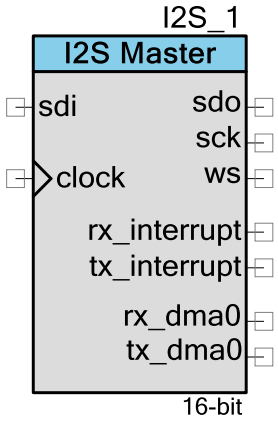
Inter-IC Sound Bus (I2S)

2.0

Features



- Master only
- 8 - 32 data bits per sample
- 16-, 32-, 48-, or 64-bit word select period
- Data rate up to 192 KHz with 64-bit word select period: 12.288 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 × 48 KHz × 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.

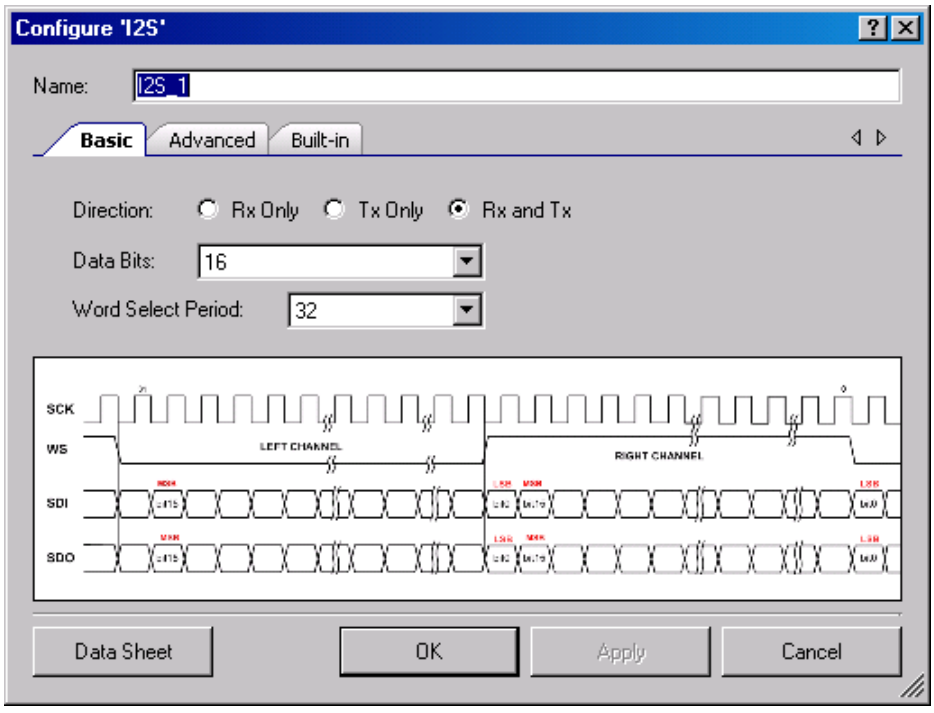
tx_DMA1 – Output *

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).

