Application Report
SLOA182 – February 2013

Digital ANC Programmer Guide

Jorge Arbona *Audio and Imaging Products/Audio Converters*

ABSTRACT

This document describes the recommended software workflow for TI's Digital ANC solution.

WARNING: EXPORT NOTICE

Recipient agrees to not knowingly export or re-export, directly or indirectly, any product or technical data (as defined by the U.S., EU, and other Export Administration Regulations) including software, or any controlled product restricted by other applicable national regulations, received from Disclosing party under this Agreement, or any direct product of such technology, to any destination to which such export or re-export is restricted or prohibited by U.S. or other applicable laws, without obtaining prior authorization from U.S. Department of Commerce and other competent Government authorities to the extent required by those laws. This provision shall survive termination or expiration of this Agreement.
According to our best knowledge of the state and end-use of this product or technology, and in compliance with the export control regulations of dual-use goods in force in the origin and exporting countries, this technology is classified as follows:

US ECCN: 3E991
EU ECCN: EAR99

And may require export or re-export license for shipping it in compliance with the applicable regulations of certain countries.

Contents

Figures

Tables

1 General Overview

TI's Digital Active Noise Canceller (DANC) for headphones brings extra flexibility that allows on-the-fly tuning of ANC parameters, music equalization (EQ), as well as production-line tuning (PLT) automation. The DANC is a system solution which consists of the following essential blocks:

- An *ANC* block
- A *Music EQ* block
- A *Microcontroller* block
- An *ANC Tuning* block

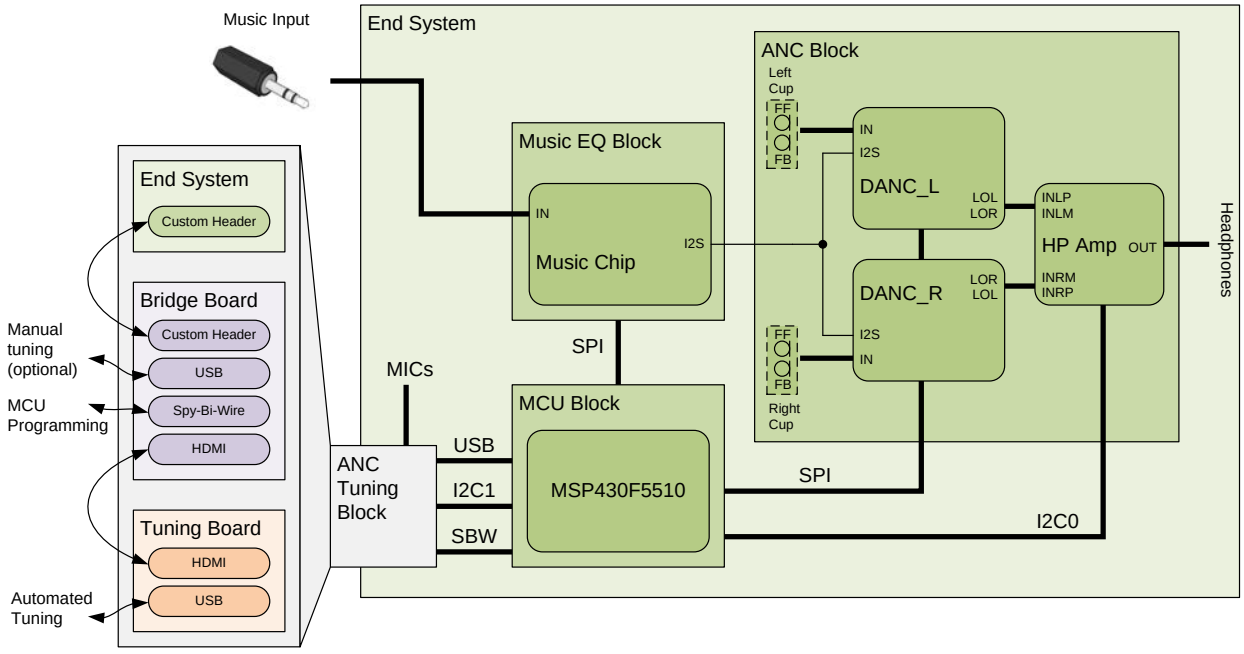

Figure 1. DANC System Solution

The *End System* consists of the *ANC*, *Music* and *MCU blocks* and typically resides in a PCB inside the ANC headset. The *ANC Tuning block* has additional collateral used for ANC and music tuning.

Many hardware configurations are possible. Unless otherwise noted, this document assumes a Hybrid DANC system in which two DANC chips are used as shown in Figure 1. In this diagram, the MSP430 controls the *ANC block* through SPI and I2C0. The I2C0 bus can be used to connect to other peripherals as well; the MSP430 is the *I2C master* in this case.

The I2C1 bus is used to communicate data between the *ANC Tuning block* and the MSP430, in which the MSP430 is the *I2C slave*. On a typical end system, the *ANC Tuning Block* consists of three separate boards: the *End System*, the *ANC Bridge Board* and the *ANC Tuning Board*. The *DANCEVM-A* evaluation board merges the *End System* and *Bridge Board* blocks into a single PCB for easier evaluation.

2 Tools

This section covers the recommended hardware and software tools as well as the *DANC Software Package*.

Physical production line test fixture hardware recommendations as well as production line tuning software are provided on a separate document.

2.1 Hardware Tools

Below is a list of recommended hardware collateral.

- **DANCEVM-A** – Digital ANC EVM-A. Used for initial prototyping, development, and evaluation. Can be obtained from TI.
- **MSP430FET430UIF** – MSP430 Flash Emulation Tool USB Interface. Used to program the DANCEVM-A or End System (through Bridge Board). Can be obtained here: http://www.ti.com/tool/msp-fet430uif.
- **DANC Bridge Board Reference Design** – Used to interface the *End System* PCB to the *ANC Tuning Board* and debug through Spy-Bi-Wire and USB. Intended for *Production Line Tuning (PLT)*. Schematic files are provided in PDF format and Mentor Graphics Design Capture.
- **DANC Tuning Board Reference Design** – Connects to the DANC Bridge Board and is used to automate DANC tuning.
- **Headphone Test Fixture and Collateral**

2.2 Software Tools

Below is a list of recommended tools to download before starting evaluation.

- **MSP430 Software Tools**
  - **Code Composer Studio** – for MSP430 firmware debugging and programming. Download here: http://www.ti.com/tool/ccstudio.
  - **USB Field Firmware Updater** – from the MSP430 USB Developers Package. Includes executable and source code to easily download MSP430 firmware code through USB. Download here: http://www.ti.com/tool/msp430usbdevpack.
  - **SmartRF Flash Programmer (optional)** – useful tool to download MSP430 hex firmware images without using CCS. Download here: http://www.ti.com/tool/flash-programmer.
- **Audio Software Tools**
  - **PurePath Studio GDE (Portable Audio)** – generates and downloads miniDSP code into AIC devices. Download here: http://www.ti.com/tool/aicpurepath_studio.
  - **AIC3254 CS** – AIC3254 Control Software. Download here: http://www.ti.com/tool/tlv320aic3254evm-k.

