Freescale Semiconductor
Application Note

AN2979
Rev. 0, 05/2004

Using Audio with the MPC5200

by: Chris Alger
Infotainment Multimedia Telematics Division

1 Introduction

Connecting audio devices to the MPC5200 is an easy task for the versatile MPC5200 peripheral serial controller (PSC). There are many different options to choose from when it comes to picking an audio digital to analog (D/A) and analog to digital (A/D) converter. Digital interconnects, analog mixing capability, encode/decode capability, and audio quality are all factors to consider when selecting an audio device. This application note discusses some of those choices and shows how the MPC5200 PSC can be easily configured for any audio converter.

Table of Contents

2 Audio Data Formats

Audio data can be stored in many different 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 different audio format basics that affect how the audio data is arranged in memory. The MPC5200 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) - 7.3dB. More bits are required for high quality audio. Of course, 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 bytes/sample * 8000 samples/sec * 2 channels = 32000 bytes/sec *Eqn. 1*

Some other common formats and their associated bit rates are listed here in Table 1:

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 (least significant byte first), so these files would have to be converted to Big-Endian which is the native format for the MPC5200 PowerPC 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 PSC 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 Mbyte of memory in raw form might fit into only 170 kbytes 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 many different 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 Mbps stream when compressed may only require 128 kbps when encoded as an mp3 file. A more expensive audio converter might have an mp3 encoder/decoder included, which means that the MPC5200 would only have to send the data at 128 kbps. However, the MPC5200 has plenty of processing power to decode mp3 audio to raw format, and it also has the bus bandwidth to transfer the 1MB/s required for audio. There is no need to use an expensive audio converter with encode/decode capability and the expensive parallel interface it would likely require.

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 bit clock 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 MPC5200 PSC.

3.1 General PCM Codec

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 the most significant bit (MSB) or least significant bit (LSB) comes first or if the data starts with the frame sync or one bit time after. Other variations have to do with frame sync and clock being active high or active low. The figures below show some examples of audio data formats.

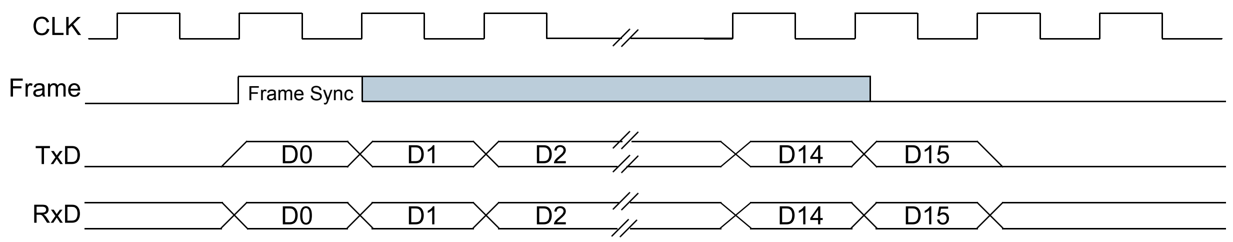

Figure 1. General PCM Interface Using 16 Bits Msb First

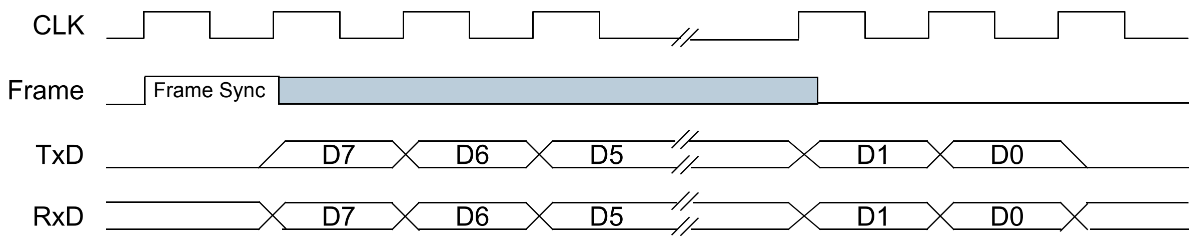

Figure 2. General PCM Interface Using 8 Bits, Lsb First, Data Shifted By One Bit

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 above figure can be one bit time or a long bit time. That is why the frame sync frequency is usually the sample rate. There are some variations to accommodate more audio channels from having every other frame be a different channel to having the bit clock be fast enough to have more than one channel data in each frame sync. For example, having 32-bits transferred each frame sync when the data sample size is 16-bits. These channel variations can be interfaced to the MPC5200 PSC, but usually stereo 2-channel converters use an I2S interface, as described in the next section.

