Application Note

Digital Audio Playback on TM4C12x MCUs

Charles Tsai

ABSTRACT

There are several options for microcontrollers (MCUs) to output digital audio data for audio playback. This application report describes two methods to output digital audio data stored in either the internal flash memory or an external SD card to a speaker. First method is to use Pulse Width Modulation (PWM) to emulate a Digital-to-Analog Converter (DAC) to reconstruct an analog signal from digital signal. PWM DAC is a low-cost method because PWM is readily available on most microcontrollers. The second method is to directly use an off-chip DAC when higher quality playback is required. This application report demonstrates how to play audio files encoded in .WAV format to an external speaker using both the methods from TM4C12x MCU.

Project collateral and source code discussed in this application note can be downloaded from the following https://www.ti.com/lit/zip/spma084.

Table of Contents

Trademarks

Code Composer Studio™ is a trademark of Texas Instruments.
All trademarks are the property of their respective owners.

1 Introduction

Digital audio is a representation of sound recorded in digital form. In a digital audio system, an analog signal representing the sound can be converted, with an analog-to-digital converter (ADC), into a digital signal using the pulse code modulation (PCM) method. In PCM, the amplitude of an analog signal is sampled regularly at uniform intervals and each sample is quantized into digital steps. The uniform interval is called the sampling rate and the number of discrete levels a signal can be quantized into is called the bit depth. For example, CD audio achieves a sampling rate of 44.1 kHz with a bit depth of 16 bits. What this means is that an analog audio signal is sampled 44100 times in a second and the amplitude of the signal is quantized into 2^{16} or 65536 steps.

Once the audio signal is digitized, it can be stored digitally as a file and it can be edited, modified, and manipulated using digital machines such as a computer or MCU. To play the digital audio file on a speaker, a reverse process is performed. The analog audio signal is simply recreated by using a digital-to-analog converter (DAC) to convert the digital signal back into an analog signal. The analog signal is then typically sent through an audio amplifier to boost the signal before reaching the speaker.

Figure 1-1 illustrates a generic digital audio synthesis and playback system. The DAC shown in the figure can be: an integrated component to the MCU, or a discrete component external to the MCU, or emulated by using PWM signal. As TM4C12x MCUs do not have an integrated DAC, this applicable report will demostrate playing back digital audio using both PWM emulation of a DAC and an external discrete DAC.

This application report focuses more on how the TM4C12x MCU can be used to playback rather than recording the digital music stored in an audio file.

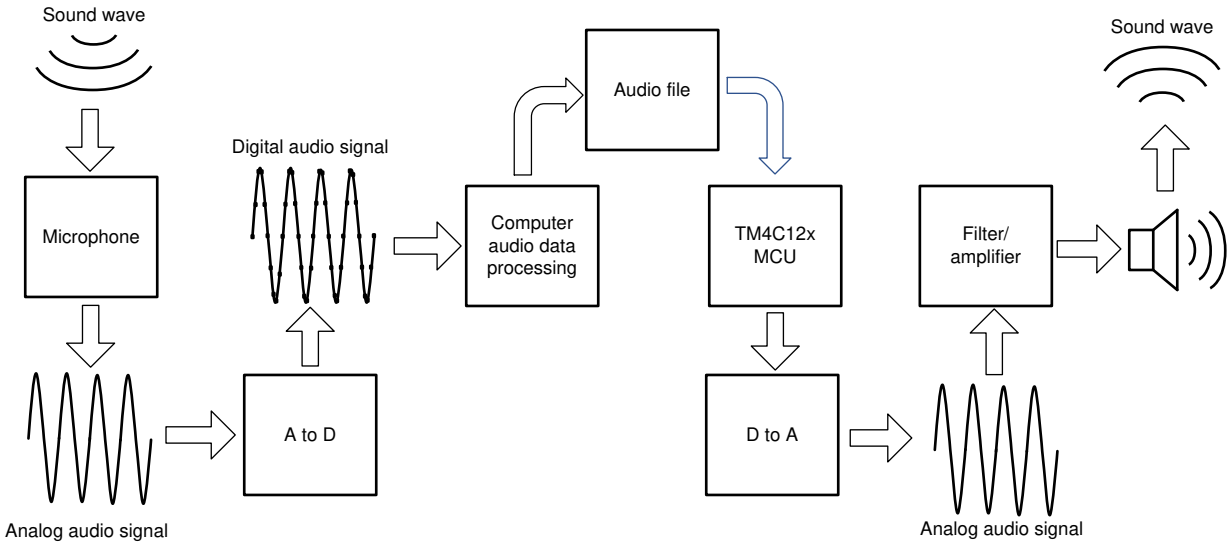

Figure 1-1. Example Audio Synthesis and Playback Systems

2 Pulse Width Modulation (PWM) emulation of a DAC

Not all microcontrollers have an integrate DAC on chip, but almost all have the capability to generate PWM signals either with a dedicated peripheral or with a timer module. PWM is a low-cost way to control analog devices (a speaker) with a digital output. PWM emulates an analog signal by means of controlling the duty cycle. Suppose a 100% duty cycle PWM is produced from a 3.3 V MCU, it is easily expected the PWM will produce a 3.3 V DC level signal. A 0% duty cycle PWM is expected to produce a 0 V DC signal as well. What about a 10% duty cycle periodic PWM? If a multimeter is used to measure the PWM output, it is not surprised the multimeter measures 3.3 V * 10% = 0.33 V. A 50% duty cycle PWM will measure 3.3 V * 50% = 1.65 V. This relationship between duty cycle and measured voltage is fairly intuitive. This can be concluded with a simple equation as shown in Equation 1:

$$\text{Average Voltage} = \text{High Voltage Level} \times \text{Duty Cycle} \quad (1)$$

Earlier in the Section 1, it was mentioned that each time an analog audio signal is sampled, the signal is recorded as a binary number and the length of the binary numbers is called the *resolution*. In another word, the resolution expresses the number of discrete steps an analog signal can be represented. If the length of binary numbers is 16, this is denoted as 16-bit resolution. A 16-bit binary value can produce 2^{16} = 65536 discrete steps of a signal. Dividing 3.3 V by 65536 is equal to about 50 µV for each step.

