

**Please note that Cypress is an Infineon Technologies Company.**

The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

**Continuity of document content**

The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

**Continuity of ordering part numbers**

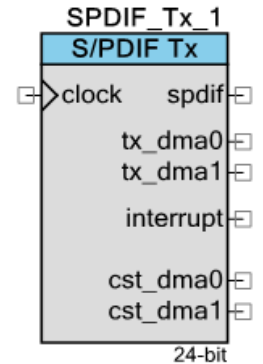
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

# S/PDIF Transmitter (SPDIF\_Tx)

1.20

## Features

- Conforms to IEC-60958, AES/EBU, AES3 standards for Linear PCM Audio Transmission
- Sample rate support for clock/128 (up to 192 kHz)
- Configurable audio sample length (8/16/24)
- Channel status bits generator for consumer applications
- DMA support
- Independent left and right channel FIFOs or interleaved stereo FIFOs



## General Description

The SPDIF\_Tx component provides a simple way to add digital audio output to any design. It formats incoming audio data and metadata to create the S/PDIF bit stream appropriate for optical or coaxial digital audio. The component supports interleaved and separated audio.

The SPDIF\_Tx component receives audio data from DMA as well as channel status information. Most of the time, the channel status DMA will be managed by the component; however, you have the option of specifying this data separately to better control your system.

## When to Use an SPDIF\_Tx

The SPDIF\_Tx component provides a fast solution whenever an S/PDIF transmitter is essential, including applications such as:

- Digital audio players
- Computer audio interfaces
- Audio mastering equipment

The use cases for the component can be:

- When programmed, a PSoC enumerates as a USB Audio HID. The PSoC is the sound card for the computer and plays through a digital audio connection.
- The component can be used in conjunction with an I2S component and external ADC to go from analog audio to digital audio.

## Input/Output Connections

This section describes the various input and output connections for the SPDIF\_Tx 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.

### clock – Input

The clock rate provided must be 128 times the desired sample frequency for the output signal. For example to produce 48-kHz audio, the clock frequency would be:

$$128 \times 48 \text{ kHz} = 6.144 \text{ MHz}$$

### spdif – Output

Serial data output.

### interrupt – Output

Interrupt output.

### tx\_dma0 – Output

DMA request for audio FIFO 0 (Channel 0 or Interleaved).

### tx\_dma1 – Output \*

DMA request for audio FIFO 1 (Channel 1). Displays if you select **Separated** under the **Audio Mode** parameter.

### cst\_dma0– Output \*

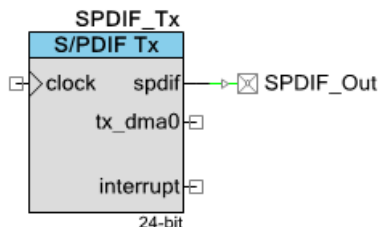
DMA request for channel status FIFO 0 (Channel 0). Displays if you deselect the checkbox under the **Managed DMA** parameter.

### cst\_dma1 – Output \*

DMA request for channel status FIFO 1 (Channel 1). Displays if you deselect the checkbox under the **Managed DMA** parameter.

## Schematic Macro Information

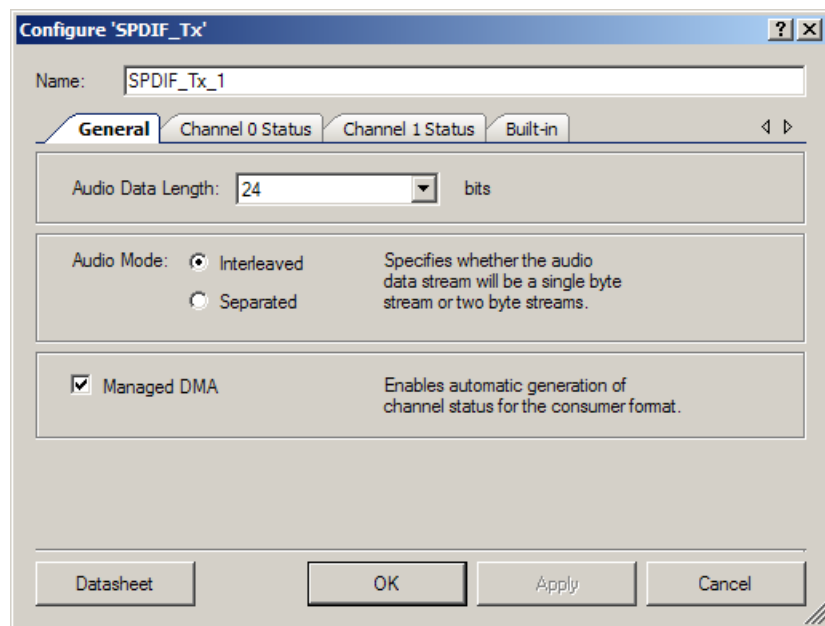
The default SPDIF Transmitter in the Component Catalog is a schematic macro using a SPDIF component with default settings. It is connected to a digital output Pin component. The generation of APIs for the pin is turned off.



## Component Parameters

Drag an SPDIF\_Tx component onto your design and double click it to open the **Configure** dialog. This dialog has three tabs to guide you through the process of setting up the SPDIF\_Tx component.

### General Tab



### Audio Data Length

Determines the number of data bits configured for each sample (hardware compiled). This value can be set to: 8, 16, or 24. The default setting is **24**.

## Audio Mode

Allows you to select whether the audio data is **Interleaved** (default) or **Separated** (hardware compiled).

## Managed DMA

Allows you to select whether the component will manage the channel status DMA (hardware compiled). If **Managed DMA** is selected, the **Channel 0 Status** and **Channel 1 Status** tabs are enabled. The option is enabled by default.

