Freescale Semiconductor
Application Note

Document Number: AN4520
Rev. 0, 5/2012

An I2S (Inter-IC Sound Bus) Application on Kinetis

I2S Driver for K60

by: Guo Jia
Automotive and Industrial Solutions Group

Contents

1 Introduction....1
2 Overview....1
3 I2S module configuration....3
3.1 Select clock source....3
3.2 Set clock according to application requirement....3
3.3 I2S FIFO feature....4
4 DMA and interrupt configuration....5
5 Example of playing sine wave....8
5.1 Sine wave generation....8
5.2 I2S initialization....8
5.3 DMA initialization....10
5.4 Interrupt service routine....10
6 Conclusion....11

1 Introduction

This application note is a quick start guide on how to use the I2S module as inter-IC sound bus on Kinetis, for the new users. In addition, DMA- and interrupt-based ping-pong buffer scheme is also discussed to reduce the CPU cost for processing the audio data stream. Finally, an example of playing two sine waves of different frequency on each channel is shown for reference.

2 Overview

The I2S module on Kinetis has the following five basic operating modes:

- Normal mode
- Network mode
- Gated clock mode
- I2S mode
- AC97 mode

In this application note, only I2S (Inter-IC Sound bus specification) mode is discussed. The I2S timing is shown in Figure 1.

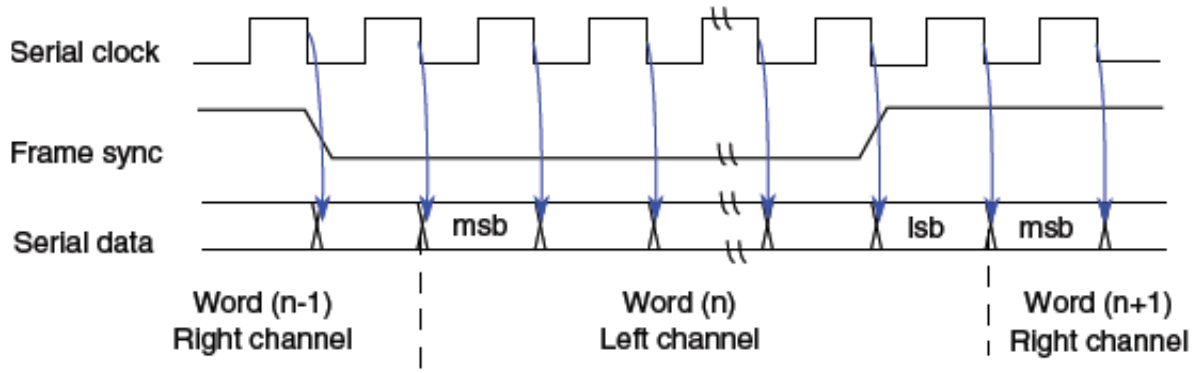

Figure 1. I2S protocol timing

From viewpoint of application, as sample rate generally ranges from 8 KHz to 48 KHz, the efficiency of the system will be very low, if CPU processes each interrupt directly. On the other hand, most of the audio algorithms process data block, that is, the system accumulates data samples in the audio stream to form buffered blocks. Then the blocks are used as input or output for audio algorithm processing. See Figure 2.