Figure 2-1 shows an example analog signal that is digitized with 12 evenly spaced samples at 3-bit resolution. With only 3 bits of resolution, there are only 8 discrete steps. At this resolution, severe quantization error can result which is the difference between the analog signal and the closest available digital value at each sampling instant from the ADC. In Figure 2-1, samples 3, 4 and 5 are converted to the same digital value when they are different in its analog form at the respective sample points.

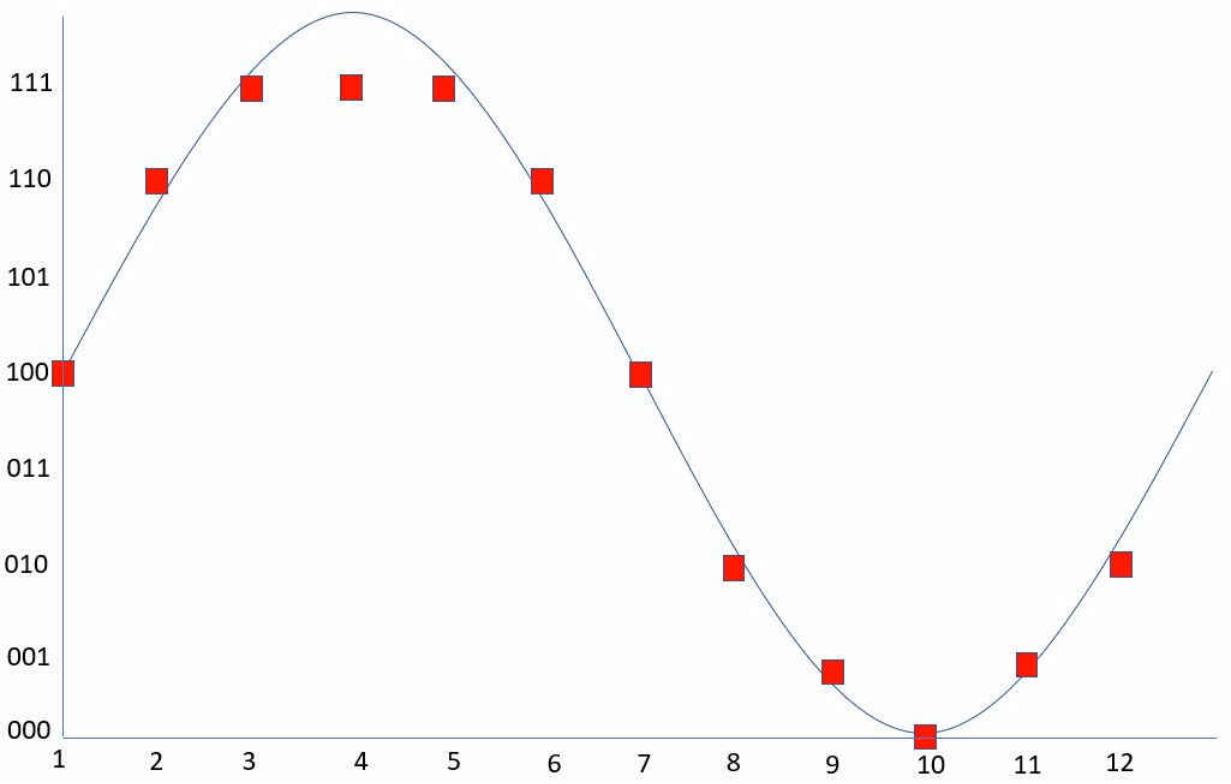

Figure 2-1. Example Analog Signal Digitized With 12 Samples at 3-Bit Resolution

To recreate the analog signal in Figure 2-1 using PWM, the duty cycle is obtained by dividing the digital value at each sample point by the bit depth which is 2^3 = 8.

Figure 2-2 illustrates the reproduction of the analog signal based on the sampling rate and resolution shown in Figure 2-1. Although it resembles the original analog signal but it does not come close to true reproduction. Improvements can be made by increasing the sampling rate and resolution. Imagine the original analog signal was digitized at 8 kHz (8000 samples per second) and 8 bit (256 discrete steps) resolution or higher, the original analog signal will be reproduced by the PWM DAC method at much higher accuracy.

