SMART ARM-based Microprocessor

Getting Started with the SAMA5D2 Audio Subsystem

APPLICATION NOTE

Introduction

This application note describes the audio peripherals embedded in the Atmel® | SMART SAMA5D2 series of microprocessors. The document demonstrates the peripheral configuration using code examples running on the SAMA5D2 Xplained Ultra board.

Reference Documents

Type	Title	Atmel Lit. No.
Datasheet	SAMA5D2 Series Datasheet	11267
Software Package	SAMA5D2 Software Package	N/A

Table of Contents

1. Introduction to SAMA5D2 Audio Peripherals

The SAMA5D2 series of microprocessors embeds a wide set of peripherals targeting audio-intensive applications. This includes the following peripherals:

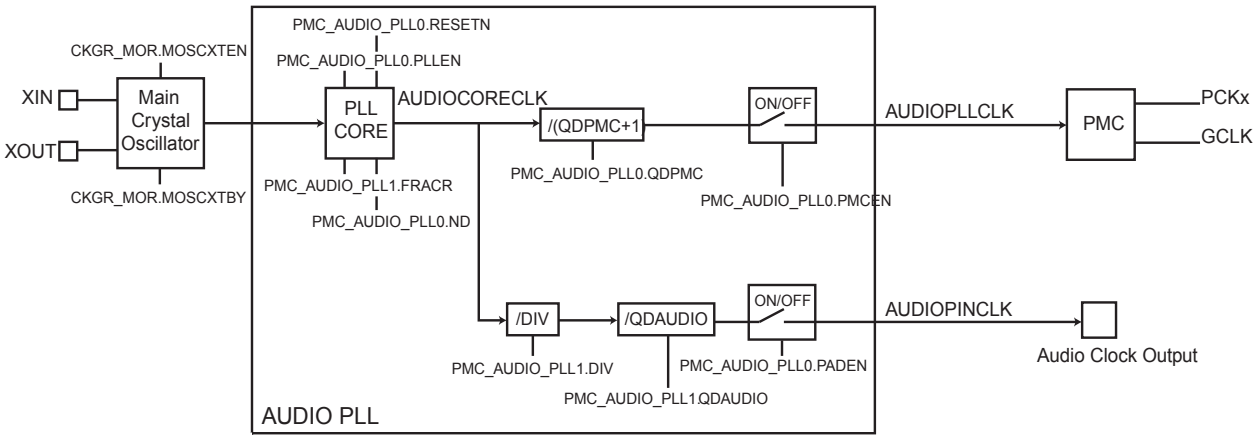
1. Digital Audio Phase Locked Loop (Audio PLL)
2. Pulse Density Modulation Interface Controller (PDMIC)
3. Digital Audio Class-D Amplifier (CLASSD)
4. Inter-IC Sound Controller (I2SC)
5. Synchronous Serial Controller (SSC)

The following sections describe each peripheral.

2. Digital Audio Phase Locked Loop (Audio PLL)

The SAMA5D2 series includes a high-resolution fractional-N digital PLL designed for low jitter operation. The PLL takes the reference clock signal from the main crystal oscillator and generates high frequency audio core clock output.

Figure 2-1. Audio PLL Block Diagram



The high frequency audio core clock output is split internally into two signal paths each having individual divider and clock gating networks. The output from one signal path is fed to the power manager (PMC) which is then used as the base clock (GCLK – Generic Clock) for audio peripherals like PDMIC, Class-D and I2SC. This output is the AUDIOPLLCK in the block diagram.

The output from other signal path is fed to the audio clock output pin (CLK_AUDIO) which can be used as a master clock for external components like an external audio codec chip. This output is the AUDIOPINCLK in the block diagram.

The Audio PLL is designed to generate an internal audio core clock frequency in the range of 620 MHz (min) to 700 MHz (max). Refer to Electrical Characteristics section in the SAMA5D2 Datasheet. This frequency is controlled by the user-configurable parameters ND and FRACR, as shown in the equation below:

$$f_{audiocoreclock} = f_{ref}\left(ND + 1 + \frac{FRACR}{2^{22}}\right)$$

f_{ref} is the reference input clock frequency from the main crystal oscillator which should be minimum 12 MHz and maximum 24 MHz. The fractional part in the above equation plays a key role in high resolution frequency adjustment. The maximum frequency resolution achievable is 2.8610Hz. This means that, for each increment in the FRACR value, the audio core clock output, typically in the 620-700 MHz range, is incremented by 2.8610Hz.

The AUDIOPLLCK path has a divider which is controlled by the user-configurable parameter QDPMC. The output frequency from the AUDIOPLLCK path is given by the equation below.

$$f_{audiopllck} = \frac{f_{audiocoreclock}}{(QDPMC + 1)}$$

The AUDIOPINCLK path has a divider which is controlled by the user-configurable parameters DIV and QDAUDIO. The output frequency is given by the equation below.

$$f_{audiopinclk} = \frac{f_{audiocoreclock}}{(DIV*QDAUDIO)}$$

The DIV and QDAUDIO parameters should be configured in such a way that $f_{audiopinclk}$ is in the range 8 MHz (min) to 48 MHz (max).

The Audio PLL output can be varied during runtime by adjusting the FRACR and ND values. The high FRACR resolution allows very fine and smooth frequency adjustments without audible artefacts. As an example, in a network application, it is possible to adjust FRACR so that the local audio clock frequency tracks a distant master clock in the network. On the contrary, when making large changes to the output frequency (e.g., when changing ND) it is good practice to first mute the relevant audio signals of the system and then let the PLL settle to the new frequency to avoid any undesirable noise. In any case, the settling time of the PLL to reach a new frequency value is maximum 100 µs. Refer to the Electrical Characteristics section in the SAMA5D2 datasheet.

2.1. Code Example - Audio PLL

The following code snippet specifies the steps to configure and enable the Audio PLL module.

Requirements:

f_{ref} = 12 MHz

$f_{audiopllck}$ = 12.288 MHz

$f_{audiopinclk}$ = 12.288 MHz

620 MHz ≤ $f_{audiocoreclock}$ ≤ 700 MHz

The values for ND, FRACR, QDPMC, QDAUDIO and DIV that meet the above requirements can be calculated by a trial and error approach or by using code routines. For simplicity, a trial and error approach is used with the following results: ND = 54, FRACR = 1241514, QDPMC = 53, QDAUDIO = 27 and DIV = 2, which meet the above requirements. Procedural calculation routines are available as part of SAMA5D2 Linux BSP. Specifically, the Linux clock driver available at https://github.com/linux4sam/linux-at91/tree/master/drivers/clk/at91 can be taken as reference.

With the above calculated Audio PLL parameters, the audio core clock is as follows:

$f_{audiocoreclock}$ = 12000000 * (54 + 1 + (1241514 / 2^22)) = 663.552 MHz and the Audio PLL clock fed to PMC will be,

$f_{audiopllck}$ = 663.552 MHz / (53 + 1) = 12.288 MHz and the audio pin clock fed to CLK_AUDIO pin will be,

$f_{audiopinclk}$ = 663.552 MHz / (2 * 27) = 12.288 MHz.

Code Snippet:

```
/* Disable Audio PLL */
PMC->PMC_AUDIO_PLL0 = 0;

/* Set Audio PLL in active state */
PMC->PMC_AUDIO_PLL0 = PMC_AUDIO_PLL0_RESETN;

/* Set Audio PLL parameters - Numbers calculated in trial and error for
above spec */
PMC->PMC_AUDIO_PLL0 |= PMC_AUDIO_PLL0_ND(54) | PMC_AUDIO_PLL0_QDPMC(53);

PMC->PMC_AUDIO_PLL1 = PMC_AUDIO_PLL1_FRACR(1241514)
                    | PMC_AUDIO_PLL1_DIV(2)
```

```
                        | PMC_AUDIO_PLL1_QDAUDIO(27);

/* Enable Audio PLL */
PMC->PMC_AUDIO_PLL0 |= PMC_AUDIO_PLL0_PLLEN | PMC_AUDIO_PLL0_PADEN |
PMC_AUDIO_PLL0_PMCEN;

/* Wait for startup time until PLL is stabilized */
delay_us(100);
```

The above code snippet has a delay loop of 100 microseconds, which is the maximum startup time for the Audio PLL module. The code has the PADEN bit set, which makes the AUDIOPINCLK signal available in the CLK_AUDIO pin.

To use AUDIOPLLCK as the clock source for an audio peripheral, it has to be enabled in the power manager. For example, the following line of code enables AUDIOPLLCK as clock source for the PDMIC peripheral.

```
PMC->PMC_PCR = PMC_PCR_PID(ID_PDMIC)
                | PMC_PCR_GCKCSS_AUDIO_CLK
                | PMC_PCR_CMD
                | PMC_PCR_EN
                | PMC_PCR_GCKEN;
```

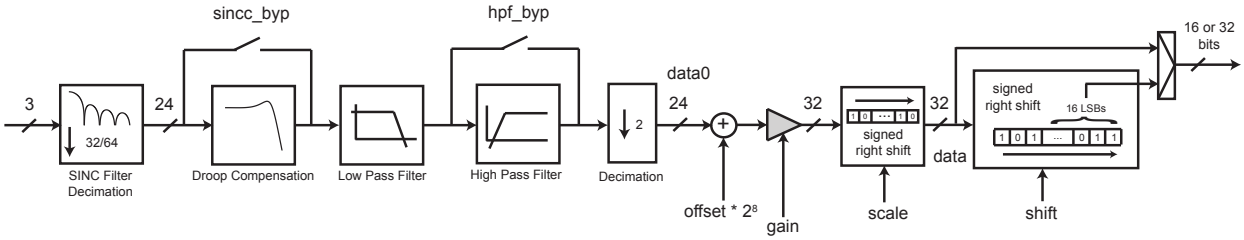
3. Pulse Density Modulation Interface Controller (PDMIC)

The PDMIC peripheral is a mono PDM decoder module that decodes an incoming PDM sample stream. The PDMIC module takes either the system clock (MCK) or the generic clock (e.g., Audio PLL PMC output) as its clock source. It then generates the PDM clock in the PDMIC_CLK pin to be fed to the connected PDM microphone and samples the data from the PDMIC_DAT pin. The PDM data is sampled on the rising edge of the PDMIC_CLK signal.

The module can output either a 16-bit or a 32-bit signed result to the result register (PDMIC_CDR – PDMIC Converted Data Register) which can be transferred to the main memory either by CPU or by DMA controller. It embeds a FIFO buffer to store up to 4 converted results.

The PDMIC module incorporates a DSP engine containing a decimation filter, a droop compensation filter, a sixth-order low-pass filter, a first order high-pass filter and an offset and gain compensation stage, as shown in the block diagram below.

Figure 3-1. PDMIC DSP Block Diagram



The overall decimation ratio of the DSP engine is either 64 or 128, which is configurable. This means, to sample an audio signal with a sample rate of 48 kHz, the clock frequency to be given to the microphone over the PDMIC_CLK pin should be (48000 * 128) = 6.144 MHz if an oversampling ratio of 128 is used, or should be (48000 * 64) = 3.072 MHz if an oversampling ratio of 64 is used.

