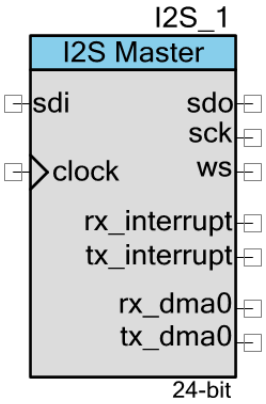
# Inter-IC Sound Bus (I2S)

2.70




## Features

- Master only
- Single- and multi-channel (up to 10 channels) I2S support
- 8 to 32 data bits per sample
- 16-, 32-, 48-, or 64-bit word select period
- Data rate up to 96 kHz with 64-bit word select period
- Audio clip detection in I2S Rx mode
- Byte swap audio samples to match USB audio class endianness requirement

## General Description

The Integrated Inter-IC Sound Bus (I2S) is a serial bus interface standard used for connecting digital audio devices together. The specification is from Philips® Semiconductor (I2S bus specification; February 1986, revised June 5, 1996).

The I2S component operates in master mode only. It also operates in two directions, as a transmitter (Tx) and a receiver (Rx). The data for Tx and Rx are independent byte streams. The byte streams are packed with the most significant byte first and the most significant bit in bit 7 of the first word. The number of bytes used for each sample (a sample for the left or right channel) is the minimum number of bytes to hold a sample.

### When to Use an I2S

The component provides a serial bus interface for stereo audio data. This interface is most commonly used by audio ADCs and DACs. This component supports I2S audio data format with configurable data bits and word select parameters. Left-Justified and Right-Justified audio data formats are not supported.

# Input/Output Connections

This section describes the various input and output connections for the I2S component. An asterisk (*) in the list of I/Os indicates that the I/O may be hidden on the symbol under the conditions listed in the description of that I/O.

## sdi – Input *

This is the serial data input. It displays if an Rx option is selected for the Direction parameter. The signal width depends on the Number of channels parameter selected for Rx direction.

If this signal is connected to an input pin, that Pins component **Sync Mode** parameter should be set to Transparent. This signal should already be synchronized to SCK, so delaying the signal with the input pin synchronizer could cause the signal to be shifted into the next clock cycle.

## clock – Input

The clock rate provided must be two times the desired clock rate for the output serial clock (SCK). For example, to produce 48-kHz audio with a 64-bit word select period the clock frequency is:

2 × 48 kHz × 64 = 6.144 MHz

Refer to Enabling

The Rx and Tx directions have separate enables. When not enabled, the Tx direction transmits all 0 values, and the Rx direction ignores all received data. The transition into and out of the enabled state occurs at a word select boundary such that a left/right sample pair is always transmitted or received.

Clock Selection section for a detailed description on clock rate calculations.

## sdo – Output *

This is the serial data output. It displays if a Tx option is selected for the Direction parameter. The signal width depends on the Number of channels parameter selected for Tx direction.

## sck – Output

This is the output serial clock.

## ws – Output

The word select output indicates the channel being transmitted.

## rx_interrupt – Output *

This is the Rx direction interrupt. It displays if an **Rx** or **Rx and Tx** option is selected for the Direction parameter.

### tx_interrupt – Output *

This is the Tx direction interrupt. It displays if a Tx or **Rx and Tx** option is selected for the Direction parameter.

### rx_dma0 – Output *

This is the Rx direction DMA request for FIFO 0 (Left or Interleaved). It displays if the DMA request is enabled for Rx direction. This output can be used as the Rx FIFO 0 status signal for devices without a DMA controller. The signal width depends on the Number of channels parameter selected for Rx direction.

### rx_dma1 – Output *

This is the Rx direction DMA request for FIFO 1 (Right). It displays if the DMA request is enabled and **Separated L/R** is selected under the Data interleaving parameter for Rx. This output can be used as the Rx FIFO 1 status signal for devices without the DMA controller. The signal width depends on the Number of channels parameter selected for Rx direction.

### tx_dma0– Output *

This is the Tx direction DMA request for FIFO 0 (Left or Interleaved). It displays if the DMA request is enabled for Tx direction. This output can be used as the Tx FIFO 0 status signal for devices without a DMA controller. The signal width depends on the Number of channels parameter selected for Tx direction.

### tx_dma1 – Output *

This is the Tx direction DMA request for FIFO 1 (Right). It displays if the DMA request is enabled and **Separated L/R** is selected under the Data interleaving parameter for Tx. This output can be used as the Tx FIFO 1 status signal for devices without a DMA controller. The signal width depends on the Number of channels parameter selected for Tx direction.

### clip – Output *

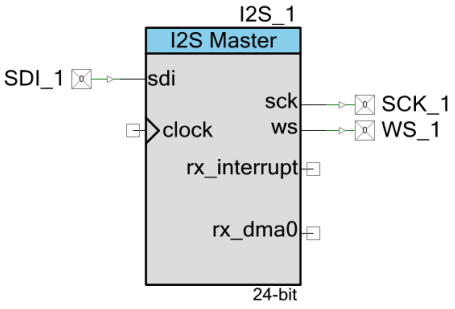
This output indicates that a clipping occurs on an input audio signal. It is high when the input signal is beyond the user configured clipping threshold. The signal width depends on the Number of channels parameter selected for Rx direction. It displays if an **Rx** or **Rx and Tx** option is selected for the Direction parameter and the Rx clip detection parameter is selected.

## Schematic Macro Information

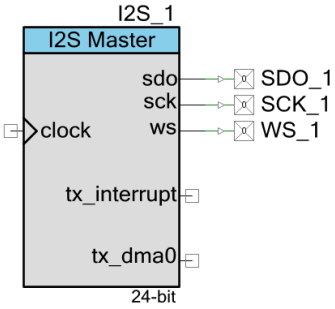
By default, the PSoC Creator Component Catalog contains three Schematic Macro implementations for the I2S component. These macros contain the I2S component already connected to digital pin components. The **Sync Mode** parameter for the SDI pin is set to

Transparent, and the generation of APIs for all of the pins is turned off. The Schematic Macros use the I2S component configured for Rx only, Tx only, and both Rx and Tx directions, as shown in the following diagrams.

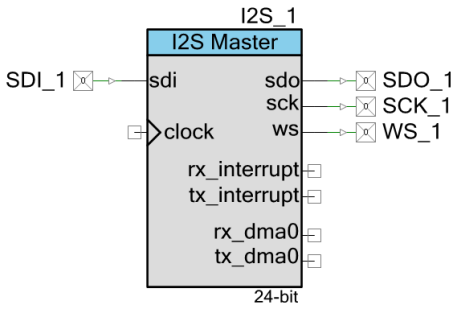
**I2S Rx Only**




**I2S Tx Only**



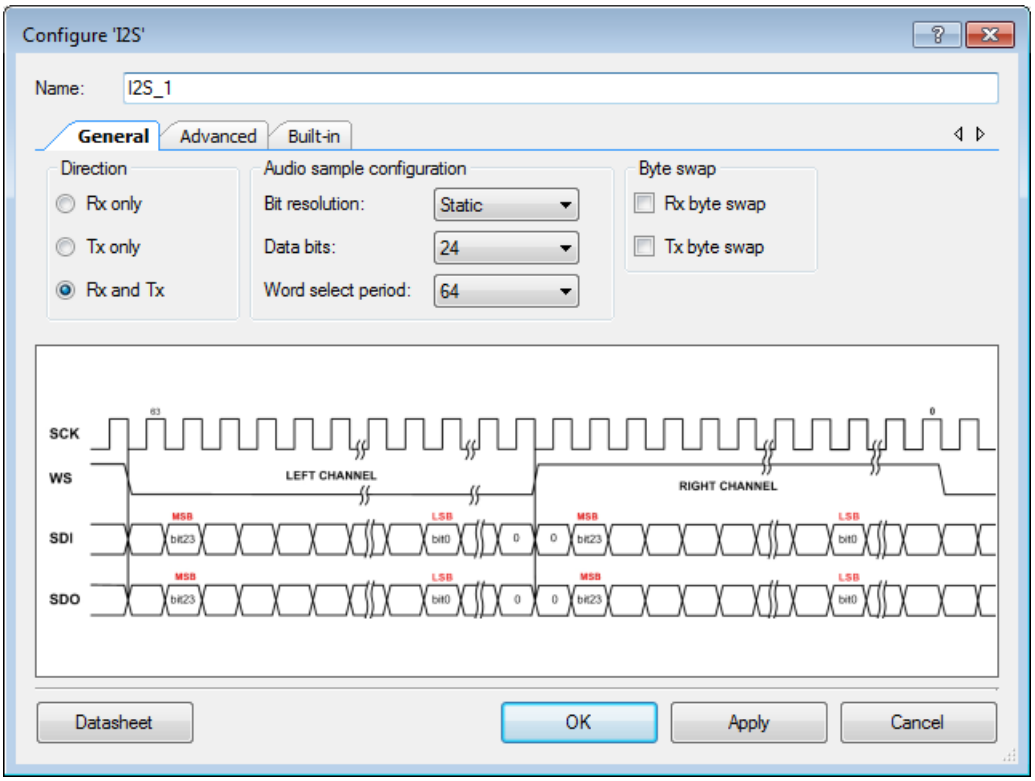

**I2S Rx and Tx**




# Component Parameters

Drag an I2S component onto your design and double-click it to open the **Configure** dialog. This dialog has the following tabs with different parameters.