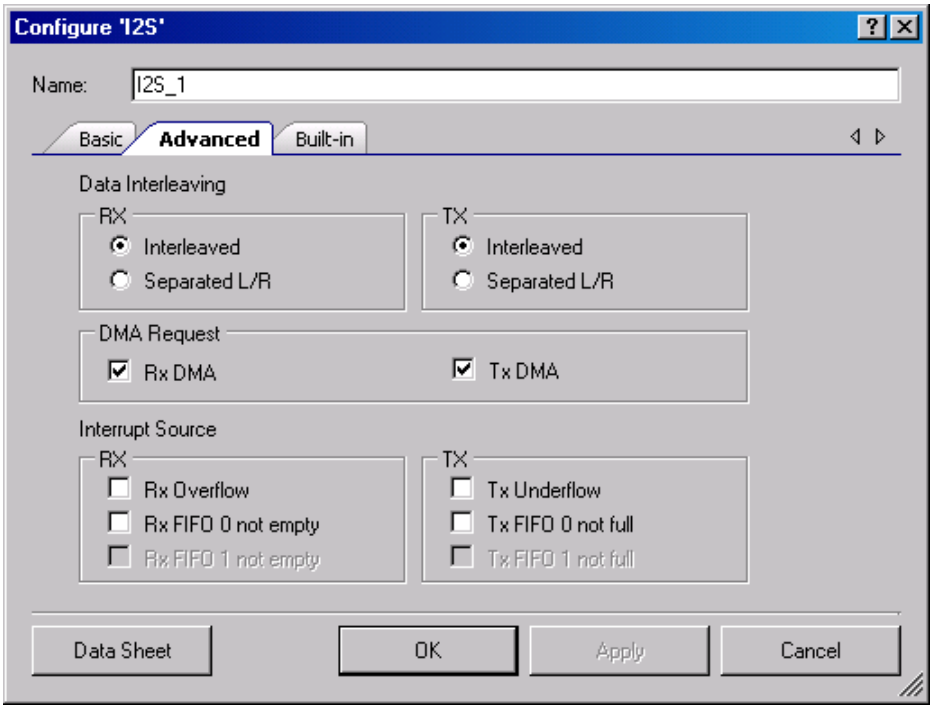
Data Bits

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

Direction	Digital Blocks					API Memory (Bytes)		Pins (per External I/O)
	Datapaths	Macro cells	Status Registers	Control Registers	Counter7	Flash	RAM	
Rx Direction	1	8	1	1	1	TBD	TBD	3
Tx Direction	1	7	1	1	1	TBD	TBD	3
Rx and Tx	2	14	2	1	1	TBD	TBD	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_Init(void)	Enables the I2S interface
void I2S_Enable(void)	Inits/Restores default I2S configuration
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

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. 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_Init(void)

Description: Inits/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: Enables the I2S interface.

Parameters: None

Return Value: None

Side Effects: None

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: Saves I2S configuration and non-retention register values. 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.

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: None

void I2S_SaveConfig(void)

Description: Saves the user configuration of I2S non-retention registers. Called by I2S_Sleep routines to save the component configuration before entering sleep.

Parameters: None

Return Value: None

Side Effects: None

void I2S_RestoreConfig(void)

Description: Restores the user configuration of I2S non-retention registers. This routines is called by I2S_Wakeup() to restore component when exit 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.

Sample Firmware Source Code

The following is a C language example demonstrating the basic functionality of the I2S component. This example assumes the component has been placed in a design with the default name "I2S_1."

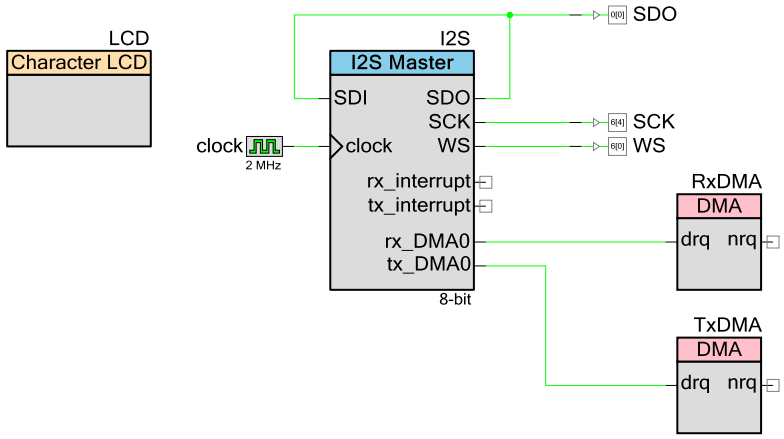
Note If you rename your component you must also edit the example code as appropriate to match the component name you specify.

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.

As shown in the following schematic, two DMA components named "TxDMA" and "RxDMA" have been placed and connected to the tx_DMA0 and rx_DMA0 terminals of the I2S component. The I2S component has been configured with the DMA Request enabled for both the Tx and Rx directions.

Note The DMA request signals from the I2S component are level (instead of edge) signals, so the DMA must be configured appropriately.

