报告目的

- 了解、熟悉音频基本知识，弄清楚什么是采样宽度，采样频率，声道数，以及DAC、ADC的作用
- 了解熟悉I2S、PCM、SPDIF等相关的接口及其通信方式、以及Codec内部的实现
- 结合音频控制器相关的接口，了解、熟悉与其相关的电路原理图

报告目的

- 熟悉各个应用场景的分析，及其相关的音频数据流
- 熟悉ALSA音频架构，及其相关的驱动编程、应用编程技巧
- 通过对音频的ALSA驱动架构的分析，对驱动的编写、调试过程有个大概的了解

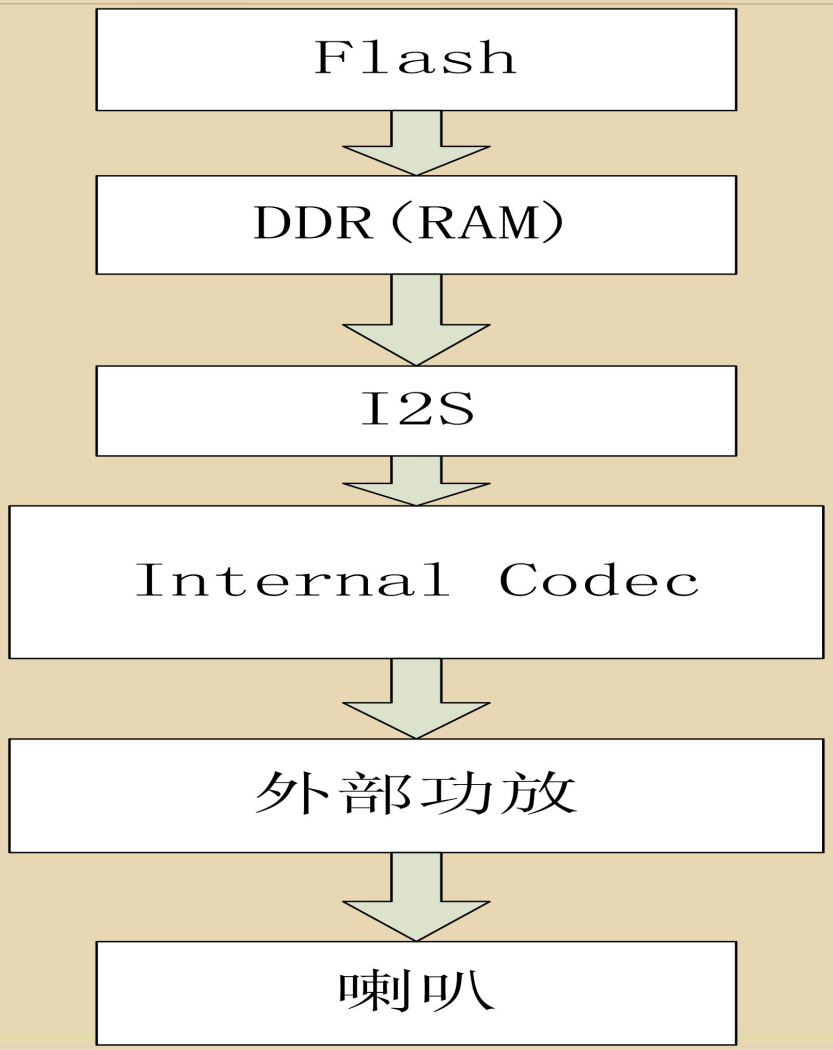
音频应用的场景

- 音频应用的场景是由SoC所具有的音频相关的设备所决定的
- 我们的SoC(以我们目前调试的M200)所具有的音频设备有AIC（包括了I2S,SPDIF,AC97）,PCM,内部codec。

M200所支持的音频应用场景

- Dorado开发板上的喇叭播放声音，使用了内部CODEC和I2S
- Dorado开发板上的模拟MIC录音，使用了内部CODEC和I2S
- 使用耳机播放声音，使用了I2S，内部CODEC
- 使用耳机录音，使用了I2S，内部CODEC
- 使用SPDIF音箱播放声音
- 使用板子上的DMIC录音（仅仅只有M200具有该功能）
- 使用外部的CODEC实现录音、放音等功能，使用到了I2S
- AC97
- 蓝牙耳机播放音乐
- 实现蓝牙耳机的功能（智能手表打电话）
- HDMI（JZ4780具有）播放声音

