BTP401平台IIS总结

目录

目录

[1、简介 2](#_Toc346197350)

[2、时序 2](#_Toc346197351)

[3、传送机制 ---- 保证最有效的数据得到传输 3](#_Toc346197352)

[4、RT5350的IIS格式如下图所示： 3](#_Toc346197353)

[5、RT5350的i2s驱动分析 3](#_Toc346197354)

[5.1 IIS模块驱动分析 3](#_Toc346197355)

[5.1.1 I2s\_mod\_init函数 3](#_Toc346197356)

[5.1.2 i2s\_mmap函数 5](#_Toc346197357)

[5.1.3 I2s\_ioctl函数 6](#_Toc346197358)

[5.2 I2S\_PUT\_AUDIO命令 7](#_Toc346197359)

[6、iis的环形缓冲操作： 8](#_Toc346197360)

[4、用户空间和内核空间的调用： 10](#_Toc346197361)

[7、数据发送函数GdmaI2sTx的调用过程分析 -----DMA中断 11](#_Toc346197362)

[7.1 GdmaI2sTx()函数调用图： 14](#_Toc346197363)

## 1、简介

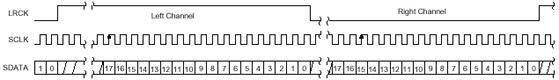
I2S（Inter-IC Sound Bus）是飞利浦公司为数字音频设备之间的音频数据传输而制定的一种总线标准。在飞利浦公司的I2S标准中，既规定了硬件接口规范，也规定了数字音频数据的格式。I2S有3个主要信号：

1.串行时钟SCLK，也叫位时钟（BCLK），即对应数字音频的每一位数据，SCLK都有1个脉冲。SCLK的频率=2×采样频率×采样位数

2. 帧时钟LRCK，用于切换左右声道的数据。LRCK为“1”表示正在传输的是左声道的数据，为“0”则表示正在传输的是右声道的数据。LRCK的频率等于采样频率。

3.串行数据SDATA，就是用二进制补码表示的音频数据。

有时为了使系统间能够更好地同步，还需要另外传输一个信号MCLK，称为主时钟，也叫系统时钟（Sys Clock），是采样频率的256倍或384倍。一个典型的I2S信号见图（图3 I2S信号）

[](http://www.edisc.com.cn/media/200107/03/NewsMedia_81.jpg)

## 2、时序

1）数据的最高位总是出现在LRCK(ws)变化的第二个SCK处

2）WS为低时传输左声道，为高时传输右声道

3）SCK每个脉冲周期传输一个bit

4）SD是串行数据，传输时首先传高位数据（MSB），数据以二进制补码的形式传送

   I2S格式的信号无论有多少位有效数据，数据的最高位总是出现在LRCK变化（也就是一帧开始）后的第2个SCLK脉冲处。这就使得接收端与发送端的有效位数可以不同。如果接收端能处理的有效位数少于发送端，可以放弃数据帧中多余的低位数据；如果接收端能处理的有效位数多于发送端，可以自行补足剩余的位。这种同步机制使得数字音频设备的互连更加方便，而且不会造成数据错位。

