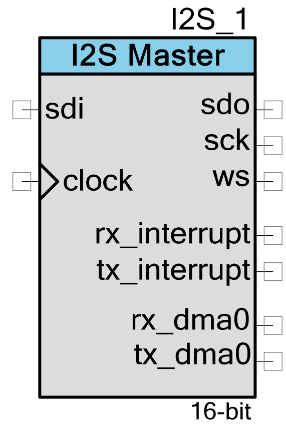
# Inter-IC Sound Bus (I2S)

**2.10**

## Features




- Master only
- 8 - 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: 6.144 MHz
- Tx and Rx FIFO interrupts
- DMA support
- Independent left and right channel FIFOs or interleaved stereo FIFOs
- Independent enable of Rx and Tx

## 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 ADC and DAC components.

# 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 *

Serial data input. Displays if you select an Rx for the **Direction** parameter.

If this signal is connected to an input pin, then the "Input Synchronized" selection for this pin should be disabled. This signal should already be synchronized to SCK and 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 would be:

$$2 \times 48\ KHz \times 64 = 6.144\ MHz$$

## sdo – Output *

Serial data output. Displays if you select a Tx option for the **Direction** parameter.

## sck – Output

Output serial clock.

## ws – Output

Word select output indicates the channel being transmitted.

## rx_interrupt – Output *

Rx direction interrupt. Displays if you select an Rx option for the **Direction** parameter.

## tx_interrupt – Output *

Tx direction interrupt. Displays if you select a Tx option for the **Direction** parameter.

## rx_DMA0 – Output *

Rx direction DMA request for FIFO 0 (Left or Interleaved). Displays if you select "Rx DMA" under the **DMA Request** parameter.

### rx_DMA1 – Output *

Rx direction DMA request for FIFO 1 (Right). Displays if you select "Rx DMA" under the **DMA Request** parameter and "Separated L/R" under the **Data Interleaving** parameter for Rx.

### tx_DMA0– Output *

Tx direction DMA request for FIFO 0 (Left or Interleaved). Displays if you select "Tx DMA" under the **DMA Request** parameter.

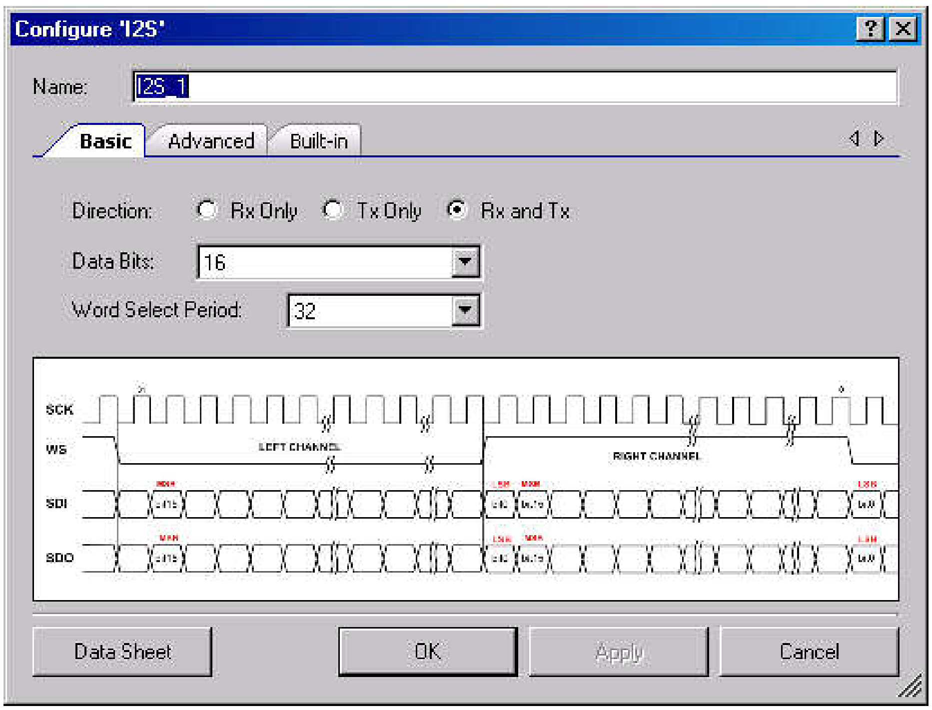
### tx_DMA1 – Output *

Tx direction DMA request for FIFO 1 (Right). Displays if you select "Tx DMA" under the **DMA Request** parameter and "Separated L/R" under the **Data Interleaving** parameter for Tx.

