采集，以及PDM位流数据到PCM数据格式转换，并可以使用I2S接口连接外部音频处理芯片，通过耳机或者扬声器实现音频播放。该模块支持各种采样率和数据格式选项。可以应用但不局限于如下场景：

* 可穿戴设备
* 物联网/物联网系统
* 虚拟现实游戏系统
* 便携式设备
* 智能汽车
* 指挥设备中的语音
* 智能家居系统
* 录音机

**1.实验目的**

* 通过案例了解PSoC6 MCU的数字音频子系统的应用和实现原理。
* 熟悉PDM数据格式和硬件接口。
* 熟悉PCM编码原理和I2S接口。
* 熟悉MEMS数字麦克风的工作原理。
* 熟练使用HAL和PDL编写程序，实现PDM数据获取、PDM-PCM转换及串口打印。
* 学习使用CMD控制台执行Python脚本文件实现从串行终端接收数据，并将数据流转换为.wav文件保存到当前目录。

**2.预备知识**

（1）PDM和PCM原理。

（2）数字麦克风原理。

（3）PSoC6 MCU音频子系统PDL和HAL API的使用。

**3.实验原理**

（1）硬件配置

* Infineon CY8CPROTO-062-4343W PSOC6开发平台。
* 串口终端。

（2）软件配置

* PC端软件：串行终端软件，用于输出PCM数据流。
* Python脚本文件：用于将串口接收到的数据流转为.wav文件，并保存到脚本文件当前目录。
* 嵌入式端软件：ModusToolbox。

（3）实验原理

这个案例主要使用PSoC6 MCU中的脉冲密度调制-脉冲编码调制（PDM-PCM）硬件模块控制数字麦克风实现音频数据采集。并利用开发平台上的用户按键、LED和UART接口实现以下功能：

* 测量声音强度（音量）。
* 使用用户按键设置噪声阈值和启动一次录音操作。
  + KEY ON时设置阈值。以此捕获环境噪声。
  + KEY OFF时读取下一帧数据。并在此期间点亮LED灯。
* 麦克风有数据请求处理时，检测音量是否大于阈值，超过阈值则通过UART向串行终端发送PDM-PCM转换后的数据。并在发送数据期间打开LED灯。
* 没有按键按下，且没有麦克风数据需要处理时，系统进入低功耗，关闭所有设备。
* 使用脚本文件将PC端接收到数据流转为.wav格式的音频文件，保存到脚本文件当前目录。

