Freescale Semiconductor
Application Note

Document Number: AN4486
Rev. 0, 03/2012

MPC5604E Serial Audio Interface

by:
Pavel Bohacik, Automotive and Industrial Solutions Group

Contents

1 Introduction

The MPC5604E microcontroller is the new member of the 32-bit microcontroller family built on Power Architecture® technology. This device is targeted at the chassis and safety market segment visual-based driver assistance, especially the CMOS Vision Sensor Gateway, Radar Sensor Gateway, and Infotainment Network Gateway.

Connecting audio devices to the MPC5604E is an easy task for the MPC5604E Serial Audio Interface (SAI). This application note shows how the MPC5604E SAI can be easily configured for any audio converter.

2 Audio data formats

Audio data can be stored in various formats, and the format of the data is important because it will have implications on the system design for the digital interconnect between the controller and converter. This section will describe some basics of different audio formats that affect the arrangement of audio data in memory. The MPC5604E audio interface is flexible enough to work with any of these formats and their derivatives.

2.1 Sample rates and bit rates

Sample rates, number of channels, and audio sample size will affect the bit rate and the memory required for digital audio. All digital audio data has a sample rate and sample size that the converter uses to accurately recreate the audio signal.

Sample rate will constrain the maximum audio frequency that can be accurately represented by the audio converter without aliasing. The number of bits will constrain the maximum signal-to-noise ratio that can be achieved by using the formula Max SNR = 6.02 * (number of bits) – 7.3 dB. More bits are required for high-quality audio. The number of channels will increase the bit rate proportionally for 2-channel stereo or 6-channel Dolby Surround® technology.

A collection of 8 kHz audio samples will mean that a new sample is needed by the converter every 125 µs. If those samples are 16-bit 2-channel stereo samples, then the resulting rate is:

$$2\frac{bytes}{sample} * 8000\frac{samples}{sec} * 2channels = 32000\frac{bytes}{sec}$$

Figure 1.

Some other common formats and their associated bit rates are listed below.

Table 1. Common digital audio sample rates source

Use	Sample rate	Number of bits	Channels
CD Audio	44.1 kHz	16	2
Telephone Audio	8 kHz	8	1
High-end Professional Audio	96 kHz	24	2
FM Radio	22.050 kHz	16	2

2.2 Raw audio

Raw or pulse code modulated (PCM) is the most basic form of data that converters use to translate a digital value to an analog signal. The audio samples are represented as twos-complement signed data. There can still be some question as to whether the data is in a big-endian or little-endian format. Wave files (.wav) are usually little-endian if they were recorded on a PC. Most of the audio converters do not accept little-endian, that is, least significant byte first, so these files would have to be converted to big-endian which is the native format for the MPC5604E Power Architecture® core. Multiple audio channels are usually interleaved in memory. For example, 2-channel stereo samples would be stored alternating left channel and right channel samples in memory. Since the SAI puts what is sent to its FIFO directly on the serial pins, it is important to pay attention to the audio converter format and the most efficient way to put audio samples into main memory.

2.3 Compressed audio

Compressed audio data is obtained by taking the raw audio format and running it through an encoding process. Therefore, a set of audio data that takes 1 MB of memory in raw form might fit into only 170 KB of memory. Compression algorithms may cause some information to be lost, but tend to remove a lot of the information that is inaudible to the human ear. There are various coding schemes with varying compression ratios and audio quality. It is important to understand that compression affects the rate at which data needs to be transferred over the interconnect. For example, a raw 1.4 MB/s stream when compressed may only require 128 KB/s when encoded as an MP3 file. A more expensive audio converter might have an MP3 encoder/decoder included, which means that the MPC5604E would have to send the data only at 128 KB/s.

3 Common converter connection formats

There are several standards for connecting microcontrollers to audio converters. In general, a frame signal, transmit data, receive data, and bitclock are used to transfer data to and from a converter. The frame rate usually represents the sample rate of the audio data, although this is not always the case. This section illustrates some examples on connecting an audio converter to the MPC5604E SAI.

3.1 Codec/DSP

Most converters use a frame sync signal to signify the beginning of a new sample of audio data. These converters are usually associated with mono or single channel converters. The frame sync pulse frequency is usually the sample rate in the single channel converter. There are a few variations such as to whether the most significant bit (MSB) or least significant bit (LSB) comes first or if the data starts with the frame sync or after one bit time. Other variations have to do with frame sync and clock being active high or active low. The frame sync signal determines when the next audio sample is to be transferred between the controller and the converter. Also, the frame sync signal as seen in the figure below can be one bit time or a long bit time. That is why the frame sync frequency is usually the sample rate. Codec/DSP mode can be compared to I2S protocol with frame sync asserted for one bitclock cycle as shown on the figures below. Based on the implementation, frame length may support up to 16 words per frame.

