Freescale Semiconductor
应用笔记

Document Number: AN4520
Rev 0, 5/2012

I2S（Inter-IC Sound 总线）在 Kinetis 上的应用

面向 K60 的 I2S 驱动器

作者: Guo Jia
Automotive and Industrial Solutions Group

内容

1 简介

本应用笔记是针对新用户的快速指南，旨在说明如何将 Kinetis 上的 I2S 模块用作 Inter-IC 音频总线。

另外，还将讨论基于 DMA 和中断的乒乓缓冲区方案，该方案旨在降低用于处理音频数据流的 CPU 开销。最后将举例说明如何在每个通道上播放频率不同的两个正弦波，以供大家参考。

2 概述

Kinetis 上的 I2S 模块有下列五种基本工作模式:

- 常规模式
- 网络模式
- 门控时钟模式
- I2S 模式
- AC97 模式

本应用笔记只讨论 I2S（Inter-IC Sound 总线规范）模式。I2S 时序如图 1 所示。

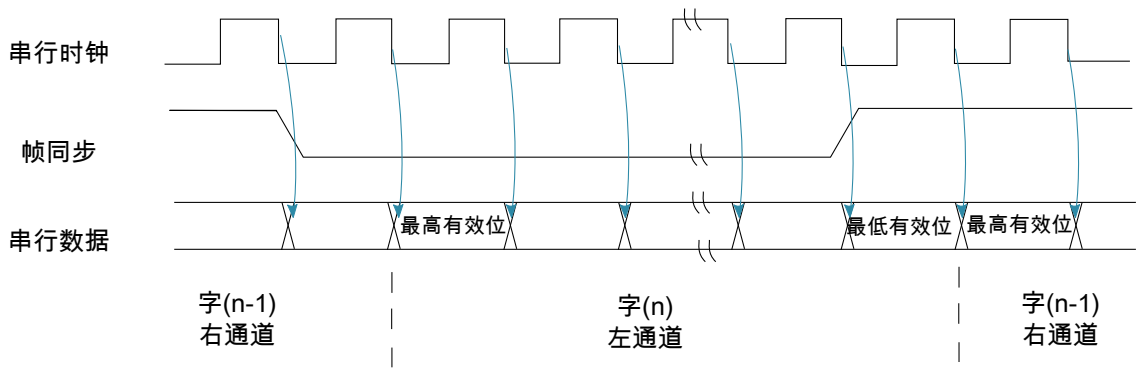

图 1. I2S 协议时序

从应用角度来看，由于采样率一般在 8 KHz 至 48 KHz 之间，因此，如果 CPU 需要直接处理每个中断，则系统效率将非常低。另一方面，多数音频算法会处理数据块，即系统会累积音频流中的数据采样以形成缓冲数据块。随后这些数据块会作为输入或输出供音频算法处理。请参见图 2 。

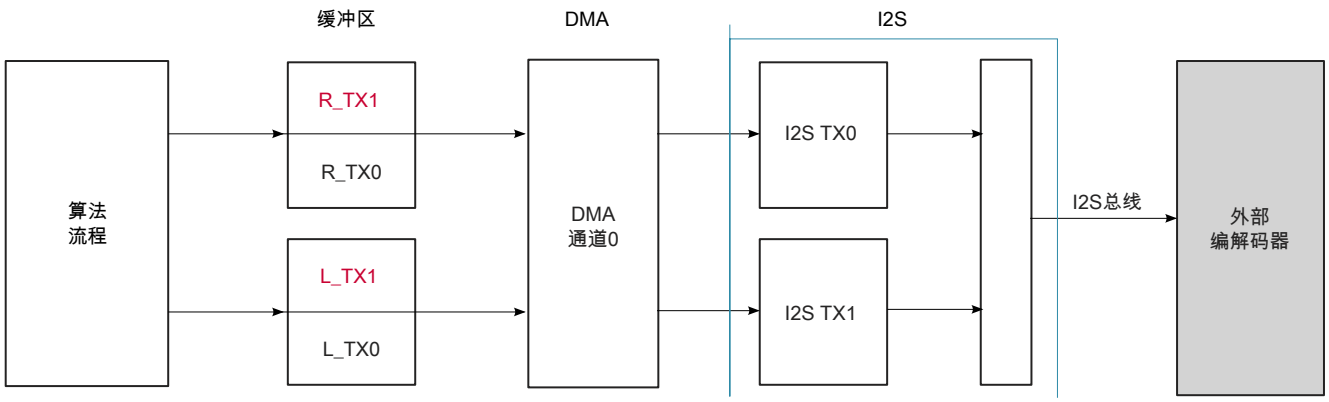

图 2. 基于 DMA 和中断的乒乓缓冲区设计

注

在缓冲区名称中:

- R/L 表示右通道/左通道
- TX 表示发送
- 0/1 表示乒乓缓冲区变址

共有四个缓冲区数据块，建议使所有数据块处于连续的物理空间中以便实现。

I2S 接口中有两个通道——左通道和右通道。每个通道都具有作为乒乓缓冲区的两个数据块。DMA 处理其中一个数据块时，CPU 处理另一个数据块。当前数据块处理完成时，DMA 和 CPU 将交换刚处理过的缓冲区。在图 2 中，四个数据块分成标为红色和黑色的两组。DMA 使用红色数据块时，CPU 使用黑色数据块。类似地，如果 DMA 使用黑色数据块，则 CPU 使用红色数据块。

系统运行时，DMA 传送数据。根据应用要求，假设 N 为某个缓冲区数据块中的采样数。传送了 N 个采样后，DMA 将向 CPU 生成一个中断。

由于所有这些传送都是同步进行的，或者在某个中断期间发生，因此，可以操作两个数据块。在该中断中，CPU 必须完成下列任务:

1. 执行音频解码算法以获得输出数据。
2. 将输出数据填入传送数据块。依据当前乒乓缓冲区变址的不同，可能是 BLOCK0 + BLOCK1，也可能是 BLOCK1 + BLOCK3。

注

由于音频信号具有较强的实时性要求，因此，所有计算都必须在下个中断发生之前完成，否则会导致系统故障。

3 I2S 模块配置

3.1 选择时钟源

为使用 I2S 模块，首先需为该模块配置时钟。如果 I2S 充当主机，则该模块的时钟源必须通过设置 SIM_SOPT2 寄存器中的 I2SSRC 字段（或 SOPT2[I2SSRC]）来决定。可能的选项包括：

- 内核/系统时钟除以 I2S 小数时钟分频器
- MCGPLLCLK/MCGFLLCLK 时钟除以 I2S 小数时钟分频器
- OSCERCLK 时钟
- 外部旁路时钟(I2S_CLK_IN)

用户可根据需要从上述列表中选择一项。

3.2 根据应用要求设置时钟

假设将内核/系统时钟用作 I2S 模块时钟源，则可通过下列公式计算 I2S 模块的时钟：

$$I2S\ \text{clock} = \text{Core}\,/\,\text{system clock} * \frac{(I2SFRAC+1)}{(I2SDIV+1)}$$

注

在上面给出的公式中，I2SFRAC 和 I2SDIV 在寄存器 SIM_CLKDIV2 中定义。I2SFRAC 由 8 位构成，范围为 0 至 255，I2SDIV 有 12 位，范围为 0 至 4095。

现在我们讨论根据当前应用要求设置 I2SFRAC 和 I2SDIV 的方法。

应用中一些典型的音频流采样率可以分成两组。第一组包括 11,025 (44100⁄4)、22,050 (44100⁄2)、44,100，第二组包括 8000 (48000⁄6)、12,000 (48000⁄4)、16,000 (48000⁄3)、24,000 (48000⁄2)、32,000 (48000 * 2⁄3)和 48,000。

根据所使用的采样率，将 I2S 时钟配置为 44,100 或 96,000(48000 * 2)的倍数。因此，建议在第一组中，将从内核/系统时钟获得的 I2S 模块时钟设为 11.2896 MHz (44.1 KHz * 256)，在第二组中，将 I2S 时钟设为 12.288 MHz (48 KHz × 256)。请参见表 1 。