- **AIC3256 CS** – AIC3256 Control Software. Download here: http://www.ti.com/tool/tlv320aic3256evm-u.

2.3 The DANC Package

The DANC_Package_vX_X_X.zip file consists of the following:

- **\Documents** – guides, schematics, and so forth
  - **DANC Programmer Guide vX_X_X.pdf** – this document
  - **DANCEVM-A User Guide vX_X_X.pdf** – User Guide for the DANCEVM-A evaluation module
  - Schematics
- **\Components** – PurePath Studio ANC components. After installing PurePath Studio, copy its files to *"[User]\Documents\Texas Instruments Inc\PurePath Studio…\UserComponentLibrary"*
- **\ProcessFlows** – sample PurePath Studio miniDSP code
- **\MSP430_Code** – sample MSP430F5510 firmware code for DANC

The “\ProcessFlows” directories are structured as shown below. Below are some examples as well.

.\ATHY1-HYBRID_2

.\ATHY1-HYBRID_2\ATHY1-MUS-12.0000-3BQ-NiceEQ

.\ATHY1-HYBRID_2\ATHY1-DNC-12.0000-B-HY-30dB_Atten-4FF-2FB

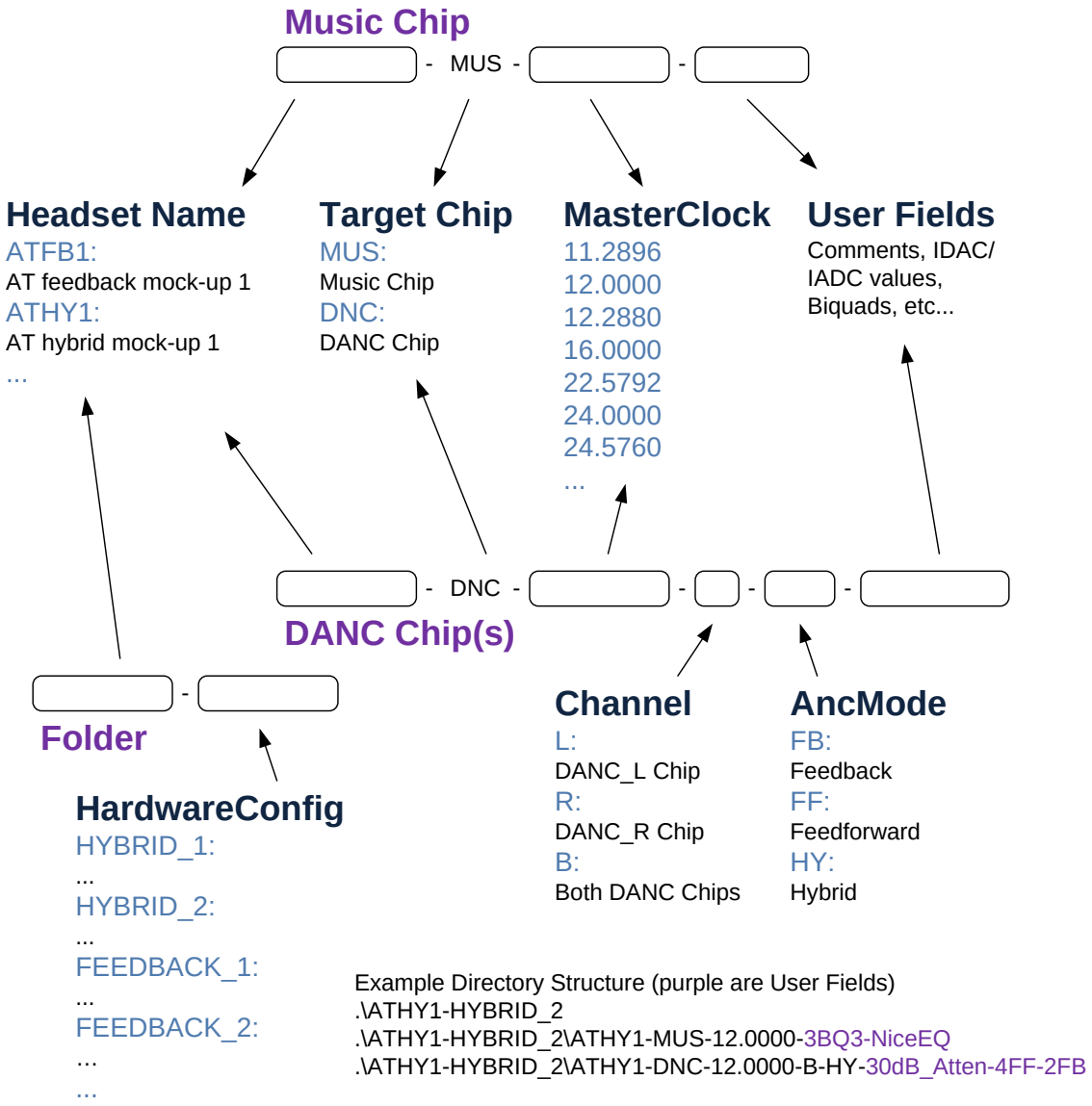

Figure 2. Process Flow Naming Convention

3 Step-by-Step Code Development Process

To get familiarized with the code development process, it is recommended to start with this process with the DANCEVM-A board.

1. Configure the DANCEVM-A board as described in its User Guide to enable Spy-Bi-Wire debugging. If using a custom board, ensure proper connections are in place for debugging.
2. In Code Composer Studio, import the DANC project found in the DANC Package.
3. Configure the */Device* folder.
    a. Choose a PCB board in */Device/Device.h* as explained in Section 4.1 and Figure 5. By default, the code is set up for the DANCEVM-A. Now is a good moment to try downloading the code into the DANCEVM-A board. First ensure the correct revision is chosen and then compile and download the code.
    b. Verify the configuration in */Device/Device_clocks.c* matches the hardware. No change should be necessary if using the DANCEVM-A, or if the custom board has a 24-MHz crystal connected to XT2 and the audio clock is output through P1.0. If the crystal is different than 24 MHz, then modify USB_XT_FREQ in */USB_config/descriptors.h*.
    c. Configure the device initialization and ports in */Device/Device.c*. Here is where defaults are set and I/O ports are configured. This does not need to be changed if working with the DANCEVM-A.
    d. Configure which P1 ports (for example, push buttons) trigger interrupt events in */Device/Device_interrupts.c*. This does not need to be changed if working with the DANCEVM-A.
    e. Configure device events in */Device/Device_eventHandler.c*. Here is where push-button events are handled and API events are serviced. This does not need to be changed if working with the DANCEVM-A.
4. Generate a PurePath Studio Header file as explained in Section 5.1. For the first test, it is recommended to keep ANC off as the default by programming the feed-back and feed-forward coefficients to zero. This step can be skipped if using the pre-loaded configuration.
5. Configure /AUDIO_config.
    a. Select the desired ANC configuration in *AUDIO_config/AudioLoad.h* (for example, *ANC1_ENABLED* if using the pre-loaded configuration or *MYBOARD1_ENABLED* if using a custom configuration).
    b. Create a new folder in */AUDIO_config* or use */AUDIO_config/MyBoard1* as a template to place the generated PurePath header files and audio configuration. There are two sample configurations already provided as well.
    c. Integrate the PurePath header file into the MSP430 code as explained in Section 5.2.

d. Add any additional device configurations in */AUDIO_config/xxxx/xxxx_Configs.h*.

e. Configure the Production Line Tuning (PLT) API. For the first run this is not required since, by default, the MSP430 information memory defaults to ignore PLT settings. This changes, however, once the EVM/device is programmed by the ANC Tuning Board.

   i. Configure the PLT API to map the PLT memory to the audio device C-RAM, as explained in Section 6.1.

   ii. Configure PLT_001 as explained in Section 6.1. This mode basically outputs the music input into the headphone outputs, while sending the microphone signals through the PLT connector.

f. Configure */AUDIO_config/AudioLoad.c* as explained in Section 6.1.