## Basic Tab



### Direction

This parameter determines which direction the component operates. This value can be set to **Rx Only**, **Tx Only**, or **Rx and Tx** (default).

### Bit resolution

This parameter determines if the Tx and Rx sample bit resolution is static or dynamically configurable in firmware. This value can be set to **Static** (default) or **Dynamic**.

### Data bits

This parameter determines the number of data bits configured for each sample. If the **Static** Bit resolution is selected, this value can be set between 8 and 32. If the **Dynamic** Bit resolution is selected, the data bits can only be set to 8, 16, 24 and 32. The default setting is **24**.

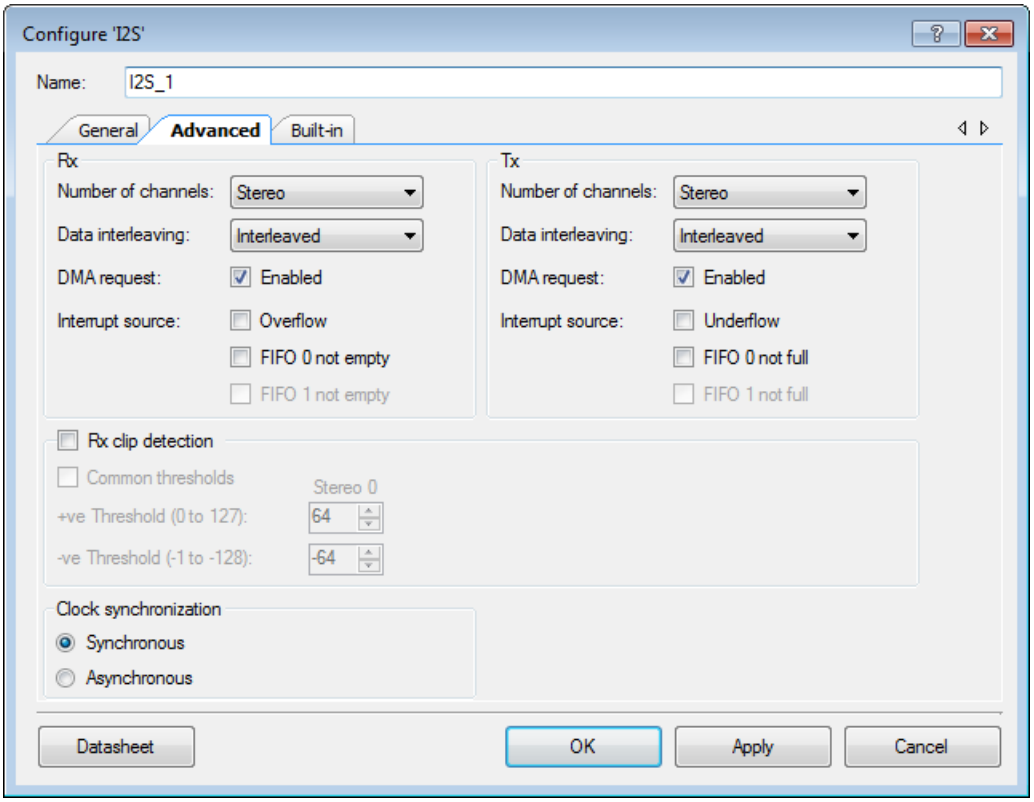
### Word select period

This parameter determines the period of a complete sample of both left and right channels. This value can be set to **16**, **32**, **48**, or **64** (default).

### Byte swapping

This parameter determines if the Tx and Rx data endianness conversion will be performed. Refer to Byte Swapping section for a detailed description of the byte swap operation. This option can be enabled only when the Word select period parameter is set to 64, the Data bits parameter value is between 9 and 24 bits, and the Data interleaving parameter is set to Interleaved for the associated direction. The Byte swapping parameter is disabled by default.

## Advanced Tab




### Number of channels

This parameter determines whether the I2S data is for mono, stereo, or multi-stereo channels. If mono, the left or right data channel can be additionally selected. The number of channels can be individually set for the Rx and Tx directions. Supported options are **Mono left**, **Mono right**, **Stereo** (Default) or **4/6/8/10** channels.

### Data interleaving

This parameter is used to select whether the data is **Interleaved** (default) or **Separate L/R**. Rx and Tx are selected independently. When the **Interleaved** option is selected, the left and right channels are interleaved with a sample for the left channel first, followed by the right channel in a single FIFO (FIFO 0). In **Separate L/R** is selected, the left and right channel byte streams use separate FIFOs: FIFO 0 and FIFO 1. Each FIFO has 4-byte depths.

### DMA request

This parameter is used to enable and disable the DMA request signals for the component. Rx and Tx are selected independently. These options are **enabled** by default.

### Interrupt source

This parameter is used to configure the interrupt sources for Rx and Tx direction. Rx and Tx directions have separate interrupt outputs. Multiple sources for each direction are ORed together. The software can re-configure these sources at any time; these parameters simply define an initial configuration. Settings include:

- Rx:
  - **Rx Overflow**
  - **Rx FIFO 0 not empty** (Left or Interleaved)
  - **Rx FIFO 1 not empty** (Right) - Only an option if not Interleaved
- Tx:
  - **Tx Underflow**
  - **Tx FIFO 0 not full** (Left or Interleaved)
  - **Tx FIFO 1 not full** (Right) - Only an option if not Interleaved

### Rx clip detection

If selected, this parameter enables audio sample clip detection (MSB threshold comparison) on the I2S Rx data. Positive and negative clipping thresholds are user configurable from the Configure dialog settings or can be changed dynamically in the firmware using the SetPositiveClipThreshold() or SetNegativeClipThreshold() API. This option is unselected by default. Refer to the Clip Detection section for more details on digital audio clipping.

### Common thresholds

The Common thresholds option, when selected, applies Stereo 0 thresholds to all the other stereo channels (if multi-channel configuration is enabled). The option is unselected by default.

### +ve Threshold

This parameter specifies 8-bit positive clipping threshold for incoming audio samples. The value can be set between 0 and 127. The default setting is 64.

### -ve Threshold

This parameter specifies 8-bit negative clipping threshold for incoming audio samples. The value can be set between -1 and -128. The default setting is -64.

### Clock synchronization

This parameter specifies the intended synchronization of the clock signal with respect to the system clock. Setting this to **Synchronous** forces the input clock to be synchronized to the system clock. Setting it to **Asynchronous** leaves the synchronization unchanged from the synchronization of

the input clock. This option is useful when the clock for the I2S component is supplied from an external source and the CPU itself is running off of its own clock.

# Application Programming Interface

Application Programming Interface (API) routines allow configuration of the component using the software. The following table lists and describes the interface for each function. The subsequent sections cover each function in more detail.

By default, PSoC Creator assigns the instance name “I2S_1” to the first instance of a component in a given design. The instance can be renamed to any unique value that follows the syntactic rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol. For readability, the instance name used in the following table is “I2S.”

