Application Report
SPRA966 − December 2003

Hands-Free Kit: Integration of the AIC24 Driver in Reference Framework 3

Texas Instruments HFK DP Software Applications

ABSTRACT

This application report describes the AIC24 codec driver used in the Hands-Free Kit (HFK) system and presents its integration in the Reference Framework 3 (RF3).

The Reference Frameworks have been developed by Texas Instruments to provide a foundation for easy integration of eXpressDSP-compliant algorithms in a DSP/BIOS environment. This report was designed as a tutorial to help the users of the HFK understand how to integrate the software modules in RF3.

The application report presents first the implementation of the AIC24 driver. Second the customization of the standard RF package for the HFK system is presented. Third, the integration of the AIC24 driver with RF3 is described.

Contents

List of Figures

1 Introduction

The Hands-Free Kit Development Platform (HFK DP) system described in this application report was developed as an open system based on the DSP/BIOS RF3. This system was implemented on the HFK motherboard that features a TMS320C5407 DSP and a TLV320AIC24 codec from Texas Instruments.

This application reports describes the implementation of the AIC24 driver and its integration in RF3. The integration of the AIC24 codec is presented in a tutorial format to help you understand the process. The final result of this tutorial is provided in the HFK SDK in the folder HFK\Src\Examples\referenceframeworks\apps\rf3\hfk5407_aic24. The block diagram of this example is provided in Figure 1

The example hfk5407_aic24 was developed using the files provided in the HFK SDK and Code Composer Studio v 2.20.

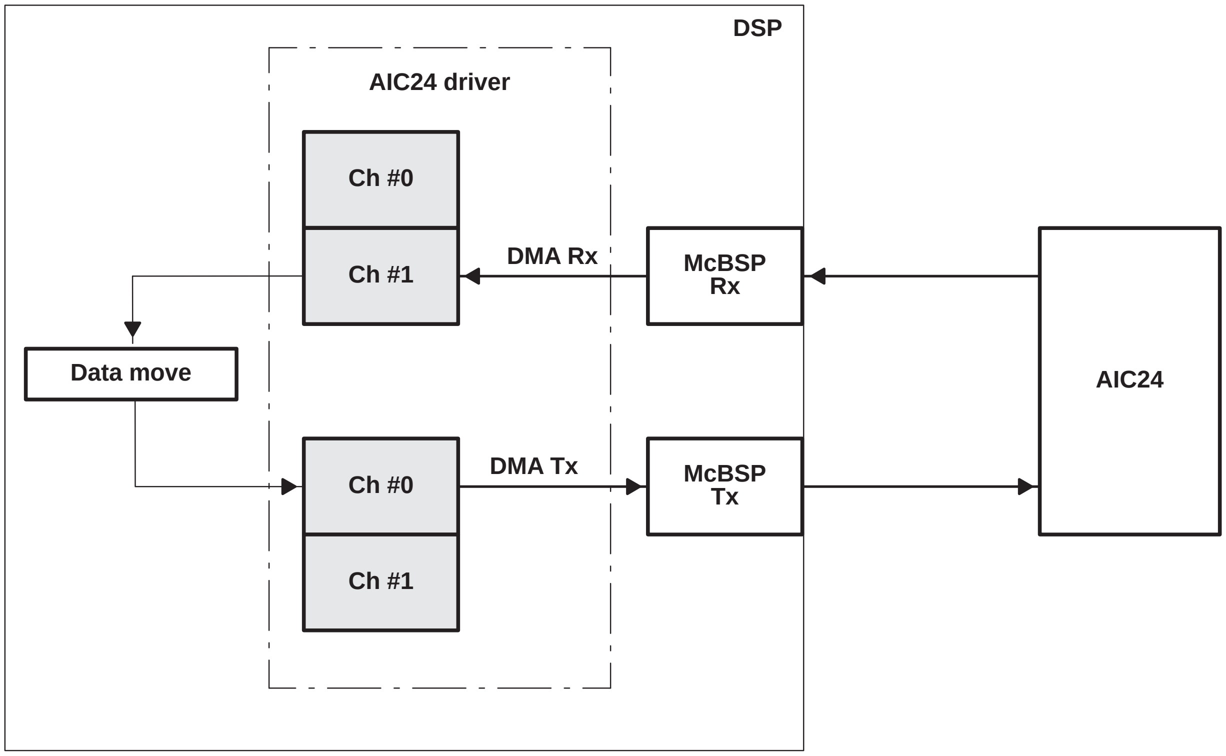


Figure 1. Block Diagram of the HFK Loopback Audio Example hfk5407_aic24

NOTE: Before executing any HFK-related code in Code Composer Studio, please update the chip support library (CSL) in Code Composer Studio with the one provided in the HFK SDK. The CSL is located in HFK\Src\misc\CSL.

2 Getting Started

This section will provide the necessary steps to run the audio loopback example, hfk5407_aic24 provided with the HFK SDK.

Updating the CSL libraries

The updated CSL libraries and header files are provided in the SDK in the file HFK\Src\misc\CSL\c54xx.zip

1. Unzip the file c54xx.zip to a temporary folder temp
2. Create the temp\include and the temp\lib directories in the temporary folder.
3. Inspect the folder where the current CSL libraries are located. The CSL libraries are installed as part of the Code Composer Studio installation process. By default, they are located in c:\ti\c5400\bios\include and c:\ti\c5400\bios\lib.
4. Copy the content of temp\lib and temp\include to the respective Code Composer Studio folders c:\ti\c5400\bios\include and c:\ti\c5400\bios\lib.
5. Overwrite the existing file. Do NOT replace the Code Composer Studio folders with the ones in the temporary folder because the Code Composer Studio folders contain additional useful DSP/BIOS libraries.

Preparing the hardware

6. Connect the JTAG cable to the HFK EVM
7. Connect a CD player to the microphone input of the HFK EVM using the audio cable provided in the HFK.
8. Connect an amplified speaker to the speaker output of the HFK EVM.
9. Power-up the HFK EVM

Running the audio loopback example hfk5407_aic24

10. Start Code Composer Studio and open the project called app.pjt located in HFK\Src\Examples\referenceframeworks\apps\rf3\hfk5407_aic24 by using Project → Open Menu command.
11. Load the GEL file provided in HFK\Src\GEL
12. Load the executable app.out from .\Debug subdirectory with File → Load Program
13. Start the audio source.
14. Run the code: Select Debug → Run from the Debug menu or press F5.

    The audio signal will be heard on the amplified speaker.

Inspecting the project files

15. Use the project view window on the left-hand side of the Code Composer Studio screen, to examine the project.
16. Open the Source folder to examine the project files.
17. Open the file hfk5407_aic24.c. This file defines two functions: the function HFK5407_AIC24_init() (line 154) is used to initialize the aic24 driver and the function HFK5407_AIC24_DMA_isr() (line 295) is the aic24 driver interrupt service routine (ISR).

    Note that the swiHfkAudio SWI object is posted in the ISR HFK5407_AIC24_DMA_isr(line 328).
18. Set a breakpoint in the HFK5407_AIC24_DMA_isr() function and select Debug → Animate from the Debug menu or press F12. The ISR will be called every time there are buffers available for processing.

19. Remove the breakpoint and close the file hfk5407_aic24.c.
20. Open the file hfk5407_drvGbl.c. This file defines the function HFK5407_audioBuffer_init() (line 58) used to initialize the aic24 driver buffers. This function is called during the initialization process.
21. In the file hfk5407_drvGbl.c, the aic24 driver buffers liMicData[] and loSpkData[] are defined . Determine the starting addresses of the buffers by setting the cursor on the first letter of the name. Write down the addresses of the buffers. They will be used later to view the signal with the graph tool.
22. Close the file hfk5407_ drvGbl.c
23. In the Code Composer Studio window select in the View menu View → Graph → Time/Frequency. A Graph Property Dialog window will open. Modify the following fields:

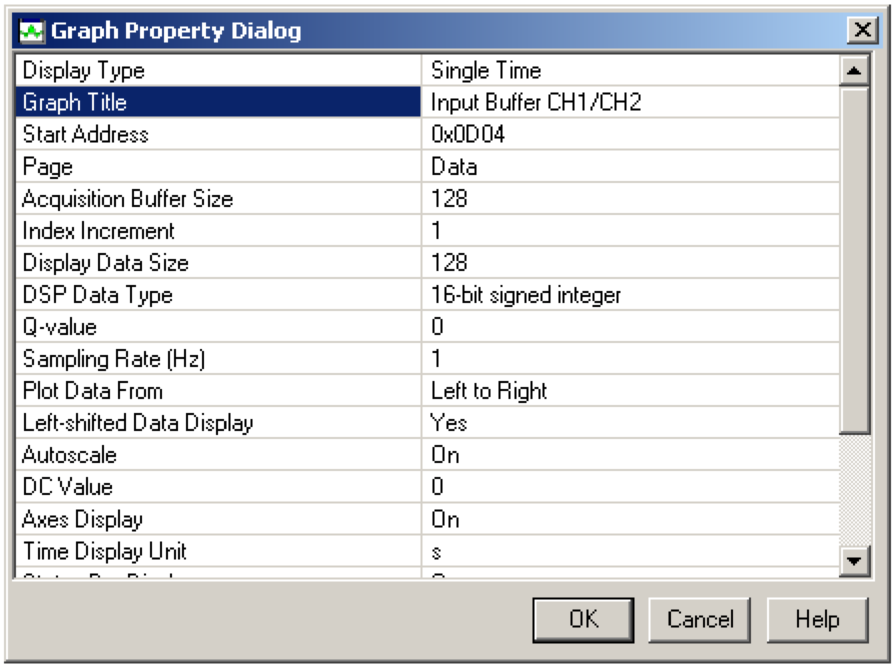
    Start Address: address of the liMicData[] buffer