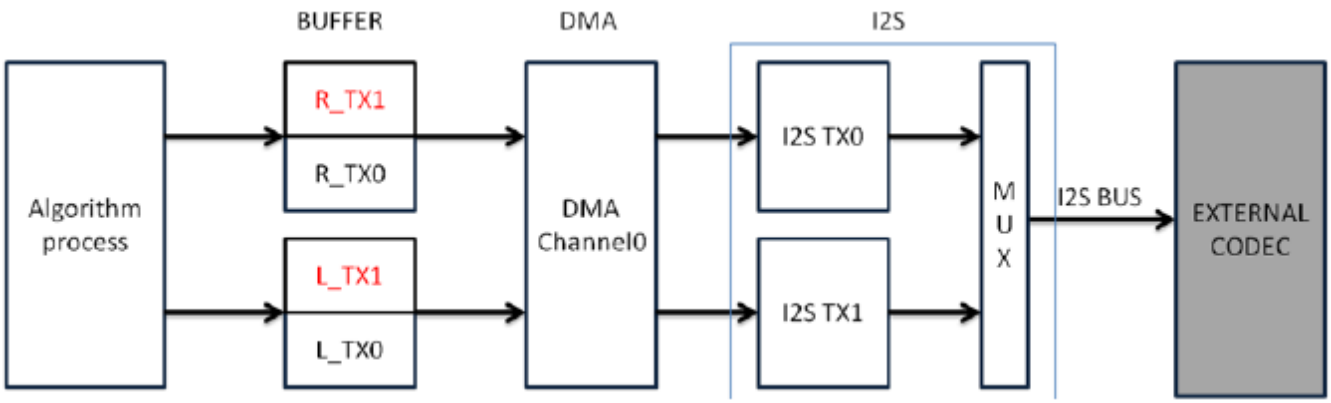

Figure 2. DMA and interrupt based ping-pong buffer design

NOTE

In the buffer name:

- R/L means Right/Left channel
- TX means Transmit
- 0/1 means the ping-pong index

There are total four buffer blocks and it is recommended that all the blocks be continuous in physical space for implementation.

There are two channels in the I2S interface—the left and right channel. Each channel has two blocks working as ping-pong buffer. When DMA is processing one, the CPU is processing the other one. When the current block processing is over, DMA and CPU will exchange the buffer that was just manipulated. In Figure 2, the four blocks are divided into two groups marked red and black. When DMA is using the red blocks, the CPU is using the black blocks. Likewise, if the DMA is using the black blocks, the CPU is using the red block.

When the system is running, DMA is transmitting data. According to the application requirement, it is assumed that N is the sample count in one buffer block. When N samples are transmitted, DMA will generate an interrupt to CPU.

As all these transmissions are synchronous, or occur during one interrupt, two blocks can be operated. In this interrupt, the CPU must finish the following tasks:

1. Execute audio decoding algorithm to get the output data.
2. Fill the output data to transmission blocks. Depending on the current ping-pong index, it could be BLOCK0 + BLOCK1, or BLOCK1 + BLOCK3.

NOTE

As audio signal has strong real-time requirement, all the computation must be finished before next interrupt occurs, or, this may cause system failure.

3 I2S module configuration

3.1 Select clock source

In order to use the I2S module, first of all, configure the clock for this module. If I2S is working as a master, the clock source of this module must be decided by setting the I2SSRC field in the SIM_SOPT2 register, or SOPT2[I2SSRC]. The possible options include:

- Core/system clock divided by the I2S fractional clock divider
- MCGPLLCLK/MCGFLLCLK clock divided by the I2S fractional clock divider
- OSCERCLK clock
- External bypass clock (I2S_CLK_IN)

The user can choose one from the above list according to the requirement.

3.2 Set clock according to application requirement

Assuming that the core/system clock is used as the I2S module clock source, I2S module clock can be calculated by the following formula:

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

NOTE

In the formula given above, I2SFRAC and I2SDIV are defined in the register SIM_CLKDIV2. I2SFRAC consists of 8 bits, ranging from 0 to 255, and I2SDIV has 12 bits, ranging from 0 to 4095.

Now, the method to set I2SFRAC and I2SDIV according to current application requirement, is discussed.

Two groups can be formed considering some typical sample rate of audio streams in application. The first group includes 11,025 (44100⁄4), 22,050 (44100⁄2), 44,100, and the second group includes 8000 (48000⁄6), 12,000 (48000⁄4), 16,000 (48000⁄3), 24,000 (48000⁄2), 32,000 (48000 * 2⁄3), and 48,000.

Based on the sample rate being used, configure I2S clock to be multiple of 44,100 or 96,000(48000 * 2). So it is recommended that for the first group, I2S module clock obtained from core/system clock be set to 11.2896 MHz (44.1 KHz * 256) and for the second group, I2S clock be set to 12.288 MHz (48 KHz × 256). See Table 1.

Table 1. Recommended I2S clock for different groups

Items	Group1	Group2
Typical sample rate1 (Hz)	11025	8000
Typical sample rate2 (Hz)	22050	12000
Typical sample rate3 (Hz)	44100	16000
Typical sample rate4 (Hz)		24000

Table continues on the next page...