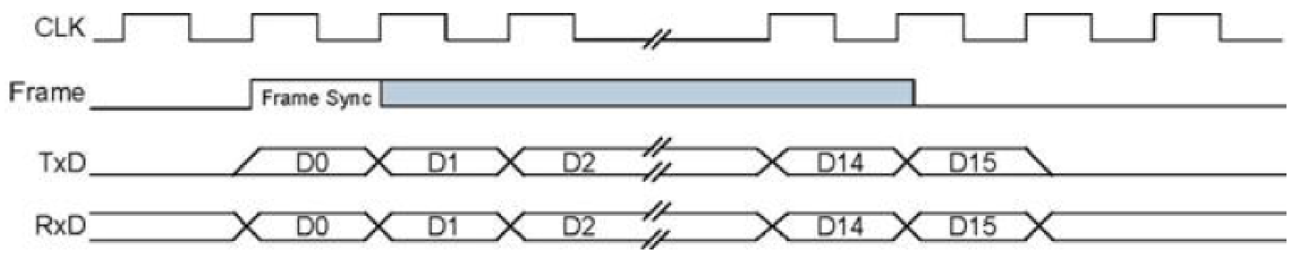

Figure 2. General PCM interface using 16 bits LSB first

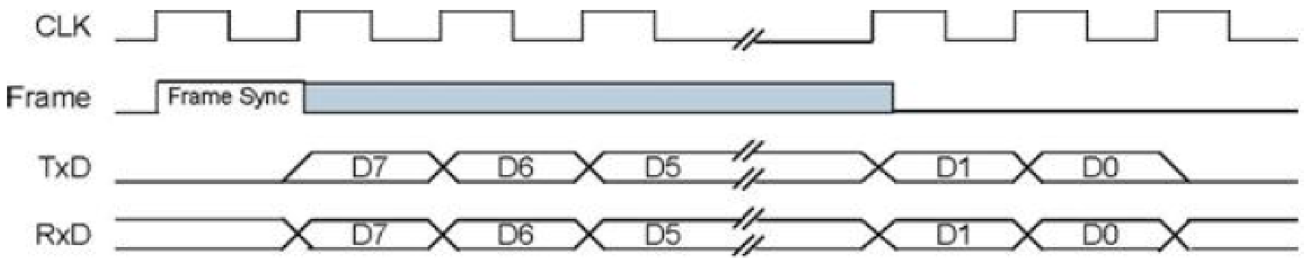

Figure 3. General PCM interface using 16 bits MSB first

3.2 I2S

I2S was defined by Philips as the source for 2-channel stereo audio streams. It is serial master–slave communication protocol which requires three communication signals – bitclock, frame sync, and data. The left or right channel audio data is defined by the state of the LRCK signal. The LRCK is the frame sync signal and defines the sample frequency for the data. I2S can accommodate any data size usually from 8 to 32 bits for each channel with the most significant bit (MSB) first. Note that the data is shifted by one bit from the start of the LRCLK. Since the MSB comes first, the controller can output more or less bits than the converter expects. For example, if the converter is 32-bit, but the controller only has 16-bit samples, the data can be left-justified to the MSB and have the lower 16-bit set 0. The converter can still accurately represent the signal in 32-bit. The same connection can be used for 8- or 32-bit data samples without changing anything except the number of bits used in the audio sample. A variation on I2S which is called left-justified swaps the state meaning of the frame sync signal from low meaning left to high meaning left, and it removes the single clock delay for the first bit in relation to the frame sync signal. The MPC5604E SAI can easily work with either format.

