Adaptive Filtering 功能详解及代码实现

王凡 Ryan Wang (Fan) South China OEM Team

摘 要

Texas instruments 推出的超低功耗 miniDSP 音频 Codec 集成了 miniDSP 内核，可在耗电极低的工作状态下为电池供电的便携式产品提供高性能的语音及音乐处理能力。Adaptive Filtering 是 miniDSP 的一项重要功能。本文详细介绍了该功能的使用方法及注意事项，并给出了示例代码供参考。

目 录

图 表

表 格

1 miniDSP Codec 简介

德州仪器半导体公司（Texas Instruments）推出的内嵌 miniDSP 的音频编解码器（简称 miniDSP Codec）在普通音频编解码器的基础上提供了强大、灵活的低功耗 DSP 引擎来满足消费类电子应用中对音质、音效的需求。

miniDSP 的内核是完全可编程的，支持录音和回放的专用算法。例如：多段均衡（Multi-Band Equalization）、动态噪声消除（Dynamic Noise Filter）、回声消除（Echo Cancellation）等。miniDSP Codec 具有非常优秀的电源管理功能，在提供强大的音效处理能力的同时兼顾了低功耗特性，非常适合电池供电的便携式产品应用，例如智能手机，多媒体播放器，导航仪，电子相框等。

1.1 miniDSP 简介

以 AIC3254 为例，该 miniDSP Codec 集成了两个 miniDSP 内核。如图 1 所示，miniDSP-A 位于 ADC 信号路径上，主要负责 ADC 采样后的数字音频流处理，miniDSP-D 位于 DAC 路径上主要负责 I2S 总线输入的数字音频流处理。miniDSP-A 和 miniDSP-D 之间也有互联的内部数据总线可用于数据交换及共享代码空间。

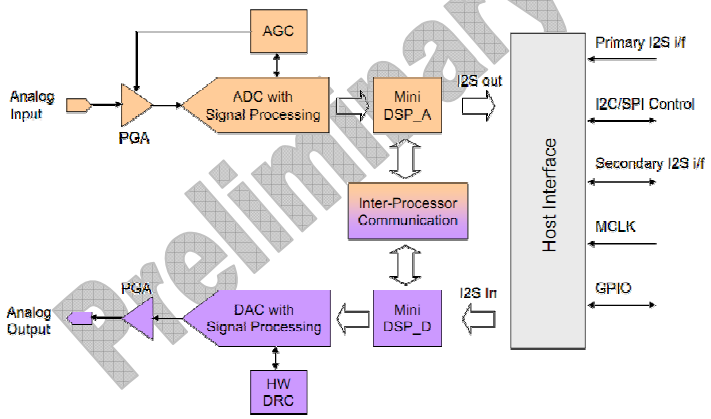


图 1. miniDSP 音频 Codec 内部简化框图

除了 Audio Codec 以外，某些 ADC、DAC 也拥有 miniDSP 内核，本文介绍的部分内容也适用于该类器件。

1.2 miniDSP 内存架构及寄存器地址

miniDSP-A 和 D 分别有自己独立的内存空间。每个 DSP 的内存分为指令内存（Instruction RAM）、数据内存（Data RAM）和系数内存（Coefficient RAM）三类。

- 指令内存用于存储 miniDSP 的运行指令及程序。
- 数据内存用来暂存 miniDSP 运行时的运算结果等临时数据。
- 系数内存用来存储 miniDSP 音效、增益等控件的参数设置。

指令内存和系数内存均可通过映射的 I2C 或 SPI 地址来进行读写。Codec 上电时主控芯片需要通过 I2C 或 SPI 接口将 miniDSP 程序下载到指令内存和系数内存以供运行。以 TLV320AIC3254 为例，指令内存、系数内存的寄存器映射地址及功能如表 1 所示：

