Application Report
SLAA605A–October 2013–Revised October 2013

Using the MSP430 Launchpad as a Standalone I2C Host for Audio Products

Derek Xie and Dafydd Roche — *Audio and Imaging Products (AIP)*

ABSTRACT

This document and its associated files (MSP430 Audio Boot Code - SLAC640) give a foundation for customers to use the MSP430 Launchpad as a tool to evaluate and demonstrate TI Audio products, without the need to be connected to a computer.

This application code will take header files exported from TI Audio development tools and create standalone, bootable demo's – using the MSP430 as a host.

Contents

List of Figures

1 Introduction

For many, moving from an evaluation board and software to a standalone platform is a daunting task.

Getting a self-booting demonstration requires significant time and effort to generate the correct I2C script and generate the code for the microcontroller that will eventually boot your device.

This project removes many of those barriers by providing a framework that *simple wakes up, and boots over I2C*. This framework is designed to work on the MSP430 Launchpad, a $9.99 easy-to-use development tool.

Connection between the launchpad and the target hardware (your own hardware, or an evaluation board) is simple. You only need to connect the three wires and plug in the power. The framework also allows for a serial port terminal to be used to send additional I2C instructions after bootup, another useful debug point.

2 MSP430 Booting Demo Overview

The Audio Boot Platform requires two pieces of hardware. The MSP430 Launchpad (www.ti.com/lsds/ti/microcontroller/16-bit_msp430/overview.page) and the device under configuration (must be I2C controllable).

The code provides two kinds of the I2C configuration code. One is the GPIO_I2C (I2C over GPIO pins), the other is the USCI_I2C (I2C using the USCG peripheral).

The hardware connection only needs three wires, SCL, SDA and GND. The software development environment used is Code Composer Studio™.

The software project (downloaded from www.ti.com/tool/ccstudio-msp430) only has two files. One is the Main.c file, the other is the I2CBoot.h file.

Once your code is downloaded to the Launchpad, the LaunchPAD can self boot the demo through the I2C. You can also use the USB Serial port and the onboard Launchpad buttons to do additional debug and demos. Figure 1 gives a visual overview of the platform from both a hardware and software point of view.

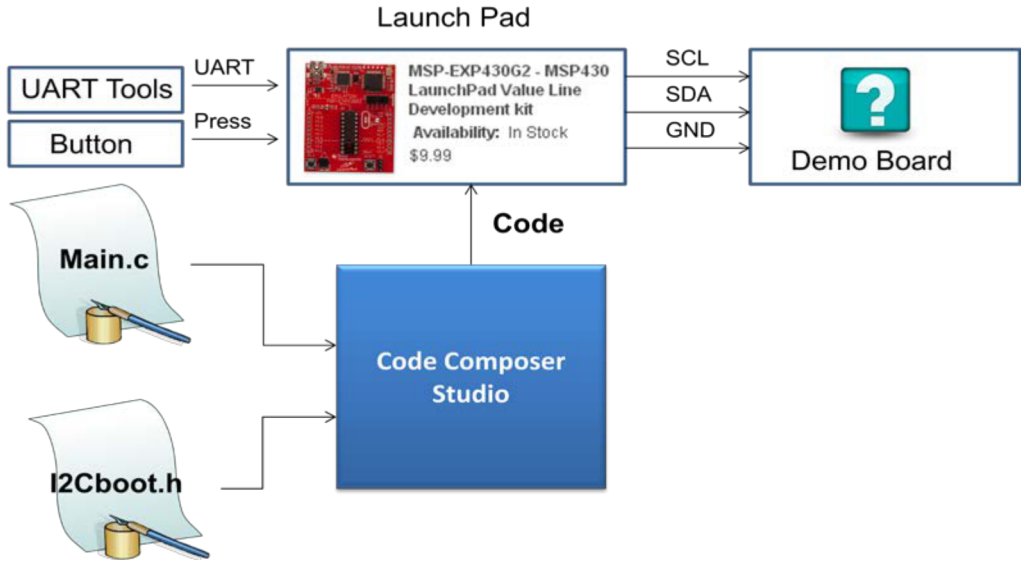

Figure 1. MSP430 Booting Demo Architecture

2.1 Main.c file

All the functions are defined in the Main.c file, like the I2C and UART.