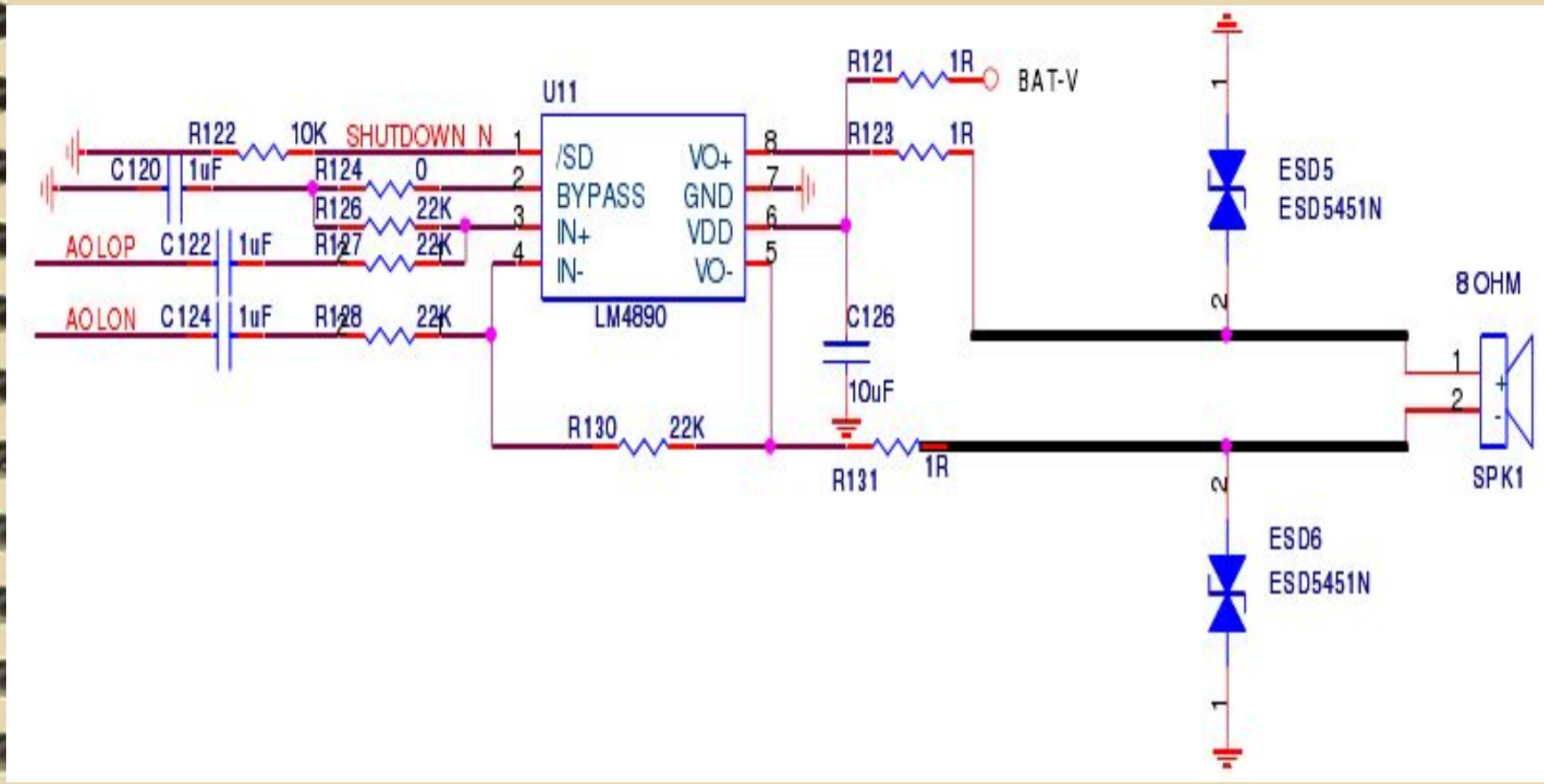
喇叭播放声音数据流图



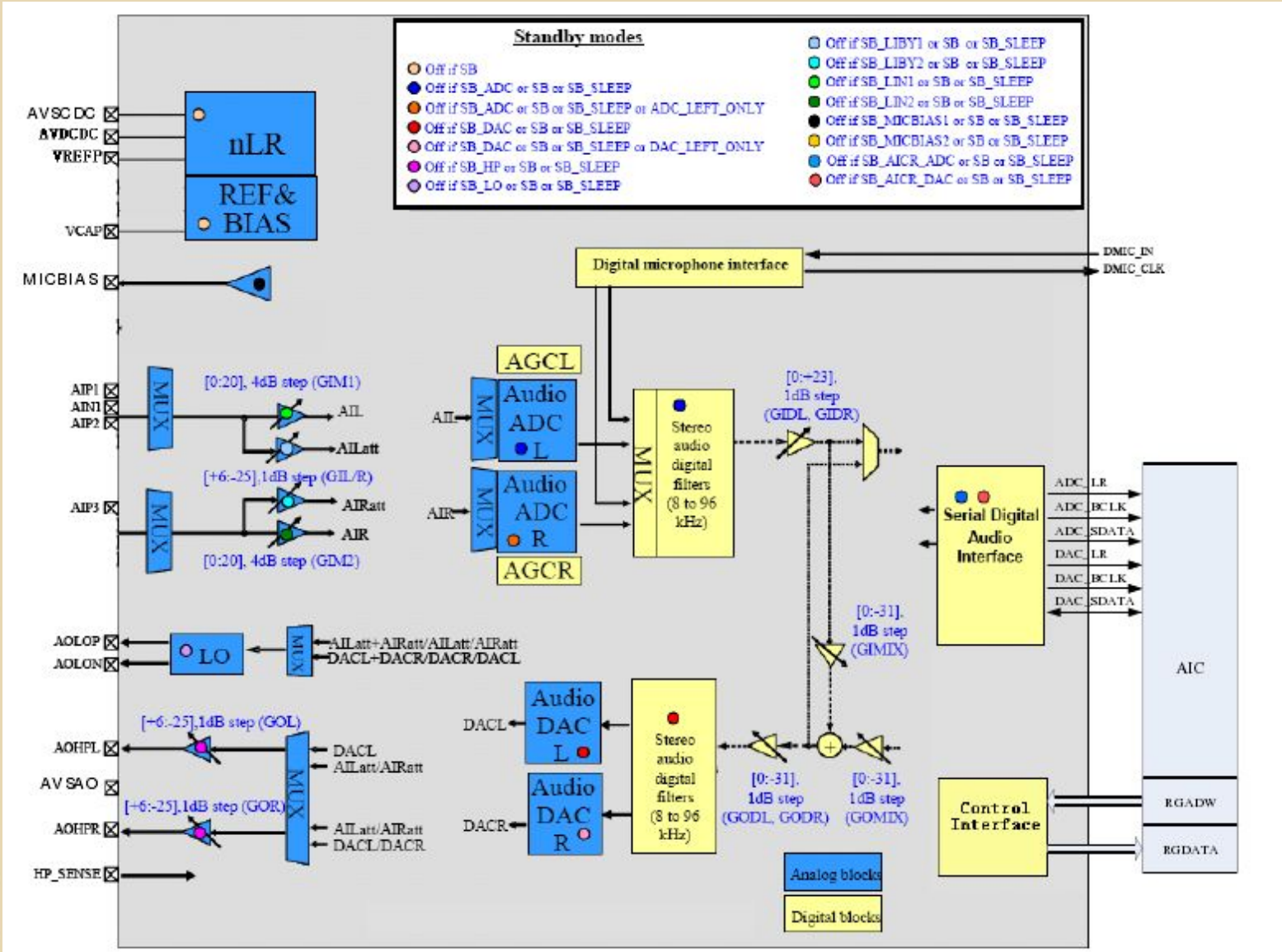
喇叭播放声音

音频应用将需要播放的音频文件中的音频数据按照特定的方式解码（如果需要）到DDR中，然后调用DMA相关驱动触发DMA将DDR中的原始的PCM音频数据搬运到I2S的FIFO的入口，与此同时调用I2S相关驱动触发I2S控制器把FIFO中的数据按照一定的格式传输给Internal Codec，音频应用会调用Internal Codec相关的驱动，将Int Codec接收到的数字音频数据通过内部的DAC转换为模拟音频信号，通过特定的pin脚传输到SoC外部的功率放大器，最后功率放大器将接收到的音频信号放大后通过外部的喇叭播放出来

