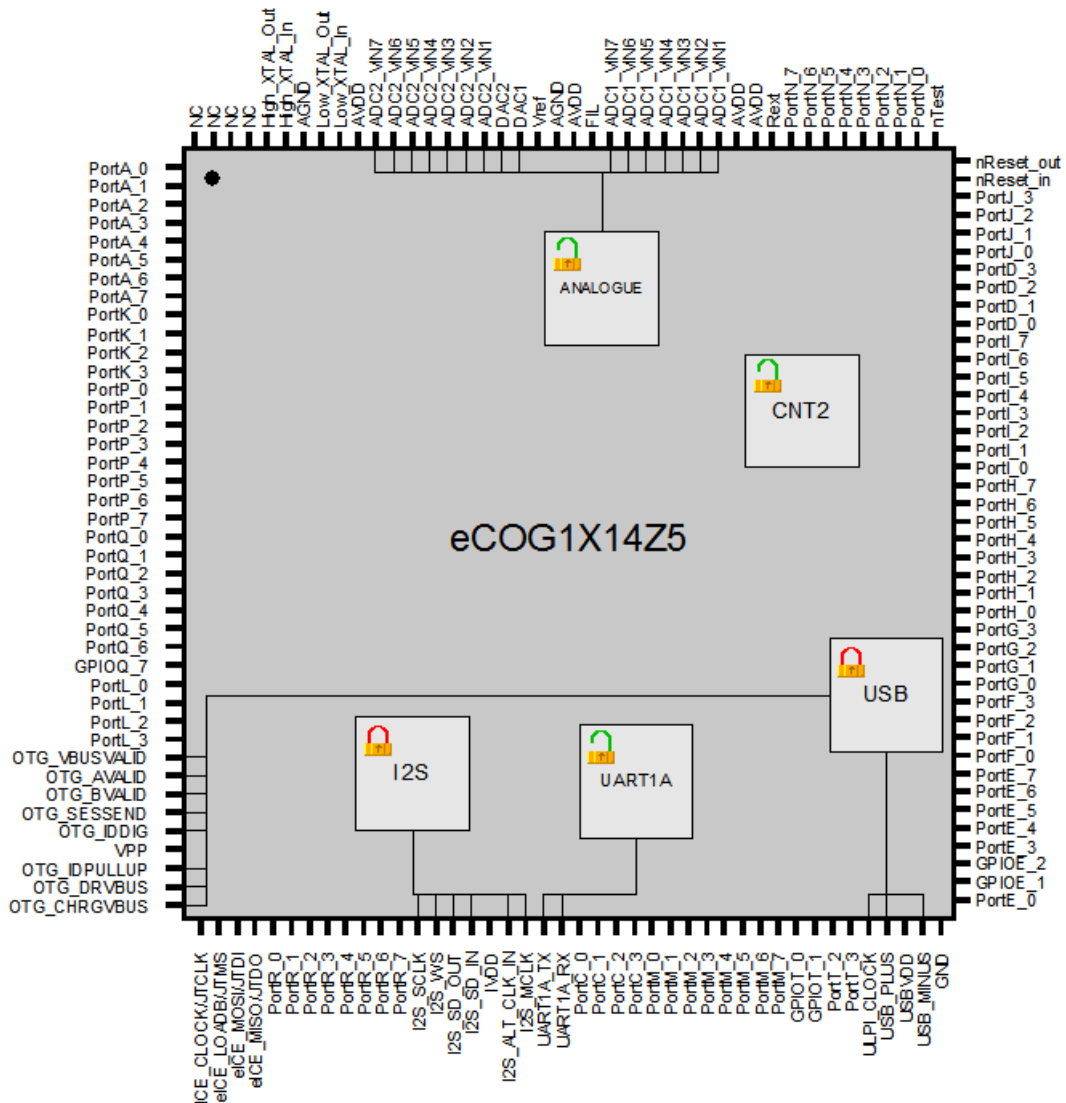


AN056 – eCOG1X USB Audio Example

Version 1.0

This application note describes an example implementation of a USB audio device on the eCOG1X development board.