3.2 I2S

I2S was defined by Philips source for 2-channel stereo audio streams. 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.

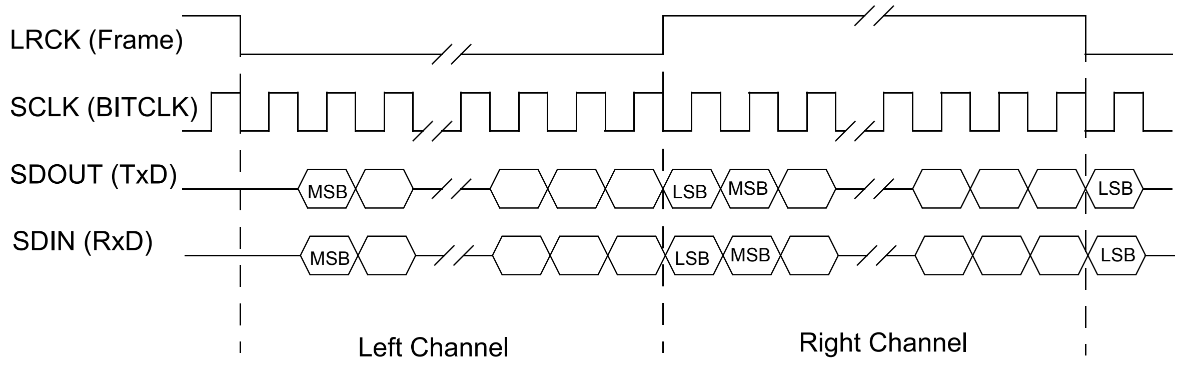

Figure 3. Sample I2S Interface

Notice 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 is expecting. 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-bits set 0. The converter can still accurately represent the signal in 32 bits. 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 MPC5200 PSC can easily work with either format.

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. Almost all personal computers today have an AC97 defined codec on the motherboard.

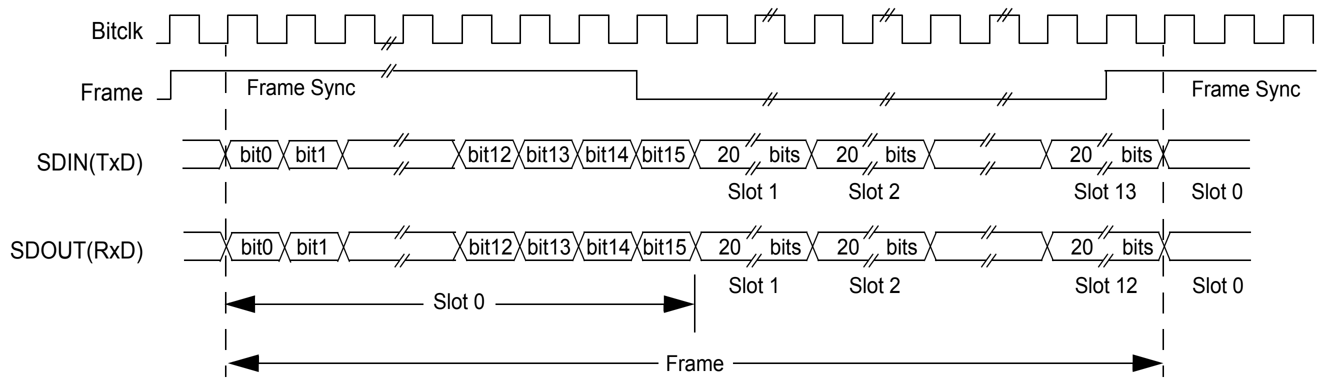

Figure 4. Sample AC97 Frame

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.

Since there are so many AC97 codecs to choose from, they are relatively cheap, and some manufactures are starting to carry industrial temperature versions. They are clearly finding applications other than PCs.

3.4 Clock Conventions

There are some issues that need to be discussed surrounding the clocks in the audio subsystem. There are a few capabilities of the MPC5200 PSC that the system designer should be aware of.