Table 1. Recommended I2S clock for different groups (continued)

Items	Group1	Group2
Typical sample rate5 (Hz)		32000
Typical sample rate6 (Hz)		48000
Recommended I2S clock(MHz)	**11.2896 (44.1 KHz * 256)**	**12.288(48 KHz * 256)**

The following discussion shows how to configure the bit rate.

The relationship of bit rate, sample rate, and I2S clock can be calculated by the following formula:

$$\text{Bit rate} = \text{Sample rate} * \text{Word count} * \text{Word length}$$

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

NOTE

- In Master mode, word length is fixed to be 32. The number of valid data bits in the word can be set, but while computing the bit rate, it must be set to 32.
- In the formula given above:
  - DIV2, PSR, and PM are defined in the register I2Sx_RCCR.
  - DIV2 can be 1(bypass) or 2
  - PSR can be 1(bypass) or 8
  - PM ranges from 1 to 256

In application, the sample rate, word count, and bit length in one sample are already known. In fact, in Master mode, bit length per sample is always kept as 32. Therefore, bit rate can be obtained.

If PM is greater than 256, assume DIV2=2 and PSR=8, and this would reduce PM in its working range.

For example, assume that:

- I2S clock = 12.888 MHz
- Word count = 2 (for left and right channel)
- Word length = 16

Now, the bit rate = 48 * 2* 32 KHz = 3.072 MHz

Assuming that DIV2 be 1(bypass), and PSR be 1(bypass), then:

PM = I2S clock /(bit rate * 2) = 12.288/3.072 MHz = 4 MHz.

3.3 I2S FIFO feature

In order to implement the system in Figure 2, it is essential to know the FIFO feature of the I2S module. Though the FIFO depth is 15 and each data width is 32 bits, only 24-bit width is valid according to the current configuration. For transmission, DMA can be triggered by the empty data count in the FIFO. It is a significant feature to implement this system. Data in FIFO is sent to the left and the right channel alternatively.

Set the empty data count to 2, so that each time DMA loads one data to the left channel and one data to the right channel with the destination address fixed.

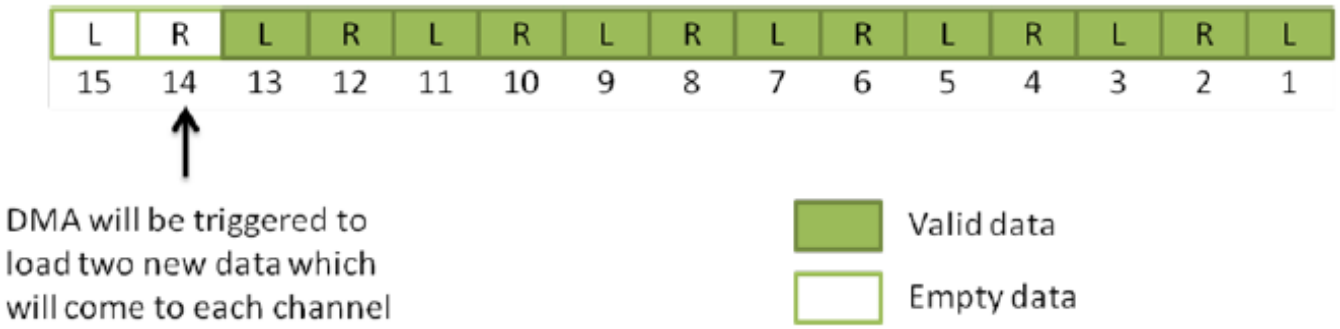

Figure 3. Trigger DMA when there are two empty data in FIFO

4 DMA and interrupt configuration

Each time, one sample of each audio channel is transmitted from the buffer by DMA. When all the data in a block is sent out, an interrupt is generated. This section shows how this can be achieved by configuring only one DMA channel.

The DMA on Kinetis features a two-layer loop:

- Minor loop: Minor loop is triggered when the DMA event occurs in which there are two or more than two empty data in the FIFO. Minor loop transfer count is 1, 2, or 4 bytes, according to the valid bits count defined in the frame. It also depends on the shift direction, which can be MSB or LSB. The user can look into the data alignment description in Figure 4 to determine it.
- Major loop: Major loop is triggered when a minor loop is finished. Major loop can generate interrupt when the loop ends or the loop is half finished. See Figure 5. This is a very important feature which makes it easier to implement ping-pong buffer. The major loop count can be set as twice the sample count needed in one block. Figure 5 shows how the DMA is working to implement it.