I2C is driven either by GPIO Pins or by using the USCI peripheral integrated on the device. The I2C method is decided by the Parameter GPIO. (A variable in the code).

If the GPIO is 1, the I2C Init() is initialized by the GPIO_I2C function. If the GPIO is not 1, the I2C Init() is initialized by the USCI_I2C function.

NOTE: Using the USB Serial UART for additional debugging requires configuring the device to use GPIO mode for I2C.

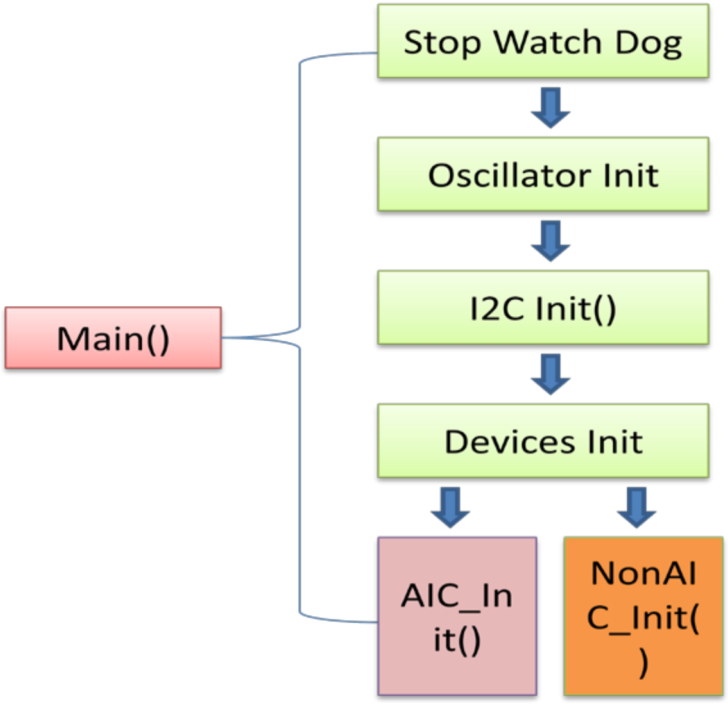

Figure 2. Main.c File Function Flow

Figure 2 shows the Main.c file function flow. There are two different kinds of I2C write available in this implementation.

The non-miniDSP based devices that just require a simple register configuration, (for example, TAS5717 or PCM9211, and so forth) can use the NonAIC_Init(). It simply takes an array for data and downloads it via I2C.

The miniDSP based devices (such as AIC3254, PCM514x and so forth) use multiple registers for peripherals and others for coefficients and instruction memory. Some additional logic is required to decode the I2C addresses into additional delay instructions to allow configuration to take place.

2.2 I2Cboot.h File

The I2Cboot.h file contains the register and register's value. When the MSP430 is booting, the I2C command will use the I2Cboot.h's content to configure the demo board.

```
 *
 *  Created on: 2013-5-1
 *      Author: a0219299
 */

#ifndef AIC_H_
#define AIC_H_


static const reg_value REG_Section_program[] = {

    {  0,0x00},
//          # reg[  0][  1] = 0x01  ; Initialize the device through software reset
    {  1,0x01},
    {254,0x0A},
    {  0,0x01},
//          # reg[  1][  1] = 0x08  ; Power up AVDD LDO; Disable weak AVDD to DVDD co
    {  1,0x08},
//          # reg[  1][  2] = 0x00  ; Enable Master Analog Power Control
    {  2,0x00},
//          # reg[  1][ 71] = 0x32  ; Set the input power-up time to 3.1ms
    { 71,0x32},
//          # reg[  1][123] = 0x01  ; Set REF charging time to 40ms (automatic)
    {123,0x01},
    {255,0x00},
    {255,0x01},
    {  0,0x00},
//          # reg[  0][ 60] = 0x00    ; DAC prog Mode: miniDSP_A and miniDSP_D NOT pc
    { 60,0x00},
```

Figure 3. I2Cboot.h Example

3 How to Connect the Hardware

The MSP430 booting demo's hardware connection is very easy. You only need to connect three wires, including SCL, SDA, and GND.