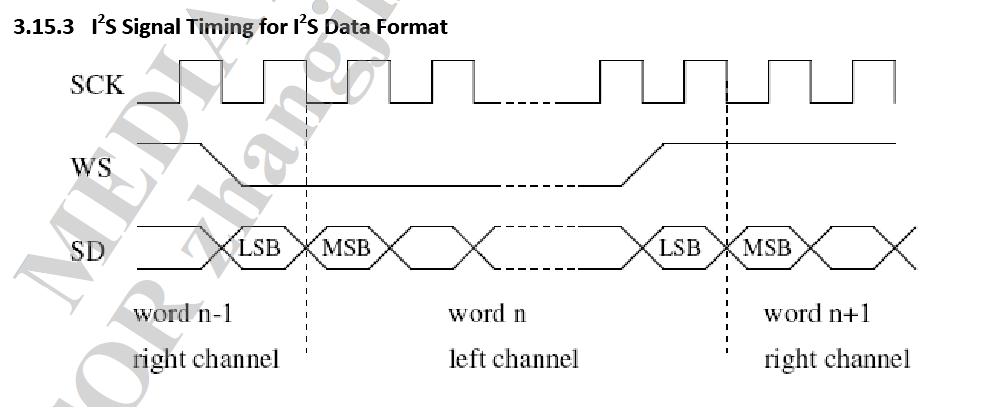
## 3、传送机制 ---- 保证最有效的数据得到传输

1）如果数据接收端接收到的数据比它规定的字长要长，超出部分的低位会被忽略

2）如果数据接收端接收到的数据比它规定的字长要短，空余部分会以零填充

3）产生SCK（同步时钟）和 WS（声道选择时钟）的一端作为主

## 4、RT5350的IIS格式如下图所示：



## 5、RT5350的i2s驱动分析

### 5.1 IIS模块驱动分析

#### 5.1.1 I2s\_mod\_init函数

/\* 1、申请设备号，注册到内核 \*/

result = register\_chrdev(i2sdrv\_major, I2SDRV\_DEVNAME, &i2s\_fops);

if (i2sdrv\_major == 0) {

i2sdrv\_major = result; /\* dynamic \*/

}

/\* 2、向sys虚拟文件系统注册一个i2s的类，同时创建一个设备 \*/

i2smodule\_class=class\_create(THIS\_MODULE, I2SDRV\_DEVNAME);

device\_create(i2smodule\_class, NULL, MKDEV(i2sdrv\_major, 0), I2SDRV\_DEVNAME);

* 1. I2s\_open函数

static int i2s\_open(struct inode \*inode, struct file \*filp)

{

int Ret;

/\* 从设备节点中提取此设备号 \*/

int minor = iminor(inode);

/\* 获取模块 \*/

try\_module\_get(THIS\_MODULE);

/\* 判断是否为非阻塞打开，不支持 \*/

if (filp->f\_flags & O\_NONBLOCK) {

MSG("filep->f\_flags O\_NONBLOCK set\n");

return -EAGAIN;

}

/\* set i2s\_config，分配一个pi2s\_config\_type类型的结构体变量 \*/

pi2s\_config = (i2s\_config\_type\*)kmalloc(sizeof(i2s\_config\_type), GFP\_KERNEL);

/\* 将pi2s\_config结构体保存到file结构体的私有数据成员中，以被其他函数使用 \*/

filp->private\_data = pi2s\_config;

/\* 清零pi2s\_config \*/

memset(pi2s\_config, 0, sizeof(i2s\_config\_type));

/\* 分配一个i2s\_status\_type类型的结构体，用于提示i2s的状态 \*/

pi2s\_status = (i2s\_status\_type\*)kmalloc(sizeof(i2s\_status\_type), GFP\_KERNEL);

memset(pi2s\_status, 0, sizeof(i2s\_status\_type));

/\* pi2s\_config初始化 \*/

pi2s\_config->flag = 0;

pi2s\_config->dmach = GDMA\_I2S\_TX0;

/\* tx\_ff\_thres: 是指tx的fifo阀值，这个函数没有使用fifo模式 \*/

pi2s\_config->tx\_ff\_thres = CONFIG\_I2S\_TFF\_THRES;

pi2s\_config->tx\_ch\_swap = CONFIG\_I2S\_CH\_SWAP;

pi2s\_config->rx\_ff\_thres = CONFIG\_I2S\_TFF\_THRES;

pi2s\_config->rx\_ch\_swap = CONFIG\_I2S\_CH\_SWAP;

/\* slave mode: using an external clock \*/

pi2s\_config->slave\_en = CONFIG\_I2S\_SLAVE\_EN;

/\* set sampling rate \*/

pi2s\_config->srate = 44100;