Finally, there is a Character LCD component named "LCD" that is used to display the results.




```
#include <device.h>

void Dma_Rx_Configuration(void);
void Dma_Tx_Configuration(void);

#define TD_SIZE   0x08U

/* DMA channels and Transaction Descriptors */
uint8 RxChannel;
uint8 TxChannel;
uint8 RxTD;
uint8 TxTD;

uint8 RxBuff[8];     /* IN buffer */

/* Data to be transmitted by Tx */
```

```
uint8 test_data[8] = {0x00,0x11,0x22,0x33,0x44,0x55,0x66,0x77};

void main()
{
    uint8 i;

    /* Clear IN buffer */
    for(i=0; i<8; i++)
    {
        RxBuff[i] = 0;
    }

    LCD_Start();
    LCD_PrintString("WS=16 DATA=8");

    /* Enable I2S component */
    I2S_1_Start();

    /* Configure DMAs for each direction */
    Dma_Rx_Configuration();
    Dma_Tx_Configuration();

    LCD_Position(1,0);

    /* Through the APIs the Tx and Rx directions will be started independently. */
    /* Depending on exactly when the Rx and Tx get enabled relative to each     */
    /* other the bytes in RxBuff will all be present and in order, but they may */
    /* start with any of the values (cyclic shifted testData).                  */
    I2S_1_EnableRx();   /* Enable Rx direction */
    I2S_1_EnableTx();   /* Enable Tx direction */

    CyDelay(200);
    for(i=0; i<8; i++)
    {
        LCD_PrintHexUint8(RxBuff[i]);
    }
    for(;;);
}

void Dma_Rx_Configuration()
{
    uint32 SrcAddr, DstAddr;
    uint8 val;

    SrcAddr = I2S_1_Rx_dpRx_u0__F0_REG;
    DstAddr = ((uint32) RxBuff);
#if (defined(__C51__))
    /* In the case of the PSoC 3 8051 processor the upper bytes of an SRAM  */
    /* pointer is a type field that does not apply for DMA operations.      */
    DstAddr &= 0xFFFF;
#endif

    /* Rx DMA Config */
    /* Init DMA, 1 byte bursts, each burst requires a request */
    RxChannel = RxDMA_DmaInitialize(1, 1, HI16(SrcAddr), HI16(DstAddr));
```

```
    RxTD = CyDmaTdAllocate();

    /* Configure this Td as follows: */
    /*  - The TD is looping on itself */
    /*  - Increment the destination address, but not the source address */
    CyDmaTdSetConfiguration(RxTD,
                            TD_SIZE,
                            RxTD,
                            TD_INC_DST_ADR);

    /* From the FIFO to the memory */
    CyDmaTdSetAddress(RxTD, LO16(SrcAddr), LO16(DstAddr));

    /* Associate the TD with the channel */
    CyDmaChSetInitialTd(RxChannel, RxTD);

    /* Enable the channel */
    CyDmaChEnable(RxChannel, 1);
}

void Dma_Tx_Configuration()
{
    uint32 SrcAddr, DstAddr;
    uint8 val;

    SrcAddr = (uint32) test_data;
    DstAddr = I2S_1_Tx_dpTx_u0__F0_REG;
#if (defined(__C51__))
    /* In the case of the PSoC 3 8051 processor the upper bytes of an SRAM  */
    /* pointer is a type field that does not apply for DMA operations.      */
    SrcAddr &= 0xFFFF;
#endif

    /* Tx DMA Config */
    /* Init DMA, 1 byte bursts, each burst requires a request */
    TxChannel = TxDMA_DmaInitialize(1, 1, HI16(SrcAddr), HI16(DstAddr));
    TxTD = CyDmaTdAllocate();

    /* Configure this Td chain as follows: */
    /*  - The TD is looping on itself */
    /*  - Increment the source address, but not the destination address */
    CyDmaTdSetConfiguration(TxTD,
                            TD_SIZE,
                            TxTD,
                            TD_INC_SRC_ADR);

    /* From the memory to the FIFO */
    CyDmaTdSetAddress(TxTD, LO16(SrcAddr), LO16(DstAddr));

    /* Associate the TD with the channel */
    CyDmaChSetInitialTd(TxChannel, TxTD);

    /* Enable the channel */
    CyDmaChEnable(TxChannel, 1);
}
```