表 1. TLV320AIC3254 内存架构及寄存器地址

页面	寄存器说明	类型
Page 0	用来配置时钟，I2S 格式等项目。	控制寄存器地址段
Page 1	用来配置电源、路径管理、增益等项目。	控制寄存器地址段
Page 8/Reg 0	miniDSP-A Adaptive filtering 控制寄存器。	miniDSP-A 系数内存地址段（Coefficient RAM）
Page 8/Reg 8 - 127	miniDSP-A 系数内存空间 Buffer-A ；参数 C0 ~C29.	
Page 9 - 16	miniDSP-A 系数内存空间 Buffer-A ；参数 C30~C255.	
Page 26 - 34	miniDSP-A 系数内存空间 Buffer-B ；参数 C0 ~C255.	
Page 44/Reg 0	miniDSP-D Adaptive filtering 控制寄存器。	miniDSP-D 系数内存地址段（Coefficient RAM）
Page 44/Reg 8 - 127	miniDSP-D 系数内存空间 Buffer-A ；参数 C0 ~C29.	
Page 45 - 52	miniDSP-D 系数内存空间 Buffer-A ；参数 C30~C255.	
Page 62 - 70	miniDSP-D 系数内存空间 Buffer-B ；参数 C0 ~C255.	
Page 80 - 144	miniDSP-A 指令内存空间。	miniDSP-A 指令内存地址段（Instruction RAM）
Page 152 - 186	miniDSP-D 指令内存空间 。	miniDSP-D 指令内存地址段（Instruction RAM）

从寄存器映射地址可发现，miniDSP-A 和 miniDSP-D 的系数内存控件均被等分成两块：Buffer-A 和 Buffer-B。当 Adaptive Filtering 功能开启时，两个 Buffer 的内容是完全同步并相互备份的。这种内存架构是 Adaptive Filtering 功能的基础。

2 Adaptive Filtering

2.1 功能概述

Adaptive Filtering 是一种在 miniDSP 运行时的滤波器、音效控件、混音比例等参数的实时切换功能。启动该功能后，控制器可对系数内存内的参数数据进行实时更新。

当 miniDSP Codec 运行在 non-Adaptive Filtering 模式下，系数内存中的滤波器、音效控件、混音比例等参数将被锁定，无法实时更改。这种模式适用于不需要实时调节参数的场合。miniDSP 的程序和参数将在启动时一次性下载并执行，运行过程中不会进行任何 miniDSP 系数内存参数的修改。

当 Adaptive Filtering 功能启动后，系数内存将启用缓存（Buffer-A 和 Buffer-B）设置。允许用户在 miniDSP 工作时实时修改系数内存中的参数设置，从而满足用户实时调节音效等参数的需求。例如，启用了 Adaptive Filtering 模式后，用户可在听歌过程中将 EQ 均衡器从流行（POP）转换为古典（Classic）模式，该效果实时产生作用无需中断播放。

2.2 Adaptive Filtering 控制寄存器

Adaptive Filtering 功能是通过特定寄存器来控制和实现的，以 AIC3254 的 miniDSP-A 为例，表 2 列出了 miniDSP-A 的 Adaptive Filtering 控制寄存器的含义。用户可通过 D2 位开启或关闭 miniDSP-A 的 Adaptive Filtering 功能。D1 位用来指示 miniDSP 运行时哪一个 Buffer 被锁定，用户可读取该位来确认 Buffer 使用的状态。D0 位用来控制 Buffer 的切换，向 D0 位写入 1 后 miniDSP-A 会切换到新的 Buffer 进行工作，并向控制端口释放原先使用的 Buffer。切换完成后 D0 的值会自动清零，用户可通过读取 D0 的值来判断 Buffer 的切换已经完成。

表 2. AIC3254 miniDSP-A Adaptive Filtering 控制寄存器（P8_R1）

BIT	读/写	复位值	功能
D7-D3	R	00000	保留
D2	R/W	0	ADC Adaptive Filtering 控制 0：禁止 miniDSP-A 的 Aadaptive Filtering 功能 1：启动 miniDSP-A 的 Aadaptive Filtering 功能
D1	R	0	ADC Adaptive Filtering 内存 Buffer 标志位 0：miniDSP-A 锁定 Buffer-A，控制端口可读写 Buffer-B 1：miniDSP-A 锁定 Buffer-B，控制端口可读写 Buffer-A
D0	R/W	0	ADC Adaptive Filtering Buffer 切换控制 0：在下次运算前不执行 Buffer 切换操作 1：在下次运算前执行 Buffer 切换操作 ***Buffer 切换完成后，该 bit 自动清零**