3.4.1 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 bit clock. Not all converters require this clock. If they do then there will usually be a relationship between the MCLK and the bit clock.

The MPC5200 has an MCLK output pin on PSC1, 2, and 3. The clock distribution module (CDM) of the MPC5200 is responsible for creating the MCLK frequency. Even if this clock signal is not output on the MCLK pin, it is still used to derive the serial bit clock used by the PSC.

3.4.2 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 MPC5200 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 bitclk 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 MPC5200 has features that help with clock integration issues that are explained in the next section.

4 MPC5200 Peripheral Serial Controller (PSC)

The peripheral serial controller (PSC) of the MPC5200 is a versatile module that, in addition to acting as a UART or IR controller, can accommodate the many different digital audio connection options available to system designers. This section will describe the different audio connection modes and how to configure the PSC registers to use them. For the programmer, the job will be to set the MPC5200 PSC to a compatible mode and make sure the audio data in memory gets to the PSC data FIFO in time. The names used for the registers in the source code can be found in the *MPC5200 User Guide*.

4.1 PSC Modes

The PSC has a couple of modes for connecting to audio codecs. They are classified into two categories: codec modes and AC97 modes. Codec modes are used for general purpose codecs and I2S codecs. The AC97 mode is specific for AC97 codecs only.

NOTE

The AC97 mode can only be used on PSC1 and PSC2, and the codec modes can only be used on PSC1, PSC2, PSC3 and PSC6.

4.1.1 Codec Mode

When the PSC is in codec mode, it can be used to connect to basic converters using the frame sync, bitclk, and data signals. By setting the correct registers, the general codec signals can be made to look like an I2S interface. The MCLK signal can also be an output, on PSC1, PSC2, and PSC3.

4.1.1.1 General Codec Mode

This mode is defined by setting the SICR register bits[4:7]. The choices are 8-, 16-, 24-, and 32-bit codec mode. This setting will define the smallest data size transferred by the PSC at the start of every frame sync signal going active.

For example, in Figure 2 the sample C source code to write the SICR register is:

```
psc2->SICR = 0x31100000;

/*

dts1=1, first data bit shifted right one bit

shdir=1, Least significant bit first

Codec mode = 0001, 8-bit codec mode,

clkpol = 0, sample input data on falling edge

syncpol =1, sync signal high is the start of the frame

genclk =0, clock and frame sync signals are inputs

*/
```

Since genclk=0, the MPC5200 PSC is a slave to the incoming clock and there is no need to initialize the frame sync and bit clock registers. If genclk = 1, then the MPC5200 drives the bit clock and frame sync signals. In this case, the timing registers will need to be initialized. For example, if the sample rate is 8 kHz and the bit clock is 64 kHz, then the sample code if genclk =1 would be:

```
psc2->CTUR = 0;

/* Frame sync is (CTUR + 1) bit long. The frame sync can be set to any length. */

cdm->clock_config = 0x000080F9;

/* Create MCLK by enabling it in the CDM and setting the divider to create a 2.112 MHz
MCLK from the 528MHz system VCO frequency by dividing by 250. Also bit 16 is set to
enable the mclk to the PSC */

psc2->CCR = 0x0720;

/* Creates a 64 kHz bit clock and an 8 kHz frame sync by dividing the MCLK by 33 and
the bitclock by 8*/
```

Even if the MCLK output pin is not used, the internal MCLK is still being used by the PSC to generate the bitclock from the divider. For example, PSC6 can not output an MCLK signal, yet it is still needed to generate bit clock and needs to be initialized in the CDM. The MCLK dividers are described in detail in the CDM chapter of the *MPC5200 Users Guide*.

4.1.1.2 I2S Mode

Interfacing to an I2S converter is similar to using normal codec mode except for a few settings that need to be configured. The basic settings use the 8-,16-, 24-, or 32-bit codec mode for the audio sample data width. Then, by setting the multiword bit (bit 9 in the PSC SICR register), the PSC will output more than one audio sample per frame sync. The following connection diagram and code shows how to enable the PSC for I2S mode using a 32-bit data width with a 48 kHz sample rate.