## Channel 0 Status Tab

The screenshot shows the 'Configure SPDIF\_Tx' dialog box with the 'Channel 0 Status' tab selected. The 'Name' field is 'SPDIF\_Tx\_1'. The 'Frequency' dropdown is set to 'Unknown'. The 'Data Type' dropdown is 'Linear PCM', 'Copyright' is 'Audio is copyrighted', 'Pre-emphasis' is 'No Pre-emphasis', 'Category' is 'General', and 'Clock Accuracy' is 'Level II'. The 'Source Number' and 'Channel Number' spinners are both set to 0. At the bottom are buttons for 'Datasheet', 'OK', 'Apply', and 'Cancel'.

## Frequency

Allows you to select the value of channel status for the specified frequency. This value applies to both channels. The source frequency can be: 22 kHz, 24 kHz, 32 kHz, 44 kHz, 48 kHz, 64 kHz, 88 kHz, 96 kHz, 192 kHz, or Unknown. The default is **Unknown**.

## Data Type

Specifies the data type value for channel status 0. This value can be set to **Linear PCM** (default) or **Other Data**.

## Copyright

Allows you to select whether the **Audio is copyrighted** (default) or **Audio is not copyrighted**.

## Pre-emphasis

Determines the PCM pre-emphasis value for channel status 0. This value can be set to **No Pre-emphasis** or **50/15 µs**. The default setting is **No Pre-emphasis**.

## Category

Specifies the category type for channel status 0. This value can be set to **General** (default) or **Digital to Digital**.

## Clock Accuracy

Allows you to select the clock accuracy for channel status 0. This value can be set to **Level I**, **Level II**, or **Level III**. The default setting is **Level II**.

## Source Number

Determines the source number for channel status 0. This value can be set between 0 and 15. The default setting is **0**.

## Channel Number

Determines the channel number for channel status 0. This value can be set between 0 and 15. The default setting is **0**.

## Channel 1 Status Tab

The screenshot shows the 'Configure SPDIF\_Tx' dialog box with the 'Channel 1 Status' tab selected. The 'Name' field is 'SPDIF\_Tx\_1'. The 'Channel 1 Status' tab is active, showing a 'Copy defaults from Channel 0' checkbox which is checked. Below this, there are five dropdown menus: 'Data Type' (Linear PCM), 'Copyright' (Audio is not copyrighted), 'Pre-emphasis' (No Pre-emphasis), 'Category' (General), and 'Clock Accuracy' (Level II). At the bottom, there are two spinners for 'Source Number' and 'Channel Number', both set to 1. At the very bottom are buttons for 'Data Sheet', 'OK', 'Apply', and 'Cancel'.

| Field                        | Value                               |
|------------------------------|-------------------------------------|
| Name                         | SPDIF_Tx_1                          |
| Copy defaults from Channel 0 | <input checked="" type="checkbox"/> |
| Data Type                    | Linear PCM                          |
| Copyright                    | Audio is not copyrighted            |
| Pre-emphasis                 | No Pre-emphasis                     |
| Category                     | General                             |
| Clock Accuracy               | Level II                            |
| Source Number                | 1                                   |
| Channel Number               | 1                                   |

## Copy defaults from Channel 0

Allows you to select whether the channel status for channel 1 is the same as for channel 0. If the checkbox is selected, all of the dropdown boxes will be disabled on the **Channel 1 Status** tab. The setting is checked by default.

The remaining parameters for the **Channel 1 Status** tab are identical to the **Channel 0 Status** tab.

## 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 “SPDIF\_Tx\_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 “SPDIF.”

## Functions

| Function                 | Description                                                      |
|--------------------------|------------------------------------------------------------------|
| SPDIF_Start()            | Starts the S/PDIF interface.                                     |
| SPDIF_Stop()             | Disables the S/PDIF interface.                                   |
| SPDIF_Sleep()            | Saves configuration and disables the S/PDIF interface.           |
| SPDIF_Wakeup()           | Restores configuration of the S/PDIF interface.                  |
| SPDIF_EnableTx()         | Enables the audio data output in the S/PDIF bit stream.          |
| SPDIF_DisableTx()        | Disables the audio output in the S/PDIF bit stream.              |
| SPDIF_WriteTxByte()      | Writes a single byte into the audio FIFO.                        |
| SPDIF_WriteCstByte()     | Writes a single byte into the channel status FIFO.               |
| SPDIF_SetInterruptMode() | Sets the interrupt source for the S/PDIF interrupt.              |
| SPDIF_ReadStatus()       | Returns state in the S/PDIF status register.                     |
| SPDIF_ClearTxFIFO()      | Clears out the audio FIFO.                                       |
| SPDIF_ClearCstFIFO()     | Clears out the channel status FIFOs.                             |
| SPDIF_SetChannelStatus() | Sets the values of the channel status at run time.               |
| SPDIF_SetFrequency()     | Sets the values of the channel status for a specified frequency. |
| SPDIF_Init()             | Initializes or restores default S/PDIF configuration.            |

| Function                              | Description                                 |
|---------------------------------------|---------------------------------------------|
| <a href="#">SPDIF_Enable()</a>        | Enables the S/PDIF interface.               |
| <a href="#">SPDIF_SaveConfig()</a>    | Saves configuration of S/PDIF interface.    |
| <a href="#">SPDIF_RestoreConfig()</a> | Restores configuration of S/PDIF interface. |

### void SPDIF\_Start(void)

**Description:** Starts the S/PDIF interface. Starts the channel status DMA if the component is configured to handle the channel status DMA. Enables Active mode power template bits or clock gating as appropriate. Starts the generation of the S/PDIF output with channel status, but the audio data is set to all 0s. This allows the S/PDIF receiver to lock on to the component's clock.

**Parameters:** None

**Return Value:** None

**Side Effects:** None

### void SPDIF\_Stop(void)

**Description:** Disables the S/PDIF interface. Disables Active mode power template bits or clock gating as appropriate. The S/PDIF output is set to 0. The audio data and channel data FIFOs are cleared. The SPDIF\_Stop() function calls SPDIF\_DisableTx() and stops the managed channel status DMA.

