

Introduction

This application note illustrates how you can use an Altera® MAX® II CPLD to provide protocol convergence to control data flow to audio devices on an inter-IC sound (I²S) bus through the serial peripheral interface (SPI).

I²S and SPI Interfaces

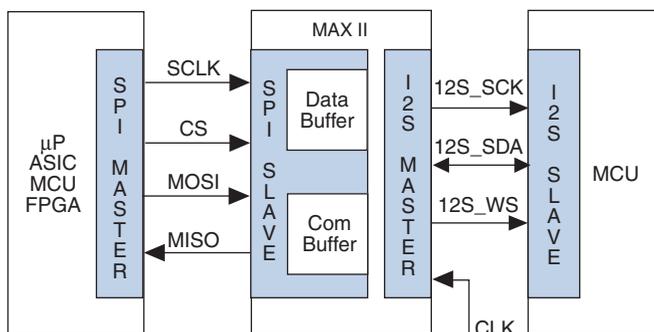
The I²S is a 3-wire, half-duplex serial interface commonly used to interconnect audio devices in a system through a common 3-wire bus known as the I²S bus. I²S devices and the bus use three wires: serial data (SDA) carrying two time division multiplexed audio data corresponding to each audio channel, serial clock (SCK), and word select (WS), which controls digitized audio data to and from the different devices on the I²S bus. An I²S system is designed to handle the serial audio data separately from the serial clock to eliminate jitter.

The SPI is a 4-wire, full-duplex serial interface popularly used to connect system processors (master) to peripheral devices (slaves) within or outside the system. The SPI communicates with separate data-out signals (master-out slave-in or MOSI) and data-in signals (master-in slave-out or MISO) lines, along with the clock (SCLK) and a slave select or chip select signal (CS).

This design enables an SPI-interface-equipped host to control data flow to devices such as A/D converters, data signal processors, digital filters, audio processors, PC multi-media audio converters, and other devices that are on an I²S bus. There are several embedded systems that do not contain an I²S interface for audio data communication, but do contain an SPI interface. You can use this design effectively under circumstances similar to these.

SPI to I²S Using MAX II CPLDs

The bridge interface to the SPI host is an SPI slave, which has four lines (CS, SCLK, MISO, and MOSI). The side interfacing with the I²S bus is an I²S master, which has three lines (I²S_SCK, I²S_SDA and I²S_WS). CLK is a master clock that is external to the I²S master. [Figure 1](#) shows implementing an SPI-to-I²S interface using a MAX II CPLD.

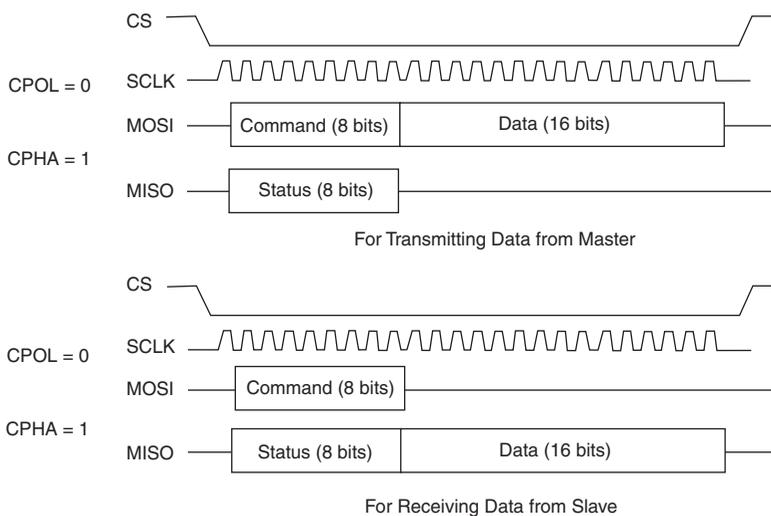
Figure 1. Implementing an SPI-to-I²S Interface Using a MAX II CPLD

This design provides a protocol convergence to control data flow to the audio devices on the I²S bus (a relatively low-bandwidth protocol used to interconnect digital audio devices in a system) through the SPI interface (the more common 4-wire serial interface used to connect serial peripheral interfaces to a microprocessor or host). The design uses the MAX II CPLD to bridge an SPI host to communicate with devices connected on an I²S bus. In Figure 1, the MAX II CPLD bridge is seen as an SPI slave to the SPI master (host processor) and in turn, performs the function of an I²S master to bridge the host to the I²S bus.

During transmission, the SPI data has the first 8 bits as the command byte, followed by the 16-bit data. The data bits are valid only when data transmission to I²S is required. During reception, the first 8 bits, known as the status byte, are followed by a 16-bit data word. The validity of this data word is specified by the data-valid bit in the status byte.