For a more detailed description of the individual components of the DSP engine, refer to "Digital Signal Processing (Digital Filter)", in Section "Pulse Density Modulation Interface Controller (PDMIC)" of the SAMA5D2 device datasheet.

The digital output after all the decimation and filter stages is fed to an offset and gain compensation stage which follows the equation below.

$$d_{out} = \frac{\left(d_{in} + \left(offset * 2^{8}\right)\right) * dgain}{2^{(scale + shift + 8)}}$$

Where:

- d_{in} is the output from the filter stages (signed 24-bit) and input to the offset and gain compensation stage.
- d_{out} is the final output which will be placed in the PDMIC_CDR register.
- *offset* is a signed 16-bit integer which is multiplied by 2^8 to have the same weight as d_{in}.
- d_{gain} is an unsigned 15-bit integer which is set to 0 after reset. The user must configure this parameter to a non-zero value to get a valid result. The output after d_{gain} multiplication of d_{in} (with offset added) can be more than 32 bits in size. Only 32 MSB bits will be used for the next scaling and shifting stage.
- *scale* and *shift* are unsigned 4-bit integers each. The multiplication result will be shifted right by (*scale* + *shift* + 8) bits.

When the 32-bit result mode is selected, the scaling and shifting stage division is not performed and the 32-bit multiplication result (32-bit MSBs) is placed in the result register. When the 16-bit result mode is selected, the scaling and shifting stage division is performed. The result is then saturated to be within $\pm(2^{15} - 1)$ and the 16 LSBs of this saturation operation are placed in the result register.

3.1. Code Example - PDMIC

The following code example demonstrates the steps to record mono audio data using a PDMIC interface at a 48-kHz sample rate for 10 seconds with a sample size being signed 16-bit. A PDM microphone is connected to the PDMIC interface through I/O pins PB26 (PDMIC_DAT) and PB27 (PDMIC_CLK). The converted data is stored in a buffer in the DDR memory on the SAMA5D2 Xplained board.

3.1.1. Audio PLL Initialization

The code snippet given below initializes the Audio PLL to output AUDIOPLLCK at a frequency of 98.304 MHz with a 12-MHz reference input clock frequency.

```
/* Disable Audio PLL */
PMC->PMC_AUDIO_PLL0 = 0;

/* Set Audio PLL in active state */
PMC->PMC_AUDIO_PLL0 = PMC_AUDIO_PLL0_RESETN;

/* Set Audio PLL parameters - Numbers calculated in trial and error for
above spec */
PMC->PMC_AUDIO_PLL0 |= PMC_AUDIO_PLL0_ND(56) | PMC_AUDIO_PLL0_QDPMC(6);

PMC->PMC_AUDIO_PLL1 = PMC_AUDIO_PLL1_FRACR(1442840);

/* Enable Audio PLL */
PMC->PMC_AUDIO_PLL0 |= PMC_AUDIO_PLL0_PLLEN | PMC_AUDIO_PLL0_PMCEN;

/* Wait for startup time until PLL is stabilized */
delay_us(100);
```

A frequency value of 98.304 MHz is chosen because the same frequency can be used to source the PDMIC interface as well as the CLASSD interface which will be discussed in the next chapter.

3.1.2. PDMIC I/O Pin Initialization

PIO pins PB26 and PB27 with their peripheral function D will be assigned to the PDMIC interface. The code snippet given below will initialize the I/O pins accordingly.

```
/* Set PORTB mask register bit 26 */
PIOA->PIO_IO_GROUP[1].PIO_MSKR = (1u << 26);

/* Enable peripheral function D and set I/O pin as input */
PIOA->PIO_IO_GROUP[1].PIO_CFGR = PIO_CFGR_FUNC_PERIPH_D |
PIO_CFGR_DIR_INPUT;

/* Set PORTB mask register bit 27 */
PIOA->PIO_IO_GROUP[1].PIO_MSKR = (1u << 27);

/* Enable peripheral function D and set I/O pin as output */
PIOA->PIO_IO_GROUP[1].PIO_CFGR = PIO_CFGR_FUNC_PERIPH_D |
PIO_CFGR_DIR_OUTPUT;
```

3.1.3. PDMIC XDMAC Channel Initialization

A DMA channel from the XDMAC module is configured to transfer the converted data from the PDMIC interface to the main memory buffer. The code snippet given below configures the XDMAC channel 0 in a single block – Single Micro Block mode with micro block length equal to 10*48000 (48-kHz sample rate for 10 seconds).

```
/* Enable peripheral clock for XDMAC0 */
PMC->PMC_PCER0 = (1u << ID_XDMAC0);

/* Read the interrupt status register to clear the interrupt flags */
temp = XDMAC0->XDMAC_CHID[0].XDMAC_CIS;

/* Set source address as PDMIC_CDR register */
XDMAC0->XDMAC_CHID[0].XDMAC_CSA = (uint32_t)&PDMIC->PDMIC_CDR;

/* Set destination address as starting address of audio buffer */
XDMAC0->XDMAC_CHID[0].XDMAC_CDA = (uint32_t)audio_data;

/* Set micro block length */
XDMAC0->XDMAC_CHID[0].XDMAC_CUBC = 10*48000;

/* Set DMA channel parameters */
XDMAC0->XDMAC_CHID[0].XDMAC_CC =
XDMAC_CC_TYPE_PER_TRAN
                                    | XDMAC_CC_MBSIZE_SINGLE
                                    |
XDMAC_CC_DSYNC_PER2MEM
                                    | XDMAC_CC_CSIZE_CHK_1
                                    | XDMAC_CC_DWIDTH_HALFWORD
                                    | XDMAC_CC_SIF_AHB_IF1
                                    | XDMAC_CC_DIF_AHB_IF0
                                    | XDMAC_CC_SAM_FIXED_AM
                                    | XDMAC_CC_DAM_INCREMENTED_AM
                                    | XDMAC_CC_PERID(50);

/* Set all registers related to descriptor to 0 */
XDMAC0->XDMAC_CHID[0].XDMAC_CNDC = 0;
XDMAC0->XDMAC_CHID[0].XDMAC_CBC = 0;
XDMAC0->XDMAC_CHID[0].XDMAC_CDS_MSP = 0;
XDMAC0->XDMAC_CHID[0].XDMAC_CSUS = 0;
XDMAC0->XDMAC_CHID[0].XDMAC_CDUS = 0;
```

The DMA channel source address is set to the PDMIC_CDR result data register, and the destination address is set to the start of the buffer. Here, "audio_data" is an array of type signed 16-bit integer and of size 10*48000. The transfer size is set to half-word with a fixed source address and a destination address incrementing for each transfer.

No descriptors are used for the transfer, so the XDMAC registers related to the descriptor configuration are set to 0.

3.1.4. PDMIC Initialization

The following code snippet initializes the PDMIC interface with its clock source set to Audio PLL through PMC's GCLK source.

```
/* Enable Audio PLL as source for PDMIC through GCLK */
PMC->PMC_PCR = PMC_PCR_PID(ID_PDMIC)
               | PMC_PCR_GCKCSS_AUDIO_CLK
               | PMC_PCR_CMD
               | PMC_PCR_GCKDIV(7)
```

```
                | PMC_PCR_EN
                | PMC_PCR_GCKEN;

/* Wait until GCLK is ready */
while (!(PMC->PMC_SR & PMC_SR_GCKRDY));

/* Perform software reset of PDMIC peripheral */
PDMIC->PDMIC_CR = PDMIC_CR_SWRST;

/* Select GCLK as clock source and set perscaler */
PDMIC->PDMIC_MR = (1u << 4) | PDMIC_MR_PRESCAL(1);

/* Set oversampling ratio to 64 */
PDMIC->PDMIC_DSPR0 = PDMIC_DSPR0_OSR(1);

/* Set DGAIN to 1 */
PDMIC->PDMIC_DSPR1 = PDMIC_DSPR1_DGAIN(1);

/* Enable PDMIC */
PDMIC->PDMIC_CR = PDMIC_CR_ENPDM;

/* Enable DMA channel */
XDMAC0->XDMAC_GE = XDMAC_GE_EN0;

/* Wait until DMA transfer is done */
while(!(XDMAC0->XDMAC_CHID[0].XDMAC_CIS & XDMAC_CIS_BIS));
```

The Audio PLL clock output of 98.304 MHz is fed to the PDMIC module after prescaling it down to 12.288 MHz using the GCLK controller in PMC (GCKDIV is set to 7 so the GCLK division factor is 8).

The PDMIC module is configured with an oversampling ratio of 64 with unity gain and then enabled. Once enabled, the PDMIC generates the PDM clock and then starts to convert audio data.

The XDMAC channel is also enabled which starts transferring the converted result from PDMIC to the buffer in the main memory. The XDMAC channel block interrupt flag (BIS) is set once the specified number of micro block length data are transferred to the main memory and the CPU comes out of the last while loop in the above code snippet.

The overall gain of the PDMIC module can be configured by modifying the DGAIN and SCALE parameters which follow the equation below (assuming offset and shift are zero).

$$Gain\ (dB) = 20{*}log\left(\frac{2^{(scale\ +\ 8)}}{dgain}\right)$$

4. Audio Class D Amplifier (CLASSD)

The Audio Class D Amplifier (CLASSD) is a digital input, Pulse Width Modulated (PWM) output stereo Class D amplifier. It features a high quality interpolation filter embedding a digitally-controlled gain, an equalizer and a de-emphasis filter.