6. Connect an audio source to the Music Chip input and headphones to the headphone output connector (options are described in DANCEVM-A User Guide if using this EVM). The headphones can be of any kind for initial testing. The microphone inputs are pulled low in the DANCEVM-A, so the microphone inputs can be left disconnected if desired. Play music before wearing headphones.

7. Follow the steps in the Tuning Board documentation for actual headphone tuning.

4 DANC MSP430 Software Stack

The end application is a headset in which a user controls the ANC feature. This means loading code to the audio devices based on button presses, jack insertion and removal, and so forth, which makes an interrupt-driven architecture a good fit.

There are two additional modes in which the device may operate: a *Production Line Tuning (PLT)* mode and a *USB Tuning* mode. If MSP430 flash memory space permits or if desired, these two modes can be stored in the production code, but are not necessary for the end application.

The *PLT* mode is activated upon receiving data from the USCI_B1 I2C line. These data are mapped into the MSP430 information memory (non-volatile flash accessible by ROM code) by the MSP430. Certain commands sent at certain locations will set the audio devices into special tuning modes. More details are covered in Section 6 and Appendix A.

The USB tuning mode is activated when a USB host is connected to the device. This allows on-the-fly tuning of the DANC and Music chips using the PurePath Studio GDE software.

The suggested MSP430 software layer stack is shown below.

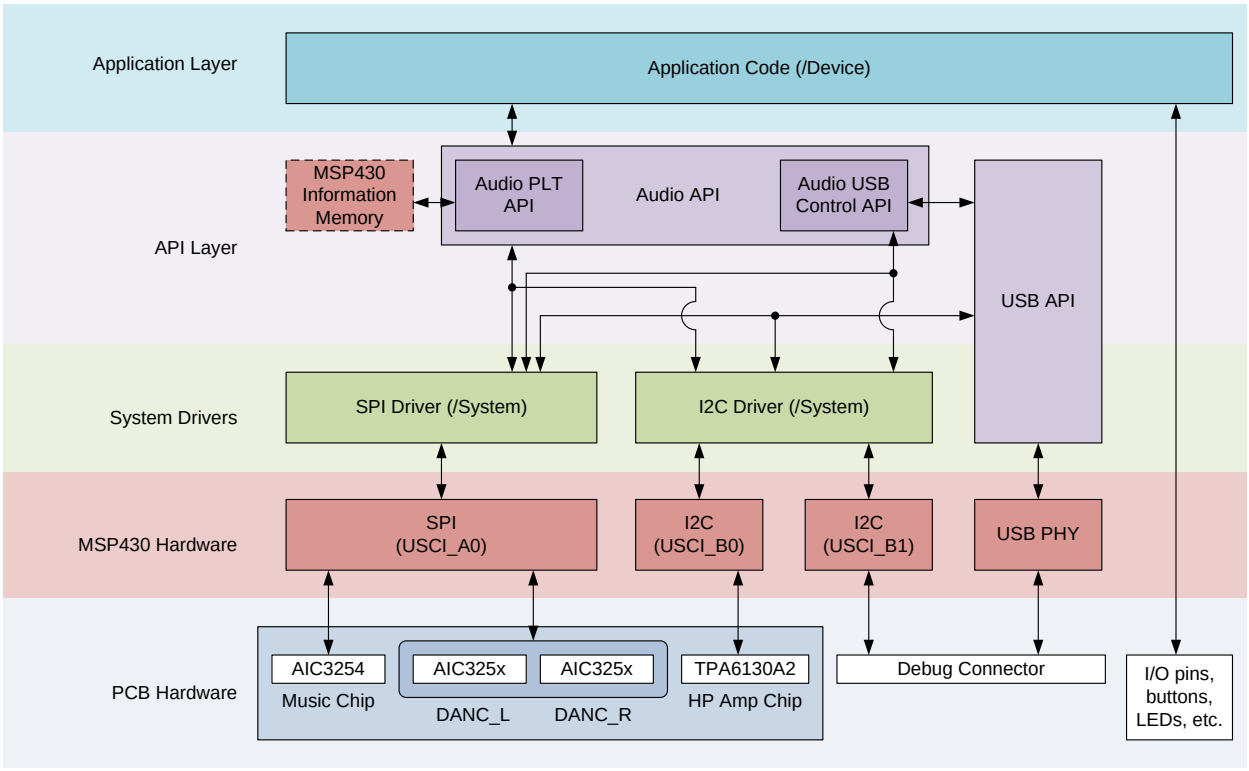

Figure 3. Software Layer Stack-up

The *Application Code* communicates with the *Audio API* to enable specific functions of the audio chip. It also has access to the *I2C Driver* and other miscellaneous hardware functions. Power management is handled in the *Application Code*.

For example, if a hypothetical *ANC On* switch is pressed, this would trigger an interrupt which the *Application Code* would service by calling an *Audio API* function that downloads either coefficients, miniDSP code, or both.

The *Audio USB Control API* and *USB API* work autonomously from the *Application Code* to allow on-the-fly tuning using PurePath Studio GDE. The same is true for the *Production Line Tuning* that occurs over USCI_B1.

4.1 Sample Code

The *Digital ANC Package* includes sample code that follows the architecture in the previous section. Each function and source file is well documented within the code itself. To keep this guide concise, it will not explain how all functions work, rather, it will go through examples and use cases.

The code is organized in various subfolders based on specific device functions as shown below:

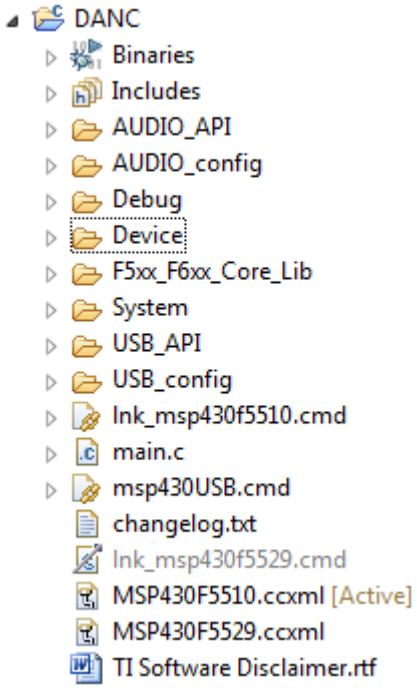

Figure 4. DANC Source Code Organization

The root folder contains *main.c* which has the first function that is executed. The */Device* and */System* folders contain application code that is meant to be modified by the end-user. The */AUDIO_API*, */USB_API* and */F5xx_F6xx_Core_Lib* folders are built-in APIs and each has a */xxxx_config* user-configurable folder. Only the AUDIO_API is discussed in this document. The USB_API is a slightly modified version of the one found in the MSP430 USB Developer's Package (see Section 2.2).