**Parameters:** None

**Return Value:** None

**Side Effects:** None

### void SPDIF\_Sleep(void)

**Description:** This is the preferred routine to prepare the component for sleep. The SPDIF\_Sleep() routine saves the current component state. Then it calls the SPDIF\_Stop() function and calls SPDIF\_SaveConfig() to save the hardware configuration. Disables Active mode power template bits or clock gating as appropriate. The spdif output is set to 0.

Call the SPDIF\_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 SPDIF\_Wakeup(void)**

- Description:** Restores SPDIF configuration and nonretention register values. The component is stopped regardless of its state before sleep. The SPDIF\_Start() function must be called explicitly to start the component again.
- Parameters:** None
- Return Value:** None
- Side Effects:** Calling the SPDIF\_Wakeup() function without first calling the SPDIF\_Sleep() or SPDIF\_SaveConfig() function may produce unexpected behavior.

**void SPDIF\_EnableTx(void)**

- Description:** Enables the audio data output in the S/PDIF bit stream. Transmission will begin at the next X or Z frame.
- Parameters:** None
- Return Value:** None
- Side Effects:** None

**void SPDIF\_DisableTx(void)**

- Description:** Disables the audio output in the S/PDIF bit stream. Transmission of data will stop at the next rising edge of clock and constant 0 value will be transmitted.
- Parameters:** None
- Return Value:** None
- Side Effects:** None

**void SPDIF\_WriteTxByte(uint8 wrData, uint8 channelSelect)**

- Description:** Writes a single byte into the audio data FIFO. The component status should be checked before this call to confirm that the audio data FIFO is not full.
- Parameters:** uint8 wrData: Byte containing the audio data to transmit.  
uint8 channelSelect: Byte containing the constant for Channel to write. See channel status macros below. In the interleaved mode this parameter is ignored
- Return Value:** None
- Side Effects:** None

**void SPDIF\_WriteCstByte(uint8 wrData, uint8 channelSelect)**

- Description:** Writes a single byte into the specified channel status FIFO. The component status should be checked before this call to confirm that the channel status FIFO is not full.
- Parameters:** uint8 wrData: Byte containing the status data to transmit.  
uint8 channelSelect: Byte containing the constant for Channel to write. See channel status macros below.
- Return Value:** None
- Side Effects:** None

**void SPDIF\_SetInterruptMode(uint8 interruptSource)**

- Description:** Sets the interrupt source for the S/PDIF interrupt. Multiple sources may be ORed.
- Parameters:** uint8 byte containing the constant for the selected interrupt sources.

| SPDIF Tx Interrupt Source | Value | Type          |
|---------------------------|-------|---------------|
| AUDIO_FIFO_UNDERFLOW      | 0x01  | Clear on Read |
| AUDIO_0_FIFO_NOT_FULL     | 0x02  | Transparent   |
| AUDIO_1_FIFO_NOT_FULL     | 0x04  | Transparent   |
| CHST_FIFO_UNDERFLOW       | 0x08  | Clear on Read |
| CHST_0_FIFO_NOT_FULL      | 0x10  | Transparent   |
| CHST_1_FIFO_NOT_FULL      | 0x20  | Transparent   |

- Return Value:** None
- Side Effects:** If clear on read bits are used as the sources for the interrupt generation, the interrupt output remains asserted until the SPDIF status register is read.

**uint8 SPDIF\_ReadStatus(void)****Description:** Returns state of the SPDIF status register.**Parameters:** None**Return Value:** uint8 state of the SPDIF status register.

| SPDIF Status Masks    | Value | Type          |
|-----------------------|-------|---------------|
| AUDIO_FIFO_UNDERFLOW  | 0x01  | Clear on Read |
| AUDIO_0_FIFO_NOT_FULL | 0x02  | Transparent   |
| AUDIO_1_FIFO_NOT_FULL | 0x04  | Transparent   |
| CHST_FIFO_UNDERFLOW   | 0x08  | Clear on Read |
| CHST_0_FIFO_NOT_FULL  | 0x10  | Transparent   |
| CHST_1_FIFO_NOT_FULL  | 0x20  | Transparent   |

**Side Effects:** Clears the bits of the SPDIF status register that are clear on read.**void SPDIF\_ClearTxFIFO(void)****Description:** Clears out the audio data FIFO. Any data present in the FIFO will be lost. In the case of separated audio mode, both audio FIFOs will be cleared. Call this function only when transmit is disabled.**Parameters:** None**Return Value:** None**Side Effects:** None**void SPDIF\_ClearCstFIFO(void)****Description:** Clears out the channel status FIFOs. Any data present in either FIFO will be lost. Call this function only when the component is stopped.**Parameters:** None**Return Value:** None**Side Effects:** None

**void SPDIF\_SetChannelStatus(uint8 channel, uint8 byte, uint8 mask, uint8 value)**

**Description:** Sets the values of the channel status at run time. This API is only valid when the component is managing the DMA.

**Parameters:** uint8 channel: Byte containing the constant to specify the channel to modify. See channel status macros below.

uint8 byte: Byte to modify [0..23]. See channel status macros below.

uint8 mask: Mask on the byte. See channel status macros below.

uint8 value: Value to set. See channel status macros below.

**Return Value:** None

**Side Effects:** None

**uint8 SPDIF\_SetFrequency(uint8 frequency)**

**Description:** Sets the values of the channel status for a specified frequency and returns 1. This function only works if the component is stopped. If this is called while the component is started, a zero will be returned and the values will not be modified. This API is only valid when the component is managing the DMA.