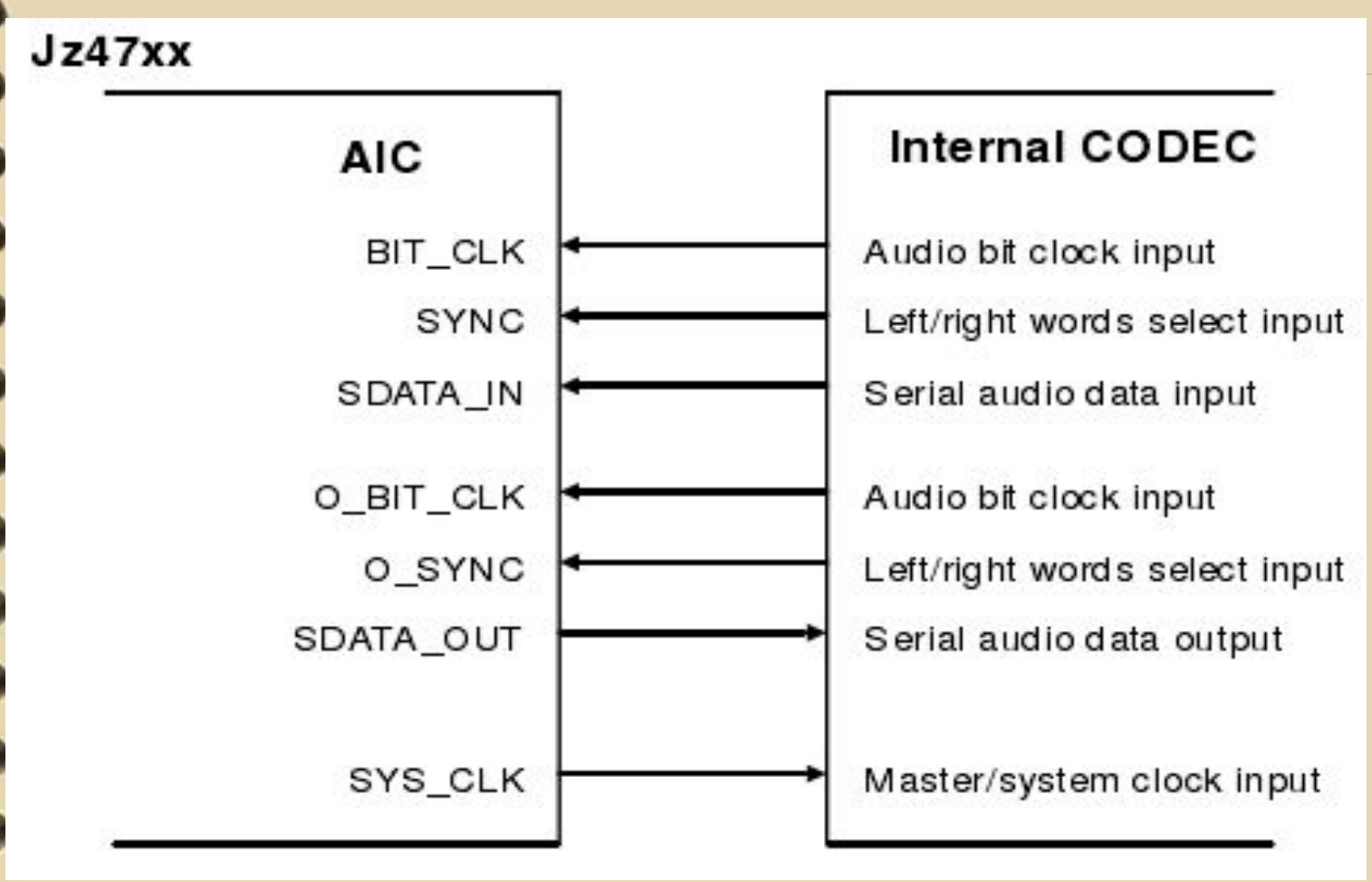
喇叭电路原理图


U11
/SD
BYPASS
IN+
IN-
VO+
GND
VDD
VO-
LM4890
R121 1R
BAT-V
R123 1R
R122 10K SHUTDOWN N
C120 1uF
R124 0
R126 22K
R127 22K
AOLOP C122 1uF
AOLON C124 1uF
R128 22K
R130 22K
C126
10uF
R131 1R
ESD5
ESD5451N
ESD6
ESD5451N
8 OHM
SPK1

内部Codec的结构图



I2S接口


Jz47xx
AIC
Internal CODEC
BIT_CLK
SYNC
SDATA_IN
O_BIT_CLK
O_SYNC
SDATA_OUT
SYS_CLK
Audio bit clock input
Left/right words select input
Serial audio data input
Audio bit clock input
Left/right words select input
Serial audio data output
Master/system clock input

I2S音频数据传输格式

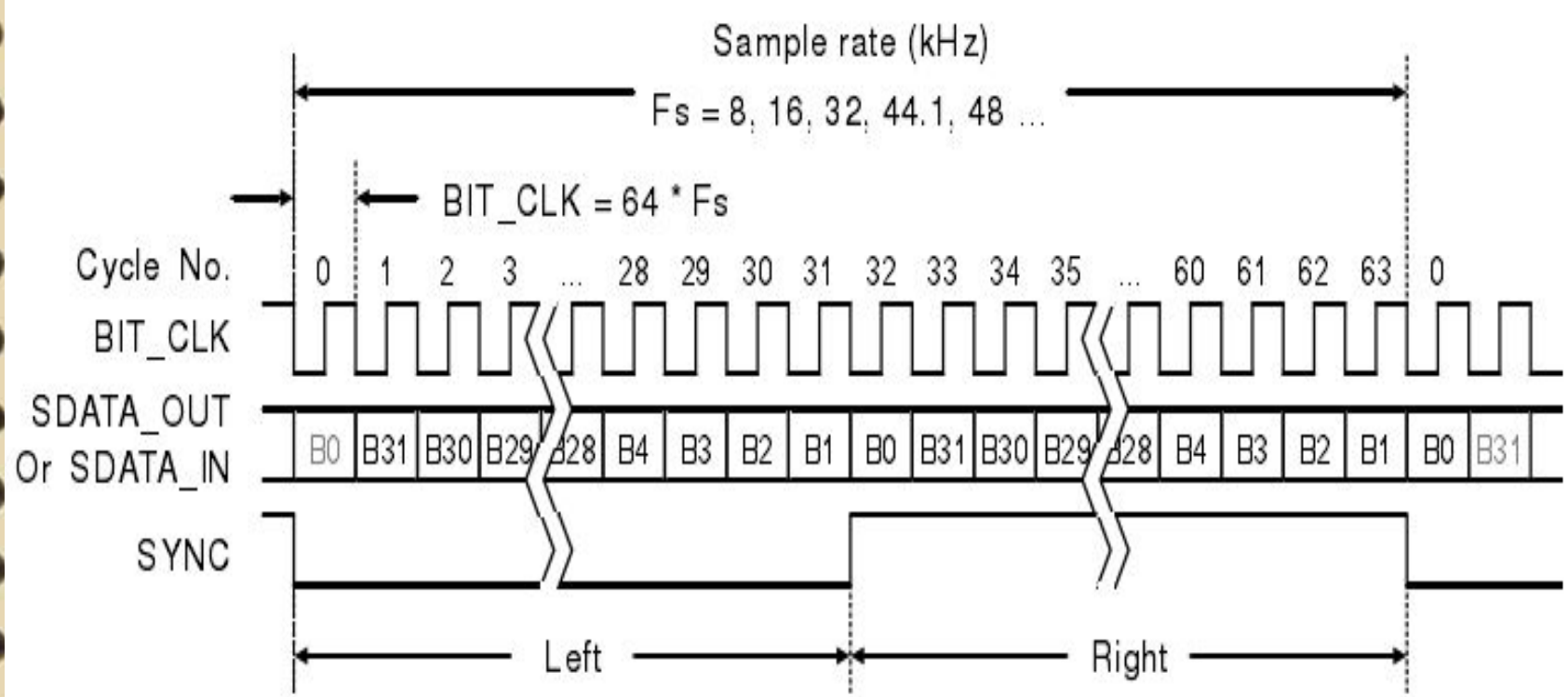
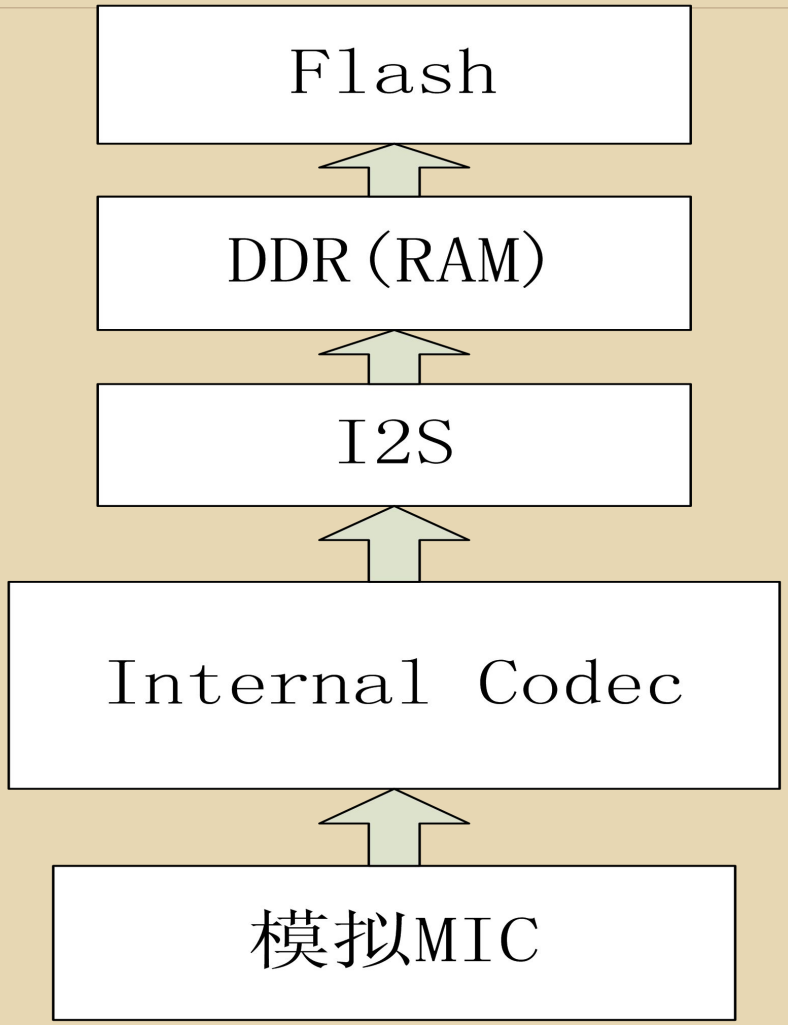