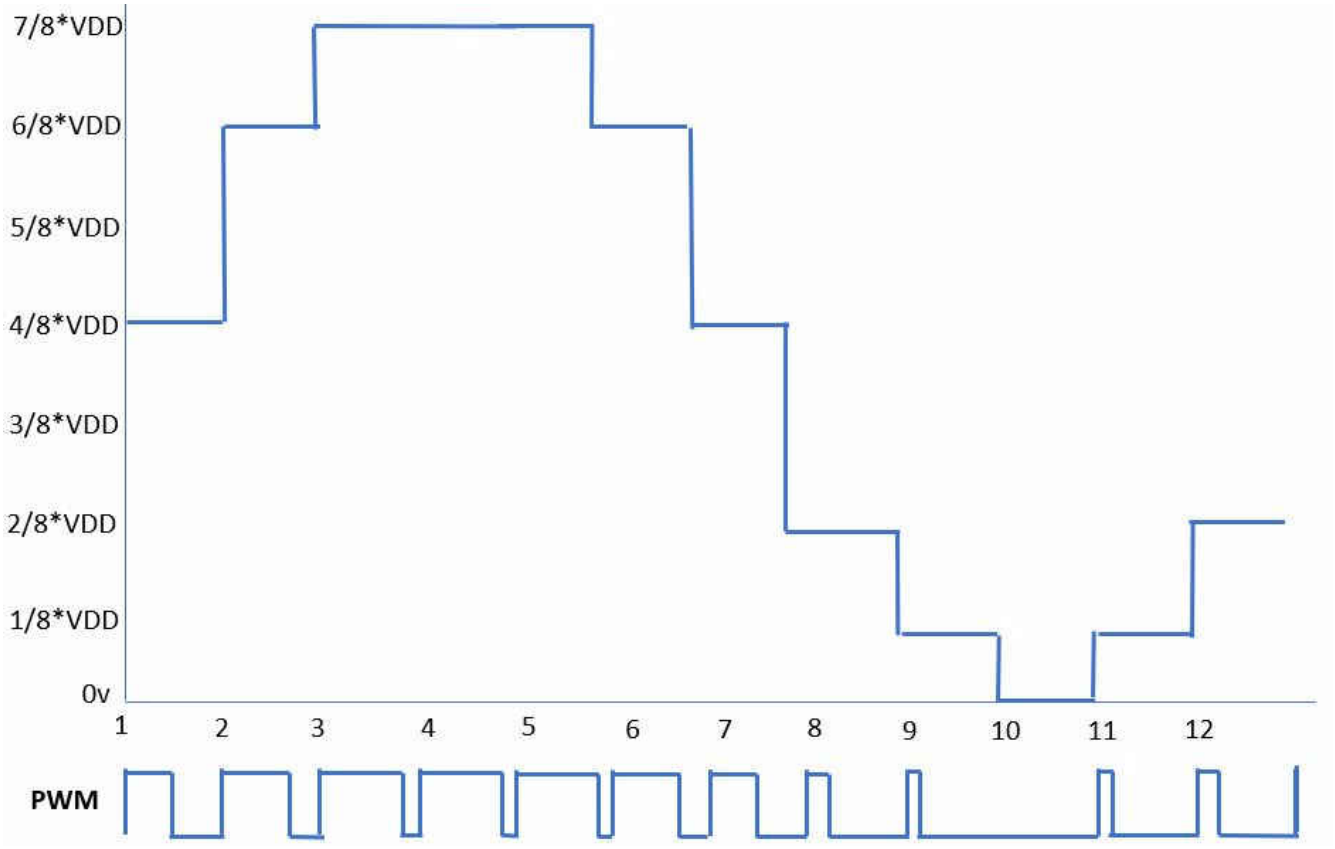

Figure 2-2. Reproduction of Analog Signal Using PWM DAC

3 Audio Sampling Rate and Resolution Analysis

3.1 Using the PWM DAC Method

It is understood that a CD quality audio is digitized at a 44.1 kHz sampling rate with 16-bit resolution. The question to then ask is can an MCU achieve this sampling rate and resolution during playback using the PWM DAC method?

To achieve the audio sampling rate during playback, MCU can create a simple timer at the expected frequency. At each timer interrupt, a PWM is generated for the required duty cycle at the desired resolution. This is straight forward for most microcontrollers. Normally the desired sampling rate can be generated by dividing from the timer's timebase. If the desired sampling rate is integer divisible by the timebase, then the sampling rate can be fully achieved. For sampling rates that are not fully integer divisible, MCU can still reach the desired sampling rate at very high accuracy. Table 3-1 column C shows the achieved sampling rates when the timer uses SYSCLK as the timebase operating at 120 Mhz. As can be seen, for sampling rates at 22.05 kHz and 44.1 kHz, the achieved sampling rates are within 0.00% of the expected. In another word, MCU will be playing back the audio at three decimal points faster than the expected playback speed. For a non-audiophile person, this is not discernible.

To achieve the desired resolution is a different story. For 16-bit resolution, it means there are 2^{16} = 65536 discrete steps. In another word, to achieve 16-bit resolution, PWM must be able to vary its duty cycle at the granularity of 1/65536 of a PWM period in order to achieve a 16-bit DAC equivalent. This is evidently not possible as shown in the second column of Table 3-1 where there are only 2721 SYSCLK cycles in a 44.1 kHz PWM period.

The maximum achievable resolution at the respective sampling rate by MCU can be calculated by the equation as shown in Equation 2:

$$\text{Achievable Resolution} = \text{Log}_2(\text{Number of timebase clocks in one PWM period}) \quad (2)$$

Fifth column shows the maximum achieved resolution at the respective sampling rate. For 44.1 kHz the achieved resolution is 11.41. Like most microcontrollers, TM4C12x MCUs cannot achieve 16-bit resolution using PWM DAC method, but 8-bit audio resolution is certainly achievable.

Sixth and Seventh columns show the minimum timebase frequency that is required to achieve either 16-bit or 8-bit resolution for different desired sample rates.

Table 3-2 shows the analysis for TM4C123x MCU operating at maximum 80 Mhz.

Table 3-1. Sampling Rate and Resolution Analysis for TM4C192x MCU

Desired Sample Rate	Number of System Clocks at 120 Mhz	Achieved Sample Rate	Error in Achieved Sample Rate	Achieved Resolution	Minimum SYSCLK (Mhz) Required to Achieve 16-Bit Resolution	Minimum SYSCLK (Mhz) Required to Achieve 8-Bit Resolution
8000	15000	8000.00	0.00%	13.87	524.29	2.05
16000	7500	16000.00	0.00%	12.87	1048.58	4.10
22050	5442	22050.72	0.00%	12.41	1445.07	5.64
44100	2721	44101.43	0.00%	11.41	2890.14	11.29
48000	2500	48000.00	0.00%	11.29	3145.73	12.29

Table 3-2. Sampling Rate and Resolution Analysis for TM4C123x MCU

Desired Sample Rate	Number of System Clocks at 80 Mhz	Achieved Sample Rate	Error in Achieved Sample Rate	Achieved Resolution	Minimum SYSCLK (Mhz) Required to Achieve 16-Bit Resolution	Minimum SYSCLK (Mhz) Required to Achieve 8-Bit Resolution
8000	10000	8000.00	0.00%	13.29	524.29	2.05
16000	5000	16000.00	0.00%	12.29	1048.58	4.10
22050	3628	22050.75	0.00%	11.82	1445.07	5.64
44100	1814	44101.43	0.00%	10.82	2890.14	11.29
48000	1666	48019.21	0.04%	10.70	3145.73	12.29

Although it is not possible to achieve 16-bit resolution, it is still possible to generate the equivalent duty cycle ratio at the maximum achievable resolution for MCU by scaling. For example, if the audio file digitizes a sample point with a value of 0x1234 as a 16-bit value at 44.1 kHz sampling rate, the duty cycle ratio is equal to 0x1234 / 0xFFFF = 7.1%. The TM4C129x PWM counter would be configured with 2721 to generate a PWM period that is equal to 44.1 kHz and the match value would be configured to 7.1% x 2721 = 193 to generate the high phase of the duty cycle.