The CLASSD takes 16-bit signed data with most common audio sample rates at its input and generates PWM output that can drive either:

- High-impedance single-ended or differential output loads (Audio DAC application), or
- External MOSFETs through an integrated non-overlapping circuit (Class D power amplifier application)

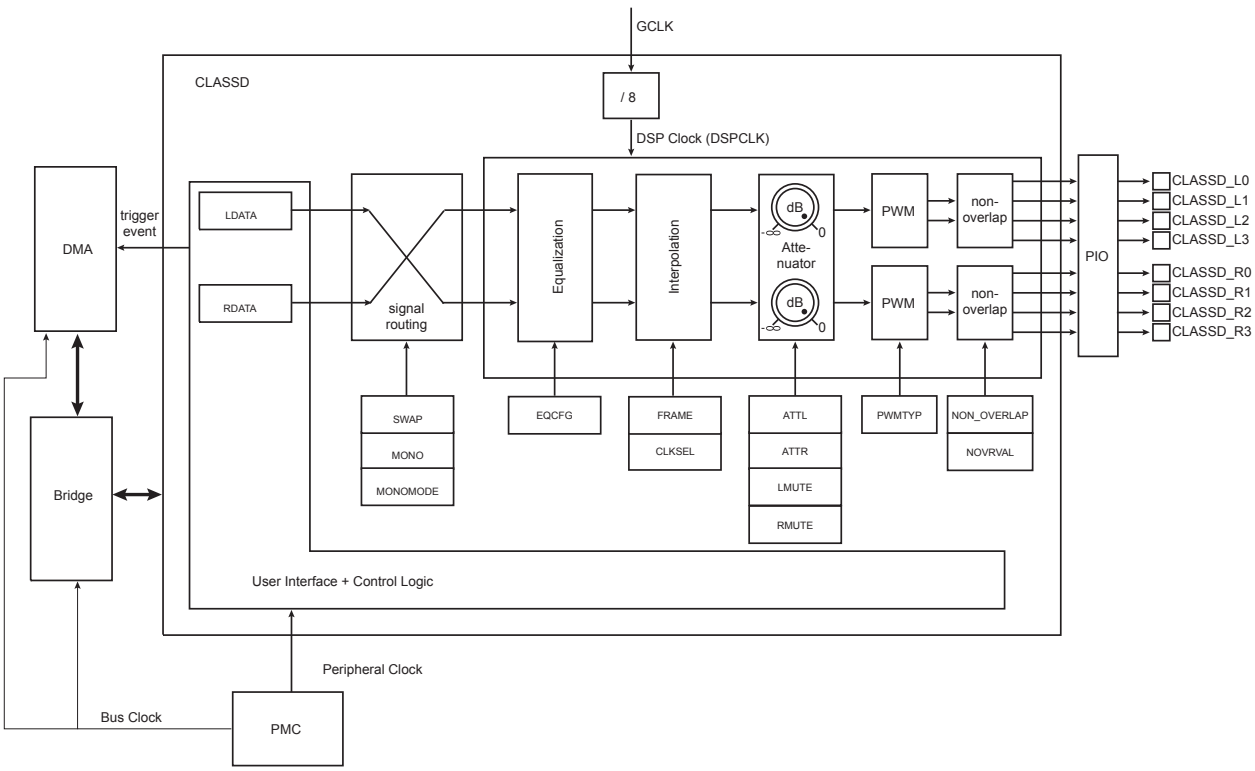
The DSP section of the CLASSD has been designed to run at two clock frequencies (DSPCLK - 12.288 MHz and 11.289 MHz) and supports input sampling rates as listed below:

8 kHz, 16 kHz, 22.05 kHz, 32 kHz, 44.1 kHz, 48 kHz, 88.2 kHz and 96 kHz.

The CLASSD peripheral has an internal fixed divide by 8 prescaler before the DSP section, so the generic clock frequency from the power manager (PMC) to the CLASSD should be 8 times the desired DSP clock. For example, if DSPCLK is 12.288 MHz, the GCLK should be (12.288 MHz * 8) = 98.304 MHz. The Audio PLL has to be configured to generate the GCLK clock accordingly.

The following figure shows the CLASSD amplifier block diagram.

Figure 4-1. CLASSD Block Diagram



The CLASSD has an interpolator, an equalizer, a de-emphasis filter, an attenuator and a PWM stage. The following sections give a brief description of each DSP component. Refer to the SAMA5D2 datasheet section "Audio Class D Amplifier (CLASSD)" for more descriptive information about each DSP components of the CLASSD module.

4.1. CLASSD Interpolator

The CLASSD interpolator embeds three filters with different frequency responses. One of the three filters is used for a given DSPCLK and input sample rate. Refer to Table 4-1 for the filter type for a given DSPCLK and sample rate.

Interpolation is performed with a combination of Infinite Impulse Response (IIR) and Cascaded Integrator-Comb (CIC) filters. Given an input configuration, the filter's coefficients are automatically redefined to optimize the filter's transfer function in order to optimize the audio bandwidth.

4.2. CLASSD Equalizer

The CLASSD offers 12 programmable equalization filters as listed below.

1. Bass boost +12 dB
2. Bass boost +6 dB
3. Bass cut -12 dB
4. Bass cut -6 dB
5. Medium boost +3 dB
6. Medium boost +8 dB
7. Medium cut -3 dB
8. Medium cut -8 dB
9. Treble boost +12 dB
10. Treble boost +6 dB
11. Treble cut -12 dB
12. Treble cut -6 dB

A zero-cross detection system is used to modify the equalizer on-the-fly with minimum disturbance of the output signal.

4.3. CLASSD De-emphasis Filter

The CLASSD includes a de-emphasis filter to attenuate the high frequency components. This filter can only be used with input sample rates 32 kHz, 44.1 kHz and 48 kHz.

4.4. CLASSD Attenuator

The CLASSD features a digital attenuator with an attenuation range of 0–77 dB and a step size of 1 dB. When a greater than 77 dB attenuation is programmed, the attenuator mutes the channel. Attenuation can be done individually for both right and left channels.

The following table shows the allowed settings for input sample rates, DSPCLK and filter support for each available sample rate.

Table 4-1. DSPCLK and Sample Rates with Respective Filter Types

Sample Rate (fs)	Filter Type / DSPCLK		Equalizer	De-emphasis Filter	Attenuator
	12.288 MHz	11.2896 MHz			
8 kHz	2	-	Y	N	Y
16 kHz	2	-	Y	N	Y
32 kHz	2	-	Y	Y	Y
48 kHz	1	-	Y	Y	Y
96 kHz	3	-	Y	N	Y
22.05 kHz	-	1	Y	N	Y
44.1 kHz	-	1	Y	Y	Y
88.2 kHz	-	3	Y	N	Y

Note:

1. The "Filter Type / DSPCLK" column in the above table without a numerical entry implies an unavailable CLASSD setting. Such configuration will raise the Configuration Error (CFGERR) flag.
2. Y – DSP component supported.
3. N – DSP component not supported.
4. For the attenuator, a minimum attenuation of 1dB should be configured to avoid saturation in the PWM stage.

4.5. CLASSD PWM Stage

The CLASSD Pulse Width Modulator (PWM) generates fixed-frequency PWM outputs. The following table shows the PWM frequency generated for different sample rates and conditions.

Table 4-2. PWM Frequencies for Different Sample Rates

Sample Rate (fs)	PWM Frequency Calculation	PWM Frequency
44.1 kHz	16 * fs	705.6 kHz
48 kHz		768 kHz
8 kHz	Adapted 16x Interpolation	768 kHz
16 kHz		
24 kHz		
96 kHz		
22.05 kHz		705.6 kHz
88.2 kHz		

Depending on the NON_OVERLAP bit value in mode register (CLASSD_MR), the CLASSD can:

- Work as a DAC, loaded by a medium-to-high resistive load (1 kΩ to 100 kΩ) – Single ended or differential resistive loads (NON_OVERLAP = 0), or

- Work as a CLASS D power amplifier driving external power stage – Full or Half MOSFET H-bridges (NON_OVERLAP = 1)

When driving an external power stage (NON_OVERLAP = 1), the CLASSD generates the signals to control complementary MOSFET pairs (PMOS and NMOS) with a non-overlapping delay between the NMOS and PMOS controls to avoid short-circuit current. The non-overlapping delay can be adjusted in the CLASSD_MR.NOVRVAL field.

For each NON_OVERLAP bit value, the PWM stage can generate a single-ended or differential output depending on the PWMTYP bit available in mode register.

For a single-ended output (CLASSD_MR.PWMTYP = 0), the PWM acts only on the falling edge of the PWM waveform (trailing edge PWM). For a differential output (CLASSD_MR.PWMTYP = 1), both the rising and the falling edges of the PWM waveform are modulated (symmetric PWM).

The following figures show the modulated PWM output waveforms when PWMTYP = 0 and when PWMTYP = 1 respectively.

Figure 4-2. Output Waveform Modulation for PWMTYP = 0

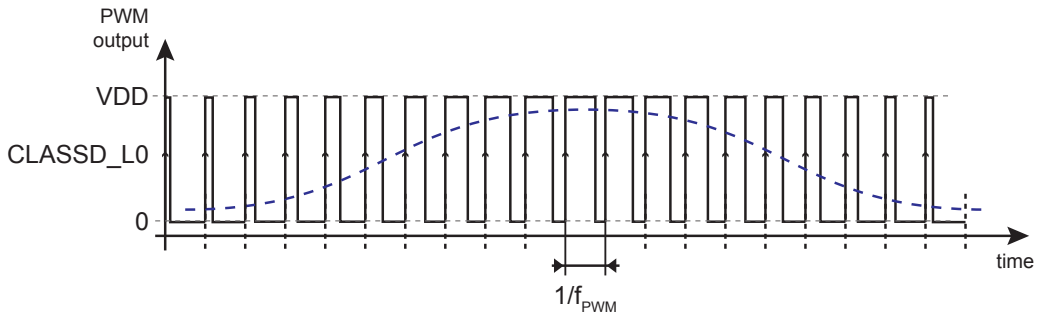
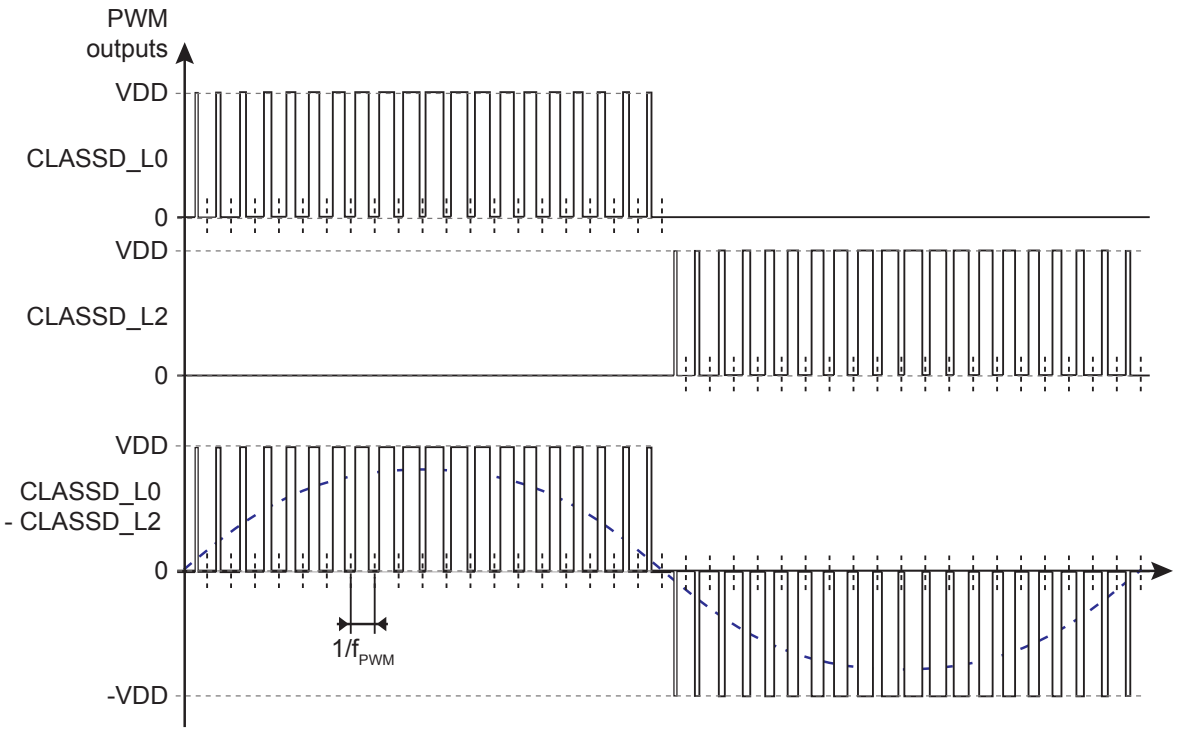