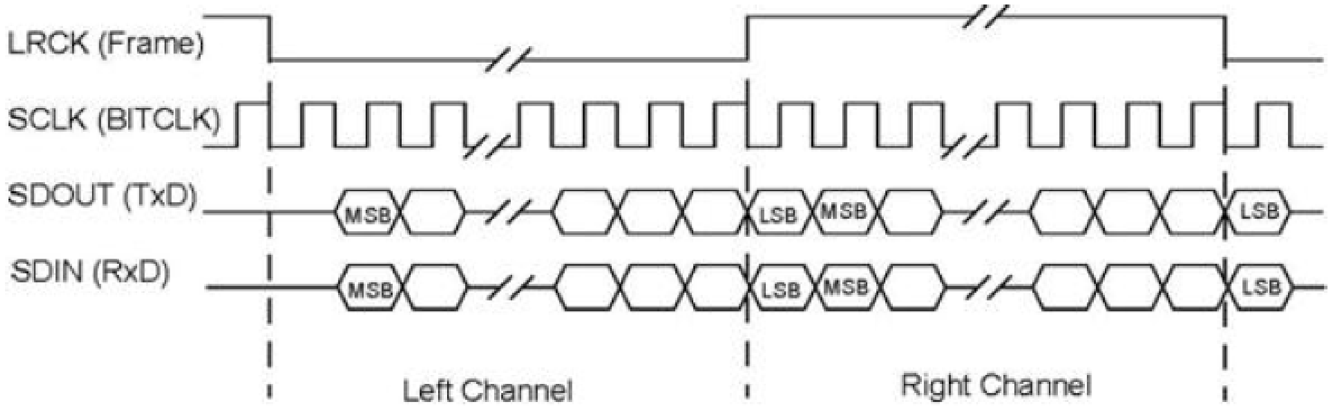

Figure 4. Sample I2S interface

3.3 AC97

AC97 was a standard for audio codecs used for personal computer design in 1997. The specification has been updated since then to include new features. AC97 was designed to handle multiple 20-bit channels at a 48 kHz rate. It also incorporates sample rate conversion through hardware, so the controller can work directly with some popular sample rates. The AC97 standard defines a complex analog mixer that would likely be needed for telephony, recording, and entertainment applications to be integrated into the converter. The AC97 frame contains control and audio data in the same serial stream. Other audio interconnect standards deal only with data, and there is usually another set of pins for control data that is an SPI or I2C interface. Since the control for the AC97 converter is embedded into the data channel, driver software is more complex than just sending the audio data. AC97 also has the ability of using different sample rates using the data slot tags in slot 0 for incoming data and slot 1 for transmitted data. The AC97 device tells the microcontroller when to put the next output sample in the data out stream. This lets the interface still run at 48 kHz while letting the converter sample rate change to a lower rate such as 8 kHz or 44.1 kHz.

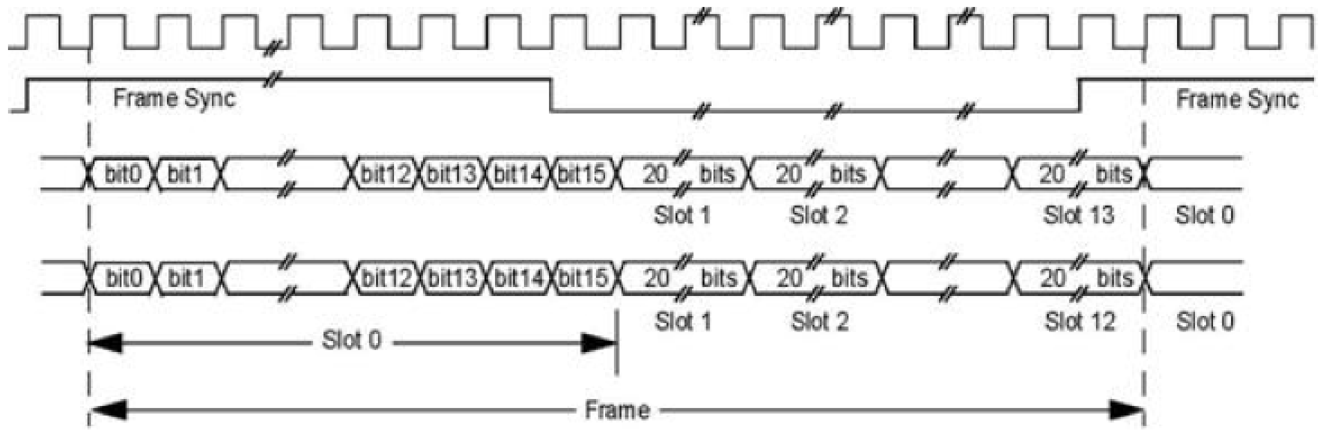

Figure 5. Sample AC97 frame

4 Clock conversion

There are some issues that need to be discussed surrounding the clocks in the audio subsystem. There are a few capabilities of the MPC5604E SAI that the system designer should be aware of. These issues and capabilities are discussed in this section.

4.1 MCLK

Another signal found on stand-alone audio converters is called MCLK. This is the master clock that controls the internal logic of a converter or the over-sampling clock of a delta-sigma converter. This clock is usually much faster than the bitclock. Not all converters require this clock. If they do, then there will usually be a relationship between the MCLK and the bitclock. The MPC5604E has an MCLK output pin on all SAI instances. The fractional clock divider (FCD) is responsible for creating the MCLK frequency. Even if this clock signal is not an output of the MCLK pin, it is still used to derive the serial bitclock used by the SAI. The MPC5604E also supports input direction for MCLK frequency from external source. This solution is used for high-quality audio application.

4.2 FCD

FCD provides audio master clock source (MCLK) to SAI module. FCD provides clock with low jitter with fine granularity which can be applied for low-quality audio playback. Input to FCD can be selected between multiple clock sources:

- XOSC_CLK – crystal oscillator clock source
- FMPLL_0_CLK – FMPLL output without FMPLL_CLK_DIV divider clock source
- PLL_Divider – FMPLL output with FMPLL_CLK_DIVIDR divider connected clock source
- IRCOSC_CLK – RC-Oscillator clock source

To achieve a minimum clock jitter, the input frequency shall be selected as high as possible, up to the maximum input frequency the FCDs can work on. Because the system clock shall not be below 50 MHz due to the FEC minimum clock, the FCD should operate at input frequencies of 200 MHz and above.