Each module (folder) contains a module_name_**types.h** file. This is meant to be included only within its API and its config folder. It should not be included elsewhere.

The first step when working with this code is to select a board in */Device/Device.h*. The code is configured to work with the DANCEVM-A by default. Note that the board revision should be selected to match the DANCEVM-A board. The board revision is shown close to the TI logo silkscreen in the board. When working on a custom PCB, it is recommended not to edit the *BOARD_DANCEVM_A_REVx* settings. A user-editable sample board configuration is provided as *BOARD_CUSTOM_1*.

Figure 5 shows a */Device/Device.h* snippet.

```
// Board selection
//#define     BOARD_DANCEVM_A_REVA
//#define     BOARD_DANCEVM_A_REVB
#define     BOARD_DANCEVM_A_REVC
//#define     BOARD_CUSTOM_1

// Audio control signals
#define TPA_nSD(x)          ( (x != 0) ? (P5OUT |= BIT4) : (P5OUT &= ~BIT4) )
#define AUD_nRESET(x)       ( (x != 0) ? (P5OUT |= BIT5) : (P5OUT &= ~BIT5) )

// SPI Slave Select lines

#if (defined BOARD_DANCEVM_A_REVA || defined BOARD_DANCEVM_A_REVB)
    #define SPI_SS1(x)          ( (x != 0) ? (P4OUT |= BIT3) : (P4OUT &= ~BIT3) )
    #define SPI_SS2(x)          ( (x != 0) ? (P4OUT |= BIT4) : (P4OUT &= ~BIT4) )
    #define SPI_SS3(x)          ( (x != 0) ? (P4OUT |= BIT5) : (P4OUT &= ~BIT5) )
#elif (defined BOARD_DANCEVM_A_REVC)
    #define SPI_SS1(x)          ( (x != 0) ? (P6OUT |= BIT1) : (P6OUT &= ~BIT1) )
    #define SPI_SS2(x)          ( (x != 0) ? (P6OUT |= BIT2) : (P6OUT &= ~BIT2) )
    #define SPI_SS3(x)          ( (x != 0) ? (P6OUT |= BIT3) : (P6OUT &= ~BIT3) )
#elif (defined BOARD_CUSTOM_1)
    #define SPI_SS1(x)          ( (x != 0) ? (P6OUT |= BIT1) : (P6OUT &= ~BIT1) )
    #define SPI_SS2(x)          ( (x != 0) ? (P6OUT |= BIT2) : (P6OUT &= ~BIT2) )
    #define SPI_SS3(x)          ( (x != 0) ? (P6OUT |= BIT3) : (P6OUT &= ~BIT3) )
#endif
```

Figure 5. Board-Specific Settings (/Device/Device.h)

4.1.1 AUDIO_API

The **AUDIO_API** file organization is shown below.

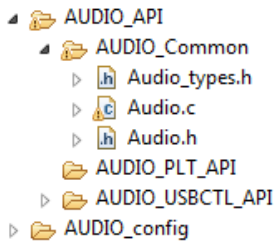

Figure 6. AUDIO_API File Organization

It consists of the following modules:

- **AUDIO_Common** – functions that allow communication between the *Application Code* layer and the audio devices.
- **AUDIO_PLT_API** (Audio Production Line Tuning API) – production line tuning functions.

- **AUDIO_USBCTL_API** (Audio USB Control API) – handles communication between the USB host (that is, PC) and the audio devices.
- **AUDIO_config** – Audio API user configuration.

5 MiniDSP Code Loading Process

Care has been taken to ensure that the miniDSP code loading process is as seamless as possible. This document only discusses loading miniDSP in flash. On-the-fly tuning using PurePath Studio is discussed in the DANCEVM-A User Guide.

5.1 DANC Header File Generation

The miniDSP code that would be loaded into the MSP430 is generated by the PurePath Studio GDE software. This tool can output the miniDSP code in a variety of ways. For the DANC system, the simplest is using header (.h) files.

The DANC Package includes sample process flows as discussed earlier. Each example has miniDSP code for the Music and DANC chips.

Clicking the *Framework* of a *Process Flow* will show its properties in the Properties windows to the right.

Figure 7. Music and DANC Frameworks

The *Framework* component (blue icons in Figure 7) provides several options for code generation. Below is a screenshot of Music and DANC Framework Properties. Based on these properties, the register settings found in the **SystemSettingsCode** property are executed to match the settings required for the EVM board. Conditional statements add or remove register sections based on the desired configuration.

SystemSettingsCode	**; Music Chip Config**
TargetBoard	**DANCEVM_A**
BoardRevision	**C**
TargetMode	**USB**
DeviceMode	**miniDSP**
Device	**AIC3254**

Figure 8. Music Framework Properties

SystemSettingsCode	; DANC Chip Config
TargetBoard	DANCEVM_A
BoardRevision	C
TargetMode	USB
AncMode	Hybrid
AnalogOutMode	LO_SE
Device	AIC3256

Figure 9. DANC Framework Properties

The **TargetMode** property is meant to be used to either load the complete code into the devices or just load a portion of the code. In the default **SystemSettingsCode**, three options are used: *USB, Init* and *Routing*. For USB Debugging simply use *USB*. *Init* and *Routing* are described in Section 5.2.

The user can edit the **SystemSettingsCode** conditional statements, (marked in blue below) to configure the device appropriately. For example, **TargetBoard** can be labeled *"MyBoard"*. In such case, the code below will be parsed into the output header file when compiling. Note that this code also looks at the **miniDSP_D_Cycles** and **miniDSP_A_Cycles** properties to configure the clock dividers (which will save additional power). If **TargetMode** = USB, all audio devices will be powered. If **TargetMode** = Init, then the audio devices are kept in a powered-down state, which allows writing PLT coefficients before powering up the devices and leaves routing and power-up to be handled by the MCU.

```
;-------------------------------------------------------------------------------
; MyBoard Target Only
;-------------------------------------------------------------------------------
%%if ("%%prop(TargetBoard)" == "MyBoard")
; MyBoard configuration

        ;-------------------------------------------------------------------------
        ; Clock and Interface Configuration
        ;-------------------------------------------------------------------------

        ; MyBoard case (assumes 24MHz clock)
        %%if (%%prop(miniDSP_D_Cycles) <= 32 && %%prop(miniDSP_A_Cycles) <= 32)
                        ; For IDAC <= 32 and IADC <= 32
                        reg[  0][ 11] = 0x02       ; NDAC = 2, divider powered off
                        reg[  0][ 12] = 0x04       ; MDAC = 4, divider powered off
                        reg[  0][ 18] = 0x02       ; NADC = 2, divider powered off
                        reg[  0][ 19] = 0x08       ; MADC = 8, divider powered off
        %%else
                        ; For IDAC > 32 or IADC > 32
                        reg[  0][ 11] = 0x01       ; NDAC = 1, divider powered off
                        reg[  0][ 12] = 0x08       ; MDAC = 8, divider powered off
                        reg[  0][ 18] = 0x01       ; NADC = 1, divider powered off
                        reg[  0][ 19] = 0x10       ; MADC = 16, divider powered off
        %%endif
        reg[  0][ 13] = 0x00       ; DOSR
        reg[  0][ 14] = 0x08       ; DOSR = 8
        reg[  0][ 20] = 0x04       ; AOSR = 4

%%endif
```

Figure 10. Custom TargetBoard SystemSettingsCode Snippet

To enable header file generation in PurePath Studio, go to Tools>Options>Build. Ensure the option marked in Figure 11 is enabled.