Confidential and Proprietary Information

©Cyan Technology Ltd, 2008

This document contains confidential and proprietary information of Cyan Technology Ltd and is protected by copyright laws. Its receipt or possession does not convey any rights to reproduce, manufacture, use or sell anything based on information contained within this document.

Cyan Technology™, the Cyan Technology logo and Max-eICE™ are trademarks of Cyan Holdings Ltd. CyanIDE® and eCOG® are registered trademarks of Cyan Holdings Ltd. Cyan Technology Ltd recognises other brand and product names as trademarks or registered trademarks of their respective holders.

Any product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by Cyan Technology Ltd in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. Cyan Technology Ltd shall not be liable for any loss or damage arising from the use of any information in this guide, any error or omission in such information, or any incorrect use of the product.

This product is not designed or intended to be used for on-line control of aircraft, aircraft navigation or communications systems or in air traffic control applications or in the design, construction, operation or maintenance of any nuclear facility, or for any medical use related to either life support equipment or any other life-critical application. Cyan Technology Ltd specifically disclaims any express or implied warranty of fitness for any or all of such uses. Ask your sales representative for details.



Revision History

Version	Date	Notes
V1.0	03/07/2007	First release

Contents

1	Introduction	5
2	Glossary	5
3	USB Audio Data	6
3.1	Overview.....	6
3.2	Processing Audio Data for Playback	6
3.2.1	<i>Simple Algorithm with Low CPU/Memory Requirements.....</i>	<i>6</i>
3.2.2	<i>Doubling the Output Sample Rate.....</i>	<i>7</i>
3.2.3	<i>Possible Enhancements When Using Internal DAC.....</i>	<i>7</i>
3.2.4	<i>Circular Buffer Algorithms.....</i>	<i>7</i>
3.2.5	<i>Resampling.....</i>	<i>7</i>
4	Audio Input and Output.....	8
4.1	Using the Dual DAC	8
4.2	Buffering the DAC Outputs	9
4.3	DAC Software Overview	9
4.4	I2S Audio	10
4.5	I2S Overview	10
4.6	eCOG1X I2S implementation	11
4.7	Wolfson WM8731S Codec	11
4.8	Circuit Diagram and Interconnection	12
4.9	I2S Software Overview	13

1 Introduction

CyanIDE 1.4 includes an extensive USB support library for the eCOG1X. It includes a number of software “plugins” that can be added to applications to support common USB functions. One of these plugins allows the eCOG1X to act as a USB audio peripheral. A simple minimal example using this plugin is supplied with CyanIDE 1.4; however, that example does not make use of any audio hardware to play or record sounds.

This application note discusses how to extend this example to play and record audio using different hardware devices available in the eCOG1X.

2 Glossary

A table of abbreviations used in this document.

ADC	Analogue to Digital Converter
Codec	COder / DECoder
DAC	Digital to Analogue Converter
eCOG1	Cyan Technology target micro controller
FIFO	First in First Out
I2S	Inter IC Sound
ISR	Interrupt Service Routine
SSM	System Support Module
UART	Universal Asynchronous Receiver/Transmitter

3 USB Audio Data

3.1 Overview

Standard USB Audio Class Devices use isochronous transfers to stream real time audio from the host to the peripheral device. Every 1ms a new packet is sent containing approximately 1ms of audio data.

Individual samples are known as *audio subslots* and occupy 1, 2, 3 or 4 bytes. Samples that are not a multiple of 8 bits long are left justified and padded with enough zeros to meet this requirement. The samples are encoded as two's complement numbers. They also use little endian byte ordering, and must be converted to and from big endian when used with a big endian CPU such as the eCOG1X.

One sample (*audio subslot*) is taken from each audio channel at the same time and grouped together to form an *audio slot*. This application note uses stereo audio, so each audio slot contains two audio subslots (left and right). However, the USB audio format can also support other numbers of channels.