可以使用音频播放软件验证收集到的音频信号是否正确，同时采样到的.wav格式文件也可以作为机器学习的输入数据，通过构建机器学习模型实现语音控制系统或语音识别系统。

实验原理框图10.1.1所示。

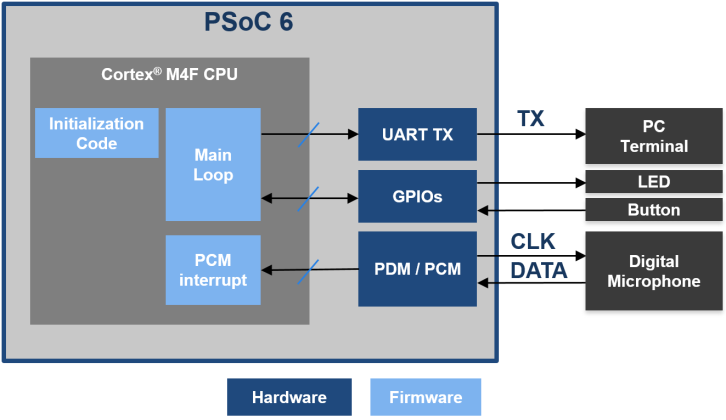


图10.1.1 音频数据采集与存储原理框图

### 10.1.2基于PDM-PCM的音频数据采集与处理设计与实现

**1.硬件设计**

本案例使用的外设主要有LED、用户按键和数字麦克风，硬件设计原理图如图10.1.2～10.1.4所示。

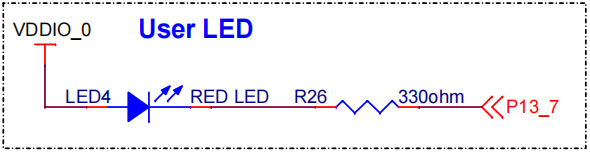


图10.1.2. LED外部连接电路

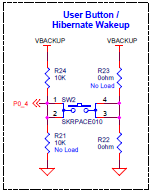


图10.1.3. User Button外部连接电路

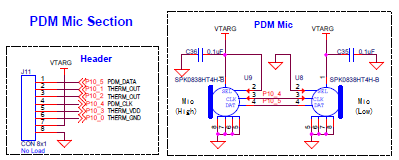


图10.1.4. PDM数字麦克风外部连接电路

数字麦克风通常由5个引脚组成，分别是电源（VDD），地（GND），时钟（CLK），数据（DAT）和通道选择（SRL）。采用MEMS技术的数字麦克风接口芯片SPK0838HT4H-B的连接情况如图10.1.4所示。数字麦克风的电源连接系统供电，外部时钟信号频率范围为1.024～3.074MHz。具有低功耗功能，其工作模式迁移图如图10.1.5所示。

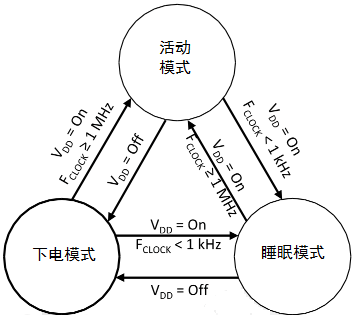


图10.1.5数字麦克风工作模式迁移图

PDM接口可以挂接两个数字麦克，两个麦克风共享时钟和数据线，由PSoC6 MCU的音频子系统提供时钟信号。图10.1.4中左边的麦克风（Mic（High））通道选择引脚（SRL）接高电平，右边的麦克风（Mic（Low））的通道选择引脚（SRL）接低电平。分别在时钟的上升沿和下降沿触发选择Source 1/2作为数据输入。采样时序图如图10.1.6所示。在时钟的上升沿采样（Mic（High））声道数据，在时钟下降沿采样（Mic（Low））声道数据。

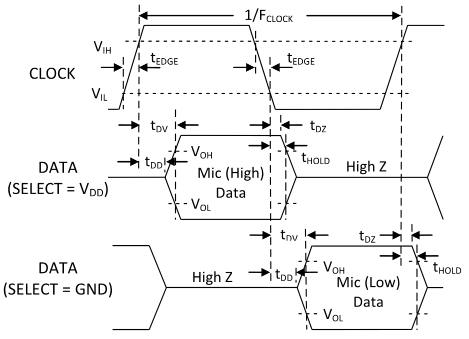


图10.1.6数字麦克风采样时序图

**2. PSoC6 MCU内部资源配置**

ModusToolbox将应用程序的配置设置存储在design.modus文件中,使用此文件可以生成应用程序的设备配置。该文件存储在应用程序的生成源文件夹中。

1. 引脚配置

案例中LED、按键及PDM-PCM模块使用的引脚配置如下表10.1.1所示。

表10.1.1 引脚配置

|  |  |  |
| --- | --- | --- |
| 端口 | 引脚 | 引脚功能名称 |
| Port0 | P0[4] | CYBSP\_USER\_BTN |
| Port10 | P10[4] | CYBSP\_PDM\_CLK |
| Port10 | P10[5] | CYBSP\_PDM\_DATA |
| Port13 | P13[7] | CYBSP\_USER\_LED |

（2）PDM-PCM配置

使用Device Configurator配置相关引脚。在ModusToolbox中打开Device Configurator工具，选择“CY8C624ABZI-S2D44”选项卡下面的“Peripherals”选项卡。勾选“Digital”下面的“PDM-PCM Converter 0”，然后，在右面面板中设置“Channels”为“Mono L”，“Left Ghannel Gain”设置为“+10.5db”；“out data->Out Data Word Length,In Bits”设置为16；然后进行时钟和PDM引脚设置。如图10.1.7所示。