## Functions

| Function | Description |
|---|---|
| I2S_Start() | Starts the I2S interface. |
| I2S_Stop() | Disables the I2S interface. |
| I2S_Init() | Initializes or restores default I2S configuration. |
| I2S_Enable() | Enables the I2S interface. |
| I2S_SetDataBits() | Sets the number of data bits for each sample. |
| I2S_EnableTx() | Enables the Tx direction of the I2S interface. |
| I2S_DisableTx() | Disables the Tx direction of the I2S interface. |
| I2S_SetTxInterruptMode() | Sets the interrupt source for the I2S Tx direction interrupt. |
| I2S_ReadTxStatus() | Returns state in the I2S Tx status register. |
| I2S_WriteByte() | Writes a single byte into the Tx FIFO. |
| I2S_ClearTxFIFO() | Clears out the Tx FIFO. |
| I2S_EnableRx() | Enables the Rx direction of the I2S interface. |
| I2S_DisableRx() | Disables the Rx direction of the I2S interface. |
| I2S_SetRxInterruptMode() | Sets the interrupt source for the I2S Rx direction interrupt. |
| I2S_ReadRxStatus() | Returns state in the I2S Rx status register. |
| I2S_ReadByte() | Returns a single byte from the Rx FIFO. |
| I2S_ClearRxFIFO() | Clears out the Rx FIFO. |
| I2S_SetPositiveClipThreshold() | Sets the positive clip detection threshold. |
| I2S_SetNegativeClipThreshold() | Sets the negative clip detection threshold. |

| Function | Description |
|---|---|
| I2S_Sleep() | Saves configuration and disables the I2S interface. |
| I2S_Wakeup() | Restores configuration and enables the I2S interface. |

## void I2S_Start(void)

**Description:** This function starts the I2S interface. It starts the generation of the sck and ws outputs. The Tx and Rx directions remain disabled.

**Parameters:** None

**Return Value:** None

**Side Effects:** None

## void I2S_Stop(void)

**Description:** This function disables the I2S interface. The sck and ws outputs are set to 0. The Tx and Rx directions are disabled and their FIFOs are cleared.

**Parameters:** None

**Return Value:** None

**Side Effects:** None

## void I2S_Init(void)

**Description:** This function initializes or restores default I2S configuration provided with customizer. It does not clear data from the FIFOs and does not reset component hardware state machines. It is not necessary to call I2S_Init() because the I2S_Start() routine calls this function, and I2S_Start() is the preferred method to begin component operation.

**Parameters:** None

**Return Value:** None

**Side Effects:** None

## void I2S_Enable(void)

**Description:** This function enables the I2S interface. It starts the generation of the sck and ws outputs. The Tx and Rx directions remain disabled. It is not necessary to call I2S_Enable() because the I2S_Start() routine calls this function, and I2S_Start() is the preferred method to begin component operation.

**Parameters:** None

**Return Value:** None

**Side Effects:** None

## cystatus I2S_SetDataBits(uint8 dataBits)

**Description:** This function sets the number of data bits for each sample. The component must be stopped before calling this API. The API is only available when the Bit resolution parameter is set to Dynamic.

**Parameters:** dataBits: the number of data bits for each sample. Valid values - 8/16/24/32.

**Return Value:**

| Value | Description |
|---|---|
| CYRET_SUCCESS | Succesful |
| CYRET_BAD_PARAM | Invalid parameter value |

**Side Effects:** Calling this API while the component is operating may cause unexpected behavior.

## void I2S_EnableTx(void)

**Description:** This function enables the Tx direction of the I2S interface. Transmission begins at the next word select falling edge.

**Parameters:** None

**Return Value:** None

**Side Effects:** None

## void I2S_DisableTx(void)

**Description:** This function disables the Tx direction of the I2S interface. Transmission of data stops and a constant 0 value is transmitted at the next word select falling edge.

**Parameters:** None

**Return Value:** None

**Side Effects:** None

### void I2S_SetTxInterruptMode(interruptSource) / void I2S_SetTxInterruptMode(channel, interruptSource)

**Description:** This macro sets the interrupt source for the specified Tx stereo channel. Multiple sources may be ORed.

**Note** The macro expects channel parameter only when more than one stereo channel is selected for Tx direction.

**Parameters:** channel (Tx multi-channel): Constant to specify the stereo channel number.

Format: CHn, where n = 0..4. For example CH0 refers to Stereo channel 0, CH1 to stereo channel 1, etc.

Byte containing the constant for the selected interrupt sources (uint8).

| I2S Tx Interrupt Source | Value | Type |
|---|---|---|
| TX_FIFO_UNDERFLOW | 0x01 | Clear on Read |
| TX_FIFO_0_NOT_FULL | 0x02 | Transparent |
| TX_FIFO_1_NOT_FULL | 0x04 | Transparent |

**Return Value:** None

**Side Effects:** If clear on read bits are used as the sources for the interrupt generation, the tx_interrupt output remains asserted until the status register for the corresponding stereo channel is read.

### uint8 I2S_ReadTxStatus(void) / uint8 I2S_ReadTxStatus(channel)

**Description:** This macro returns the status of the specified stereo channel(s). In a multi-channel configuration, the status bits of stereo channel 0 are combined with stereo channel 1, and the bits of channel 2 are combined with channel 3. Therefore the API will return the combined status of stereo channel 0 and stereo channel 1 if the status for channel 0 or channel 1 is requested.

**Note** The macro expects channel parameter only when more than one stereo channel is selected for Tx direction.

**Parameters:** channel (Tx multi-channel): Constant to specify the stereo channel number.

Format: CHn, where n = 0..4. For example CH0 refers to Stereo channel 0, CH1 to stereo channel 1, etc.

**Return Value:** Status bits associated with the specified stereo channel (uint8). Refer to the corresponding I2S_TX_STATUS_REG register for a detailed description.

**Side Effects:** Clears the status register bits associated with the specified channel that are clear on read.

### void I2S_WriteByte(wrData, wordSelect) / void I2S_WriteByte(channel, wrData, wordSelect)

**Description:** This macro writes a single byte into the Tx FIFO for the specified stereo channel. You have to check the Tx status before this call to confirm that the Tx FIFO is not full.

**Note** The macro expects channel parameter only when more than one stereo channel is selected for Tx direction.

**Parameters:** channel (Tx multi-channel): Constant to specify the stereo channel number.

Format: CHn, where n = 0..4. For example CH0 refers to Stereo channel 0, CH1 to stereo channel 1, etc.

wrData: Byte containing the data to transmit (uint8).

wordSelect: Indicates to write to the Left (0) or Right (1) channel (uint8). In interleaved mode, this parameter is ignored.

**Return Value:** None

**Side Effects:** None

### void I2S_ClearTxFIFO(void)

**Description:** This function clears out the FIFOs for all Tx channels. Any data present in the FIFOs is lost. Call this function only when the Tx direction is disabled.

**Parameters:** None

**Return Value:** None

**Side Effects:** None

### void I2S_EnableRx(void)

**Description:** This function enables the Rx direction of the I2S interface. Data reception begins at the next word select falling edge.

**Parameters:** None

**Return Value:** None

**Side Effects:** None

### void I2S_DisableRx(void)

**Description:** This function disables the Rx direction of the I2S interface. At the next word select falling edge, data received is no longer sent to the receive FIFO.

**Parameters:** None

**Return Value:** None

**Side Effects:** None

### void I2S_SetRxInterruptMode(interruptSource) / void I2S_SetRxInterruptMode(channel, interruptSource)

**Description:** This macro sets the interrupt source for the specified Rx stereo channel. Multiple sources may be ORed.

**Note** The macro expects channel parameter only when more than one stereo channel is selected for Rx direction.

**Parameters:** channel (Rx multi-channel): Constant to specify the stereo channel number.

Format: CHn, where n = 0..4. For example CH0 refers to Stereo channel 0, CH1 to stereo channel 1, etc.

Byte containing the constant for the selected interrupt sources (uint8).

| I2S Tx Interrupt Source | Value | Type |
|---|---|---|
| RX_FIFO_OVERFLOW | 0x01 | Clear on Read |
| RX_FIFO_0_NOT_EMPTY | 0x02 | Transparent |
| RX_FIFO_1_NOT_EMPTY | 0x04 | Transparent |

**Return Value:** None

**Side Effects:** If clear on read bits are used as the sources for the interrupt generation, the rx_interrupt output remains asserted until the status register for the corresponding stereo channel is read.