    Acquisition Buffer Size: 128

    Display Data Size: 128

    DSP Data Type: 16-bit signed integer

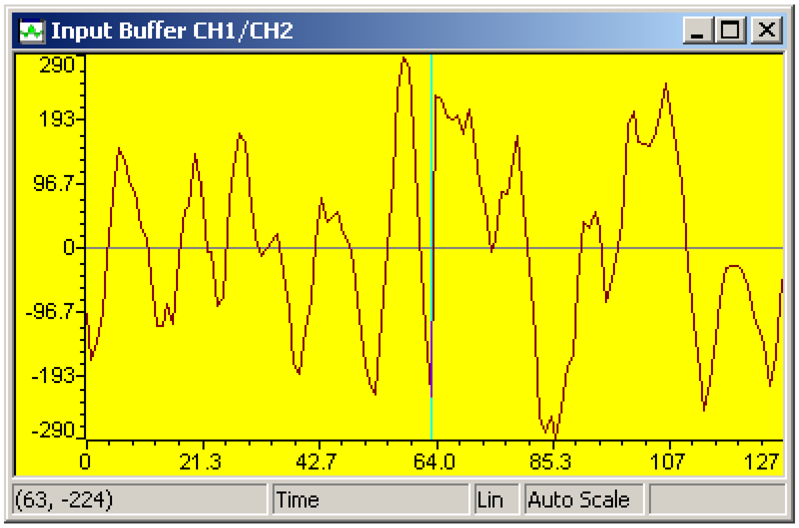
    You should see something similar to:



Display Type	Single Time
Graph Title	Input Buffer CH1/CH2
Start Address	0x0D04
Page	Data
Acquisition Buffer Size	128
Index Increment	1
Display Data Size	128
DSP Data Type	16-bit signed integer
Q-value	0
Sampling Rate (Hz)	1
Plot Data From	Left to Right
Left-shifted Data Display	Yes
Autoscale	On
DC Value	0
Axes Display	On
Time Display Unit	s

24. Select OK

25. The Graph Window will display the content of the buffer. In order to update the display right-click and select Refresh.



The first 64 locations represent the AIC24 Channel 1 receive buffer and the next 64 locations (64 through 127) represent the AIC24 Channel 2 receive buffer.

26. Open the file thrHfkAudio_aic24.c. This file defines two functions: the function thrHfkAudioInit() (line 19) is used to initialize the Hfk Audio thread and the function thrHfkAudioRun() (line 32) is the thread processing function associated with the swiHfkAudio SWI object.
27. The function thrHfkAudioRun() is called by the DSP/BIOS kernel after the swiHfkAudio SWI object is posted in the ISR. In the example hfk5407_aic24 this function performs a simple data move.

3 The AIC24 Driver

The AIC24 driver provided with the HFK SDK is available in the folder HFK\Src\drivers\drv8kHz_nobsync. The driver includes:

- header files: aic24.h, hfk5407_aic24.h
- source code: aic24.c, hfk5407_aic24.c

Global variables and buffers used by the driver are provided in the same folder.

- header files: hfk5407_drvGbl.h
- source code: hfk5407_drvGbl.c

3.1 Implementation of the AIC24 Driver

The AIC24 driver is implemented in the files aic24.c, hfk5407_aic24.c and hfk5407_drvgbl.c associated to their respective header files. The operation of the driver depends also on functions implemented in the file hfk5407.c. Before describing these files in detail a summary for each file is presented.

- hfk5407.c

  The file hfk5407.c located in HFK\Src\evmInit implements the function AIC24_resetInit() used to take the AIC24 out of reset.
- hfk5407_drvGbl.c

  The file hfk5407_drvGbl.c located in HFK\Src\drivers\drv8kHz_nobsync defines and initializes the global variables and structures used by the AIC24 driver.
- hfk5407_aic24.c

  The file hfk5407_aic24.c located in HFK\Src\drivers\drv8kHz_nobsync implements the AIC24 driver initialization function that defines, initializes and starts the McBSP port and DMA channels used by the AIC24 driver. It also implements the Interrupt Service Routine (ISR) used by the AIC24 driver.
- aic24.c

  This file located in HFK\Src\drivers\drv8kHz_nobsync implements the AIC24_setParams() function that programs the AIC24 with the parameters defined in the file aic24.h

3.1.1 Hfk5407.c

The file hfk5407.c contains the HFK5407 EVM initialization functions. The function AIC24_resetInit()is used to take the AIC24 out of reset. This is accomplished by writing a 1 in the CDCRST field of the MISCREG register defined in the PLD. For more information about this register the reader is referred to the *Texas Instruments Hands-Free Kit Development Platform User's Guide* (SPRU703)

3.1.2 Hfk5407_drvGbl.c

The file hfk5407_drvGbl.c defines global variables, structures and arrays used by the AIC24 driver. It also implements functions that initialize these global variables.

The data buffers used by the AIC24 driver are defined in this file. The line input Microphone data buffer, liMicData[], and the line output Speaker data buffer , loSpkData[], have the same size FRAME_SIZE_8K*4. This corresponds to two ping/pong buffers for each channel. Since the AIC24 has two channels this represents a total of four buffers for the input and four buffers for the output. The size of one buffer is defined as, FRAME_SIZE_8K = 64.

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                          AIC24
 *                        Data Buffers
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
#pragma DATA_SECTION(liMicData,".HFK_AUDIO");
#pragma DATA_SECTION(loSpkData,".HFK_AUDIO");
#pragma DATA_SECTION(BufferActive,".HFK_AUDIO");
#pragma DATA_SECTION(dma_sync,".HFK_AUDIO");

// Codec Buffers – 8kHz
int liMicData[FRAME_SIZE_8K*4];
int loSpkData[FRAME_SIZE_8K*4];
```

It is to notice that the AIC24 buffers have been allocated specifically in the .HFK_AUDIO memory section using the #pragma DATA_SECTION () directive. This is accomplished in order to ensure that the AIC24 buffers will be allocated in internal memory. Had this not been specified, the buffers would have been allocated as part of the general .bss section which may be allocated by the user in external memory.
The AIC24 driver uses a global variable, bufActive, to specify the buffers available for processing. It also uses a global structure dmaBufPtrs to store the pointers to the driver buffers.

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                          AIC24
 *                      Global Variables
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
volatile unsigned short bufferActive;
volatile DMA_BUF_PTRS dmaBufPtrs[1];
```

The global variables and structures are initialized by the function void HFK5407_audioBufferInit(void) implemented in the file hfk5407_drvGbl.c This function initializes bufActive and dmaBufPtrs. The output buffers are initialized to zero in order to avoid re-programming the AIC24. Indeed, the AIC24 is normally operated in continuous data transfer mode. In this mode, the control mode can be reactivated when the AIC24 is operated in the 15+1 data format mode by setting the LSB of the AIC24 input data to 1. For more information about the two modes of operation of the AIC24 the reader is referred to the *TLV320AIC24 Data Manual* (SLAS366).

```
void HFK5407_audioBufferInit(void)
{
       int off_0,i ;

       /* Initialize Global Variables */
       bufActive  = 1;

       /* MUST initialize the buffers to 0 otherwise will re-program the AIC24*/
       for (i=0; i< FRAME_SIZE_8K*4; i++){
              loSpkData[i] = 0;
       }

       /* Set Up DMA Buffers */
       off_0  = 0;
       dmaBufPtrs[AIC24_IDX].SndBufs[0] = (unsigned short)&loSpkData[off_0];
       dmaBufPtrs[AIC24_IDX].RcvBufs[0] = (unsigned short)&liMicData[off_0];
       off_0  = 2*FRAME_SIZE_8K;
//     off_0  = 2*FRAME_SIZE_16K;
       dmaBufPtrs[AIC24_IDX].SndBufs[1] = (unsigned short)&loSpkData[off_0];
       dmaBufPtrs[AIC24_IDX].RcvBufs[1] = (unsigned short)&liMicData[off_0];
}
```

3.1.3 Hfk5407_aic24.c

The file hfk5407_aic24.c implements the initialization function and the ISR of the AIC24 driver.

The HFK EVM connects the AIC24 to the serial port McBSP 2. This is a property of the board and can not be changed. The DMA channels connected to the McBSP 2 are defined in software and can be changed. However before changing the DMA channels the user must be familiar with the HFK system in order to fully understand where the changes need to be made.

The AIC24 software uses the McBSP port and DMA channels defined in the header file hfk5407_aic24.h:

```
#define HFK5407_AIC24_RXDMAID_DEFAULT       0
#define HFK5407_AIC24_TXDMAID_DEFAULT       1
#define HFK5407_AIC24_MCBSPID_DEFAULT       2
```

The chip support library (CSL) is used to program the McBSP port and DMA channels.

The CSL McBSP configuration structure, MCBSP_Config mcbspCfgAic24, is defined in the file hfk5407_aic24.c. Some remarks about the serial port configuration:

- Free running mode.

  The Free running mode is enabled in Serial Port Control Register 2 (SPCR2). This feature is used for emulation purposes.

- External frame-synchronization and clock signals

  The frame-synchronization and clock signals are provided externally by the AIC24. The Pin Control Register (PCR) is set accordingly.

- Frame and word lengths.

  The AIC24 is operated in continuous mode. In this mode the serial port operates with two 16-bit words per frame. These settings are defined in the Receive Control Register 1 (RCR1) and the Transmit Control Register 1 (XCR1).