/\* 音量大小 \*/

pi2s\_config->txvol = 0;

pi2s\_config->rxvol = 0;

/\* 测试时使用 \*/

pi2s\_config->lbk = CONFIG\_I2S\_INLBK;

pi2s\_config->extlbk = CONFIG\_I2S\_EXLBK;

pi2s\_config->fmt = CONFIG\_I2S\_FMT;

/\* dma channel,这里的通道是由该驱动定义的，实际使用需要跟具体硬件匹配 \*/

pi2s\_config->dmach = GDMA\_I2S\_TX0;

/\* 初始化发送和接收的等待队列，用于发送接收环形缓冲满的时候调用使其休眠 \*/

init\_waitqueue\_head(&(pi2s\_config->i2s\_tx\_qh));

init\_waitqueue\_head(&(pi2s\_config->i2s\_rx\_qh));

return 0;

}

综述整个open函数做了一下的事情：

1. 提取此设备号
2. 检查是否设置nonblock
3. 分配pi2s\_config和pi2s\_status变量，分别用于设置和保存状态
4. pi2s\_config的通用设置

#### 5.1.2 i2s\_mmap函数

static int i2s\_mmap(struct file \*filp, struct vm\_area\_struct \*vma)

{

int nRet;

/\* 获取要申请的空间大小 \*/

unsigned long size = vma->vm\_end-vma->vm\_start;

if((pi2s\_config->pMMAPBufPtr[0]==NULL)&&(pi2s\_config->mmap\_index!=0))

pi2s\_config->mmap\_index = 0;

/\* 如果被赋值则不重新进行开辟空间，直接创建页表即可 \*/

if(pi2s\_config->pMMAPBufPtr[pi2s\_config->mmap\_index]!=NULL)

goto EXIT;

/\* 分配空间，进行dma流式映射 \*/

**/\* dma流式映射和一致性映射的区别，流式映射中数据发送成功后，缓冲区中的数据是否还 会存在 \*/**

pi2s\_config->pMMAPBufPtr[pi2s\_config->mmap\_index] = kmalloc(size, GFP\_DMA);

i2s\_mmap\_addr[pi2s\_config->mmap\_index] = (dma\_addr\_t)dma\_map\_single(NULL, pi2s\_config->pMMAPBufPtr[pi2s\_config->mmap\_index], size, DMA\_BIDIRECTIONAL);

EXIT:

memset(pi2s\_config->pMMAPBufPtr[pi2s\_config->mmap\_index], 0, size);

/\* 创建页表 \*/

nRet = remap\_pfn\_range(vma, vma->vm\_start, virt\_to\_phys((void\*)pi2s\_config->pMMAPBufPtr[pi2s\_config->mmap\_index]) >> PAGE\_SHIFT, size, vma->vm\_page\_prot);

/\* mmap的缓冲区加1 \*/

pi2s\_config->mmap\_index++;

/\* TX和RX最大的缓冲区数量都是8 \*/

#if defined(CONFIG\_I2S\_TXRX)

if(pi2s\_config->mmap\_index>MAX\_I2S\_PAGE\*2)

#else

if(pi2s\_config->mmap\_index>MAX\_I2S\_PAGE)

#endif

/\* 缓冲区数量超过8，则不再开辟内存空间 \*/

pi2s\_config->mmap\_index = 0;

return 0;

}

I2s\_mmap做了一下事情：

1. 检查开辟的缓冲区数量是否超过最大个数，如果超过了，则不进行空间的开辟直接映射到用户空间即可。
2. 申请DMA空间，并且映射给设备，最终创建页表，让用户空间可以操作它。

#### 5.1.3 I2s\_ioctl函数

1. I2S\_TX\_ENABLE命令

spin\_lock\_irqsave(&ptri2s\_config->lock, flags);

/\* allocate tx buffer \*/

/\*

\* 动态分配一个一致性的缓冲区，与i2s\_mmap中的缓冲区功能想区别

\* 这里的缓冲区从产生就通过iis开始发送数据，在mmap模式下用户不能在用户

\* 空间给这个空间写入任何数据，它的存在主要的目的是通过iis发送乱码，即在

\* 没有数据的状态下，会一直发送0，以防止数据乱码，发出异常的声音。便于在

\* 有数据的时候随时切换到发送数据的状态下。

\*/

ptri2s\_config->pPage0TxBuf8ptr = (u8\*)pci\_alloc\_consistent(NULL, I2S\_PAGE\_SIZE\*2 , &i2s\_txdma\_addr);

ptri2s\_config->pPage1TxBuf8ptr = ptri2s\_config->pPage0TxBuf8ptr + I2S\_PAGE\_SIZE;

/\* 将iis\_mmap函数中的开辟的空间的首地址一次保存到

ptri2s\_config->pMMAPTxBufPtr[i]数组中，已便于后面调用函数进行发送\*/

for( i = 0 ; i < MAX\_I2S\_PAGE ; i ++ )

{

ptri2s\_config->pMMAPTxBufPtr[i] = ptri2s\_config->pMMAPBufPtr[i];

}

/\* 当i2S使能之后就开始发送ptri2s\_config->pPage0TxBuf8ptr中的数据，以防止

声音异常，当有数据时可以随时切换到有效数据的发送的状态下 \*/

GdmaI2sTx((u32)ptri2s\_config->pPage0TxBuf8ptr, I2S\_FIFO\_WREG, 0, I2S\_PAGE\_SIZE, i2s\_dma\_tx\_handler, i2s\_unmask\_handler);

GdmaI2sTx((u32)ptri2s\_config->pPage1TxBuf8ptr, I2S\_FIFO\_WREG, 1, I2S\_PAGE\_SIZE, i2s\_dma\_tx\_handler, i2s\_unmask\_handler);

/\* 清零pi2s\_conifg \*/

i2s\_reset\_tx\_config(ptri2s\_config);

ptri2s\_config->bTxDMAEnable = 1;

/\* 对i2s进行通用的设置 \*/

i2s\_tx\_config(ptri2s\_config);

/\* i2s\_clock\_enable主要是设置sampling rate和share pin以及slave mode \*/

if(ptri2s\_config->bRxDMAEnable==0)

i2s\_clock\_enable(ptri2s\_config);

audiohw\_set\_lineout\_vol(1, ptri2s\_config->txvol, ptri2s\_config->txvol);

i2s\_tx\_enable(ptri2s\_config);

GdmaUnMaskChannel(GDMA\_I2S\_TX0);

spin\_unlock\_irqrestore(&ptri2s\_config->lock, flags);

break;

I2S\_TX\_ENABLE命令的作用：

1、分配一致性的缓冲区，用于防止音频的异常和随时触发有效数据的发送

1. 调用GdmaI2STx函数准备发送数据，同时配置DMA（hard mode/src/des/burstsize）
2. 调用i2s\_clock\_enable函数配置sampling rate、share pin、slave mode

4、配置i2s,使能i2s开始发送数据

### 5.2 I2S\_PUT\_AUDIO命令

do{

/\* 禁止中断，保存中断标志位到flags中 \*/

spin\_lock\_irqsave(&ptri2s\_config->lock, flags);

/\* 如果写缓冲的索引值和读缓冲的索引值相差不为4，则还可以继续为环形缓冲区中写数据，否则还有四个缓冲区的数据没有发送出去，不可以进行写操作，休眠 \*/ if(((ptri2s\_config->tx\_w\_idx+4)%MAX\_I2S\_PAGE)!=ptri2s\_config->tx\_r\_idx)

{

/\* 写环形缓冲后，使环形缓冲的索引值加1 \*/

ptri2s\_config->tx\_w\_idx=(ptri2s\_config->tx\_w\_idx+1)%MAX\_I2S\_PAGE; /\* 将写环形缓冲区的索引值(ptri2s\_config->tx\_w\_idx)输出到用户空间， 便于用户空间进行写环形缓冲区的操作 \*/

put\_user(ptri2s\_config->tx\_w\_idx, (int\*)arg);

/\* txbuffer\_len反映到i2s状态上 \*/

pi2s\_status->txbuffer\_len++;

/\* 释放自选所，使能中断，恢复中断状态标志位 \*/

spin\_unlock\_irqrestore(&ptri2s\_config->lock, flags);

break;

}

/\* (tx\_w\_idx % 8) – rx\_w\_idx = 4 \*/

else

{

pi2s\_status->txbuffer\_ovrun++;

spin\_unlock\_irqrestore(&ptri2s\_config->lock, flags);

/\* 已经有四个缓冲区没有发送了，休眠等待数据发送完 \*/

interruptible\_sleep\_on(&(ptri2s\_config->i2s\_tx\_qh));

}

}while(1);

break;

I2S\_PUT\_AUDIO命令的功能如下：

1. 主要控制环形缓冲区的操作，防止数据的丢失和覆盖

## 6、iis的环形缓冲操作：

1、初始时，读写游标都为0，指向环形缓冲的第一个区域

2、用户每次想环形缓冲区写入数据之后tx\_w\_idx需要执行以下操作：

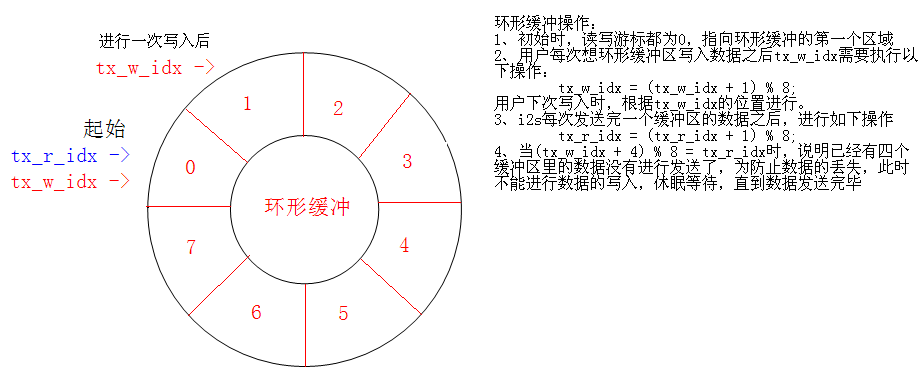
tx\_w\_idx = (tx\_w\_idx + 1) % 8;

用户下次写入时，根据tx\_w\_idx的位置进行。

3、i2s每次发送完一个缓冲区的数据之后，进行如下操作

tx\_r\_idx = (tx\_r\_idx + 1) % 8;

1. 当（tx\_w\_idx + 4）% 8 = tx\_r\_idx 时，说明已经有四个缓冲区里的数据没有进行发送了，为了防止数据的丢失，此时不能进行数据的写入，休眠等待，直到条件不成立。



## 4、用户空间和内核空间的调用：



## 7、数据发送函数GdmaI2sTx的调用过程分析 -----DMA中断

int GdmaI2sTx(

uint32\_t Src, /\* 源地址 \*/

uint32\_t Dst, /\* 目的地址 \*/

uint8\_t TxNo, /\* 发送的通道 \*/

uint16\_t TransCount, /\* 发送的字节数 \*/

void (\*DoneIntCallback)(uint32\_t data), /\* 发送成功后的回调函数 \*/

void (\*UnMaskIntCallback)(uint32\_t data) /\* 发送成功后的回调函数 \*/

)

{

GdmaReqEntry Entry;

Entry.Src= (Src & 0x1FFFFFFF);

Entry.Dst= (Dst & 0x1FFFFFFF);

Entry.TransCount = TransCount;

Entry.SrcBurstMode=INC\_MODE; /\* 增加模式，每次读取数据后自动加1 \*/

Entry.DstBurstMode=FIX\_MODE; /\* 固定模式，每次写入之后，地址不变 \*/

Entry.BurstSize=BUSTER\_SIZE\_4B;

Entry.SrcReqNum=DMA\_MEM\_REQ;

Entry.DstReqNum=DMA\_I2S\_TX\_REQ;

Entry.DoneIntCallback=DoneIntCallback;

Entry.UnMaskIntCallback=UnMaskIntCallback;

Entry.SoftMode=0; /\* 使用硬件模式 \*/

Entry.ChMask=1;

Entry.CoherentIntEbl=0;

if(TxNo==0) { //TX0

//enable chain feature

Entry.ChNum=GDMA\_I2S\_TX0;

Entry.NextUnMaskCh= (TransCount==4) ? 0 : GDMA\_I2S\_TX1;

}else if(TxNo==1) { //TX1

//enable chain feature

Entry.ChNum=GDMA\_I2S\_TX1;

Entry.NextUnMaskCh= (TransCount==4) ? 0 : GDMA\_I2S\_TX0;

}else {

GDMA\_PRINT("I2S Tx Number %x is invalid\n", TxNo);

return 0;

}

return \_GdmaReqEntryIns(&Entry);

}

\_GdmaReqEntryIns函数

int \_GdmaReqEntryIns(GdmaReqEntry \*NewEntry)

{

/\* 将回调的函数加入数组中保存，以便在中断中回调 \*/

if(NewEntry->UnMaskIntCallback!=NULL) {

Data |= (0x01<<CH\_UNMASKINT\_EBL\_OFFSET);

GdmaUnMaskIntCallback[NewEntry->ChNum] = NewEntry->UnMaskIntCallback;

}

/\* 将回调的函数加入数组中保存，以便在中断中回调 \*/

if(NewEntry->DoneIntCallback!=NULL) {

Data |= (0x01<<CH\_DONEINT\_EBL\_OFFSET);

GdmaDoneIntCallback[NewEntry->ChNum] = NewEntry->DoneIntCallback;

}

return 1;

}

DMA中断函数分析：

irqreturn\_t GdmaIrqHandler(int irq, void \*irqaction)

{

u32 Ch=0;

u32 flags;

spin\_lock\_irqsave(&gdma\_int\_lock, (unsigned long)flags);

for(Ch=0;Ch<MAX\_GDMA\_CHANNEL;Ch++) {

/\*判断该通道是否被屏蔽了，如果有回调函数则调用它的回调函数\*/

if(GdmaUnMaskStatus & (0x1 << (UNMASK\_INT\_STATUS(Ch))) ) {

if(GdmaUnMaskIntCallback[Ch] != NULL) {

GdmaUnMaskIntCallback[Ch](Ch);

}

}

}

//write 1 clear

GDMA\_WRITE\_REG(RALINK\_GDMA\_UNMASKINT, GdmaUnMaskStatus);

//processing done

for(Ch=0;Ch<MAX\_GDMA\_CHANNEL;Ch++) {

if(GdmaDoneStatus & (0x1<<Ch)) {

if(GdmaDoneIntCallback[Ch] != NULL) {

GdmaDoneIntCallback[Ch](Ch);

}

}

//write 1 clear

GDMA\_WRITE\_REG(RALINK\_GDMAISTS, GdmaDoneStatus);

spin\_unlock\_irqrestore(&gdma\_int\_lock, flags);

return IRQ\_HANDLED;

}

**void i2s\_dma\_tx\_handler(u32 dma\_ch)**

{

u32 i2s\_status;

u32 data;

i2s\_status=i2s\_inw(I2S\_INT\_STATUS);

pi2s\_config->tx\_isr\_cnt++;

/\* 没有音频数据 \*/

if(pi2s\_config->tx\_r\_idx==pi2s\_config->tx\_w\_idx)

{

/\* Buffer Empty \*/

memset(pi2s\_config->pPage0TxBuf8ptr, 0, I2S\_PAGE\_SIZE);

GdmaI2sTx((u32)pi2s\_config->pPage0TxBuf8ptr, pi2s\_config->pPage1TxBuf8ptr, 0, I2S\_PAGE\_SIZE, i2s\_dma\_tx\_handler, i2s\_unmask\_handler);

}

goto EXIT;

}

/\* 缓冲区没有被映射 \*/

if(pi2s\_config->pMMAPTxBufPtr[pi2s\_config->tx\_r\_idx]==NULL)

{

MSG("mmap buf NULL [%d]\n",pi2s\_config->tx\_r\_idx);

GdmaI2sTx((u32)pi2s\_config->pPage0TxBuf8ptr, pi2s\_config->pPage1TxBuf8ptr, 0, I2S\_PAGE\_SIZE, i2s\_dma\_tx\_handler, i2s\_unmask\_handler);

goto EXIT;

}

/\* 准备发送下一个缓冲区的数据 \*/

pi2s\_status->txbuffer\_len--;

dma\_sync\_single\_for\_device(NULL, i2s\_mmap\_addr[pi2s\_config->tx\_r\_idx], I2S\_PAGE\_SIZE, DMA\_TO\_DEVICE);

GdmaI2sTx((u32)(pi2s\_config->pMMAPTxBufPtr[pi2s\_config->tx\_r\_idx]), pi2s\_config->pPage0TxBuf8ptr, 0, I2S\_PAGE\_SIZE, i2s\_dma\_tx\_handler, i2s\_unmask\_handler);

pi2s\_config->dmach = GDMA\_I2S\_TX0;

pi2s\_config->tx\_r\_idx = (pi2s\_config->tx\_r\_idx+1)%MAX\_I2S\_PAGE;

EXIT:

wake\_up\_interruptible(&(pi2s\_config->i2s\_tx\_qh));

return;

}

缓冲区分为三种情况：

1. 缓冲区的数据为空
2. 缓冲区没有申请
3. 缓冲区有数据，准备下个缓冲区数据的发送

void i2s\_unmask\_handler(u32 dma\_ch)

{

MSG("i2s\_unmask\_handler ch=%d\n",dma\_ch);

if((dma\_ch==GDMA\_I2S\_TX0)&&(pi2s\_config->bTxDMAEnable))

{

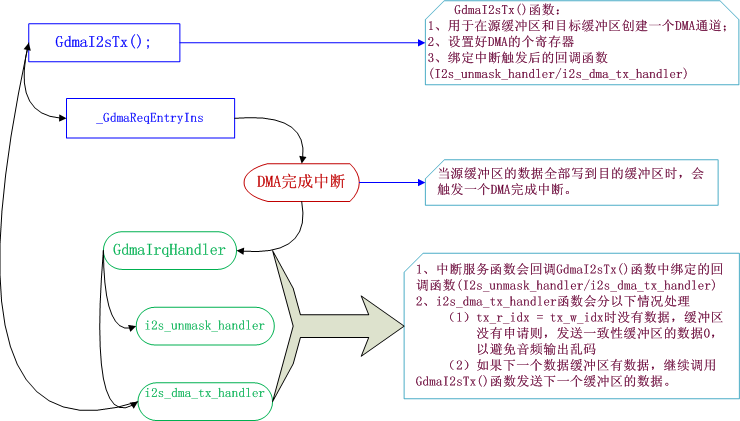
GdmaI2sTx((u32)pi2s\_config->pPage0TxBuf8ptr, pi2s\_config->pPage1TxBuf8ptr, 0, I2S\_PAGE\_SIZE, i2s\_dma\_tx\_handler, i2s\_unmask\_handler);

GdmaUnMaskChannel(GDMA\_I2S\_TX0);

}

}

### 7.1 GdmaI2sTx()函数调用图：



关于声道选择：

#define CONFIG\_I2S\_WS\_EDGE 1

以上的宏是定义音频数据的声道的，在主模式上，如果使用双声道的话一定要注释掉，不然会导致LRCLK线一致为低电平。