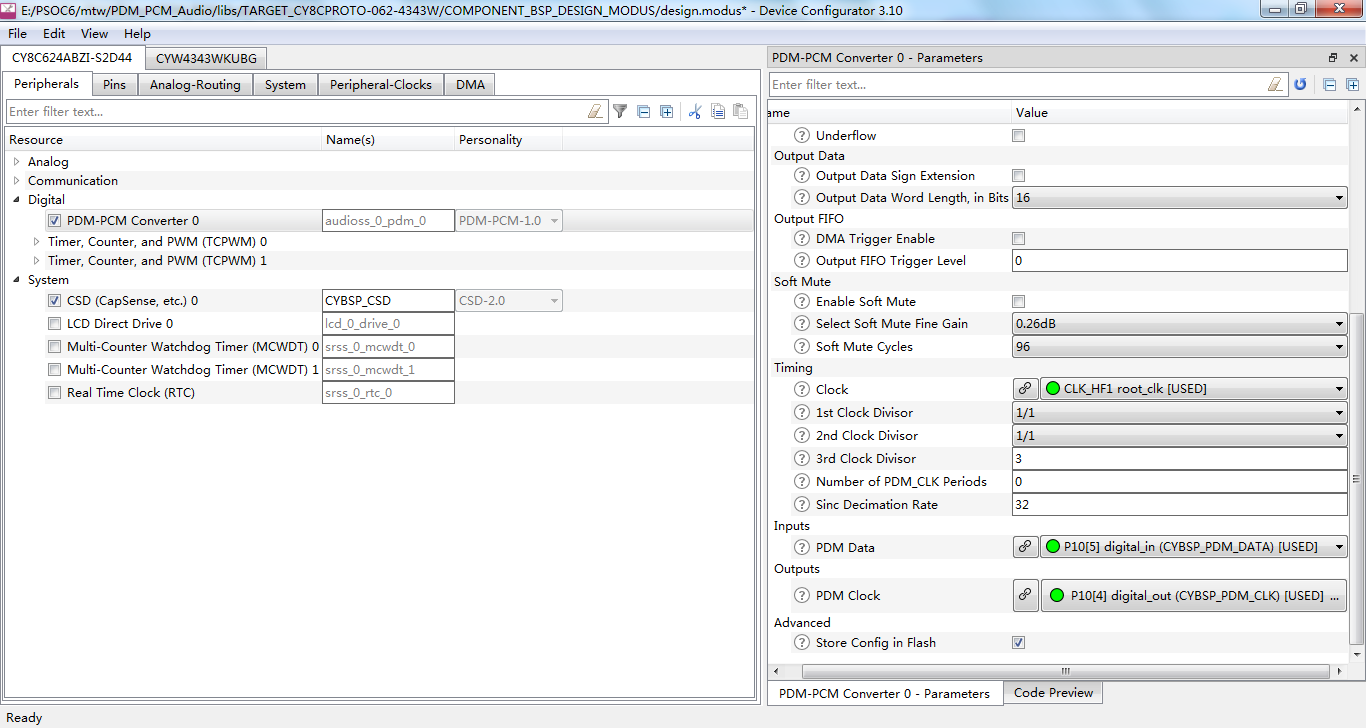


图10.1.7 PDM-PCM模块设置

接下来，选择“CY8C624ABZI-S2D44”选项卡下面的“pins”选项卡。勾选P10[4]作为PDM时钟输出引脚，在右面的General-Drive Mode中设置驱动模式为“Strong Drive,Input buffer off”；勾选P10[5]引脚作为PDM数据输入，在右面的General-Drive Mode中设置驱动模式为“Digital High-Z,Input buffer on”。

**3.基于PDM-PCM的音频数据采集与存储软件设计**

要实现前面实验原理章节中的功能要求，系统中使用了两个中断服务，作为前台，用于触发事件，main函数作为后台处理相关操作：

一个是PDM中断：在这个案例中，数字麦克风将捕获的任何音频转换为数字信号（PDM）。PSoC 6 MCU设备将该数字信号转换为量化16位值（PCM）。当有足够的数据需要处理时（至少128个样本。程序设置为32768/2个样本，可根据需要调整样本量），就会触发中断。在这个中断回调中设置数据处理标志pdm\_pcm\_flag=true，以便在main函数中通过检测pdm\_pcm\_flag标志来处理PCM数据。

另一个是按键中断：将样本的绝对值相加可以获得声音的整体响度，该值与音量成正比。可以根据音量设置阈值以消除环境噪音。因此本案例可以在噪声环境中运行。通过按下开发平台上的用户按键，可以根据当前音量设置新的噪声阈值。按键被按下（即KEY ON状态发生）时，触发按键中断，在中断回调中设置按键状态标志button\_flag = true，以便main函数中根据按键状态设置噪声阈值和检测按键的松开动作。因此在按键被按下时，应移除任何声源，并避免靠近麦克风说话。PDM-PCM模块配置为16 ksps的采样率。理想情况下，捕获的音频波应该在零附近振荡。PDM-PCM硬件集成了高通滤波器。该滤波器可以滤除转换器的低频噪声。