## Parameters and Setup

Drag an I2S component onto your design and double-click it to open the Configure dialog. This dialog has two tabs to guide you through the process of setting up the I2S component.

### Basic Tab



### Direction

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

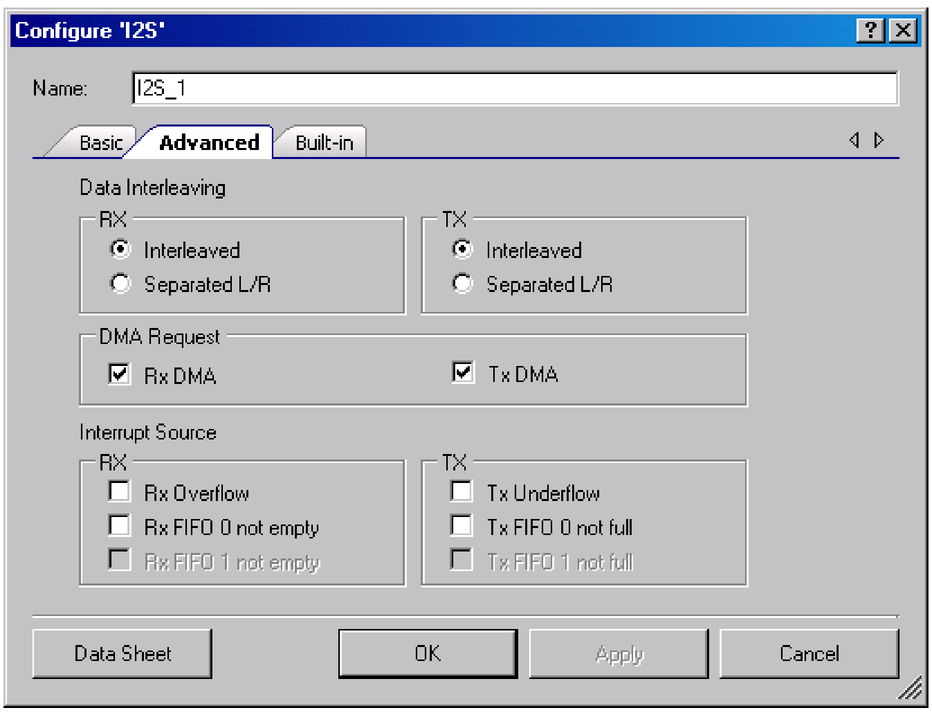
### Data Bits

Determines the number of data bits configured for each sample (hardware compiled). This value can be set between 8 and 32. The default setting is 16.

### Word Select Period

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

## Advanced Tab




### Data Interleaving

Allows you to select whether the data is Interleaved (default) or Separate L/R. You select Rx and Tx independently.

### DMA Request

Allows you to enable and disable the DMA request signals for the component. You set Rx and Tx independently. These options are enabled by default.

### Interrupt Source

Select the source of the I2S interrupts. Rx and Tx interrupts are separate. Multiple sources may be ORed together. 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

## 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).

## Placement

The I2S component is placed throughout the UDB array and all placement information is provided to the API through the *cyfitter.h* file.

## Resources

| Resources | Resource Type | | | | API Memory (Bytes) | | Pins (per External I/O) |
|---|---|---|---|---|---|---|---|
| | Datapath Cells | PLDs | Status Cells | Control/ Count7 Cells | Flash | RAM | |
| Rx Direction | 1 | 2 | 1 | 2 | 220 | 3 | 3 |
| Tx Direction | 1 | 2 | 1 | 2 | 220 | 3 | 3 |
| Rx and Tx | 2 | 4 | 2 | 2 | 326 | 3 | 4 |

# Application Programming Interface

Application Programming Interface (API) routines allow you to configure the component using software. The following table lists and describes the interface to 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. You can rename the instance 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."