表 1. 针对不同组别建议采用的 I2S 时钟

项目	第一组	第二组
典型采样率 1 (Hz)	11025	8000
典型采样率 2 (Hz)	22050	12000
典型采样率 3 (Hz)	44100	16000
典型采样率 4 (Hz)		24000
典型采样率 5 (Hz)		32000
典型采样率 6 (Hz)		48000
建议采用的 I2S 时钟(MHz)	**11.2896 (44.1 KHz * 256)**	**12.288(48 KHz * 256)**

下面的讨论展示比特率的配置方式。

比特率、采样率和 I2S 时钟的关系可通过下列公式计算得出：

$$\text{比特率} = \text{采样率} * \text{字数} * \text{字长}$$

$$= \text{I2S时钟} / (\text{DIV2} * \text{PSR} * \text{PM} * 2)$$

注

- 在主模式下，字长固定为 32。字中的有效数据位数是可以设置的，但在计算比特率时，应该使用固定长度 32。
- 在上面给出的公式中：
  - DIV2、PSR 和 PM 在寄存器 I2Sx_RCCR 中定义。
  - DIV2 可以是 1（旁路）或 2
  - PSR 可以是 1（旁路）或 8
  - PM 的范围为 1 至 256

在应用中，采样率、字数和位长都是已知的。实际上，在主机模式下，每个采样的位长始终保持为 32。这样就可得到比特率。

如果 PM 大于 256，假设 DIV2=2 且 PSR=8，结果将使 PM 的工作范围缩小。

例如，假设：

- I2S 时钟 = 12.888 MHz
- 字数 = 2（左通道和右通道）
- 字长 = 16

则比特率 = 48 * 2* 32 KHz = 3.072 MHz

假设 DIV2 为 1（旁路）且 PSR 为 1（旁路），则：

PM = I2S 时钟/（比特率 * 2）= 12.288/3.072 MHz = 4 MHz。

3.3 I2S 的 FIFO 特性

为了在图 2 中实现该系统，有必要了解 I2S 模块的 FIFO 特性。尽管 FIFO 深度为 15，数据宽度为 32 位，但根据当前配置仅 24 位有效。传送时，FIFO 中的空数据计数可触发 DMA。这是实现本系统的一个重要特性。FIFO 中的数据交替发送至左右通道。

将空数据计数设为 2，可在目标地址固定不变的情况下，使 DMA 每一次都将一条数据载入左通道，一条数据载入右通道。

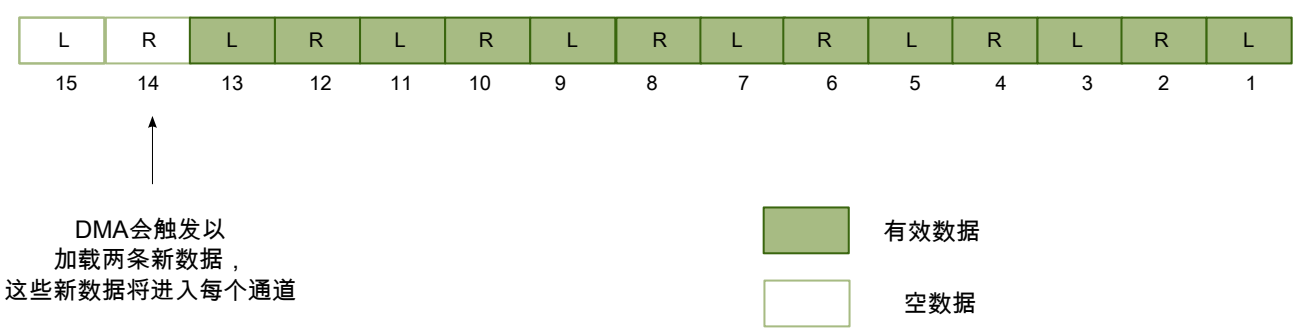

图 3. 在 FIFO 中有两条空数据时触发 DMA

4 DMA 和中断配置

每次都由 DMA 从缓冲区传送每个音频通道的某个采样点。当数据块中的所有数据均发出时，会生成一个中断。本节旨在展示如何通过只配置一个 DMA 通道来实现这一目的。