The PWM method is a low-cost way to emulate digital-to-analog conversion for most microcontrollers. For audio applications requiring only 8-bit resolution, PWM DAC method is certainly a viable option. For high quality audio applications, some loss of fidelity is expected.

3.2 Using Off-Chip DAC

If there is a DAC in the system, then it is just a matter of outputting the 16-bit digital value to DAC to reproduce the analog signal. Many off-chip DAC use interfaces such as SPI to take digital data from the host MCU. In this setup, MCU still uses the timer to generate a periodic interrupt at the desired sampling rate. In the interrupt ISR, the digital value is sent to the DAC through the SPI interface. If a high resolution DAC supporting at least 16-bit resolution can be used, a high fidelity audio with CD quality can be played back by microcontrollers.

4 Audio File Data Format

There are many types of audio files. Common file formats used for storing audio data are MP3, WMA, FLAC, AAC and so forth. Audio files basically fall into three major categories – audio formats with lossy compression, audio formats with lossless compression and uncompressed audio formats.

A good representation of audio formats with lossy compression is MP3. MPEG-1 Audio Layer 3 (MP3) is the most popular audio format with compression. For lossless compression, Free Lossless Audio Codec (FLAC) has become one of the most popular lossless formats. Waveform Audio File Format (WAV) is an uncompressed audio format that captures and converts real sound waves to digital format without any further processing.

It is beyond the scope of this application note to go through the compression technologies and the pros and cons of different audio formats. For simplicity to demonstrate playing audio using DAC, the examples provided in this application report only use WAV format, which is an uncompressed audio format.

4.1 WAV File Size Calculation

WAV is an uncompressed audio format based on Pulse Code Modulation (PCM) to digitally represent sampled analog signals. Details of WAV format can be readily found on the internet. Since WAV is an uncompressed format, the file size for a single audio clip can be very large. The required amount of storage can be quickly calculated for a WAV file as follows:

$$S = SR * (BL / 8) * C * L / 10^6 \quad (3)$$

Where:

- S: Storage (in MB)
- L: Length of audio (in seconds)
- SR: Sample rate (8 kHz, 16 kHz, 22.05 kHz, 44.1 kHz or 48 kHz)
- BL: Bit Length (8 or 16)
- C: Number of audio channels (1 for mono or 2 or stereo)

For a typical CD quality 4-minute song recorded in WAV format, it will take:

$$S = = 44100 * (16 / 8) * 2 * (4 * 60) / 10^6 = 42.3MB \quad (4)$$

This is certainly much larger than MP3 where a same length song may be less than 5MB. Unless the audio is short, it cannot fit into MCU's internal flash memory. Some of the examples in this application report use short audio clips that will fit into internal flash memory. Other examples store the WAV files on an external SD card for playback. Details of these examples are discussed in later sections.

4.2 Storing WAV Audio on Internal Flash

For short audio clips, the WAV data can be stored on the internal flash memory. This is actually the fastest way with least amount of latency to play the audio. In order to store digital audio on the internal flash memory, what is needed is a tool that can convert a WAV file into a C array in the header file that can be included during compile time. *Bin2c* is a open source, command line PC tool to create C files from binary files. To *bin2c*, WAV is nothing but some binary data. Therefore, *bin2c* does not need to understand or decode the WAV data. After the conversion, the contents of external files can be embedded to a C or C++ program. From there, the application running on the MCU is responsible for decoding the WAV format.

5 Hardware Setup

The examples provided in this application report can be used with three different TM4C development boards and an Audio BoosterPack Plug-in Module. The LaunchPad examples only store the WAV audio data in the internal flash memory of the MCU. The examples for the DK-TM4C129x Development Kit also play audio from WAV files that are stored on an external SD card in addition to WAV audio data stored in internal flash memory.

- EK-TM4C1294XL LaunchPad Evaluation Kit
- EK-TM4C123GXL LaunchPad Evaluation Kit
- DK-TM4C129X Development Kit
- BOOSTXL-AUDIO BoosterPack Plug-in Module

5.1 BOOSTXL-AUDIO BoosterPack Overview

BOOSTXL-AUDIO BoosterPack Plug-in Module can capture audio input from a microphone and output audio through an on-board speaker. The audio input/output lets developers experiment with the digital signal processing (DSP) and filtering capabilities of the microcontroller found on the attached development kit.

There are several options selectable by a jumper on the BoosterPack for connecting the speaker to the MCU. The following options are available for use with TM4C12x MCUs.

- Output audio data over SPI to the SPI DAC provided on the Audio BoosterPack. DAC on the BoosterPack only supports 14 bits of resolution. Therefore, MCU will scale a 16-bit audio data to 14-bit.
- Use MCU's PWM output together with a basic R/C filter on the BoosterPack, to create a simple, low-cost analog audio signal.

When running examples that use PWM to emulate a DAC, the jumper must be placed to select PWM on the BoosterPack. When running examples that use SPI DAC, the jumper must be placed to select SPI DAC. Figure 5-1 shows the three jumper selections.

Figure 5-1. BOOSTXL-AUDIO BoosterPack

5.2 Hardware Setup for EK-TM4C1294XL

EK-TM4C1294XL LaunchPad has two BoosterPack connectors. The provided examples require the BOOSTXL-AUDIO BoosterPack connect to BoosterPack 1 connector. Figure 5-2 shows the connection.