Figure 4-3. Output Waveform Modulation for PWMTYP = 1 (Only left channel pins shown)



To summarize the CLASSD outputs for the left channel:

1. When NON_OVERLAP = 0, in Single-ended mode (PWMTYP = 0), only one PWM output signal is generated per channel (L0 for the left channel) and it covers both positive and negative sides of the analog output.
2. When NON_OVERLAP = 0, in Differential mode (PWMTYP = 1), two PWM outputs are generated per channel (L0 and L2 for the left channel) where one PWM output covers the positive half and another PWM output covers the negative half of the analog output.
3. When NON_OVERLAP = 1, in Single-ended mode (PWMTYP = 0), two PWM outputs are generated per channel (L0 and L1 for the left channel) to drive a single PMOS-NMOS complementary pair. These two PWM outputs are exactly the same, with a small delay between them during the level transition (edges) to avoid short-circuit. The output from the PMOS-NMOS complementary pair is now a single PWM signal covering both the positive and negative sides of the analog output.
4. When NON_OVERLAP = 1, in Differential mode (PWMTYP = 1), four PWM outputs are generated per channel (L0, L1, L2 and L3) to drive two PMOS-NMOS complementary pairs. The first two PWM outputs (L0 and L1) are used to drive the upper PMOS-NMOS complementary pair and the next two PWM outputs (L2 and L3) are used to drive the lower PMOS-NMOS complementary pair. The effective PWM signal from the upper complementary pair covers the positive side of the analog output and the effective PWM signal from the lower complementary pair covers the negative side of the analog output. Individual delay is applied to the PWM inputs of each complementary pair just like in case 3 above, to avoid short-circuit.

The table below lists the combinations of the possible PWM modulations with the corresponding I/O pins.

Table 4-3. CLASSD Signal and Pin Assignment for Different Modulation Settings

I/O Pin & Peripheral Function	CLASSD Signal	External MOS Driver (NON_OVERLAP = 1)		Direct Load (NON_OVERLAP = 0)	
		Full H-Bridge (PWMTYP = 1)	Half H-Bridge (PWMTYP = 0)	Differential Load (PWMTYP = 1)	Single-Ended Load (PWMTYP = 0)
		Case 4	Case 3	Case 2	Case 1
PA28 - F	CLASSD_L0	left_pos_pmos	left_pmos	left_pos	left
PA29 - F	CLASSD_L1	left_pos_nmos	left_nmos	unused	unused
PA30 - F	CLASSD_L2	left_neg_pmos	unused	left_neg	unused
PA31 - F	CLASSD_L3	left_neg_nmos	unused	unused	unused
PB1 - F	CLASSD_R0	right_pos_pmos	right_pmos	right_pos	right
PB2 - F	CLASSD_R1	right_pos_nmos	right_nmos	unused	unused
PB3 - F	CLASSD_R2	right_neg_pmos	unused	right_neg	unused
PB4 - F	CLASSD_R3	right_neg_nmos	unused	unused	unused

4.6. Code Examples

The following code example demonstrates the steps to playback audio data from a buffer using the CLASSD interface at a 48-kHz sample rate with a stereo sample size signed 16 bits. The Audio PLL initialization is the same as in Code Example - Audio PLL.

4.6.1. CLASSD I/O Pin Initialization

PIO pins PA28, PA29, PA30, PA31 (for the left channel) and PB1, PB2, PB3, PB4 (for the right channel) with peripheral function F will be assigned to the CLASSD interface. The code snippet given below will initialize the I/O pins accordingly.

```
/* Set PORTA mask register bits 28, 29, 30 & 31 */
PIOA->PIO_IO_GROUP[0].PIO_MSKR = (0xF << 28);

/* Enable peripheral function F and set I/O pins as output */
PIOA->PIO_IO_GROUP[0].PIO_CFGR = PIO_CFGR_FUNC_PERIPH_F |
PIO_CFGR_DIR_OUTPUT;

/* Set PORTB mask register bits 1, 2, 3 & 4 */
PIOA->PIO_IO_GROUP[1].PIO_MSKR = (0xF << 1);

/* Enable peripheral function F and set I/O pins as output */
PIOA->PIO_IO_GROUP[1].PIO_CFGR = PIO_CFGR_FUNC_PERIPH_F |
PIO_CFGR_DIR_OUTPUT;
```

4.6.2. XDMAC Channel Initialization

A DMA channel from the XDMAC module is configured to transfer the audio samples from the main memory buffer to the CLASSD data register. The code snippet below configures the XDMAC channel 0 in a single block – Single Micro Block mode with micro block length equal to the size of the buffer in bytes, divided by 4. The division factor is set to 4 because the buffer is arranged as an array of 16-bit signed integers with alternating left channel and right channel data. Array index 0 is the left channel data of sample 0, and array index 1 is the right channel data of sample 0, and so on. So, each successive four bytes in the array amounts to one audio sample.

```
/* Enable peripheral clock for XDMAC0 */
PMC->PMC_PCER0 = (1u << ID_XDMAC0);

/* Read the interrupt status register to clear the interrupt flags */
temp = XDMAC0->XDMAC_CHID[0].XDMAC_CIS;

/* Set source address as starting address of audio buffer */
XDMAC0->XDMAC_CHID[0].XDMAC_CSA = (uint32_t)audio_data;

/* Set destination address as CLASSD_THR register */
XDMAC0->XDMAC_CHID[0].XDMAC_CDA = (uint32_t)&CLASSD->CLASSD_THR;

/* Set micro block length */
XDMAC0->XDMAC_CHID[0].XDMAC_CUBC = sizeof(audio_data)/4;

/* Set DMA channel parameters */
XDMAC0->XDMAC_CHID[0].XDMAC_CC = XDMAC_CC_TYPE_PER_TRAN
                                 | XDMAC_CC_MBSIZE_SINGLE
                                 | XDMAC_CC_DSYNC_MEM2PER
                                 | XDMAC_CC_CSIZE_CHK_1
                                 | XDMAC_CC_DWIDTH_WORD
                                 | XDMAC_CC_SIF_AHB_IF0
                                 | XDMAC_CC_DIF_AHB_IF1
                                 | XDMAC_CC_SAM_INCREMENTED_AM
                                 | XDMAC_CC_DAM_FIXED_AM
                                 | XDMAC_CC_PERID(47);

/* Set all registers related to descriptor to 0 */
XDMAC0->XDMAC_CHID[0].XDMAC_CNDC = 0;
XDMAC0->XDMAC_CHID[0].XDMAC_CBC = 0;
```

```
XDMAC0->XDMAC_CHID[0].XDMAC_CDS_MSP = 0;
XDMAC0->XDMAC_CHID[0].XDMAC_CSUS = 0;
XDMAC0->XDMAC_CHID[0].XDMAC_CDUS = 0;
```

The DMA channel source address is set to the start of the buffer and the destination address is set to the CLASSD transmit holding register. The transfer size is set to one word (one audio sample per transfer) with source address incrementing and destination address fixed for each transfer.

No descriptors are used for the transfer, so the XDMAC registers related to the descriptor configuration are set to 0.

4.6.3. CLASSD Initialization

The following code snippet initializes the CLASSD interface with its clock source set to Audio PLL through PMC's GCLK source.

```
/* Enable Audio PLL as source for CLASSD through GCLK */
PMC->PMC_PCR = PMC_PCR_PID(ID_CLASSD)
                 | PMC_PCR_GCKCSS_AUDIO_CLK
                 | PMC_PCR_CMD
                 | PMC_PCR_EN
                 | PMC_PCR_GCKEN;

/* Wait until GCLK is ready */
while (!(PMC->PMC_SR & PMC_SR_GCKRDY));

/* Perform software reset of CLASSD peripheral */
CLASSD->CLASSD_CR = CLASSD_CR_SWRST;

/* Configure CLASSD parameters */
CLASSD->CLASSD_MR = CLASSD_MR_LEN
                      | CLASSD_MR_REN
                      | CLASSD_MR_PWMTYP
                      | CLASSD_MR_NON_OVERLAP
                      | CLASSD_MR_NOVRVAL_20NS;

CLASSD->CLASSD_INTPMR = CLASSD_INTPMR_DSPCLKFREQ_12M288
                          | CLASSD_INTPMR_FRAME_FRAME_48K
                          | CLASSD_INTPMR_ATTL(10)
                          | CLASSD_INTPMR_ATTR(10);

/* Enable DMA channel */
XDMAC0->XDMAC_GE = XDMAC_GE_EN0;

/* Wait until DMA transfer is done */
while(!(XDMAC0->XDMAC_CHID[0].XDMAC_CIS & XDMAC_CIS_BIS));
```

The Audio PLL clock output of 98.304 MHz is fed to the CLASSD module using the GCLK controller in PMC. The CLASSD has an internal fixed divide by 8 prescaler which provides the 12.288 MHz clock to the DSP engine of the CLASSD module.

The CLASSD module is configured in Non-overlap mode with PWMTYP=1 and a non-overlap delay of 20 ns. The interpolator is configured for 12.288 MHz and a sample rate of 48 kHz. The attenuation for the left and right channels are set to -10dB.

The XDMAC channel is enabled, which starts the audio sample transfers from the buffer to the CLASSD transmit holding register, one audio sample (with both left and right channel data) at a time. The XDMAC channel block interrupt flag (BIS) will be set once the specified number of micro block length data are transferred and the CPU will come out of the last while loop in the above code snippet.

5. Inter-IC Sound Controller (I2SC)

The Inter-IC Sound Controller (I2SC) is an I2S bus specification-compliant 5-wire, bidirectional, synchronous digital audio interface that can communicate with external I2S devices like audio codecs.

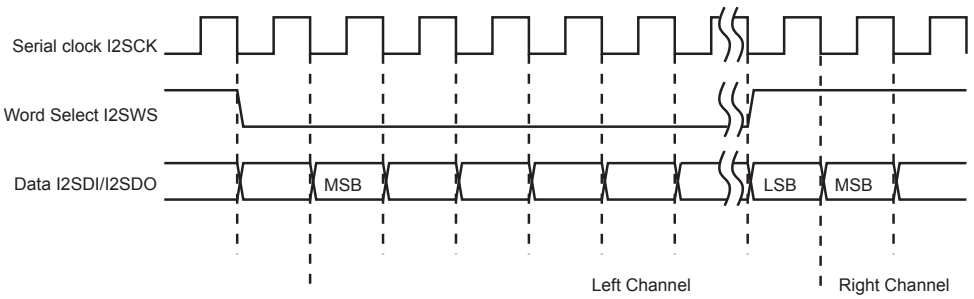
The I2SC signals are:

- I2SDI – I2S Data Input line
- I2SDO – I2S Data Output line
- I2SWS – I2S Word Select line
- I2SCK – I2S Bit Clock line
- I2SMCK – I2S Master Clock line

5.1. I2S Frame

The figure below depicts a standard I2S bus frame structure.