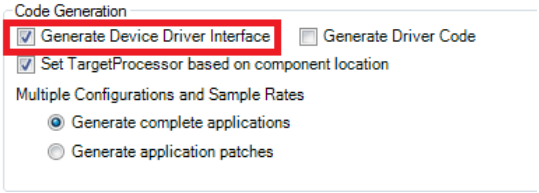

Figure 11. Header File Generation Option

To compile the miniDSP code and generate the header file, navigate to the option shown in Figure 12.

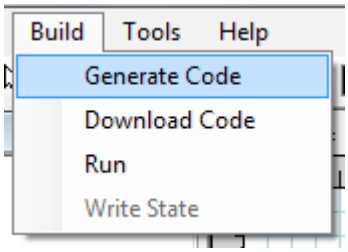

Figure 12. Generate Code

This generates an .h file in the working directory. This file contains 3 important arrays of struct: REG_Section_program[], miniDSP_A_reg_values[] and miniDSP_D_reg_values[]. These are then downloaded by the *Audio API* in the MSP430 code.

```
reg_value REG_Section_program[] = {
// Initialize the device through software reset
    {  0,0x00},
    {  1,0x01},
    {254,0x0A},
    {255,0x00},
    {255,0x01},
    {  0,0x00},
.
.
// NADC = 1, divider powered on
    { 18,0x01},
// MADC = 16, divider powered on
    { 19,0x10},
// DOSR
    { 13,0x00},
// DOSR = 8
    { 14,0x08},
// AOSR = 4
    { 20,0x04},
};

reg_value miniDSP_A_reg_values[] = {
    {  0,0x08},
    {  8,0x01},
    {  9,0x10},
    { 10,0x3D},
.
.
};

reg_value miniDSP_D_reg_values[] = {
    {  0,0x2C},
    {  8,0x00},
    {  9,0x8E},
    { 10,0x21},
    { 11,0x00},
    { 12,0x00},
.
.
};
```

Figure 13. Generated Header File

The first byte of the struct is the register offset. The second byte is the data byte. Note that 3 lines in the code above are marked in purple. These lines have reserved numbers that the MCU code uses.

A **'254'** means a delay in ms. In this case, this instructs the MCU to delay for 10 ms after software reset.

A **'255'** means 'jump to another array of struct'. The line with the value of *0x00* is instructing the MCU to jump to *miniDSP_A_reg_values[]*. The line with the value of *0x01* should link to *miniDSP_D_reg_values[]*.

With the proper MCU code and with the above considerations in place, downloading *REG_Section_program[]* will program the device and its miniDSP.

To allow a seamless user experience, there are special numbers that are used to instruct the MSP430 microcontroller in the DANCEVM-A which device(s) it should download the code to.

These numbers allow:

- A **Forced Write** (252 / 0xFC)
- A **Linked Write** (253 / 0xFD)

The MSP430 enumerates as 4 HID devices. Each "device" has a **Device Index** (devIndex) that maps to the Music Chip (0), DANC_L Chip (1), DANC_R Chip (2) and HP Amp Chip (3).

Once a **Forced Write** is engaged, only the devices in Forced Write bit mask will be written.

Once a **Linked Write** is engaged, the same data will be written to all the devices in the Linked Write bit mask only if PurePath (or the device Control Software or MSP430 code) is communicating to a device in the Linked Write mask. A Linked Write is very useful when writing the same filter coefficients or gain settings to both DANC chips.

A Forced Write has precedence over a Linked Write.

An example of this concept is shown in Figure 14.

```
; The lines below engage a linked write to all AIC chips. This will effectively reset
; all devices (if communicating with either devIndex 1, 2 or 3) with a single write.
reg[  0][253] = 0x07 ; Link Write Music Chip / DANC_L / DANC_R (USB devIndex = 0/1/2)
reg[  0][  1] = 0x01 ; Initialize the device through software reset
reg[  0][253] = 0x00 ; Disable Linked Write

; The lines below engage a forced write to the HP Amp Chip. Note that the mask is 0x08,
; which means it will only write to devIndex = 3, regardless of the current devIndex.
reg[  0][252] = 0x08 ; Force Write to HP Amp Chip (USB devIndex = 3)
reg[  0][  1] = 0xC1 ; Mute TPA6130A2
reg[  0][252] = 0x00 ; Disable Forced Write
```

Figure 14. Forced Write Precedence Example

5.2 Loading a PurePath Header File into the Audio API

The AUDIO_config folder of the MSP430 code contains custom miniDSP header files. To generate a header compliant with the Audio API, ensure that the **TargetMode** property of both Music and DANC Process Flows is set to **Init**. This will configure the devices without routing, ADC/DAC power-up, and so forth. Once the code is compiled, the header file data (as shown in Figure 13) should be pasted in MusicChip_Code.h (Music Chip) and in DancChip_Code.h (DANC Chips).

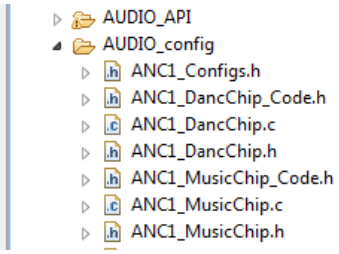

Figure 15. AUDIO_config File Organization

Small changes need to be done to the PurePath output header file when integrating into the MSP430 code:

1. Header guards should be added.
2. Audio_types.h should be included.
3. The reg_value name should be re-defined to be const so it is loaded into the MSP430 ROM and not RAM.

```
// (c)2012 by Texas Instruments Incorporated, All Rights Reserved.

#ifndef ANC1_DANCCHIP_CODE_H_
#define ANC1_DANCCHIP_CODE_H_

#include <AUDIO_API/AUDIO_Common/Audio_types.h>
#define reg_value static const struct reg_value_t

reg_value REG_Section_program[] = {
    {  0,0x00},
//          # reg[  0][  1] = 0x01  ; Initialize the device through software reset
    {  1,0x01},
    {254,0x0A},
    {255,0x00},
..
    {126,0xFF},
    {127,0x00},
};

#endif /* ANC1_DANCCHIP_CODE_H_ */
```

Figure 16. DANC miniDSP code header file (ANC1_DancChip_Code.h)

The miniDSP code array is loaded into a structure of type struct miniDspApp_t to make a single package that relates to a particular chip. This structure is shown in Figure 16.

```
struct miniDspApp_t ANC1_DancChip =
{
    REG_Section_program,
    sizeof(REG_Section_program)/sizeof(struct reg_value_t),
    miniDSP_A_reg_values,
    sizeof(miniDSP_A_reg_values)/sizeof(struct reg_value_t),
    miniDSP_D_reg_values,
    sizeof(miniDSP_D_reg_values)/sizeof(struct reg_value_t)
};
```

Figure 17. MiniDSP code source file (ANC1_DancChip.c)

This structure is then declared as an extern variable in its corresponding header file to be used by the Audio API.