### uint8 I2S_ReadRxStatus(void) / uint8 I2S_ReadRxStatus(channel)

**Description:** This macro returns the status of the specified stereo channel(s). In a multi-channel configuration, the status bits of stereo channel 0 are combined with stereo channel 1, and the bits of channel 2 are combined with channel 3. Therefore the API will return the combined status of stereo channel 0 and stereo channel 1 if the status for channel 0 or channel 1 is requested.

**Note** The macro expects channel parameter only when more than one stereo channel is selected for Rx direction.

**Parameters:** channel (Rx multi-channel): Constant to specify the stereo channel number.

Format: CHn, where n = 0..4. For example CH0 refers to Stereo channel 0, CH1 to stereo channel 1, etc.

**Return Value:** Status register bits associated with the specified stereo channel (uint8). Refer to the corresponding I2S_RX_STATUS_REG register for a detailed description.

**Side Effects:** Clears the status register bits associated with the specified channel that are clear on read.

### uint8 I2S_ReadByte(wordSelect) / uint8 I2S_ReadByte(channel, wordSelect)

**Description:** This macro returns a single byte from the Rx FIFO. You have to check the Rx status before this call to confirm that the Rx FIFO is not empty.

**Note** The macro expects channel parameter only when more than one stereo channel is selected for Rx direction.

**Parameters:** channel (Rx multi-channel): Constant to specify the stereo channel number.

Format: CHn, where n = 0..4. For example CH0 refers to Stereo channel 0, CH1 to stereo channel 1, etc.

wordSelect: Indicates to read from the Left (0) or Right (1) channel (uint8). In interleaved mode, this parameter is ignored.

**Return Value:** Byte containing the data received (uint8)

**Side Effects:** None

### void I2S_ClearRxFIFO(void)

**Description:** This function clears out the FIFOs for all Rx channels. Any data present in the FIFOs is lost. Call this function only when the Rx direction is disabled.

**Parameters:** None

**Return Value:** None

**Side Effects:** None

### void I2S_SetPositiveClipThreshold(posThreshold) / void I2S_SetPositiveClipThreshold(channel, posThreshold)

**Description:** This macro sets the 8-bit positive clip detection threshold for the specified channel. This API is available if the Rx clip detection parameter is selected in the Configure dialog.

**Note** The macro expects channel parameter only when more than one stereo channel is selected for Rx direction.

**Parameters:** channel (Rx multi-channel): Constant to specify the stereo channel number.

Format: CHn, where n = 0..4. For example CH0 refers to Stereo channel 0, CH1 to stereo channel 1, etc.

posThreshold: Specifies the positive clip detection threshold (uint8).

Valid range: 0..127

**Return Value:** None

**Side Effects:** None

### void I2S_SetNegativeClipThreshold(negThreshold) / void I2S_SetNegativeClipThreshold(channel, negThreshold)

**Description:** This API sets the 8-bit negative clip detection threshold for the specified channel. This API is available if the Rx clip detection parameter is selected in the Configure dialog.

**Note** The macro expects channel parameter only when more than one stereo channel is selected for Rx direction.

**Parameters:** channel (Rx multi-channel): Constant to specify the stereo channel number.

Format: CHn, where n = 0..4. For example CH0 refers to Stereo channel 0, CH1 to stereo channel 1, etc.

negThreshold: Specifies the negative clip detection threshold (int8).

Valid range: -1..-128

**Return Value:** None

**Side Effects:** None

### void I2S_Sleep(void)

**Description:** This is the preferred routine to prepare the component for sleep. The I2S_Sleep() routine saves the current component state and calls the I2S_Stop() function. The sck and ws outputs are set to 0. The Tx and Rx directions are disabled.

Call the I2S_Sleep() function before calling the CyPmSleep() or the CyPmHibernate() function. Refer to the PSoC Creator *System Reference Guide* for more information about power management functions.

**Parameters:** None

**Return Value:** None

**Side Effects:** None

### void I2S_Wakeup(void)

**Description:** This is the preferred routine to prepare the component operation after exit from sleep. Starts the generation of the sck and ws outputs if the component operated before sleep. Enables Rx and/or Tx direction according to their states before sleep.

**Parameters:** None

**Return Value:** None

**Side Effects:** Calling the I2S_Wakeup() function without first calling the I2S_Sleep() function may produce unexpected behavior.

## Global Variables

| Variable | Description |
|---|---|
| I2S_initVar | Indicates whether the I2S has been initialized. The variable is initialized to 0 and set to 1 the first time I2S_Start() is called. This allows the component to restart without reinitialization after the first call to the I2S_Start() routine.<br>If it is necessary to reinitialize the component, then the I2S_Init() function can be called before the I2S_Start() or I2S_Enable() function. |

## Sample Firmware Source Code

PSoC Creator provides numerous example projects that include schematics and example code in the Find Example Project dialog. For component-specific examples, open the dialog from the Component Catalog or an instance of the component in a schematic. For general examples, open the dialog from the Start Page or **File** menu. As needed, use the **Filter Options** in the dialog to narrow the list of projects available to select.

Refer to the “Find Example Project” topic in the PSoC Creator Help for more information.

## MISRA Compliance

This section describes the MISRA-C:2004 compliance and deviations for the component. There are two types of deviations defined:

- project deviations – deviations that are applicable for all PSoC Creator components
- specific deviations – deviations that are applicable only for this component

This section provides information on component-specific deviations. Project deviations are described in the MISRA Compliance section of the *System Reference Guide* along with information on the MISRA compliance verification environment.

The I2S component has the following specific deviations:

| MISRA-C:2004 Rule | Rule Class (Required/ Advisory) | Rule Description | Description of Deviation(s) |
|---|---|---|---|
| 19.7 | A | A function should be used in preference to a function-like macro. | Deviated for more efficient code. |
| 19.12 | R | There shall be at most one occurrence of the # or ## preprocessor operators in a single macro definition. | Deviated to simplify the code and provide more compact representation of similar functionalities. |
| 19.13 | A | The # and ## preprocessor operators should not be used. | |

## API Memory Usage

The component memory usage varies significantly depending on the compiler, device, number of APIs used and component configuration. The following table provides the memory usage for all APIs available in the given component configuration.

The measurements have been done with an associated compiler configured in Release mode with optimization set for Size. For a specific design, the map file generated by the compiler can be analyzed to determine the memory usage.

| Configuration [1] | PSoC 3 (Keil_PK51) | | PSoC 4 / PSoC 5LP (GCC) | |
|---|---|---|---|---|
| | Flash Bytes | SRAM Bytes | Flash Bytes | SRAM Bytes |
| Rx Only | 200 + 32 × N | 2 | 302 + 46 × N | 2 |
| Tx Only | 200 + 32 × N | 2 | 302 + 46 × N | 2 |
| Rx and Tx | 286 + 24 × N | 2 | 458 + 45 × N | 2 |

# Functional Description

## Left/Right and Rx/Tx Configuration

The configuration for the Left and Right channels – the Rx and Tx direction, number of bits, and word-select period – are identical. If the application must have different configurations for the Rx and Tx, then use two unidirectional component instances.