Kinetis 上的 DMA 采用了一个双层循环：

- 次循环：当发生 DMA 事件，即 FIFO 中存在两条或两条以上空数据时，次循环触发。根据帧中定义的有效位数，次循环传输计数为 1、2 或 4 个字节。它还取决于移位方向，可以使最高有效位或最低有效位。用户可查看图 4 中的数据对齐描述，以确定移位方向。
- 主循环：主循环在次循环完成时触发。主循环可在循环结束或循环完成一半时生成中断。请参见图 5 。这一特性极为重要，可使乒乓缓冲区的实现变得更为容易。主循环计数可设为一个数据块中所需采样数的两倍。图 5 所示为 DMA 实现乒乓缓冲区的方式。

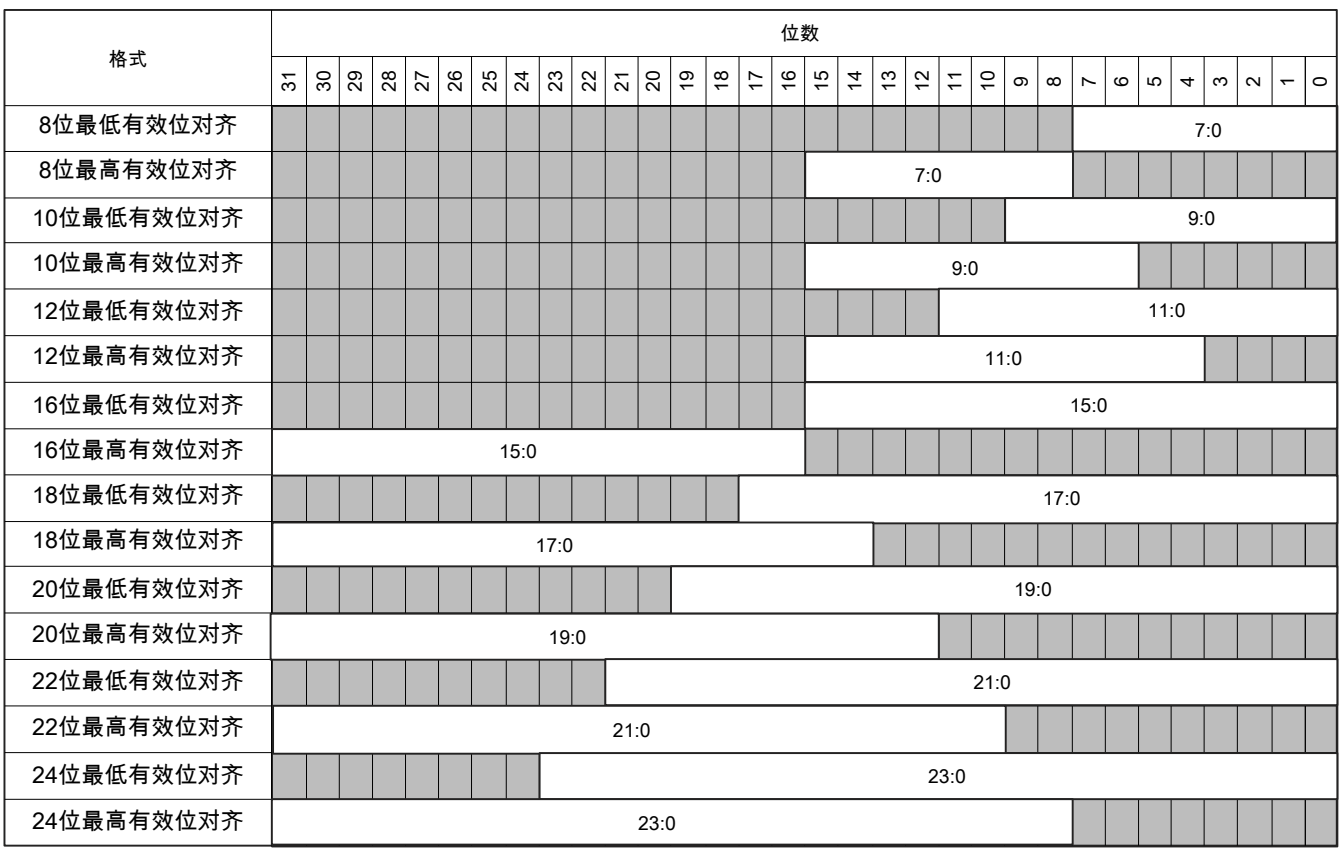

图 4. 数据对齐

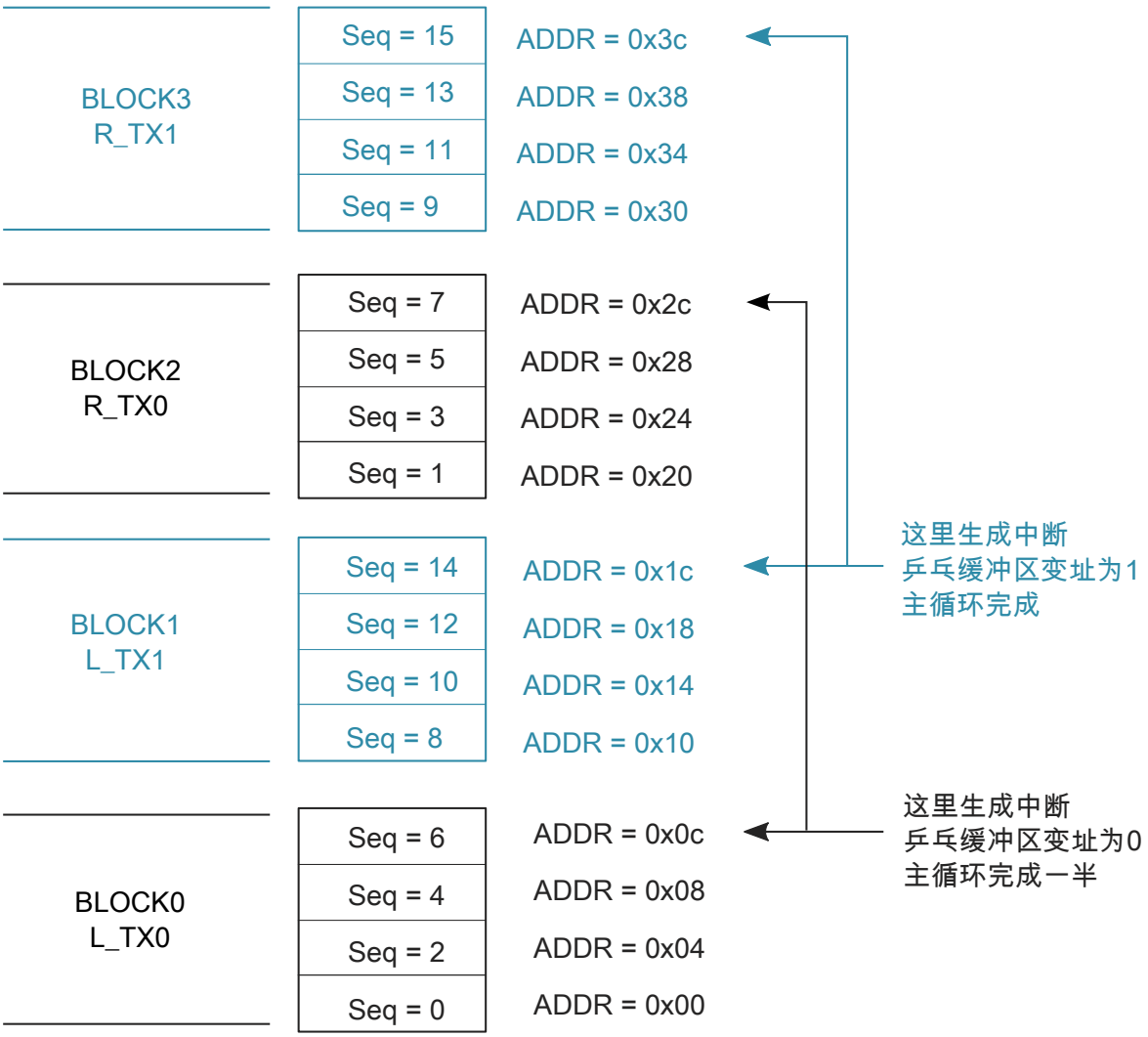

图 5. 循环完成和完成一半时中断

在图 5 中，每个数据块都有四个采样，每个采样都有四个字节。DMA 操作数据的序列由 Seq 值给定。可以看出，根据 Seq 值，DMA 所访问地址的顺序是 0x00、0x20、0x04、0x24…。