- Data Delay (R/X) DATDLY

  The AIC24 requires the McBSP be operated with a 1 bit data delay.

```
/* CSL config structure for McBSP */
/* AIC24 connected to MCBSP #2 */

const MCBSP_Config mcbspCfgAic24 = {
    MCBSP_SPCR1_RMK(
        MCBSP_SPCR1_DLB_OFF,                   /* DLB      = 0 */
        MCBSP_SPCR1_RJUST_RZF,                 /* RJUST    = 0 */
        MCBSP_SPCR1_CLKSTP_DEFAULT,            /* CLKSTP   = 0 */
        MCBSP_SPCR1_DXENA_NA,                  /* DXENA    = 0 */
        MCBSP_SPCR1_RINTM_RRDY,                /* RINTM    = 0 */
        MCBSP_SPCR1_RRST_DISABLE               /* RRST     = 0 */
    ),
    MCBSP_SPCR2_RMK(
        MCBSP_SPCR2_FREE_YES,                  /* FREE     = 1 */
        MCBSP_SPCR2_SOFT_DEFAULT,              /* SOFT     = 0 */
        MCBSP_SPCR2_FRST_RESET,                /* FRST     = 0 */
        MCBSP_SPCR2_GRST_RESET,                /* GRST     = 0 */
        MCBSP_SPCR2_XINTM_XRDY,                /* XINTM    = 00 */
        MCBSP_SPCR2_XRST_DISABLE               /* XRST     = 0 */
    ),
    MCBSP_RCR1_RMK(
        MCBSP_RCR1_RFRLEN1_OF(1),              /* RFRLEN1  = 1 ;2 words/frame */
        MCBSP_RCR1_RWDLEN1_16BIT               /* RWDLEN1  = 2 ;16-bit words */
    ),
    MCBSP_RCR2_RMK(
        MCBSP_RCR2_RPHASE_SINGLE,              /* RPHASE   = 0 */
        MCBSP_RCR2_RFRLEN2_OF(0),              /* RFRLEN2  = 0 */
        MCBSP_RCR2_RWDLEN2_8BIT,               /* RWDLEN2  = 0 */
        MCBSP_RCR2_RCOMPAND_MSB,               /* RCOMPAND = 0 */
        MCBSP_RCR2_RFIG_YES,                   /* RFIG     = 0 */
        MCBSP_RCR2_RDATDLY_1BIT                /* RDATDLY  = 1 */
    ),
    MCBSP_XCR1_RMK(
        MCBSP_XCR1_XFRLEN1_OF(1),              /* XFRLEN1  = 1; 2 words/frame */
        MCBSP_XCR1_XWDLEN1_16BIT               /* XWDLEN1  = 2; 16-bit words */
    ),
    MCBSP_XCR2_RMK(
        MCBSP_XCR2_XPHASE_SINGLE,              /* XPHASE   = 0 */
        MCBSP_XCR2_XFRLEN2_OF(0),              /* XFRLEN2  = 0 */
        MCBSP_XCR2_XWDLEN2_8BIT,               /* XWDLEN2  = 0 */
        MCBSP_XCR2_XCOMPAND_MSB,               /* XCOMPAND = 0 */
        MCBSP_XCR2_XFIG_YES,                     /* XFIG      = 0 */
        MCBSP_XCR2_XDATDLY_1BIT                  /* XDATDLY   = 1 */
    ),
    MCBSP_SRGR1_RMK(
        MCBSP_SRGR1_FWID_OF(0),                  /* FWID      = 0 */
        MCBSP_SRGR1_CLKGDV_OF(0)                 /* CLKGDV    = 0 */
    ),
    MCBSP_SRGR2_RMK(
        MCBSP_SRGR2_GSYNC_FREE,                  /* FREE      = 0 */
        MCBSP_SRGR2_CLKSP_RISING,                /* CLKSP     = 0 */
        MCBSP_SRGR2_CLKSM_DEFAULT,               /* CLKSM     = 0 */
        MCBSP_SRGR2_FSGM_DXR2XSR,                /* FSGM      = 0 */
        MCBSP_SRGR2_FPER_OF(0)                   /* FPER      = 0 */
    ),

    MCBSP_MCR1_DEFAULT,
    MCBSP_MCR2_DEFAULT,
    MCBSP_PCR_RMK(
        MCBSP_PCR_XIOEN_SP,                      /* XIOEN     = 0 */
        MCBSP_PCR_RIOEN_SP,                      /* RIOEN     = 0 */
        MCBSP_PCR_FSXM_EXTERNAL,                 /* FSXM      = 0 */
        MCBSP_PCR_FSRM_EXTERNAL,                 /* FSRM      = 0 */
        MCBSP_PCR_CLKXM_INPUT,                   /* CLKXM     = 0 */
        MCBSP_PCR_CLKRM_INPUT,                   /* CLKRM     = 0 */
        MCBSP_PCR_FSXP_ACTIVEHIGH,               /* FSXP      = 0 */
        MCBSP_PCR_FSRP_ACTIVEHIGH,               /* FSRP      = 0 */
        MCBSP_PCR_CLKXP_RISING,                  /* CLKXP     = 0 */
        MCBSP_PCR_CLKRP_FALLING                  /* CLKRP     = 0 */
    ),
    MCBSP_RCERA_DEFAULT,
    MCBSP_RCERB_DEFAULT,
    MCBSP_XCERA_DEFAULT,
    MCBSP_XCERB_DEFAULT
};
```

The CSL DMA structures are also defined in this file. One structure is provided for the receive channel and one structure for the transmit channel.

```
/*  CSL config structure for AIC24 DMAs */


const DMA_Config dmaRxCfgAic24 = {
    0x0000,                     /*  Channel Priority (0x0000 or 0x0001)  */
    0xC055,                     /*  Transfer Mode Control Register (DMMCR)  */
    0x3000,                     /*  Sync Event and Frame Count Register (DMSFC)  */
    (DMA_AdrPtr)0x0031,         /*  Source Address Register (DMSRC) - Numeric  */
    (DMA_AdrPtr)0x0000,         /*  Destination Address Register (DMDST) - Numeric  */
    0x0000,                     /*  Element Count Register (DMCTR)  */
};

const DMA_Config dmaTxCfgAic24 = {
    0x0000,                     /*  Channel Priority (0x0000 or 0x0001)  */
    0xC541,                     /*  Transfer Mode Control Register (DMMCR)  */
    0x3000,                     /*  Sync Event and Frame Count Register (DMSFC)  */
    (DMA_AdrPtr)0x0000,         /*  Source Address Register (DMSRC) - Numeric  */
    (DMA_AdrPtr)0x0033,         /*  Destination Address Register (DMDST) */
     0x0000,                    /*  Element Count Register (DMCTR)  */
};
```

Some remarks about the DMA channel configurations:

- Autoinitialization

  The HFK system uses the autoinitialization of the DMA channels enabled (AUTOINIT = 1).

- DMA Interrupt Generation

  The interrupts generated by the DMA transfers are defined in the DMA Transfer Mode Control (DMMCRn) register. In the HFK system interrupts are generated ONLY by the receive DMA channel. The transmit DMA channel does NOT generate interrupts.

- DMA Synchronization Events

  The DMA synchronization events are defined in the DSYN field of the DMA Sync Event and Frame Count (DMSFCn) Register. In the HFK system BOTH DMA channels are synchronized to the McBSP 2 receive event (REVT2). This setting enables to synchronize the receive and transmit channel and to use only the receive DMA channel interrupt.

- Addressing Modes

  In the HFK system the DMA channels operate in multiframe mode. This mode of operation is selected by setting to zero the CTMOD bit in the DMMCR. In multiframe mode, the data transfers can be structured as multiple elements in a frame and multiple frames in a block. The HFK system uses the element index register DMIDX0 to step to next value within a frame and the frame-index register DMFRI0 to step to next frame. The source and destination address transfer mode bits are defined in the SIND and DIND fields in the DMMCR. The HFK system uses the Postincrement with index offset (DMIDX0 and DMFRIO) for both fields.

3.1.3.1 AIC24 Initialization Function

The initialization function of the AIC24 driver, HFK5407_AIC24_init() is defined in the hfk5407_aic24.c file.

This function calls the AIC24 configuration function AIC24_setParams() and passes a pointer to the default AIC24 configuration array. The configuration function AIC24_setParams () is defined in the file aic24.c and will be described in the next section. The default AIC24 configuration is defined in the file aic24.h.

```
/*
 *  ======== HFK5407_AIC24_init ========
 *  Should be called once at power up to init the AIC24
 */
void HFK5407_AIC24_init(void)
{
       Int                 eventRx;
       Uns         imask;
       volatile int dummy;

       HFK5407_AIC24_DevParams  *params;

       attrsAic24 =  HWI_ATTRS;  /* get all the defaults attrs */

       params = (HFK5407_AIC24_DevParams *)&defaultParams;


       /* AIC Configuration Function */
       AIC24_setParams(&(params->aic24));
```

After configuring the AIC24, the function initializes the params object with the structures that will be used to configure the McBSP port and DMA channels.

```
params->mcbspCfg = (MCBSP_Config*)&mcbspCfgAic24;
params->dmaRxCfg = (DMA_Config*)&dmaRxCfgAic24;
params->dmaTxCfg = (DMA_Config*)&dmaTxCfgAic24;
```

After these initial steps, the HFK5407_AIC24_init() function uses DSP/BIOS and CSL functions to disable the interrupts and to open and configure the McBSP port and DMA channels.

```
// Disable interrupts globally (INTM)
imask = HWI_disable();
/* open and configure the McBSP */
params->hMcbsp = MCBSP_open(params->McbspId, MCBSP_OPEN_RESET);

MCBSP_config(params->hMcbsp, params->mcbspCfg);

/* open and configure the DMAs */
params->hDmaRx = DMA_open(params->dmaRxId, DMA_OPEN_RESET);
params->hDmaTx = DMA_open(params->dmaTxId, DMA_OPEN_RESET);

DMA_config(params->hDmaRx , params->dmaRxCfg);
DMA_config(params->hDmaTx , params->dmaTxCfg);
```

Then, the function completes the initialization of the DMA controller. First, some fields in the DMA channel priority and enable control (DMPREC) Register are set.