4.3 Bitclock and frame sync masters

A separate issue for clock and frame sync signals besides what format is used is which device drives the clock. In some cases the audio device will drive the clock and frame sync, and in other cases the controller or MPC5604E drives the clock and frame sync. AC97 specifies a different method where the codec drives the clock, and the controller drives the frame sync derived from the input clock. In an audio system, it is a good idea to have one clock driving all of the frame sync and bitclock signals. This device will drive the clock that all of the audio samples will be synchronized to. Having all the samples synchronized is important in digital signal processing applications, such as acoustic echo cancellation systems. The MPC5604E has features that help in clock integration issues that are explained in the Audio communication protocols examples.

4.4 Synchronous modes

MPC5604E comprises three SAI modules which can support up to six (stereo) audio channels, that is, up to six data pins. There are several types of synchronization but not all are available for each SAI instance.

- Synchronous mode - SAI transmitter and receiver can be configured to operate with synchronous bitclock and frame sync and controlled by the same module enable. In this mode, SAI can transfer or receive synchronous serial data on all its data pins. SAI instance can transfer up to four stereo audio channels. Synchronous mode supported for only SAI0 as SAI1 and SAI2 has single data line each. The synchronization is via the enable and frame sync. Since they all see the same enable and frame sync, they will start on the same frame and stop on the same frame.

- Synchronous instances or Multiple SAI Synchronous mode - allows synchronous transfer or receive operation within multiple SAI modules. When operating in synchronous mode only the bitclock, frame sync, and module enable are shared. The separate SAI peripherals otherwise operate independently, although configuration registers should be configured consistently across both the transmitter and receiver. All three SAI modules can be synchronized to operate from the same BCLK and FSYNC from SAI2. Modules will operate with similar bitclock provided by SAI2 module. SAI2 can be configured in either Master or Slave mode and SAI0/SAI1 will use the same BCLK that is used by SAI2, either internally generated from MCLK or externally provided to the SoC.

- Asynchronous mode - SAI transmitter or SAI receiver can be configured to operate asynchronously with free running bitclock that can be generated internally from an audio master clock or supplied externally.

Synchronous or asynchronous operation is configured in TCR2/RCR2 register by CLKMODE parameter.

SAI master can operate in Synchronous or Asynchronous mode. SAI slave can operate in Asynchronous mode, but SAI0 slave can operate in Synchronous or Asynchronous mode.

NOTE

CLKMODE for TX and RX cannot be 1 each simultaneously.

For more information about configuration of CLKMODE, see chapter 23.1.3.2 CLKMODE in SAI/I2S TCR2 Register in the MPC5604E Reference manual available at http://www.freescale.com.

5 Audio communication protocols examples

This section provides configuration examples for I2S, DSP, and AC97 audio communication protocols. It is recommended to use DMA to fill transmit buffer and empty receive buffer to reduce CPU loading.

5.1 FCD example

Fractional clock divider (FCD) provides audio clock source to SAI module. FCD output frequency can be reconfigured easily, usually with the use of closed control loop. The equation below describes calculation of the output FCD frequency. It is strongly recommended to use input source frequency higher than 200 MHz for system clock frequency 50 MHz.

$$MCLK\,output = MCLKinput * \left(\frac{FRACT + 1}{Divide + 1}\right)$$

Figure 6.

CAUTION

FCD MCLK fraction must be set equal to or less than the FCD MCLK divider.

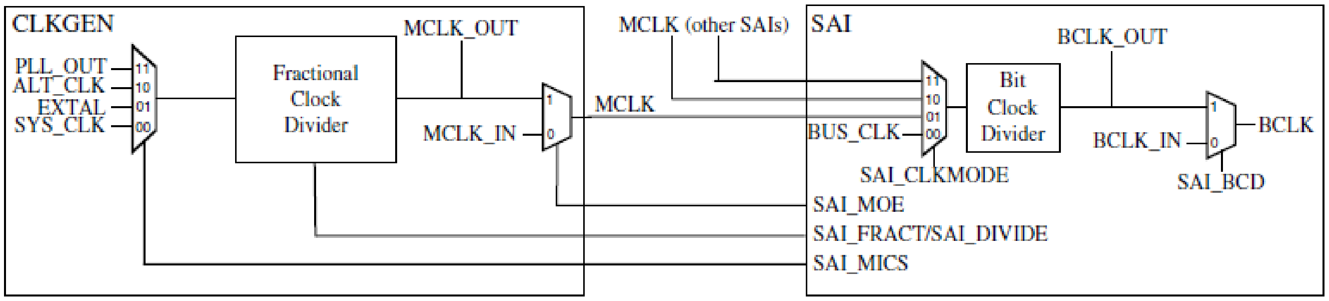

Figure 7. SAI clocking diagram

Table 2. Example of FCD configuration for common audio frequencies

FCD input source frequency	MCLK output frequency		
	11.289 MHz	12.288 MHz	24.576 MHz
Fractional ratios for FMPLL_0_CLK = 240 MHz	31/659	32/625	64/625
Fractional ratios for FMPLL_0_CLK = 256 MHz	40/907	6/125	12/125

FCD is initialized in five steps:

1. Disable transmit and receive function of SAI

```
SAI0.STCSR.B.TE =0;      /* Disable transmitter */
SAI0.STCR3.B.TCE = 0;    /* Disable a data channel for a transmit operation */
SAI0.STCSR.B.BCE=0;      /* Disable transmit bitclock */
SAI0.SRCSR.B.RE =0;      /* Disable receiver */
SAI0.SRCR3.B.RCE = 0;    /* Disable a data channel for a receive operation */
SAI0.SRCSR.B.BCE=0;      /* Disable receive bitclock */
```

2. Disconnect FCD from SAI module with SAI_MOE bit. When SAI module is disconnected, reconfiguration of FCD input clock source is allowed.

```
SAI0.SMCR.B.MOE=0;       /* SAI_MCLK pin is configured as an input that bypasses the
MCLK Divider. */
```

3. Configure input clock source to FCD with use input clock selector. This field cannot be changed when MCLK divider is enabled.

```
SAI0.SMCR.B.MICS = FCD_input_clock_source;// MCLK input clock select
```

4. Configure fraction and divider of FCD

```
SAI0.SMDR.B.FRACT=fract_value; /* MCLK Fraction. "-1" calculate required FRACT register
value from real value. */
SAI0.SMDR.B.DIVIDE=Divide-1;   /* MCLK Divide. "-1" calculate required FRACT register
value from real value. */
while(SAI0.SMCR.B.DUF==1){;}   /* MCLK divider ratio is updating on-the-fly when DUF=1
*/
```

5. Connect FCD to SAI module with SAI_MOE bit

```
SAI0.SMCR.B.MOE=MCLK_direction; /* SAI_BCLK pin is configured as an output from the
MCLK Divider and MCLK Divider is enabled. */
```

FCD can reconfigured easily in two steps:

1. Configure fraction and divider of FCD

```
SAI0.SMDR.B.FRACT=fract_value; /* MCLK Fraction. "-1" calculate required FRACT register
value from real value. */
```

```
SAI0.SMDR.B.DIVIDE=Divide-1;   /* MCLK Divide. "-1" calculate required FRACT register
value from real value. */
```

2. Wait for synchronization

```
while(SAI0.SMCR.B.DUF==1){;} /* MCLK divider ratio is updating on-the-fly when DUF=1 */
```

5.2 I2S example

When the SAI is configured for I2S mode, it can be used to connect to I2S converters using the frame sync, bitclock, and data signals. By setting the correct registers, the general codec signals can be made to look like an I2S interface. The basic settings use the 8-, 12-, 16-, 20-, 24-, or 32-bit for the audio sample data width. The following connection diagram and code shows how to enable the SAI for I2S mode using a 32-bit data width. Even if the MCLK output pin is not used, the internal MCLK is still being used by the SAI to generate the bitclock from the divider. For example, SAI cannot output an MCLK signal, yet it is still needed to generate bitclock and needs to be initialized in the MCR and MDR registers. The MCLK dividers are described in detail in the FCD chapter in the MPC5604E Reference manual available at http://www.freescale.com.

1. Configuration of SAI clock mode
   - Issue software reset and FIFO reset for Transmitter and Receiver sections before starting configuration.

     ```
     SAI0.STCSR.B.SR=1; /* Issue software reset */
     SAI0.STCSR.B.SR=0; /* Release software reset */
     SAI0.STCSR.B.FR=1; /* FIFO reset */
     ```
   - Configure FIFO watermark. FIFO watermark is used as an indicator for DMA trigger when read or write data from/to FIFOs.

     ```
     SAI0.STCR1.B.TFW=4;/* Watermark level for all enabled transmit channels of one SAI
     module. */
     ```
   - Configure the clocking mode, bitclock polarity, direction, and divider. Clocking mode defines synchronous or asynchronous operation for SAI module. Bitclock polarity configures polarity of the bitclock. Bitclock direction configures direction of the bitclock. Bus master has bitclock generated externally, slave has bitclock generated internally – see Figure 7.

     ```
     SAI0.STCR2.B.CLKMODE= CLKMODE;/* Synchronous or asynchronous operation. See
     Synchronous modes for detail description */
     SAI0.STCR2.B.BCP = 1;         /* Bitclock is active low (drive outputs on falling
     edge and sample inputs on rising edge */
     SAI0.STCR2.B.BCD = 0;         /* Bitclock is generated internally (Master mode) */
     SAI0.STCR2.B.DIV = 0;         /*Divides down the audio master clock to generate the
     bitclock when configured for an internal bitclock (master). The division value is
     calculated according to the equation below */
     ```

$$BitClock = \frac{InputClock}{(divider+1)*2} = \frac{32MHz}{(0+1)*2} = 16\text{MHz}$$

Figure 8.

   - Configure frame size, frame sync width, MSB first, frame sync early, polarity, and direction Frame size – configures the number of words in each frame. The maximum supported frame size depends on SAI implementation. I2S requires two words per frame. Frame sync width – configures the length of the frame sync in number of bitclocks. The sync width cannot be longer than the first word of the frame. I2S requires frame sync asserted for first word.