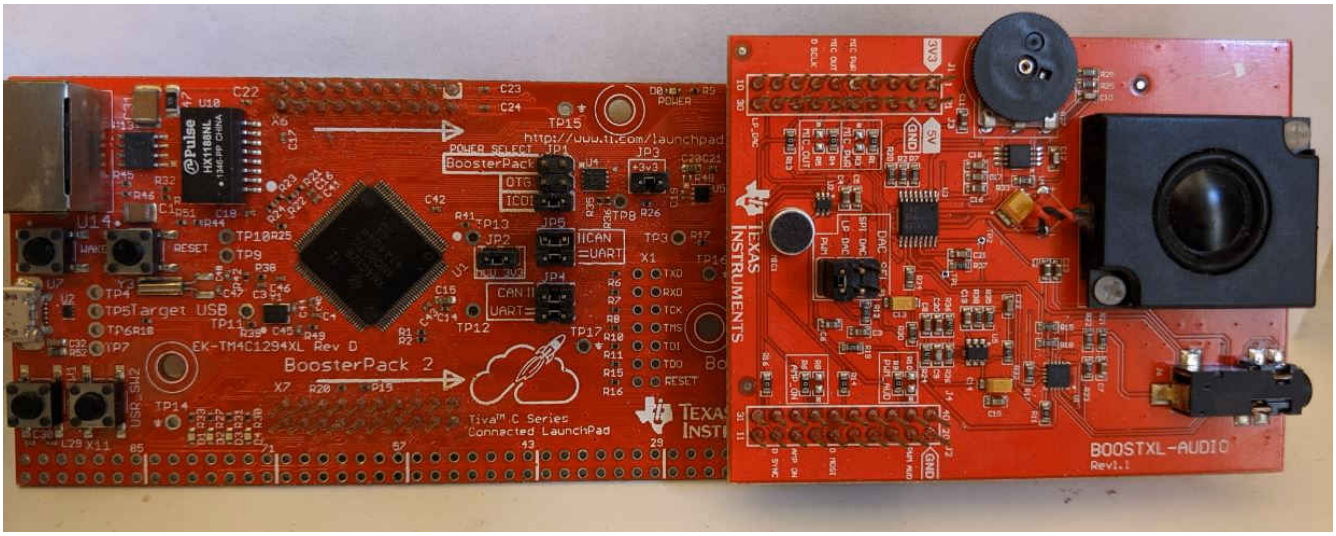

Figure 5-2. EK-TM4C1294XL LaunchPad with Audio BoosterPack

5.3 Hardware Setup for EK-TM4C123GXL

Figure 5-3 shows the BoosterPack connection to EK-TM4C123GXL LaunchPad.

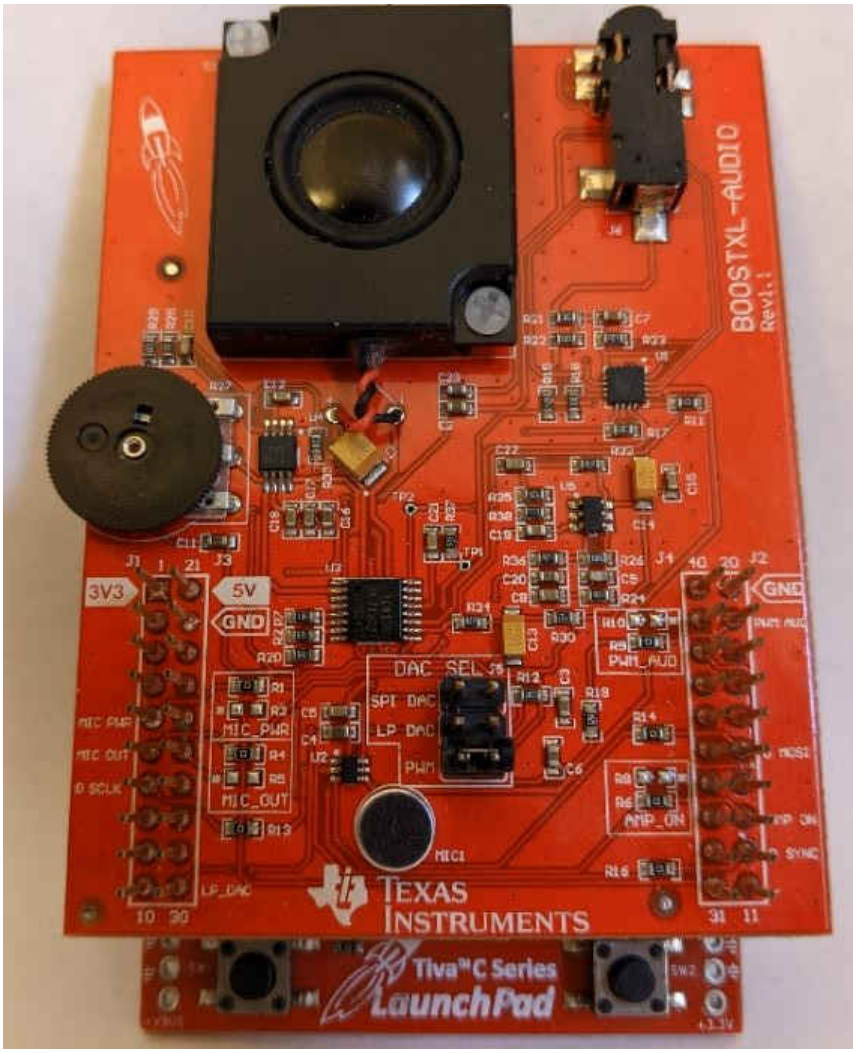

Figure 5-3. EK-TM4C123GXL LaunchPad With Audio BoosterPack

5.4 Hardware Setup for DK-TM4C129x Development Board

DK-TM4C129x Development Board has two BoosterPack connectors. The examples require the BOOSTXL-AUDIO BoosterPack connect to BoosterPack 1 connector. Figure 5-4 shows the connection.

Figure 5-4. EK-TM4C129x Development Board With Audio BoosterPack

6 Application Examples

A collection of eight examples are presented in Table 6-1 to demonstrate running the audio playback applications on different develop boards using either PWM DAC or SPI DAC methods.