Figure 1-10 I2S data format (A: LR mode)

ALSA框架代码（驱动层）函数调用关系

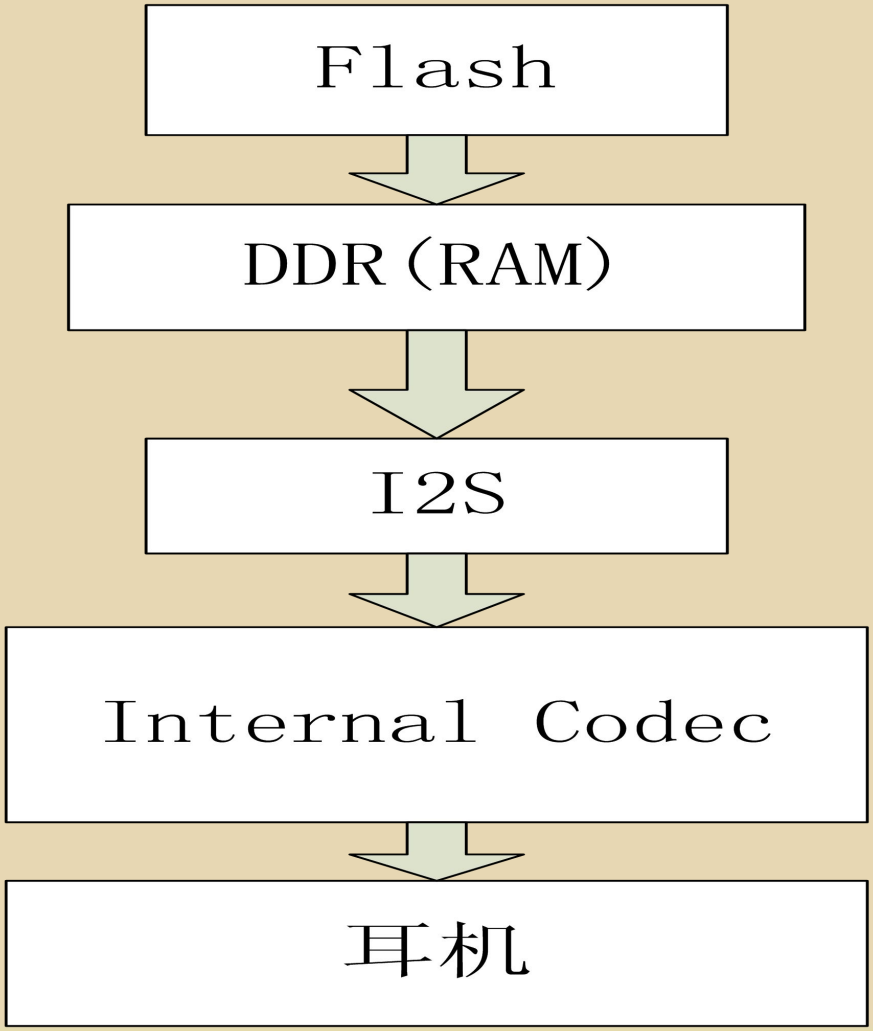
模拟MIC录音过程数据流图



模拟MIC录音

- 应用程序打开录音对应的ALSA架构的设备节点pcmC0D0c后，设定好相应的音频数据格式（采样宽度，采样率，声道数），应用程序会调用Internal Codec驱动，触发Internal Codec将在模拟MIC（模拟MIC将外部的音频转换成模拟的音频电信号）收到的音频信号通过ADC转换成数字的音频数据，然后按照一定的格式传输到I2S，应用程序调用I2S驱动启动I2S将收到的音频数据按照一定的格式将音频数据转换成PCM格式的音频数据放到FIFO中，然后应用程序会通过函数调用DMA驱动触发DMA将I2Sfifo中的数据取出放到特定的RAM中，音频应用将RAM中的音频数据保存到FLASH中。

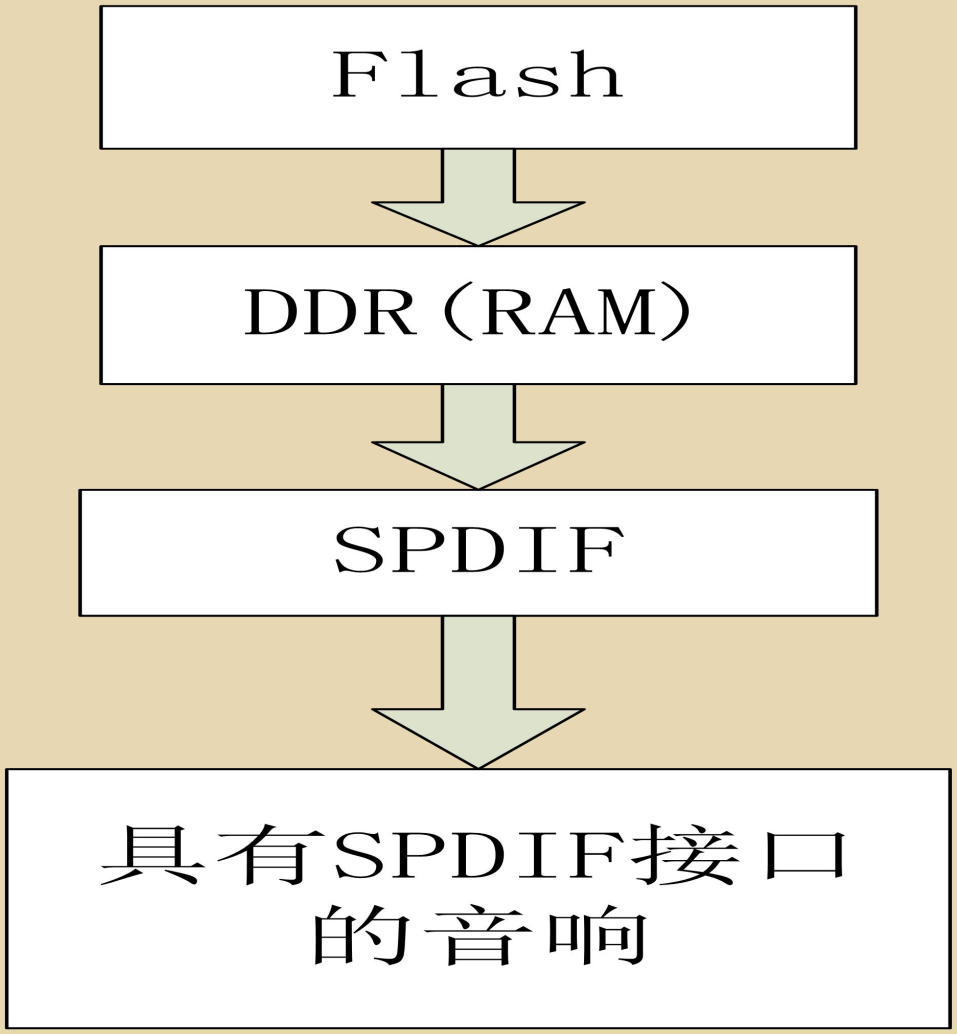
耳机放音过程数据流图



耳机放音

- 音频应用将需要播放的音频文件中的音频数据按照特定的方式解码（如果需要）到DDR中，DMA将DDR中的原始的PCM音频数据搬运到I2S的FIFO的入口，然后I2S把FIFO中的数据按照一定的格式传输给Internal Codec，Internal Codec将接收到的数字音频数据通过内部的DAC转换为模拟音频信号，通过特定的pin脚传输到SoC外部的耳机播放出来。

