Application Report
SLEA001 – February 2001

I2C and the TAS3001C

Digital Audio Group

ABSTRACT

The TAS3001C stereo audio digital equalizer provides a serial control interface using the I2C protocol. Since the TAS3001C is more complex than the typical serial EEPROM often found on an I2C bus, designers must consider additional aspects of the I2C specification. Methods of interfacing to the TAS3001C, applicable to a wide variety of I2C masters, are discussed.

Introduction

The I2C Protocol

An I2C bus consists of a serial data signal, SDA, and a synchronous clock, SCL. Both signals are open-drain wire-and, with passive pullups. An I2C *master* initiates and controls the bus transaction, while an I2C *slave* device responds to the transaction.

I2C transactions are byte-oriented. A transaction begins with a slave address byte (actually 7 address bits plus a read/write bit), followed by a subaddress byte (for addressing registers within the slave), followed by one or more bytes either written to the slave device or read from it.

During the transaction, the device that is actually sending a byte at any point in time is called a *transmitter*, while the other device is a *receiver*. When writing data to a slave, the master device is the transmitter. When the master is reading data bytes, the slave is the transmitter. At the end of each byte, the receiver asserts an acknowledgement by pulling down the SDA for one clock cycle.

During a transaction, SDA changes only during the low state of SCL. The start and stop of a transaction are signaled by violating this rule. When the I2C bus is idle, both SCL and SDA are pulled high by passive pullups. A start condition is signaled by transitioning SDA from high to low while SCL is still high. Likewise, a stop condition is signaled by transitioning SDA from low to high while SCL is high.

START OF I2C TRANSACTION

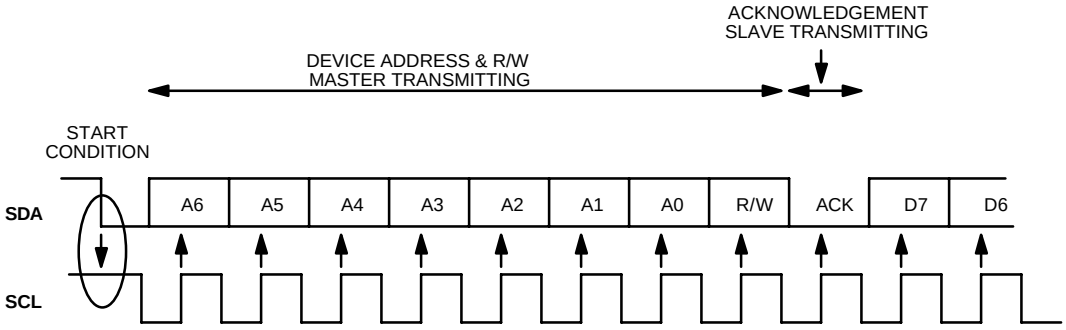

Figure 1. Start of I2C Transaction

END OF I2C TRANSACTION

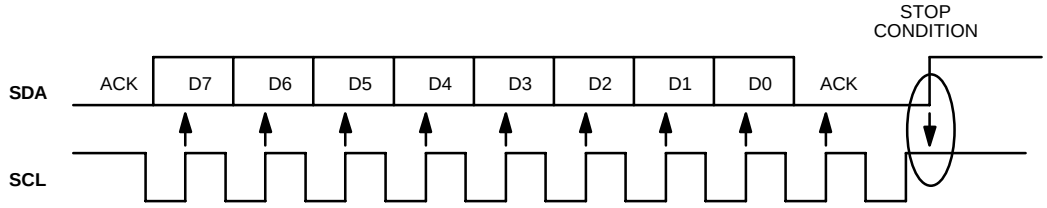

Figure 2. End of I2C Transaction

I2C Flow Control

The I2C specification allows for devices that need *flow control* that is devices that cannot keep up with the transaction at full bus speed. This is often needed by a device in which an embedded processor is shared between managing the I2C interface and attending to real-time tasks. I2C accomplishes this by means of *wait states* for clock synchronization.

A slave device inserts a wait state by holding the SCL signal low. A fully compliant I2C master will detect this condition and synchronize its internal SCL generator by waiting for SCL to be released by the slave.

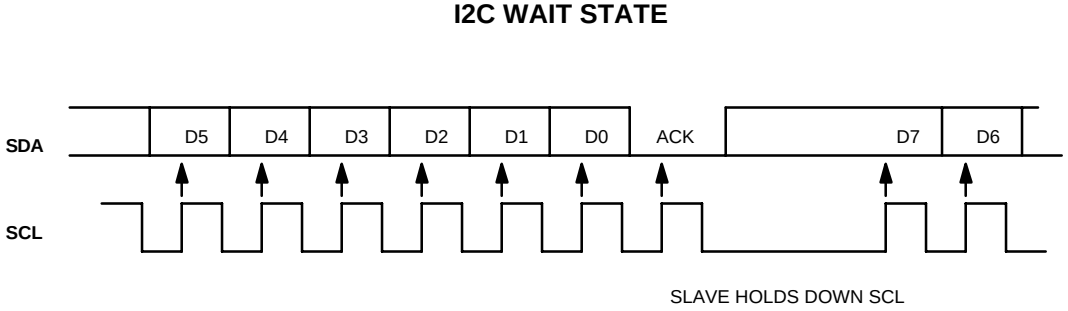

Figure 3. I2C Wait State

Any I2C master that supports wait states directly can manage the TAS3001C without any further work by the software programmer. However, if the application is real-time-critical, the designer should consider whether the selected master handles wait states by causing the software to stay in a polling loop during the wait state. This report will illustrate how to reduce, and sometimes eliminate, I2C wait states with the TAS3001C.

The designer should also be aware that some I2C masters do not implement clock synchronization, and therefore cannot handle wait states directly. This report will illustrate some techniques for working around this limitation where flow control is needed.

Structure of TAS3001C Commands

Each command to the TAS3001C consists of the device address followed by a one-byte subaddress and one or more additional data bytes.

The subaddress is an I2C data byte that addresses a particular control function inside the TAS3001C. For example, the volume control and the bass and treble controls each have their own subaddress. After the subaddress, there are six data bytes for the volume control, three for each stereo channel, while the tone controls each receive one data byte. See the TAS3001C data sheet for more detailed information.

The command structure of the TAS3001C dictates that there is a fixed number of bytes for each subaddress. In the example above, volume control requires six bytes. If only five of the six volume control bytes are received and a stop condition is detected, the TAS3001C will use the next data byte that is sent to complete the volume transaction, regardless of the subaddress.

Address	Subaddress	Next Data Byte
68h	xx	FC

If the criteria for numbers of data bytes are not met, errors will occur.

Understanding the TAS3001C Interface

Command Delays

Internal processing of commands takes a certain amount of time in the TAS3001C. These times are predictable (see Calculating Command Delays, below). If the TAS is allowed time to process a command (by a delay programmed into the I2C master software), then it will be immediately ready to accept a new one. If a delay is not provided, then the TAS will generate wait states between the first and second data bytes (on the acknowledgement of the first byte) of the next command.

There are several ways to implement command delays. The simplest is to write software that waits the required amount of time after each command. This method has the drawback that the master microcontroller is tied up waiting after each command.

A better method, if processor time is at a premium, is to start a hardware timer at the end of each command. The timer will run while the software does other tasks. The timer is then checked by the software just prior to sending a new command. If the timer has expired, the new command is sent immediately; otherwise, the software must wait by polling the timer. Since the length of each wait state can be calculated, it is preferable to use this method and assign an optimal delay to each command given to the TAS3001C.

If processor time is extremely critical, the software can instead place each new command on a queue to be serviced when the timer expires and generates an interrupt.

Timing illustrations of command delays may be found below, in the section *Programming Notes*.

DRC

The TAS3001C has an internal processor that manages on-chip peripherals. The processor selects one of two management modes, depending upon whether the dynamic range compression feature has been invoked.

If DRC has been invoked, the mode is altered such that wait states will be generated during each I2C transaction. The wait states will appear between data bytes, and will be a maximum of two sample clocks. This mode persists until the next reset, even if DRC is turned off.

Eliminating Wait States When DRC not Used

A simplified I2C interface may be implemented by preventing wait states from occurring with the TAS3001C, which allows operation with even the simplest I2C master and minimal software.

There are two steps to accomplish this. First, verify that dynamic range compression subaddress is never invoked. Second, ensure that the TAS is allowed sufficient time to completely process a command before a new command is sent. Sample code to handle command delays can be found in the zip file of code accompanying this application report.

Handling Wait States

TAS3001C Wait States

In general, the TAS3001C generates I2C wait states in two situations. First, if DRC is invoked, short wait states are generated after each data byte. Second, if command delays are not inserted between sending commands, the TAS3001C will generate a wait state after the first data byte of a command, until it is finished processing the previous command.

When the TAS3001C generates a wait state, it will do so only on the I2C clock following an acknowledgement bit. If the master has I2C clock synchronization capability, then the software is in principle relieved of the responsibility for dealing with wait states or command delays.

If processing time of I2C master is at a premium, then the software must still be designed with cognizance of command delays (see *Command Delays*, above), if the alternative is to lose processing time while waiting to send the next byte in a multibyte transaction.

In any case, if it is anticipated that the TAS will sometimes generate wait states, then the master device must take steps to handle them in order to prevent loss of synchronization between master and slave, which could leave the slave in an indeterminate state.

Example: Simple I2C Master

Since the behavior of the TAS3001C is highly predictable, it is feasible to control it with many I2C masters that do not have internal support for I2C wait states. We will use for illustration the TUSB3200.

The TUSB3200 contains an 8051-derivitive core and an I2C interface that does not handle clock synchronization. Despite this limitation, it is feasible to use the TUSB3200 to directly control a TAS3001C, access all features including dynamic range compression, and optimize real-time performance. We assume for this illustration that the software manages command delays, and that the TAS3001C only generates wait states due to DRC.

The method to follow takes advantage of the fact that the TUSB3200 holds SCL low after a byte is written, if the STPWR bit of the I2CCTL register is not set. (See *TUSB3200 Data Manual.*) In other words, as long as the TUSB3200 is not programmed to generate a stop condition on the I2C bus, it will hold the clock line down between data bytes. Since the I2C bus specification requires the slowest device to determine the low period of SCL, the TAS3001 can generate a wait state, while the TUSB3200 is between data bytes, and not lose synchronization. Therefore, the software should delay for the equivalent of two sample clocks after each data byte.

However, after the last data byte, the TAS may not see the TUSB3200-generated stop condition, which immediately follows the last data byte. Since the TAS may be in a wait state and holding SCL low while the TUSB3200 sends the stop condition, it is likely that the TAS will miss the I2C stop.

The TAS needs to see a stop condition before a new command is sent. A stop condition can be forced on the I2C bus by using an open-drain general-purpose I/O pin from the TUSB3200 to toggle SDA while SCL is high. After completing the I2C command, the software should wait for two sample clocks (allowing for any TAS wait state), and then use a GPIO pin to pull SDA down and then release it. This will actually send a start condition followed by a stop condition, but the TAS will see the stop and be ready for the next command.

Here is an illustration using a tone command:

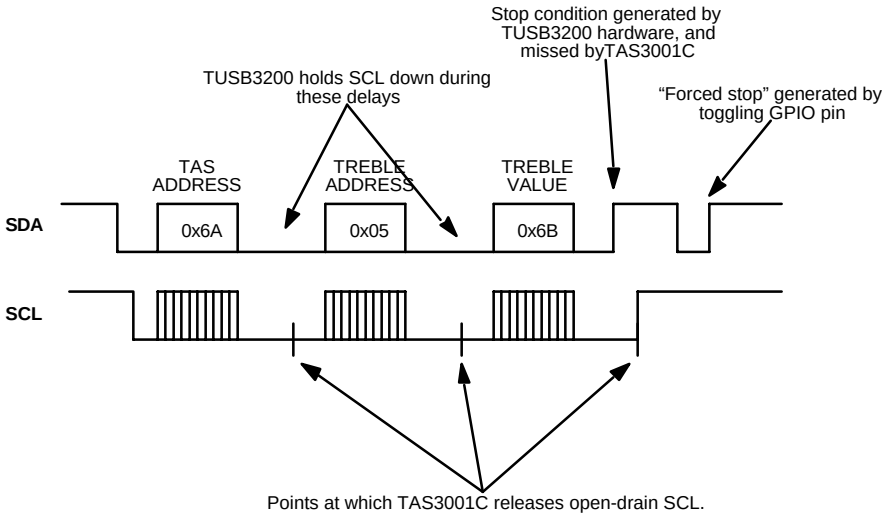

Figure 4. I2C Stop Condition by TUSB3200

To sum up, an I2C master without clock synchronization capability can be used to control the TAS3001C and still access DRC. The method is to use short delays (greater than two sample clocks) after each data byte, and force a stop condition with a GPIO pin two sample clocks after the end of each transaction. Sample code that runs on the TUSB3200 and handles TAS3001C wait states as well as command delays is found in the zip file accompanying this application report.

Programming Notes

Internal Control Ramping

Since the audio signal handled by the TAS3001C is represented digitally, and since all filtering and scaling is accomplished by mathematical operations, instantaneous changes to the filtering and scaling parameters would ordinarily cause audible artifacts (pops or thumps).

For this reason, the volume, bass, and treble controls are automatically adjusted by the TAS according to internal algorithms. These algorithms have to be more sophisticated than a simple linear ramp in order to eliminate all audible artifacts, and are designed to work with each particular control.

It is not necessary, or even desirable, for the designer to attempt control ramping in software, unless for special effects (such as fade-in or fade-out) that require especially long ramp times.

By the same token, the designer can utilize the volume control to mute and un-mute, simply by changing the setting with a single command, without concern for pops or clicks.

Calculating Command Delays

The command processing delays for volume, bass, and treble controls are calculated in terms of the sample frequency (LRCLK). All commands require a few sample clocks for overhead, but the volume and tone controls require substantially more for internal control ramping.

The volume control requires 2048 sample clocks to process both left and right channels.

The tone controls require 64 sample clocks per step. For example, if treble is changed to +3dB from 0dB, this represents 0x72 - 0x6B or 7 steps. (See the appendix of *TAS3001C Stereo Audio Digital Equalizer* data book, document number SLAS226.) This would take 448 sample clocks. At 44 KHz, it would represent 10.2mS.

The command delay can be included by the software between commands:

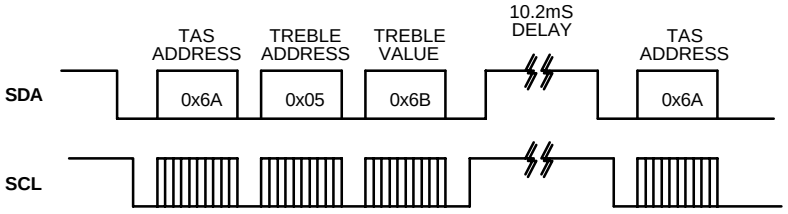

Figure 5. I2C Software Delay

Otherwise, it will appear as a wait state after the first data byte of the next command:

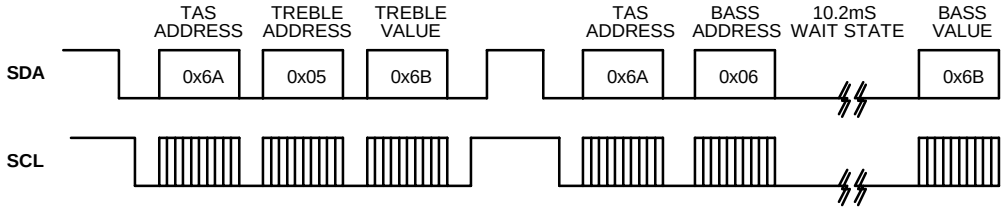

Figure 6. I2C Wait State of TAS3001

Note that the number of steps is the difference in the actual integers programmed, not the difference in decibel setting. The largest step change would be from –18 to +18dB, which represents 0x85 steps, or 8512 sample clocks.

Using Fast Load

Since the TAS3001C biquad filters can be programmed with arbitrary coefficients, there is no internal algorithm available to smoothly ramp from one filter implementation to another. Furthermore, it is likely that hundreds of samples must pass through the filter before it stabilizes mathematically. The best way to change the biquad coefficients is to use fast load mode.

In fast load, audio processing ceases. I2C transfers to the TAS3001C proceed without wait states. In order to avoid clicks and pops associated with abruptly changing the state of audio processing, the designer can make use of internal control ramping and simply mute the TAS, enter Fast Load, reprogram the biquads, leave fast mode, and restore volume level.

Aborted Transfers

Aborted transfers are a part of I2C. However, the I2C specification only deals with what might be thought of as the datalink layer. What happens after the abort depends on the application. In the case of the TAS3001C, the correct method for handling an aborted transaction depends on whether it was a single- or multibyte transaction.

In the case of a single-byte transaction (or a multibyte transaction that aborted before data bytes were sent), the I2C master should simply retry the command. However, if the abort occurred during a data byte of a multibyte read or write transaction, the master will have to flush the partial transaction out of the TAS before a retry. To accomplish this, the master should send sixteen null bytes to the same slave device and subaddress, and then retry.

Conclusion

The TAS3001C presents a more sophisticated I2C interface than the usual serial EEPROM. However, the behavior of the TAS is highly predictable, and this allows the designer to interface with virtually any I2C master.

Definitions

aborted transaction – An I2C bus transaction that could not be completed because a byte was not followed by an acknowledgement.

control ramping – The process of changing the setting of a control in small steps to avoid audible clicks, pops, or other artifacts.

controls – Predefined variable audio functions provided by the TAS3001C, such as bass and treble.

DRC – Dynamic range compression.

I2C clock – The clock cycle that appears on the SCL signal of the I2C interface.

I2C clock synchronization – The method employed by the I2C master to wait until a slave releases the low state of SCL, before timing the high state of SCL.

master – The I2C device that initiates and controls a transaction.

master microcontroller – The microprocessor associated with the I2C master.

multibyte transaction – An I2C transaction in which more than one data byte is read or written. Note that all I2C transactions involve more than one byte for addressing. It is the data bytes that make it a single-byte or multibyte transaction. See single-byte *transaction*.

null byte – Eight consecutive zero bits that are logically grouped together.

receiver – The I2C device that is receiving data bytes (the slave device during a write command, or the master during a read).

sample clock – The clock that defines the PCM sampling period. The TAS3001C pin that receives the sampling clock is *LRCLK*.

single-byte transaction – An I2C transaction involving only one data byte. Note that all I2C transactions involve more than one byte for addressing. See *multibyte transaction*.

slave – The I2C device that responds is addressed by the master.

software – The software program running on the microprocessor that controls the master I2C interface, and presumably runs the application that programs the TAS3001C.

start condition – the beginning of an I2C transaction, signaled by transitioning SDA from high to low while SCL is high.

stop condition – the end of an I2C transaction, signaled by transitioning SDA from low to high while SCL is high.

subaddress – The I2C address of a register within the slave device.

transaction – An I2C bus transaction, consisting of a start condition, slave address, read/write bit, subaddress, one or more data bytes, and a stop condition. Each byte is followed by an acknowledgement bit from the receiver.

transmitter – The device that is sending a byte.

wait state – The low state of SCL when extended by a slave device which holds it low after the master has released it.

References

1. *The I2C-Bus Specification*, Version 2.1
2. *TAS3001C Stereo Audio Digital Equalizer* (SLAS226), September 1999
3. *TUSB3200 Data Manual - USB Streaming Controller (STC)* (SLAS240), October 1999

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgment, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Customers are responsible for their applications using TI components.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, license, warranty or endorsement thereof.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations and notices. Representation or reproduction of this information with alteration voids all warranties provided for an associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Resale of TI's products or services with *statements different from or beyond the parameters* stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Also see: Standard Terms and Conditions of Sale for Semiconductor Products. www.ti.com/sc/docs/stdterms.htm

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265