**Parameters:** uint8: byte containing the constant for the specified frequency.

| Name              | Description                         |
|-------------------|-------------------------------------|
| SPDIF_SPS_22KHZ   | Clock rate is set for 22-kHz audio  |
| SPDIF_SPS_44KHZ   | Clock rate is set for 44-kHz audio  |
| SPDIF_SPS_88KHZ   | Clock rate is set for 88-kHz audio  |
| SPDIF_SPS_24KHZ   | Clock rate is set for 24-kHz audio  |
| SPDIF_SPS_48KHZ   | Clock rate is set for 48-kHz audio  |
| SPDIF_SPS_96KHZ   | Clock rate is set for 96-kHz audio  |
| SPDIF_SPS_32KHZ   | Clock rate is set for 32-kHz audio  |
| SPDIF_SPS_64KHZ   | Clock rate is set for 64-kHz audio  |
| SPDIF_SPS_192KHZ  | Clock rate is set for 192-kHz audio |
| SPDIF_SPS_UNKNOWN | Clock rate is not specified         |

**Return Value:** uint8: 1 on success  
0 on failure

**Side Effects:** None

**void SPDIF\_Init(void)**

- Description:** Initializes or restores default S/PDIF configuration provided with customizer that defines interrupt sources for the component and channel status if the component is configured to handle the channel status DMA.
- Parameters:** None
- Return Value:** None
- Side Effects:** Restores only mask registers for interrupt generation and channel status if the component is configured to handle the channel status DMA. It will not clear data from the FIFOs and will not reset component hardware state machines.

**void SPDIF\_Enable(void)**

- Description:** Activates the hardware and begins component operation. It is not necessary to call SPDIF\_Enable() because the SPDIF\_Start() routine calls this function, which is the preferred method to begin component operation.
- Parameters:** None
- Return Value:** None
- Side Effects:** None

**void SPDIF\_SaveConfig(void)**

- Description:** This function saves the component configuration. This will save nonretention 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 SPDIF\_Sleep() function.
- Parameters:** None
- Return Value:** None
- Side Effects:** None

**void SPDIF\_RestoreConfig(void)**

- Description:** This function restores the component configuration. This will restore nonretention registers. This function will also restore the component parameter values to what they were prior to calling the SPDIF\_Sleep() function. This routine is called by SPDIF\_Wakeup() to restore component when it exits sleep.
- Parameters:** None
- Return Value:** None
- Side Effects:** Must be called only after SPDIF\_SaveConfig() routine. Otherwise the component configuration will be overwritten with its initial setting.

## Global Variables

| Variable                                        | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SPDIF_initVar                                   | SPDIF_initVar Indicates whether the S/PDIF component has been initialized. The variable is initialized to 0 and set to 1 the first time SPDIF_Start() is called. This allows the component to restart without reinitialization after the first call to the SPDIF_Start() routine.<br>If reinitialization of the component is required, then the SPDIF_Init() function can be called before the SPDIF_Start() or SPDIF_Enable() function.                                                                                                                                                                                                                                           |
| SPDIF_wrkCstStream0[],<br>SPDIF_wrkCstStream1[] | Channel Status arrays for Channel 0 and Channel 1 respectively. These arrays are conditionally compiled and are presented to the component only if the component is managing channel status DMA.<br>The Channel Status streams are stored in two different buffers large enough to store the streams for a complete SPDIF block (2x24 bytes).<br>Note: To set the values of the channel status it is strictly recommended to use the SPDIF_SetChannelStatus() function along with the channel status macros.<br>For example to set the category field of channel 0 status to general use the following function call :<br>SPDIF_SetChannelStatus(SPDIFF_CHANNEL_0, SPDIF_CAT_GEN); |

## Channel Status Macros

The channel status macros encapsulate the mask and value for a specific channel status setting. These macros along with channel macros are used to set the values of channel status at run time by the SPDIF\_SetChannelStatus() API.

The recommended best practice is to set both channels at the same time. The following is an example:

```
SPDIF_SetChannelStatus(SPDIFF_CHANNEL_0, SPDIF_DATA_TYPE_LINEAR_PCM);
```

## Channel Name Constants

| Name             | Description |
|------------------|-------------|
| SPDIFF_CHANNEL_0 | Channel 0   |
| SPDIFF_CHANNEL_1 | Channel 1   |

## Channel Status Constants

| Name                        | Byte | Mask | Value |
|-----------------------------|------|------|-------|
| SPDIFF_DATA_TYPE_LINEAR_PCM | 0    | 0x02 | 0x00  |
| SPDIFF_DATA_TYPE_OTHERDATA  | 0    | 0x02 | 0x02  |
| SPDIFF_COPY_HAS_CP_RIGHT    | 0    | 0x04 | 0x00  |