每一个支持 Adaptive Filtering 功能的 miniDSP 均有自己的控制寄存器，miniDSP-A 和 miniDSP-D 的 Adaptive Filtering 功能是独立运行的。

2.3 系数内存（Coefficient Memory）存取规范

在 non-Adaptive Filtering 模式下，系数内存在 miniDSP 停止运行时可通过控制端口（I2C/SPI）直接存取。在 miniDSP 工作时，系数内存将被锁定只有 miniDSP 能够存取。表 3 给出了该模式下的存取规范供参考：

表 3. 在 non-Adaptive Filtering 模式下系数内存的存取规范

miniDSP 状态	miniDSP 存取 Buffer-A&B	控制端口存取 Buffer-A&B	说明
Power Down	NO	YES	miniDSP 停止时，miniDSP 释放系数内存控制权。控制端口(I2C 或 SPI)可以存取系数内存。
Power Up	YES	NO	miniDSP 运行时，miniDSP 使用系数内存。控制端口（I2C 或 SPI）无法访问系数内存。

当启动了 Adaptive Filtering 模式，系数内存将分为 Buffer-A 和 Buffer-B 两块，两块内存内容完全一致，相互备份。miniDSP 工作时将锁定 Buffer-A 或者 Buffer-B 其中的一个，从中获取参数信息。控制端口（I2C 或 SPI）可以读写未锁定的另一块 Buffer。表 4 给出了该模式下的存取规范供参考：

表 4. **Adaptive Filtering 模式下系数内存的存取规范**

miniDSP 状态	miniDSP 存取	控制端口存取	说明
Power Down	NO	YES (Buffer-A and Buffer-B)	miniDSP 停止时，miniDSP 释放系数内存控制权。控制端口(I2C 或 SPI)可以存取系数内存。
Power Up （p8_r1_b1 = 0）	Buffer-A	Buffer-B	miniDSP 运行时，miniDSP 锁定 Buffer-A。控制端口可以存取 Buffer-B。 ***此时向 Buffer-A 内地址进行的读写操作均映射到未锁定的 Buffer-B 地址内。**
Power Up （p8_r1_b1 = 0）	Buffer-B	Buffer-A	miniDSP 运行时，miniDSP 锁定 Buffer-B。控制端口可以存取 Buffer-A ***此时向 Buffer-B 内地址进行的读写操作均映射到未锁定的 Buffer-A 地址内。**

为了简化 Buffer 切换的操作，在 miniDSP 运行时，控制端口访问系数内存的地址均映射到未锁定的 Buffer 内。即如果控制端口向锁定 Buffer 某地址写入的参数将直接更新未锁定 Buffer 内的镜像参数。这个设置允许用户在切换 Buffer 后无需修改写入地址即可向释放出来的 Buffer 内存更新参数。注意在 miniDSP 停止运行的时候，Buffer-A 和 Buffer-B 的页面地址均恢复正常模式，用户需使用它们各自的地址进行参数更新。

2.4 Adaptive Filtering 控制流程

本节详细介绍了如何使用 Adaptive Filtering 功能来进行系数内存内参数的更新操作。附录 A 给出了推荐的 Adaptive Filtering 更新参数操作时序图供参考。

2.4.1 miniDSP 停止时的参数更新流程

如 2.3 节所示，在 miniDSP 停止运行时，控制端口可以存取所有的系数内存地址（Buffer-A 和 Buffer-B）。则该模式下参数更新的流程为：

1. 通过控制接口直接向 Buffer-A 写入新参数。
2. 通过控制接口直接向 Buffer-B 写入同样参数，使 Buffer-A 和 Buffer-B 保持同步。

以 AIC3254 为例，若需要更新的参数在 miniDSP-A 的 Buffer-A 内，其地址为 p8_r44，新参数值为 0xAB。则更新流程为：

1. 向 p8_r44 写入新参数 0xAB。
2. 向 p26_r44 写入新参数 0xAB。（p26_r44 为 Buffer-B 内与 Buffer-A 的 p8_r44 同步的寄存器）