| Function | Description |
|---|---|
| void I2S_Start(void) | Starts the I2S interface. |
| void I2S_Stop(void) | Disables the I2S interface. |
| void I2S_EnableTx(void) | Enables the Tx direction of the I2S interface. |
| void I2S_DisableTx(void) | Disables the Tx direction of the I2S interface. |
| void I2S_EnableRx(void) | Enables the Rx direction of the I2S interface. |
| void I2S_DisableRx(void) | Disables the Rx direction of the I2S interface. |
| void I2S_SetRxInterruptMode(uint8 interruptSource) | Sets the interrupt source for the I2S Rx direction interrupt. |
| void I2S_SetTxInterruptMode(uint8 interruptSource) | Sets the interrupt source for the I2S Tx direction interrupt. |
| uint8 I2S_ReadRxStatus(void) | Returns state in the I2S Rx status register. |
| uint8 I2S_ReadTxStatus(void) | Returns state in the I2S Tx status register. |
| uint8 I2S_ReadByte(uint8 wordSelect) | Returns a single byte from the Rx FIFO. |
| void I2S_WriteByte(uint8 wrData, uint8 wordSelect) | Writes a single byte into the Tx FIFO. |
| void I2S_ClearRxFIFO(void) | Clears out the Rx FIFO. |
| void I2S_ClearTxFIFO(void) | Clears out the Tx FIFO. |
| void I2S_Sleep(void) | Saves configuration and disables the I2S interface |
| void I2S_WakeUp(void) | Restores configuration and enables the I2S interface |
| void I2S_SaveConfig(void) | Saves configuration of I2S interface |
| void I2S_RestoreConfig(void) | Restores configuration of I2S interface |
| void I2S_Init(void) | Enables the I2S interface |
| void I2S_Enable(void) | Initializes or restores default I2S configuration |

## 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 in after the first call to the I2S_Start() routine.<br>If reinitialization of the component is required the variable should be set to 0 before the I2S_Start() routine is called. Alternately, the I2S can be reinitialized by calling the I2S_Init() and I2S_Enable() functions. |

## void I2S_Start(void)

**Description:** Starts the I2S interface. Enables Active mode power template bits or clock gating as appropriate. 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:** Disables the I2S interface. Disables Active mode power template bits or clock gating as appropriate. 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_EnableTx(void)

**Description:** Enables the Tx direction of the I2S interface. At the next word select falling edge, transmission will begin.

**Parameters:** None

**Return Value:** None

**Side Effects:** None

## void I2S_DisableTx(void)

**Description:** Disables the Tx direction of the I2S interface. At the next word select falling edge, transmission of data will stop and a constant 0 value will be transmitted.

**Parameters:** None

**Return Value:** None

**Side Effects:** None

## void I2S_EnableRx(void)

**Description:** Enables the Rx direction of the I2S interface. At the next word select falling edge, reception of data will begin.

**Parameters:** None

**Return Value:** None

**Side Effects:** None

## void I2S_DisableRx(void)

**Description:** Disables the Rx direction of the I2S interface. At the next word select falling edge, reception of data will no longer be sent to the receive FIFO.

**Parameters:** None

**Return Value:** None

**Side Effects:** None

## void I2S_SetRxInterruptMode(uint8 interruptSource)

**Description:** Sets the interrupt source for the I2S Rx direction interrupt. Multiple sources may be ORed.

**Parameters:** (uint8) byte containing the constant for the selected interrupt sources.

| I2S Rx Interrupt Source | Value |
|---|---|
| RX_FIFO_OVERFLOW | 0x01 |
| RX_FIFO_0_NOT_EMPTY | 0x02 |
| RX_FIFO_1_NOT_EMPTY | 0x04 |

**Return Value:** None

**Side Effects:** None

## void I2S_SetTxInterruptMode(uint8 interruptSource)

**Description:** Sets the interrupt source for the I2S Tx direction interrupt. Multiple sources may be ORed.

**Parameters:** (uint8) byte containing the constant for the selected interrupt sources.

| I2S Tx Interrupt Source | Value |
|---|---|
| TX_FIFO_UNDERFLOW | 0x01 |
| TX_FIFO_0_NOT_FULL | 0x02 |
| TX_FIFO_1_NOT_FULL | 0x04 |

**Return Value:** None

**Side Effects:** None

## uint8 I2S_ReadRxStatus(void)

**Description:** Returns state in the I2S Rx status register.

**Parameters:** None

**Return Value:** (uint8) state of the I2S Rx status register.

| I2S Rx Status Masks | Value | Type |
|---|---|---|
| RX_FIFO_OVERFLOW | 0x01 | Clear on Read |
| RX_FIFO_0_NOT_EMPTY | 0x02 | Transparent |
| RX_FIFO_1_NOT_EMPTY | 0x04 | Transparent |

**Side Effects:** Clears the bits of the I2S Rx status register that are Clear on Read type.

## uint8 I2S_ReadTxStatus(void)

**Description:** Returns state in the I2S Tx status register.

**Parameters:** None

**Return Value:** (uint8) state of the I2S Tx status register.

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

**Side Effects:** Clears the bits of the I2S Rx status register that are Clear on Read type.

## uint8 I2S_ReadByte(uint8 wordSelect)

**Description:** Returns a single byte from the Rx FIFO. The Rx status should be checked before this call to confirm that the Rx FIFO is not empty.