| Name                   | Byte | Mask | Value |
|------------------------|------|------|-------|
| SPDIF_COPY_NO_CP_RIGHT | 0    | 0x04 | 0x04  |
| SPDIF_PREEMP_NO_PREEMP | 0    | 0x38 | 0x00  |
| SPDIF_PREEMP_PREEMP50  | 0    | 0x38 | 0x08  |
| SPDIF_CAT_GEN          | 1    | 0xFF | 0x00  |
| SPDIF_CAT_D2D          | 1    | 0xFF | 0x02  |
| SPDIF_SRC_NUM00        | 2    | 0x0F | 0x00  |
| SPDIF_SRC_NUM01        | 2    | 0x0F | 0x01  |
| SPDIF_SRC_NUM02        | 2    | 0x0F | 0x02  |
| SPDIF_SRC_NUM03        | 2    | 0x0F | 0x03  |
| SPDIF_SRC_NUM04        | 2    | 0x0F | 0x04  |
| SPDIF_SRC_NUM05        | 2    | 0x0F | 0x05  |
| SPDIF_SRC_NUM06        | 2    | 0x0F | 0x06  |
| SPDIF_SRC_NUM07        | 2    | 0x0F | 0x07  |
| SPDIF_SRC_NUM08        | 2    | 0x0F | 0x08  |
| SPDIF_SRC_NUM09        | 2    | 0x0F | 0x09  |
| SPDIF_SRC_NUM10        | 2    | 0x0F | 0x0A  |
| SPDIF_SRC_NUM11        | 2    | 0x0F | 0x0B  |
| SPDIF_SRC_NUM12        | 2    | 0x0F | 0x0C  |
| SPDIF_SRC_NUM13        | 2    | 0x0F | 0x0D  |
| SPDIF_SRC_NUM14        | 2    | 0x0F | 0x0E  |
| SPDIF_SRC_NUM15        | 2    | 0x0F | 0x0F  |
| SPDIF_CH_NUM00         | 2    | 0xF0 | 0x00  |
| SPDIF_CH_NUM01         | 2    | 0xF0 | 0x10  |
| SPDIF_CH_NUM02         | 2    | 0xF0 | 0x20  |
| SPDIF_CH_NUM03         | 2    | 0xF0 | 0x30  |
| SPDIF_CH_NUM04         | 2    | 0xF0 | 0x40  |
| SPDIF_CH_NUM05         | 2    | 0xF0 | 0x50  |
| SPDIF_CH_NUM06         | 2    | 0xF0 | 0x60  |
| SPDIF_CH_NUM07         | 2    | 0xF0 | 0x70  |
| SPDIF_CH_NUM08         | 2    | 0xF0 | 0x80  |
| SPDIF_CH_NUM09         | 2    | 0xF0 | 0x90  |

| Name           | Byte | Mask | Value |
|----------------|------|------|-------|
| SPDIF_CH_NUM10 | 2    | 0xF0 | 0xA0  |
| SPDIF_CH_NUM11 | 2    | 0xF0 | 0xB0  |
| SPDIF_CH_NUM12 | 2    | 0xF0 | 0xC0  |
| SPDIF_CH_NUM13 | 2    | 0xF0 | 0xD0  |
| SPDIF_CH_NUM14 | 2    | 0xF0 | 0xE0  |
| SPDIF_CH_NUM15 | 2    | 0xF0 | 0xF0  |
| SPDIF_CLKLVL_1 | 3    | 0x30 | 0x10  |
| SPDIF_CLKLVL_2 | 3    | 0x30 | 0x00  |
| SPDIF_CLKLVL_3 | 3    | 0x30 | 0x20  |
| SPDIF_STDLN    | 4    | 0x0F | 0x00  |
| SPDIF_24BLEN   | 4    | 0x0F | 0x0B  |

## Macro Callbacks

Macro callbacks allow users to execute code from the API files that are automatically generated by PSoC Creator. Refer to the PSoC Creator Help and *Component Author Guide* for the more details.

In order to add code to the macro callback present in the component's generated source files, perform the following:

- Define a macro to signal the presence of a callback (in *cyapicallbacks.h*). This will “uncomment” the function call from the component's source code.
- Write the function declaration (in *cyapicallbacks.h*). This will make this function visible by all the project files.
- Write the function implementation (in any user file).

| Callback Function <sup>[1]</sup> | Associated Macro                | Description                                                                                                         |
|----------------------------------|---------------------------------|---------------------------------------------------------------------------------------------------------------------|
| SPDIF_Cst0Copy_EntryCallback     | SPDIF_CST0_COPY_ENTR_Y_CALLBACK | Used at the beginning of the SPDIF_Cst0Copy() interrupt handler to perform additional application-specific actions. |
| SPDIF_Cst0Copy_ExitCallback      | SPDIF_CST0_COPY_EXIT_CALLBACK   | Used at the end of the SPDIF_Cst0Copy() interrupt handler to perform additional application-specific actions.       |

<sup>1</sup> The callback function name is formed by component function name optionally appended by short explanation and “Callback” suffix.



| Callback Function <sup>[1]</sup> | Associated Macro                | Description                                                                                                         |
|----------------------------------|---------------------------------|---------------------------------------------------------------------------------------------------------------------|
| SPDIF_Cst1Copy_EntryCallback     | SPDIF_CST1_COPY_ENTR_Y_CALLBACK | Used at the beginning of the SPDIF_Cst1Copy() interrupt handler to perform additional application-specific actions. |
| SPDIF_Cst1Copy_ExitCallback      | SPDIF_CST1_COPY_EXIT_CALLBACK   | Used at the end of the SPDIF_Cst1Copy() interrupt handler to perform additional application-specific actions.       |

## 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 SPDIF\_Tx component has the following specific deviations:

| MISRA-C:2004 Rule | Rule Class (Required/Advisory) | Rule Description                                                                                                                                                                                                                                                                                                                                                                                                                                 | Description of Deviation(s)                                                                                                                                                                                                                                                                                                                              |
|-------------------|--------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 10.1              | R                              | <p>The value of an expression of integer type shall not be implicitly converted to a different underlying type if:</p> <ul style="list-style-type: none"> <li>a) it is not a conversion to a wider integer type of the same signedness, or</li> <li>b) the expression is complex, or</li> <li>c) the expression is not constant and is a function argument, or</li> <li>d) the expression is not constant and is a return expression.</li> </ul> | <p>The library function memcpy has generic int argument for number of bytes to be copied.</p> <p>The unsigned 8-bits integer is passed as an argument to this function.</p> <p>This action does not cause any side effects because number of bytes to copy always less than 256.</p> <p>The DMA component provides general integer type definitions.</p> |

| MISRA-C:2004 Rule | Rule Class (Required/Advisory) | Rule Description                                                                                                     | Description of Deviation(s)                                                                                                                                                                                                                                                                         |
|-------------------|--------------------------------|----------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 11.5              | R                              | A cast shall not be performed that removes any const or volatile qualification from the type addressed by a pointer. | The library function memcpy has pointer to void arguments for source and destination.<br><br>The pointers to volatile array are passed as the source and destination arguments and lost qualification.<br><br>The memcpy function implementation ensure correct operation and it is safe to use it. |
| 17.4              | R                              | Array indexing shall be the only allowed form of pointer arithmetic.                                                 | To access DMA registers the pointer arithmetic with array indexing is used.<br><br>This type of access is provided by DMA component.                                                                                                                                                                |

This component has the following embedded components: DMA. Refer to the corresponding component datasheet for information on their MISRA compliance and specific deviations.