1. 当 DMA 到达地址 0x2c 时，会生成一个中断。
2. 随后，CPU 从现在开始一直到 DMA 到达地址 0x3c 之前，可以将数据填入 BLOCK0 和 BLOCK2 中，在此期间，DMA 处理 BLOCK1 和 BLOCK3。
3. 当 DMA 到达地址 0x3c 时，会生成另一个中断。
4. 随后，CPU 从现在开始一直到 DMA 到达地址 0x2c 之前，可以将数据填入 BLOCK1 和 BLOCK3 中，在此期间，DMA 处理 BLOCK0 和 BLOCK2。

出于一般考虑，假设每个数据块有 N 个采样，每个采样有 L 个字节。则通过下列步骤可初始化 DMA 模块:

1. 次循环偏移量使能且只应用于源地址，且值为 L – [(N *2) * (L*2)]。
2. 次循环传输计数设为 L*2。
3. 源地址初始化至缓冲区基地址。
4. 源地址偏移量在每次写操作后设为 N* L*2。
5. 主循环结束的源地址偏移量设为–[(N*2) * (L*3) – L]。
6. 目标地址初始化并固定到 I2S_TX0 寄存器。
7. 目标地址偏移量在每次写操作后设为 0。
8. 主循环结束的目标地址偏移量设为 0。
9. 主循环计数设为 N*2。
10. 使能半满中断。

注

地址偏移量可为负值。

5 正弦波播放示例

根据 DMA 和中断配置 中的描述，在每个声道上播放不同频率的正弦波。该例是在塔板上实现的，采用的是 Freescale 塔式系统，搭载 K60 卡和声卡。

5.1 正弦波生成

所选频率必须符合某些要求以避免出现动态正弦波计算问题。设正弦波频率为 f，则周期为 T = 1/f。设采样率为 fs，则采样时间为 Ts = 1/fs。有两项要求：

- T/Ts 必须为整数。如果符合本条件，无需计算就可用一个周期的采样值来产生后面周期中的所有正弦波值。
- 如果选择的两个频率为 f1 和 f2，则建议使 f1/f2 为整数。这并非强制要求，但有助于简化问题。

在本例中：

- 采样率设为 32 KHz。
- 正弦波频率设为 200 Hz 和 1000 Hz。

由于每个采样都有一个 24 位值，因此，这 24bit 实际占用 4 字节空间，这样可以构成 4 字节边界对齐的数组。缓冲区总大小为：160 *4 *2 = 1280 字节。

5.2 I2S 初始化

在本例中，I2S 初始化包括三个步骤：

1. 配置引脚多路复用。
2. 配置时钟和比特率。
3. 配置 I2S 模块特性。

```
void hal_i2s_init(void)
{
    _i2s_io_init();
    _i2s_set_rate(32000);
    _i2s_init();
}
```

