AND9530/D

Sound Recording Software of LC823450 Series for Audio Applications

ON Semiconductor®

www.onsemi.com

APPLICATION NOTE

Introduction

This application note describes a software outline of sample application for standalone sound recording implemented in LC823450XGEVK, which is an evaluation board kit, to help customers to understand and reuse the sample application.

Intended audience is customers who are building audio application using LC823450 Series (called LC823450 hereafter).

BACKGROUND

LC823450XGEVK is an audio processing system evaluation board kit used to demonstrate LC823450. This part can playback and record, and offers High-Resolution 32-bit & 192 kHz audio processing capability. It is possible to cover most of the functions necessary for a portable audio with only this LSI.

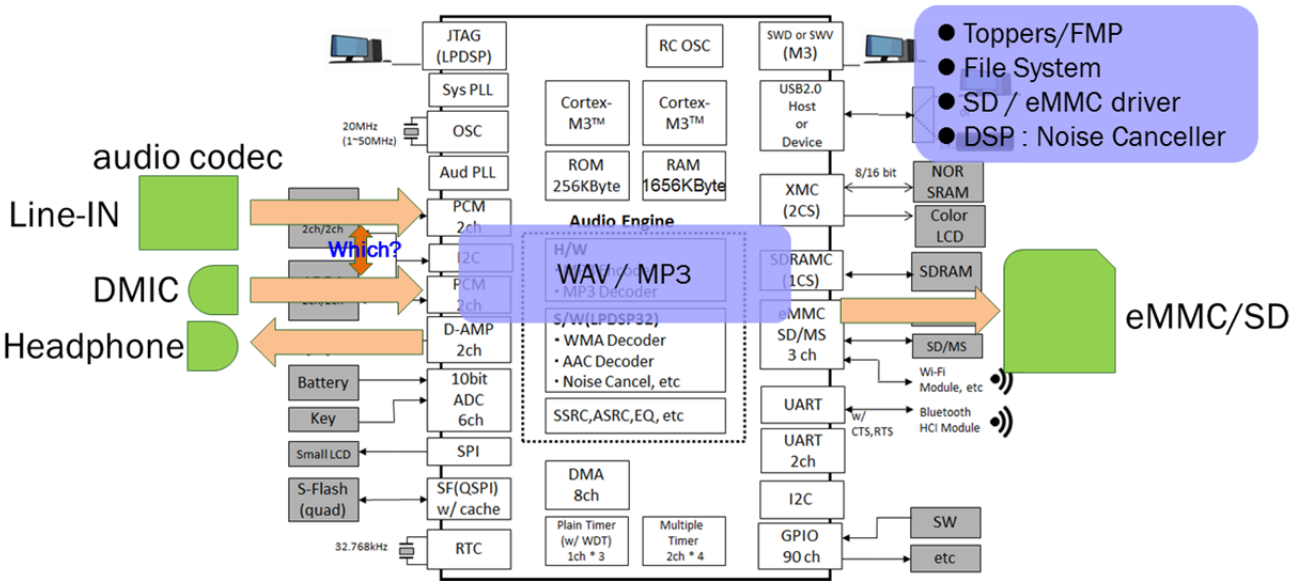
This application note focuses on the standalone sound recording application implemented in LC823450XGEVK and describes a software outline of the sample code.

STANDALONE SOUND RECORDING

System structure

Figure 1 shows the system structure of the standalone sound recording mode. The Audio Engine block in the center is composed of audio hardware blocks and a software block which is LPDSP32. The recorded sound data is input through Digital Microphone Data Input terminal from Digital Microphone with PDM format or through PCM Data Input terminal from external audio codec with PCM format.

Figure 1. Standalone Sound Recording mode



The recorded sound data stored into the eMMC device or microSD card by MP3 format or WAV format. The MP3 format data is encoded by Hard-Wired MP3 Encoder inside LC823450. In addition, LC823450 can monitor the recording sound through the headphone concurrently.