Table 6-1. Application Examples

Example	Board	Description
audio_playback_with_ sdcard_by_PWM	DK-TM4C129x	This example application demonstrates playing .wav files from an SD card that is formatted with a FAT file system. The application can browse the file system on the SD card and displays all files that are found on the LCD display. Files can be selected to show their format and then played if the application determines that they are a valid .wav file. Only PCM format (uncompressed) files may be played. This example creates an audio signal to an external speaker on BOOSTXL-AUDIO BoosterPack using PWM DAC method. MCU's PWM output passes through a basic R/C filter on the BoosterPack creates a simple, low-cost analog audio signal.
audio_playback_with_sdcard_by_SPIDAC	DK-TM4C129x	Similar to the above examples where .wav files are stored on the SD card; this example creates an audio signal to an external speaker on BOOSTXL-AUDO BoosterPack by sending the audio data stored in .wav file to a DAC on the BoosterPack through a SPI interface.
audio_playback_internal_wavefile_by_PWM	DK-TM4C129x	This example application demonstrates playing a .wav audio format stored internally on the flash memory. Several example .wav files can be selected during compile time by using the #define DEMO directive. Various preloaded .wav files digitized in different sampling rates (8 kHz, 16 kHz, 22.05 kHz, 41.1 kHz and 48 kHz) and resolutions (8 bit and 16 bit) can be selected. Only PCM format (uncompressed) wav files may be played.. This example creates an audio signal to an external speaker on BOOSTXL-AUDIO BoosterPack using PWM method. MCU's PWM output passes through a basic R/C filter on the BoosterPack creates a simple, low-cost analog audio signal.
audio_playback_internal_wavefile_by_SPIDAC	DK-TM4C129x	Similar to the above example but sends the audio data stored on the flash to a DAC on the BoosterPack through a SPI interface.
audio_playback_internal_wavefile_by_PWM_EKTM4C129	EK-TM4C1294XL	Similar to audio_playback_internal_wavefile_by_PWM example but for EK-TM4C1294XL LaunchPad.
audio_playback_internal_wavefile_by_SPIDAC_EKTM4C129	EK-TM4C1294XL	Similar to audio_playback_internal_wavefile_by_SPIDAC example but for EK-TM4C1294XL LaunchPad.
audio_playback_internal_wavefile_by_PWM_EKTM4C123	EK-TM4C123GXL	Similar to audio_playback_internal_wavefile_by_PWM example but for EK-TM4C123GXL LaunchPad.
audio_playback_internal_wavefile_by_SPIDAC_EKTM4C123	EK-TM4C123GXL	Similar to audio_playback_internal_wavefile_by_SPIDAC example but for EK- TM4C123GXL LaunchPad.

6.1 Examples: audio_playback_internal_wavefile_by_PWM_EKTM4C129 and audio_playback_internal_wavefile_by_SPIDAC_EKTM4C129

These two examples demostrates playing a .wav file which is stored internally on the flash memory. Several example .wav files can be selected during compile time by using the #define DEMO directive. To play a different audio file, assign a new value to the macro DEMO. Once a new value is assigned, rebuild the project and the corresponding audio file becomes part of the program image that is ready to be programmed to the flash memory. These two examples are to be run on the EK-TM4C1294XL LaunchPad board. Remember to adjust the DAC SEL jumper on the BOOSTXL-AUDIO BoosterPack to select PWM or SPI DAC when changing between projects. The selected audio will be playbacked repeatedly. Press the USR_SW1 button on the board to pause the playback. Press the USR_SW2 button to resume playback.

Remember to adjust the DAC SEL jumper on the BOOSTXL-AUDIO BoosterPack to select PWM or SPI DAC when changing between projects.

```
#define DEMO 1

#if DEMO == 1
#include "demo_gettysburg.h"
#define AUDIOSIZE (uint32_t )__gettysburg10_wav_size
#define AUDIODATA __gettysburg10_wav
#endif
#if DEMO == 2
#include "demo_8bit_8kHz.h"
#define AUDIOSIZE (uint32_t )__pcm_mono_8_bit_8kHz_wav_size
#define AUDIODATA __pcm_mono_8_bit_8kHz_wav
#endif
#if DEMO == 3
#include "demo_chirp_16kHz.h"
#define AUDIOSIZE (uint32_t )__chirp_16kHz_wav_size
#define AUDIODATA __chirp_16kHz_wav
#endif
#if DEMO == 4
#include "demo_chirp_8kHz.h"
#define AUDIOSIZE (uint32_t )__chirp_8kHz_wav_size
#define AUDIODATA __chirp_8kHz_wav
#endif
#if DEMO == 5
#include "demo_16bit_8kHz.h"
#define AUDIOSIZE (uint32_t )__pcm_mono_16_bit_8kHz_wav_size
#define AUDIODATA __pcm_mono_16_bit_8kHz_wav
#endif
#if DEMO == 6
#include "demo_8bit_48kHz.h"
#define AUDIOSIZE (uint32_t )__tone_wav_size
#define AUDIODATA __tone_wav
#endif
```

6.2 Examples: audio_playback_internal_wavefile_by_PWM_EKTM4C123 and audio_playback_internal_wavefile_by_SPIDAC_EKTM4C123

These two examples are similar to the above examples. However, these two examples are to be run on the EK-TM4C123GXL LaunchPad. Press the USR_SW1 (LEFT) button on the board to pause the playback. Press the USR_SW2 (RIGHT) button to resume playback.

6.3 Examples: audio_playback_internal_wavefile_by_PWM and audio_playback_internal_wavefile_by_SPIDAC

These two examples are to be run on the DK-TM4C129x development board. The selected audio will be playbacked repeatedly. Press the DOWN button on the board to pause the playback. Press the UP button to resume playback.

6.4 Examples: audio_playback_with_sdcard_by_PWM and audio_playback_with_sdcard_by_SPIDAC

These two example applications demonstrate playing .wav files stored on an SD card that is formatted with a FAT file system. These two examples are ready to run on the DK-TM4C129x development board where a SD card slot is populated. The application can browse the file system on the SD card and displays all files that are found on the LCD screen. Files can be selected on the LCD screen to show their format and then played if the application determines that they are a valid .wav file.

This application note does not provide any .wav files as collaterals. Only PCM format (uncompressed) .wav files may be played. Users are free to load any compatible .wav files of their choice onto a SD card and then insert it to the SD card slot on the development board.

7 Download and Import the Examples

There are eight Code Composer Studio™ CCS project examples attached as collateral to this document. They can be unzipped into a local directory or kept in the zip format. Both formats can be imported to the CCS.