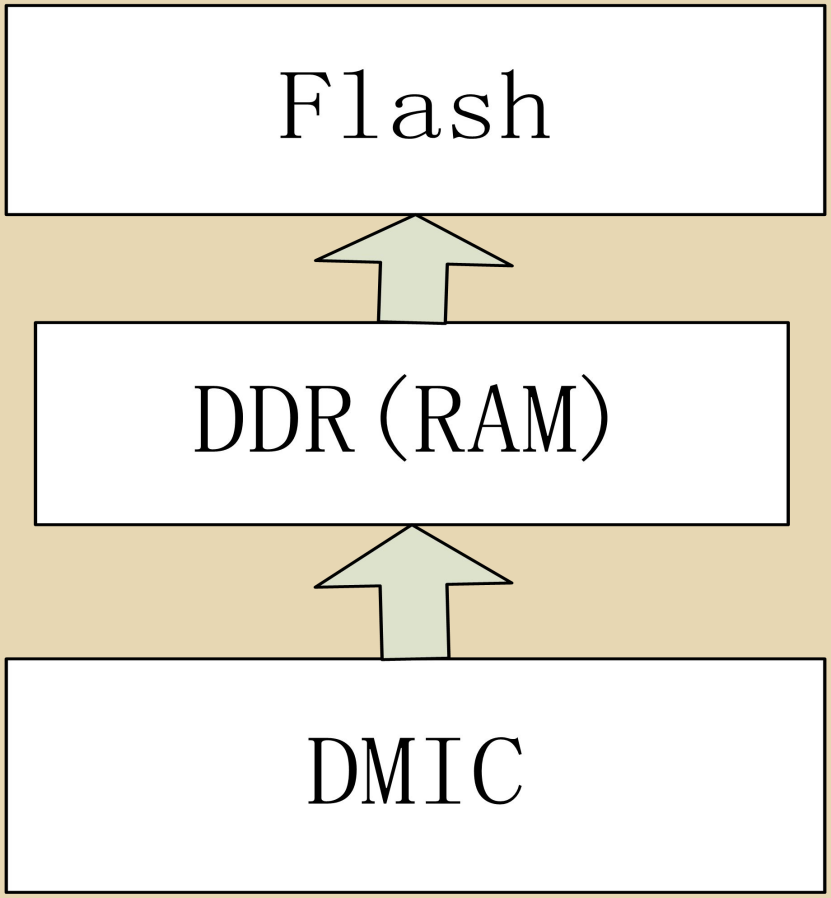
SPDIF放音过程数据流图



SPDIF放音

- 音频应用将需要播放的音频文件中的音频数据按照特定的方式解码（如果需要）到DDR中，DMA将DDR中的原始的PCM音频数据搬运到SPDIF的FIFO的入口，然后SPDIF把FIFO中的数据按照一定的格式传输给SoC外部的SPDIF音响中，SPDIF音响负责将这种特定格式的音频数据解码出来并播放

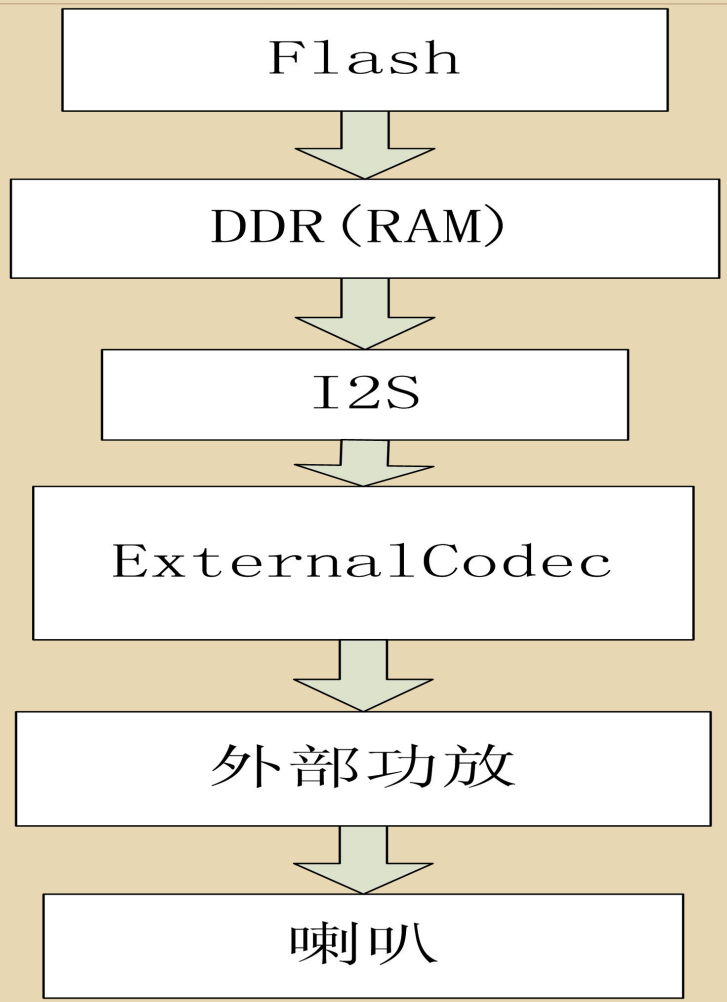
DMIC录音过程数据流图



DMIC录音

- 说明：DMIC将外部的音频转换成模拟的音频数据，并放到FIFO中，DMA负责将DMIC fifo中的数据取出放到特定的RAM中，音频应用将RAM中的音频数据保存到FLASH中。

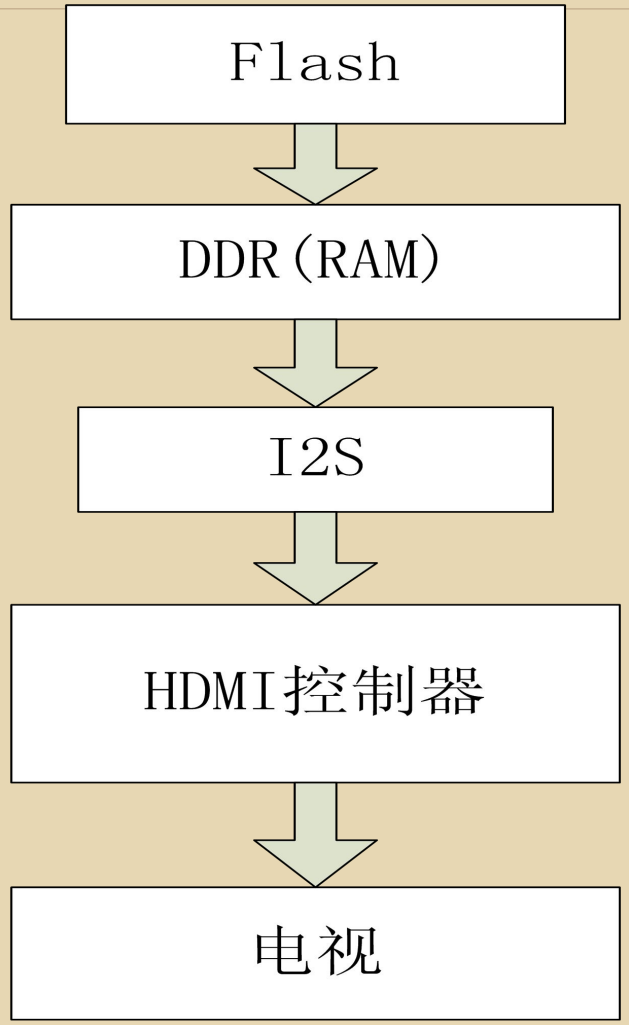
外部Codec放音过程数据流图



外部Codec放音

- 音频应用将需要播放的音频文件中的音频数据按照特定的方式解码（如果需要）到DDR中，DMA将DDR中的原始的PCM音频数据搬运到I2S的FIFO的入口，然后I2S把FIFO中的数据按照一定的格式传输给External Codec，External Codec将接收到的数字音频数据通过内部的DAC转换为模拟音频信号，通过特定的pin脚传输到SoC外部的功率放大器，然后功率放大器将接收到的音频信号放大后通过外部的喇叭播放出来。