2.4.2 miniDSP 运行时的参数更新流程

miniDSP 运行时，系数内存的其中一个 Buffer 被 miniDSP 锁定，控制端口无法直接修改该 Buffer 内的参数。用户需要先更新未锁定 Buffer 内的参数，然后通知 miniDSP 切换 Buffer 来使用新的参数。切换后原先被锁定的 Buffer 将被释放，用户需对它更新相同的参数以确保两块 Buffer 的参数同步。

在 miniDSP 运行时，系数内存参数更新的流程如下：

1. 通过控制端口向目标寄存器更新参数。
2. 向 Adaptive Filtering 寄存器写入 Buffer 切换命令。
3. 回读 Adaptive Filtering 寄存器状态位，判断 Buffer 切换是否完成。
4. 确认切换完成后，再次向目标寄存器更新参数确保 Buffer-A 和 Buffer-B 参数同步。

以 AIC3254 为例，若需要更新的参数在 miniDSP-A 的 Buffer-A 内，其地址为 p8_r44，新参数值为 0xAB。则更新流程为：

1. 向 p8_r44 写入新参数 0xAB。
2. 将 p8_r1_d0，Adaptive Filtering 寄存器的 D0 位置 1，执行 Buffer 切换操作。
3. 回读 p8_r1_d0，Adaptive Filtering 寄存器状态位，判断 Buffer 切换是否完成。
4. 确认切换完成后，再次向目标寄存器 p8_r44 更新参数 0xAB，确保 Buffer-A 和 Buffer-B 参数同步。