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

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

**Side Effects:** None

## void I2S_WriteByte(uint8 wrData, uint8 wordSelect)

**Description:** Writes a single byte into the Tx FIFO. The Tx status should be checked before this call to confirm that the Tx FIFO is not full.

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

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

**Return Value:** None

**Side Effects:** None

## void I2S_ClearRxFIFO(void)

**Description:** Clears out the Rx FIFO. Any data present in the FIFO will be lost. Call this function only when the Rx direction is disabled.

**Parameters:** None

**Return Value:** None

**Side Effects:** None

## void I2S_ClearTxFIFO(void)

**Description:** Clears out the Tx FIFO. Any data present in the FIFO will be lost. Call this function only when the Tx direction is disabled.

**Parameters:** None

**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. Then it calls the I2S_Stop() function and calls I2S_SaveConfig() to save the hardware configuration. Disables Active mode power template bits or clock gating as appropriate. 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:** Restores I2S configuration and non-retention register values. Enables Active mode power template bits or clock gating as appropriate. Starts the generation of the sck and ws outputs. 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 _Sleep() or _SaveConfig() function may produce unexpected behavior.

## void I2S_SaveConfig(void)

**Description:** This function saves the component configuration. This will save non-retention registers. This function will also save the current component parameter values, as defined in the Configure dialog or as modified by appropriate APIs. This function is called by the I2S_Sleep() function.

**Parameters:** None

**Return Value:** None

**Side Effects:** None

## void I2S_RestoreConfig(void)

**Description:** This function restores the component configuration. This will restore non-retention registers. This function will also restore the component parameter values to what they were prior to calling the I2S_Sleep() function.This routines is called by I2S_Wakeup() to restore component when it exits sleep.

**Parameters:** None

**Return Value:** None

**Side Effects:** Must be called only after I2S_SaveConfig() routine. Otherwise the component configuration will be overwritten with its initial setting.

## void I2S_Init(void)

**Description:** Initializes or restores default I2S configuration provided with customizer that defines interrupt sources for the component.

**Parameters:** None

**Return Value:** None

**Side Effects:** Restores only mask registers for interrupt generation. It will not clear data from the FIFOs and will not reset component hardware state machines.

### void I2S_Enable(void)

**Description:** Activates the hardware and begins component operation. It is not necessary to call I2S_Enable() because the I2S_Start() routine calls this function, which is the preferred method to begin component operation.

**Parameters:** None

**Return Value:** None

**Side Effects:** None

# 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.

# 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 it is necessary for the application to have different configurations for the Rx and Tx, then two unidirectional component instances should be used.

## Data Stream Format

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 (for the left or right channel) is the minimum number of bytes to hold a sample. Any unused bits will be ignored on Tx and will be 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 the two stream case the left and right channel byte streams use separate FIFOs.

## DMA

The I2S interface is a continuous interface which requires an uninterrupted stream of data. For most applications this will require 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 direction. The DMA Wizard can be used to configure DMA operation as follows:

| Name of DMA source/ destination in the DMA Wizard | Direction | DMA Request Signal | DMA Request Type | Description |
|---|---|---|---|---|
| I2S_RX_FIFO_0_PTR | Source | rx_DMA0 | Level | Receive FIFO for Left or Interleaved channel |
| I2S_RX_FIFO_1_PTR | Source | rx_DMA1 | Level | Receive FIFO for Right channel |
| I2S_TX_FIFO_0_PTR | Destination | tx_DMA0 | Level | Transmit FIFO for Left or Interleaved channel |
| I2S_TX_FIFO_1_PTR | Destination | tx_DMA1 | Level | Transmit FIFO for Right channel |