     ```
     SAI0.STCR4.B.FRSZ = 1;/* Configures number of words in each frame. The value
     written should be one less than the number of words in the frame. */
     SAI0.STCR4.B.SYWD =31;/* Configures length of the frame sync. Example for I2S and
     32-bit words are transmitted. The value written should be one less than the number
     of bitclocks. */
     SAI0.STCR4.B.MF = 1;  /* MSB is transmitted first */
     ```

```
SAI0.STCR4.B.FSE=1;   /* Frame sync asserted one bit before the first bit of the
frame */
SAI0.STCR4.B.FSP=1;   /* Frame sync is active low */
SAI0.STCR4.B.FSD = 1; /* Frame sync is generated internally (Master mode) */
```

- Configure the Word 0 and next word sizes and first bit shifted. W0W – defines number of bits in the first word in each frame. The maximum supported bits in the first word in each frame are 32 bits. Words of fewer than 8 bits wide are not supported if there is only one word per frame. WNW – defines number of bits in each word for each word except the first in the frame. The maximum supported bits in the first word in each frame are 32 bits. Words of fewer than 8 bits wide are not supported.

```
SAI0.STCR5.B.WNW = 31;    /* Number of bits in each word in each frame except the
first word. Example for I2S and 32-bit words are transmitted. */
SAI0.STCR5.B.W0W =31;     /* Number of bits in first word in each frame. Example for
I2S and 32-bit words are transmitted. */
SAI0.STCR5.B.FBT = 0;     /* Configures the bit index for the first bit transmitted
for each word in the frame. */
```

- Clear the Transmit and Receive Mask registers.

```
SAI0.STMR0.R=0;           /* Enable or mask word N in the frame. */
```

2. Configuration of eDMA module
   The configuration of eDMA module is done, see application note AN2865 available at http://www.freescale.com. SAI module has two triggers which are used to start DMA transfer. FIFO warning DMA enable – trigger is generated when FIFO is empty (FIFO watermark reaches zero), FIFO request DMA enable – trigger is generated when number of words in an enabled transmit channel FIFO is less than or equal to the transmit FIFO watermark.

```
SAI0.STCSR.B.FWDE=0; /* FIFO warning DMA trigger is disabled */
SAI0.STCSR.B.FRDE=1; /* FIFO request DMA trigger is enabled. In our example, trigger
occur when words in an enabled transmit channel FIFO is less or equal to the transmit
FIFO watermark = 4 words. */
```

3. Execution of I2S transfer requires following sequence

```
SAI0.STCR3.B.TCE = 1;  /* Enables a data channel for a transmit operation. */
EDMA.EDMA_SERQR.R = 0; /* Start DMA transfer to TX FIFO - A channel must be enabled
before its FIFO can be accessed - see line above. */
SAI0.STCSR.B.TE =1;    /* Enables the transmitter. */
```

4. Stop execution of SAI transfer requires following sequence

```
SAI0.STCSR.B.TE =0;     /* Disable transmitter. When software clears this bit, the
transmitter remains enabled and this bit remains set until the end of current frame. */
while (SAI0.STCSR.B.TE);/* Poll TE until it is read as zero */
```

NOTE

Clearing TE/RE will take several cycles to resolve because the SAI will gracefully stop at the end of the current frame and the TCE/RCE should not change during this time. To gracefully disable the SAI, software should clear TE/RE and then poll until it is read as zero. For a non-graceful disable, software can clear TE/RE and set the software reset.

The figure below summarizes the register settings and the corresponding waveform. After setting the SAI, the data should be written to the SAI TX FIFO with alternating channels starting with the left then right channel.

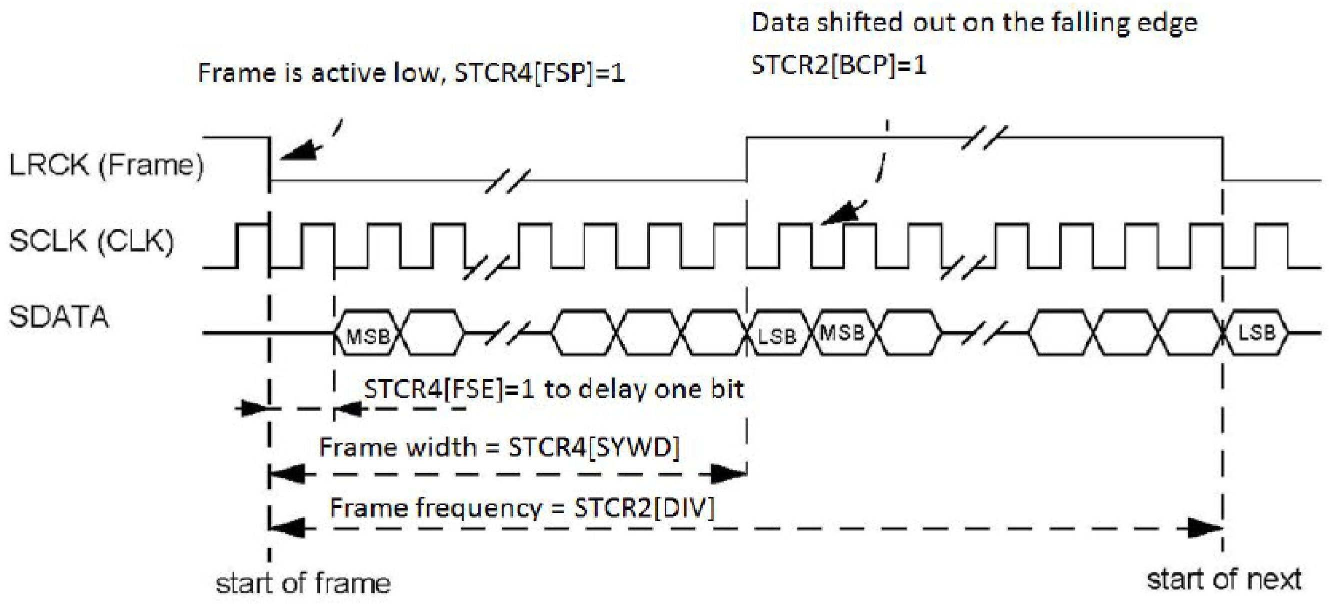
Data shifted out on the falling edge
STCR2[BCP]=1
Frame is active low, STCR4[FSP]=1
LRCK (Frame)
SCLK (CLK)
SDATA
MSB
LSB
MSB
LSB
STCR4[FSE]=1 to delay one bit
Frame width = STCR4[SYWD]
Frame frequency = STCR2[DIV]
start of frame
start of next

Figure 9. I2S register settings and signal diagram

5.3 CODEC example

When the SAI is configured to behave like a codec mode, it can be used to connect to basic converters using the frame sync, bitclock, and data signals. The MCLK signal can also be an output, on SAI0, SAI1, and SAI2. The basic settings use the 8-, 12-, 16-, 20-, 24-, or 32-bit for the audio sample data width. This setting will define the smallest data size transferred by the SAI at the start of every frame sync signal going active.

Configuration of DSP mode is similar to the configuration of I2S mode with difference:

```
SAI0.STCR4.B.SYWD =0; /* Configures length of the frame sync. DSP mode is generally used to
indicate a mode where the frame sync is bit wide instead of in I2S where it is word width. */
```

After setting the SAI, the data should be written to the SAI TX FIFO with alternating channels starting with the left then right channel.

5.4 AC97 example

AC97 mode is different from the other modes. Clock frequency and other signals are defined by the AC97 specification. For AC97, the bitclock is 12.288 MHz and the frame sync is 48 kHz. Once the SAI is put into AC97 mode and enabled, the SAI will start sampling the RXD line for a valid ready bit before putting any data on the TXD line. The AC97 codec will put the valid bit on the SDATAIN input after it has been reset.

NOTE

The pins on the AC97 codec are named in relation to the SAI controller. This means that SDATAIN on the codec is connected to the RXD pin of the SAI and SDATAOUT of the codec is connected to the SAI TXD pin.

Since control commands are included in the AC97 frame, the format has to be built in software before being put into the SAI transmits FIFO. The audio sample data also has to be stripped from the incoming data stream. This means that there is some software overhead to handle the AC97 that is unnecessary with the general codec mode.

There are 13 slots of the AC97 frame, with the first slot being 16 bits long and the other 12 slots are 20 bits. The SAI transmit and receive FIFO use 32 bits for each slot of the AC97 frame. This means that all writes and reads of the SAI FIFO in AC97 mode will be 32 bits. The SAI will put the correct number of data-bits onto the TX line corresponding to the slot number.

Following are the differences between configuration of SAI in I2S and AC97.

Configuration of Frame size. Frame size – configures the number of words in each frame. AC97 requires 13 words per frame.

```
SAI0.STCR4.B.FRSZ = 12; /* Configures number of words in each frame. The value written
should be one less than the number of words in the frame. */
```

Configuration of frame sync width. Frame sync width – configures the length of the frame sync in number of bitclock. The sync width cannot be longer than the first word of the frame. AC97 requires frame sync asserted for first word.

```
SAI0.STCR4.B.SYWD =15; /* Configures length of the frame sync. The value written should be
one less than the number of bitclocks.  AC97 - 16 bits transmitted in first word. */
SAI0.STCR4.B.FSE=1;    /* Frame sync asserted one bit before the first bit of the frame */
```

Configure the Word 0 and next word sizes. W0W – defines number of bits in the first word in each frame. WNW – defines number of bits in each word for each word except the first in the frame.

```
SAI0.STCR5.B.W0W =15;  /* Number of bits in first word in each frame. AC97 - 16-bit word is
transmitted. */

SAI0.STCR5.B.WNW = 19; /* Number of bits in each word in each frame except first word. AC97
- 20-bit word is transmitted. */
```

After setting the SAI, the data should be written to the SAI TX FIFO with alternating channels starting with the left then right channel.