## 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 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   | PSoC 3 (Keil_PK51) |            | PSoC 5LP (GCC) |            |
|-----------------|--------------------|------------|----------------|------------|
|                 | Flash Bytes        | SRAM Bytes | Flash Bytes    | SRAM Bytes |
| Managed DMA     | 1507               | 107        | 1402           | 111        |
| Not Managed DMA | 276                | 1          | 430            | 1          |

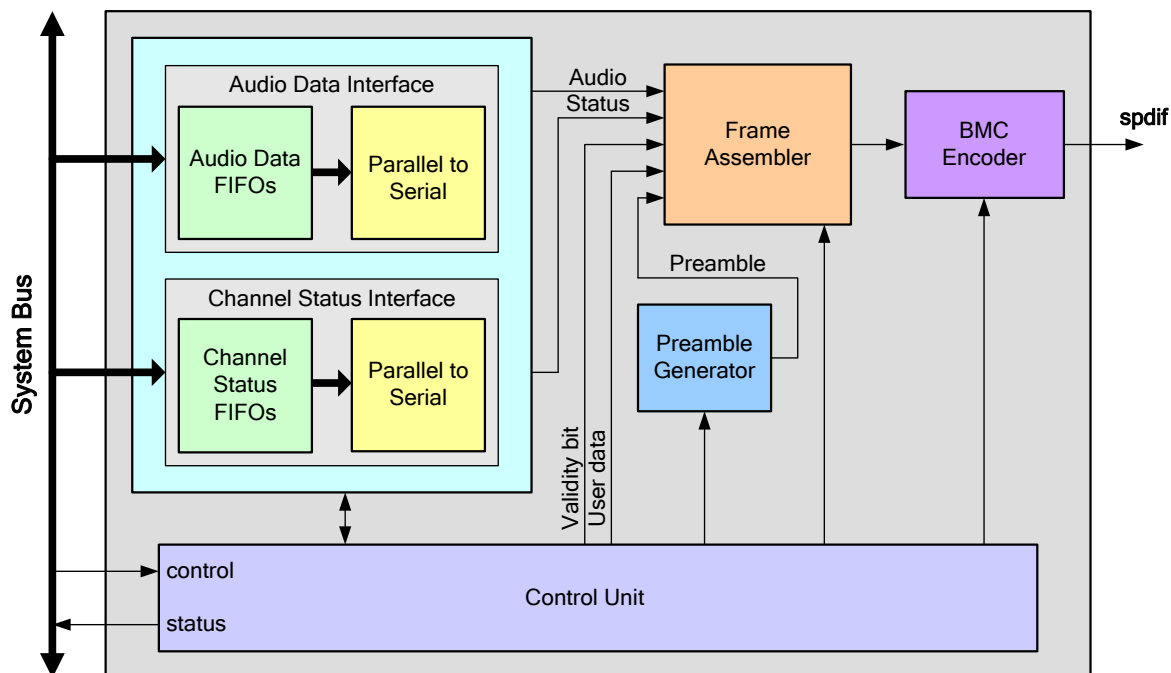
## Functional Description

This component formats incoming audio data and metadata to create the S/PDIF bit stream. The component receives audio data from DMA as well as channel status information. Most of the time, the channel status DMA is managed by the component; however, you have the option of specifying this data separately to better control your system.



## Block Diagram and Configuration

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



The incoming audio data is received through the system bus interface and can be provided through CPU or DMA. The data is byte wide with the least significant byte first and is stored in audio buffer (one or two FIFOs, depending on the component configuration).

The Channel Status stream has its own dedicated interface. Similarly to the audio data, there are two Channel Status FIFOs. Again, the channel status is byte wide data, least significant byte first. One byte is consumed from these FIFOs every eight samples. Both audio and status data are converted from parallel to serial form.

The User Data are not defined in the S/PDIF standard and may be ignored by some receivers, so they are sent as constant zero.

The validity bit, when low, indicates the audio sample is fit for conversion to analog. This bit is sent as constant zero.

The preamble patterns are generated in the Preamble Generator block and are transmitted in serial form.

This is all of the data required to form the SPDIF subframe structure, except the parity bit. The parity bit is calculated in the Frame Assembler block during assembling all the inputs to subframe structure.

The output of the Frame Assembler block goes to BMC Encoder where the data is encoded in spdif output.

The Control Unit block gets the control data from the System Bus interface and returns the status of component operation to the bus. It controls all other blocks during data transmission.

## Data Stream Format

The audio and channel status data are independent byte streams. The byte streams are packed with the least significant byte and bit first. The number of bytes used for each sample is the minimum number of bytes to hold a sample. Any unused bits will be padded with 0 starting at the left-most bit.

The audio data stream 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. The status byte stream is always two byte streams.

## Error Handling

There are two error conditions for the component that can happen if the audio is emptied and a subsequent read occurs (transmit underflow) or the channel status FIFO is emptied and subsequent read occurs (status underflow).

If transmit underflow occurs, the component forces the constant transmission of 0s for audio data and continue correct generation of all framing and status data. Before transmission begins again, transmission must be disabled, the FIFOs should be cleared, data for transmit must be buffered, and then transmission re-enabled. This underflow condition can be monitored by the CPU using the component status bit `AUDIO_FIFO_UNDERFLOW`. An interrupt can also be configured for this error condition.