Routing, sleep modes and other configurations can be loaded in **"ANC1_Configs.h"**. These configurations could be entered manually. The default SystemSettingsCode includes settings for routing, making it easy to use the same configuration that is used for USB debugging. To generate this code, set **TargetMode** of the **DANC framework** to **Routing** and compile the PurePath session. Copy the contents of the generated REG_Section_program[] and paste it into the desired struct array. Figure 17 provides an example snippet.

```
static const struct reg_value_t pANC1_ancOn[] =
{
    {  0,0x00},
//          # reg[  0][253] = 0x00  ; Linked Write disabled
    {253,0x00},
//          # reg[  0][252] = 0x00  ; Forced Write disabled
    {252,0x00},
    {  0,0x7F},
    {  0,0x00},
//          # reg[  0][253] = 0x06  ; Link Write DANC_L / DANC_R (USB devIndex = 1/2)
    {253,0x06},
    {  0,0x7F},
    {  0,0x00},

...

//          # reg[  0][253] = 0x00  ; Linked Write disabled
    {253,0x00},
//          # reg[  0][252] = 0x00  ; Forced Write disabled
    {252,0x00},
    {  0,0x7F},
};
```

Figure 18. Example Routing Configuration

So far, the *Init* and *Routing TargetMode* codes have been pasted into the MSP430 code. Figure 18 is example code usage of these structures as used in "*AudioLoad.c"*. Notice that:

1. A pointer to a *struct miniDspApp_t* is passed to *Audio_miniDspAppLoad()*. Here is where the Music Chip and DANC Codes are loaded.
2. PLT coefficients and mic gain are loaded (if necessary). This is discussed in Section 6.1.
3. Settings are loaded using the pANC1_ancOn structure pointer.

```
static int16_t ancOn(void)
{

    // Write miniDSP I-RAM, C-RAM and settings
    Audio_miniDspAppLoad(&ANC1_MusicChip, AUDIO_MUSIC_CHIP);
    Audio_miniDspAppLoad(&ANC1_DancChip, AUDIO_DANC_L_CHIP);

    // Load data from PLT Register Map
    AudioLoad(AUDIO_SET_MIC_GAIN);
    AudioLoad(AUDIO_SET_COEFFS);

    // Enable Audio
    Audio_regValueWrite(pANC1_ancOn,
                        sizeof(pANC1_ancOn)/sizeof(struct reg_value_t),
                        AUDIO_DANC_L_CHIP);

    return PASS;
}
```

Figure 19. Writing to the audio devices

6 DANC Production Line Tuning (PLT) Specification

As previously mentioned, the PLT mode is activated upon receiving data from the USCI_B1 I2C line. These data are mapped into the MSP430 information memory (non-volatile flash accessible from ROM code) by the MSP430. The MSP430F5510 has 512 bytes of information memory.

Table 1. MSP430 Information Memory

Information Memory (Flash)	Info A	128 B 0019FFh–001980h
	Info B	128 B 00197Fh–001900h
	Info C	128 B 0018FFh–001880h
	Info D	128 B 00187Fh–001800h

From an I2C perspective, the tuning memory area is accessed by writing registers and setting pages.

The MSP430 code will re-map these register commands to information memory as needed.

The I2C write protocol is defined as:

[S][I2C Slave Address/W][ACK][Register Offset][ACK][Data(0)][ACK]…[Data(n)][ACK][P]

The I2C master must perform ACK polling to allow the MSP430 time to process its data after each write. To perform ACK polling, send the I2C slave address along with a write command. If a NACK is returned as below, then issue a stop command (P).

[S][I2C Slave Address/W][NACK][P]

If an ACK is received, issue a stop and then proceed with the full command.

[S][I2C Slave Address/W][ACK][P]

A read is performed using repeated starts (Sr) by first writing the register offset and then issuing a read command.

[S][I2C Slave Address/W][ACK][Register Offset][ACK]

[Sr][I2C Slave Address/R][ACK][Data(0)][ACK]…[Data(n)][NACK][P]

A page is selected similar to an AIC device, by writing Register 0 of any page.

Page 0 contains configuration registers and flags. Pages 1 to 3 contain coefficients provided by the tuning process.

The DANC PLT Register Map is found in Appendix A.

6.1 Configuring the PLT API

The source file xxxx_*Plt.h* in *AUDIO_Config* can be edited to configure the PLT coefficient mapping as well as modifying the code for different PLT modes.

The struct reg_value_t array aPlt001[] contains the configuration for PLT_001 mode (see A.2.2). This is used to measure the headset transfer function. This needs to match the hardware configuration (for example, clocks, routing, and so forth).

In this file there are 3 struct arrays that configure the PLT coefficient mapping. An example for DANC_L is shown below. The first element is the tuning coefficient TCx (see Table 1 of Appendix A for an example layout). The second element is the miniDSP coefficient number Cx (see Figure 21). The third element is the target processor (miniDSP_A or miniDSP_D).

```
static const struct pltCoeffMap_t aANC1_pltDancLMap[] =
{
    // Feed-forward EQ
    { 50,  0, AUDIO_MINIDSP_D},      // DANC_L FF BQ0 N0
    { 51,  1, AUDIO_MINIDSP_D},      // DANC_L FF BQ0 N1
    { 52,  2, AUDIO_MINIDSP_D},      // DANC_L FF BQ0 N2
    { 53,  3, AUDIO_MINIDSP_D},      // DANC_L FF BQ0 D1
    { 54,  4, AUDIO_MINIDSP_D},      // DANC_L FF BQ0 D2
    { 55,  5, AUDIO_MINIDSP_D},      // DANC_L FF BQ1 N0
    { 56,  6, AUDIO_MINIDSP_D},      // DANC_L FF BQ1 N1
    { 57,  7, AUDIO_MINIDSP_D},      // DANC_L FF BQ1 N2
    { 58,  8, AUDIO_MINIDSP_D},      // DANC_L FF BQ1 D1
    { 59,  9, AUDIO_MINIDSP_D},      // DANC_L FF BQ1 D2
    // Feed-back EQ
    { 60,  0, AUDIO_MINIDSP_A},      // DANC_L FB BQ0 N0
    { 61,  1, AUDIO_MINIDSP_A},      // DANC_L FB BQ0 N1
    { 62,  2, AUDIO_MINIDSP_A},      // DANC_L FB BQ0 N2
    { 63,  3, AUDIO_MINIDSP_A},      // DANC_L FB BQ0 D1
    { 64,  4, AUDIO_MINIDSP_A},      // DANC_L FB BQ0 D2
    { 65,  5, AUDIO_MINIDSP_A},      // DANC_L FB BQ1 N0
    { 66,  6, AUDIO_MINIDSP_A},      // DANC_L FB BQ1 N1
    { 67,  7, AUDIO_MINIDSP_A},      // DANC_L FB BQ1 N2
    { 68,  8, AUDIO_MINIDSP_A},      // DANC_L FB BQ1 D1
    { 69,  9, AUDIO_MINIDSP_A},      // DANC_L FB BQ1 D2
    { 70, 10, AUDIO_MINIDSP_A},      // DANC_L FB BQ2 N0
    { 71, 11, AUDIO_MINIDSP_A},      // DANC_L FB BQ2 N1
    { 72, 12, AUDIO_MINIDSP_A},      // DANC_L FB BQ2 N2
    { 73, 13, AUDIO_MINIDSP_A},      // DANC_L FB BQ2 D1
    { 74, 14, AUDIO_MINIDSP_A},      // DANC_L FB BQ2 D2
    { 75, 15, AUDIO_MINIDSP_A},      // DANC_L FB BQ3 N0
    { 76, 16, AUDIO_MINIDSP_A},      // DANC_L FB BQ3 N1
    { 77, 17, AUDIO_MINIDSP_A},      // DANC_L FB BQ3 N2
    { 78, 18, AUDIO_MINIDSP_A},      // DANC_L FB BQ3 D1
    { 79, 19, AUDIO_MINIDSP_A},      // DANC_L FB BQ3 D2
};
```