因此，main函数中要处理的事务可以归纳为以下几点：

* 初始化相关硬件并注册中断回调。
* 检测是否有PDM-PCM数据需要处理：通过pdm\_pcm\_flag标志检测。如果有，则计算音量；音量大于噪声阈值时将PDM-PCM数据通过串口传送到串行终端。在此期间点亮LED灯。数据传送完成后灭灯。
* 检测是否有按键发生：通过按键状态标志button\_flag检测。如果有，则根据当前音量设置新的阈值，并检测按键松开的状态。在松开按键（即KEY OFF状态）时，设置读取下一帧PDM-PCM操作。

除main函数外的主要函数列表如下表所示。

表10.1.2 函数列表

|  |  |
| --- | --- |
| 函数名 | 功能 |
| cyhal\_gpio\_write() | 写引脚状态，用于LED灯的亮灭控制。 |
| cyhal\_uart\_putc() | 通过串口发送一帧UART数据。 |
| read\_switch\_status() | 通过检测用户按键引脚的高低电平检测KEY OFF状态。这个函数只有在按键中断发生后，即有按键按下时执行一次。获取到KEY OFF状态后，启动下一次PDM-PCM操作。 |
| cyhal\_pdm\_pcm\_read\_async() | 读下一帧PDM-PCM数据 |
| cyhal\_system\_sleep() | 系统进入睡眠模式。在没有任何事件发生时，执行此函数，系统进入低功耗。 |
| button\_isr\_handler() | 按键中断的回调函数。在回调中设置button\_flag = true，并产生新的噪声阈值。 |
| pdm\_pcm\_isr\_handler() | PDM-PCM模块中断回调。在回调中设置pdm\_pcm\_flag=true，预示有新的PDM-PCM数据需要处理。 |