Format	Bit number																													
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
8-bit lsb aligned																									7:0							
8-bit msb aligned																	7:0															
10-bit lsb aligned																							9:0									
10-bit msb aligned																	9:0															
12-bit lsb aligned																					11:0											
12-bit msb aligned																	11:0															
16-bit lsb aligned																	15:0															
16-bit msb aligned	15:0																															
18-bit lsb aligned															17:0																	
18-bit msb aligned	17:0																															
20-bit lsb aligned													19:0																			
20-bit msb aligned	19:0																															
22-bit lsb aligned											21:0																					
22-bit msb aligned	21:0																															
24-bit lsb aligned								23:0																								
24-bit msb aligned	23:0																															

Figure 4. Interrupt when loop is finished and half finished

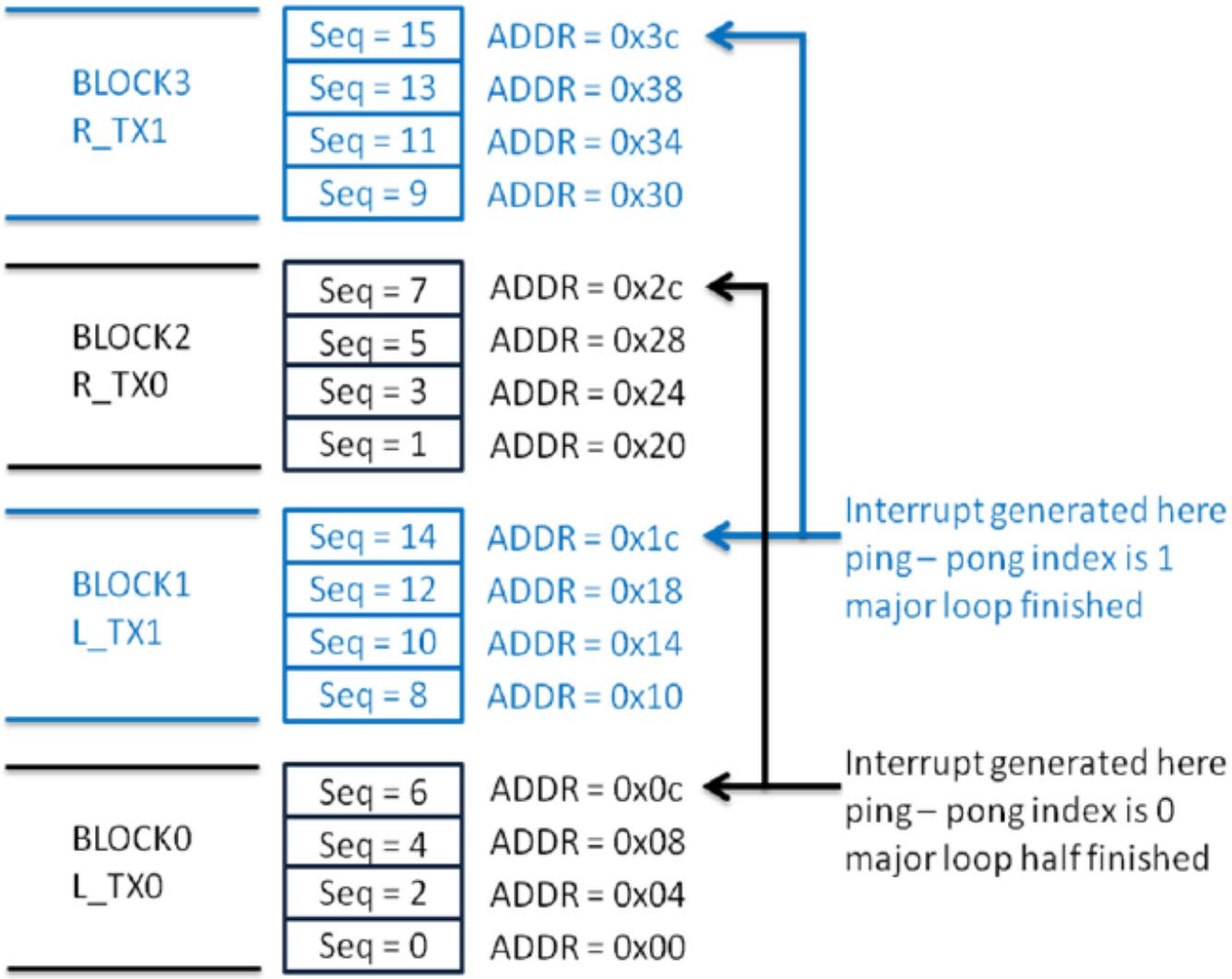

Figure 5. Interrupt when loop is finished and half finished

In Figure 5, each block has four samples and each sample has four bytes. The sequence by which DMA operates data is given by the Seq value. It can be seen that according to the Seq value, the address accessed by DMA is in sequence like 0x00, 0x20, 0x04, 0x24 …, and so on.

1. When DMA reaches address 0x2c, an interrupt is generated.
2. Then, CPU can fill data to BLOCK0 and BLOCK2 from now on until DMA reaches the address 0x3c, during which DMA process BLOCK1 and BLOCK3.
3. When DMA reaches the address 0x3c, another interrupt is generated.
4. Then, CPU can fill data to BLOCK1 and BLOCK3 from now on until DMA reaches the address 0x2c during which DMA process BLOCK0 and BLOCK2.

For general consideration, assume that each block has N samples, and each sample has L bytes. Then, the DMA module can be initialized by the following steps:

1. Minor loop offset is enabled and it is applied to source address only and the value is L – [(N *2) * (L*2)].
2. Minor loop transfer count is set to L*2.
3. Source address is initialized to buffer base address.
4. Source address offset after each write is set to N* L*2.
5. Source address offset for the end of major loop is set to –[(N*2) * (L*3) – L].
6. Destination address is initialized and fixed to the I2S_TX0 register.
7. Destination address offset after each write is set to 0.
8. Destination address offset for the end of major loop is set to 0.

9. Major loop count is set to N*2.
10. Enable half interrupt.

NOTE

The address offset can be negative.

5 Example of playing sine wave

According to the description in DMA and interrupt configuration, here is an example of playing sine waves with different frequency on each channel. This example is implemented on the tower board with the K53 card and audio card in the Freescale tower system.

5.1 Sine wave generation

The frequencies which are chosen must meet some requirement to avoid dynamic sine value computation. Let the sine wave frequency be f, then, the period is T = 1/f. Let the sample rate be fs, then the sample time is Ts = 1/fs. There are two requirements:

- T/Ts must be an integer. When this condition is met, the sample values of one period can be used to generate all the sine wave values in later periods without computation.
- If the two frequencies chosen are f1 and f2, then f1/f2 is suggested to be an integer. This requirement is not mandatory, but will help to make a simpler way.

In this example:

- Sample rate is set to 32 KHz.
- The sine wave frequencies are set to 200 Hz and 1000 Hz.

As each sample has a 24-bit value, so it actually takes a 4-byte memory to make an aligned array. Total buffer size is: 160 *4 *2 = 1280 bytes.

5.2 I2S initialization

In this example, I2S initialization include three steps:

1. Configure pin multiplex.
2. Configure clock and bit rate
3. Conifgure the I2S module feature.

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

Each step is implemented in the following code:

```
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 initialization

The following code shows how to configure DMA to implement ping-pong buffer with detailed comments.

```
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 Interrupt service routine

In the interrupt service routine, TX buffer is loaded with new data to be sent to I2S.

```
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 Conclusion

To summarize, this application note discusses:

- Some basic concepts of using I2S and DMA module on Kinetis
- The implementation of ping-pong buffer with only one DMA channel for two I2S audio channels
- An example of playing sine waves with two different frequencies.

In addition to these functions, the I2S module on Kinetis has another important feature that it can work in network mode to connect to DSP or other audio device with time-division-multiplex bus.

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

Document Number: AN4520
Rev. 0, 5/2012