Figure 20. PLT Mapping Example

This PLT map will effectively copy data from information memory to the audio device C-RAM. This is executed as shown in Figure 18 of Section 5.2.

To find a coefficient location for a component, click the component. The properties window will show the starting location of each biquad and its *TargetProcessor*. The block length is also shown when expanding the crosshair. Since a biquad has 5 coefficients, the first biquad below spans C0-C4 of the miniDSP_D. This correlates with the values found in Figure 20.

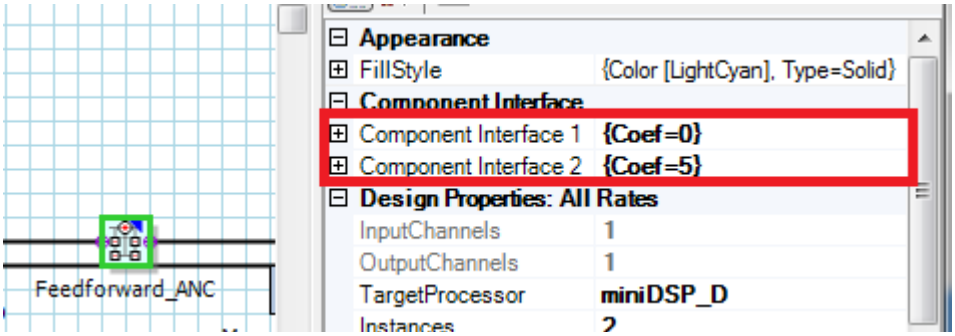

Figure 21. Feed-forward Coefficients Location in C-RAM

Appendix A. DANC PLT Register Map

The Digital ANC Production Line Tuning Register Map contains 4 pages of 8-bit registers. Each page corresponds to an *MSP430 Information Memory Segment*. The external PLT hardware sends I2C commands through USCI_B1 to program these virtual registers.

Pages 1 to 3 contain Tuning Coefficients (TC0-TC125). These coefficients are multi-purpose; it is up to the user to map these to the miniDSP or PRB C-RAM locations of the audio codec.

This configuration supports up to 25 biquad filters (5 coefficients per biquad) with 1 free extra coefficient.

An example TC layout for high performance ANC is shown in Table 1.

Table 2. Example Tuning Coefficient Layout

Device	NO. OF BIQUADS	DESCRIPTION
Music Chip	10	5 Music Left biquads (TC0-TC24) 5 Music Right biquads (TC25-TC49)
DANC_L Chip	6	2 Feedforward Biquads (TC50-TC59) 4 Feedback Biquads (TC60-TC79)
DANC_R Chip	6	2 Feedforward Biquads (TC80-TC89) 4 Feedback Biquads (TC90-TC109)

Each biquad has 5 coefficients labeled as N0, N1, N2, D1 and D2 and should match the AIC device biquad data format.

The page and base register location for each coefficient can be calculated as follows:

```
// Page Number Calculation -------------------------------
// Excel:
page(TCx) = FLOOR((TCx/42)+1, 1)
// C/C++:
page = (uint8_t) (TCx/42 + 1);
// Register Number Calculation ---------------------------
// Excel:
register(TCx) = MOD(TCx,42)*3 + 2
// C/C++:
reg = (TCx%42)*3 + 2;
```

Figure 22. Page / Register TCx Coefficient Calculation

A.1 Register Map Summary

Table 2 summarizes the DANC PLT register map.

Table 3. DANC PLT Register Map Summary

PAGE NO.	REG. NO.	DESCRIPTION
0 (0x00)	0 (0x00)	Page Select Register
0 (0x00)	1 (0x01)	PLT Mode Register
0 (0x00)	2 (0x02)	PLT Options Register
0 (0x00)	3 (0x03)	PLT Load Register
0 (0x00)	4 (0x04)	Reserved
0 (0x00)	5 (0x05)	Device Options Register
0 (0x00)	6 (0x06)	Reserved
	…	…
0 (0x00)	9 (0x09)	Reserved
0 (0x00)	10 (0x0A)	Left Feed-forward Microphone Analog Gain Register
0 (0x00)	11 (0x0B)	Left Feed-back Microphone Analog Gain Register
0 (0x00)	12 (0x0C)	Right Feed-forward Microphone Analog Gain Register
0 (0x00)	13 (0x0D)	Right Feed-back Microphone Analog Gain Register
0 (0x00)	14 (0x0E)	Reserved
	…	…
0 (0x00)	127 (0x7F)	Reserved
1 (0x01)	0 (0x00)	Page Select Register
1 (0x01)	1 (0x01)	Reserved
1 (0x01)	2 (0x02)	Tuning Coefficient TC0(23:16)
1 (0x01)	3 (0x03)	Tuning Coefficient TC0(15:8)
1 (0x01)	4 (0x04)	Tuning Coefficient TC0(7:0)
	…	…
1 (0x01)	125 (0x7D)	Tuning Coefficient TC41(23:16)
1 (0x01)	126 (0x7E)	Tuning Coefficient TC41(15:8)
1 (0x01)	127 (0x7F)	Tuning Coefficient TC41(7:0)
2 (0x02)	0 (0x00)	Page Select Register
2 (0x02)	1 (0x01)	Reserved
2 (0x02)	2 (0x02)	Tuning Coefficient TC42(23:16)
2 (0x02)	3 (0x03)	Tuning Coefficient TC42(15:8)
2 (0x02)	4 (0x04)	Tuning Coefficient TC42(7:0)
	…	…
2 (0x02)	125 (0x7D)	Tuning Coefficient TC83(23:16)
2 (0x02)	126 (0x7E)	Tuning Coefficient TC83(15:8)
2 (0x02)	127 (0x7F)	Tuning Coefficient TC83(7:0)
3 (0x03)	0 (0x00)	Page Select Register
3 (0x03)	1 (0x01)	Reserved
3 (0x03)	2 (0x02)	Tuning Coefficient TC84(23:16)
3 (0x03)	3 (0x03)	Tuning Coefficient TC84(15:8)
3 (0x03)	4 (0x04)	Tuning Coefficient TC84(7:0)