1. To import the project into CCS, first select the “File” -> ”Import”.

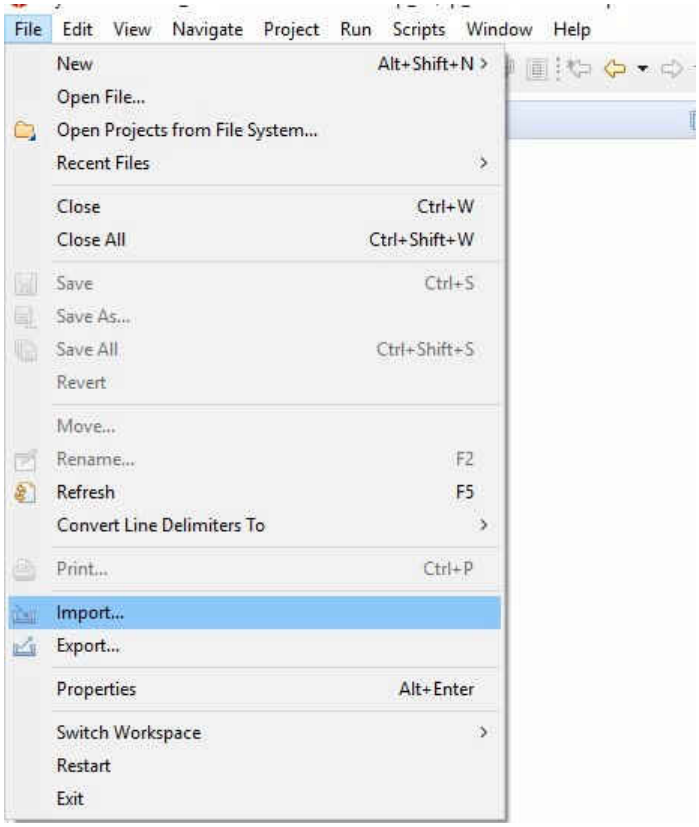

Figure 7-1. Import CCS Projects Step 1

2. Select “CCS Projects” to import the examples and then click “Next”.

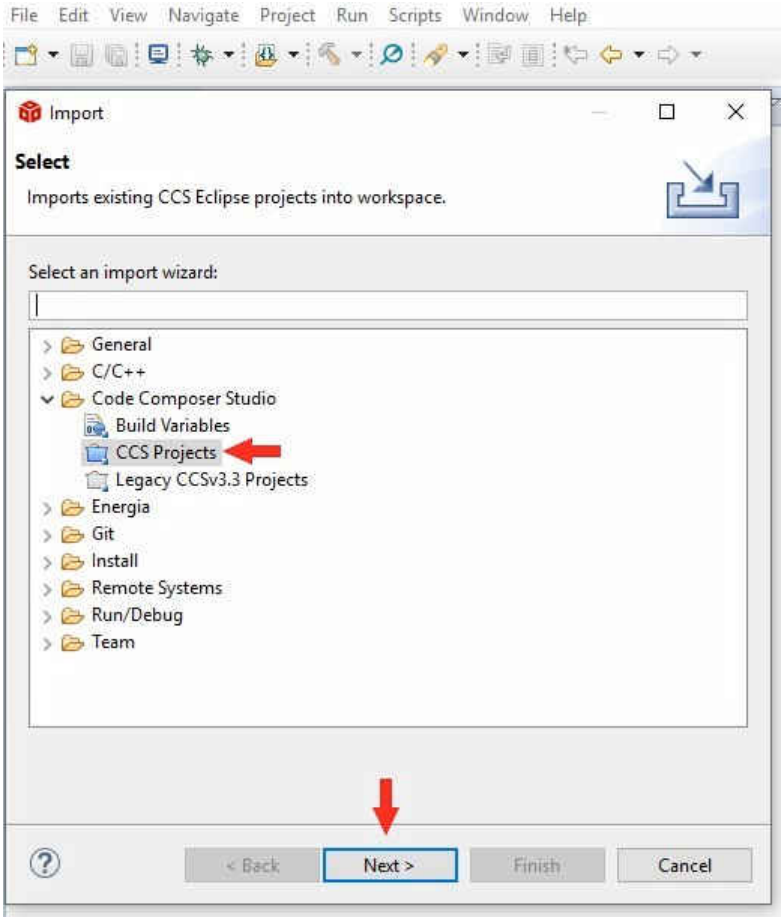

Figure 7-2. Import CCS Projects Step 2

3. Next, provide the path to either the unzipped project by selecting the first radio button or import the zip file directly by selecting the second radio button. Click the “Copy projects into workspace”.