Figure 2 shows an SPI timing diagram.

Figure 2. SPI Timing Diagram



SPI Interface

Normally, the SPI bus has only one master and many slaves connected to it. The CPLD bridge acts as one of the slaves to the SPI master device.

Table 1 describes the SPI slave lines shown in Figure 1.

| Pin Name | Direction | Description |
|----------|--------------------|---------------------|
| CS | Input (active low) | Chip/slave select |
| SCLK | Input | Serial clock |
| MISO | Output | Master-in Slave-out |
| MOSI | Input | Master-out Slave-in |

As shown in Figure 2, the SPI host sends the first 8-bit command byte followed by the 16-bit data. This data is valid only if it is to be transmitted. In the event the data is to be received, or the I²S master assumes the roll of a controller, these 16 bits are invalid.

Figure 3 shows the command byte format. A description of each command byte is given in the following section.

Figure 3. Command Byte Format

| | | | | | | | |
|-------|-----|-------|------------|----|----|-------|----|
| Check | RST | ----- | Controller | WS | RC | Clock | TR |
|-------|-----|-------|------------|----|----|-------|----|

Check

This bit, sent first, specifies the validity of the word sent by the SPI master. If this bit is 0, the remainder bits following it are invalid. If this bit is 1, the remainder bits following it are valid. This bit is used only when the status is to be read from the slave.

RST

This is a reset signal for the I²S master. If this bit is 1, the I²S master gets reset. If this command bit directs the I²S master to reset, the master takes one clock cycle to reset and another clock cycle to prepare for a new command. Therefore, it takes a total of two clock cycles to complete the reset process.

 Each time a change in the function of the master is required, a command with RST is sent to reset the I²S master. After a particular function is complete, the last command again resets the I²S master. Note that whenever a reset command is sent, the remainder bits of the command byte should always contain the previous command, along with RST set to 1. This is necessary because the command is read from the command buffer before the completion of the entire data transaction cycle (15th of the 16 clock cycle).

Controller

This bit is high when the I²S master is functioning as a controller.

 For the I²S master to function as a controller, both the RC and controller bit should be high. For the I²S master to function as a receiver, only the RC bit is high.

WS

This bit specifies the word channel for each word of data that is received or transmitted.

RC

This bit is high if the I²S master is functioning as a receiver.

Clock

This is clock enable to the I²S master.

TR

This is high if the I²S master is working as a transmitter.

I²S Status Byte Format

Each time the CS is low, 24 bits are sent or received by the SPI master. The first 8 of these 24 bits indicate the status, the next 16 bits correspond to the data received.

Figure 4 shows the 8-bit status byte format. A description of each command byte is given in the following section.

Figure 4. I²S Status Byte Format

| | | | | | | | |
|------------|-------|-------|------------|-----------|-----------|--------------|--------------|
| Current WS | TR/RC | WS/RC | Data Valid | Data Buff | Data Buff | Command Buff | Command Buff |
|------------|-------|-------|------------|-----------|-----------|--------------|--------------|

Current WS

This bit specifies the current channel selected by the master.

TR/RC

This bit specifies the status of I²S master. If this bit is set to 1, the I²S master is currently a transmitter. If this bit is set to 0, it is a receiver.

WS/RC

This bit specifies the channel from which the data is received (when the I²S master is a receiver).

Data Valid

This bit specifies the validity of the 16-bit data. When the I²S master is a receiver and this bit is set to 1, the 16 bits received are valid. When the I²S master is a receiver and this bit is set to 0, the 16 bits received are invalid.

Data Buff

These two bits specify the current status of the data buffer (there are three buffers, each two bytes, that buffer the data to be transmitted). The following are the four possible states of these buffers:

- 00: all buffers are empty
- 01: buffer 1 and 2 are empty, buffer 3 is full
- 10: buffer 1 is empty, buffers 2 and 3 are full
- 11: all buffers are full

Command Buff

These two bits specify the current status of the command buffer in a format similar to that of the data buffer mentioned previously.



The command buffer status changes on the 15th clock cycle of the data transaction on the SDA line; whereas the data buffer status changes on the 16th.

When the CPLD bridge receives the command byte, the most significant bit (MSB) is checked for a 1. The command byte is then sent to the command buffer only if this is true. The 16-bit data that follows these 8 bits are then stored in the data buffer. Both the command buffer and data buffer can store up to three sets of command bytes and three sets of data words. The host checks the status each time to learn the status of the buffers and to decide whether or not to transmit new command bytes and data words.

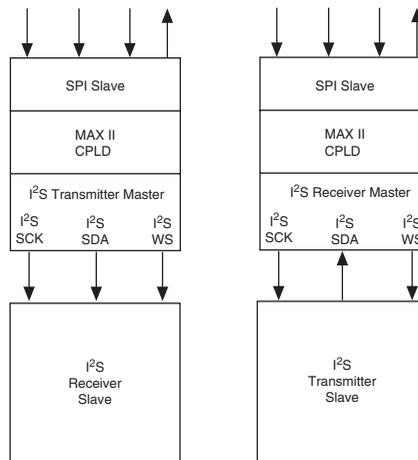
When the data-receive transaction of the I²S master is complete, the data valid bit of the status register is set to high and the I²S data begins to transfer to the SPI slave. This data can now transfer to the SPI master through the MISO pin. The data valid bit is set to 0 when this MISO transaction is complete.

I²S Interface

The I²S portion of the MAX II CPLD functions as an I²S master connected to the I²S slaves on the I²S bus. [Table 2](#) shows the I²S lines on the SPI-to-I²S bridge.

| Pin Name | Direction | Description |
|-----------------|------------------|-------------------------------|
| I2S_SCK | output | I ² S serial clock |
| I2S_SDA | bi-directional | I ² S serial data |
| I2S_WS | output | I ² S word select |
| CLK | input | inputmaster clock input |

The I²S master begins by reading data from the command and data buffers that are set by the SPI slave. If the respective command byte specifies a data transmission, the data taken from the corresponding data buffer is transmitted serially on the I2S_SDA line. After the 15th data bit is sent, the I²S master reads a new command byte from the command buffer to be ready with the next course of action as specified in the new command byte. This is especially significant for the WS bit, which changes one clock cycle prior to the next data transaction to direct the new data to a different channel. The I²S master has now functioned as a transmitter to the I²S slave receiver, as shown in [Figure 5](#).

Figure 5. Possible I²S Master Scenarios

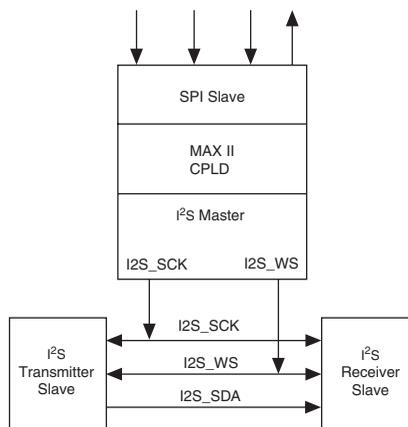
If the command byte read by the I²S master from the command buffer specifies a data reception, the data is received from the I²S-SDA line. After the 16 bits of data are received on this line, they are transferred to the SPI slave, together with setting a 1 on the data valid bit of the status register. This enables the SPI slave to transmit the data on its MISO line to the SPI host. The WS bit of the status byte is also updated, specifying the channel of the data received.

I²S Master as Controller

If the SPI master requires the I²S master to function as an I²S controller, the corresponding bits (`controller`, `RC`, and `clock`) in the command byte are set to high. The I²S master then enables the `I2S_SCK` and `I2S_WS` signals and tri-states the `I2S_SDA` line.

Figure 6 shows the I²S master in controller mode.

Figure 6. I²S Master in Controller Mode



Implementation

This design may be implemented using an EPM570 or an EPM1270. The design source code is compiled and can be programmed into the MAX II CPLD Map. The SPI interfacing ports and the I²S bus lines to suitable general purpose I/Os (GPIOs), as shown in Figure 1.

Source Code

This design has been implemented in Verilog. The source code, testbench, and complete Quartus[®] II project are available at:

www.altera.com/literature/an/an487_design_example.zip

Conclusion

As shown in this design, MAX II CPLDs are a great choice to implement interface protocol converters such as SPI to I²S. Their low cost, low power, and easy power-on features make them the ideal programmable logic devices for interface protocol converter applications.

Additional Resources

The following are additional resources for this application note:

- MAX II CPLD home page:
<http://www.altera.com/products/devices/cpld/max2/mx2-index.jsp>
- MAX II Device Literature page:
<http://www.altera.com/literature/lit-max2.jsp>

- MAX II Power-Down Designs:
<http://www.altera.com/support/examples/max/exm-power-down.html>
- MAX II Application Notes:
[AN 428: MAX II CPLD Design Guidelines](#)
[AN 422: Power Management in Portable Systems Using MAX II CPLDs](#)

Document Revision History

Table 3 shows the revision history for this application note.

| <i>Table 3. Document Revision History</i> | | |
|-------------------------------------------|---------------------|---------------------------|
| Date and Document Version | Changes Made | Summary of Changes |
| December 2007, v1.0 | Initial release. | — |



101 Innovation Drive
 San Jose, CA 95134
www.altera.com
Technical Support:
www.altera.com/support
Literature Services:
literature@altera.com

Copyright © 2007 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