## Data Stream Format

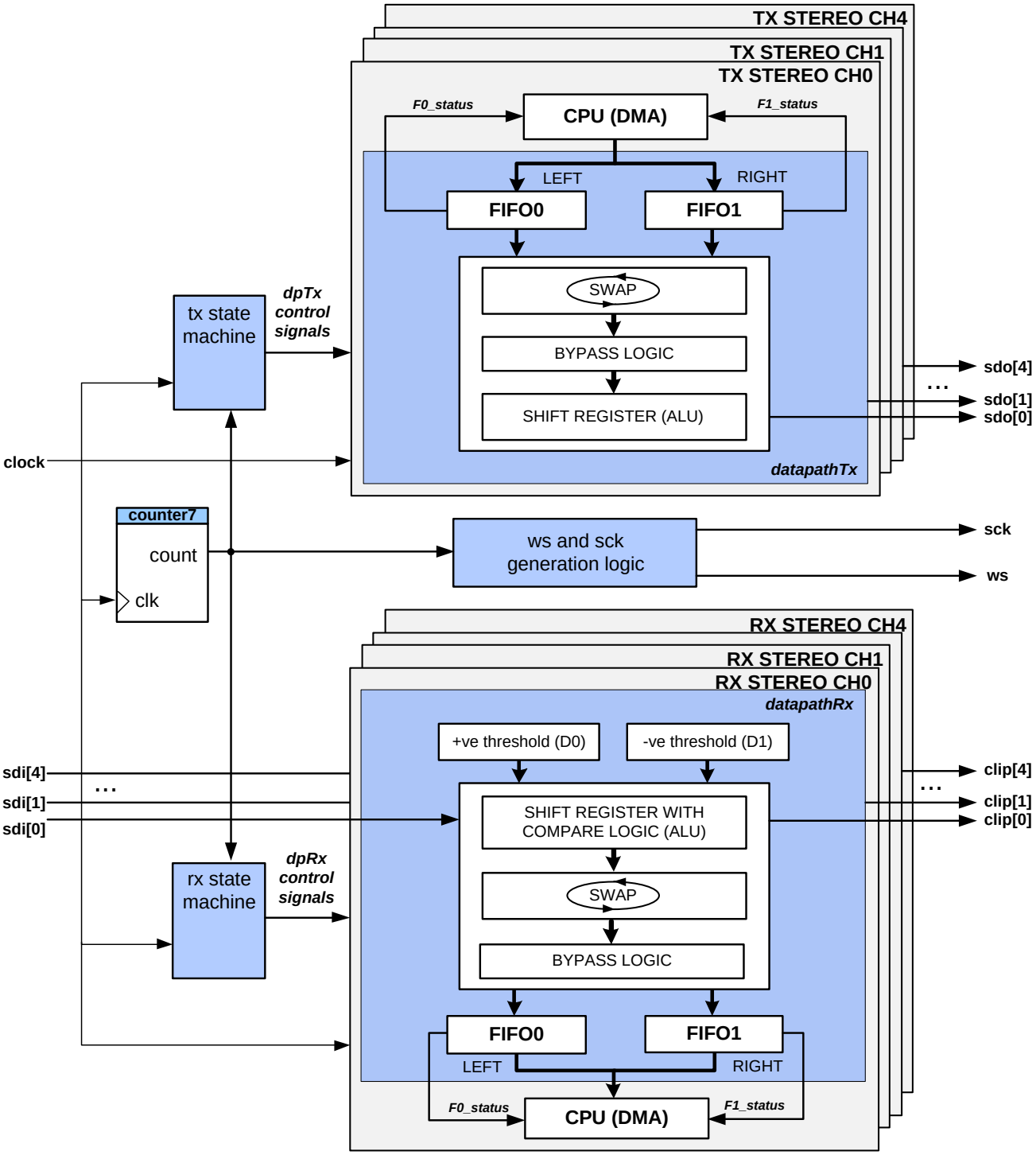
The data for Tx and Rx is independent byte streams. The byte streams are packed with the most significant byte first and the most significant bit in bit 7 of the first word. The number of bytes used for each sample (for the left or right channel) is the minimum number of bytes to hold a sample. Any unused bits are ignored on Rx and are sent by 0 on Rx.

The data stream for one direction can be a single byte stream, or it can be two byte streams. In the case of a single byte stream, the left and right channels are interleaved with a sample for the left channel first followed by the right channel in a single FIFO (FIFO 0). In the two-stream case, the left and right channel byte streams use separate FIFOs, FIFO 0 and FIFO 1. Each FIFO has 4-byte depths.

1. N – Number of stereo channel for Rx, Tx or both Rx and Tx.

## Block Diagram and Configuration

The I2S component is implemented as a set of configured UDBs. The implementation is shown in the following block diagram. Each I2S data line can support one stereo channel data (i.e. two audio outputs, left and right). Additionally the component can be configured for one mono channel data. Then only one audio output (left or right) of stereo channel 0 is used.



## Enabling

The Rx and Tx directions have separate enables. When not enabled, the Tx direction transmits all 0 values, and the Rx direction ignores all received data. The transition into and out of the enabled state occurs at a word select boundary such that a left/right sample pair is always transmitted or received.

## Clock Selection

There is no internal clock in this component. You must attach a clock source. The clock rate provided must be two times the desired clock rate for the output serial clock (sck) and can be calculated as follows:

$$f_{CLOCK} = 2 \times f_S \times t_{WS}$$

- $f_{CLOCK}$ – component clock frequency
- $f_S$ – audio sampling frequency
- $t_{WS}$ – word select period

The following table shows the component clock rates for most common audio sampling frequencies and word select period combinations.

| Sampling Frequency ($f_S$) | Component Clock Rate ($f_{CLOCK}$) MHz | | | |
|---|---|---|---|---|
| | $t_{WS}$ = 16 bits | $t_{WS}$ = 32 bits | $t_{WS}$ = 48 bits | $t_{WS}$ = 64 bits |
| 8 kHz | 0.2560 | 0.5120 | 0.7680 | 1.0240 |
| 16 kHz | 0.5120 | 1.0240 | 1.5360 | 2.0480 |
| 32 kHz | 1.0240 | 2.0480 | 3.0720 | 4.0960 |
| 44.1 kHz | 1.4112 | 2.8224 | 4.2336 | 5.6448 |
| 48 kHz | 1.5360 | 3.0720 | 4.6080 | 6.1440 |
| 88.2 kHz | 2.8224 | 5.6448 | 8.4672 | 11.2896 |
| 96 kHz | 3.0720 | 6.1440 | 9.2160 | 12.2880 |
| 192 kHz | 6.1440 | 12.2880 | N/A | N/A |

## DMA Support

The I2S interface is a continuous interface that requires an uninterrupted stream of data. For most applications, this requires the use of DMA transfers to prevent the underflow of the Tx direction or the overflow of the Rx direction.

The I2S can drive up to two DMA components for each stereo channel within Rx or Tx direction. The following table provides register definitions along with output signals intended for component operation with DMA.

| Register Name [2] | Direction | DMA Request Signal | DMA Request Type | Description |
|---|---|---|---|---|
| I2S_RX_CHn_F0_PTR | Source | rx_dma0[n] | Level | Receive FIFO for Left or Interleaved channel |
| I2S_RX_CHn_F1_PTR | Source | rx_dma1[n] | Level | Receive FIFO for Right channel |
| I2S_TX_CHn_F0_PTR | Destination | tx_dma0[n] | Level | Transmit FIFO for Left or Interleaved channel |
| I2S_TX_CHn_F1_PTR | Destination | tx_dma1[n] | Level | Transmit FIFO for Right channel |

By default, the DMA request signals are defined as Rx FIFO not empty and Tx FIFO not full. That is, the rx_dma0 / rx_dma1 signal is asserted when there is at least 1 byte available to be read from the Rx FIFO 0 / FIFO 1. Likewise, the tx_dma0 / tx_dma1 is asserted when there is room for at least 1 byte in the Tx FIFO 0 / FIFO 1.

These signals can be changed to FIFO at least half full for Rx and FIFO at least half empty for Tx by setting the FIFO level bits in the associated Auxiliary Control Register (ACTL). In this case, the rx_dma0 / rx_dma1 signal will be asserted when there is at least 2 bytes available to be read from the Rx FIFO 0 / FIFO 1, and the tx_dma0 / tx_dma1 will be asserted when there is room for at least 2 bytes in the Tx FIFO 0 / FIFO 1.

The following code shows a general example of how to set the FIFO level bits in the ACTL register:

```
I2S_<dir>_CHn_ACTL_REG |= I2S_FIFOx_LVL
```

- <dir> – direction. Valid value: RX or TX.
- n – stereo channel number. Valid range: 0-4.
- x – FIFO number. Valid value: 0 or 1.

The following example shows how to set the FIFO 0 level bit for the Tx stereo channel 1:

```
I2S_TX_CH1_ACTL_REG |= I2S_FIFO0_LVL
```

**Note** The ACTL is a shared device register that must not be concurrently accessed. Therefore, this operation must be protected by a critical section to ensure the operation is atomic.

## Bit Resolution

The number of bits is not fixed in the I2S standard, but 8 to 32 bits of data for each sample is the practical range. The bit resolution can be static or dynamic depending on the Bit resolution setting on the **Basic** tab. When static bit resolution is selected, the number of data bits for each

[2] "n" specifies the stereo channel number. Valid range 0-4 depending on the number of channels selected for the associated direction. For example I2S_RX_CH0_F0_PTR would specify FIFO 0 of Rx stereo channel 0 as the DMA source. Each FIFO has 4-byte depth.

sample can be set between 8 and 32. In case of the dynamic bit resolution is selected, the data bits can only be set to 8, 16, 24 and 32. If byte swap option is selected the maximum data bits per sample supported is 24.