主函数的参考代码如下所示。

|  |
| --- |
| /\*宏定义\*/  /\* 定义一帧信息中有多少个样本 \*/  #define FRAME\_SIZE (32768u) // (1024)  /\* 噪声阈值滞后 \*/  #define THRESHOLD\_HYSTERESIS 3u  /\* 用于噪声阈值计算的音量比率 \*/  #define VOLUME\_RATIO (2\*FRAME\_SIZE)  /\* 采样率. 典型值: 8/16/22.05/32/44.1/48kHz \*/  #define SAMPLE\_RATE\_HZ 16000u// 8000u  /\* PDM/PCM 模块的抽取率. 典型值64 \*/  #define DECIMATION\_RATE 64u  /\* 音频子系统时钟. 典型值依赖于所采用的采样率:  - 8/16/48kHz : 24.576 MHz  - 22.05/44.1kHz : 22.579 MHz \*/  #define AUDIO\_SYS\_CLOCK\_HZ 24576000u  /\* PDM/PCM 引脚 \*/  #define PDM\_DATA P10\_5  #define PDM\_CLK P10\_4  #define SWITCH\_DEBOUNCE\_CHECK\_UNIT (1u)  /\* 按键消抖用 \*/  #define SWITCH\_DEBOUNCE\_MAX\_PERIOD\_UNITS (80u)  #define AMBIENT\_TEMPERATURE\_C (20)  #define SPI\_BAUD\_RATE\_HZ (20000000)  /\*全局变量\*/  /\*中断标志 \*/  volatile bool button\_flag = false;  volatile bool pdm\_pcm\_flag = false;  volatile bool threshold\_flag = false;  /\* 音量变量 \*/  uint32\_t volume = 0;  uint32\_t noise\_threshold = THRESHOLD\_HYSTERESIS;  /\* HAL Object \*/  cyhal\_pdm\_pcm\_t pdm\_pcm;  cyhal\_clock\_t audio\_clock;  cyhal\_clock\_t pll\_clock;  /\* HAL Config \*/  const cyhal\_pdm\_pcm\_cfg\_t pdm\_pcm\_cfg =  {  .sample\_rate = SAMPLE\_RATE\_HZ,  .decimation\_rate = DECIMATION\_RATE,  .mode = CYHAL\_PDM\_PCM\_MODE\_LEFT,  .word\_length = 16, /\* bits \*/  .left\_gain = CY\_PDM\_PCM\_GAIN\_10\_5\_DB, //0, /\* dB \*/  .right\_gain = 0, /\* dB \*/  };  /\*主函数参考代码\*/  int main(void)  {  cy\_rslt\_t result;  int8\_t audio\_frame[FRAME\_SIZE] = {0};  int32\_t audio\_data;  /\*设备和板级初始化 \*/  result = cybsp\_init() ;  if (result != CY\_RSLT\_SUCCESS)  {  CY\_ASSERT(0);  }  /\* 允许全局中断 \*/  \_\_enable\_irq();  /\*初始化时钟 \*/  clock\_init();  /\*初始化retarget-io 用于 UART 端口打印信息\*/  cy\_retarget\_io\_init(CYBSP\_DEBUG\_UART\_TX, CYBSP\_DEBUG\_UART\_RX, CY\_RETARGET\_IO\_BAUDRATE);  /\* 初始化用户LED \*/  cyhal\_gpio\_init(CYBSP\_USER\_LED, CYHAL\_GPIO\_DIR\_OUTPUT, CYHAL\_GPIO\_DRIVE\_STRONG, CYBSP\_LED\_STATE\_OFF);  /\*初始化用户按键 \*/  cyhal\_gpio\_init(CYBSP\_USER\_BTN, CYHAL\_GPIO\_DIR\_INPUT, CYHAL\_GPIO\_DRIVE\_PULLUP, CYBSP\_BTN\_OFF);  cyhal\_gpio\_enable\_event(CYBSP\_USER\_BTN, CYHAL\_GPIO\_IRQ\_FALL, CYHAL\_ISR\_PRIORITY\_DEFAULT, true);  cyhal\_gpio\_register\_callback(CYBSP\_USER\_BTN, button\_isr\_handler, NULL);  /\* 初始化PDM-PCM模块 \*/  cyhal\_pdm\_pcm\_init(&pdm\_pcm, PDM\_DATA, PDM\_CLK, &audio\_clock, &pdm\_pcm\_cfg);  cyhal\_pdm\_pcm\_register\_callback(&pdm\_pcm, pdm\_pcm\_isr\_handler, NULL);  cyhal\_pdm\_pcm\_enable\_event(&pdm\_pcm, CYHAL\_PDM\_PCM\_ASYNC\_COMPLETE, CYHAL\_ISR\_PRIORITY\_DEFAULT, true);  cyhal\_pdm\_pcm\_start(&pdm\_pcm);  /\* \x1b[2J\x1b[;H - ANSI ESC sequence for clear screen \*/  printf("\x1b[2J\x1b[;H");  printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \  PDM/PCM Example \  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \r\n\n");  for(;;)  {  /\* 检测是否有麦克风数据需要处理 \*/  if (pdm\_pcm\_flag)  {  cyhal\_gpio\_write(CYBSP\_USER\_LED, CYBSP\_LED\_STATE\_ON);  /\* 清除 PDM/PCM 标志 \*/  pdm\_pcm\_flag = 0;  /\* 复位音量 \*/  volume = 0;  /\* 计算一帧数据的绝对值\*/  for (uint32\_t index = 0; index < FRAME\_SIZE/2; index++)  { audio\_data=abs(audio\_frame[index+1]<<8)+abs(audio\_frame[index]);  volume +=audio\_data;  }  if ((volume/VOLUME\_RATIO) > noise\_threshold)  {  printf("\nSTART\n");  for (uint32\_t index = 0; index < FRAME\_SIZE; index++)  {  cyhal\_uart\_putc(&cy\_retarget\_io\_uart\_obj, audio\_frame[index]);//打印音频样本  }  }  cyhal\_gpio\_write(CYBSP\_USER\_LED, CYBSP\_LED\_STATE\_OFF);  }  /\*用户按键第一次按下，重置噪声阈值 \*/  if (button\_flag)  {  /\* Reset button flag \*/  button\_flag=false;  if(threshold\_flag)  {  /\* Get the current volume and add a hysteresis as the new threshold \*/  noise\_threshold = (volume/VOLUME\_RATIO) + THRESHOLD\_HYSTERESIS;  threshold\_flag=true;  }  }  if (0UL != read\_switch\_status())  {  /\*KEY OFF时，读下一帧 \*/  cyhal\_pdm\_pcm\_read\_async(&pdm\_pcm, audio\_frame, FRAME\_SIZE/2);  }  cyhal\_system\_sleep();  }  } |
|  |