- The FREE bit is set in order for the DMA transfers to continue even if the emulator is stopped.
- The INTOSEL field is set in order to activate the DMA interrupts for the DMA channels 0 and 1.
- The AUTOIX bit is set in order to enable the presence of autoinitialization registers for all the DMA channels.

```
/* beware ... this is a global register! */
DMA_FSET(DMPREC, FREE, 1);

/* using DMA 0, 1 for AIC24 RX, TX */
DMA_FSET(DMPREC,INTOSEL,DMA_DMPREC_INTOSEL_CH0_TO_CH5);

/* Enable AUTOINIT mode for each DMA: Set AUTOIX =1 in DMPREC */
DMPREC = DMPREC|0x4000;
```

Second, the DMA channel 0 and 1 autoinitialization registers are initialized.

```
/* Initialize Autoinit Registers */
/* DMA RX: 0; DMA TX 1 */

DMSBAR  = DMGSA0_SUBADDR;
DMSBAI  = MCBSP2_DRR1_ADDR;          /* McBSP2 data rx register */
DMSBAI  = dmaBufPtrs[AIC24_IDX].RcvBufs[1];  /* RX buffer pong address */
DMSBAI  = 1;                         /* 2 words per frame*/
DMSBAI  = 0x3000|(FRAME_SIZE_8K-1);

DMSBAR  = DMGSA1_SUBADDR;
DMSBAI  = dmaBufPtrs[AIC24_IDX].SndBufs[1];   /* TX buffer pong address */
DMSBAI  = MCBSP2_DXR1_ADDR;           /* McBSP2 data tx register */
DMSBAI  = 1;                          /* 2 words per frame*/
DMSBAI  = 0x3000|(FRAME_SIZE_8K-1);
```

Third, complete the initialization of the DMA active registers.

```
        /* Complete initialization of the DMAs */

        /******************************************************************************
        * Configure DMA Channel 0 - Receive McBsp2(AIC24)                             *
        ******************************************************************************/
        DMSBAR = DMSRC0_SUBADDR;                /* point to DMSRC0 register */
        DMSBAI = MCBSP2_DRR1_ADDR;              /* McBSP2 data rx register  */
        DMSBAI = dmaBufPtrs[AIC24_IDX].RcvBufs[0];     /* RX buffer ping address  */
        DMSBAI = 1;                     /* 2 words per frame*/
//      DMSBAI = 0x3000|(FRAME_SIZE_16K-1);
        DMSBAI = 0x3000|(FRAME_SIZE_8K-1);      /* REVT2 */

    /******************************************************************************
     * Configure DMA Channel 1 - Transmit McBsp2       (AIC24)       SYNC=0010     *
     ******************************************************************************/
        DMSBAR = DMSRC1_SUBADDR;               /* point to DMSRC1 register  */
        DMSBAI = dmaBufPtrs[AIC24_IDX].SndBufs[0];     /* TX buffer ping address      */
        DMSBAI = MCBSP2_DXR1_ADDR;      /* McBSP2 data tx register */
        DMSBAI = 1;                     /* 2 words per frame */
//      DMSBAI = 0x3000|(FRAME_SIZE_16K-1);
        DMSBAI = 0x3000|(FRAME_SIZE_8K-1);      /* REVT2 */

        /******************************************************************************
         * Multi-frame for(AIC24) 2 channels (i.e. 2 buffer of length FRAME_SIZE_8K  *
         ******************************************************************************/
        DMSBAR  = DMIDX0_SUBADDR;
//      DMSBAN  = FRAME_SIZE_16K;  /* DMIDX0 = step to next sample in frame */
        DMSBAN  = FRAME_SIZE_8K;
        DMSBAR  = DMFRI0_SUBADDR;
//      DMSBANI = 1-(FRAME_SIZE_16K); /* DMFRI0 = step to next frame */
        DMSBANI = 1-(FRAME_SIZE_8K);
```

Figure 2 shows the organization of the DMA buffers in the HFK system. The DMA transfers will alternate between the ping/pong buffers. Each ping/pong buffer has a size of 2xFRAME_SIZE_8K and is divided in a channel 1 and a channel 2 buffer. In one DMA transfer a ping/pong buffer is transferred. As a result 2xFRAME_SIZE_8K samples are transferred during each transfer.

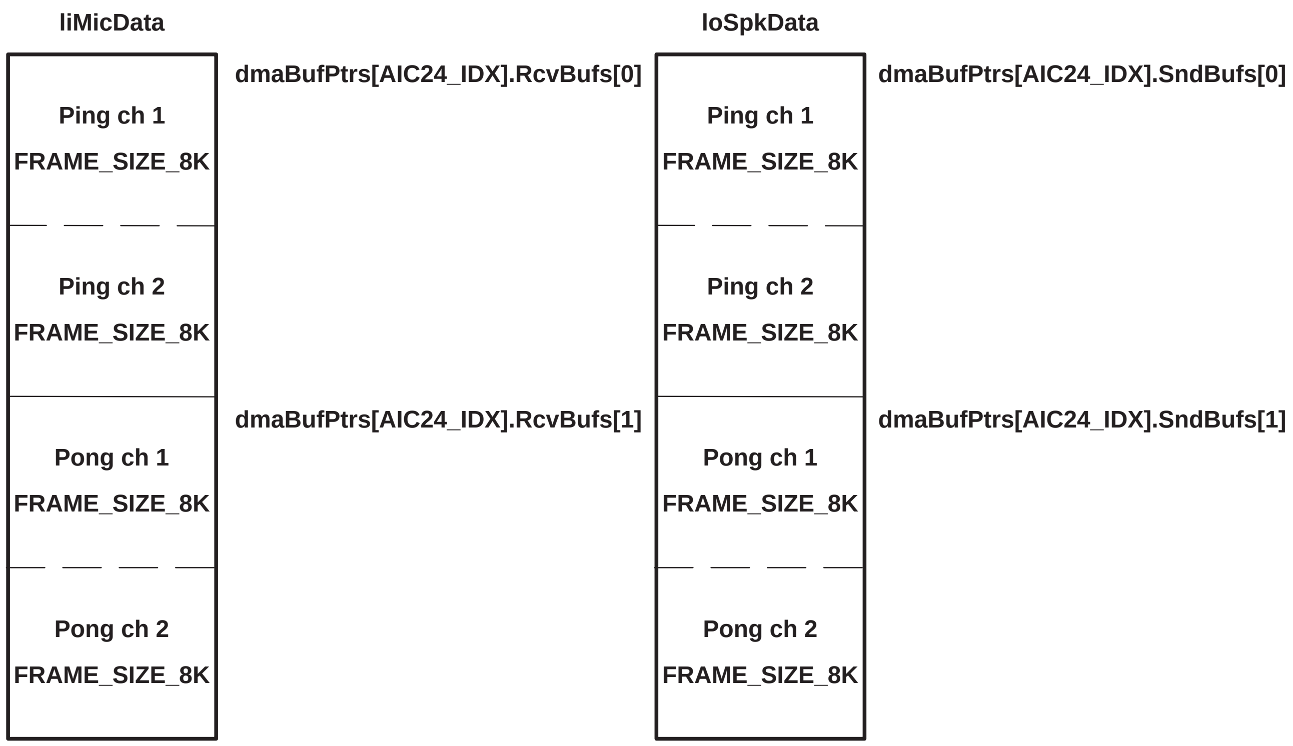

liMicData
loSpkData
dmaBufPtrs[AIC24_IDX].RcvBufs[0]
dmaBufPtrs[AIC24_IDX].SndBufs[0]
Ping ch 1
FRAME_SIZE_8K
Ping ch 2
FRAME_SIZE_8K
dmaBufPtrs[AIC24_IDX].RcvBufs[1]
dmaBufPtrs[AIC24_IDX].SndBufs[1]
Pong ch 1
FRAME_SIZE_8K
Pong ch 2
FRAME_SIZE_8K

Figure 2. DMA Buffers in the HFK System

After completing the initialization of the DMA controller, the initialization function plugs the AIC24 ISR, HFK5407_AIC24_DMA_isr, in the vector table using the HWI_dispatchPlug DSP/BIOS function.

A structure of type HWI_Attrs is used to specify the HWI's dispatcher properties. Only the intMask value is of interest. All the other values are the default ones. The function uses an intrMask value of 0xFFFF in order to mask off all interrupts while executing the HWI.

The DMA receive channel interrupt is enabled and the McBSP port and DMA channels are started. Before exiting the initialization function the global interrupts are enabled.

```
eventRx = DMA_getEventId(params->hDmaRx);
/* Clear any pending DMA Rx  interrupts */
IRQ_clear (eventRx);
/* plug interrupt vector using DSP/BIOS HWI Dispatcher */
/* Disable all interrupts during the ISR, IMR = 0x0000 */
attrsAic24.intrMask = 0xFFFF;
HWI_dispatchPlug(eventRx, (Fxn)HFK5407_AIC24_DMA_isr, &attrsAic24);
/* RESET McBsp RX and Tx registers*/

 MCBSP_RSETH(params->hMcbsp,DXR1,0);
 MCBSP_RSETH(params->hMcbsp,DRR1,0);

/* Enable DMA Rx  interrupts */
IRQ_enable(eventRx);
imask = HWI_disable();
/* Start DMAs */
DMA_start(params->hDmaRx);
DMA_start(params->hDmaTx);
```

```
        /* Start MCBSPs */
        MCBSP_start(params->hMcbsp, MCBSP_RCV_START, 0x0);
        MCBSP_start(params->hMcbsp, MCBSP_XMIT_START, 0x0);

        /* Purge McBsp Rcv Buffer */
        dummy=MCBSP_read16(params->hMcbsp);
        dummy=MCBSP_read16(params->hMcbsp);

        /* Clear any pending DMA Rx interrupts */
        IRQ_clear (eventRx);

        HWI_restore(imask);
        // Enable interrupts globally (INTM)
        IRQ_globalEnable();

}
```

3.1.3.2 AIC24 Interrupt Service Routine

The AIC24 ISR is implemented in the file hfk5407_aic24.c. The ISR is called when the DMA receive channel interrupt occurs at the completion of the transfer.