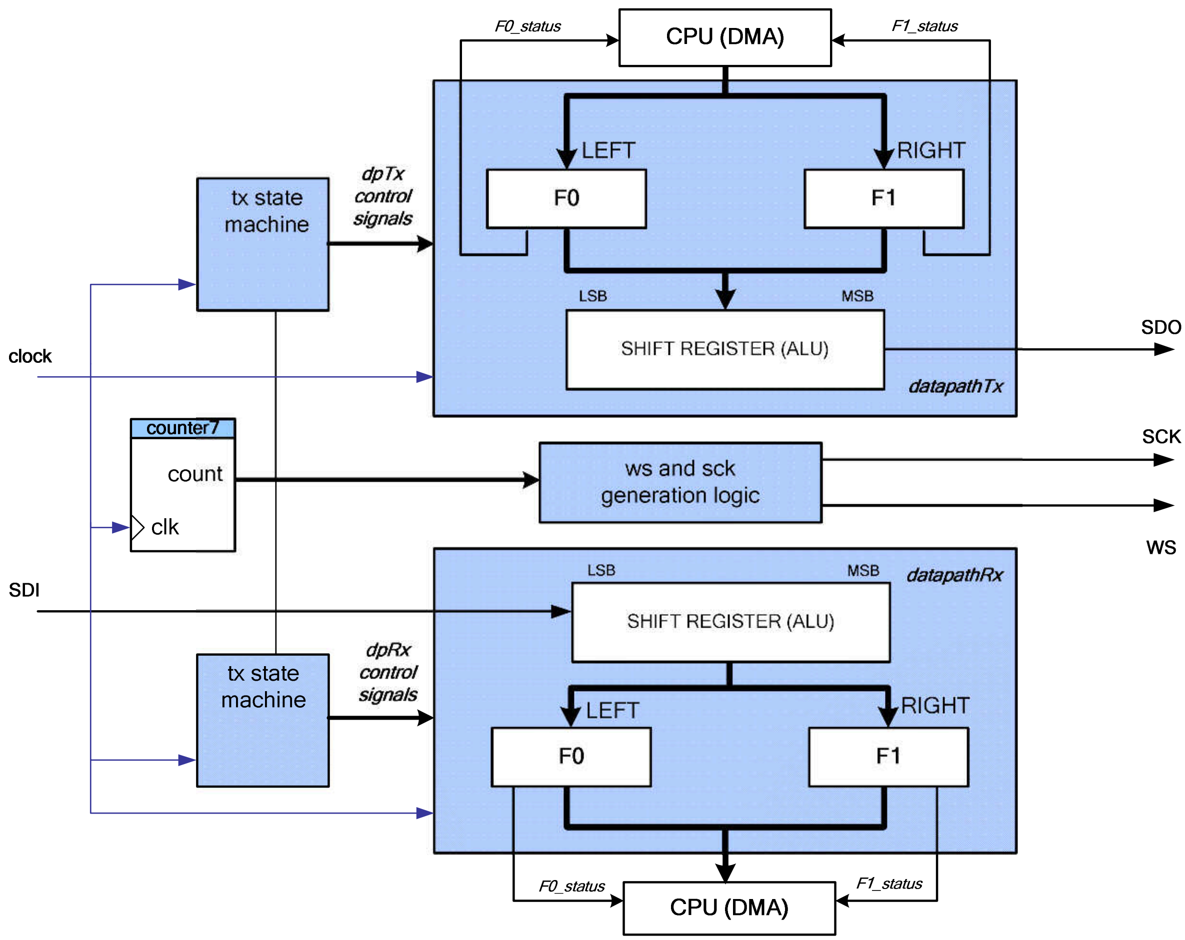
In all cases a high signal on the DMA request signal indicates that an additional single byte may be transferred.

### Enabling

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

# Block Diagram and Configuration

The I2S is implemented as a set of configured UDBs. The implementation is shown in the following block diagram.




# Registers

## I2S_CONTROL_REG

| Bits | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Value | reserved | | | | | enable | rxenable | txenable |

- enable: enable / disable I2S component
- rxenable, txenable: enable / disable Rx and Tx directions respectively

### I2S_TX_STATUS_REG

| Bits | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Value | reserved | | | | | F1_not_full | F0_not_full | underflow |

- F1_not_full: if set tx FIFO 1 is not full
- F0_not_full: if set tx FIFO 0 is not full
- underflow: if set tx FIFOs underflow event has occurred

The register value may be read with the I2S_ReadTxStatus() API function.

### I2S_RX_STATUS_REG

| Bits | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Value | reserved | | | | | F1_not_empty | F0_not_empty | overflow |

- F1_not_empty: if set rx FIFO 1 is not empty
- F0_not_empty: if set rx FIFO 0 is not empty
- overflow: if set rx FIFOs overflow event has occurred

The register value may be read with the I2S_ReadRxStatus() API function.

