Freescale Semiconductor
Application Note

Document Number:AN4800
Rev 0, 09/2013

An I2S (Integrated Interchip Sound Bus) Application on Kinetis

Updated for 2.x Silicon

by: Guo Jia

Contents

1 Introduction

This application note is a supplemental update to *AN4520: An I2S (Inter-IC) Application on Kinetis*, (available on freescale.com) which introduced how to use the I2S module on Kinetis K60 of 1.x silicon. But in 2.x silicon, the I2S module has changed to a large extent, and the previous code is now no longer working.

This application note introduces the I2S module on Kinetis of 2.x silicon, and guides the customers to learn this module quickly. Some of the concepts already covered in AN4520 are not repeated here.

2 Configuration of bit clock

To use the I2S module, first of all, it needs to be configured correctly to get the desired bit clock. In this demo application (see the following figure), the desired frequency of 12.228 MHz must be obtained.

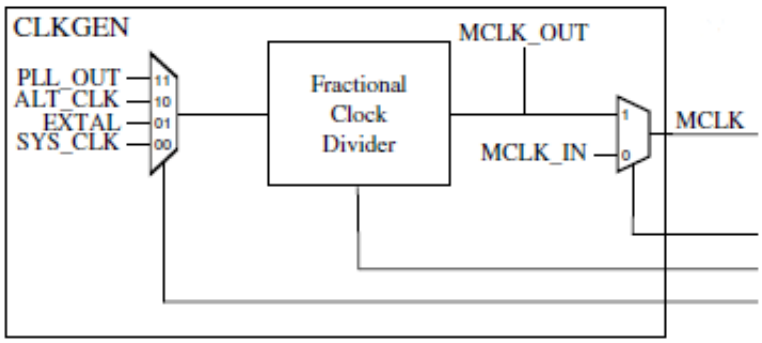

Figure 1. Set master clock (MCLK)

In this demo, if the core clock is 48 MHz, then the output clock frequency can be obtained using the following equation.

output = input *[(I2SFRAC+1) / (I2SDIV+1)] = (48M* (32/125)) = 12.288 MHz

This code line is used to set the clock frequency.

```
I2S0_MDR = I2S_MDR_FRACT(31) | I2S_MDR_DIVIDE(124);
```

After this, the bit clock rate must be set. Bit clock is generated by dividing MCLK, see this figure.

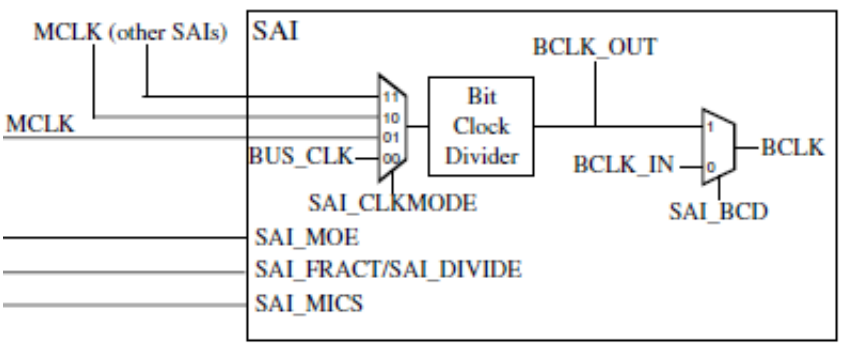

Figure 2. Select bit clock (BLCK)

For sample rate as 32 kHz, each sample has 24 bits in one channel, channel count as 2 (left channel and right channel); therefore the bit clock can be calculated using this equation.

Bit clock = sample frequency * words in a frame * bits in a word = 32K*24*2 = 1.536 MHz = 12.288 MHz/8

Thus, MCLK should be divided by 8. After this, compute the value that should be configured to the Transmit Configuration register (I2S_TCR2) by the following formula:

8 = (DIV+1)*2, DIV = 3

This code is used to set bit clock rate.

```
switch(sample_rate)
{
    …
        case 32000: div=3;  break;
    …
}
    I2S0_TCR2 = div;
```

3 How a frame is constructed and how to configure

The following figures depict the PCM frame and I2S frames.