The average number of *audio slots* in each packet is determined by dividing the sample rate by 1000. Each packet must contain only whole slots, so if the result of this calculation is not a whole number then some packets must be slightly longer than others, so that the average can match the above calculation.

3.2 Processing Audio Data for Playback

In the example code for this application note, the USB audio peripheral receives 16-bit 48kHz stereo audio data from the attached PC. This arrives in packets with almost exactly a 1ms spacing, each packet containing 192 bytes. This received audio data is placed into a buffer and sample pairs are read from it, also at a rate of 48kHz.

The challenge is that while both ends of the USB connection are using a 48kHz sample rate, they are not running from the same clock. As no clock is 100% accurate, inevitably one of the two processes is slightly faster than the other. As the clock speed of both ends drifts slightly over time, it is also possible for the "fast" and "slow" ends to swap.

Where the output sample rate is fixed, it is necessary to rate-adapt the input data rate to match the output data rate. At its simplest, such a process may simply duplicate samples and discard samples as needed. More complex systems may re-sample blocks of input data and regenerate the output data by taking weighted averages of adjacent input samples.

If the output sample rate can be fine tuned, then, in addition to the above methods, it becomes possible to store input data in a FIFO buffer and dynamically to fine tune the output sample rate to maintain the buffer level within a defined region.

There is no single correct algorithm for all situations, each approach has different requirements in terms of buffer memory requirements, CPU workload and perceived audio quality. In the following sections, some options are described in more detail.

3.2.1 Simple Algorithm with Low CPU/Memory Requirements

The simplest algorithm tested provides surprisingly good results, while only needing sufficient buffering for two USB audio packets. No additional copying or computationally expensive interpolation was performed on the data, minimising the CPU workload.

The algorithm makes use of two buffers. At any time playback is from one buffer and the other buffer is available to receive new USB audio data packets. Data is read from the appropriate buffer by the playback ISR and sent to the output device. If the end of the buffer is reached before a new packet is received, the final sample is repeated. When a new USB packet is received, this is placed into the appropriate buffer and then the roles of the two buffers are swapped. If the new buffer arrives before the previous buffer has finished playing, the remaining samples are discarded and playback starts from the new playing buffer.

This simple algorithm works quite well, the distortion introduced by the duplication and removal of samples at the boundary between audio packets is not readily apparent when playing back typical music using the internal 12-bit DAC. The distortion is easier to hear in some passages of music or when playing pure sine waves. This effect is subjectively more apparent when sending 16-bit data to the external codec; probably this is because when using the codec there is less quantisation noise to hide the added distortion.

This group of algorithms are likely to work better when using a sample rate where packets normally contain the same number of samples, for example rates that can be divided by 1000 and give a whole number, such as 48kHz. More doubling and discarding occurs at 44.1kHz where some packets are longer than others.

3.2.2 Doubling the Output Sample Rate

One method to reduce the distortion is to use an output sample rate equal to twice the input data rate. Normally each input bit is played twice, but when repeating data at the end of a packet or discarding data, this can be done in steps of half the input sample time. New input data is used for every other output value. To help keep the output data synchronised to the input data, this sequence is reset when a new packet begins playback.

This method does not add significant complexity to the code and requires no additional buffering. However, it does require twice as many DAC or I2S interrupts to be processed. A good improvement may be heard when using a 48kHz (input) sample rate.

3.2.3 Possible Enhancements When Using Internal DAC

The DAC interrupt handler is generated using a counter/timer. It is relatively easy to stop, start and dynamically adjust the period of this counter. It is possible to stop the counter/timer when the end of input data is reached, and restart it when each new USB audio packet is received. This may give even better synchronisation than the above method, without needing to double the output sample rate and the number of interrupts. The current example code does not demonstrate this method.

3.2.4 Circular Buffer Algorithms

The rate at which samples need to be duplicated and/or discarded may be reduced by pushing all received data into a large circular buffer, and then reading it out in the playback ISR. This improvement is likely to be most obvious when playing sample rates that require some packets to be longer than others. The disadvantages of this type of algorithm are the additional copying of data and the increased memory requirements for the circular buffer.