主要函数参考参考代码如下：

|  |
| --- |
| /\*按键中断回调\*/  void button\_isr\_handler(void \*arg, cyhal\_gpio\_event\_t event)  {  (void) arg;  (void) event;  button\_flag = true;  }  \*PDM-PCM模块中断回调\*/  void pdm\_pcm\_isr\_handler(void \*arg, cyhal\_pdm\_pcm\_event\_t event)  {  (void) arg;  (void) event;  pdm\_pcm\_flag = true;  }  \*按键KEY OFF状态获取\*/  uint32\_t read\_switch\_status(void)  {  uint32\_t delayCounter = 0;  uint32\_t sw\_status = 0;  /\* Check if the switch is pressed \*/  while(0UL == cyhal\_gpio\_read(CYBSP\_USER\_BTN))  {  /\* Switch is pressed. Proceed for debouncing. \*/  cyhal\_system\_delay\_ms(SWITCH\_DEBOUNCE\_CHECK\_UNIT);  ++delayCounter;  /\* Keep checking the switch status till the switch is pressed for a  \* minimum period of SWITCH\_DEBOUNCE\_CHECK\_UNIT x SWITCH\_DEBOUNCE\_MAX\_PERIOD\_UNITS \*/  if (delayCounter > SWITCH\_DEBOUNCE\_MAX\_PERIOD\_UNITS)  {  /\* Wait till the switch is released \*/  while(0UL == cyhal\_gpio\_read(CYBSP\_USER\_BTN))  {  }  /\* Debounce when the switch is being released \*/  do  {  delayCounter = 0;  while(delayCounter < SWITCH\_DEBOUNCE\_MAX\_PERIOD\_UNITS)  {  cyhal\_system\_delay\_ms(SWITCH\_DEBOUNCE\_CHECK\_UNIT);  ++delayCounter;  }  }while(0UL == cyhal\_gpio\_read(CYBSP\_USER\_BTN));  /\* Switch is pressed and released\*/  sw\_status = 1u;  }  }  return (sw\_status);  } |

上述参考代码没有采用RTOS多任务程序设计的方法。可以通过以下几个步骤编写基于FreeRTOS的任务代码：

（1）添加FreeRTOS库：选中当前工程，使用Library Manager添加FreeRTOS库，这一操作，也添加了abstraction-freertos 和 clib-support库。选择“Librarise”选项卡，勾选“freertos”，然后点击“update”按钮。如下图所示。