每一步均以下列代码实现：

```
static void _i2s_io_init(void)
{
    PORTE_PCR6  &= PORT_PCR_MUX_MASK;
    PORTE_PCR7  &= PORT_PCR_MUX_MASK;
    PORTE_PCR10 &= PORT_PCR_MUX_MASK;
    PORTE_PCR11 &= PORT_PCR_MUX_MASK;
    PORTE_PCR12 &= PORT_PCR_MUX_MASK;

    PORTE_PCR6  |= PORT_PCR_MUX(0x04);
    PORTE_PCR7  |= PORT_PCR_MUX(0x04);
    PORTE_PCR10 |= PORT_PCR_MUX(0x04);
    PORTE_PCR11 |= PORT_PCR_MUX(0x04);
PORTE_PCR12 |= PORT_PCR_MUX(0x04);
}
 static void _i2s_set_rate(int smprate)
{
    unsigned char pm_val, dc_val;
```

```
    if((smprate == 11025)||(smprate == 22050)||(smprate == 44100))
        _set_clock_112896();

    if((smprate == 8000) || (smprate == 12000) || (smprate == 16000) ||
       (smprate == 24000)|| (smprate == 32000) || (smprate == 48000) )
        _set_clock_122800();

    switch(smprate)
    {
        case 8000:  pm_val=23; dc_val=1; break;
        case 11025: pm_val=15; dc_val=1; break;
        case 12000: pm_val=15; dc_val=1; break;
        case 16000: pm_val=11; dc_val=1; break;
        case 22050: pm_val=7;  dc_val=1; break;
        case 24000: pm_val=7;  dc_val=1; break;
        case 32000: pm_val=2;  dc_val=1; break;
        case 44100: pm_val=3;  dc_val=1; break;
        case 48000: pm_val=3;  dc_val=1; break;
        default:    pm_val=3;  dc_val=1; break;
    }

    I2S0_TCCR = I2S_TCCR_WL(0xb) | // 24 bit
                I2S_TCCR_DC(dc_val) | I2S_TCCR_PM(pm_val);
}
static void _i2s_init(void)
{
    // diable
    I2S0_CR &= ~I2S_CR_SSIEN_MASK;

    I2S0_CR |=
            //I2S_CR_TCHEN_MASK  |     // Enable two channel mode
            I2S_CR_SYSCLKEN_MASK |  // Set clock out on SSI_MCLK pin, SRCK PORT
            I2S_CR_I2SMODE(1)  |    // Set I2S master mode
            I2S_CR_SYN_MASK    |    // Enable synchronous mode
            // I2S_CR_NET_MASK  |     // Enable network mode
            I2S_CR_RE_MASK |      // Enable the receive section, this does not enable
interrupts
            I2S_CR_TE_MASK;         // Enable the transmit section, this does not enable
interrupts

    I2S0_TCR |=
            I2S_TCR_TFDIR_MASK |     // internally generated frame
            I2S_TCR_TXDIR_MASK |     // internally generated clock
            I2S_TCR_TSCKP_MASK |     // sample data at rising edge, data send at falling
            I2S_TCR_TFSI_MASK  |     // frame sync active low
            I2S_TCR_TEFS_MASK  |     // tx data 1 bit delay
            I2S_TCR_TFEN0_MASK;       // enable fifo

    I2S0_TCCR = I2S_TCCR_WL(0xb)   | // 24 bit word length
                // I2S_TCCR_WL(0xb)   | // 16 bit word length
                I2S_TCCR_DC(1)     | //
                I2S_TCCR_PM(3);     //

    I2S0_RCR |=
            I2S_RCR_RXBIT0_MASK | // lsb align
            I2S_RCR_RSCKP_MASK  | // sample data at rising edge, data send at falling
            I2S_RCR_RFSI_MASK   | // Frame sync active low
            I2S_RCR_RFSL_MASK   | // frame sync length is one word long
            I2S_RCR_REFS_MASK;    // 1 bit delay

    I2S0_RCCR = I2S_RCCR_WL(0xb)   | // 24 bit word length
                // I2S_RCCR_WL(0x7)   | // 16 bit word length
                I2S_RCCR_DC(1)     | // dc pm will be configured at later
                I2S_RCCR_PM(3);

    // FIFO, when empty data count is 2, set this flag
    I2S0_FCSR = I2S_FCSR_TFWM0(2);

    // DMA request enabled
    I2S0_IER  = I2S_IER_TDMAE_MASK;
```

```
    // enable ssi
    I2S0_CR |= I2S_CR_SSIEN_MASK;
}
```