One possible enhancement of this technique when using the internal DAC is to dynamically fine tune the output sample rate, keeping the buffer at a near constant level. Effectively this resamples the input data without needing any costly calculations to be performed on the CPU.

3.2.5 Resampling

Where the output sample rate is fixed and asynchronous to the input audio samples, the best audio quality is obtained by resampling the input data to the output clock frequency. Such processes are typically complicated and CPU intensive and are beyond the scope of this application note. In high performance audio systems, this is done in hardware with dedicated sample rate converter devices rather than in software.

4 Audio Input and Output

The example source code that accompanies this application note can use the integrated dual DAC (with external buffering) or the I2S interface (with an external Wolfson codec) to drive headphones or speakers. The code uses the same basic algorithm for both output devices.

The source code also supports audio recording from the line or microphone inputs of the Wolfson codec. The code may be modified easily to support recording using the internal ADC of the eCOG1X.

4.1 Using the Dual DAC

Most members of the eCOG1X family include a two channel 12-bit DAC. Values to be output are written by the application software to the `data` fields of the `aci.dac1` and `aci.dac2` registers for DAC1 and DAC2 respectively. Once written to the data registers, these values must be transferred to the DAC output registers before the analogue output voltage changes. This can happen in one of three ways:

- The `aci.dac1` and `aci.dac2` registers contain a `load` bit as well as the `data` bit field. When writing a new value to these registers, setting the `load` bit causes the value to be transferred immediately to the output register.
- Writing a '1' to the `ws` bit field in the `aci.dac1_load` or `aci.dac2_load` register causes conversion to begin on a value previously written to the `aci.dac1` or `aci.dac2` data registers respectively.
- DAC output conversion may be configured to start on a timer/counter event. This mode is enabled for DAC1 by setting the `dac_sync_tim` bit field in the `aci.ctrl_en1` register, and the specific timer trigger event is selected by the `dac_tim_sel` field in the `aci.tim_cfg1` register. For DAC2, use the bit fields in the `aci.ctrl_en2` and `aci.tim_cfg2` registers.

It is also possible to configure the two DAC output channels to update together. This is done by setting the `dac_sync_ch1` bit field in the `aci.ctrl_en2` register. When this bit is set, the second DAC uses the same conversion event as the first DAC, ensuring that both channels update simultaneously.

In this application, both DAC channels are used to allow stereo playback. As both the left and right channels update together, the second DAC can be configured to update at the same time as the first DAC, simplifying the code.

The second counter/timer is configured to generate a periodic interrupt at the playback sample rate and new data is written to both the DAC `data` registers. To ensure that each sample period is the same length even if the ISR cannot be serviced immediately, the DACs are configured to update when the timer reaches zero. As long as the ISR is serviced within one sample time, no audio degradation occurs due to jitter.

The clock speed and reload value of the counter timer determine the playback sample rate. Some popular sample rates and corresponding reload values are shown below, for a 12MHz clock input to the counter/timer.

Desired sample rate	Reload value	Actual sample rate	Error (%)	Typical uses
8.000 kHz	1499	8.000 kHz	0.000	Digital telephone systems, PC
11.025 kHz	1087	11.029 kHz	0.040	PC
22.050 kHz	543	22.059 kHz	0.040	PC
44.100 kHz	271	44.118 kHz	0.040	CD Audio, PC
48.000 kHz	249	48.000 kHz	0.000	DAT, DVD, PC
96.000 kHz	124	96.000 kHz	0.000	

Table 1. Timer reload values

Reload values for other sample rates can be calculated with the following formula.

$$reloadValue = \left(\frac{CounterClock}{SampleRate} \right) - 1$$

4.2 Buffering the DAC Outputs

It is not possible simply to connect headphones (or speakers) to the output of the dual DAC. These devices present largely inductive loads and as such are essentially a short circuit at DC and low frequencies. For this reason, it is necessary to ensure that no DC voltage is applied and typically this is achieved by capacitively coupling them to the driver. In addition, the eCOG1X DAC outputs cannot supply the current required to drive headphones so additional buffering is required.