Software mainly used in this mode is Toppers / FMP, eMMC / SD driver, File system and Noise Canceller. Toppers / FMP is the OS used in the system. eMMC/SD driver is a device driver which controls SD interface. File system is included in libraries for Cortex®-M3 and Noise Canceller is included in libraries for LPDSP32.

Audio structure

Figure 2 shows details in the Audio Engine block. It has audio buffers in the center and some audio functions in the right upper side as hardware blocks and LPDSP32 in the right lower side as a software block.

Figure 2 also shows sound routes at the standalone recording mode in the Audio Engine block. There are 2 sound routes in the standalone recording mode, the Sound Recording route and the Monitor Play route.

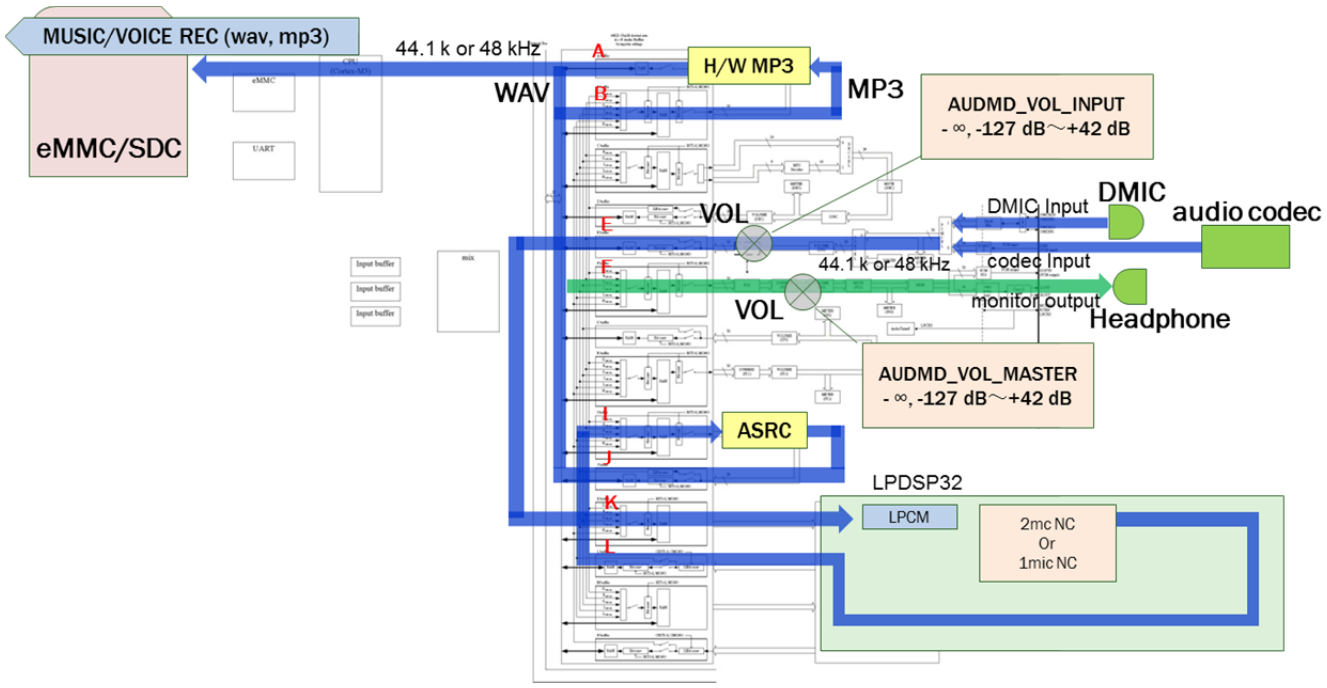
The Sound recording route is shown in light blue in Figure 2, and has a role to store the sound data into the eMMC device or microSD card. The Monitor Play route is shown in light green, and has a role to monitor the recording sound simultaneously from headphone.

In the Sound Recording route, the sound data are input from Digital Microphone on the board with the sampling rate 44.1 or 48 kHz which is selected by the setting menu in sample application. They are passed through the PDM interface and input to the LPDSP32 through the E buffer and K buffer in the audio buffers, and they are processed by LPDSP32 with the Noise Canceller if necessary. If you select MP3 as recording format, the processed data by LPDSP32 is input to Hard-Wired MP3 Encoder through the ASRC and encoded. On the other hand, if you select WAV as recording format, the processed data by LPDSP32 is input to eMMC device or microSD card directly.

In the Monitor Play route, sound data which processed by LPDSP32 are separated from the Sound Recording route after the ASRC. It is input to the F buffer and passed through the audio output functions and output to the headphone via D-AMP.

There is an input volume function before getting to the LPDSP32 on the route and there is a monitor volume function is on the Monitor Play route after the F buffer.

Figure 2. Sound route at the Standalone Sound Recording mode



Software flow

Figure 3 shows a rough software flow of the standalone sound recording mode in light orange. Some keypoints which customers should know in order to reuse the sample application are described in this chapter.

In the step(#1), the OS is set up, and initial routines which should be used only once through the system are executed. In the step(#2), some blocks which are related to the whole system are initialized. In the step(#3), the file system is initialized, and this step has some branching points to the each operating mode decided in the menu display or to the default mode assigned after power on which is standalone music playback mode. In the step(#4), some routines used at music stop state in the standalone music playback mode are executed. In the step(#5), a music file selected in the standalone music playback mode is played back. In the step(#6), the menu is displayed on the LCD, and an operating mode which should be executed at next step can be selected. Whenever getting away from the menu, the step(#3) is executed again, because the file system needs to be initialized so as to access to a music storage again when the music storage is changed from one of either the eMMC device or microSD card to the other in the menu. In the step(#7), some routines used at recording stop state in the standalone sound recording mode are executed. In the step(#8), the sound input from microphone is recorded.

Figure 3. Standalone sound recording flow

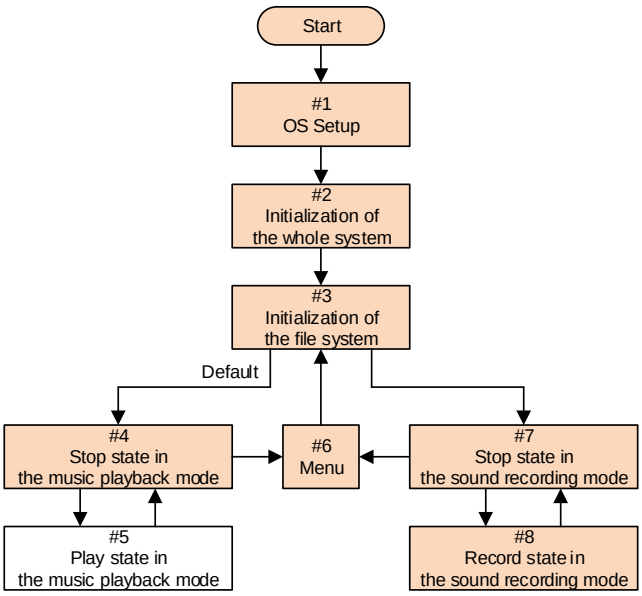


Figure 4 shows a part of a source code in the step(#1), which is the “global_inirtn” routine in the “SysAp.c” file. This routine initializes internal power domain, GPIO, DMA and so on, and it is executed only once through the system.

Especially, the “port_init” routine in red initializes GPIO setting such as port direction, drive capability switching, pull resister setting and so on. If the customers reuse the sample application, they have to edit the “port_init” routine to meet their system.