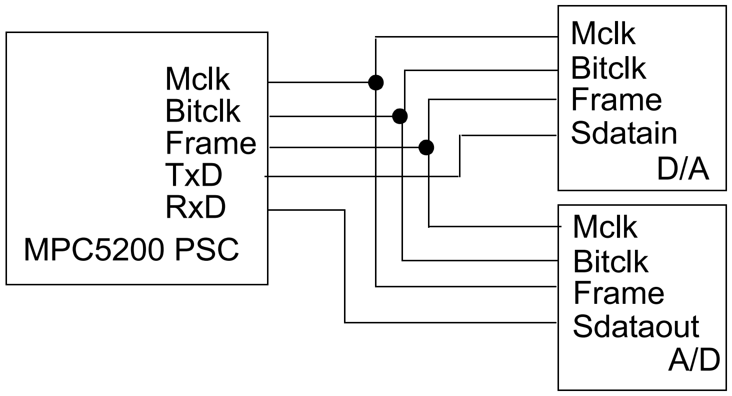

Figure 5. I2S A/D and D/A Connected to the MPC5200 PSC

4.1.1.3 Sample I2S Initialization Code

This section gives some initialization code for an I2S device.

```
/* enable mclk and divide by 4 to get a 132 MHz MCLK which is divisible into 48 KHz. */

cdm->mclken_div_psc2 = 0x00008003;

/* enable the 32-bit codec mode on PSC2 with clock settings for I2S mode*/

/* DTS1 = 1, SIM(mode) =0xF, GenClk =1, MultiWd = 1, ClkPol=1, SyncPol=0 */

psc2->SICR = 0x2FE00000;

/* The frame width is 32 bits so 31 is written to CTUR */

psc2->CTUR = 0x1F;

/* The MCLK is divided by 43 (0x2A + 1) for the bitclk and the bitclk is divided by 64
(0x3f + 1) to get the frame sync frequency */

psc2->CCR = 0x3F2A;

/* enable the mclk pin on PSC2 by setting the port_config register */

siu->port_config = 0x00000070;
```

The figure below summarizes the register settings and the corresponding waveform. After setting the PSC, the data should be written to the PSC FIFO with alternating channels starting with the left then right channel. More information about moving data to the PSC is given in section 5.

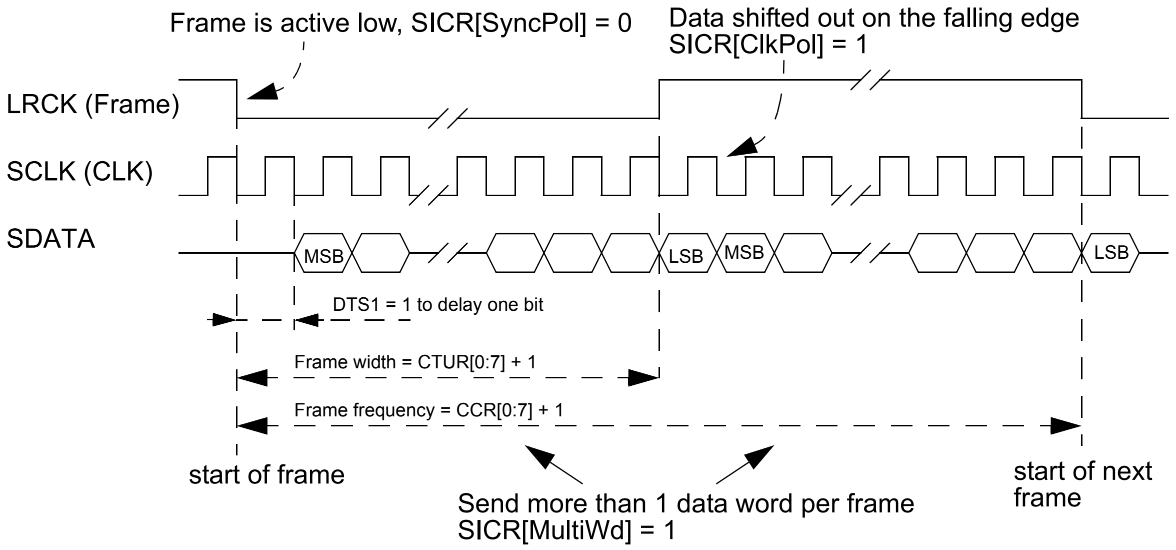
Frame is active low, SICR[SyncPol] = 0
Data shifted out on the falling edge
SICR[ClkPol] = 1
LRCK (Frame)
SCLK (CLK)
SDATA
MSB
LSB
MSB
LSB
DTS1 = 1 to delay one bit
Frame width = CTUR[0:7] + 1
Frame frequency = CCR[0:7] + 1
start of frame
Send more than 1 data word per frame
SICR[MultiWd] = 1
start of next frame