# References

Not applicable

# DC and AC Electrical Characteristics

The following values are indicative of expected performance and based on initial characterization data.

## Timing Characteristics “Maximum with Nominal Routing”

| Parameter | Description | Min | Typ | Max [1] | Unit |
|---|---|---|---|---|---|
| $f_S$ | Sampling frequency[2] | – | – | 96 | kHz |
| $t_{WS}$ | Word select period | 16 | – | 64 | $1/f_{SCK}$ |
| $f_{SCK}$ | Serial clock (output) frequency | – | – | 6.144 | MHz |
| $f_{CLOCK}$ | 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 |
| $t_{S\ SDI}$ | SDI setup time | 25 | – | – | ns |

[1] The component 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). These “Nominal” numbers provide a maximum safe operating frequency of the component under nominal routing conditions. It is possible to run the component at higher clock frequencies, at which point you will need to validate the timing requirements with STA results.

[2] Sampling frequency is given for the maximum word select period.

## Timing Characteristics “Maximum with All Routing”

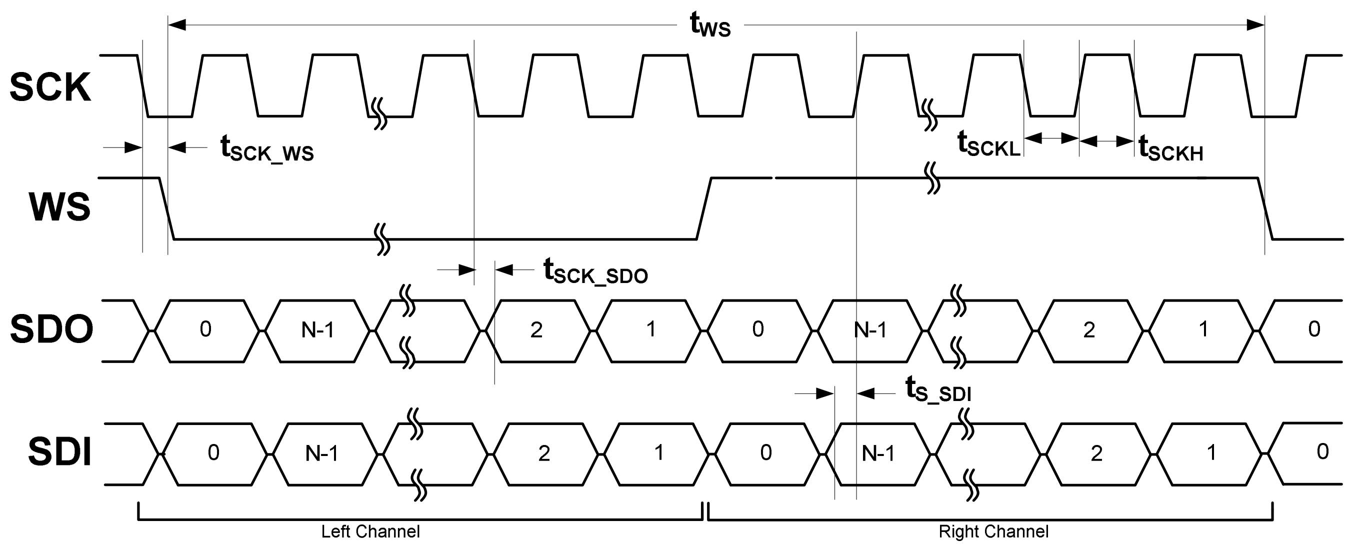
| Parameter | Description | Min | Typ | Max[1] | Unit |
|---|---|---|---|---|---|
| $f_S$ | Sampling frequency[2] | – | – | 96 | kHz |
| $t_{WS}$ | Word select period | 16 | – | 64 | $t_{SCK}$ |
| $f_{SCK}$ | Serial clock (output) frequency | – | – | 6.144 | MHz |
| $f_{CLOCK}$ | Component clock frequency | – | – | 2 * $f_{SCK}$ | MHz |

[1] Maximum for “All Routing” is calculated by <nominal>/2 rounded to the nearest integer.  This value provides a basis for the user to not have to worry about meeting timing if they are running at or below this component frequency.