PAGE NO.	REG. NO.	DESCRIPTION
	…	…
3 (0x03)	125 (0x7D)	Tuning Coefficient TC125(23:16)
3 (0x03)	126 (0x7E)	Tuning Coefficient TC125(15:8)
3 (0x03)	127 (0x7F)	Tuning Coefficient TC125(7:0)

A.2 Page 0 Registers

A.2.1 Page 0 (0x00) / Register 0 (0x00): Page Select Register

BIT	READ / WRITE	RESET VALUE	DESCRIPTION
D7-D2	R/W	0000 00	Reserved. Write only reset values.
D1-D0	R/W	00	Page Select Register 00: Selects Page 0 of the Register Map (Segment D of MSP430 Information Memory). 01: Selects Page 1 of the Register Map (Segment C of MSP430 Information Memory). 10: Selects Page 2 of the Register Map (Segment B of MSP430 Information Memory). 11: Selects Page 3 of the Register Map (Segment A of MSP430 Information Memory). **NOTE:** The MSP430 Information Memory defaults as 0xFF per byte on the first boot-up. The firmware code should read this register and if it is 0xFF, it should then set the defaults for all PLT registers.

A.2.2 Page 0 (0x00) / Register 1 (0x01): PLT Mode Register

BIT	READ / WRITE	RESET VALUE	DESCRIPTION
D7	R/W	0	Reserved. Write only reset values.
D6-D0	R/W	111 1111	Device Mode 0: Normal Mode. 1: PLT_001 (Headphone Transfer Function Measurement Mode). 2-126: Reserved for future use. 127: Device has not been tuned yet.

A.2.3 Page 0 (0x00) / Register 2 (0x02): PLT Options Register

BIT	READ / WRITE	RESET VALUE	DESCRIPTION
D7-D2	R/W	1111 11	Reserved. Write only reset values.
D1	R/W	1	PLT microphone gain option 0: Use programmed PLT microphone gain settings. 1: Do not use programmed PLT microphone gain settings.
D0	R/W	1	PLT coefficients option 0: Use programmed PLT coefficients. 1: Do not use programmed PLT coefficients.

A.2.4 Page 0 (0x00) / Register 3 (0x03): PLT Load Register

BIT	READ / WRITE	RESET VALUE	DESCRIPTION
D7-D2	R/W	1111 11	Reserved. Write only reset values.
D1	R/W	1	Load Microphone Analog Gain (programs audio devices based on P0_R10-R13 setting). 0: Load programmed microphone analog gain(s). 1: MSP430 has finished configuring the microphone analog gain(s) (ready).
D0	R/W	1	Load PLT Mode (programs audio devices based on P0_R1 setting). 0: Load programmed PLT Mode. 1: MSP430 has finished configuring the device for the selected PLT Mode (ready).

A.2.5 Page 0 (0x00) / Register 4 (0x04): Reserved Register

BIT	READ / WRITE	RESET VALUE	DESCRIPTION
D7-D0	R/W	1111 1111	Reserved. Write only reset values.

A.2.6 Page 0 (0x00) / Register 5 (0x05): Device Options Register

BIT	READ / WRITE	RESET VALUE	DESCRIPTION
D7-D0	R/W	1111 1111	Reserved. Write only reset values.

A.2.7 Page 0 (0x00) / Register 6-9 (0x03-0x09): Reserved Registers

BIT	READ / WRITE	RESET VALUE	DESCRIPTION
D7-D0	R/W	1111 1111	Reserved. Write only reset values.

A.2.8 Page 0 (0x00) / Register 10 (0x0A): Left Feed-forward Microphone Analog Gain Register

BIT	READ / WRITE	RESET VALUE	DESCRIPTION
D7	R/W	1	0: Gain is controlled by bits D6-D0. 1: Gain is set to 0dB
D6-D0	R/W	000 0000	000 0000: Volume Control = 0.0dB 000 0001: Volume Control = 0.5dB 000 0010: Volume Control = 1.0dB … 101 1101: Volume Control = 46.5dB 101 1110: Volume Control = 47.0dB 101 1111: Volume Control = 47.5dB 110 0000-111 1111: Reserved. Do not use.

A.2.9 Page 0 (0x00) / Register 11 (0x0B): Left Feed-back Microphone Analog Gain Register

BIT	READ / WRITE	RESET VALUE	DESCRIPTION
D7	R/W	1	0: Gain is controlled by bits D6-D0. 1: Gain is set to 0dB
D6-D0	R/W	000 0000	000 0000: Volume Control = 0.0dB 000 0001: Volume Control = 0.5dB 000 0010: Volume Control = 1.0dB … 101 1101: Volume Control = 46.5dB 101 1110: Volume Control = 47.0dB 101 1111: Volume Control = 47.5dB 110 0000-111 1111: Reserved. Do not use.

A.2.10 Page 0 (0x00) / Register 12 (0x0C): Right Feed-forward Microphone Analog Gain Register

BIT	READ / WRITE	RESET VALUE	DESCRIPTION
D7	R/W	1	0: Gain is controlled by bits D6-D0. 1: Gain is set to 0dB

BIT	READ / WRITE	RESET VALUE	DESCRIPTION
D6-D0	R/W	000 0000	000 0000: Volume Control = 0.0dB 000 0001: Volume Control = 0.5dB 000 0010: Volume Control = 1.0dB … 101 1101: Volume Control = 46.5dB 101 1110: Volume Control = 47.0dB 101 1111: Volume Control = 47.5dB 110 0000-111 1111: Reserved. Do not use.

A.2.11 Page 0 (0x00) / Register 13 (0x0D): Right Feed-back Microphone Analog Gain Register

BIT	READ / WRITE	RESET VALUE	DESCRIPTION
D7	R/W	1	0: Gain is controlled by bits D6-D0. 1: Gain is set to 0dB
D6-D0	R/W	000 0000	000 0000: Volume Control = 0.0dB 000 0001: Volume Control = 0.5dB 000 0010: Volume Control = 1.0dB … 101 1101: Volume Control = 46.5dB 101 1110: Volume Control = 47.0dB 101 1111: Volume Control = 47.5dB 110 0000-111 1111: Reserved. Do not use.

A.2.12 Page 0 (0x00) / Register 14-127 (0x0E-0x7F): Reserved Registers

BIT	READ / WRITE	RESET VALUE	DESCRIPTION
D7-D0	R/W	1111 1111	Reserved. Write only reset values.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

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 relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI 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. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

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