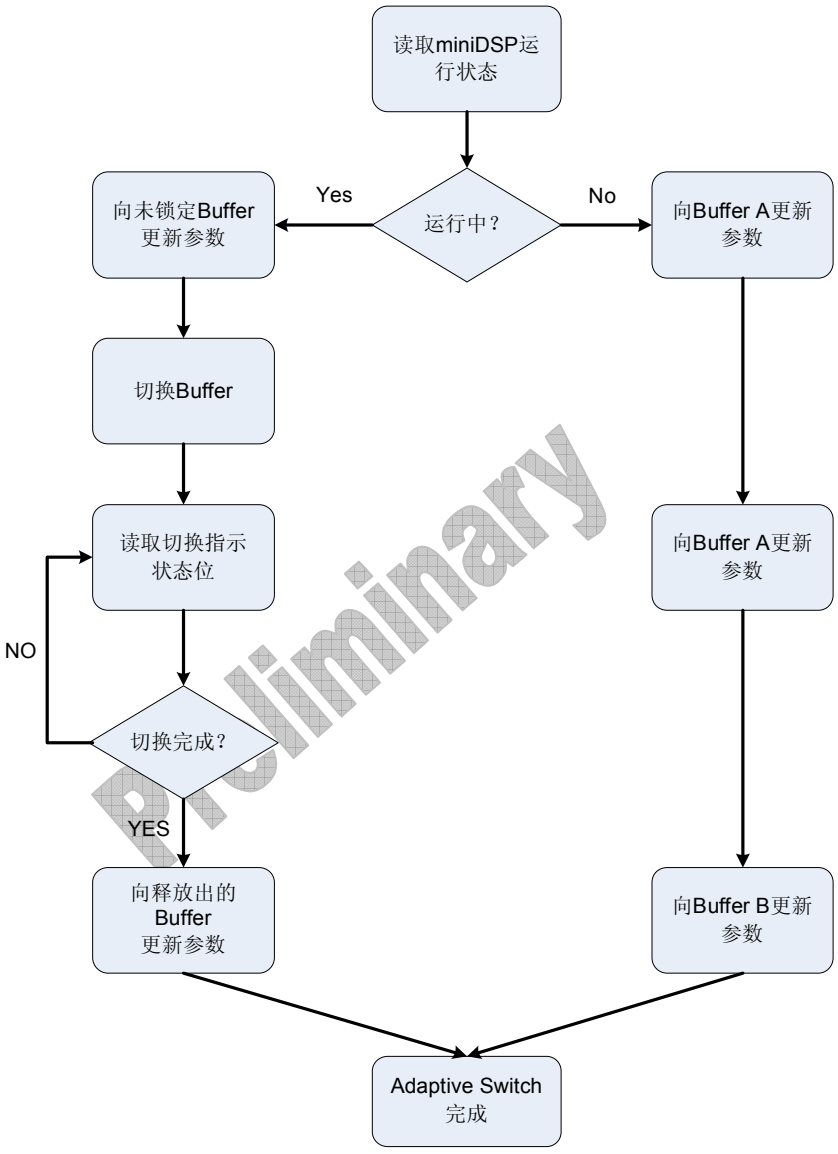
3 总结

本文详细介绍了 miniDSP Codec 的 Adaptive Filtering 功能的使用方法，并以 AIC3254 为例给出了详细的操作步骤和示例代码。在实际使用中，用户需要注意准确的获取并判断 miniDSP 工作状态，选择正确的操作流程来完成参数的切换和更新。

4 参考文献

1. TLV320AIC3254, Ultra Low Power Stereo Audio Codec With Embedded miniDSP-Data sheet (SLAS549)
2. Design and Configuration Guide for the TLV320AIC3204 & TLV320AIC3254 Audio Codec (SLAA404C)
3. Coefficient RAM Access Mechanisms (SLAA425A)

Appendix A. Adaptive Filtering 流程图



Appendix B. Adaptive Filtering 参考代码

```
int adaptive_filtering(int page, int reg, int data_len, void *data) {
      unsigned int count = 5; //count for retry checking swap bit.
      unsigned int val = 0;//temp val to store read back register value.
      unsigned int adaptive_filter_reg = ADC_ADAPTIVE_FILTER_REG;//target swap register.
      unsigned int miniDSP_Power = miniDSP_ON;//power statues.
      unsigned int Buffer_A_B_offset = 18;//define the offset between Buffer-A and B, the value for AIC325x is 18.
      //Check which miniDSP need buffer switching
      if(page < 44){ //Target reg is in miniDSP-A
            adaptive_filter_reg = ADC_ADAPTIVE_FILTER_REG;
            val = aic325x_read(ADC_FLAG);//read back ADC flag to check power statues.

            if((val & 0x44) == 0x00){// miniDSP-A is OFF
                miniDSP_Power = miniDSP_OFF;
            }
            else {// miniDSP-A is ON
                miniDSP_Power = miniDSP_ON;
            }
      }
      else if(page >= 44){ //This reg is in miniDSP-D
            adaptive_filter_reg = DAC_ADAPTIVE_FILTER_REG;
            val = aic325x_read(DAC_FLAG_1); //read back DAC flag to check power statues

            if((val & 0x88) == 0x00){// miniDSP-D is OFF
                miniDSP_Power = miniDSP_OFF;
            }
            else{// miniDSP-D is ON
                miniDSP_Power = miniDSP_ON;
            }
      }
      if (miniDSP_Power == miniDSP_OFF){ //Update Buffer-A and B without switching buffer during power off
            aic325x_change_page(page);
            aic325x_continue_write(reg, data, data_len);

            aic3256_change_page(page + Buffer_A_B_offset);
            aic325x_continue_write(reg, data, data_len);
      }
      else if(miniDSP_Power == miniDSP_ON){ //Update and Switch Buffer-A and B during miniDSP power on
            aic3256_change_page(page);
            aic325x_continue_write (reg, data, data_len); // update one buffer

            val = aic3256_read(adaptive_filter_reg);
            aic3256_write(adaptive_filter_reg, (val | 0x01)); // Switch buffer

            do { // waiting for buffer switching finish
                 mdelay(10);//delay 10mS
                 val = aic3256_read(adaptive_filter_reg);
                 if((val & 0x01) == 0){ //buffer switching successful, update another buffer
                   aic3256_change_page(page);
                   aic325x_continue_write(reg, data, data_len);
                   break;
                 }
                 count --;
                 if(count == 0){ //timeout, buffer switching fail, jump out of loop.
                   aic3256_change_page(0);// Change page back to 0 for general operation.
                   return -1;
                 }
            }while(count > 0);
       }
       aic3256_change_page(0);// Change page back to 0 for general operation.
       return 0;
      }
```