The ISR updates the DMA autoinitialization registers based on the value of the bufActive global flag. After updating the autoinitialization registers, the ISR updates the bufActive flag and posts the processing SWI, swiAudio.

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *  AIC24 Interrupt Service Routine
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */


void HFK5407_AIC24_DMA_isr(void)
// Called when Rx DMA completes transfer
{
        unsigned short st;
        volatile unsigned short tmpreg;

        /* Save DMA address register */
        tmpreg  = DMSBAR;

        st = (bufActive)?0:1;

        DMSBAR  = DMGSA0_SUBADDR;
        DMSBAI  = MCBSP2_DRR1_ADDR;    /* McBSP2 data receive register  */
        DMSBAI  = dma_sync[AIC24_IDX].RcvBufs[st];/* Receieve buffer address */
        DMSBAI  = 1;                      /* 2 words per frame*/
        DMSBAI  = FRAME_SIZE_8K-1;
//      DMSBAI  = FRAME_SIZE_16K-1;

        DMSBAR  = DMGSA1_SUBADDR;
        DMSBAI  = dma_sync[AIC24_IDX].SndBufs[st];/* Transmit buffer address */
        DMSBAI  = MCBSP2_DXR1_ADDR;       /* McBSP2 data transmit register */
        DMSBAI  = 1;                      /* 2 words per frame*/
        DMSBAI  = FRAME_SIZE_8K-1;
//      DMSBAI  = FRAME_SIZE_16K-1;
```

```
        /* Restore DMA address register */
        DMSBAR=tmpreg;

        /* Update BufferActive Flag */
        BufferActive = (BufferActive)?0:1;

        /* Post SWI */
        SWI_post(&swiHfkAudio);

}
```

3.1.4 Aic24.c

The file aic24.c implements the AIC24 configuration function AIC24_setParams ().

The AIC24 is programmed through the serial port 2 (McBSP 2). The McBSP is programmed using the Chip Support Library (CSL) with the following initialization structure:

```
/*This config will be used only for AIC24 initialization*/

const MCBSP_Config mcbspCfgAic24Init = {
    MCBSP_SPCR1_RMK(
        MCBSP_SPCR1_DLB_OFF,                   /* DLB      = 0 */
        MCBSP_SPCR1_RJUST_RSE,                 /* RJUST    = 1 */
        MCBSP_SPCR1_CLKSTP_DEFAULT,            /* CLKSTP   = 0 */
        MCBSP_SPCR1_DXENA_NA,                  /* DXENA    = 0 */
        MCBSP_SPCR1_RINTM_RRDY,                /* RINTM    = 0 */
        MCBSP_SPCR1_RRST_DISABLE               /* RRST     = 0 */
    ),
    MCBSP_SPCR2_RMK(
        MCBSP_SPCR2_FREE_YES,                  /* FREE     = 1 */
        MCBSP_SPCR2_SOFT_DEFAULT,              /* SOFT     = 0 */
        MCBSP_SPCR2_FRST_RESET,                /* FRST     = 0 */
        MCBSP_SPCR2_GRST_RESET,                /* GRST     = 0 */
        MCBSP_SPCR2_XINTM_FRM,                 /* XINTM    = 10 ; XINT generated by
FS*/
        MCBSP_SPCR2_XRST_DISABLE               /* XRST     = 0 */
    ),
    MCBSP_RCR1_RMK(
        MCBSP_RCR1_RFRLEN1_OF(3),              /* RFRLEN1  = 1; 4 words/rx frame */
        MCBSP_RCR1_RWDLEN1_16BIT                /* RWDLEN1  = 2; 16-bit worfds */
    ),
    MCBSP_RCR2_RMK(
        MCBSP_RCR2_RPHASE_SINGLE,              /* RPHASE   = 0 */
        MCBSP_RCR2_RFRLEN2_OF(0),              /* RFRLEN2  = 0 */
        MCBSP_RCR2_RWDLEN2_8BIT,               /* RWDLEN2  = 0 */
        MCBSP_RCR2_RCOMPAND_MSB,               /* RCOMPAND = 0 */
        MCBSP_RCR2_RFIG_YES,                   /* RFIG     = 0 */
        MCBSP_RCR2_RDATDLY_1BIT                /* RDATDLY  = 1 */
    ),
    MCBSP_XCR1_RMK(
        MCBSP_XCR1_XFRLEN1_OF(3),              /* XFRLEN1  = 1 ; 4 words/tx prog frame
*/
        MCBSP_XCR1_XWDLEN1_16BIT               /* XWDLEN1  = 2 ; 16-bit words*/
    ),
    MCBSP_XCR2_RMK(
        MCBSP_XCR2_XPHASE_SINGLE,              /* XPHASE   = 0 */
        MCBSP_XCR2_XFRLEN2_OF(0),              /* XFRLEN2  = 0 */
        MCBSP_XCR2_XWDLEN2_8BIT,               /* XWDLEN2  = 0 */
        MCBSP_XCR2_XCOMPAND_MSB,               /* XCOMPAND = 0 */
        MCBSP_XCR2_XFIG_YES,                    /* XFIG     = 0 */
        MCBSP_XCR2_XDATDLY_1BIT                 /* XDATDLY  = 1 */
    ),
    MCBSP_SRGR1_RMK(
        MCBSP_SRGR1_FWID_OF(0),                 /* FWID     = 0 */
        MCBSP_SRGR1_CLKGDV_OF(0)                /* CLKGDV   = 0 */
    ),
    MCBSP_SRGR2_RMK(
        MCBSP_SRGR2_GSYNC_FREE,                 /* FREE     = 0 */
        MCBSP_SRGR2_CLKSP_RISING,               /* CLKSP    = 0 */
        MCBSP_SRGR2_CLKSM_DEFAULT,              /* CLKSM    = 0 */
        MCBSP_SRGR2_FSGM_DXR2XSR,               /* FSGM     = 0 */
        MCBSP_SRGR2_FPER_OF(13)                 /* FPER     = 0*/
    ),
    MCBSP_MCR1_DEFAULT,
    MCBSP_MCR2_DEFAULT,
    MCBSP_PCR_RMK(
        MCBSP_PCR_XIOEN_SP,                     /* XIOEN    = 0 */
        MCBSP_PCR_RIOEN_SP,                     /* RIOEN    = 0 */
        MCBSP_PCR_FSXM_EXTERNAL,                /* FSXM     = 0 */
        MCBSP_PCR_FSRM_EXTERNAL,                /* FSRM     = 0 */
        MCBSP_PCR_CLKXM_INPUT,                  /* CLKXM    = 0 */
        MCBSP_PCR_CLKRM_INPUT,                  /* CLKRM    = 0 */
        MCBSP_PCR_FSXP_ACTIVEHIGH,              /* FSXP     = 0 */
        MCBSP_PCR_FSRP_ACTIVEHIGH,              /* FSRP     = 0 */
        MCBSP_PCR_CLKXP_RISING,                 /* CLKXP    = 0 */
        MCBSP_PCR_CLKRP_FALLING                 /* CLKRP    = 0 */
    ),
    MCBSP_RCERA_DEFAULT,
    MCBSP_RCERB_DEFAULT,
    MCBSP_XCERA_DEFAULT,
    MCBSP_XCERB_DEFAULT
};
```

The AIC24_setParams() function disables first the global interrupts using the DSP/BIOS HWI_disable() function.

```
Void AIC24_setParams(AIC24_Params *params)
{
    Uns imask;
    unsigned int XmtEventId;

    MCBSP_Handle hMcbsp;


  /* Temporarily disable all maskable interrupts, preserving  */
  /* previous state of INTM                                   */
    imask = HWI_disable();
```

Then, the INTOSEL field of the DMPREC register is set in order to activate the McBSP 2 transmit interrupt (BXINT2)

```
       /* using DMA 0, 1 for AIC24 RX, TX */
    DMA_FSET(DMPREC,INTOSEL,DMA_DMPREC_INTOSEL_CH4_CH5);
```

The McBSP 2 is opened and configured using CSL functions and the ISR is plugged in the interrupt vector table using the HWI_dispatchPlug() DSP/BIOS function. Then, the McBSP 2 transmit interrupt is enabled. At the end of the function the global interrupts are enabled and the McBSP 2 is started.

```
/* open and configure McBSP */

hMcbsp = MCBSP_open(MCBSP_PORT2, MCBSP_OPEN_RESET);
MCBSP_config(hMcbsp,(MCBSP_Config*) &mcbspCfgAic24Init);

 /* get the event ID's for the McBSP TX interrupt */
 /* provides the location of the TX INT in the IFR */
XmtEventId = MCBSP_getXmtEventId(hMcbsp);

/* Clear any pending MCBSP #2 TX interrupts */
IRQ_clear(XmtEventId);


/* plug interrupt vector using HWI_dispatchPlug() */
   HWI_dispatchPlug(XmtEventId, (Fxn)ISR_frameSync, NULL);

/* Enable MCBSP #2 TX interrupt */
IRQ_enable(XmtEventId);


   // Enable interrupts globally (INTM)
IRQ_globalEnable();


/* Start McBSP */
MCBSP_start(hMcbsp, MCBSP_XMIT_START, 100);
```

The McBSP 2 Tx interrupt is used in order to synchronize the transmission of control words with the frame synch signal. After the AIC24 comes out of reset it is operating in programming mode. In this mode every frame has 4 slots: 2 data slots and 2 control slots. A special function static void AIC24_writeControlWords () was implemented to manage the transmission of the control words. It is to notice that this function sends 0 during the frame following the valid data transmission frame in order to avoid over writing the channel 1 control word with the channel 2 control word which is still present in the MCBSP Data Transmit Register (DXR) when the next frame synch occurs. The value 0 is not a valid command and therefore will not alter the values already programmed.