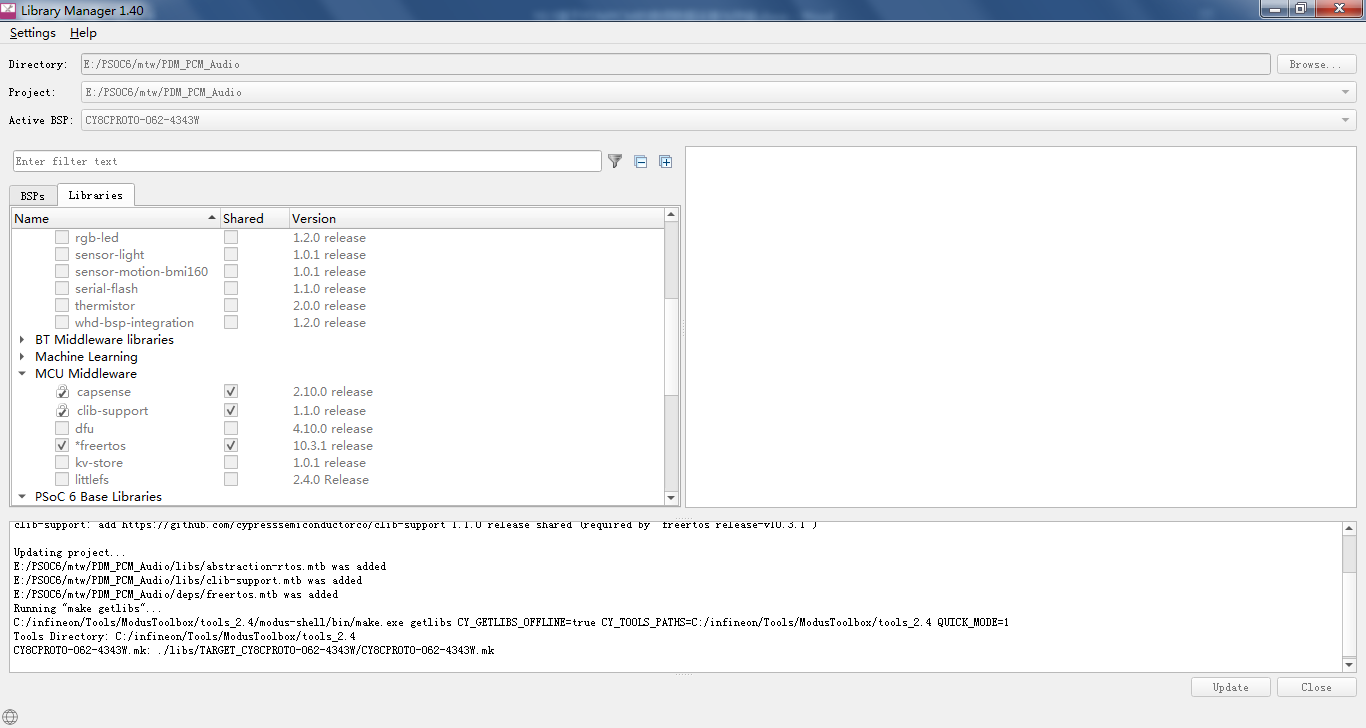


图10.1.8 加载FreeRTOS库

（2）更改Makefile文件，将FREERTOS和 RTOS\_AWARE 添加到COMPONENTS变量：COMPONENTS=FREERTOS RTOS\_AWARE。

（3）将FreeRTOSConfig.h文件从加载后的FreeRTOS库文件夹中（如：\mtb\_shared\freertos\release-v10.3.1\Source\portable\COMPONENT\_CM4）拷贝到应用程序目录中。

（4）在main.c中添加包含文件：

|  |
| --- |
| #include <FreeRTOS.h>  #include <task.h> |

（5）编写基于RTOS的任务代码。

### 10.1.3执行音频数据采集与存储

**1.程序烧录**

（1）使用提供的USB电缆将主板通过KitProg3 USB端口连接到电脑。

（2）打开串行终端程序并选择KitProg3 COM端口。将串行端口参数设置为8N1和115200波特。

（3）使用Eclipse IDE for ModusToolbox烧录程序。在项目资源管理器中选择应用程序项目。在Quick Panel中选择<Application Name> Program (KitProg3\_MiniProg4)执行烧录。

**2．音频采集操作**

编程结束后，应用程序将自动启动。使用设备管理器查看当前串行终端虚拟串口号。运行cmd命令行，执行record\_audio\_serial.py脚本。执行脚本前需要先安装python及pip等环境。如record\_audio\_serial.py文件存放目录为e:\PSOC6\mtw>，则执行以下脚本命令：

e:\PSOC6\mtw>python record\_audio\_serial.py com39 115200 32768 n

其中，com38为虚拟串口端口号，115200为串口波特率，32768为串口接收的字节数，n1为.wav文件名的一部分，每次录入声音后生成的文件名为n.000.wav、n.001.wav……n.00x.wav（x为第x次生成文件）。CMD控制台打印信息如下所示：

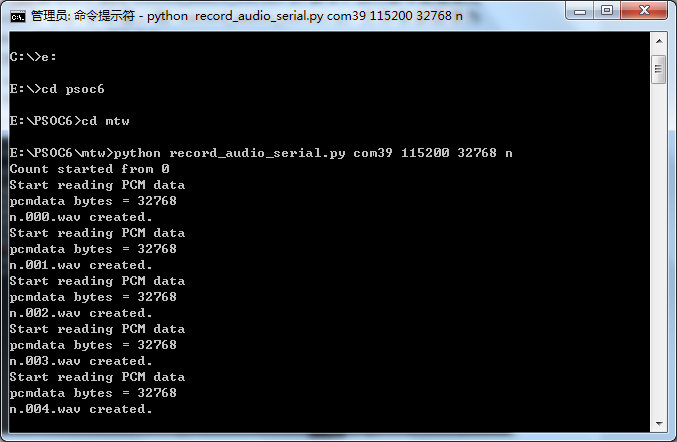


图10.1.9 wav文件生成log信息

（1）录制文件

* 按下用户按键，按下时不要靠近麦克风发出声音，按键放开后，开始录音，观察LED状态。
* 录制后，可以在record\_audio\_serial.py文件所在文件夹中看到生成的.wav文件。使用播放软件，检查生成的.mav文件。
* 使用按键多次操作，找到合适的噪声阈值后，开始录制。可以更改程序中一帧数据的样本数量，通过多次实验，找到适合项目要求的样本数量。