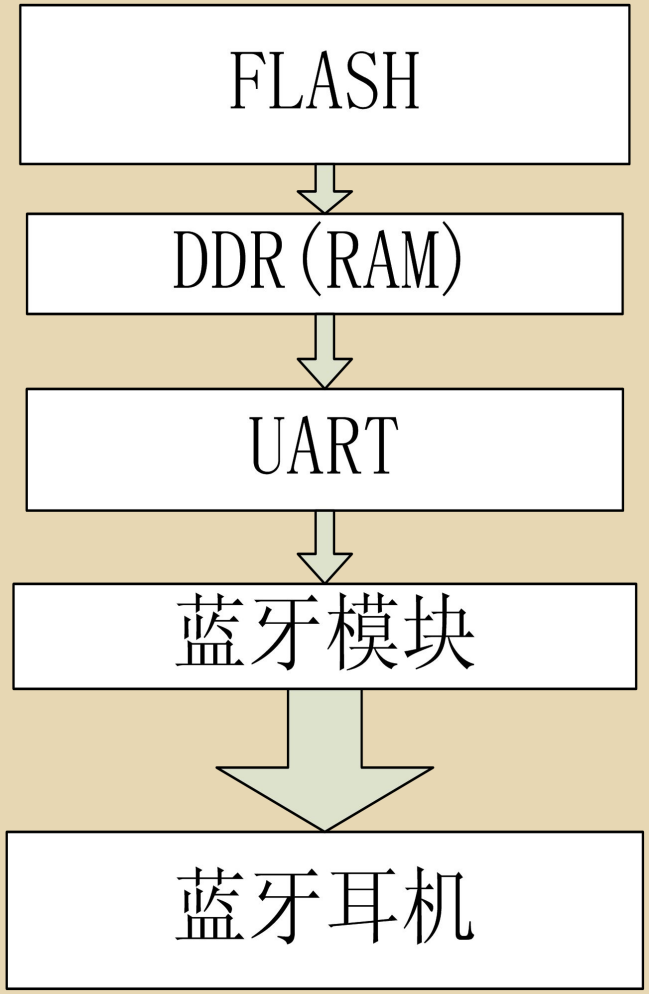
HDMI放音过程数据流图



HDMI放音

- 音频应用将需要播放的音频文件中的音频数据按照特定的方式解码（如果需要）到DDR中，DMA将DDR中的原始的PCM音频数据搬运到I2S的FIFO的入口，然后I2S把FIFO中的数据按照一定的格式传输给HDMI控制器，HDMI控制器将接收到的数字音频数据通过HDMI的某些特定的线传输到SoC外部的电视并播放出来。

蓝牙耳机播放音乐过程数据流图



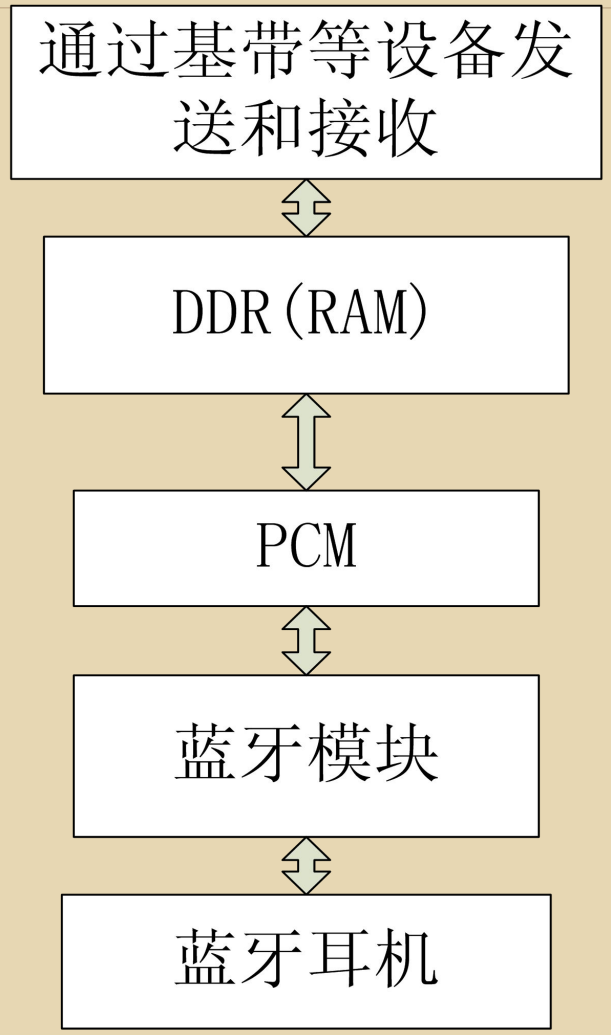
蓝牙耳机播放音乐

- 音频应用将音频文件加载到DDR中，HAL层的代码根据蓝牙连接状态选择蓝牙播放的接口，并按照特定的格式写入到tty相关的节点中，这样CPU或者DMA就可将DDR中的数据传输到UART的FIFO中，UART将数据按照串口的数据传输方式传输给蓝牙模块，蓝牙模块将收到的数据发送出去(to 与其相连的蓝牙耳机设备)，蓝牙耳机收到数据够，通过一定的解码方式将数据解码成音频数据，并将其播放出来

蓝牙打电话功能

- 蓝牙打电话分为两种情况，一种是SoC作为源端，绝大多数情况都是这样子的，比如普通的我们手机用蓝牙打电话的情况；第二种情况，是用SoC作为蓝牙耳机端，与其设备（比如手机）相连，实现蓝牙耳机的功能，这种情况以前比较少见，但是现在在智能手表中比较常见