Figure 6. I2S Register Settings and Signal Diagram

4.1.2 AC97 Mode

AC97 mode is different from the other codec modes. Clock frequency and other signals are defined by the AC97 specification. There is no need to set clock frequencies because the bitclock pin is an input that the frame sync is derived from. For AC97, the bitclock is 12.288 MHz and the frame sync is 48 kHz. Once the PSC is put into AC97 mode and enabled, the PSC will start sampling the RxD line for a valid ready bit before putting any data on the TxD line. The AC97 reset pin is controlled through one of the PSC registers, and 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 PSC controller. This means that SDATAIN on the codec is connected to the RxD pin of the PSC and SDATAOUT of the codec is connected to the PSC TxD pin.

Since control commands are included in the AC97 frame, the format has to be built in software before being put into the PSC transmit 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, but the PSC provides some features to help handle the extra software load.

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 PSC transmit and receive FIFO use 32-bits for each slot of the AC97 frame. This means that all writes and reads of the PSC FIFO in AC97 mode will be 32-bits. The PSC will put the correct number of data-bits onto the TX line corresponding to the slot number. This data should be left-justified in the register because the least significant bits will be ignored. For example, if the value 0x98000000 is written to the FIFO, the PSC will transmit 0x9800 in slot 0 truncating the least-significant 16 bits.

Synchronizing the AC97 frame to the PSC FIFO is very important. If the interface gets out of sync then control and data will get mixed up, possibly causing a state on the AC97 codec requiring a codec reset to recover. To help synchronize the PSC to the codec, the FIFO has a couple of mechanisms. One mechanism is the use of bit 20 on the receive data FIFO. This bit is set to a 1 if the rest of the word in the FIFO corresponds to the beginning of a frame. Also, when the TX FIFO is enabled, the PSC will always begin transmitting data at the beginning of a frame. The sample section contains code that demonstrates how to use the PSC AC97 mode.

4.2 Cell Phone Mode for Clock Synchronization

A useful feature for synchronizing different audio devices connected to the PSC is the cell phone slave mode, which is activated using bit 12 (cell2slave) in the SICR register. This bit is only meaningful for PSCs 2, 3, and 6, because it makes the specific PSC use the input bitclock of PSC1 as the MCLK for that PSC. This mode was originally intended for synchronizing other converters with a device connected to PSC1, such as a wireless chipset that can only be a bit clock master. Another audio codec on PSC2 would need to be synchronized to the wireless chipset clock on PSC1 if the MPC5200 was used for echo cancellation. An acoustic echo cancellation algorithm is used in a speakerphone application where the MPC5200 is the controller and doing the digital signal processing. A figure that describes its application can be found in “Section 15.3: PSC Operation Modes” of the *MPC5200 Users Guide*.

It is important to note that this mode can also be used when PSC1 is used in AC97 mode. The bit clock is an input for AC97 mode. Another PSC can be synchronized to the AC97 clock when the cell phone slave mode on that PSC is used. This can be helpful for synchronizing another audio device to an AC97 device on PSC1.

5 Moving Data from Main Memory to the PSC

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, etc. There are a number of features in the MPC5200 that make the task of keeping the real-time requirements of the converter fulfilled. Audio samples are stored in main memory or SDRAM. 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.

5.1 PSC FIFOs

The main mechanism that is used to manage any data in the PSC is the 512-byte FIFO for both the transmitter and the receiver. The size of the FIFO has implications on how often the FIFO is serviced and the latency of audio data from SDRAM to the audio device connected to the PSC.

In addition to the FIFO, which can be accessed directly by the 603e core of the MPC5200, there is also the Bestcomm DMA unit. The Bestcomm can transfer data to and from the PSC FIFO without involving the 603e core.