5.3 DMA 初始化

以下代码展示了如何配置 DMA 以实现乒乓缓冲区，其中给出了详细的说明。

```
void hal_dma_init_for_i2s(uint buf_rx, uint buf_tx, uint block_n_sample, uint sample_n_byte)
{
    uint size_bit;
    DMA_TCD tcd;

    switch(sample_n_byte)
    {
        case 1:  size_bit = DMA_SIZE_8_BIT;  break;
        case 2:  size_bit = DMA_SIZE_16_BIT; break;
        case 4:  size_bit = DMA_SIZE_32_BIT; break;
        default: size_bit = DMA_SIZE_32_BIT; break;
    }

    SIM_SCGC6     |= SIM_SCGC6_DMAMUX_MASK;
    DMAMUX_CHCFG0 = DMAMUX_CHCFG_ENBL_MASK | DMAMUX_CHCFG_SOURCE(DMA_SRC_I2S_T);

    DMA_CR        |= DMA_CR_EMLM_MASK;
    DMA_CSR(0)     = DMA_CSR_INTHALF_MASK | DMA_CSR_INTMAJOR_MASK;
    nvic_enable_irq(IRQ_DMA0);

    tcd.channel    = 0;
    tcd.nbytes     = DMA_NBYTES_MLOFFYES_SMLOE_MASK |
                     DMA_NBYTES_MLOFFYES_MLOFF(sample_n_byte -
block_n_sample*2*sample_n_byte*2)|
                     DMA_NBYTES_MLOFFYES_NBYTES(sample_n_byte*2);
    tcd.attr       = DMA_ATTR_SSIZE(size_bit) | DMA_ATTR_DSIZE(size_bit);

    tcd.saddr      = buf_tx;
    tcd.soff       = block_n_sample*2*sample_n_byte;
    tcd.slast      = -(block_n_sample*2*sample_n_byte*3 - sample_n_byte);

    tcd.daddr      = (uint)(&I2S0_TX0);
    tcd.doff       = 0;
    tcd.dlast_sga = 0;

    tcd.citer      = block_n_sample*2;
    tcd.biter      = block_n_sample*2;
    _dma_init(&tcd);

    // enable DMA channel
    DMA_SERQ = DMA_SERQ_SERQ(0);
```

5.4 中断服务例程

在中断服务例程中，TX 缓冲区中载有将发送至 I2S 的数据。

```
void hal_fill_tx_buf(s32 *p_r, s32 *p_l, uint buf_n_sample)
{
    static int index = 0;
    static int data_index = 0;
    int i;
    s32 *p_r_tx;
    s32 *p_l_tx;
```

```
    // get buffer pointer
    if(index == 0)
    {
        p_r_tx = (int*)i2s_buf.buf_i2s_r_tx;
        p_l_tx = (int*)i2s_buf.buf_i2s_l_tx;
    }
    else
    {
        p_r_tx = (int*)(i2s_buf.buf_i2s_r_tx+I2S_BLOCK_N_SAMPLES*I2S_SAMPLE_N_BYTE);
        p_l_tx = (int*)(i2s_buf.buf_i2s_l_tx+I2S_BLOCK_N_SAMPLES*I2S_SAMPLE_N_BYTE);
    }

    // set content in the buffer
    for(i=0;i<I2S_BLOCK_N_SAMPLES;i++)
    {
        *p_r_tx++ = p_r[data_index];
        *p_l_tx++ = p_l[data_index];
        data_index++;
        if(data_index >= buf_n_sample)
            data_index = 0;
    }
    index ^= 1;
}
```

6 结语

归纳起来，本应用笔记讨论了：

- 有关使用 Kinetis 上 I2S 和 DMA 模块的一些基本概念
- 只用一个 DMA 通道为两个 I2S 音频通道实现乒乓缓冲区的方法
- 播放具有两个不同频率的正弦波的例子。

除这些功能外，Kinetis 上的 I2S 模块还有另一个重要特性，即可在网络模式下工作，通过时分多路复用总线连接 DSP 或其他音频器件。

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

本文档中的信息仅供系统和软件实施方使用 Freescale 产品。 本文并未明示或者暗示授予利用本文档信息进行设计或者加工集成电路的版权许可。 Freescale 保留对此处任何产品进行更改的权利，恕不另行通知。

Freescale 对其产品在任何特定用途方面的适用性不做任何担保、表示或保证，也不承担因为应用程序或者使用产品或电路所产生的任何责任，明确拒绝承担包括但不局限于后果性的或附带性的损害在内的所有责任。 Freescale 的数据表和/或规格中所提供的“典型”参数在不同应用中可能并且确实不同，实际性能会随时间而有所变化。 所有运行参数，包括“经典值”在内，必须经由客户的技术专家对每个客户的应用程序进行验证。 Freescale 未转让与其专利权及其他权利相关的许可。 Freescale 销售产品时遵循以下网址中包含的标准销售条款和条件：freescale.com/SalesTermsandConditions.

Document Number AN4520
Revision 0, 5/2012