[2] Sampling frequency is given for the maximum word select period.

| Parameter | Description | Min | Typ | Max[1] | Unit |
|---|---|---|---|---|---|
| $t_{SCKH}$ | 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 |
| $t_{S\ SDI}$ | SDI setup time | 25 | – | – | ns |

**Figure 1. Data Transition Timing Diagram**



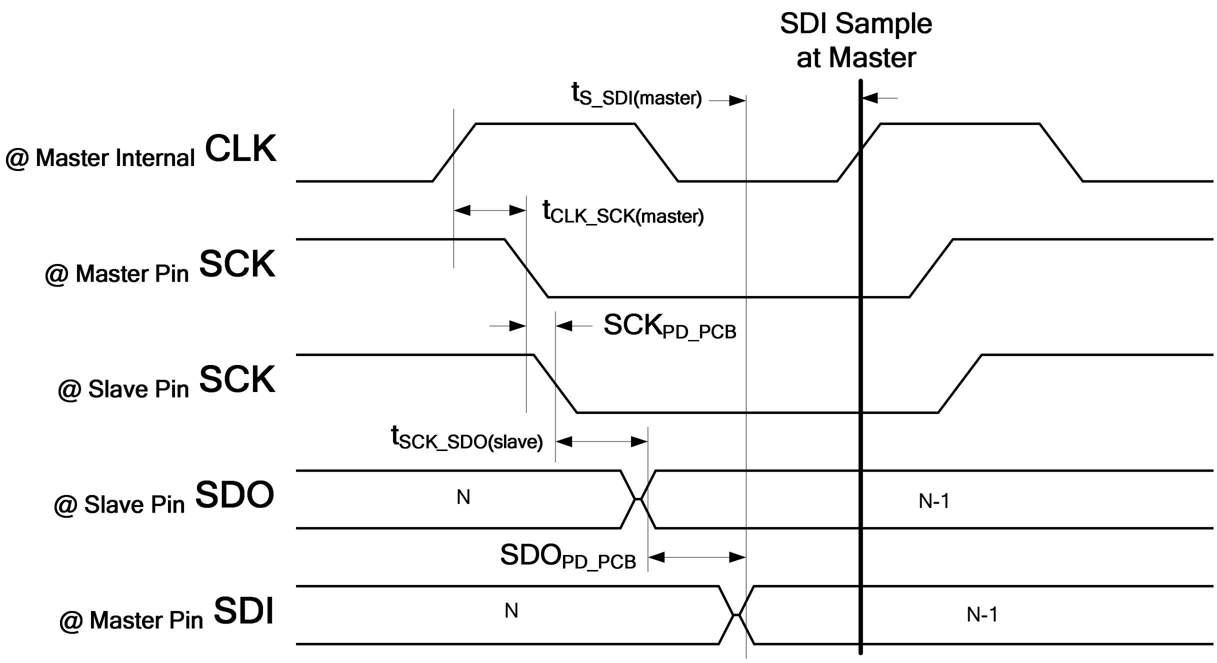

## Component Clock Rates for Common Audio Sampling Frequencies

| Sampling frequency | Component Clock Frequency ($f_{CLK}$) MHz | | | |
|---|---|---|---|---|
| | 8 bit | 16 bit | 24 bit | 32 bit |
| 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 |

## 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 indicate some of the internal logic timing constraints. To calculate the maximum bit-rate several factors must be taken into account. Board Layout and slave communication device specs are needed to fully understand the maximum. The main 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 must be calculated using the equation below:

$$f_{SCK} < 1 / [2 \times [t_{RT\_PD} + t_{CLK\_SCK(master)} + t_{S\_SDI(master)}]]$$

Where

$$t_{RT\_PD} = [SCK_{PD\_PCB} + t_{SCK\_SDO(slave)} + SDO_{PD\_PCB}]$$

and:

$SCK_{PD\_PCB}$ is the PCB Path delay of SCK from the pin of the master component to the pin of the slave device.

$t_{SCK\_SDO(slave)}$ must come from the Slave Device Datasheet

$SDO_{PD\_PCB}$ is the PCB path delay of the SDO from the pin of the slave component to the SDI pin of the master component

$t_{CLK\_SCK(master)}$ is the internal CLK to SCK pin path delay of the master component. This is provided in the STA results clock to output times as shown below:

**–Clock to output times**

**+Clock to output times from clock BUS_CLK**

**–Clock to output times from clock CLK**

| Start | Register | End | Delay (ns) |
|---|---|---|---|
| CLK | \I2S_1:SyncCtl:ControlReg\:controlcell.control_2 | WS(0):iocell.pad_out | 34.450 |
| CLK | \I2S_1:BitCounter\:count7cell.count_5 | WS(0):iocell.pad_out | 33.052 |
| CLK | \I2S_1:Tx:dpTx:u0\:datapathcell.regq_a0 | SDO(0):iocell.pad_out | 27.673 |
| CLK | Net_9:macrocell.mc_q | SCK(0):iocell.pad_out | 23.299 |

$t_{S\_SDI(master)}$ is the SDI pin to internal logic path delay of the master component. This is provided in the STA results input setup times as shown below:

**–Setup times**

**–Setup times to clock CLK**

| Start | Register | Clock | Delay (ns) |
|---|---|---|---|
| SDI(0):iocell.pad_in | \I2S_1:Rx:dpRx:u0\:datapathcell.regd_a0 | CLK | 18.965 |

The final equation that will provide the maximum frequency of SCK and therefore the maximum bit-rate is:

$f_{SCK}$ (Max.) = 1 / [2 x [$t_{CLK\_SCK(master)}$ + $SCK_{PD\_PCB}$ + $t_{SCK\_SDO(slave)}$ + $SDO_{PD\_PCB}$ + $t_{S\_SDI(master)}$]]

**$f_{clock}$** Maximum Component Clock Frequency is provided in Timing results in the clock summary as the named external clock (CLK in this case). An example of the internal clock limitations from the _timing.html file is below:

**–Clock Summary**

| Clock | Actual Freq | Max Freq | Violation |
|---|---|---|---|
| BUS_CLK | 24.000 MHz | 119.023 MHz | |
| CLK | 24.000 MHz | 61.224 MHz | |

**$t_{SCKH}$** The I2S component generates a 50% duty cycle SCK

**$t_{SCKL}$** The I2S component generates a 50% duty cycle SCLK

**$t_{SCK\_WS}$** The delay between SKC falling edge and WS valid. This value could be calculated as difference between clock to output times for WS and SCK pins. The data could be extracted from the _timing.html as shown below:

**–Clock to output times**

**+Clock to output times from clock BUS_CLK**

**–Clock to output times from clock Clock_1**

| Start | Register | End | Delay (ns) |
|---|---|---|---|
| Clock_1 | \I2S_1:SyncCtl:ControlReg\:controlcell.control_2 | WS(0):iocell.pad_out | 32.907 |
| Clock_1 | \I2S_1:BitCounter\:count7cell.count_5 | WS(0):iocell.pad_out | 31.699 |
| Clock_1 | \I2S_1:Tx:dpTx:u0\:datapathcell.regq_a0 | SDO(0):iocell.pad_out | 27.125 |
| Clock_1 | Net_3:macrocell.mc_q | SCK(0):iocell.pad_out | 24.595 |

$t_{SCK\_SDO}$ The delay between SKC falling edge and WS valid. This value could be calculated as difference between clock to output times for SDO and SCK pins. The values are provided in the STA results clock to output times as shown below:

**–Clock to output times**

**+Clock to output times from clock BUS_CLK**

**–Clock to output times from clock Clock_1**

| Start | Register | End | Delay (ns) |
|---|---|---|---|
| Clock_1 | \I2S_1:SyncCtl:ControlReg\:controlcell.control_2 | WS(0):iocell.pad_out | 32.907 |
| Clock_1 | \I2S_1:BitCounter\:count7cell.count_5 | WS(0):iocell.pad_out | 31.699 |
| Clock_1 | \I2S_1:Tx:dpTx:u0\:datapathcell.regq_a0 | SDO(0):iocell.pad_out | 27.125 |
| Clock_1 | Net_3:macrocell.mc_q | SCK(0):iocell.pad_out | 24.595 |

$t_{S\_SDI}$ SDI setup time is the SDI pin to internal logic path delay of the master component. This is provided in the STA results input setup times as shown below:

**–Setup times**

**–Setup times to clock CLK**

| Start | Register | Clock | Delay (ns) |
|---|---|---|---|
| SDI(0):iocell.pad_in | \I2S_1:Rx:dpRx:u0\:datapathcell.regd_a0 | CLK | 18.965 |

# Component Changes

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

| Version | Description of Changes | Reason for Changes / Impact |
|---|---|---|
| 2.10 | Resampled FIFO block status signals to DP clock. | Allows component to function with the same timing results for all PSoC3 and PSoC5 silicons. |
| | Added characterization data to datasheet | |
| | Minor datasheet edits and updates | |

| | | |
|---|---|---|
| 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_empty 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. |