Figure 4 shows a typical setup, an MSP430 Launchpad board booting a TLV320AIC3254 codec. The only interface between both boards are the I2C pins (SDA, SCL) and GND.

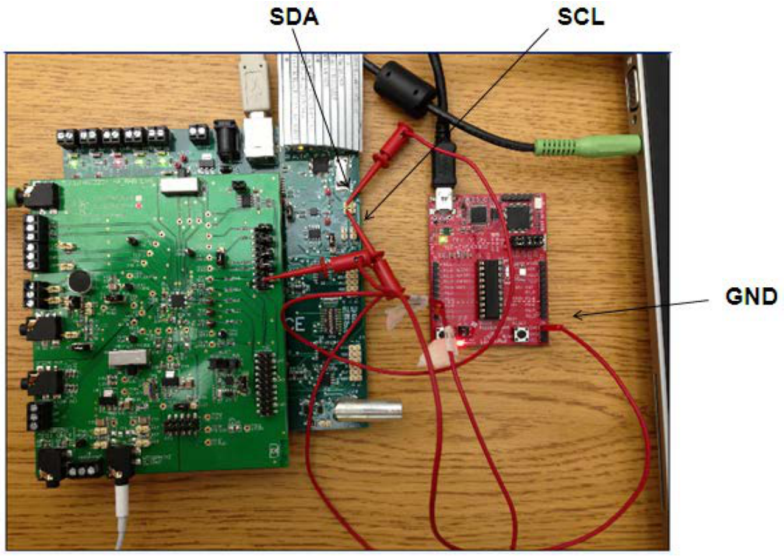

Figure 4. Hardware Connection

4 I2CBoot.h file Create and Modify

In this section, the I2Cboot.h file is illustrated with an example, using the AIC3254 device (miniDSP device) and the TAS5717 (non-miniDSP device).

4.1 MiniDSP Device

The majority of miniDSP devices use a development tool called PurePath™ Studio. PurePath Studio (PPS) has the ability to export header files that can be integrated into an embedded system.

Exporting the header file and importing it correctly into your code composer studio project is done with the following steps:

Step 1: PurePath Studio→Tools→Option→Build→Generate Device Driver Interface

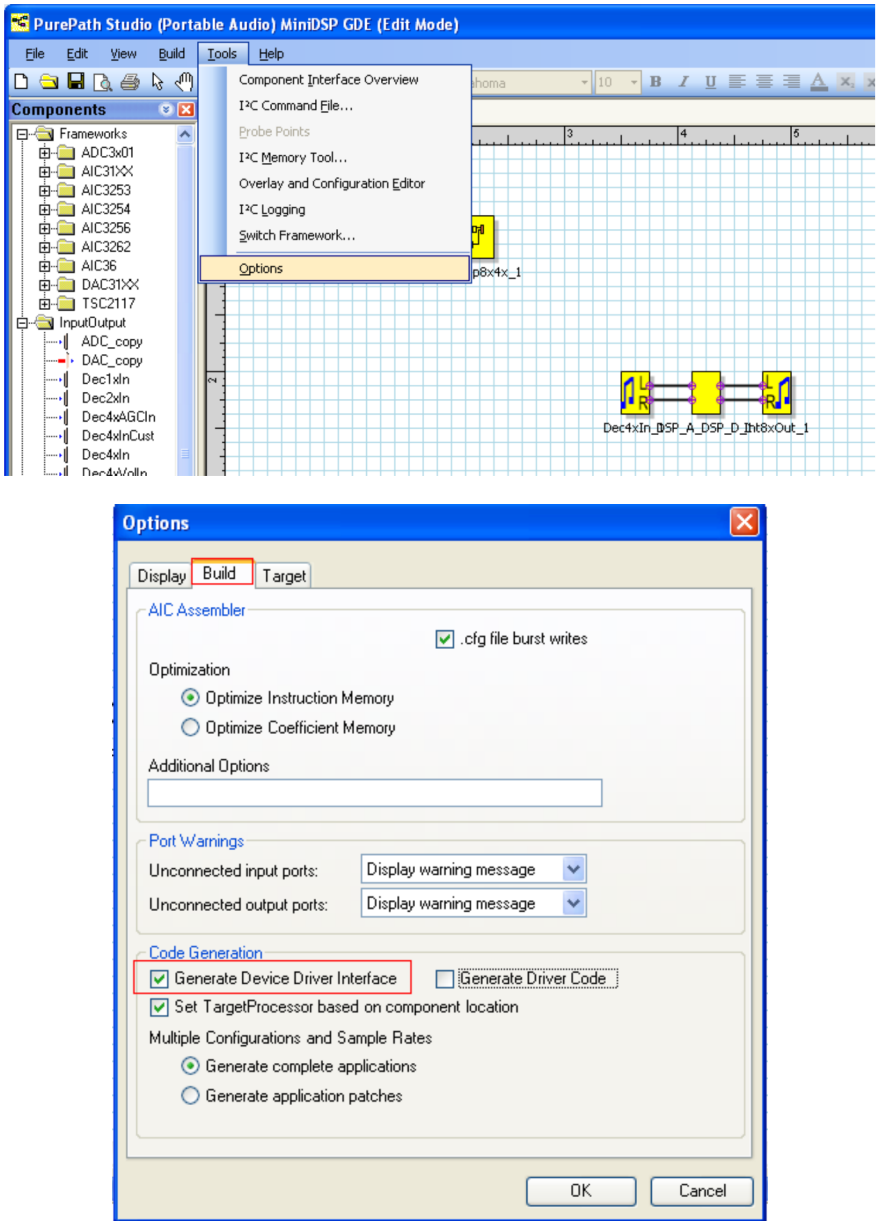

Figure 5. Configure the Option in the PPS

Step 2: Build→Generate Code

The base_main_Rate44_pps_driver.h file is found in the project folder.

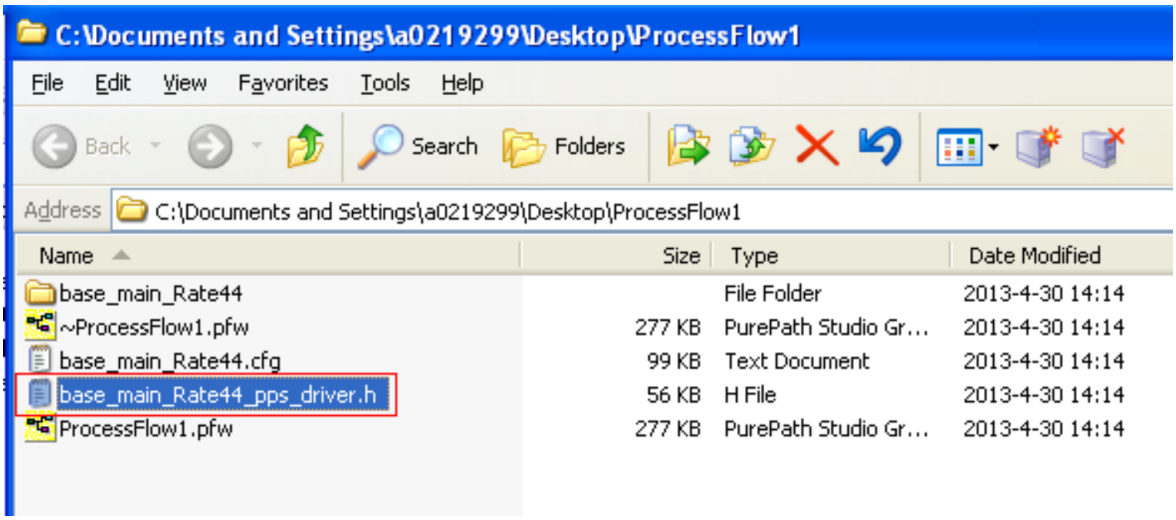

Figure 6. Generate the Configure File Through PPS

Step 3: Modify the base_main_Rate44_pps_driver.h file:

Remove the definition except the three definitions and the lengths.

1. Add “static const” before the arrays;
2. After modification, the base_main_Rate44_pps_driver.h is as illustrated in Figure 7:

```
12 static const reg_value REG_Section_program[] = {
13
14     {  0,0x00},
15 //          # reg[  0][  1] = 0x01  ; Initialize
16     {  1,0x01},
17     {254,0x0A},
18     {  0,0x01},
19 //          # reg[  1][  1] = 0x08  ; Power up AV
20     {  1,0x08},
21 //          # reg[  1][  2] = 0x00  ; Enable Mast
22     {  2,0x00},
                    ●
                    ●
                    ●
167 static const reg_value miniDSP_A_reg_values[] = {
168     {  0,0x08},
169     {  8,0x00},
170     {  9,0xB7},
171     { 10,0x98},
172     { 11,0x00},
                    ●
                    ●
                    ●
1590 #define miniDSP_A_reg_values_COEFF_START    0
1591 #define miniDSP_A_reg_values_COEFF_SIZE     590
1592 #define miniDSP_A_reg_values_INST_START     590
1593 #define miniDSP_A_reg_values_INST_SIZE      831
1594
1595 static const reg_value miniDSP_D_reg_values[] = {
1596     {  0,0x2C},
1597     {  8,0xFF},
1598     {  9,0xFF},
1599     { 10,0xFF},
1600     { 11,0x00},
1601     { 12,0x80},
                    ●
                    ●
                    ●
3136     {102,0xFF},
3137     {103,0x00},
3138 };
3139 #define miniDSP_D_reg_values_COEFF_START    0
3140 #define miniDSP_D_reg_values_COEFF_SIZE     356
3141 #define miniDSP_D_reg_values_INST_START     356
3142 #define miniDSP_D_reg_values_INST_SIZE      1186
```

Figure 7. miniDSP Device I2Cboot.h File Example

4.2 Non-MiniDSP Device

For non-miniDSP devices, the header file can include a much simpler array. The import flow is as follows:

Step 1: Generate a new header file. The header file name can be the device name. For example: Tas5717.h.

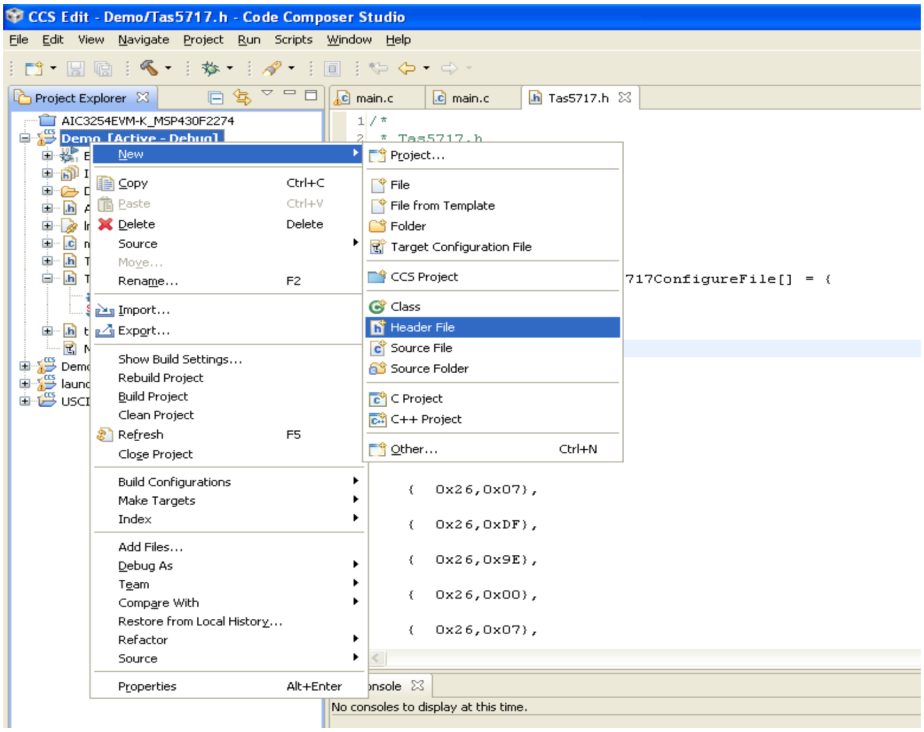

Figure 8. Generate the I2Cboot.h File in CCS

Step 2: Write an array with a format like the following:

Tas5717ConfigureFile is the name which is used in the main function.

If the device you are writing to uses multibyte registers, the same register should be written to many times. Each write moves the position of that particular write within that register.

For example, the register 0x07 is two bytes in the TAS5717 device. You should write twice. Figure 9 shows the array to write a multi-byte register.

```
static const reg_value Tas5717ConfigureFile[] = {

    {  0x1B,0x00},

    {  0x05,0x00},

    {  0x07,0x00},

    {  0x07,0x30},

    {  0x26,0x00},

    {  0x26,0x07},

    {  0x26,0xDF},

    {  0x26,0x9E},
```

Figure 9. Non-miniDSP Device I2Cboot.h File Example

Step 3: Include "Tas5717.h" at the main function.

```
/*****************************************************************************************/
/*    Include                                                                            */
/*****************************************************************************************/
#include    "msp430g2553.h"
#include    "AIC.h"
#include    "type.h"
#include    "Tas5717.h"
```

Figure 10. Non-miniDSP Device Head File

Step 4: Define the device address.

If using GPIO_I2C, use the data sheet address (0x30). If using USCI_I2C, modify the address (right-shift). 0x30→0x18

```
/*****************************************************************************************/
#define   DIR         P2DIR
#define   OUTP        P2OUT
#define   IN          P2IN      //For I2C
#define   SCL         BIT1      //Bit 1 (SCL)
#define   SDA         BIT0      //Bit 0 (SDA)
#define   AIC3254     0x30      //Address
#define   Tas5717     0x54      //Address
```

Figure 11. Non-miniDSP Device I2C Address

Step 5: Modify the function:

NonAIC_Init**(Tas5717ConfigureFile,sizeof(Tas5717ConfigureFile)/2,Tas5717)**;

Where:

Tas5717ConfigureFile *is the array's name*.
Sizeof(Tas5717ConfigureFile)/2 *is the array's length.*
Tas5717 is the device's name.

Figure 12 shows an example of the main() function used to boot both an AIC3254 miniDSP and a TAS5717 digital amplifier from the same microcontroller.

```
/*****************************************************************************
void main(void) {

      WDTCTL = WDTPW + WDTHOLD;          // Stop WDT
      I2C_METHOD = GPIO;
    // If the I2C_METHOD is 1, the code will use the GPIO I2C mode. And if the I2C I
          if(I2C_METHOD == 1)
      {
          BCSCTL1 = CALBC1_1MHZ;
          DCOCTL = CALDCO_1MHZ;
          GPIO_I2C_Init();
          Delay_ms(10);
          AIC_Init(AIC3254);
          NonAIC_Init(Tas5717ConfigureFile,sizeof(Tas5717ConfigureFile)/2,Tas5717);
          Delay_ms(20);
```

Figure 12. Non-miniDSP Device NonAIC_Init

5 Summary

This hardware and software platform provides a quick and easy process to demonstrate and debug your project, in a system or on a standalone demonstration board. Developers can go from a system that sounds good on the EVM and PC development tools, to a standalone booting platform in a matter of minutes.

Both common I2C devices, and miniDSP-based devices are supported on the demo hardware, as well as the ability to boot using USCI or using GPIO's on the MSP430. Buttons on the launchpad can be used to do things like "audio effect on, audio effect off", making demonstration and debugging easy and effective.

For technical questions and discussions, visit the audio amplifiers forum at e2e.ti.com

Appendix A miniDSP Device Head File

A.1 miniDSP Head File Structure

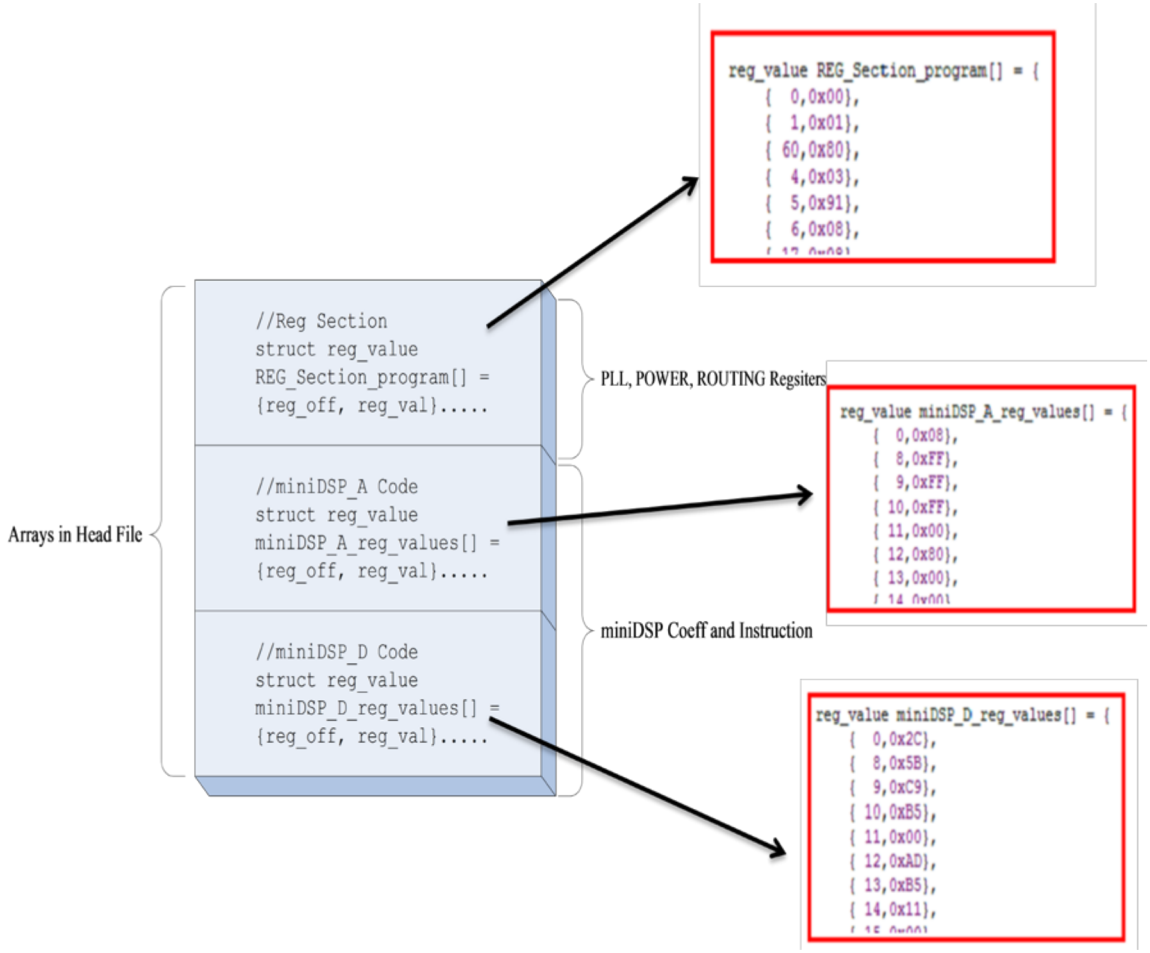

Figure 13. miniDSP Head File Structure

A.2 miniDSP Device Flow Chart for Reg_Section

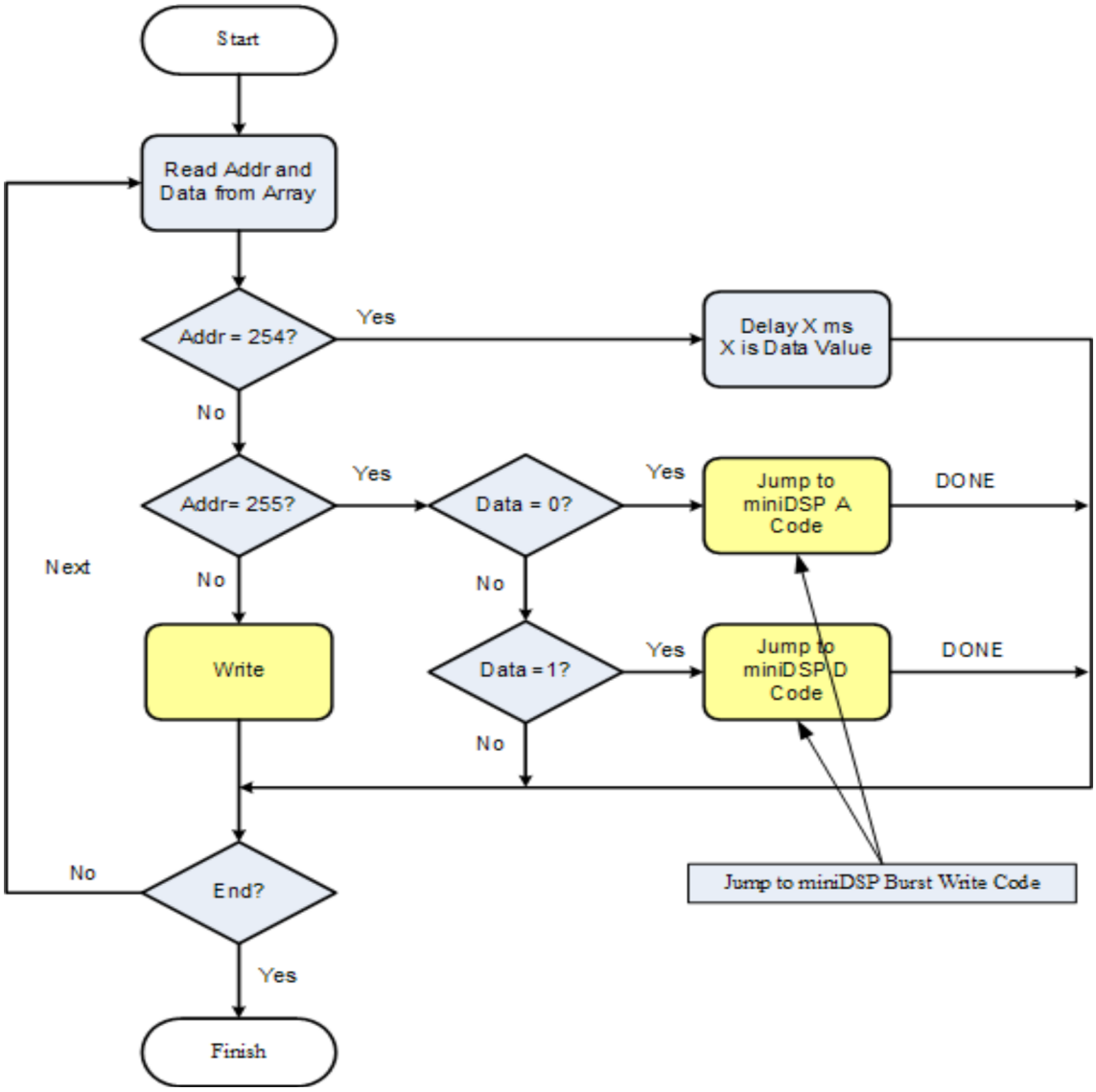

Figure 14. Decoder for Reg Section

Decoder for Reg Section:
If reg = 254 : Command for delay. The delay time is the value of data (ms).
If reg = 255 , data=0 : command for jump to miniDSP-A programming.
If reg = 255 , data=1 : command for jump to miniDSP-D programming.

Revision History

Changes from Original (October, 2013) to A Revision	Page
• Added link to MSP430 Audio Boot Code (SLAC640)	1

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

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