## Byte Swapping

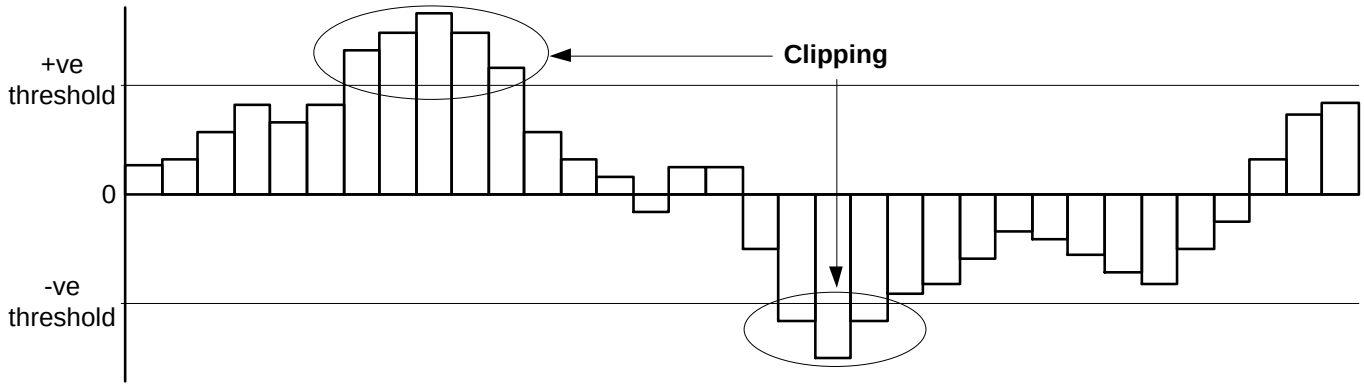
The I2S data format is big-endian (MSB first), whereas other audio source/sink interfaces (such as USB) accept audio samples in little-endian format (LSB first). To allow data transfer between a source and destination that have different endianess the I2S component supports data endianness conversion. When byte swap option is selected, the component swaps the order of the data bytes before capturing the audio samples to Rx FIFO(s) or outputting them onto the Tx sdo line.

Byte swap is supported only in the following configuration:

- The Data bits parameter is set between 9 and 24 bits. If dynamic bit resolution is selected, the data bits must be set to 16 or 24 bits.
- Word select period = 64.
- Data interleaving must be set to **Interleaved** for the I2S direction which requires byte swapping.

## Clip Detection

A digital audio signal is clipping when the numerical values of data samples that represent the signal exceed the maximum permissible limits. An example of digital signal clipping is shown in the following diagram.




If Rx clip detection is enabled, the component detects that incoming digital audio data is clipping and provides an indication by triggering a clip signal. In this mode, the MSB of the received audio sample is compared with a predefined negative and positive clipping thresholds and a clip output is asserted when the sample amplitude crosses the thresholds. The threshold values can be entered using +ve Threshold and -ve Threshold parameters on the **Advanced** tab or by SetPositiveClipThreshold() / SetNegativeClipThreshold() API call.

## Error Handling

Two error conditions for the component can occur if the transmit FIFO is empty and a subsequent read occurs (transmit underflow), or if the receive FIFO is full and a subsequent write occurs (receive overflow).

While transmission is enabled, if the transmit FIFO becomes empty and data is not available for transmission (Transmit underflow), the component forces the constant transmission of 0s. Before transmission begins again, transmission must be disabled, the FIFOs should be cleared, data for transmit must be buffered, and then transmission re-enabled. The CPU can monitor this underflow condition using the transmit status bit I2S_TX_FIFO_UNDERFLOW. An interrupt can also be configured for this error condition.

While reception is enabled, if the receive FIFO becomes full and additional data is received (Receive overflow), the component stops capturing data. Before reception begins again, reception must be disabled, the FIFOs should be cleared, and then reception re-enabled. The CPU can monitor this overflow condition using the receive status bit I2S_RX_FIFO_OVERFLOW. An interrupt can also be configured for this error condition.

# Registers

## I2S_CONTROL_REG

| Bits | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| **Name** | NA:000 | | | data bits | | enable | rxenable | txenable |

| Bits | Description |
|---|---|
| data bits | Sets the number of data bits for each channel. Applicable only if a dynamic bit resolution is selected.<br>0x0: Data bits = 8<br>0x1: Data bits = 16<br>0x2: Data bits = 24<br>0x3: Data bits = 32 |
| enable | Enables/Disables I2S component.<br>0: Disabled<br>1: Enabled |
| rxenable | Enables/Disables Rx and Tx directions, respectively. |
| txenable | 0: Disabled<br>1: Enabled |

## I2S_TX_STATUS0_REG (CH0 and CH1)

| Bits | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | INT | NA:0 | CH1 F1 | CH1 F0 | CH1 UND | CH0 F1 | CH0 F0 | CH0 UND |

## I2S_TX_STATUS1_REG (CH 2 and CH3)

| Bits | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | INT | NA:0 | CH3 F1 | CH3 F0 | CH3 UND | CH2 F1 | CH2 F0 | CH2 UND |

## I2S_TX_STATUS2_REG (CH4)

| Bits | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | INT | NA:0000 | | | | CH4 F1 | CH4 F0 | CH4 UND |

| Bits | Description |
|---|---|
| INT | Interrupt. Logical OR of bits 5:0. |
| CHn F1 | Tx stereo channel n FIFO 1 not full status. Set when FIFO 1 is not full. |
| CHn F0 | Tx stereo channel n FIFO 0 not full status. Set when FIFO 0 is not full. |
| CHn UND | Tx stereo channel n underflow. Set when FIFO 0 or FIFO 1 underflow event has been captured. The bit is clear on read. Once the bit is set, it stays set until a subsequent status read by the CPU. |

The registers can be read with the I2S_ReadTxStatus() API function.

## I2S_RX_STATUS0_REG (CH0 and CH1)

| Bits | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | INT | NA:0 | CH1 F1 | CH1 F0 | CH1 OVER | CH0 F1 | CH0 F0 | CH0 OVER |

## I2S_RX_STATUS1_REG (CH2 and CH3)

| Bits | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | INT | NA:0 | CH3 F1 | CH3 F0 | CH3 OVER | CH2 F1 | CH2 F0 | CH2 OVER |

### I2S_RX_STATUS2_REG (CH4)

| Bits | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | INT | NA:0000 | | | | CH4 F1 | CH4 F0 | CH4 OVER |

| Bits | Description |
|---|---|
| INT | Interrupt. Logical OR of bits 5:0. |
| CHn F1 | Rx stereo channel n FIFO 1 not empty status. Set when FIFO 1 is not empty. |
| CHn F0 | Tx stereo channel n FIFO 0 not empty status. Set when FIFO 0 is not empty. |
| CHn UND | Tx stereo channel n overflow. Set when FIFO 0 or FIFO 1 overflow event has been captured. The bit is clear on read. Once the bit is set, it stays set until a subsequent status read by the CPU. |

The registers can be read with the I2S_ReadRxStatus() API function.

# Component Debug Window

PSoC Creator allows you to view debug information about components in your design. Each component window lists the memory and registers for the instance. For detailed hardware registers descriptions, refer to the appropriate device technical reference manual. For detailed UDB registers descriptions used in the component, refer to the Registers section of this datasheet.

To open the Component Debug window:

1. Make sure the debugger is running or in break mode.
2. Choose **Windows > Components**… from the **Debug** menu.
3. In the Component Window Selector dialog, select the component instances to view and click **OK**.

The selected Component Debug window(s) will open within the debugger framework. Refer to the "Component Debug Window" topic in the PSoC Creator Help for more information.

# Resources

The I2S component is placed throughout the UDB array. The component utilizes the following resources.

| Configuration [3] | Resource Type | | | | | |
|---|---|---|---|---|---|---|
| | Datapath Cells | Macrocells | Status Cells [4] | Control Cells | DMA Channels | Interrupts |
| Rx Only | N | 8 + 2 × N | 1 + (N - 1)/2 | 2 | – | – |
| Tx Only | N | 7 + 2 × N | 1 + (N - 1)/2 | 2 | – | – |
| Rx and Tx | N | 13 + 2 × N | 1+N/2 | 2 | – | – |
| Clip detect [5] | – | N + 1 | – | – | – | – |

# DC and AC Electrical Characteristics

Specifications are valid for –40 °C ≤ $T_A$ ≤ 85 °C and $T_J$ ≤ 100 °C, except where noted.
Specifications are valid for 1.71 V to 5.5 V, except where noted.

## DC Characteristics

| Parameter | Description | Min | Typ [6] | Max | Units [7] |
|---|---|---|---|---|---|
| $I_{DD(RX)}$ | Component current consumption (Rx Only) | | | | |
| | Idle current [8] | – | 15 | – | µA/MHz |
| | Operating current [9] | – | 5 × N | – | µA/MHz |
| $I_{DD(TX)}$ | Component current consumption (Tx Only) | | | | |
| | Idle current [8] | – | 15 | – | µA/MHz |
| | Operating current [9] | – | 5 × N | – | µA/MHz |

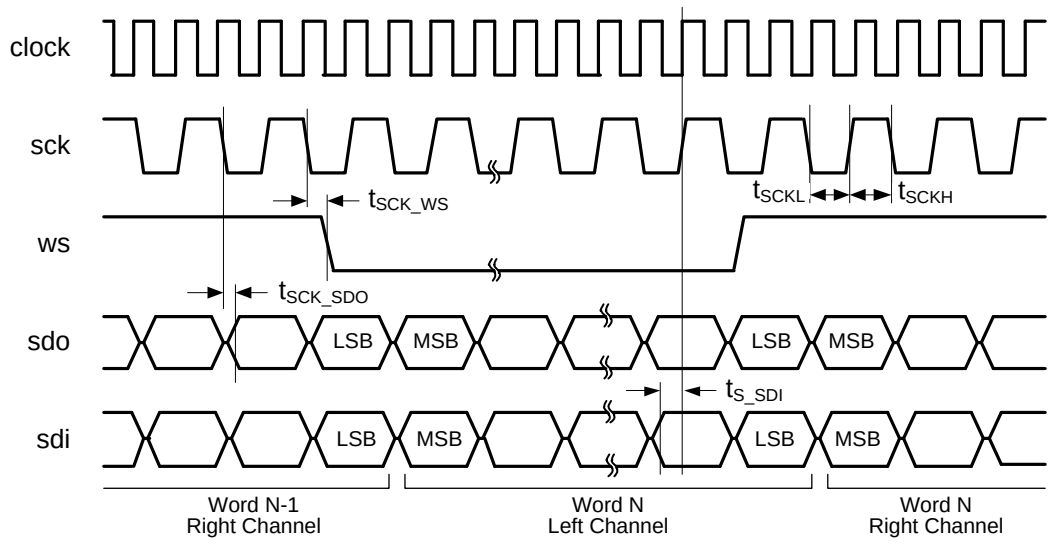
3. N – Number of stereo channel for Rx, Tx or both Rx and Tx.
4. One more status cell is used when the Clock synchronization parameter is set to Asynchronous. Division sign (/) denotes the integer division. For example, when N=2, the number of status cells is 1+(2-1)/2=1;
5. Additional macrocell usage when clip detection is enabled on Rx. N refers to the number of Rx stereo channels.
6. Device IO and clock distribution current are not included. The values are at 25 °C.
7. Current consumption is specified with respect to the incoming component clock.
8. Current consumed by component while it is enabled but not transmitting/receiving data.
9. Additional current consumed by component while it is enabled and transmitting/receiving data. This value should be added to the Idle current. N = Number of the stereo channel for Rx, Tx, or both Rx and Tx.

| Parameter | Description | Min | Typ [6] | Max | Units [7] |
|---|---|---|---|---|---|
| $I_{DD(RX_TX)}$ | Component current consumption (Rx and Tx) | | | | |
| | Idle current [8] | – | 16 | – | µA/MHz |
| | Operating current [9] | – | 5 × N | – | µA/MHz |

## AC Characteristics

| Parameter | Description | Min | Typ | Max | Unit |
|---|---|---|---|---|---|
| $f_{SCK}$ | Serial clock (output) frequency | – | – | 6.144 | MHz |
| $f_{CLOCK}$ [10] | Component clock frequency | – | 2 × $f_{SCK}$ | – | MHz |
| $t_{SCKH}$ | SCK high-level time | – | 0.5 | – | 1/$f_{SCLK}$ |
| $t_{SCKL}$ | SCK low-level time | – | 0.5 | – | 1/$f_{SCLK}$ |
| $t_{SCK_WS}$ | Delay time, SCK falling edge to WS valid | –20 | – | 20 | ns |
| $t_{SCK_SDO}$ | Delay time, SCK falling edge to SDO valid | –20 | – | 20 | ns |
| $t_{S_SDI}$ [11] | SDI setup time | 25 | – | – | ns |

### Figure 1. Data Transition Timing Diagram




10. The maximum component clock frequency is derived from $t_{CLK_SCK}$ in combination with the routing path delays of the SCK output and the SDI input (described later in this document). Typical values provide a maximum safe operating frequency of the component. The component can run at higher clock frequencies, at which point validation of the timing requirements with STA results is necessary.

11. The $V_{DDIO}$ supply voltage from 3.0 V to 5.5 V.

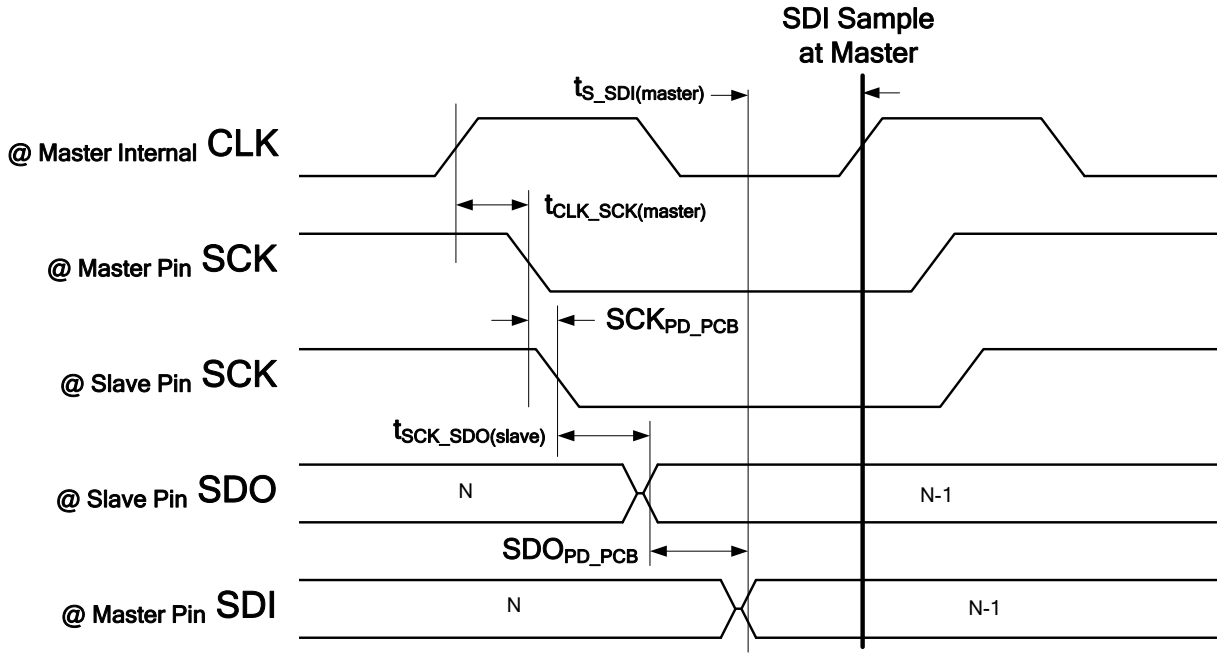
**Note** All of the component internal logic operates entirely from the input 2X clock. This causes the need to reference some timing constraints, such as $t_{S_SDI}$ and $t_{CLK_SCK}$, with respect to this internal clock (described later in this document).

## How to Use STA results for Characteristics Data

### $f_{SCK}$

The maximum frequency of SCK (or the maximum bit rate) is not provided directly in the STA. However, the data provided in the STA results indicates some of the internal logic timing constraints. To calculate the maximum bit rate, several factors must be considered. Board layout and slave communication device specs are needed to fully understand the maximum. A limiting factor in this parameter is the round trip path delay from the falling edge of SCK at the pin of the master to the slave, and the path delay of the SDO output of the slave back to the SDI input of the master. In this case, the component must meet the setup time of SDI at the master with the following equation:

**Figure 2. Calculating Maximum $f_{SCK}$ Frequency**




In this case, the $f_{SCK}$ frequency is calculated using the equation below:

$$f_{SCK} < 1 \div [2 \times [t_{RT_PD} + t_{CLK_SCK(master)} + t_{S_SDI(master)}]]$$

Where:

$$t_{RT_PD} = [SCK_{PD_PCB} + t_{SCK_SDO(slave)} + SDO_{PD_PCB}]$$

$SCK_{PD_PCB}$ is the PCB path delay of SCK from the pin of the master component to the pin of the slave device.

$t_{SCK_SDO(slave)}$ comes from the Slave Device datasheet.

$t_{CLK_SCK(master)}$ is the internal CLK to SCK pin path delay of the master component.

This is provided in the STA results clock to output section:

**- Clock To Output Section**

**- CLK**

| Source | Destination | Delay (ns) |
|---|---|---|
| \I2S:BitCounter\/count_0 | SCK(0)_PAD | 24.484 |
| Net_5/q | SDO(0)_PAD | 23.808 |
| Net_4/q | WS(0)_PAD | 22.958 |

$t_{S_SDI(master)}$ is the SDI pin to the internal logic path delay of the master component. This is provided in the STA results input to clock section:

**- Input To Clock Section**

**- CLK**

| Source | Destination | Delay (ns) |
|---|---|---|
| SDI(0)_PAD | \I2S:bI2S:rx_data_in_0\/main_2 | 15.863 |

The final equation that provides the maximum frequency of SCK follows; the maximum bit rate is:

$$f_{SCK}\,(\text{Max.}) = 1 \div [2 \times [t_{CLK_SCK(master)} + SCK_{PD_PCB} + t_{SCK_SDO(slave)} + SDO_{PD_PCB} + t_{S_SDI(master)}]]$$

### $f_{CLOCK}$

The maximum component clock frequency is provided in the Timing results in the clock summary as the named external clock (CLK in this case). An example of the internal clock limitations from the STA report follows:

**- Clock Summary Section**

| Clock | Domain | Nominal Frequency | Required Frequency | Maximum Frequency | Violation |
|---|---|---|---|---|---|
| CyILO | CyILO | 1.000 kHz | 1.000 kHz | N/A | |
| CyIMO | CyIMO | 3.000 MHz | 3.000 MHz | N/A | |
| CyMASTER_CLK | CyMASTER_CLK | 24.000 MHz | 24.000 MHz | N/A | |
| CLK | CyMASTER_CLK | 12.000 MHz | 12.000 MHz | 61.573 MHz | |
| CyBUS_CLK | CyMASTER_CLK | 24.000 MHz | 24.000 MHz | N/A | |
| CyPLL_OUT | CyPLL_OUT | 24.000 MHz | 24.000 MHz | N/A | |

### $t_{SCKH}$

The I2S component generates a 50-percent duty cycle SCK.

### $t_{SCKL}$

The I2S component generates a 50-percent duty cycle SCK.

### $t_{SCK_WS}$

The delay between SCK falling edge and WS valid. This value can be calculated as the difference between clock to output times for WS and SCK pins. The data is extracted from the STA report:

**- Clock To Output Section**

**- CLK**

| Source | Destination | Delay (ns) |
|---|---|---|
| \I2S:BitCounter\/count_0 | SCK(0)_PAD | 24.484 |
| Net_5/q | SDO(0)_PAD | 23.808 |
| Net_4/q | WS(0)_PAD | 22.958 |

### $t_{SCK_SDO}$

The delay between SCK falling edge and WS valid. This value can be calculated as the difference between clock to output times for SDO and SCK pins. The values are provided in the STA results clock to output times:

**- Clock To Output Section**

**- CLK**

| Source | Destination | Delay (ns) |
|---|---|---|
| \I2S:BitCounter\/count_0 | SCK(0)_PAD | 24.484 |
| Net_5/q | SDO(0)_PAD | 23.808 |
| Net_4/q | WS(0)_PAD | 22.958 |

### $t_{S_SDI}$

SDI setup time is the SDI pin to internal logic path delay of the master component. This is provided in the STA results Input to Clock Section:

**- Input To Clock Section**

**- CLK**

| Source | Destination | Delay (ns) |
|---|---|---|
| SDI(0)_PAD | \I2S:bI2S:rx_data_in_0\/main_2 | 15.863 |

# Component Changes

This section lists the major changes in the component from the previous version.

| Version | Description of Changes | Reason for Changes / Impact |
|---|---|---|
| 2.70 | Added support for asynchronous component clocking. | Allows component to run off of external clock that is asynchronous with respect to the system clock. |
| | Synchronized Rx enabling with respect to SCK. | Depending on when the Rx direction was enabled, the received data might be shifted left by one bit position. |
| 2.60 | Added dynamic bit resolution control. | Allows the number of bits to be dynamically configurable in firmware to support different audio streams. |
| | Added mono channel support. | Allows transferring of audio samples only in one of the stereo channel. |
| | Added multi-channel support. | Extends the number of I2S stereo streams up to 5 (10 audio channels). |
| | Added clip detection capability for the input audio samples on Rx direction. | Detects and indicates that incoming digital audio data is clipping. |
| | Added byte swap support for endian compatibility. | Allows data transfer between a source and destination that have different endianess. |
| | Datasheet update and corrections. | To reflect all changes in version 2.60. |
| 2.50 | Removed PSoC 5 support. | PSoC 5 is no longer supported starting from PSoC Creator 3.0 release. |
| 2.40.a | Updated datasheet with memory usage for PSoC 4. | |
| 2.40 | Added MISRA Compliance section. | The component does not have any specific deviations. |
| 2.30 | Added all component APIs with the CYREENTRANT keyword when they are included in the .cyre file. | Not all APIs are truly reentrant. Comments in the component API source files indicate which functions are candidates.<br>This change is required to eliminate compiler warnings for functions that are not reentrant used in a safe way: protected from concurrent calls by flags or Critical Sections. |
| | Added PSoC 5LP support. | |
| | Added DC characteristics section to datasheet. | |
| 2.20 | Added internal FIFO error detection by the component. | Improved error handling by the component in case of an overflow/underflow error occurs. |
| 2.10 | Resampled FIFO block status signals to DP clock. | Allows component to function with the same timing results for all PSoC 3 and PSoC 5 silicons. |
| | Added characterization data to datasheet. | |
| | Minor datasheet edits and updates. | |

| Version | Description of Changes | Reason for Changes / Impact |
|---|---|---|
| 2.0 | Hardware implementation of this component was changed to require a 2X frequency signal on the clock input. The SCK output signal will be generated by dividing the incoming clock by 2. | Improves the control of the timing relationship between the SCK, WS, SDO and SDI signals. |
| | I2S_Start() function updated to match changes in the implementation. The functionality is unchanged. | Simplified implementation required changes to the initialization. |
| | The sleep mode APIs were added. | To support low-power modes. |
| | The status bits tx_not_full and rx_not_emty were changed from clear-on-read to transparent mode. | The status bits for any Full or Empty status from a FIFO need to be transparent to represent just the current live status of the FIFO. |
| | Added DMA Capabilities file to the component. | This file allows I2S to be supported by the DMA Wizard tool in PSoC Creator. |
| | I2S_Stop() API was changed to clear rx and tx FIFOs after component is disabled. | Resets the Tx and Rx FIFOs status to the initial values. Prevents unexpected operations after component is re-enabled. |
| | Added Keil function reentrancy support. | Add the capability for customers to specify individual generated functions as reentrant. |

© Cypress Semiconductor Corporation, 2015. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC® is a registered trademark, and PSoC Creator™ and Programmable System-on-Chip™ are trademarks and of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.