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

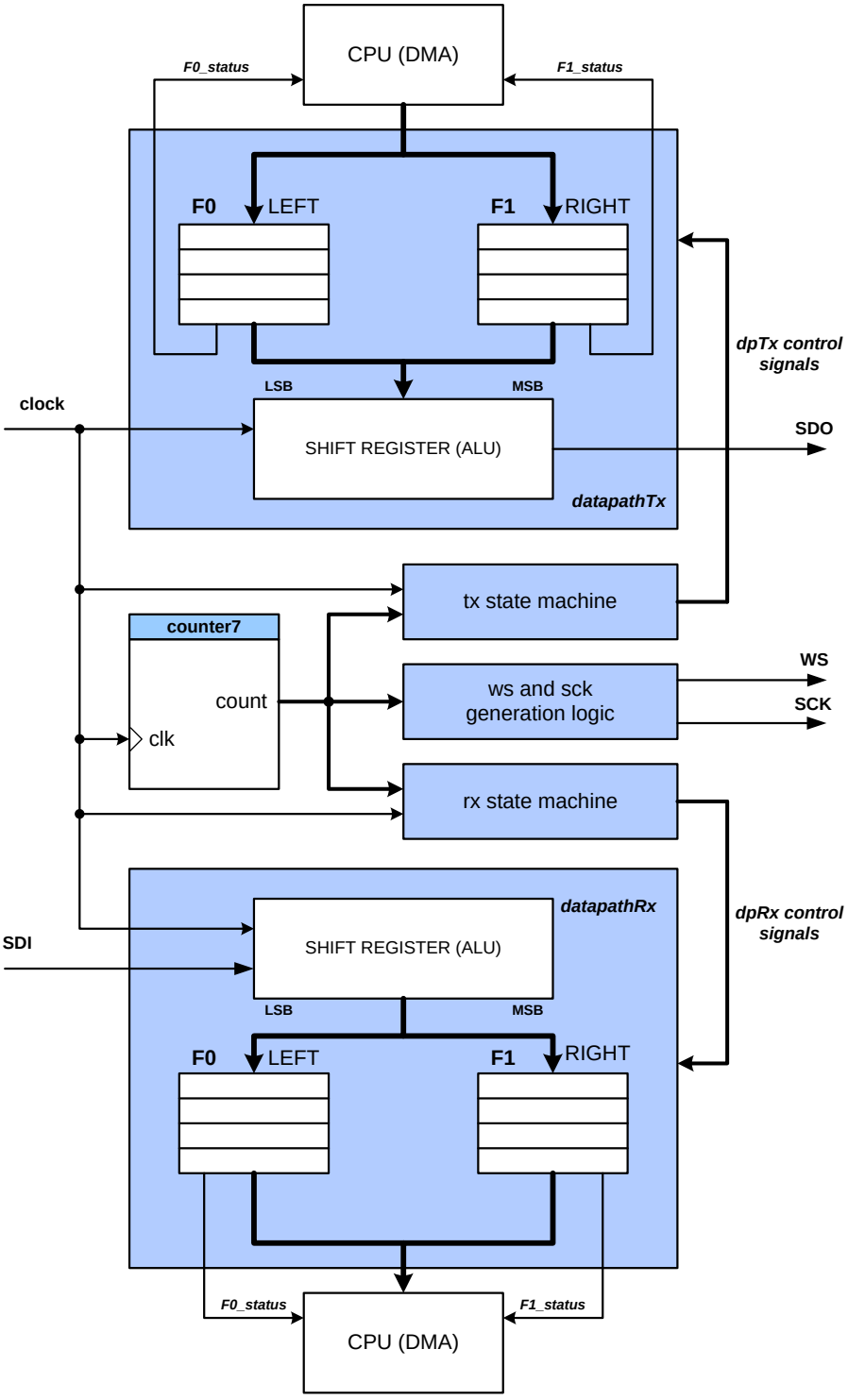
In all cases a high signal on the DMA request signal indicates that an additional single byte may be transferred.

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

5.0V/3.3V DC and AC Electrical Characteristics

Parameter	Typical	Min	Max	Units	Conditions and Notes
Input					
Input Voltage Range	---		Vss to Vdd	V	
Input Capacitance	---		---	pF	
Input Impedance	---		---	Ω	
Maximum Clock Rate	---		67	MHz	

Component Changes

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

Version	Description of Changes	Reason for Changes / Impact
2.0.b	Minor datasheet edit.	
2.0.a	Minor datasheet edit.	
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, 2010-2016. 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, 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. Other names and brands may be claimed as property of their respective owners.