Figure 5-1. I2S Frame Structure



I2S transfers data synchronously based on the bit clock (I2SCK). Data bits are set up on the falling edge of the bit clock and sampled on the rising edge of the bit clock. The word select line (I2SWS) is used to identify which audio channel (left or right) the current data bits correspond to. Typically, the word select line is held low when transmitting left channel data, and held high when transmitting right channel data.

As per the I2S protocol, data bits are left-justified with the MSB transmitted first, starting one bit clock period after the transition in the word select line.

An I2S bus typically includes one more clock line called the master clock line. The frequency of the master clock signal is a 2^x integer multiple of the audio sample rate fs, for example 256*fs. This master clock signal will be used by the external audio codec device to time its internal circuitry.

5.2. I2SC Modes

The I2SC supports Master, Slave and Controller modes with an integrated transmitter, a receiver and a clock generator that can be enabled separately. Data transfer can be done either through the CPU or through the DMA controller with a single DMA channel for both audio channels (left and right channels).

In Master and Controller modes, the I2SC provides the master clock, the serial clock and the word select signal. I2SMCK, I2SCK, and I2SWS pins are outputs.

In Controller mode, the I2SC receiver and transmitter are disabled. Only the clocks are enabled and used by an external receiver and/or transmitter.

In Slave mode, the I2SC receives the serial clock and the word select from an external master. I2SCK and I2SWS pins are inputs.

The mode is selected by writing the MODE field in the I2SC_MR register. Since the MODE field changes the direction of the I2SWS and I2SCK pins, the I2SC_MR must be written when the I2SC is stopped.

The I2SC supports a Loop-back mode which loops back the signal from transmitter to receiver. Writing a '1' to the I2SC_MR.LOOP bit internally connects I2SDO to I2SDI, so that the transmitted data is also received.

5.3. I2SC Supported Sample Rates and Data Formats

I2SC supports a wide range of audio sample rates including 32 kHz, 44.1 kHz, 48 kHz, 88.2 kHz, 96 kHz and 192 kHz.

In Master mode, the I2SC can generate a 32*fs to 1024*fs master clock (I2SMCK) that provides an over-sampling clock to an external audio codec or a digital signal processor (DSP).

It supports multiple data formats that include 32-bit, 24-bit, 20-bit, 18-bit, 16-bit and 8-bit mono and stereo formats. 16-bit and 8-bit compact stereo formats are supported with left and right channel samples packed in the same word to reduce data transfers.

The slot length in an I2S frame can be defined as the number of bits occupied per channel, which includes both the actual data bits and the unused pad bits (zeros). The slot length in I2SC is configurable using the bit field DATALENGTH in mode register (I2SC_MR). The table below shows the possible values for the DATALENGTH field and their corresponding slot lengths.

Table 5-1. I2SC DATALENGTH Settings and Corresponding Slot Length

I2SC_MR.DATALENGTH	Word Length (useful data)	Slot Length (useful data + pad bits)
0	32 bits	32
1	24 bits	32 if I2SC_MR.IWS = 0 24 if I2SC_MR.IWS = 1
2	20 bits	
3	18 bits	
4	16 bits	16
5	16 bits compact stereo	
6	8 bits	8
7	8 bits compact stereo	

The data words are right-justified in the receive holding register (I2SC_RHR) and the transmit holding register (I2SC_THR). The I2SC_RHR and I2SC_THR registers are used for data reception and transmission respectively.

For the 16-bit compact stereo data format, the left sample uses bits 15:0 and the right sample uses bits 31:16 of the same data word. For the 8-bit compact stereo data format, the left sample uses bits 7:0 and the right sample uses bits 15:8 of the same data word.

The I2SC supports mono audio format where only the left channel is used. When the Transmit Mono bit (TXMONO) in I2SC_MR is set, data written to the left channel is duplicated to the right output channel.

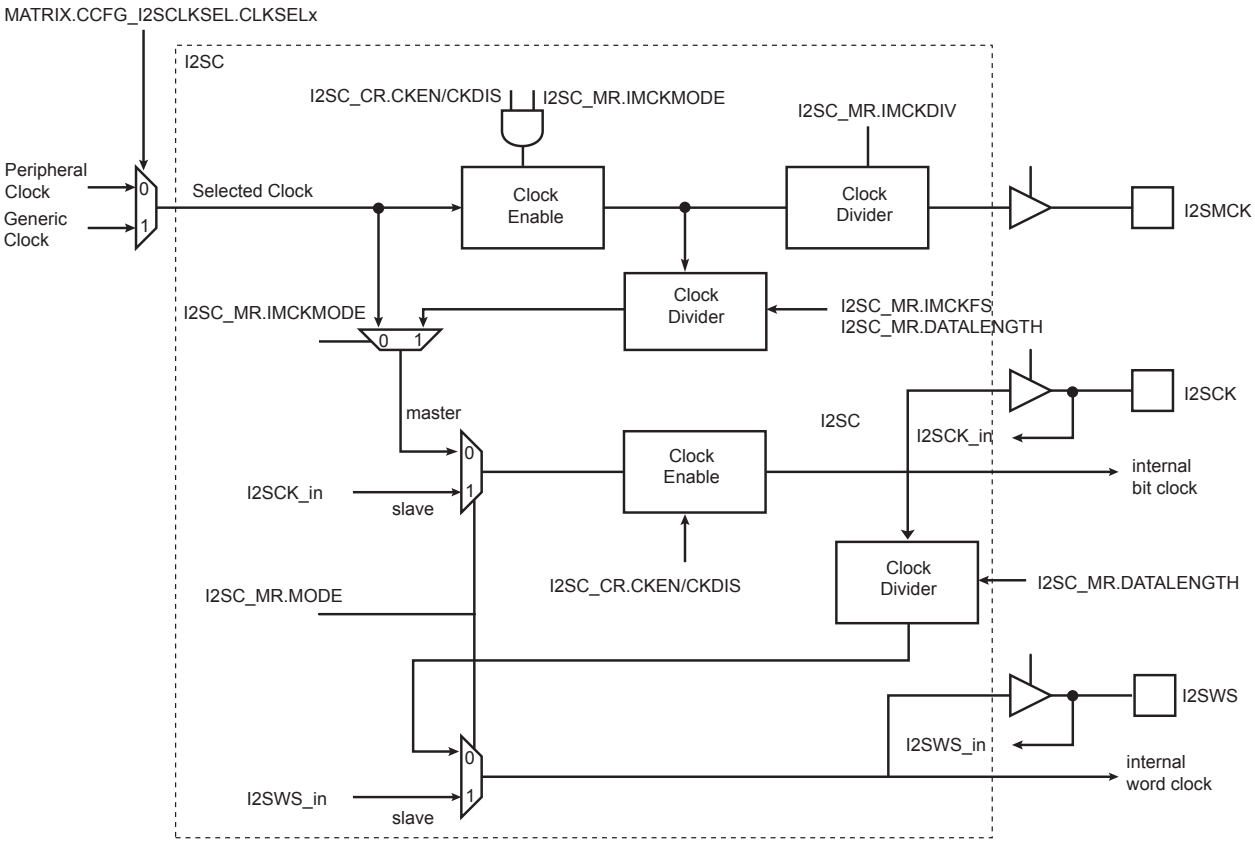
When the Receive Mono bit (RXMONO) in I2SC_MR is set, data received from the left channel is duplicated to the right channel.

5.4. I2SC Clock Generator

The I2SC integrates a clock generator that generates the master clock (I2SMCK) and the bit clock (I2SCK) in Master mode and in Controller mode.

The following figure shows the block diagram of the I2SC internal clock generator.

Figure 5-2. I2SC Clock Generator



The input to the I2SC clock generator can be either the peripheral clock (MCK) or the generic clock (GCLK) from the power manager (PMC) which is selected by the bit CLKSELx in matrix special function register SFR_I2SCLKSEL. The ‘x’ in CLKSELx stands for the I2SC instance number. SAMA5D2 device has a maximum of two I2SC instances. One CLKSEL bit per instance is available. In Master mode, if the peripheral clock frequency is higher than 96 MHz, the GCLK clock from PMC must be selected as I2SC input clock.

The I2SC master clock (I2SMCK) is derived from the selected clock. It has a master clock gate and a master clock divider in its path. The master clock gate enables/disables the I2SMCK that is controlled by register bits I2SC_CR.CKEN/CKDIS and I2SC_MR.IMCKMODE as shown in the figure. The master clock divider follows the master clock gate which prescales the input clock by the division factor (I2SC_MR.IMCKDIV +1).

Assuming the I2SC master clock is 256*fs and the audio sample rate is 44.1 kHz, the I2SMCK obtained is 11.2896 MHz. The selected clock can be an integer multiple of I2SMCK. Assuming the selected clock is 4 times I2SMCK, the master clock division factor IMCKDIV should be 3. The power manager should be configured to generate a peripheral clock or the generic clock which is 4 * 11.2896 MHz = 45.1584 MHz.

If the I2SC master clock is disabled (IMCKMODE = 0), the selected clock is taken as the I2SC bit clock (I2SCK). The I2SC bit clock path has a bit clock gate circuit which is controlled by the I2SC_CR.CKEN/CKDIS bits.

If the I2SC master clock is enabled (IMCKMODE = 1), the clock output from the master clock gate is taken as the I2S bit clock with a bit clock divider included in the path. The bit clock divider is controlled by two parameters, namely I2SC_MR.IMCKFS and I2SC_MR.DATALENGTH, and its division factor is equal to (I2SC_MR.IMCKFS + 1).

Basically the bit clock is defined by the following equation:

Bit Clock = fs * no. of channels * no. of bits per channel

Number of bits per channel is the slot length. For above example, fs = 44.1kHz, master clock = 256*fs, number of channels = 2, slot length = 16.

Bit clock = 44100 * 2 * 16 = 1.411200 MHz.

The bit clock divider should be configured to have a division factor of 32 (= 45.1584 MHz / 1.4112 MHz). This means that I2SC_MR.IMCKFS should be set to 31.

Note that the bit clock division factor is always equal to (fs * 2 * 16) regardless of the slot length. This means that even if the slot length is 32, the bit clock division factor is 1.411200 MHz for a sample rate of 44.1 kHz.

The word select clock signal is derived from the bit clock signal and it has the word select clock divider in its path. The division factor of this clock divider is controlled by the DATALENGTH field such that the word select frequency is the same as the audio sample rate fs.

In Slave mode, the I2SCK pin acts as an input and the bit clock is derived from this clock.