5.2 Timing and the FIFO

The PSC FIFO will transmit or receive data at a constant rate defined by the parameters of the interface, To help service the FIFO, there are FIFO alarms that can be set to notify the CPU or Bestcomm when a FIFO has a certain amount of bytes left in it. The value which is written to the TFALARM and RFALARM register in the PSC depends on the data rate and the system load. For example, a 32-bit I2S interface with a frame sync frequency of 44.1 kHz, the 512 byte transmit FIFO will empty in:

512 bytes / (4 bytes/sample * 44100 samples /sec * 2 samples) = 1.45ms ***Eqn. 2***

It takes 1.45 ms for the entire PSC FIFO to underrrun at these settings. The same is true for the receive FIFO. If there is an underrun or overrun error, then the PSC generates an interrupt if enabled and sends 0s for the data on the next frame sync transition. For AC97, the rate is very high because the interface is effectively 32-bit data transferred at a very high rate.

13 slots/frame * 32 bits * 48000 frames/sec = 19.04 Mbit/sec ***Eqn. 3***

In AC97 mode, the FIFO can underrun in 200us, which means that the CPU is interrupted on the FIFO alarm level, which is 100us when the TFALARM and RFALARM registers are set to 256 bytes. The high data rate requires the Bestcomm DMA engine to handle the data movement unless the CPU handles only the AC97 interface.

The data rate defines the maximum amount of time that the CPU or Bestcomm has before it must service the FIFO. The alarm level should be set to give the system time to transfer from what it is currently doing over to servicing the PSC FIFO. This can be a complex problem to solve in the MPC5200, but there are enough priority settings to make sure that the system can service the FIFO in time.

5.3 Using the 603e Core to Transfer Audio Data

One option for transferring data from SDRAM to the PSC FIFO is using the 603e 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 PSC FIFO data register to transmit data. This will generate two internal bus accesses on the XL-Bus for each data transfer to the FIFO. One is a read from the SDRAM and the other is the write to the PSC registers. For receive, this process is the same in the opposite direction.

NOTE

There are two data registers that can be used to access the FIFO. Both the RB_TB register and the TFDATA register can be written to and the data will end up in the FIFO. However, TFDATA can also be read to see what is in the transmit FIFO. When reading from the TFDATA register, it is important to have the PSC disabled or there will be contention for the data between the PSC transmit logic and the register read. This functionality goes for the RFDATA register as well.

When the alarm goes off and triggers the PSC interrupt, the interrupt service routine should fill the FIFO until the alarm disappears. This is done by writing to the RB_TB register and then checking the alarm bit in the TFSTAT register or the TXRDY bit in the SR 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 codec mode set in the SICR 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 TFSTAT and RFSTAT register, and they do not cause in interrupt.

5.4 Using Bestcomm to Transfer Audio Data

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

- Fewer XL Bus accesses
- Fewer cycles being used by the 603e core to transfer data
- Fewer interrupts to the 603e core

The Bestcomm DMA engine has a direct bus connection called the CommBus to all of the peripheral FIFOs. This means that in one cycle, the Bestcomm can read from SDRAM through the XL Bus and write the FIFO through the CommBus. The core would have to generate two XL Bus transactions for the same operation.

Having the DMA engine transferring data, the core has more time to spend on processing the data. The Bestcomm 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 603e core.

Instead of enabling the PSC TxRDY interrupt to the 603e core, the alarm signal tells the Bestcomm that a FIFO needs servicing. While the alarm signal is on, the Bestcomm 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 Bestcomm task. The reason for the granularity in the TFCNTL or RFCNTL register is that the Bestcomm has the ability to read ahead from the FIFO. It is possible that the Bestcomm can read too much from the transmit FIFO, and it will underflow. For this reason, the granularity is set to something greater than 4.

The sample code that accompanies this application note shows how to use the PSC AC97 device. It transfers audio data for a 1kHz sine wave to an AC97 device connected to PSC1 using Bestcomm to transfer the data. This sample code uses the Bestcomm API software library and was built using Metrowerks CodeWarrior. The API and the sample code can be downloaded from freescale.com. Search for keyword ‘AN2979SW.’

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

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)
support@freescale.com

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 Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

freescale™
semiconductor

AN2979
Rev. 0
05/2004