```
void AIC24_writeControlWords(MCBSP_Handle hMcbsp,unsigned int EventId,int datch1, int
datch2)
{
       /* Wait for next Frame Synch start */
        AIC24_waitNextFS();


       /* write dummy data in the Data Frame slots */
       // Slot 0 written with Garbage    ; //CH1 (Slot0) Data Frame
       // Garbage = content of XSR when FS occurs
       AIC24_write(hMcbsp,0)             ; //CH2 (Slot1) Data Frame

       /* write control data in the Control Frame slots */
       AIC24_write(hMcbsp, datch1)       ; //CH1 (Slot2) Control Frame
       AIC24_write(hMcbsp, datch2)       ; //CH2 (Slot3) Control Frame


    /* wait one FS for new config to take effect */
    AIC24_waitNextFS();
```

```
        // Slot 0 written with Garbage    ; //CH1 (Slot0) Data Frame
        // Garbage = content of XSR and FS occurs
        AIC24_write(hMcbsp,0)             ; //CH2 (Slot1) Data Frame

        /* write control data in the Control Frame slots */
        AIC24_write(hMcbsp, 0)            ; //CH1 (Slot2) Control Frame – NOP
        AIC24_write(hMcbsp, 0)            ; //CH2 (Slot3) Control Frame – NOP