5.5. Code Examples

This section provides two code examples: one to demonstrate I2SC in Master mode and another in Slave mode. Both examples demonstrate the steps to play back a 48 kHz stereo 16-bit PCM audio with an I2SC0 module and an external audio DAC IC (AD1934 from Analog Devices).

5.5.1. I2SC Master Mode Code Example

This code example configures the I2SC0 in Master mode to generate the master clock, the bit clock and the LR clock, and to transmit the data to AD1934 DAC.

The 12.288 MHz master clock is generated from the I2SC module and fed to the MCLKI pin of AD1934. The AD1934 is configured with a sample rate set to 48 kHz and accepts the LR clock and bit clock from the external master (I2SC). The word width of AD1934 is always 32 bits and is left-justified with MSB first. It is configured to accept audio data in I2S frame format. Following are the register settings for the AD1934, configured using the SPI interface:

- PLL and Clock Control 0 --> 0x98
- DAC Control 2 --> 0x18

Other registers of AD1934 are left to their default reset values.

The audio PLL output is used to clock the I2SC0 module through the GCLK interface. The steps given in section Audio PLL Initialization can be used for this example.

The following sections provide the code snippet to configure the I2SC I/O pins, the DMA channel and the I2SC peripheral.

5.5.1.1. I2SC I/O Pin Initialization

PIO pins PC1 (I2SCK), PC2 (I2SMCK), PC3 (I2SWS) and PC5 (I2SDO) with peripheral function E will be assigned to the I2SC0 interface. The code snippet given below will initialize the I/O pins accordingly.

```
/* Set PORTC mask register bits 1, 2, 3 & 5 */
PIOA->PIO_IO_GROUP[2].PIO_MSKR = (0x17 << 1);

/* Enable peripheral function E and set I/O pins as output */
PIOA->PIO_IO_GROUP[2].PIO_CFGR = PIO_CFGR_FUNC_PERIPH_E |
PIO_CFGR_DIR_OUTPUT;
```

5.5.1.2. XDMAC Channel Initialization

A DMA channel from the XDMAC module is configured to transfer the audio data from an array in the main memory to the Transmit Holding Register (THR) of I2SC. The code snippet below configures the XDMAC channel 0 in a single block – Single Micro Block mode with micro block length equal to the total number of audio samples * 2 (for 2 channels).

The data transfer is done one channel sample (16 bits) at a time.

```
/* Enable peripheral clock for XDMAC1 */
PMC->PMC_PCER0 = (1u << ID_XDMAC1);

/* Read the interrupt status register to clear the interrupt flags */
temp = XDMAC1->XDMAC_CHID[0].XDMAC_CIS;

/* Set source address as starting address of audio buffer */
XDMAC1->XDMAC_CHID[0].XDMAC_CSA = (uint32_t)audio_samples;

/* Set destination address as I2SC_THR register's 16 MSB bits */
XDMAC1->XDMAC_CHID[0].XDMAC_CDA = ((uint32_t)&I2SC0->I2SC_THR) + 2;

/* Set micro block length */
XDMAC1->XDMAC_CHID[0].XDMAC_CUBC = sizeof(audio_samples)/2;

/* Set DMA channel parameters */
XDMAC1->XDMAC_CHID[0].XDMAC_CC = XDMAC_CC_TYPE_PER_TRAN
                                    |
XDMAC_CC_MBSIZE_SINGLE
                                    | XDMAC_CC_DSYNC_MEM2PER
                                    | XDMAC_CC_CSIZE_CHK_1
                                    | XDMAC_CC_DWIDTH_HALFWORD
                                    | XDMAC_CC_SIF_AHB_IF0
                                    | XDMAC_CC_DIF_AHB_IF1
                                    | XDMAC_CC_SAM_INCREMENTED_AM
                                    | XDMAC_CC_DAM_FIXED_AM
                                    | XDMAC_CC_PERID(31);

/* Set all registers related to descriptor to 0 */
XDMAC1->XDMAC_CHID[0].XDMAC_CNDC = 0;
XDMAC1->XDMAC_CHID[0].XDMAC_CBC = 0;
XDMAC1->XDMAC_CHID[0].XDMAC_CDS_MSP = 0;
XDMAC1->XDMAC_CHID[0].XDMAC_CSUS = 0;
XDMAC1->XDMAC_CHID[0].XDMAC_CDUS = 0;
```

The DMA channel source address is set to the start of the audio sample array, and the destination address is set to the start of 16 MSB of the THR register. This is because we need to send the 16-bit audio sample first in the MSB part of the I2SC 32-bit word. "audio_samples" is an array of the type "signed 16-bit integer". Transfer size is set to half word with source address incrementing and destination address fixed for each transfer.

No descriptors are used for the transfer, so the XDMAC registers related to the descriptor configuration are set to 0.

Note that the XDMAC1 controller is used instead of the XDMAC0 controller as described in previous chapters because the I2SC0 peripheral is connected to the APB1 bridge, whose DMA transactions are handled by the XDMAC1 controller.

5.5.1.3. I2SC Initialization

The following code snippet initializes the I2SC0 interface.

```
/* Enable Audio PLL as source for I2SC0 through GCLK */
PMC->PMC_PCR = PMC_PCR_PID(ID_I2SC0)
               | PMC_PCR_GCKCSS_AUDIO_CLK
               | PMC_PCR_GCKDIV(1)
               | PMC_PCR_CMD
               | PMC_PCR_EN
               | PMC_PCR_GCKEN;

/* Wait until GCLK is ready */
while (!(PMC->PMC_SR & PMC_SR_GCKRDY));

/* Select GCLK as clock source for I2SC0 in SFR module */
SFR->SFR_I2SCLKSEL = SFR_I2SCLKSEL_CLKSEL0;

/* Perform software reset of I2SC0 peripheral */
I2SC0->I2SC_CR = I2SC_CR_SWRST;

/* Configure I2SC0 parameters */
I2SC0->I2SC_MR = I2SC_MR_MODE_MASTER
                   | I2SC_MR_DATALENGTH_32_BITS
                   | I2SC_MR_FORMAT_I2S
                   | (3u << 16)
                   | I2SC_MR_IMCKFS(31)
                   | I2SC_MR_IMCKMODE;

/* Enable I2SC0 master clock and transmitter */
I2SC0->I2SC_CR = I2SC_CR_CKEN | I2SC_CR_TXEN;

/* Wait until transmitter is enabled */
while(!(I2SC0->I2SC_SR & I2SC_SR_TXEN));

/* Enable DMA channel */
XDMAC1->XDMAC_GE = XDMAC_GE_EN0;

/* Wait until DMA transfer is done */
while(!(XDMAC1->XDMAC_CHID[0].XDMAC_CIS & XDMAC_CIS_BIS));
```

I2SC0 is used for this example, which is configured in Master mode to generate the master clock, the bit clock and the LR clock with one LR clock period containing 64-bit clocks. AD1934 should be configured to accept all these clocks along with the data. Data is sent as a 32-bit word with original 16-bit audio data placed in 16 MSB bits and 16 LSB bits left to 0.

98.304 MHz output from Audio PLL is divided by 2 by the GCLK controller and this GCLK output (49.152 MHz) is selected as the clock source for the I2SC0 module in Special Function Register (SFR_I2SCLKSEL).

The master clock divider in I2SC0 is enabled with the master clock division factor set to 4, which generates (49.152 / 4) = 12.288 MHz in the I2SCMCK pin.

The bit clock division factor is calculated as (49.152MHz / (48000 * 2 * 16)) = 32, so IMCKFS is set to 31.

The master clock and the I2SC0 transmitter are enabled along with the DMA channel to start the transfer.

5.5.2. I2SC Slave Mode Code Example

This code example configures the I2SC0 in Slave mode, which accepts the bit clock and the LR clock and transmits the data to AD1934 DAC.

The 12.288 MHz master clock is generated from the Audio PLL as in section Code Example - Audio PLL. The generated 12.288 MHz signal from the CLK_AUDIO pin is fed to the MCLKI pin of AD1934. The AD1934 is configured to generate the bit clock and LR clock with a sample rate set to 48 kHz. The word width of AD1934 is always 32 bits and is configured to accept audio data in I2S frame format and left-justified with MSB first. Following are the register settings for the AD1934, which is configured using the SPI interface.

- PLL and Clock Control 0 --> 0x98
- DAC Control 1 --> 0x70
- DAC Control 2 --> 0x18

Other registers of AD1934 are left to their default reset values.

The same Audio PLL initialization steps given in section "Code Example - Audio PLL" can be used for this example.

The following sections provide the code snippet to configure the I2SC I/O pins, the DMA channel and the I2SC peripheral.

5.5.2.1. I2SC I/O Pin Initialization

PIO pins PC1 (I2SCK), PC3 (I2SWS) and PC5 (I2SDO) with peripheral function E will be assigned to the I2SC0 interface. The code snippet given below will initialize the I/O pins accordingly.

```
/* Set PORTC mask register bits 1 & 3 */
PIOA->PIO_IO_GROUP[2].PIO_MSKR = (0x5 << 1);

/* Enable peripheral function E and set I/O pins as output */
PIOA->PIO_IO_GROUP[2].PIO_CFGR = PIO_CFGR_FUNC_PERIPH_E |
PIO_CFGR_DIR_INPUT;

/* Set PORTC mask register bit 5 */
PIOA->PIO_IO_GROUP[2].PIO_MSKR = (0x1 << 5);

/* Enable peripheral function E and set I/O pins as output */
PIOA->PIO_IO_GROUP[2].PIO_CFGR = PIO_CFGR_FUNC_PERIPH_E |
PIO_CFGR_DIR_OUTPUT;
```

5.5.2.2. XDMAC Channel Initialization

The DMA channel initialization is same as in section I2SC I/O Pin Initialization.

5.5.2.3. I2SC Initialization

The following code snippet initializes the I2SC0 interface.

```
/* Enable system clock for I2SC0 */
PMC->PMC_PCR = PMC_PCR_PID(ID_I2SC0)
               | PMC_PCR_CMD
               | PMC_PCR_EN;

/* Perform software reset of I2SC0 peripheral */
I2SC0->I2SC_CR = I2SC_CR_SWRST;

/* Configure I2SC0 parameters */
I2SC0->I2SC_MR = I2SC_MR_MODE_SLAVE
```

```
                    | I2SC_MR_DATALENGTH_32_BITS
                    | I2SC_MR_FORMAT_I2S;

/* Enable I2SC0 transmitter */
I2SC0->I2SC_CR = I2SC_CR_TXEN;

/* Wait until transmitter is enabled */
while(!(I2SC0->I2SC_SR & I2SC_SR_TXEN));

/* Enable DMA channel */
XDMAC1->XDMAC_GE = XDMAC_GE_EN0;

/* Wait until DMA transfer is done */
while(!(XDMAC1->XDMAC_CHID[0].XDMAC_CIS & XDMAC_CIS_BIS));
```

I2SC0 is used for this example, which is configured in Slave mode to accept the bit clock and the LR clock with one LR clock period containing 64 bit clocks. A 12.288-MHz signal generated using Audio PLL is fed through the CLK_AUDIO pin as the master clock for AD1934. AD1934 is configured to generate the bit clock and the LR clock and received data samples from the I2SC0 slave. Data is sent as 32-bit word with original 16-bit audio data placed in 16 MSB bits and 16 LSB bits left to 0.