While the component is started, if the status underflow occurs, the component will send all 0s for channel status with the correct generation of X, Y, Z framing and correct parity. The audio data is continuous, not impacted. To correct channel status data transmission, the component must be stopped and restarted again. This underflow condition can be monitored by the CPU using the status bit `CHST_FIFO_UNDERFLOW`. An interrupt can also be configured for this error condition. If the component doesn't manage DMA, the status data must be buffered before restarting the component.

## Enabling

Audio data transmission has dedicated enabling. When the component is started but not enabled, the S/PDIF output with channel status is generated, but the audio data is set to all 0s. This allows the S/PDIF receiver to lock on the component clock. The transition into the enabled state occurs at X or Z frame.

## Clock Selection

There is no internal clock in this component. You must attach a clock source. The clock rate provided must be 128 times the desired sample frequency for the output signal. For example to produce 48-kHz audio, the clock frequency would be:

$$128 \times 48 \text{ kHz} = 6.144 \text{ MHz}$$

## DMA Support

The S/SPDIF 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 audio data or channel status FIFOs. Typically, the Channel Status DMA is done completely in the component using two channel status arrays and can be modified using macros; however, you can provide the data using your own DMA or the CPU to allow flexibility.

The S/SPDIF can drive up to four DMA components depending on the component configuration. 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                                           |
|----------------------------------------------------|-------------|--------------------|------------------|-------------------------------------------------------|
| SPDIF_TX_FIFO_0_PTR                                | Destination | tx_dma0            | Level            | Transmit FIFO for Channel 0 or Interleaved audio data |
| SPDIF_TX_FIFO_1_PTR                                | Destination | tx_dma1            | Level            | Transmit FIFO for Channel 1 audio data                |
| SPDIF_CST_FIFO_0_PTR                               | Destination | cst_dma0           | Level            | Transmit FIFO for Channel 0 channel status data       |
| SPDIF_CST_FIFO_1_PTR                               | Destination | cst_dma1           | Level            | Transmit FIFO for Channel 1 channel status data       |

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

## Registers

### SPDIF\_Tx\_CONTROL\_REG

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

- enable: Enable/disable SPDIF\_Tx component. When not enabled the component is in reset state.
- txenable: Enable/disable audio data output in the S/SPDIF bit stream.

**SPDIF\_Tx\_STATUS\_REG**

| Bits  | 7        | 6 | 5                   | 4                   | 3                   | 2                    | 1                    | 0                    |
|-------|----------|---|---------------------|---------------------|---------------------|----------------------|----------------------|----------------------|
| Value | reserved |   | chst1_fifo_not_full | chst0_fifo_not_full | chst_fifo_underflow | audio1_fifo_not_full | audio0_fifo_not_full | audio_fifo_underflow |

- chst1\_fifo\_not\_full: If set channel status FIFO 1 is not full
- chst0\_fifo\_not\_full: If set channel status FIFO 0 is not full
- chst\_fifo\_underflow: If set channel status FIFOs underflow event has occurred
- audio1\_fifo\_not\_full: If set audio data FIFO 1 is not full
- audio0\_fifo\_not\_full: If set audio data FIFO 0 is not full
- audio\_fifo\_underflow: If set audio data FIFOs underflow event has occurred

The register value may be read with the SPDIF\_Tx\_ReadStatus() API function.

**Note** Bit 3 and bit 0 of the status register are configured as clear on read. In this mode, the input status is sampled each cycle of the status register clock. When the input goes high, the register bit is set and stays set regardless of the subsequent state of the input. The register bit is cleared on a subsequent read by the CPU.

**Resources**

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

| Configuration   | Resource Type  |            |              |               |              |            |
|-----------------|----------------|------------|--------------|---------------|--------------|------------|
|                 | Datapath Cells | Macrocells | Status Cells | Control Cells | DMA Channels | Interrupts |
| Managed DMA     | 4              | 23         | 1            | 2             | 2            | 2          |
| Not Managed DMA | 4              | 23         | 1            | 2             | —            | —          |

## DC and AC Electrical Characteristics

Specifications are valid for  $-40\text{ }^{\circ}\text{C} \leq T_A \leq 85\text{ }^{\circ}\text{C}$  and  $T_J \leq 100\text{ }^{\circ}\text{C}$ , except where noted.  
Specifications are valid for 1.71 V to 5.5 V, except where noted.

### DC Characteristics

| Parameter               | Description                                     | Min | Typ <sup>[2]</sup> | Max | Units <sup>[3]</sup> |
|-------------------------|-------------------------------------------------|-----|--------------------|-----|----------------------|
| I <sub>DD(DMA)</sub>    | Component current consumption (Managed DMA)     |     |                    |     |                      |
|                         | Idle current <sup>[4]</sup>                     | –   | 34                 | –   | μA/MHz               |
|                         | Operating current <sup>[5]</sup>                | –   | 39                 | –   | μA/MHz               |
| I <sub>DD(NO_DMA)</sub> | Component current consumption (Not Managed DMA) |     |                    |     |                      |
|                         | Idle current <sup>[4]</sup>                     | –   | 30                 | –   | μA/MHz               |
|                         | Operating current <sup>[5]</sup>                | –   | 35                 | –   | μA/MHz               |