蓝牙打电话数据流图



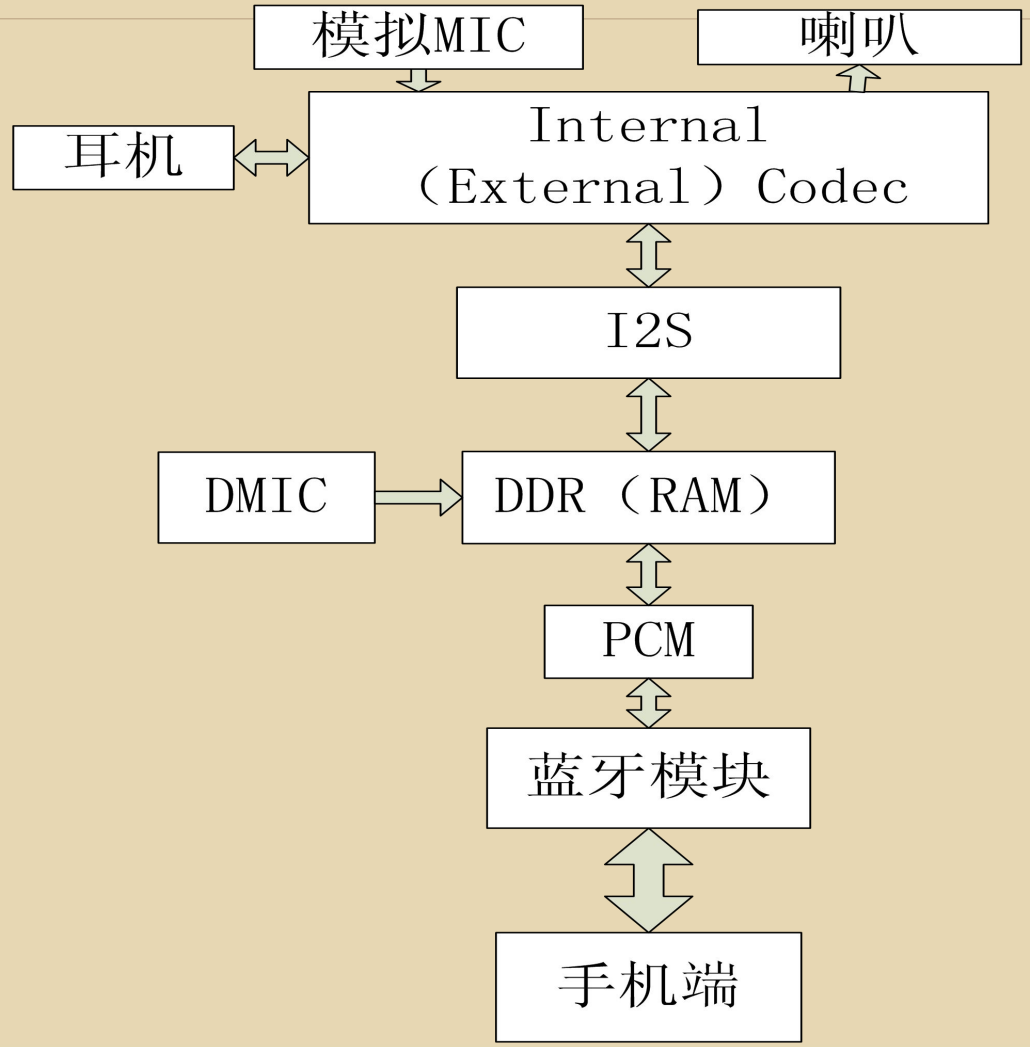
蓝牙打电话

- SoC作为源端的情况。本地说话对方听时：蓝牙耳机负责将现实中的音频信号采集并转换成数字的音频数据，并通过一定的格式传输给蓝牙模块，蓝牙模块将音频数据通过PCM接口传输给PCM，PCM控制器将接收到的数据转换成PCM格式的音频数据放到FIFO中，DMA负责将PCM FIFO中的数据搬运到DDR中，然后电话应用将DDR中的PCM格式的音频数据通过基带发送出去

蓝牙打电话

- SoC作为源端的情况。对方说话本地听时：电话应用将基带中传输过来的音频数据放到DDR中，然后触发DMA将数据从DDR中搬运到PCM的fifo的入口处，PCM将fifo中的数据按照一定的格式传输给蓝牙模块，蓝牙负责将收到的音频数据打包后发送给蓝牙耳机，蓝牙耳机将数据解码成音频数据后播放出来

实现蓝牙耳机功能数据流图



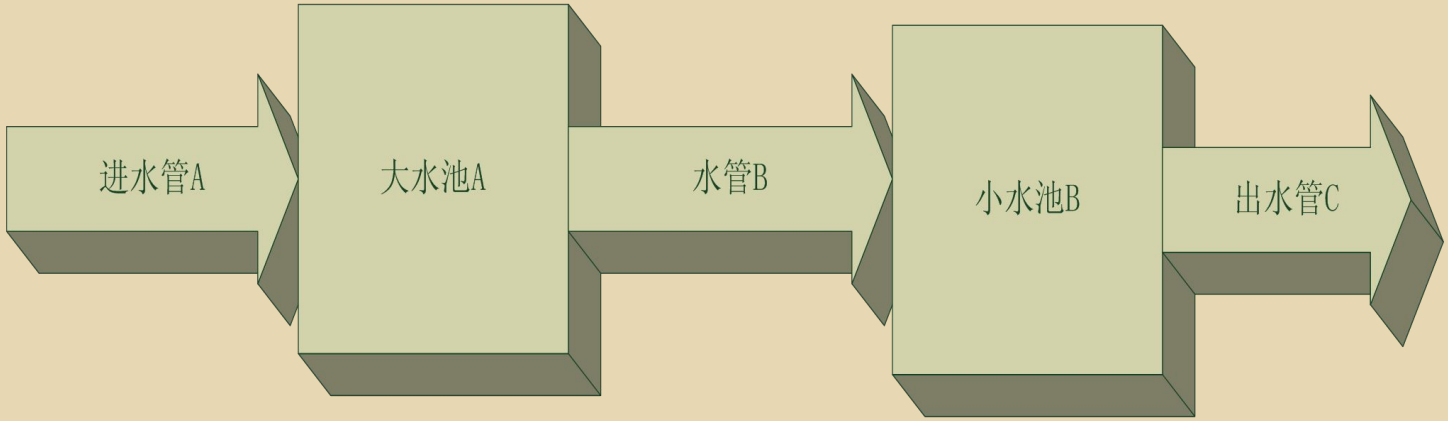
实现蓝牙耳机功能

- SoC实现蓝牙耳机的情况。相关应用的作用采集音频信号，将采集的音频信号通过蓝牙模块发送出去，并且接收从蓝牙模块过来的音频信号，并播放出来。四个部分，采集、发送，接收、播放。首先应用先采集音频信号，这一部分可以通过模拟MIC->Interanl(ExterNAl)CODec->I2S->DDR或者耳机->Interanl(External)Codec->I2S->DDR或者DMIC->DDR通路完成，发送通过DDR->PCM->蓝牙模块通路完成，接收通过蓝牙模块->PCM->DDR通路完成，播放通过DDR->I2S->Interanl(External)Codec->耳机（或者喇叭）

目前驱动中存在的问题

- 播放过程中存在断音的问题，为什么？

出水管C在什么情况下会断水？



进水管A单位时间流过的水量大，但是进水的频率慢，水管B将水从大水池A放进小水池B，水管B属于自动调节，如果规定好放水的次数后，当小水池B水不够满时，会启动水管B放水进来，当小水池B满了后，会停止水管B放水