Figure 4. OS Setup

Step(#1) : SysAp.c

```
void global_inirtn( intptr_t exinf )
{
        :
        :
  AdcInit();
  AdcExit();

  port_init();

  StgCoreValtageChange(TRUE_T);
        :
        :
  return;
}
```

Figure 5 shows a part of a source code in the step(#2), which is the “SysApInit” routine in the “SysAp.c” file. The “SysUartInit” routine in red initializes a UART function and this initialization enables the customers to use “Printf” routines through the UART function on PC at debug. If the customers need to use I2C functions in their system, they should activate the “I2c0Init” and/or “I2c1Init” routine in red by removing the comment-out mark. The “LcdInit” routine in red initializes to clear up the LCD display as a blank on LC823450XGEVK. If the customers don’t need to use the LCD display in their system, they should inactivate the routine by adding the comment-out mark.

Figure 5. Initialization of the whole system

Step(#2) : SysAp.c

```
static SINT_T SysApInit(void)
{
        :
        :
  SysUartInit();

  SysMdInitTimer();

//  I2c0Init();
//  I2c1Init();

  LcdInit();

  if(MediaDrInit(PwrDrGetAHBClock(), STG_VDDIF_33V)){
    console_Printf(USE_CONSOLE_CH,"Err Init¥n");
      goto ERR_END;
  }
  SysMdChangeClock(100000000, TRUE_T, FALSE_T);

  if(MediaDrIdentfyFixed(PwrDrGetAHBClock())){
    console_Printf(USE_CONSOLE_CH,"Err Idetfy¥n");
      goto ERR_END;
  }
        :
        :
  return(ret);
}
```

In addition, The “MediaDrIdentfyFixed” routine in red recognizes the music storage by establishing a connection to it by sending commands. This is a preparation so as to access to music data and LPDSP codes in the storage.

Figure 6 shows a part of a source code in the step(#3), which is the “app_Init_Init” routine in the “app_init.c” file. This routine initializes the file system and branches according to the operation modes in the menu. The default shows the standalone music playback mode.

The “FsInit” routine in red initializes internal SRAM area assigned in order to mount the file system. The next grouped routines in red mount the file system. The file system corresponds to FAT12, FAT16, FAT32 and exFAT.

The “app_MakePlayList” routine in red makes a playlist of music files stored in the “music” folder in the storage. The “app_SetMPMode” routine in red initializes the audio function in LC823450, for example, PLL setting, DAC setting, volume setting and so on.

Figure 6. Initialization of the file system

Step(#3) : app_init.c

```
SINT_T app_Init_Init(t_SYS_MASTER_TBL *m_tbl)
{
        :
        :
   FsInit();

   if(FsMount(0)){
      FsFormat(0);
      FsMount(0);
   }
   dbg_warning_value( FsMount( 1 ) );
        :
        :
   switch(SetParaGetMode()){
      case MODE_REC:
        :
        :
        break;
      default:
        dbg_warning_value( app_MakePlayList(SetParaGetDrive(),
                                            SetParaGetRecMode()) );
        app_SetMPMode();
        SysMdSetStatus(m_tbl, APP_STS_STOP, TRUE_T);
        break;
   }

   return( 0 );
}
```

Figure 7 shows a part of a source code in the step(#4), which is the “app_Stop_Init” routine in the “app_stop.c” file. This routine is called at first whenever the standalone music playback mode works, and it initializes some functions for music stop state, because the standalone music playback mode always starts from the music stop state. The “app_lcd_com” routine in red sets the LCD device to display characters which mean the music stop state. If the customers don’t need to use the LCD display in their system, they should inactivate the routine by adding the comment-out mark.

After the initialization of the music stop state, the continuing task waits for any available keys in the music stop state to be pushed. If the key which means “Playback” is pushed, the state of the standalone music playback mode will change from the music stop state to the music play state. If the key which means “Menu” is pushed, the sample application will get away from the standalone music playback mode to move the menu.

Figure 7. Stop state in the music playback mode

Step(#4) : app_stop.c

```
static SINT_T app_Stop_Init(t_SYS_MASTER_TBL *m_tbl)
{
        :
        :
   app_lcd_com(DISP_PART_TITLE, m_tbl);
        :
        :
   return (0);
}
```

Figure 8 shows a part of a source code in the step(#7), which is the “app_Stop_Init” routine in the “app_stop.c” file. This routine is called at first whenever the standalone sound recording mode works, and it initializes some functions for recording stop state, because the standalone sound recording mode always starts from the recording stop state. The “app_lcd_com” routine in red sets the LCD device to display characters which mean the recording stop state. If the customers don’t need to use the LCD display in their system, they should inactivate the routine by adding the comment-out mark.

After the initialization of the recording stop state, the continuing task waits for any available keys in the recording stop state to be pushed. If the key which means “Recording” is pushed, the state of the standalone sound recording mode will change from the recording stop state to the sound recording state. If the key which means “Menu” is pushed, the sample application will get away from the standalone sound recording mode to move the menu.

Figure 8. Stop state in the sound recording mode

Step(#7) : app_stop.c

```
static SINT_T app_Stop_Init(t_SYS_MASTER_TBL *m_tbl)
{
        :
        :
   app_lcd_com(DISP_PART_TITLE, m_tbl);
        :
        :
   return (0);
}
```

Figure 9 shows a part of a source code in the step(#8), which is the “app_Stop_Sub” routine in the “app_stop.c” file. This routine runs whenever the state becomes the sound recording state. The “app_RecStop” routine instructs other tasks controlled in the multitask OS to record sound files as shown in the Sound Recording route in Figure 2. After the instruction, the continuing task waits for any available keys in the sound recording state to be pushed, and the other tasks for sound recording simultaneously work to make a sound file with header in storage device and to input sound data to LPDSP from Digital Microphone for several audio effects and to push it to the Hard-Wired MP3 encoder through audio buffers (in the case that MP3 is selected as the recording format) and to storage the encoded sound file to eMMC device or microSD card.

Figure 9. Recording state in the sound recording mode

Step(#8) : app_stop.c

```
static BOOL_T app_Stop_Sub(t_SYS_MASTER_TBL *m_tbl,
t_MSG_DATA_BASETYPE *pMsg)
{
                        :
                        :
                app_RecStop(TRUE_T);
                        :
                        :
    return (TRUE_T);
}
```

ARM, the ARM logo, and Cortex are registered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere.

ON Semiconductor and the ON Semiconductor logo are trademarks of Semiconductor Components Industries, LLC dba ON Semiconductor or its subsidiaries in the United States and/or other countries. ON Semiconductor owns the rights to a number of patents, trademarks, copyrights, trade secrets, and other intellectual property. A listing of ON Semiconductor’s product/patent coverage may be accessed at www.onsemi.com/site/pdf/Patent-Marking.pdf. ON Semiconductor reserves the right to make changes without further notice to any products herein. ON Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does ON Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation special, consequential or incidental damages. Buyer is responsible for its products and applications using ON Semiconductor products, including compliance with all laws, regulations and safety requirements or standards, regardless of any support or applications information provided by ON Semiconductor. “Typical” parameters which may be provided in ON Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals” must be validated for each customer application by customer’s technical experts. ON Semiconductor does not convey any license under its patent rights nor the rights of others. ON Semiconductor products are not designed, intended, or authorized for use as a critical component in life support systems or any FDA Class 3 medical devices or medical devices with a same or similar classification in a foreign jurisdiction or any devices intended for implantation in the human body. Should Buyer purchase or use ON Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold ON Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that ON Semiconductor was negligent regarding the design or manufacture of the part. ON Semiconductor is an Equal Opportunity/Affirmative Action Employer. This literature is subject to all applicable copyright laws and is not for resale in any manner.