The eCOG1X development board includes buffering on the analogue output signals using high speed op-amps. However, these devices do not have adequate current drive or the coupling capacitors required to drive headphones. It is recommended that a simple headphone amplifier circuit is used, such as an LM488. If a true line level is required, the output from the eCOG1X should be attenuated slightly.

4.3 DAC Software Overview

By default, the supplied software uses the W8731 codec as the analogue output device. If a suitable output buffer circuit is added to the eCOG1X analogue outputs, then the software can easily be modified to use the on-chip dual DAC. Edit the include file *options.h* to define the symbol *USE_ACI* and undefine the symbol *USE_I2S* before rebuilding the project.

4.4 I2S Audio

The eCOG1X includes an I2S (Inter IC Sound) peripheral interface that allows easy connection to external audio devices, such as an audio codec. The eCOG1X development board includes a Wolfson WM8731S codec that uses this interface to transfer audio data to and from the eCOG1X. The codec device provides stereo line level input, stereo line level output, a headphone amplifier, and a microphone amplifier with a bias supply suitable for use with typical electret microphones.

4.5 I2S Overview

The Inter-IC sound interface (I2S) was developed by Philips to allow the easy interconnection of digital audio devices such as ADCs, DACs, digital signal processors, etc. This interface allows the transmission of stereo audio streams using a simple 3-wire interface. The I2S interface is used exclusively for audio data transmission and cannot be used to configure devices on the bus.

The left and right channels are time multiplexed and sent on the data line, SD. The SCK signal is used to clock the data bits and the WS (word select) signal is used to indicate both the start of each sample in the serial data and which channel is currently being sent. The change of channel occurs on the second rising edge of SCK after a change of WS. This is illustrated in I2S timing.

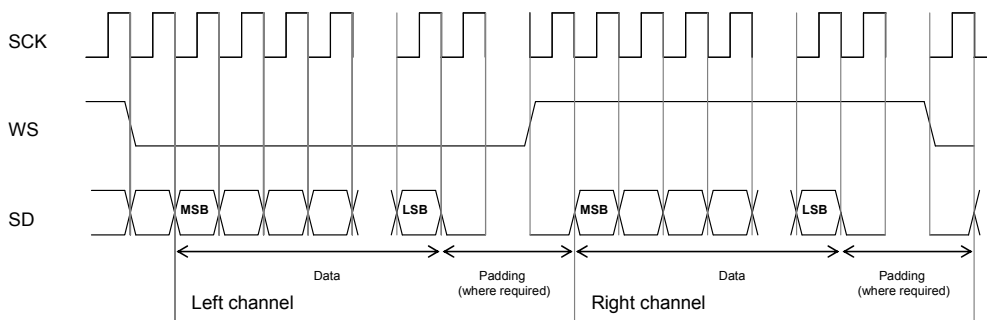


Figure 1. I2S timing

The serial data sent on SD is clocked into the receiver on the rising edge of SCK. It is normal for the transmitter to update SD on the falling edge of SCK, as this gives the maximum possible data setup and hold times. The WS signal may change on either edge of SCK; it is sampled by the slave on the rising edge of SCK. The I2S specification allows the WS signal to be asymmetrical, with a different number of SCK periods for the left and right channels, although the eCOG1X I2S in master mode does not support this.

I2S is designed to allow communication between devices using different sample sizes. To support operation in this way, data is sent MSB first. If the receiver is sent fewer bits than its sample size, it sets the missing bits to zero. If it receives more bits than it needs, the extra bits at the end of the sample are discarded. Similarly, if the transmitter does not see enough SCK edges to send the full sample, the LSBs are simply not sent. If more SCK clock edges are received than are required for the word size, then the sample is padded to the required length. The data is sent in two's complement format.

One device on an I2S bus is the master and drives the SCK and WS signals. This can be either the transmitter or the receiver depending on the system. The sample rate is equal to the frequency of the WS signal.

For more information on I2S, refer to the document *I2S Bus Specification* available from the NXP website (www.nxp.com).

4.6 eCOG1X I2S implementation

Full information on using this block is contained in the eCOG1X User Manual. The following material is a brief introduction and highlights some additional points of note.

The eCOG1X I2S peripheral can be configured either as the timing master, where it generates SCK and WS, or as the slave and use externally generated timing signals. The eCOG1X can be both a transmitter and a receiver simultaneously.

The eCOG1X transmitter and receiver both share the same WS and SCK signals. This is not normally a limitation, as the most common application is with the transmitter and receiver connected to a single external audio codec that provides the analogue audio input and output signals at the same sample rate. Note that this does restrict its use to applications where input and output audio data use the same sample rate, although the external codec may support other configurations.

In common with many I2S devices, the polarity of various signals can be changed. However, it is simplest to configure everything to operate as described in the I2S specification and in the previous section.

When operating as the master, the eCOG1X produces a symmetrical WS signal. The length of both the high and low phase (in multiples of the SCK period) is set by the *word_len* bit field in the *i2s_cfg1* register.

When operating as the master, the eCOG1X I2S block can use either an internal or external clock signal.

- Internal clock from the eCOG1X SSM block.
This clock can be driven out on I2S_MCLK. It may be used to provide a high speed clock that the external codec uses for its internal oversampled digital filters.
The serial bit clock I2S_SCLK is generated directly from this clock, or by dividing it by one of 15 available ratios, ranging from $\div 2$ to $\div 1024$.
- External clock input signal on I2S_ALT_CLK_IN.
This input provides an alternate clock source for the I2S peripheral. It allows the use of clock frequencies from an external source that cannot be achieved by the SSM.

Alternatively, the eCOG1X can be set to slave mode and an external I2S device can act as the master. In this case, all timing is derived from the SCLK and WS signals, which are now inputs from the external master. Note that in this mode the MCLK output is a buffered version of the SCLK input.

In audio applications, normally it is necessary to achieve accurately the required standard sample rate, such as 44.1kHz for CD audio data. The sample rate is equal to the frequency of the WS signal and (in master mode) to the frequency of SCLK divided by $2 \times \text{word_len}$. An external codec may also require an MCLK signal to be supplied at one of several device specific ratios to the sample rate, so that its digital filters operate correctly. It may not be possible to achieve both these requirements using the standard recommended crystals on the eCOG1X. In such cases, either use a crystal with a suitable frequency, or connect a separate clock source of the required frequency to the I2S_ALT_CLK_IN input.

Where the external codec can operate as the timing master, this may be the easiest approach.

4.7 Wolfson WM8731S Codec

The eCOG1X development board is fitted with a Wolfson WM8731S codec to allow audio playback and recording. This part uses three serial interfaces; the first is a write only interface (similar to SPI) that is used to configure the device. The other two interfaces are used for the audio data, one for audio input and the other for audio output. These can be configured for a number of common synchronous audio formats including I2S.

On the eCOG1X development board, a 12.288MHz crystal is connected to the built-in crystal oscillator on the WM8731. This supports the following common sample rates: 96kHz, 48kHz, 32kHz and 8kHz. Note that with this crystal frequency, it is not possible to use a sample rate of exactly 44.1kHz. The manufacturer's data sheet for the WM8731 indicates that this audio sample rate can be achieved only by fitting a crystal of a different frequency.