6 Moving data from main memory to the SAI

An audio device requires hard real-time audio data or the analog audio will contain artifacts that will annoy the listener with pops, clicks, dead space, and so on. There are a number of features in the MPC5604E that make the task of keeping the real-time requirements of the converter fulfilled. Audio samples are stored in main memory or SRAM. There are a couple of different ways to get the audio samples from main memory to the audio device, and this section describes what is necessary in terms of the system.

6.1 SAI FIFOs

SAI contains eight words depth FIFO for both transmitter and receiver. These FIFOs can be accessed directly by the e200z0 core of the MPC5604E; there is also eDMA unit. The eDMA can transfer data to and from SAI FIFO without involving the e200z0 core.

6.2 Using the e200 core to transfer audio data

One option for transferring data from SDRAM to the SAI FIFO is to use the e200z0 core. It is important to understand the data path and the interrupt load on the system when doing this. The FIFO alarm level will set the rate at which interrupts happen to the core. The CPU interrupt routine will then have to read from the audio buffer in SDRAM and write to the SAI FIFO data register to transmit data. This will generate two internal bus accesses on the XBAR for each data transfer to the FIFO. One is a read from the SDRAM and the other is the write to the SAI registers via PBRIDGE. For receive, this process is the same in the opposite direction. When the alarm goes off and triggers the SAI interrupt, the interrupt service routine should fill the FIFO until the alarm disappears. This is done by writing to the TDR register and then checking the alarm bit FWF or FRF in the TCSR register. The interrupt routine should stop writing to the FIFO when either bit clears. Data can be transferred as 8-bit, 16-bit, or 32-bit values to the FIFO even if the transfer mode set in the STCR5 register is a different size. It is a good idea for the data size write to match the mode being used. A loop to write to the FIFO is appropriate while checking the alarm bit in the FIFO status register. It is possible to overrun the transmit FIFO by writing too much to the transmit FIFO. Likewise, the receive FIFO can overflow by reading too much. These errors are only reflected in the TCSR register and cause in interrupt.

6.3 Using eDMA to transfer audio data

Another option for transferring data to/from the SAI FIFOs is the eDMA. There are a few advantages in using the eDMA:

- Fewer XBAR accesses
- Fewer interrupts to the e200 core

The eDMA engine is bus master on the XBAR and it has access to the SAI peripheral FIFOs via PBRIDGE. The eDMA also would have to generate two XL Bus transactions for the same operation. Having the eDMA engine transferring data, the core has more time to spend on processing the data. The eDMA can be configured to send an interrupt when a large buffer of audio data has been transferred from SDRAM to the FIFO reducing the interrupt rate to the e200 core. Instead of enabling the SAI FRF or FWF interrupt to the e200 core, the alarm signal tells the eDMA that a FIFO needs servicing. While the alarm signal is on, the eDMA will fill or empty the FIFO directly from or to SDRAM depending on whether the FIFO is a transmit or receive FIFO. The alarm registers and granularity registers still need to be initialized before starting the eDMA task. The reason for the granularity in the TCR1 or RCR1 register is that the eDMA has the ability to read ahead from the FIFO. It is possible that the eDMA can read too much from the transmit FIFO, and it will underflow. For this reason, the granularity is set to something greater than four.

7 FIFO underflow/overflow issue

This section describes synchronization issue related to the expected use case for MPC5404E audio interface. In the high- or low-quality audio output scenario, the MPC5604E is receiving the audio bit stream, up to 4x stereo, via the Ethernet interface. Hence the Audio source clock or bit stream rate is derived from a different crystal oscillator, the audio clock needs to be tuned to avoid buffer overflows or underruns.

If the source crystal produces a 44.090 KHz audio clock and the MPC5604E clock runs on 44.110 KHz then in each second 20 less samples are received than sent out. Having 2k samples buffered at the start means that this buffer is consumed after 100 s. This is a buffer underrun. The same happens in the other relation. To avoid this, the MPC5604E clock must adjust to the source clock. In high quality scenario, an external low jitter clock generator is tuned via a pulse width modulated (PWM) signal or alternatively via I2C. To allow a closed loop control avoiding long term jitter, the audio clock has to be measured. Measuring of the external audio clock is done via an eTimer input channel.

In low quality scenario, the FCD internal low jitter clock generator is tuned via reconfiguration that is fraction divider.

There is requirement to implement following steps for high or low quality scenario close loop control to compensate the frequency difference between the crystals:

- measure the number of sent samples – using the eTimer to count the syncs and the number of received samples, which can be measured via the low pass filtered buffer fill level
- mechanism to tune the audio clock – for internal clock using the FDC settings and for external via a PWM signal or the IIC.
  - SW to compute from the difference of received and sent samples of the new control value.

8 References

1. AN2979 – Available at http://www.freescale.com

2. AN2865 – Available at http://www.freescale.com
3. MPC5604E reference manual – Available at http://www.freescale.com

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: AN4486
Rev. 0, 03/2012