I2SC0 is not clocked from the Audio PLL clock in this case and it is sufficient to clock it with the default power manager system clock.

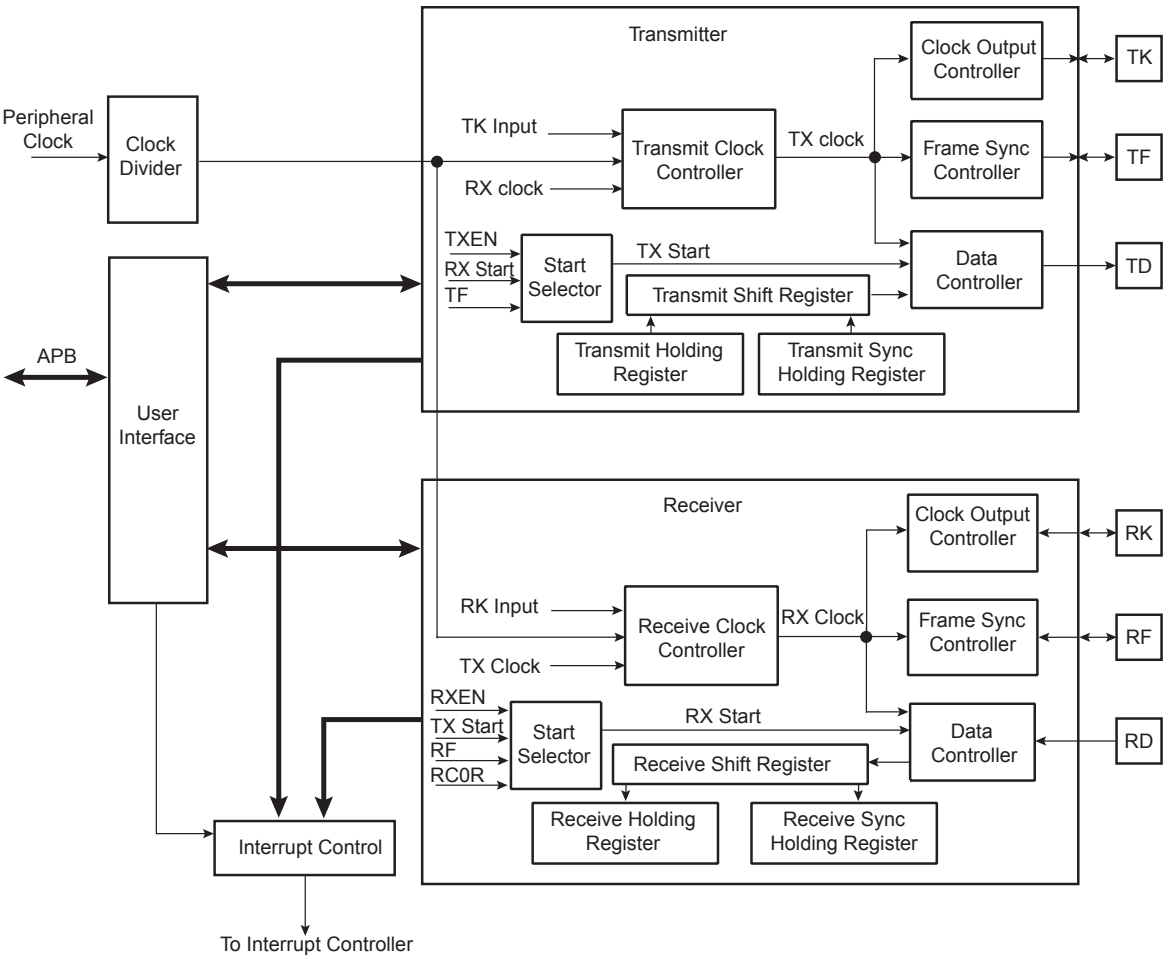
The I2SC0 transmitter is enabled along with the DMA channel to start the transfer.

6. Synchronous Serial Controller (SSC)

The Synchronous Serial Controller (SSC) is a synchronous communication link that supports several serial protocols generally used in audio and telecom applications like I2S, Short Frame Sync, Long Frame Sync.

SSC contains an independent transmitter and a receiver with a common clock divider. The figure below shows the block diagram of the SSC module.

Figure 6-1. SSC Block Diagram



The transmitter and the receiver consist of three pins each. One data pin (TD for transmitter and RD for receiver), one clock pin (TK for transmitter and RK for receiver) and one frame sync pin (TF for transmitter and RF for receiver).

6.1. SSC Clock Divider

The SSC has an internal clock divider which takes the peripheral clock (MCK from PMC) as its input and divides the clock with a fixed divide by 2 divider followed by a configurable divider with a maximum division factor of 4095 (12-bit divider).

The overall division factor of the clock divider is 8190. Each level of the divided clock has a duration of the peripheral clock period multiplied by DIV. This ensures a 50% duty cycle for the divided clock regardless of whether the DIV value is even or odd.

The divided clock can be input to both the transmitter and the receiver blocks.

Note: There is a limitation for the bit clock frequency when SSC is operated in Slave mode. When SSC is operated in Slave mode, MCK frequency > (6 * bit clock frequency). For an MCK frequency of 166 MHz (maximum for SAMA5D2), the maximum bit clock frequency can be ~27.666 MHz.

6.2. SSC Transmitter

The SSC transmitter block is designed with three main parts, namely the transmit clock controller, the start selector and the transmit shift register.

The transmitter block is fed with three different clock sources as listed below, one of which will be selected by the transmit clock controller as the transmitter clock:

1. Clock from the TK pin (TK pin is input).
2. Clock from the SSC clock divider.
3. Clock from the receiver block.

The start selector controls when a transmit frame has to be started in the transmitter block. The frame start event is configurable and can be one of the events below.

1. Start frame with transmitter enabled (TXEN). This is a Continuous mode operation where the transmission starts as soon as the data is written to the transmitter data register (SSC_THR), provided TXEN is set.
2. Start frame on event trigger from the receiver's start selector (RX Start).
3. Start frame on TF event. The TF event can be low level / high level / any level change / rising edge / falling edge / any edge in the TF pin.

The transmit shift register is used to transmit the serial data. An SSC frame can have sync bits sent before the actual data is sent on the TD pin. Two registers are used for this purpose, namely SSC_THR (to hold the actual data) and SSC_TSHR (to hold the sync bits). If configured and enabled, the sync bits are sent first, followed by the actual data. The sync bits can be either a fixed logic level for a given number of bit cycles, or a bit pattern written in SSC_TSHR.

6.3. SSC Receiver

The SSC receiver block is designed with three main parts, namely the receive clock controller, the start selector and the receive shift register.

The receiver block is fed with three different clock sources as listed below, one of which will be selected by the receive clock controller as the receiver clock.

1. Clock from the RK pin (RK pin is input).
2. Clock from the SSC clock divider.
3. Clock from the transmitter block.

The start selector controls when a receive frame has to be started in the receiver block. The frame start event is configurable and can be one of the events below.

1. Start frame with receiver enabled (RXEN). This is a Continuous mode operation where data is received immediately after the end of the transfer of the previous data, provided RXEN is set.
2. Start frame on event trigger from the transmitter's start selector (TX Start).
3. Start frame on RF event. The RF event can be low level / high level / any level change / rising edge / falling edge / any edge.

4. Start frame on compare match of incoming bit pattern with bit pattern in SSC_RC0R register.

The receive shift register is used to receive the serial data. An SSC frame can have sync bits received before the actual data is received on the RD pin. Two registers are used for this purpose, namely SSC_RHR (to hold the actual data) and SSC_RSHR (to hold the sync bits). If configured and enabled, the sync bits are received first, followed by the actual data. The sync bits can be either a fixed logic level for a given number of bit cycles or can be a bit pattern.

6.4. SSC Frame Format

The data framing format of both the transmitter and the receiver are programmable through the Transmitter Frame Mode Register (SSC_TFMR) and the Receiver Frame Mode Register (SSC_RFMR). In either case, the user can independently select the following parameters:

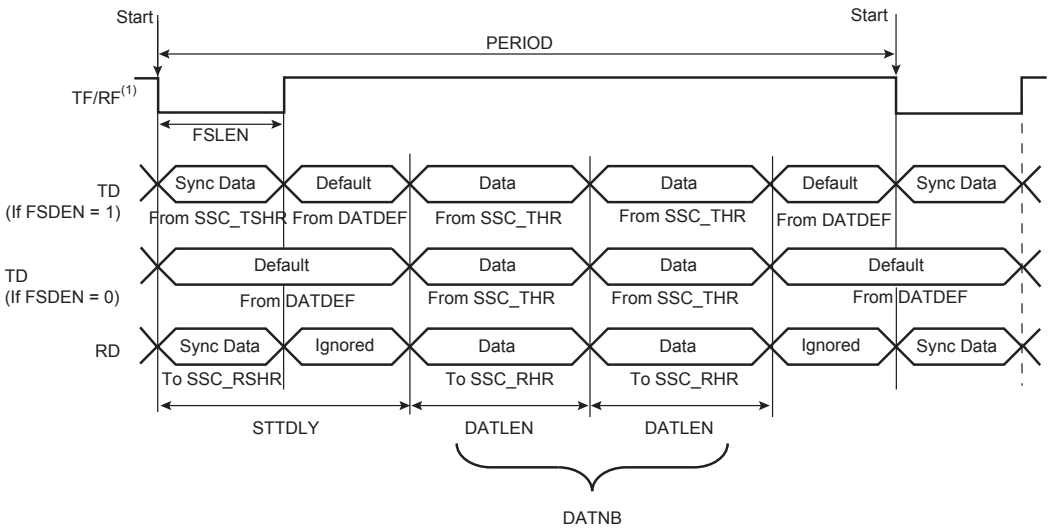
- Event that starts the data transfer (START)
- Delay, in number of bit periods, between the start event and the first data bit (STTDLY)
- Length of the data (DATLEN)
- Number of data to be transferred for each start event (DATNB)
- Length of synchronization transferred for each start event (FSLEN)
- Bit sense: most significant bit first (MSBF)
  MSBF = 1 : MSB sent first

  MSBF = 0 : LSB sent first

Additionally, the transmitter can be used to transfer synchronization and select the level driven on the TD pin while not in data transfer operation. This is done respectively by the Frame Sync Data Enable (FSDEN) and by the Data Default Value (DATDEF) bits in SSC_TFMR.

The figure below shows the frame format of an SSC frame.

Figure 6-2. SSC Frame Format



Note: 1. Example of input on falling edge of TF/RF.

As SSC frame start is triggered by the start event generated by the start selector and starts transmitting/receiving the sync data on the TD/RD pins. The sync pulse and the sync data are available on the bus for FSLEN bit cycles. The sync data will be the configured value of SSC_TSHR if FSDEN = 1 or will be the configured default level (value of DATDEF bit) if FSDEN = 0.

In transmitter with FSDEN = 1, once the FSLEN bit cycles are passed, the configured default logic level is sent on the TD line for (STTDLY – FSLEN) bit cycles if STTDLY > FSLEN. If STTDLY < FSLEN, the remaining sync data will not be sent but the actual data will be sent, so such configuration should be avoided.

In receiver with FSDEN = 1, once the FSLEN bit cycles are passed, the RD line will not be sampled for (STTDLY – FSLEN) bit cycles if STTDLY > FSLEN. If STTDLY < FSLEN, the remaining sync data will be sampled as actual data, so such configuration should be avoided.

Once the STTDLY bit cycles are passed, the actual data transmission/reception starts. The data length for each word can be defined in the DATLEN field and the number of words in each frame can be defined in the DATNB field. The sync pulse can be generated periodically every (2 * (PERIOD + 1)) bit cycles.

If the frame size PERIOD > (STTDLY + (DATNB * DATLEN)), then for the remaining bit cycles:

- In the transmitter, the default logic level defined by DATDEF will be sent on the TD pin.
- In receiver, the RD bits will be ignored.

The table below shows the bit fields and registers used to construct an SSC frame with their maximum lengths.

Table 6-1. SSC Transfer Parameters with Respective Bit Lengths

Transmitter Register	Receiver Register	Field	Length (no. of bits)	Comment
SSC_TFMR	SSC_RFMR	DATLEN	Up to 32	Size of a word
		DATNB	Up to 16	Number of words in a frame
		MSBF	-	Most Significant Bit First
		FSLEN	Up to 256	Size of sync data register
	-	DATDEF	0 or 1	Data default value ended
	-	FSDEN	-	Enable send SSC_TSHR
SSC_TCMR	SSC_RCMR	PERIOD	Up to 512	Frame size
		STTDLY	Up to 255	Size of transmit start delay

Note that from the above table, up to 256 bits can be allotted for FSLEN. However, the FSLEN is just 4 bits in SSC_TFMR / SSC_RFMR. The 4-bit FSLEN is used along with a 4-bit field FSLEN_EXT in the same register so that the total FSLEN slot can accommodate 256 bits. The pulse length is calculated as:

FSLEN + (FSLEN_EXT * 16) + 1 clock cycle

When FSDEN is set to 1, the frame synchronization data from SSC_TSHR is sent for FSLEN cycles. However, SSC_TSHR is only 32 bits in length, so a synchronization data of more than 32 bits is sent in the following order:

1. SSC_TSHR (32-bit)
2. DATDEF (32-bit)
3. SSC_THR (32-bit)
4. Repeat SSC_THR until 256 bits is reached.

6.5. Code Examples

The code example given in this section demonstrates the steps to play back a 48-kHz stereo 16-bit PCM audio using the SSC interface and an external audio DAC IC (AD1934 from Analog Devices).

The 12.288 MHz master clock is generated from the Audio PLL as in section Code Example - Audio PLL. The generated 12.288 MHz signal from the CLK_AUDIO pin is fed to the MCLKI pin of AD1934. The AD1934 is configured to generate the bit clock and the LR clock with a sample rate set to 48 kHz. The word width of AD1934 is always 32 bits and is configured to accept audio data in I2S frame format and left-justified with MSB first. Following are the register settings for the AD1934, which is configured using the SPI interface.

- PLL and Clock Control 0 --> 0x98
- DAC Control 1 --> 0x70
- DAC Control 2 --> 0x18

The other AD1934 registers are left to their default reset values.

The following sections provide the code snippet to configure the SSC I/O pins, the DMA channel and the SSC peripheral.

6.5.1. SSC I/O Pin Initialization

PIO pins PA14 (TK), PA15 (TF) and PA16 (TD) with their peripheral function B will be assigned to the SSC1's transmitter interface. The code snippet below will initialize the I/O pins accordingly.

```
/* Set PORTA mask register bit 14 & 15 */
PIOA->PIO_IO_GROUP[0].PIO_MSKR = (3u << 14);

/* Enable peripheral function B and set I/O pin as input */
PIOA->PIO_IO_GROUP[0].PIO_CFGR = PIO_CFGR_FUNC_PERIPH_B |
PIO_CFGR_DIR_INPUT;

/* Set PORTA mask register bits 16 */
PIOA->PIO_IO_GROUP[0].PIO_MSKR = (1u << 16);

/* Enable peripheral function B and set I/O pins as output */
PIOA->PIO_IO_GROUP[0].PIO_CFGR = PIO_CFGR_FUNC_PERIPH_B |
PIO_CFGR_DIR_OUTPUT;
```

6.5.2. SSC XDMAC Channel Initialization

A DMA channel from the XDMAC module is configured to transfer the audio data from an array in main memory to the Transmit Holding Register (THR) of SSC. The code snippet given below configures the XDMAC channel 0 in a single block – Single Micro Block mode with micro block length equal to total number of audio samples * 2 (for 2 channels).

Data transfer is done one channel sample (16-bit) at a time.

```
/* Enable peripheral clock for XDMAC0 */
PMC->PMC_PCER0 = (1u << ID_XDMAC0);

/* Read the interrupt status register to clear the interrupt flags */
temp = XDMAC0->XDMAC_CHID[0].XDMAC_CIS;

/* Set source address as starting address of audio buffer */
XDMAC0->XDMAC_CHID[0].XDMAC_CSA = (uint32_t)audio_samples;
```

```
/* Set destination address as SSC_THR register's 16 MSB bits */
XDMAC0->XDMAC_CHID[0].XDMAC_CDA = ((uint32_t)&SSC1->SSC_THR) + 2;

/* Set micro block length */
XDMAC0->XDMAC_CHID[0].XDMAC_CUBC = sizeof(audio_samples)/2;

/* Set DMA channel parameters */
XDMAC0->XDMAC_CHID[0].XDMAC_CC = XDMAC_CC_TYPE_PER_TRAN
                                    | XDMAC_CC_MBSIZE_SINGLE
                                    | XDMAC_CC_DSYNC_MEM2PER
                                    | XDMAC_CC_CSIZE_CHK_1
                                    | XDMAC_CC_DWIDTH_HALFWORD
                                    | XDMAC_CC_SIF_AHB_IF0
                                    | XDMAC_CC_DIF_AHB_IF1
                                    | XDMAC_CC_SAM_INCREMENTED_AM
                                    | XDMAC_CC_DAM_FIXED_AM
                                    | XDMAC_CC_PERID(23);

/* Set all registers related to descriptor to 0 */
XDMAC0->XDMAC_CHID[0].XDMAC_CNDC = 0;
XDMAC0->XDMAC_CHID[0].XDMAC_CBC = 0;
XDMAC0->XDMAC_CHID[0].XDMAC_CDS_MSP = 0;
XDMAC0->XDMAC_CHID[0].XDMAC_CSUS = 0;
XDMAC0->XDMAC_CHID[0].XDMAC_CDUS = 0;
```

The DMA channel source address is set to the start of the audio sample array and the destination address is set to the start of 16 MSB of the THR register. This is because we need to send the 16-bit audio sample first in the MSB part of the SSC 32-bit word. "audio_samples" is an array of type signed 16-bit integer. The transfer size is set to half word with source address incrementing and destination address fixed for each transfer.

No descriptors are used for the transfer, so the XDMAC registers related to the descriptor configuration are set to 0.

6.5.3. SSC Initialization

The following code snippet initializes the SSC interface.

```
/* Enable peripheral clock for SSC1 */
PMC->PMC_PCR = PMC_PCR_PID(ID_SSC1) | PMC_PCR_CMD | PMC_PCR_EN;

/* Perform software reset of SSC1 peripheral */
SSC1->SSC_CR = SSC_CR_SWRST;

/* Configure SSC1 parameters */
SSC1->SSC_TCMR = SSC_TCMR_CKS_TK
                    | SSC_TCMR_STTDLY(1)
                    | SSC_TCMR_START_TF_FALLING;

SSC1->SSC_TFMR = SSC_TFMR_DATLEN(31)
                    | SSC_TFMR_MSBF
                    | SSC_TFMR_DATNB(1);

/* Enable SSC1 transmitter */
SSC1->SSC_CR = SSC_CR_TXEN;

/* Enable DMA channel */
XDMAC0->XDMAC_GE = XDMAC_GE_EN0;

/* Wait until DMA transfer is done */
while(!(XDMAC0->XDMAC_CHID[0].XDMAC_CIS & XDMAC_CIS_BIS));
```

SSC1 is used for this example. The TK and TF pins are configured as inputs which accept the bit clock and the LR clock respectively from the AD1934 module. Data is sent as 32-bit word with the original 16-bit audio data placed in the 16 MSB bits and the 16 LSB bits left to 0.

SSC is configured to have a frame of two words and to start a frame on the falling edge of the TF pin but one bit clock delayed (STTDLY = 1) to comply with the I2S frame format.

7. Differences Between I2SC and SSC

Though SSC and I2SC are synchronous serial links commonly used for digital audio communication, there are differences between the two peripherals based on their implementation and configurability. Users can choose SSC or I2SC for the application based on peripheral's capability, the connected device's requirements and the I/O pin availability.

1. I2SC only supports the I2S protocol, whereas SSC is a highly configurable peripheral which supports the I2S protocol among others. For example, the data transfer delay is fixed to 1 bit clock cycle in I2SC, whereas in SSC it is configurable. This helps to interface SAMA5D2 with different types of codec devices that use different frame formats than the standard I2S frame format.
2. The I2SC in SAMA5D2 can be used only with a single slave or a single master. It does not support the TDM feature which is used to interface with multiple I2S devices. SSC can be configured to interface with multiple I2S devices in a single bus.
3. The I2SC has an internal clock divider for master clock output using a dedicated master clock pin (I2SCMCK), and can be clocked with Audio PLL output using the GCLK interface. But SSC cannot be clocked with the GCLK interface; it is only clocked by the system clock (MCK). To clock the SSC module with Audio PLL clock, the system clock should also be clocked with Audio PLL. This drawback is taken care of in the SAMA5D2 product, where a dedicated pin is given to Audio PLL (CLK_AUDIO) to output the master clock frequency required for the audio devices connected to SSC. The SSC can sample the data based on TK/RK pins (timed based on CLK_AUDIO) while still clocked by the MCK clock.
4. SSC may be prone to channel swapping issues in cases where data transfer to Transmit Holding Register (THR) through DMA is triggered by the falling/rising edge of the TF/RF signal due to delay in DMA transfer. This issue may not be faced in I2SC since the data transfer through DMA is always triggered by TXRDY/RXRDY signals.

8. Revision History

Table 8-1. Revision History

Doc. Rev.	Date	Changes
A	13-July-16	First release

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**