### AC Characteristics

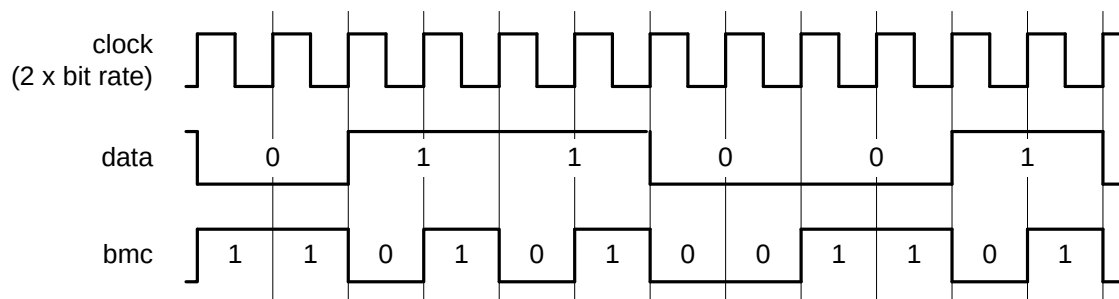
| Parameter          | Description               | Min | Typ                  | Max | Unit |
|--------------------|---------------------------|-----|----------------------|-----|------|
| f <sub>S</sub>     | Sampling frequency        | –   | –                    | 192 | kHz  |
| f <sub>CLOCK</sub> | Component clock frequency | –   | 128 × f <sub>S</sub> | –   | MHz  |

### S/PDIF Channel Encoding

S/PDIF is a single-wire serial interface. The bit clock is embedded within the S/PDIF data stream. The digital signal is coded using Biphase Mark Code (BMC), which is a kind of phase modulation. The frequency of the clock is twice the bit-rate. Every bit of the original data is represented as two logical states, which, together, form a cell. The logical level at the start of a bit is always inverted to the level at the end of the previous bit. To transmit a '1' in this format,

- 
2. Device IO and clock distribution current not included. The values are at 25 °C.
  3. Current consumption is specified with respect to the incoming component clock.
  4. Current consumed by component while it is enabled but not transmitting data.
  5. Current consumed by component while it is enabled and transmitting data.

there is a transition in the middle of the data bit boundary. If there is no transition in the middle, the data is considered a '0'.



## S/PDIF Protocol Hierarchy

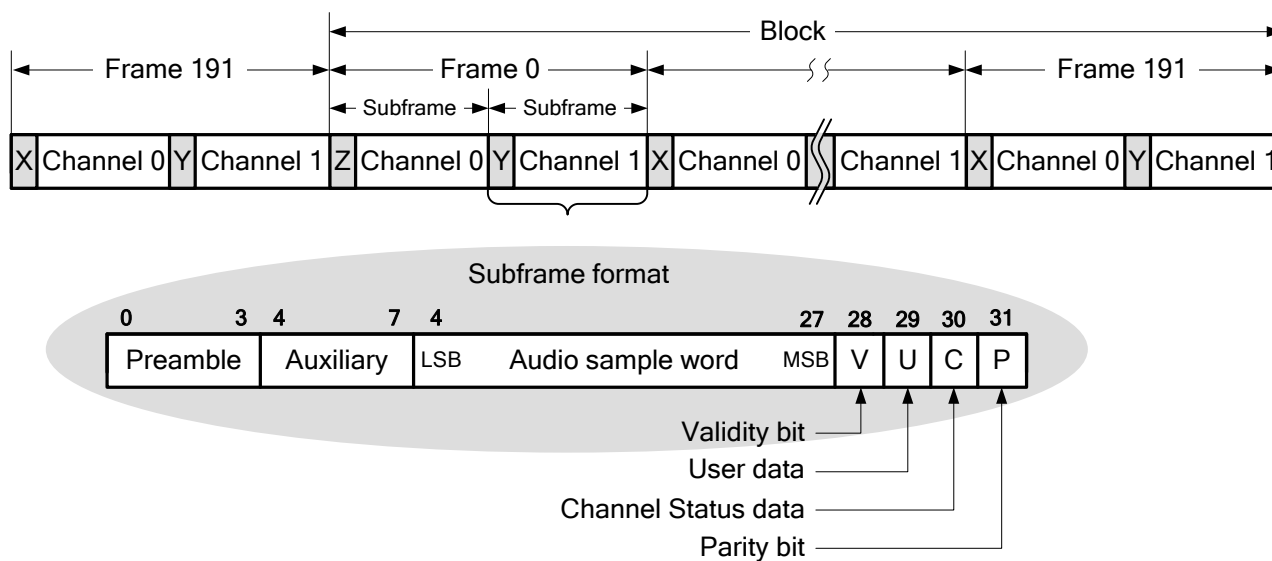
The SPDIF signal format is shown in the figure below. Audio data is transmitted in sequential blocks. The block contains 192 frames. Each frame consists of two subframes that are the basic unit into which digital audio data is organized.

A subframe contains:

- A preamble pattern
- A single audio sample that may be up to 24 bits wide
- A validity bit that indicates whether the sample is valid
- A bit containing user data
- A bit containing the channel status
- An even parity bit for this subframe

There are three types of preambles: X, Y and Z. Preamble Z indicates the start of a block and the start of subframe channel 0. Preamble X indicates the start of a channel 0 subframe when not at the start of a block. Preamble Y always indicates the start of a channel 1 subframe.





## Component Changes

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

| Version | Description of Changes                                                                          | Reason for Changes / Impact                                                                                                                                                                                                                                                                             |
|---------|-------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1.20.d  | Minor datasheet edits.                                                                          |                                                                                                                                                                                                                                                                                                         |
| 1.20.c  | Minor datasheet edits.                                                                          |                                                                                                                                                                                                                                                                                                         |
| 1.20.b  | Datasheet update.                                                                               | Added Macro Callbacks section.                                                                                                                                                                                                                                                                          |
| 1.20.a  | Datasheet edits and corrections.                                                                | Minor corrections and clarifications.                                                                                                                                                                                                                                                                   |
| 1.20    | Added MISRA Compliance section.                                                                 | The component has specific deviations described.                                                                                                                                                                                                                                                        |
| 1.10    | 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.                                                  |                                                                                                                                                                                                                                                                                                         |

© Cypress Semiconductor Corporation, 2012-2017. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit [cypress.com](http://cypress.com). Other names and brands may be claimed as property of their respective owners.