4.8 Circuit Diagram and Interconnection

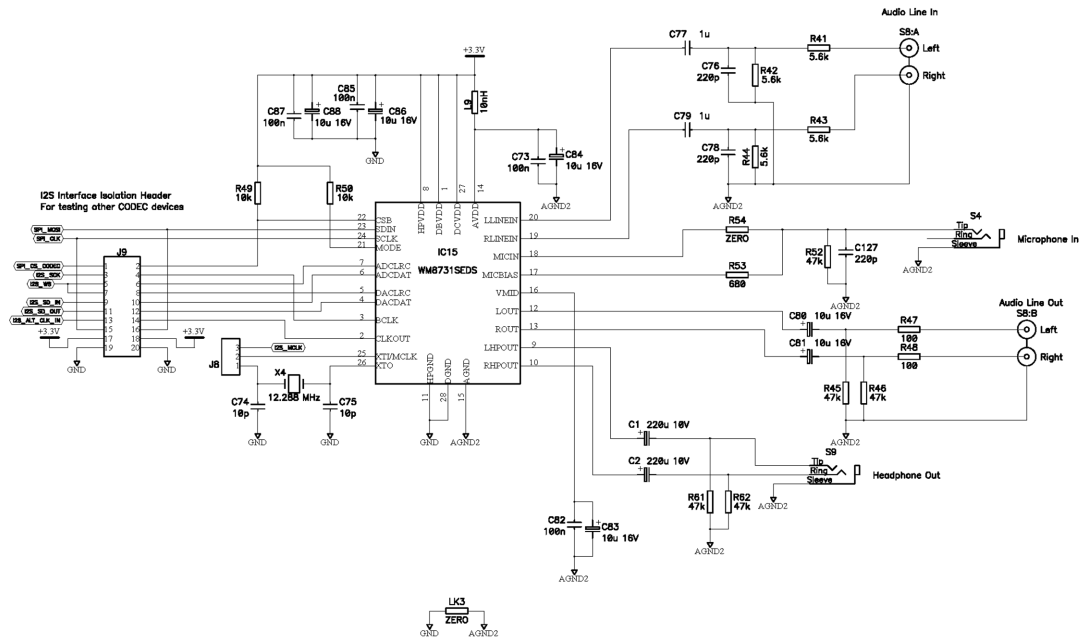


Figure 2. Connections to WM8731 codec

The above diagram shows the connections to the WM8731 codec on the eCOG1X development board. Jumper links are provided for flexibility. These allow the user to:

- Configure the circuit for either the eCOG1X or the WM8731 to be the I2S timing master.
- Configure which device provides MCLK.
- Isolate the on-board WM8731 and allow easy access to all the I2S signals, so that systems using other I2S devices can be prototyped.

The software for this application note should be used with the following jumper links:

- J8: link pins 1-2.
- J9: link pins 1-2, 3-4, 7-8, 9-10, 11-12.

The eCOG1X prefixes all I2S signals with "I2S_" for clarity. It has two SD signals, one for transmit data and one for receive data, labelled with the suffixes "_IN" and "_OUT". The WM8731 codec uses different signal names. The table below provides a cross-reference for the signal names.

Function	I2S standard	eCOG1X	WM8731	
			DAC	ADC
I2S Clock	SCK	I2S_SCK	BCLK	
Word Select	WS	I2S_WS	DACLR	ADCLR
Data	SD	I2S_SD_IN		ADCDAT
		I2S_SD_OUT	DACDAT	
Master Clock		I2S_MCLK	MCLK	

Table 2. I2S signal names

4.9 I2S Software Overview

The example software performs the following initialisation steps to use the codec.

- 1) Initialise the WM8731 codec using its SPI-like configuration port to:
 - a. Disable mute and set gain for left and right channels.
 - b. Set the required sample rate.
 - c. Set the audio input source to line in.
 - d. Set the digital audio interface for I2S format.
 - e. Set as the timing master.
 - f. Disable power down.
 - g. Enable the digital audio interface.
- 2) Configure the eCOG1X I2S peripheral for:
 - a. 16-bit data.
 - b. Enable transmit.
 - c. Enable receive.
 - d. Enable ready interrupts for transmit and receive on both channels.
 - e. Set as a timing slave.
 - f. Use external SCLK and WS.

Once initialisation is complete, a single ISR services all the I2S interrupts. It polls the I2S peripheral to determine which data registers are ready and transfers the corresponding samples between the I2S block and the audio buffers.