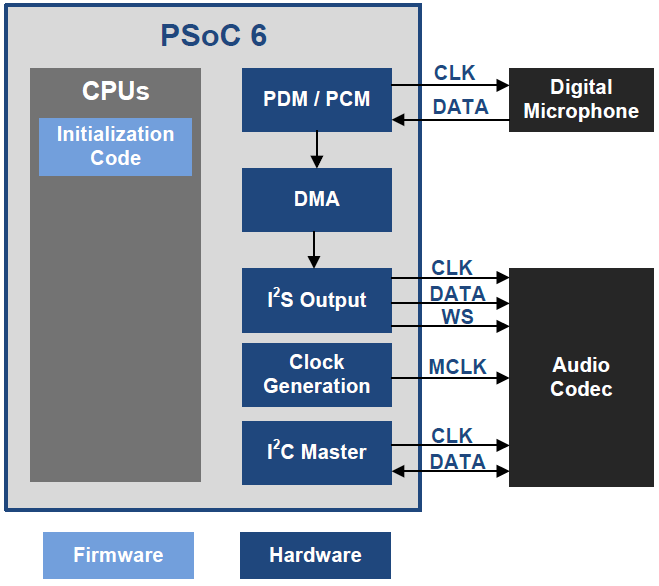
### 10.1.4实现音频数据采集与播放

对于CY8CPROTO-062-4343W开发平台，可以将Pmod I2S2模块焊接到Pmod连接头J15上。各引脚对应关系如下表所示。

表10.1.3 Pmod I2S2模块和开发平台的引脚对应关系

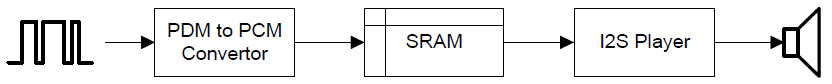
|  |  |  |
| --- | --- | --- |
| I2S 信号 | 开发平台上的引脚 | Pmod IS2S Pin |
| TX\_SCK | P5[1] | J1.2 |
| TX\_DATA | P5[3] | J1.4 |
| TX\_WS | P5[2] | J1.3 |
| MCLK | P5[0] | J1.1 |
| GND | GND | GND |
| VCC | VTARG | VCC |

完成音频编解码器模块的焊接后，便可以进行硬件配置和编写代码实现从数字麦克风录制短音频样本，然后在扬声器或耳机上播放。和上面的案例一样，使用PDM-PCM模块与数字麦克风接口。音频编解码器（AK4954A）的接口信号定义如下图所示。可以通过连接一个扬声器或者耳机，使用音频编解码器播放麦克风记录的数据。系统硬件原理如图10.1.10所示。



10.1.10 音频数据采集与播放原理框图

所有记录的数据都存储在内部SRAM中。录制完成后，I2S模块开始向外部音频编解码器发送数据。按下开发平台上的用户按钮可启动记录。一旦松开按钮，它就会播放按下按钮时录制的内容。处理流程如图10.1.11所示。



10.1.11 音频数据采集与播放处理流程

第一阶段是录制来自麦克风的任何声音，并将其放入PSoC 6 MCU的SRAM中。整个过程可以通过使用PDM-PCM驱动程序将数据从Rx缓冲区传输到SRAM中分配的数组来实现。记录声音后，I2S驱动程序将数据从SRAM传输到Tx缓冲区。

为了从麦克风录制更长的音频流，在两个硬件模块（PDM-PCM和I2S）中仅配置了8 ksps的采样率。PDM-PCM Rx缓冲器和I2S Tx缓冲器的字长设置为16位。为记录的数据数组元素个数设置为65536，总共消耗128 KB的SRAM。

PSoC 6 MCU还为音频编解码器提供时钟源。基于AK4954A数据表，该编解码器在8 kHz的采样频率下需要2.048 MHz MCLK。PSoC 6 MCU通过I2C主机模式配置音频编解码器。案例中包括了AK4954A库（保存路径为deps/audio-codec-AK4954A.mtb）依赖项，可以轻松配置AK4954A。如果不想使用AK4954A，可以编辑Makefile文件，删除行DEFINES+=use\_AK4954A。

MCLK是通过定时器的PWM信号产生的。为PWM和音频子系统提供的时钟源必须相同，以避免任何同步问题。扩展实验使用PLL为CPU/外围设备和音频子系统提供时钟源。案例中PLL的时钟源不能从IMO（8 MHz）获得。因此，程序中的设置值略有不同，注意避免PWM和I2S初始化期间出现问题。

**参考资料**

[1] 002-20730\_PSoC\_R\_6\_MCU\_PSoC\_62\_Architecture\_Technical\_Reference\_Manual

[2]CY8CPROTO-062-4343W\_PSoC\_6\_Wi-Fi\_BT\_Prototyping\_Kit\_Quick\_Start\_Guid

[3]SPK0838HT4H-B datasheet