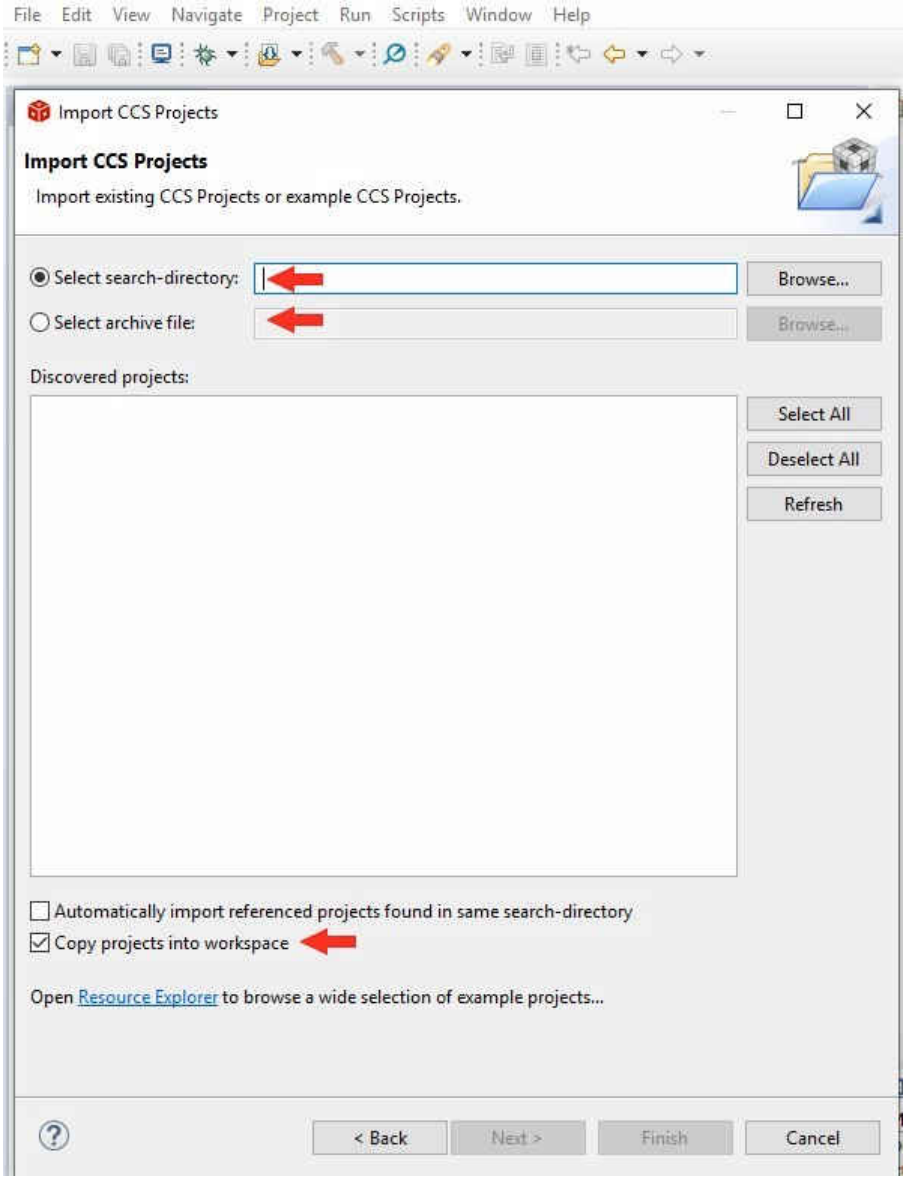

Figure 7-3. Import CCS Projects Step 3

4. After the project path is provided, a total of eight discovered projects will show up. First click “Select All” button and then click the “Finish” button to complete the import.

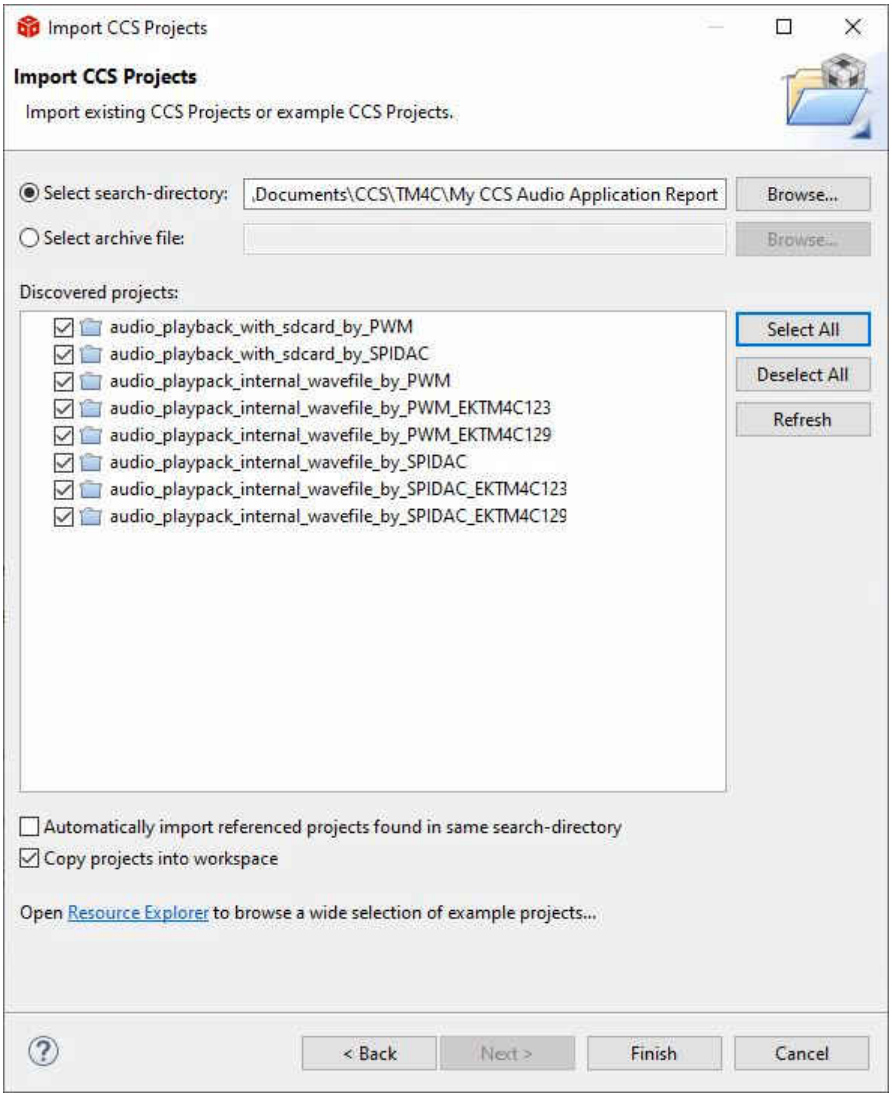

Figure 7-4. Import CCS Projects Step 4

8 References

- Texas Instruments: *Tiva™ C Series TM4C1294 Connected LaunchPad Evaluation Kit User's Guide*
- Texas Instruments: *Tiva™ C Series TM4C123G LaunchPad Evaluation Board User's Guide*
- Texas Instruments: *Tiva™ TM4C129X Development Board User's Guide*
- Texas Instruments: *BOOSTXL-AUDIO Audio BoosterPack™ Plug-in Module*
- Texas Instruments: *Tiva™ TM4C1294NCPDT Microcontroller Data Sheet*
- Texas Instruments: *TivaWare™ Peripheral Driver Library User's Guide*
- Bin2c: *Download*

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265