}
```

There are some more functions which are used in the file aic24.c to perform the codec programming. Since these functions are easy to understand they are not described here. The reader is invited to inspect these functions and study how they are used.

4 Modifying the RF3 for the HFK System

The standard Reference Framework (RF) package is available for download on the Texas Instruments website. The tutorial provided in this application report is based on RF version 2.1. The source code of the reference framework is provided in the HFK SDK in the folder HFK\Src\misc\RF2.1 At the time of publication RF version 2.2 was also available, however there was no need to upgrade to RF2.2 since this release only brings in support for two new C6000 DSKs. This tutorial was designed to provide the concepts of the integration of the HFK system in an RF environment .

The HFK system uses the RF3 which is best suited for static applications which do not require many channels. For more detailed information about RF3 the reader is referred to *Reference Frameworks for eXpressDSP Software: RF3 A Flexible, Multichannel, Multi-Algorithm, Static System* (SPRA793). The following section is an excerpt of this document.

4.1 The RF3 Folder Structure

The Reference Framework folder tree contains application sources and library modules. You can begin to explore RF3 by examining the folder tree that contains the application and associated files. We recommend that you retain the provided structure for your development.

Figure 3 shows the folders used by RF3 and highlights some important files they contain.

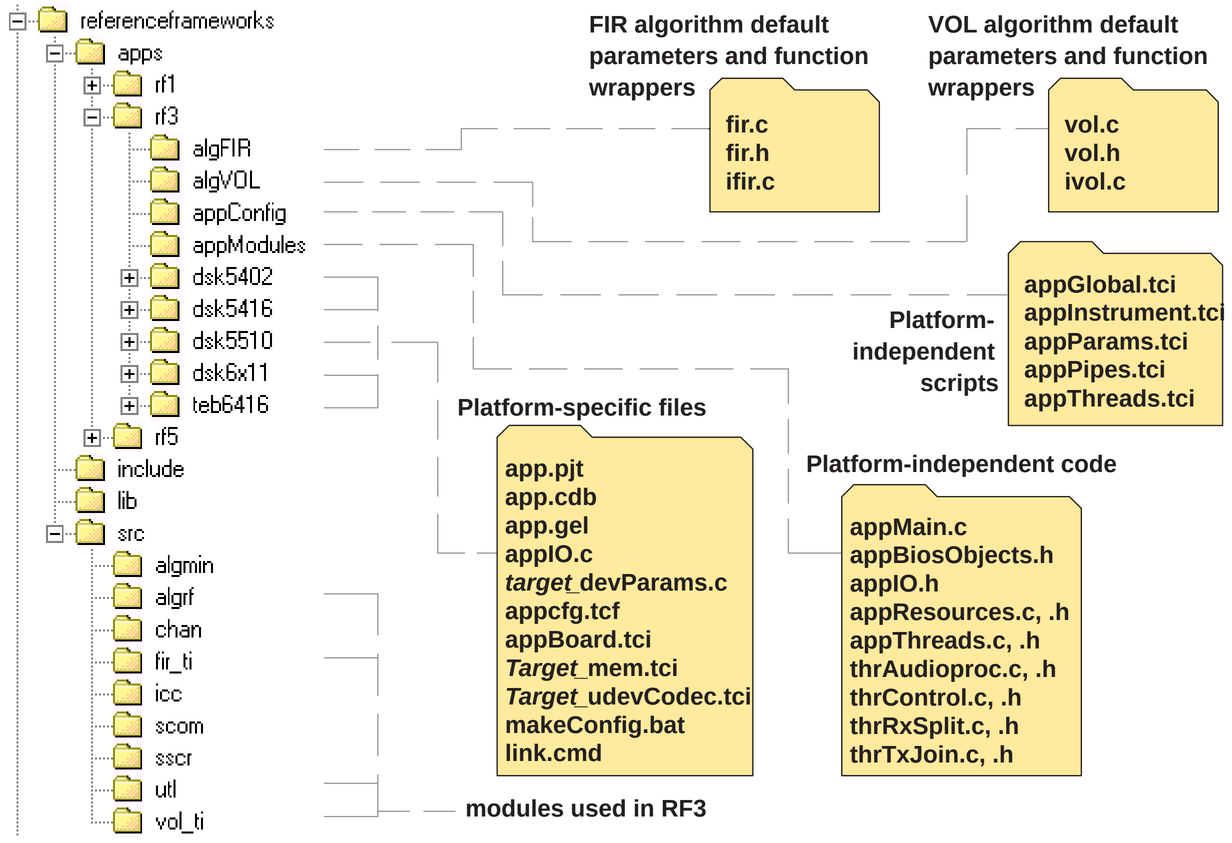


Figure 3. RF3 Folder Structure

Folders to notice include:

- **apps\rf3.** Contains the Code Composer Studio projects that make up RF3. To modify RF3, make a copy of the rf3\ tree at the same folder level, and modify the copy.
  - **appConfig.** Contains the hardware-independent script files for the RF3 configuration.
  - **appModules.** Contains the hardware-independent files for the RF3 application. This includes the implementation of threads (thr*.* files) and the initialization code (app*.*).
  - **target.** Contains hardware-specific files for the RF3 application. This includes the configuration files, the project file, and the linker file. These files are placed in platform-named folders so that RF3 can be provided for multiple platforms.
  - **algFIR.** Contains application-side wrappers for communicating with the FIR algorithm. Your algorithms can be wrapped similarly to match the RF3 structure. For example, you might create an algMP3 folder.
  - **algVOL.** Contains application-side wrappers for communicating with the VOL algorithm.
- **include.** Contains a number of public header files used by Reference Frameworks. RF3 uses some, but not all, of these header files.

  Public header files are referenced by both algorithm and framework code. In contrast, private header files are stored with the source code that includes them and are not intended for use by other modules. Each library module has one header file in this folder.

- **lib.** Contains a number of library files linked in with Reference Framework applications. RF3 uses some, but not all, of these libraries. Each library module has one library per DSP chip in this folder.
- **src.** Contains folders with source files for the modules in the include and lib folders. The readme.txt files in each of these folders provide information about the modules and their use. Library modules typically need little or no modification. The modules used by RF3 are:
  - **algrf.** Contains source files for the IALG implementation used by RF3 to instantiate eXpressDSP-compliant algorithms using the DSP/BIOS MEM module for dynamic memory allocation.
  - **fir_ti.** Contains files for the Finite Impulse Response (FIR) filter, an eXpressDSP-compliant algorithm provided by Texas Instruments and used by default in RF3.
  - **vol_ti.** Contains files for the Volume algorithm, an eXpressDSP-compliant algorithm provided by Texas Instruments and used by default in RF3.
  - **utl.** Contains source files for the UTL debugging and diagnostics module.

  The Reference Frameworks distribution does not include source code for IOM device driver modules. Such files can be obtained as part of the DSP/BIOS Driver Developer's Kit (DDK). You do not need the DDK in order to run the Reference Frameworks—the driver library and public header files are included in the Reference Frameworks distribution. For details about the DDK and mini-driver development and use, see the *DSP/BIOS Driver Developer's Guide* (SPRU616).

4.2 Customizing RF3 for the HFK System

This section describes how to customize the RF3 package for the HFK system. The information in this section is presented as a Lab that will help the reader to understand step by step the process.

LAB 1A: Removing folders and files not required by HFK system

1. Start with the standard RF package:

   Start Windows Explorer and extract the file sprc063.zip available in the HFK\Src\misc\RF2.1 folder. A self extractable file will be produced. This file will extract to a zip file rf_v2_10_00_11.zip . Unzip this file to a folder called RF3_myHFK
2. Delete the RF1 and RF5 folders:

   RF3_myHFK\referenceframeworks\apps\r1

   RF3_myHFK\referenceframeworks\apps\rf5
3. Delete the batch files used to build the projects for the RF platforms:

   RF3_myHFK\referenceframeworks\build.bat

   RF3_myHFK\referenceframeworks\build5000.bat

   RF3_myHFK\referenceframeworks\build6000.bat
4. Delete ALL source code folders EXCEPT:

   RF3_myHFK\referenceframeworks\src\algmin

   RF3_myHFK\referenceframeworks\src\algrf

   RF3_myHFK\referenceframeworks\src\utl

5. Delete ALL libraries EXCEPT:

   HFK\RF3_myHFK\referenceframeworks\lib\algmin.l54, algmin.l54f

   HFK\RF3_myHFK\referenceframeworks\lib\algrf.l54, algrf.l54f

   HFK\RF3_myHFK\referenceframeworks\lib\utl.l54, utl.l54f

6. Delete ALL include files EXCEPT:

   RF3_myHFK\referenceframeworks\include\algmin.h

   RF3_myHFK\referenceframeworks\include\algrf.h

   RF3_myHFK\referenceframeworks\include\utl.h

7. Delete ALL the board specific application folders EXCEPT:

   RF3_myHFK\referenceframeworks\apps\rf3\dsk5402

8. Delete the XDAIS wrapper function folders:

   RF3_myHFK\referenceframeworks\apps\rf3\algFIR

   RF3_myHFK\referenceframeworks\apps\rf3\algVOL

9. Delete the following Applications Modules Files:

   RF3_myHFK\referenceframeworks\apps\rf3\appModules\ appIO.h

   RF3_myHFK\referenceframeworks\apps\rf3\dsk5402\ appIO.c

   RF3_myHFK\referenceframeworks\apps\rf3\dsk5402\ dsk5402_devParams.c

   RF3_myHFK\referenceframeworks\apps\rf3\appModules\ thrRxSplit.c,h

   RF3_myHFK\referenceframeworks\apps\rf3\appModules\ thrTxSplit.c,h

   RF3_myHFK\referenceframeworks\apps\rf3\appModules\ thrControl.c,h

   RF3_myHFK\referenceframeworks\apps\rf3\appModules\ thrAudioproc.c,h

LAB 1B: Preparing the project folder for the HFK 5407 EVM

In this Lab the dsk5402 application folder will be modified for the HFK 5407 platform.

10. Change the name of the RF3_myHFK\referenceframeworks\apps\rf3\dsk5402 folder to RF3_myHFK\referenceframeworks\apps\rf3\hfk5407_aic24

11. Open in Code Composer Studio the project called app.pjt located in RF3_myHFK\referenceframeworks\apps\rf3\hfk5407_aic24\app.pjt

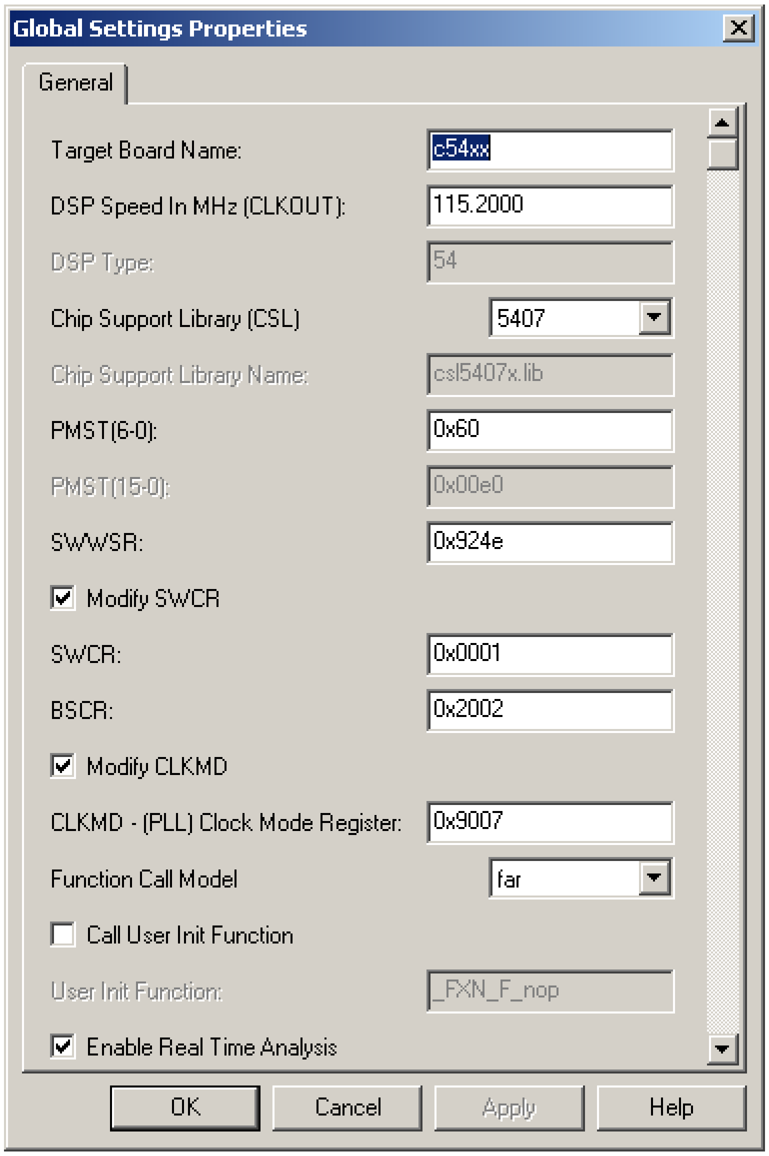
    A new Code Composer Studio window will open with the message “Cannot find file ..”. Select “Remove”. Repeat this action as many times as prompted to remove the files from the project.

Updating the global settings

12. In the Code Composer Studio Project View window open the DSP/BIOS Config folder. Open the configuration file app.cdb by double-clicking the file in the Project View window.

13. Open the System folder, right-click on Global Settings and inspect its properties.

We need to initialize the settings with the following values for the HFK 5407 EVM:


Global Settings Properties

General

Target Board Name:	c54xx
DSP Speed In MHz (CLKOUT):	115.2000
DSP Type:	54
Chip Support Library (CSL)	5407
Chip Support Library Name:	csl5407x.lib
PMST(6-0):	0x60
PMST(15-0):	0x00e0
SWWSR:	0x924e
✓ Modify SWCR	
SWCR:	0x0001
BSCR:	0x2002
✓ Modify CLKMD	
CLKMD - (PLL) Clock Mode Register:	0x9007
Function Call Model	far
☐ Call User Init Function	
User Init Function:	_FXN_F_nop
✓ Enable Real Time Analysis	

14. Select Apply and OK.

Removing the DSP/BIOS objects not required by the HFK system

15. In the Scheduling: CLK-Clock Manager folder remove the clkControl object by right-clicking and by selecting Delete.
16. In the Scheduling: SWI folder remove all the SWI objects.
17. In the Input/Output: PIP folder remove all the PIP objects.
18. In the Input/Output: Device Drivers: User-Defined Devices folder remove the udevCodec object
19. Save and close the DSP/BIOS config file app.cdb

5 Integrating the AIC24 Driver in RF3

This section describes how to integrate the AIC24 codec driver in RF3. The information in this section is presented as a Lab.

LAB 2: Integrating the AIC24 driver with the customized RF3

1. Add the AIC24 driver files to RF3.

   Copy the folders:

   HFK\Src\drivers
   HFK\Src\evmInit

   to

   RF3_myHFK\referenceframeworks\src.

2. Copy the header files to the RF3 include folder.

   Copy the files:

   RF3_myHFK\referenceframeworks\src \evmInit\ hfk5407.h
   RF3_myHFK\referenceframeworks\src \drivers\ include\aic24.h
   RF3_myHFK\referenceframeworks\src \drivers\ include\hfk5407_aic24.h
   RF3_myHFK\referenceframeworks\src \drivers\ include\hfk5407_drvGbl.h

   to

   RF3_myHFK\referenceframeworks\include

Creating the HFK audio processing thread

3. Create the files thrHfkAudio_aic24.c and thrHfkAudio.h and save them to the RF3_myHFK\referenceframeworks\apps\rf3\appModules folder.

   These files can be created by using the New File icon in the Code Composer Studio window. The thrHfkAudio_aic24.c is specific to this example whereas the file thrHfkAudio.h is generic and will not be updated for future examples.

4. In the file thrHfkAudio.h include the following:

```
#ifndef THRHFKAUDIO_
#define THRHFKAUDIO_

#include ”appResources.h”   /* application–wide common info */
#include ”appThreads.h”     /* thread–wide common info */

#ifdef __cplusplus
extern ”C” {
#endif
```

   Include the declaration of the thread functions:

```
/*
 *  Declaration of ”public” thread functions
 */
/* init function, called from thrInit() */
extern Void thrHfkAudioInit( Void );


/* run function, called by the swiHfkAudio SWI object */
extern Void thrHfkAudioRun( Void );


#ifdef __cplusplus
}
#endif /* extern ”C” */


#endif /* THRHFKAUDIO_ */
```

5. Save and close the file thrHfkAudio.h.
6. Open the file thrHfkAudio_aic24.c and include the following:

```
#include <std.h>
#include <utl.h>            /* debug/diagnostics utility functions */


#include ”appResources.h”   /* application–wide common info */
#include ”appThreads.h”     /* thread–wide common info */
#include ”thrHfkAudio.h”        /* definition of thrHfkAudio object */


#include <hfk5407_drvGbl.h>   /* driver global varaibles */
#include <hfk5407.h>        /* hfk5407  board info */
#include <hfk5407_aic24.h>   /* aic24 driver info */


#include <xdas.h>    /* XDAIS types definition */
```

Next define the HfkAudio thread initialization function. This function calls the AIC24 initialization functions.