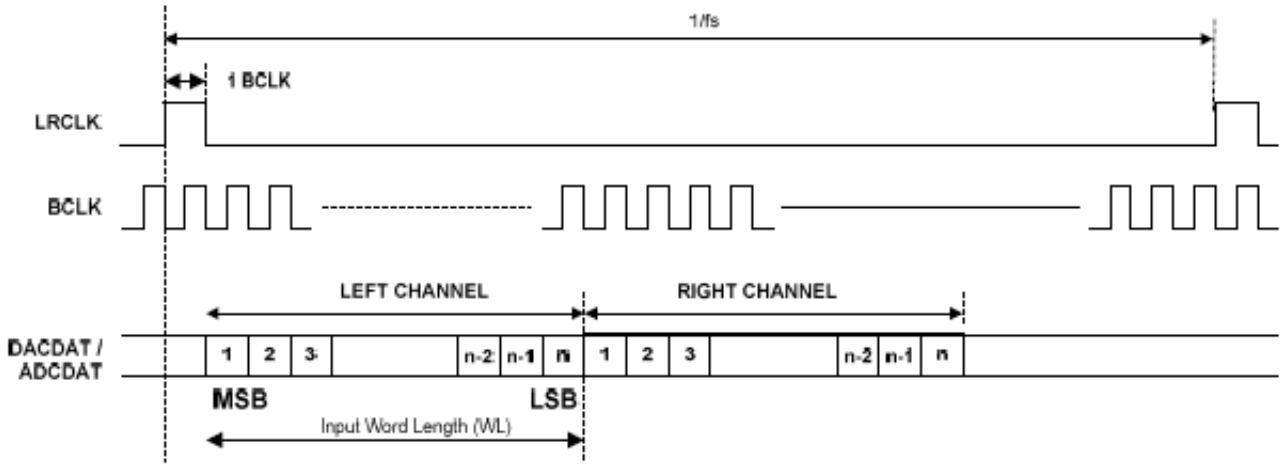

Figure 3. PCM frame

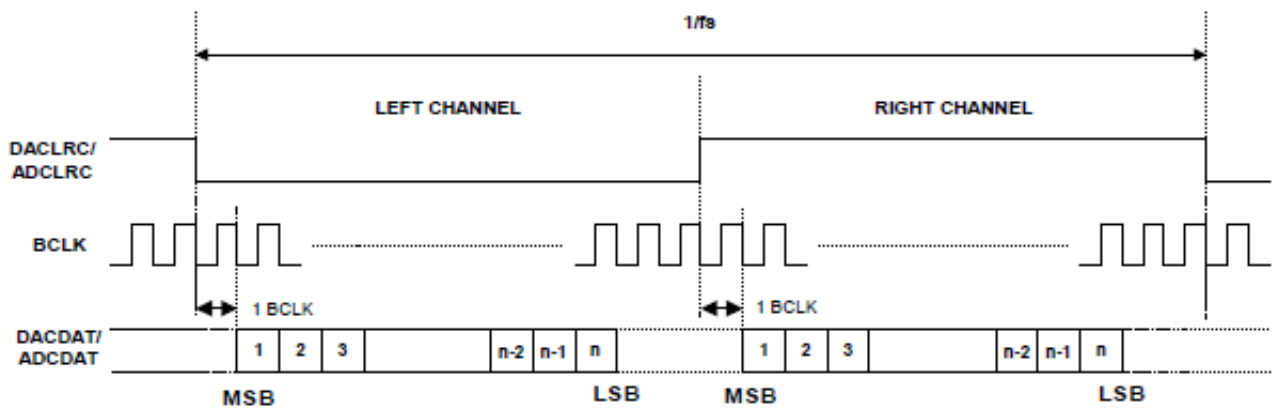

Figure 4. I2S frame

From Figure 3 and Figure 4, it can be seen that a frame is constructed by frame start signal, clock, and data. These are some considerations which must be taken into account, for the PCM and I2S frames.

- Start of a frame on rising-edge or falling-edge and the duration for which the frame active level lasts.
- Transfer of data on rising- or falling-edge of clock
- Number of words in a frame
- Number of bits in a word
- Order of the transfer of most significant and least significant bits
- Need of left- or right-adjustment
- Requirement of 1-bit delay
- External or internal clock source

4 Other configurations

For other configurations, these considerations must be accounted.

1. **Watermar**k: The key point is that when the FIFO Request Flag in the SAI Transmit Control Register (I2Sx_TCSR[FRF]) is set, two data packets should be loaded to FIFO—one for left channel and one for right channel. So, watermark is suggested to be 4 or 6 here.
2. **Sync mode and Async mode**: Here, it is needed to figure out whether the clock and frame signal are generated by the SAI module itself or provided by the external signal source.
3. **Use DMA or interrupt**: As audio signals are of quite high speed, DMA is suggested to be used to enhance system performance. For detailed information, see *AN4520: An I2S Application on Kinetis.*

5 Key code

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

static void _i2s_init(void)
{
    #define I2S_CONFIG_WORDS_IN_A_FRAME 2
    #define I2S_CONFIG_BITS_IN_A_WORD   24

    I2S0_TCR1 = 4;// 6;    // water mark
    I2S0_TCR2 |= (0<<30) | // master mode(Async mode)
                 (1<<26) | // MSEL = MCLK
                 (1<<25) | // CLK = drive on falling edge
                 (1<<24) ; // CLK = OUTPUT

    I2S0_TCR3 = (1<<16); // enable channel 0

    I2S0_TCR4 = ((I2S_CONFIG_WORDS_IN_A_FRAME-1)<<16)  | // words in a frame
                ((I2S_CONFIG_BITS_IN_A_WORD  -1)<<8)   | // bits in a word
                 (1<<4)                                | // MSB
                 (1<<3)                                | // one bit early
                 (1<<1)                                | // frame active low
                 (1<<0)                                ; // frame = output

    I2S0_TCR5 = ((I2S_CONFIG_BITS_IN_A_WORD-1) <<24) | // word N width
                ((I2S_CONFIG_BITS_IN_A_WORD-1) <<16) | // word 0 width
                 (0x17<<8);                            // right adjust, where the
                                                          first bit starts
    I2S0_TMR = 0;

    // enable TX
    I2S0_TCSR = (0<<31) | // enable tx
                (1<<28) | // enable bit clock
                (1<<0);   // enable DMA request
}

static void _i2s_set_rate(int smprate)
{
    unsigned char div;
    SIM_SCGC6 |= SIM_SCGC6_I2S_MASK;

    // Select MCLK input source
    I2S0_MCR = (1<<30)| // MCLK = output
```

```
                (0<<24); // MCLK SRC = core clock = 48M

    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 32000: div=3;  break; // 12.288M/(32K*48) = 8, 8 = (DIV+1)*2, DIV = 3
    }

    I2S0_TCR2 = div;
}

void hal_i2s_enable(void)
{
    I2S0_TCSR |= 1u<<31;
}
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;

    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);
    }

    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;
}
void app_audio(void)
{
    sin_table_init();
    hal_i2c_init();
    hal_i2s_init();
    hal_audio_init();
    hal_i2s_enable();
    hal_audio_play(sin_table, sin_table2, BUF_N_SAMPLES);
}
```

6 Conclusion

In continuation with *AN4520: An I2S Application on Kinetis*, this application note discusses:

- how to configure the bit clock
- how a frame is constructed and the considerations to be taken into account for creating a frame
- other required configurations
- reference code to operate I2S module on Kinetis application for 2.x silicon

7 References

The following reference documents are available on freescale.com :

- *K60 Sub-Family Reference Manual* (document number K60P144M100SF2V2RM)
- *An I2S (Inter-IC) Application on Kinetis* (document number AN4520)

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number Document Number AN4800
Revision 0, 09/2013

ARM POWERED

freescale™