- 直接原因，小水池B空了，会断水。

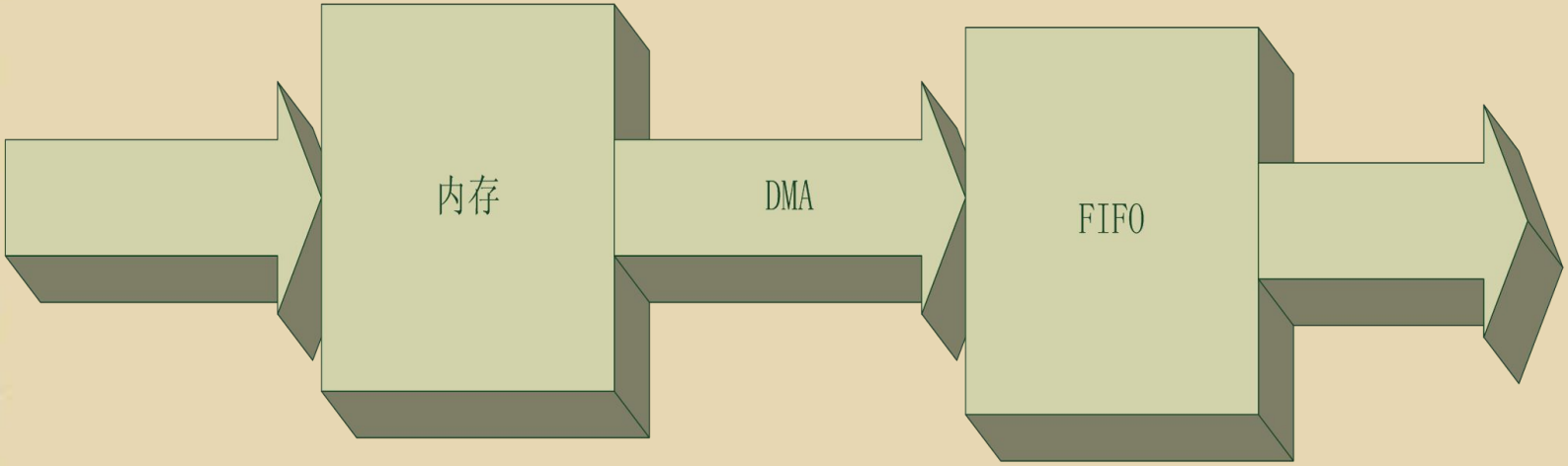
造成小水池B空的原因有：

1，水管B没有及时把水从大水池A中放进来

2，由于进水管A没来得及补充水，大水池A没有水了

3，大水池A不够大

实际情况



- 如果FIFO足够大，比如1M个entry，驱动中的申请的内存足够大，比如5M bytes，那么就不会再存在断音的情况，让我们算一下
- 假如播放的是音乐格式是采样率为192KHz，采样宽度为24bit，声道为2，驱动中的buffer大小为1M bytes（256 pages），深度为64 entry，DDR为150MHz（假如cpu和DDR间的数据拷贝仅仅为200M bytes/s）

- 那么1s播放出去的声音数据为192000 * 24 * 2/8 (bytes)=1152000 bytes，而拷贝到buffer的数据为200M bytes，即每10ms播放1152 byte,进入buffer的数据为2M bytes,而buffer仅仅有1M空间，所以播放完1 * 1024 * 1024 / 1152 =910次，即910 * 10ms=9s中内需要将程序调度回来，显然这是没有太大问题，因为我们的系统中不可能跑910个线程的。也就是说 驱动中的buffer为1M byte的空间是没有问题的。

- 那么接下来分析FIFO
- fifo为64个entry，那么存放的音频数据仅仅是64 * 24=1536 bits=192bytes，播放完这些数据需要的时间是192/115.2 (ms)=1.67mS
- 也就是说需要播放完fifo中的数据后的2ms内就要把数据添加到fifo中，如果这段时间没有做到这一点，那么就断音了。

接下来的问题

- 为什么2ms内，没有数据进入FIFO中？这2ms内，cpu干嘛去了？在什么情况下会发生这样的情况？
- 我们先看看整个的函数调用关系，当DMA将数据搬运到FIFO中完成后，会产生一个中断，这个中断处理函数中会调用一个
- tasklet_schedule(&dmac->tasklet);

- 这个函数的是告诉cpu，发生了tasklet软中断了，cpu可以运行dmac->tasklet相对应的处理函数了，也就是jzdma_chan_tasklet（），而这个函数中又调用了
- dmac->tx_desc.callback(dmac->tx_desc.callback_param);
- 这个callback函数就是我们用过来启动下次DMA续传音频数据的接口

- 如果2ms内没有调用到这个callback，那么就段音了，什么时候会发生这样的情况呢？那么我们就要看看tasklet的相关知识点了。
- tasklet是用软中断实现的，软中断是运行在中断上下文中。在硬件中断处理函数处理完毕以后，会在do_IRQ()函数的最后调用irq_exit()，这个函数的实现中

- 347 if (!in_interrupt() && local_softirq_pending())
- 348 invoke_softirq();
- 349
- 其中local_softirq_pending()就是在我们调用 tasklet_schedule(&dmac->tasklet)时设置为非0的。

- 而in_interrupt() 在这里的作用是判断当前是否是在软中断环境中，如果两个条件都满足的情况下，会调用invoke_softirq()
- 这个函数的实现
- 328 if (!force_irqthreads)
- 329 do_softirq();
- 而if的条件在我们的板子上总是满足的，所以就调用了do_softirq(), 其中在do_softirq

（）的实现中

```
282         pending = local_softirq_pending();
283
284         if (pending)
285                 __do_softirq();
```

在__do_softirq()实现中会遍历所有的软中断，依次为

- 408 HI_SOFTIRQ=0,
- 409 TIMER_SOFTIRQ,
- 410 NET_TX_SOFTIRQ,
- 411 NET_RX_SOFTIRQ,
- 412 BLOCK_SOFTIRQ,
- 413 BLOCK_IOPOLL_SOFTIRQ,
- 414 TASKLET_SOFTIRQ,
- 415 SCHED_SOFTIRQ,
- 416 HRTIMER_SOFTIRQ,
- 417 RCU_SOFTIRQ, /* Preferable RCU should always be the last softirq */
- 418
- 调用到TASKLET_SOFTIRQ对应的action函数tasklet_action()时，开始真正的运行在调用tasklet_init注册的func(),不过要注意，这个是在开中断情况下运行的。

- 根据上面的分析，有人根据软中断调用的关系，决定不用tasklet了，因为调用到它太晚了，需要提升几个优先级，比如用定时器或者其他的，在这里用定时器是不合适的，那么就用最高优先级的HI_SOFTIRQ，这是从一定程度上解决问题，可是不能从根本上解决，为啥？
- 前面提到过，软中断的处理函数是在硬件中断打开的情况下进行的，假如硬件中断处理函数占用时间过长，中断又频繁发生，那么你无论你用什么软中断，都可能出现断音的情况

如何解决断音呢？

- DMA搬运完毕的中断回调函数的实现是通过tasklet实现的，根本上是由软中断实现，软中断的调用顺序是HI_SOFTIRQ,TIMER_SOFTIRQ,NET_TX_SOFTIRQ,NET_RX_SOFTIRQ,BLOCK_SOFTIRQ,

- TASKLET_SOFTIRQ,SCHED_SOFTIRQ等，并且软中断虽然在中断上下文执行，可是执行软中断过程中是开硬件中断的，所以软中断是会被硬中断打断的，如果中断足够频繁
- 那么就可能导致DMA的回调函数在2ms以内没有执行完成。即使你将DMA回调用HI_SOFTIRQ实现，除非你将软中断的实现是关中断的，但这是绝对不可能的（否则会严重降低系统的实时性）。

解决方式

- DMA的Descriptor传输方式+定时器

实现方式

- 在DMA还没有完全传输完毕的时候（传输最后一个descriptor之前），将最后一个descriptor之后再挂一个descriptor，这样DMA就不会出现间隔。
- 从而解决了这个问题。

调试内核代码需要考虑问题

- 1，当前函数、代码所处的是什么环境中（中断上下文、进程上下文）
- 2，当前函数、代码是不是处于原子环境中，能不能被调度（能不能被多个线程重复调用，内核代码本身就是一个全局资源）？
- 3，当前函数、代码中有没有用到共用的资源包括全局变量、寄存器等等，要不要进行保护？用什么方法进行保护？

- 4，当前函数、代码有没有用到同步机制，如果需要，那么应该怎么选择？（一般用到同步的地方，都会放弃cpu，主动请求调度器调度，如果放弃了cpu，那么什么时候唤醒本线程？如何唤醒？在哪里唤醒？）
- 5，代码是否符合内核规范？还能不能被优化？是否利于别人阅读、调试？

问题

- 内核中断子系统
- 中断产生后，CPU执行代码的过程是怎样的？先跳转到哪里开始执行？在执行中断处理函数前做了哪些动作？又是怎么一步步跳转到驱动中注册的中断处理函数中？中断处理函数运行返回后，又都做了些什么事情？为什么那么做？
- 你的驱动中的中断处理函数，从开始到最后执行了多长时间？
- 我们的整个中断处理函数执行完毕用了多长时间？还能不能进行优化？