```
/*
 *  ========= thrHfkAudioInit ========
 *  Initialization of data structures for the thread, called from
 *  appThreads.c:thrInit() at init time.
 */
Void thrHfkAudioInit( Void )
{
HFK5407_AIC24_init();
HFK5407_audioBuffer_init();

}
```

Finally, define the HfkAudio thread processing function. This function will be called by the SWI swiHfkAudio that will be defined in the next sections. This function will determine the active buffers and will perform the data move.

```
/*
 *  ========= thrHfkAudioRun ========
 *  The ”body” of the swiHfkAudio thread.
 *
 */
Void thrHfkAudioRun( Void )
{
        XDAS_Int16 *aicSrcBufL, *aicDstBufL;
        XDAS_Int16 *aicSrcBufR, *aicDstBufR;
        XDAS_Int16 i,off;

        off = bufActive*FRAME_SIZE_8K*2;

        aicSrcBufL                 = (XDAS_Int16*)&liMicData[off];     //INP3  Tip
        aicDstBufL                 = (XDAS_Int16*)&loSpkData[off];     //OUT2  Tip
        off += FRAME_SIZE_8K;
        aicSrcBufR                 = (XDAS_Int16*)&liMicData[off];     // INP2 Ring
        aicDstBufR                 = (XDAS_Int16*)&loSpkData[off];     // OUT3 Ring

    /*
     * Do the data move. Mask off the low bit for compatibility with
     * those codecs that interpret a low bit of ’1’ as a command flag.
     */
       for (i = 0; i < FRAME_SIZE_8K; i++) {
        /*  */
        aicDstBufL[i] = aicSrcBufL[i] & 0xfffe;
        aicDstBufR[i] = aicSrcBufR[i] & 0xfffe;
        }
}
```

Save and close the file.

7. Save and close the file thrHfkAudio_aic24.c

Modifying the application files

8. In the file RF3_myHFK\referenceframeworks\apps\rf3\appModules\appMain.c remove the references to the appIO.h file and to the appIOInit() and appIOPrime() functions.
9. Add the call to the HFK5407 EVM initialization function, HFK5407_init() BEFORE the call to the threads initialization function thrInit().

```
/*
 *  ======== appMain.c ========
 *
 *  Startup module for application initialization: first initializes the
 *  I/O, then initializes all the threads, and primes the I/O pipes.
 *  Any application–specific module initialization should be called from here.
 */
#include <std.h>
#include <utl.h>            /* debug/diagnostics utility functions */
#include ”appResources.h”   /* application–wide common info */
#include ”appThreads.h”     /* thread–wide common info (thrInit() decl.) */
/*
 *  ======== main ========
 *  main() initializes IO, threads, and anything else the application
 *  might need to initialize
 */
Void main()
{
    /* name LOG objects to be used for error/warning/general/debug messages */
    UTL_setLogs( &logTrace, &logTrace, &logTrace, &logTrace );
    /* initialize the HFK5407 EVM */
     HFK5407_init();
    /* initialize all the threads */
    thrInit();
    /* greet the user */
    UTL_logDebug( ”Application started.” );
    /* and fall into BIOS idle loop */
    return;
}
```

10. Save and close the file appMain.c
11. In the file RF3_myHFK\referenceframeworks\apps\rf3\appModules\appThreads.h remove the reference to the intermediate buffer.

```
/*
 *  Declaration of intermediate buffers used by threads’ algorithms
 */
/* this buffer is used by the Audioproc thread. ”Sample” is usually 16b wide. */
extern Sample bufAudioproc[ FRAMELEN ];
```

12. In the file RF3_myHFK\referenceframeworks\apps\rf3\appModules\appThreads.c remove the reference to the header files of the threads that have been removed and include the header files of the HFK audio thread.
13. Remove the definition of the intermediate buffer.
14. Remove the thread initialization functions for the threads that were removed and add the one for the HFK audio thread.
15. Save and close the opened files.

```
/*
 *  ======== appThreads.c ========
 *
 *  Initialization of all threads, definition of global variables
 *  and buffers
 */
#include <std.h>
```

```
#include <algrf.h>
#include <utl.h>            /* debug/diagnostics utility functions */

#include ”appResources.h”   /* application–wide common info */
#include ”appThreads.h”     /* thread–wide common info */
#include ”thrHfkAudio.h”      /* definition of the HFK Audio thread */

/*
 *  ========= thrInit ========
 *  initialize all the threads that have Init() function
 */
Void thrInit( Void )
{
    /*
     *  Configure the ALGRF module to tell it the names of heaps for algorithms:
     *  1st argument – name of the heap in internal memory: INTERNALHEAP
     *  2nd argument – name of the heap in external memory: EXTERNALHEAP
     */
    ALGRF_setup( INTERNALHEAP, EXTERNALHEAP );

    /*
     *  Here we invoke specific individual initialization functions
     *  for all the threads that have one (some of them may be empty)
     */

    thrHfkAudioInit();       /* HFK audio thread   */

    /* show heap usage, now that all threads are initialized */
    UTL_showHeapUsage( INTERNALHEAP );
    UTL_showHeapUsage( EXTERNALHEAP );
}
```

Scheduling the audio processing thread: Adding the SWI object

16. In the Code Composer Studio Project View window open the DSP/BIOS Config folder and open the configuration file app.cdb.

17. In the Scheduling: Swi folder right-click on SWI and select Insert SWI. Rename the created SWI object swiHfkAudio.

18. Open the swiHfkAudio object (i.e., right-click on swiHfkAudio and select Properties) and set the following properties:

function:	_thrHfkAudioRun
priority:	1
mailbox:	0
arg0:	0x00000000
arg1:	0x00000000

19. Select OK.

Modifying the memory map

20. In the System:MEM folder right-click on the Memory Section Manager and inspect its properties. Inspect all the tabs in the menu and move the sections allocated in EPROG to IPROG and the sections allocated in EDATA to IDATA.

21. Select OK.

22. Open the EDATA memory section and modify the following properties:

base:	0xA000
len:	0x6000
heap size:	0x300
heap identifier label:	_EXTERNALHEAP

23. Open the EPROG memory section and modify the following properties:

base:	0x0418000
len:	0x8000

    The HFK EVM has a single external memory. However this external memory can be allocated in the DSP program space and in the DSP data space simultaneously. Since there is only one physical memory overlapping will occur. For more information about the external memory on the HFK Evm the reader is referred to the *Texas Instruments Hands-Free Kit Development Platform User's Guide* (SPRU703)

24. Open the IDATA memory section and modify the following properties:

base:	0x0100
len:	0x3f00
heap size:	0x300
heap identifier label:	_EXTERNALHEAP

25. Select OK.
26. Open the IPROG memory section and modify the following properties:

base:	0x04000
len:	0x4000

27. Select OK.
28. Open the VECT memory section and modify the following properties:

base:	0x0080

29. Select OK.
30. Save and Close the CDB file.

Adding the files to the project

31. Add the AIC24 driver files to the Code Composer Studio project. Right-click on the app.pjt folder in the Code Composer Studio Project View window. Select Add Files to Project → Add the following files.

    RF3_myHFK\referenceframeworks\src \evmInit\ hfk5407.c

    RF3_myHFK\referenceframeworks\src \drivers\ drv8kHz_nobsync\aic24.c

    RF3_myHFK\referenceframeworks\src \drivers\ drv8kHz_nobsync\hfk5407_aic24.c

RF3_myHFK\referenceframeworks\src \drivers\ drv8kHz_nobsync\hfk5407_drvGbl.c

32. Add the HFK audio thread definition file to the project.

    RF3_myHFK\referenceframeworks\apps\rf3\appModules\ thrHfkAudio_aic24.c

Changing the project build options

33. Select Project\ Build Options in the Code Composer Studio window.
34. Select the Category: Preprocessor. In the Preprocessor window remove the defined symbol APPMONOCODEC and change the defined symbol CHIP_5402 to CHIP_5407.

    In the Preprocessor window remove the include search paths: ..\algFIR;..\algVOL;
35. Select OK.

Changing the linker command file

36. Open the linker command file by clicking on link.cmd in the Code Composer Studio Project View window.
37. Remove from the linker command file all the libraries not required for this example.
38. Add in the linker command file the user defined .HFK_AUDIO memory section. This section contains the DMA buffers and global variables used by the AIC24 driver. This section MUST be allocated in internal data memory.
39. The resulting linker command file should be similar to:

```
/* include config-generated link command file */
–l appcfg.cmd

/* include the RF3 module implementing XDAIS algs. instantiation procedures */
–l algrf.l54f

/* include the UTL debugging module (if needed) */
–l utl.l54f


SECTIONS
{
     .HFK_AUDIO                                : {} > IDATA PAGE 1
}
```

40. Save and close the linker command file

Building the project

41. Select Project\Build in the Code Composer Studio window to build the project.
42. Load the Gel file provided in the HFK\Src\Gel folder
43. After loading the Gel, the GEL menu in the Code Composer Studio window enables to run the C5407_Configuration functions: CPU_Reset, C5407_Init. These functions should be executed each time before loading the program.
44. Load the program using the File → Load Program → menu
45. Connect an audio source (CD player) to the HFK microphone input. If no audio source is available the two on-board microphones M1 and M2 can be used as audio source.
46. Connect the speaker output to an amplified speaker.
47. Run the program. The audio signal should be heard on the speaker.

48. If no signal can be heard, check if the CSL has been updated in the Code Composer Studio used.

6 Conclusion

This application report described the AIC24 driver used by the HFK system and presented the step by step process to integrate this driver in RF3. The result of this integration is available in the HFK SDK in the folder HFK\Src\Examples\referenceframeworks\apps\rf3\hfk5407_aic24.

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless