Hands-Free Kit: Integration of the Stonestreet One Bluetooth Stack with the Clarity AEC Algorithm in RF3

Texas Instruments
Stonestreet One
Clarity Technologies

Software Applications

ABSTRACT

This application report describes the integration of the Stonestreet One (SS1) Bluetooth Stack with the Clarity Acoustic Echo Cancellation (AEC) algorithm in the Reference Framework 3 (RF3). The AEC algorithm considered is eXpressDSP-compliant.

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 Hands-Free Kit (HFK) understand how to integrate the software modules in RF3.

The application report presents first the integration of the SS1 Bluetooth Stack in RF3. Second, the report describes the buffer synchronization required between the Bluetooth audio interface and the AIC24 audio interface. Third, the integration of the SS1 Bluetooth Stack with the Clarity AEC algorithm in RF3 is presented.

Contents

List of Figures

1 Introduction

The Hands-Free Kit (HFK) described in this application report was developed as an open system based on the DSP/BIOS Reference Framework 3. This system was implemented on the HFK board that features a TMS320C5407 DSP and a TLV320AIC24 codec from Texas Instruments.

This application reports describes the integration of the SS1 Bluetooth stack with the Clarity eXpressDSP-compliant AEC algorithm in RF3. The example provided in the HFK software development kit (SDK) in the folder HFK\Src\Examples\referenceframeworks\apps\rf3\hfk5407_aic24 is used as a foundation for the integration of the Bluetooth stack with the AEC algorithm. The step-by-step process to build the example is described in the application report *Hands Free Kit: Integration of the AIC24 Driver in Reference Framework 3* (SPRA966).

The integration is presented in a tutorial format in order to help you understand the step by step process. The final result of the first section of this tutorial is provided in the SDK folder rf3\hfk5407_bt1. The second section of the tutorial describes the integration of the Bluetooth stack with the AIC24 driver and the synchronization between the two audio interfaces. The final result of the second section of this tutorial is provided in the SDK folder rf3\hfk5407_bt2. The third section of the tutorial describes the integration of the Bluetooth stack with the AEC algorithm and the final result is provided in the SDK folder rf3\hfk5407_bt3. The last section of the tutorial describes the integration of the Bluetooth stack with the AEC algorithm using 16kHz sampling frequency The final result is provided in the SDK folder rf3\hfk5407_bt4.

This tutorial was developed using the files provided in the HFK software development kit and Code Composer Studio v 2.20.

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 reader with the necessary steps to run the 8kHz Bluetooth and acoustic echo cancellation example, hfk5407_bt3, provided with the HFK SDK. This example is provided in the SDK versions that contain the BT stack and the AEC algorithm. The SDK versions that contain only the BT stack include the example hfk5407_bt2. Apply the following steps to the example hfk5407_bt2 if the example hfk5407_bt3 is not available.

Updating the CSL libraries

If the CSL libraries have already been updated, skip this section and continue at step 6.

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. In the temporary folder the temp\include and the temp\lib directories are created
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. Plug the BT daughter board in the HFK EVM.
7. Connect the JTAG cable to the HFK EVM.

   A BT phone fis needed in this example.
8. Connect an amplified speaker to the speaker output of the HFK EVM.
9. Power-up the HFK EVM

Running the example hfk5407_bt3

10. Start Code Composer Studio and open the project called app.pjt located in HFK\Src\Examples\referenceframeworks\apps\rf3\hfk5407_bt3.by using Project → Open Menu command.
11. Load the GEL file provided in HFK\Src\GEL.
12. In the Code Composer Studio DSP/BIOS menu, select Message Log to open a message log window.
13. Load the executable app.out from .\Debug subdirectory with File → Load Program
14. Run the code: Select Debug → Run from the Debug menu or press F5.
15. Pair the BT phone with the HFK EVM.
16. Call the BT phone.
17. When the phone rings, press button one on the HFK EVM (the button the furthest from the power supply connector) to accept the call. Connection is established.
18. The signal transmitted by the dialing phone should be heard on the amplified speaker.
19. Speak in the HFK EVM on-board microphones.

    The signal transmitted by the HFK EVM should be heard on the dialing phone

Inspecting the project files

20. On the left-hand side of the Code Composer Studio screen, use the project view window to examine the project.
21. Open the Source folder to examine the project files.
22. Open the file hfk5407_aic24_bsync.c which implements the aic24 codec driver. This file defines two functions: the function HFK5407_AIC24_init() (line 152) is used to initialize the aic24 driver and the function HFK5407_AIC24_DMA_isr() (line 305) is the aic24 driver interrupt service routine (ISR).
23. 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
24. Remove the breakpoint and close the file hfk5407_aic24_bsync.c.
25. Open the file hfk5407_brf6100_bsync.c which implements the Bluetooth audio driver. This file defines two functions: the function HFK5407_BRF6100_init() (line 261) is used to initialize the brf6100 driver and the function HFK5407_BRF6100_DMA_isr() (line 387) is the brf6100 driver interrupt service routine (ISR).
26. Open the file hfk5407_drvGbl_bsync.c. This file defines the function HFK5407_audioBuffer_init() (line 90) used to initialize the aic24 driver buffers. This function is called during the initialization process. This file also defines the function ToggleBuffer()(line 74) This function is called from the drivers' ISRs to swap the active buffers and post the swiHfkAudio SWI object.
27. The aic24 driver buffers liMicData[] and loSpkData[] are defined in the file hfk5407_drvGbl_bsync.c.
28. Close the file hfk5407_ drvGbl.c
29. 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

30. Select OK
31. The Graph Window will display the content of the buffer. In order to update the display right-click and select Refresh.
32. Open the file thrHfkAudio_bt3.c. This file defines two functions: the function thrHfkAudioInit() (line 43) is used to initialize the Hfk Audio thread and the function thrHfkAudioRun() (line 69) is the thread processing function associated with the swiHfkAudio SWI object.
33. The function thrHfkAudioRun() is called by the DSP/BIOS kernel after the swiHfkAudio SWI object is posted in the function ToggleBuffer(). In the example hfk5407_bt3 this function calls the AEC algorithm processing function.

3 The Bluetooth Stack

The Bluetooth stack provided in the HFK software development kit is available in the folder HFK\Src\BT. The stack includes:

- **Stack and profiles libraries**
  - HFK\Src\BT\lib\ Bluetopia54XX.l54f
  - HFK\Src\BT\lib\ HandsFree54XX.l54f
  - HFK\Src\BT\lib\ Headset54xx.l54f
- **Documentation**
  - HFK\Src\BT\doc
- **BT related include files**
  - HFK\Src\BT\include
- **BT related UART and HCI transport layer files**
  - HFK\Src\BT\src\hcitrans
  - HFK\Src\BT\src\hwuart

NOTE: The following sections present four examples. The first two examples presented in LAB1 and LAB2 apply to the SDK versions that include the Bluetooth stack and the AIC24 driver libraries. The last two examples presented in LAB3 and LAB4 apply to the SDK versions that include the Bluetooth stack , the AIC24 drivers and the AEC algorithm libraries.

4 Integrating the Bluetooth Stack in RF3

This section describes how to integrate the Bluetooth stack (BT) in RF3. The example presented in the application report *Hands-Free Kit Integration of the AIC24 Driver in Reference Framework 3* (SPRA966) is used as a starting point for this integration. In this example the BT stack is initialized and runs in an RF3 environment. The AIC24 driver runs side by side with the BT stack, however there is no interaction between the two modules.

Before starting LAB1 the reader is invited to read the application report *Bluetopia for RF3 Integration* (SPRA944) which presents a summary of the important aspects of the operation of the SS1 BT stack in the RF3 environment.

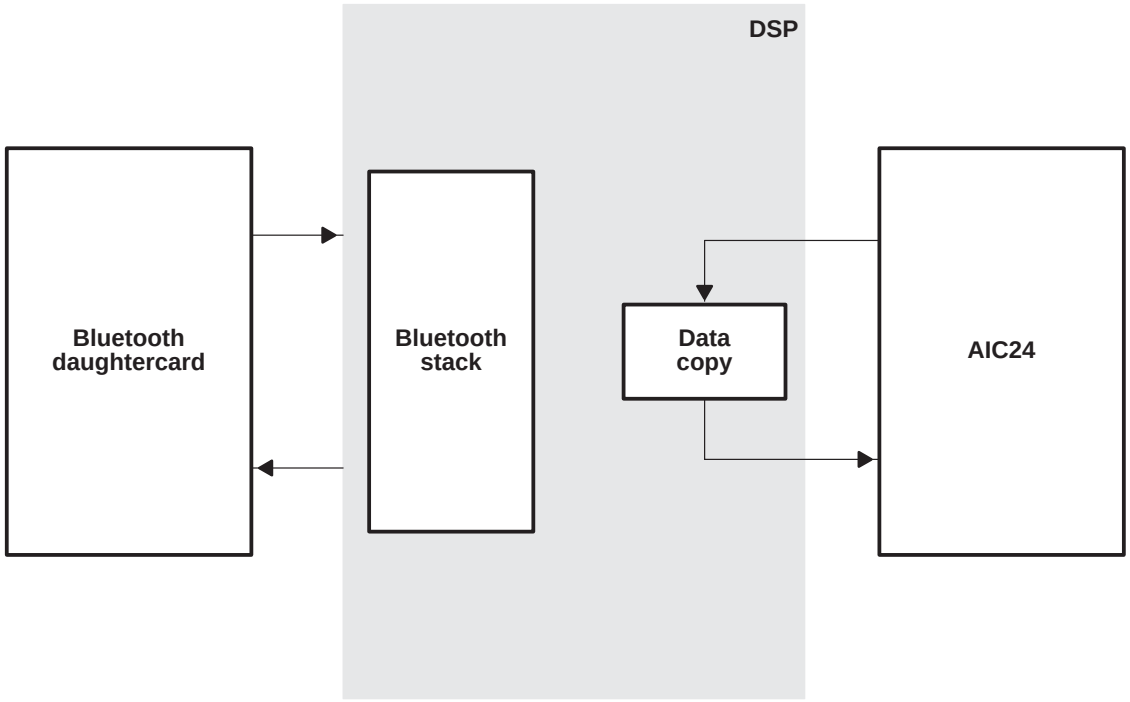


Figure 1. Block Diagram for LAB1 Example hfk5407_bt1

The reader should copy the content of the folder HFK\Src\misc\Labs\example_aic24\RF3_myHFK to a new working folder RF3_myHFK. Then, the folder RF3_myHFK\referenceframeworks\apps\rf3\hfk5407_aic24 should be copied in the same directory and the copy renamed hfk5407_bt1.

The folder RF3_myHFK\referenceframeworks\apps\rf3\hfk5407_bt1 will be the working folder for this example.

The information in this section is presented as a Lab that will help the reader to understand step by step the integration process.

LAB1: Integrating the BT stack in RF3.

Adding the BT related include files to the project

1. Copy the files from the BT folder
   HFK\Src\BT\include
   to the folder
   RF3_myHFK\referenceframeworks\include

Adding the BT related libraries to the project

2. Copy the libraries:
   HFK\Src\BT\lib\ Bluetopia54XX.l54f
   HFK\Src\BT\lib\ HandsFree54XX.l54f
   HFK\Src\BT\lib\ Headset54xx.l54f
   to the folder
   RF3_myHFK\referenceframeworks\lib

Adding the UART driver and the HCI transport layer files to RF3

3. Copy the folders

   HFK\Src\BT\src\hcitrans
   HFK\Src\BT\src\hwuart

   to the folder

   RF3_myHFK\referenceframeworks\src.

BT daughtercard reset function

4. In the HFK EVM initialization file, RF3_myHFK\referenceframeworks\src \evmInit\ hfk5407.c, add the call to the BT daughtercard reset function.

```
void HFK5407_init(void)
{
  /* Wake up the AIC24 from Reset*/
  AIC24_resetInit();

  /* Wake up the BT daughtercard from Reset*/
  BRF6100_resetInit();

}
```

Adding the DSP/BIOS CLK objects required by the BT stack

5. 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.
6. In the **Scheduling: CLK** folder right-click on **CLK-Clock Manager** and select **Insert CLK**. Right-click on newly inserted CLK, CLK0 and select **Rename**. Rename to clkBluetopiaTimerInterrupt.
7. Right-click on clkBluetopiaTimerInterrupt and select **Properties**. Set the properties for the CLK object to call _thrBluetopiaTimerIsr.
8. In the **Scheduling: CLK** folder right-click on **CLK-Clock Manager** and select **Insert CLK**. Right-click on newly inserted CLK, CLK0 and select **Rename**. Rename to clkControl.
9. Right-click on clkControl and select **Properties**. Set the properties for the CLK object to call _thrControlIsr.

Adding the DSP/BIOS SWI and HWI objects required by the BT stack.

10. In the **Scheduling: SWI** folder right-click on **SWI-Software Interrupt Manager** and select **Insert SWI**. Right In -click on newly inserted SWI, SWI0 and select **Rename**. Rename to swiBluetopiaCallbackThread.
11. Right-click on swiBluetopiaCallbackThread and select **Properties**. Set the properties for the SWI object to call _thrBluetopiaCallbackThread. Set the priority to 2. After making these changes click OK. For the other properties the default values will be used.
12. By following the same process add a new SWI called swiBluetopiaTimerThread. Set the properties for the SWI object to call _thrBluetopiaTimerThread. Set the priority to 2.
13. By following the same process add a new SWI called swiBluetopiaTransportThread. Set the properties for the SWI object to call _thrBluetopiaTransportThread. Set the priority to 3.

    The relative priority of the BT SWIs is very important for correct operation of the BT stack. The relative priorities of the BT software interrupts are discussed in detail in the *Bluetopia Integration in RF3* (SPRA944).

14. Add a new SWI called swiControl. Set the properties for the SWI object to call _thrControlRun. Set the priority to 1.
15. In the **Scheduling: HWI** folder right-click on **HWI_SINT14** and select **Properties**. Add the function *_C54XX_SS1_HWUART_Isr*. In the Dispatcher tab check the *Use Dispatcher* box.
16. Select OK.

Modifying the memory map and the linker command file

17. Open the **System:MEM** folder.
18. Delete the EPROG section.
19. Right-click on **MEM-Memory Section Manager** and insert the following new sections. All the sections will have:

    len: 0x8000

    space: code

    The bases are defined as follows:

    EXTRAM1 base: 0x418000

    EXTRAM2 base: 0x428000

    ... ...

    EXTRAM7 base: 0x478000
20. Right-click on **MEM-Memory Section Manager** and open the **Properties**. In the General window set the stack size to 0x0600.

    In the *Compiler Sections* tab check the box *"user .cmd File For Non-DSP/BIOS Sections"*. The non-DSP/BIOS sections will be defined in the user's linker command file.
21. Select OK to close the memory window.
22. Save and close the cdb file.
23. Open the linker command file link.cmd by clicking on it in the Code Composer Studio project window.
24. Add the BT related libraries to the linker command file.

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

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

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

/* include the SS1 Bluetopia Library */
-l bluetopia54XX.l54f

/* include the SS1 Headset Profile Library*/
-l headset54XX.l54f

/* include the SS1 Hands-Free Profile Library*/
-l handsfree54XX.l54f
```

25. Define the non-DSP/BIOS sections.

    The .bss and .const data memory sections will be placed in external memory.

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


   .bss:     {}  > EDATA                 PAGE 1
   .cio:     {} >  IDATA                 PAGE 1
   .data:    {} >  IDATA                 PAGE 1
   .const:   {} >  EDATA                 PAGE 1
   .cinit:   {} >  EXTRAM1               PAGE 0
   .pinit:   {} >  EXTRAM1               PAGE 0
   .switch:  {} >> EXTRAM1 | EXTRAM2     PAGE 0
   .text:    {} >> EXTRAM1 | EXTRAM2 | EXTRAM3 | EXTRAM4 | EXTRAM5 | EXTRAM6
                  | EXTRAM7 |            PAGE 0
}
```

26. Save and close the linker command file.

Creating the Control thread

The control thread files implement a simple User's Interface that enables to run a BT sample application. These files are provided in the HFK software development kit in the folder HFK\Src\misc\btSampleApplication.

27. Copy the files HFK\Src\misc\btSampleApplication\thrControl.c,.h to the folder RF3_myHFK\referenceframeworks\apps\rf3\appModules

28. Make a copy of the file appThreads.c in the same folder.

29. Rename this file appThreads_BT.c

30. Modify file appThreads_BT.c to include the Control thread header file and add the call to the thread initialization function.

```
#include "thrHfkAudio.h"      /* definition of the Hfk Audio thread */
#include "thrControl.h"      /* definition of Control 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   */
   thrControlInit();        /* Control thread */
    /* show heap usage, now that all threads are initialized */
    UTL_showHeapUsage( INTERNALHEAP );
    UTL_showHeapUsage( EXTERNALHEAP );
}
```

31. Save and close the appThreads_BT.c file.

Adding the files to the project

32. Add the following 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*…®

    RF3_myHFK\referenceframeworks\src \hcitrans\ hcitrans.c

    RF3_myHFK\referenceframeworks\src \hwuart\c54xx_ss1_hwuart.c

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

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

33. Remove the file appThreads.c from the Code Composer Studio project.

34. Copy the file

    RF3_myHFK\referenceframeworks\src\hwuart\c54xx_ss1_hwuart.h

    to the folder

    RF3_myHFK\referenceframeworks\include

Modifying the build options.

35. In the Code Composer Studio menu open *Project\Build Options*… Select the Preprocessor Category. In the *Include Search Path* window add the path for the AEC folder: ..\..\include;

36. Select OK to save and close the Build Options window.

Verifying the BT UART parameters.

37. Open the file RF3_myHFK\referenceframeworks\src \hcitrans\ hcitrans.h

    Inspect the value of HCITR_PROCESSOR_CLOCK_FREQUENCY. It is required that this value be equal to the one defined in the *DSP Speed In MHz (CLKOUT)* field of **System:Global Settings** in the cdb file.

38. Update the file if necessary

39. Save and close hcitrans.h.

Executing the code.

40. In the Code Composer Studio menu select *Project\Build* to build the project.

41. Load the GEL file provided in the HFK software development kit. Perform *CPU_Reset* and *C5407_Init* in the Code Composer Studio GEL menu.

42. Load the code.

43. In the Code Composer Studio DSP/BIOS menu select *Message Log* to open a message log window.

44. Run the code.

45. The following message should appear in the message log window:

```
0   Heap INTERNALHEAP: size = 256
1        used: 4 (1%)
2   Heap EXTERNALHEAP: size = 256
3        used: 0 (0%)
4   Application started.
5   Bluetooth Device Initialized successfully.
```

46. The reader is invited to read more about the sample application in the *Hands-Free Kit and Headset for the RF3 Reference Platform Sample Application* (SPRA843).

5 Integrating the BT Stack With the Audio Drivers in RF3

This section will first describe the audio drivers implemented to support the buffer synchronization in the HFK system. Second, the integration of the BT stack with these drivers will be described.

5.1 Buffer Synchronization in the HFK System − Overview

The HFK system integrates an AEC algorithm and a BT stack. It processes audio samples from these two different audio interfaces. The processing of the audio samples is performed by the AEC algorithm which requires that the audio interfaces be synchronized. Since the two audio interfaces are driven by different clocks there will be a difference between their sampling frequencies. This difference is compensated for in software in the HFK system by the buffer synchronization.

The buffer synchronization is implemented in the HFK system in the audio drivers. The concept is that the faster audio interface will drop samples until the slower interface recovers the delay.

In the HFK system the buffer synchronization is implemented by modifying the size of the DMA transfers and making the DMA transfers dependant on the global state of the system defined in a global variable.

Figure 2 shows that in an HFK system without buffer synchronization the DMA transfers have always the same size. This DMA configuration was implemented in the example hfk5407_aic24 provided in the HFK SDK. In the HFK system with buffer synchronization only the first DMA transfer has a size N. The following DMA transfers alternate sizes 1 and N−1. In the HFK system the DMA controller uses the autoinitialization registers. These registers are always programmed with a transfer size value of 1.

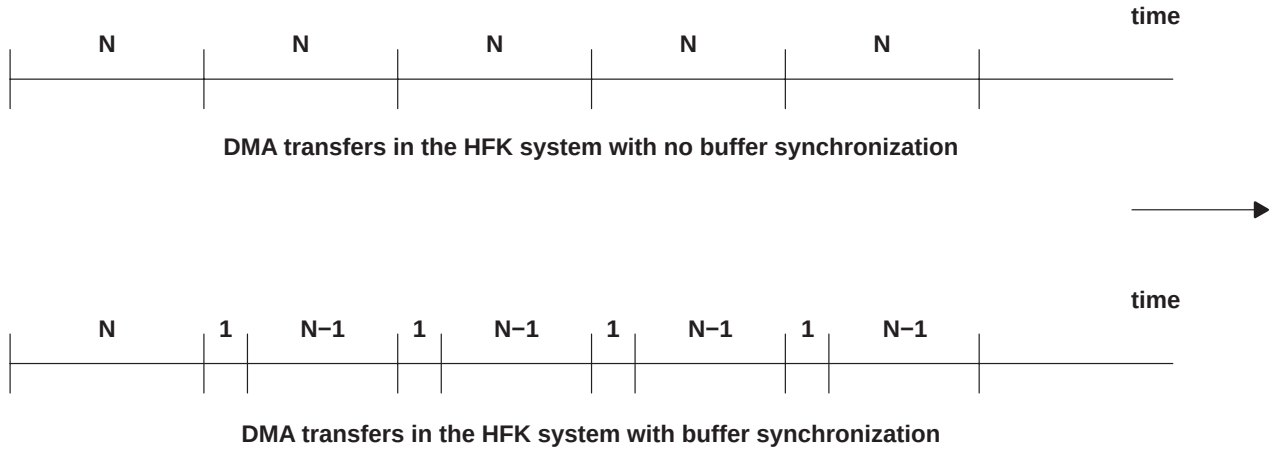


Figure 2. DMA Transfer Sizes in the HFK System With and Without Buffer Synchronization

In the HFK system the slowest audio interface will update the global state of the system. After completing a transfer of size 1, the DMA interrupt service routine will check the global state of the system and if the slower interface has not updated the system, i.e., it has not completed the previous N−1 transfer,(the slower interface has a delay larger than a sample) the faster interface will keep sending one sample. The faster interface will keep sending one sample until the synchronization is re-established.

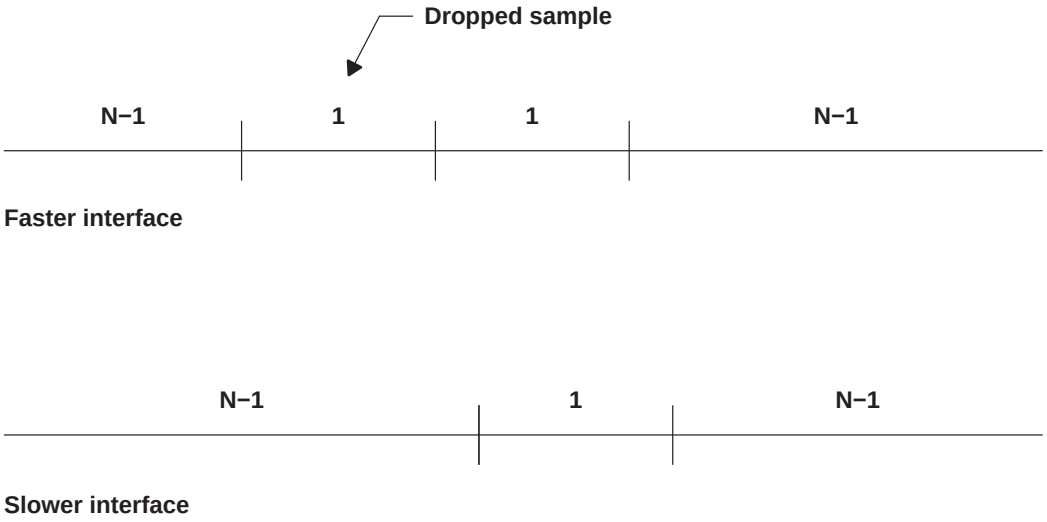
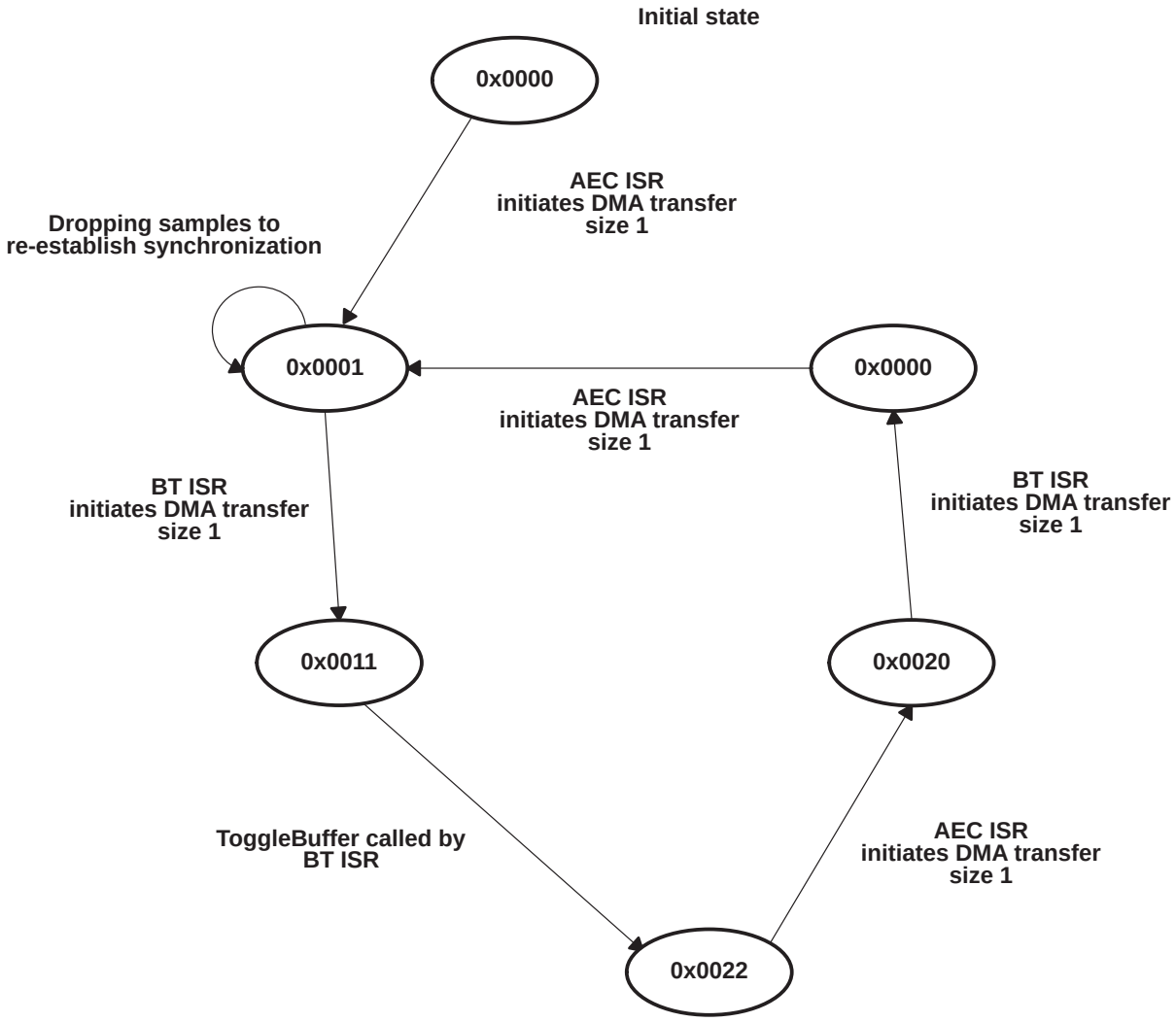


Figure 3. Synchronization Re-Established by Dropping a Sample in the HFK System

In the HFK system the buffer synchronization is accomplished by making the behavior of the AIC and BT ISRs dependant on a global variable, *dmaState*. This variable is updated in the AIC, BT ISRs and in a global function called *ToggleBuffer()*. This function is called only from the AIC and BT ISRs and it is used to swap the DMA buffers after the slowest interface has updated the dmaState flag. Figure 3 shows the flow diagram of the dmaState variable.



Assumption: The AEC audio interface is the faster interface.

Figure 4. Flow Diagram of the dmaState Global Variable

5.2 Buffer Synchronization in the HFK System – Driver Files

The drivers are implemented in the files aic24.c, hfk5407_aic24_bsync.c, hfk5407_brf6100_bsync.c and hfk5407_drvgbl_bsync.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 static void AIC24_resetInit(void) used to take the AIC24 out of reset and the function static void BRF6100_resetInit(void) used to take the BT daughter card out of reset.

- hfk5407_drvGbl_bsync.c

  The file hfk5407_drvGbl_bsync.c located in HFK\Src\drivers\drv8kHz_bsync defines and initializes the global variables and structures used by the AIC24 and BRF6100 drivers.

- hfk5407_aic24_bsync.c

  The file hfk5407_aic24_bsync.c located in HFK\Src\drivers\ drv8kHz_bsync 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\ drv8kHz_nobsync implements the Void AIC24_setParams(AIC24_Params *params) function that programs the AIC24 with the parameters defined in the file aic24.h

5.2.1 Hfk5407.c

The file hfk5407.c contains the HFK5407 EVM initialization functions. The function static void AIC24_resetInit(void)is used to take the AIC24 out of reset. The function static void BRF6100_resetInit(void)is used to take the BT daughter card out of reset. This is accomplished by writing a 1 in the BTRST 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).

5.2.2 Hfk5407_drvGbl_bsync.c

The file hfk5407_drvGbl_bsync.c defines global variables, structures and arrays used by the AIC24 and BRF6100 drivers. It also implements functions that initialize these global variables.

This file was created by modifying the file hfk5407_drvGbl.c used in the example hfk5407_aic24 and described in the application report *Hands-Free Kit Integration of the AIC24 Driver in Reference Framework 3* (SPRA966). The reader is invited to open the two files and to inspect them. The following additions have been made to the original file:

- BT data buffers defined.
- dmaState global variable used in the buffer synchronization.
- global function *ToggleBuffer()* used in the buffer synchronization. This function updates the *dmaState* variable and switches the ping/pong buffers by updating the *bufActive* global variable. The processing SWI is posted from this function.
- global debug variables *aicDovr* and *btDovr* added. These global variables are used in the ISRs to count the number of samples dropped by the interface.
- the DMA buffer set up includes the BT buffers

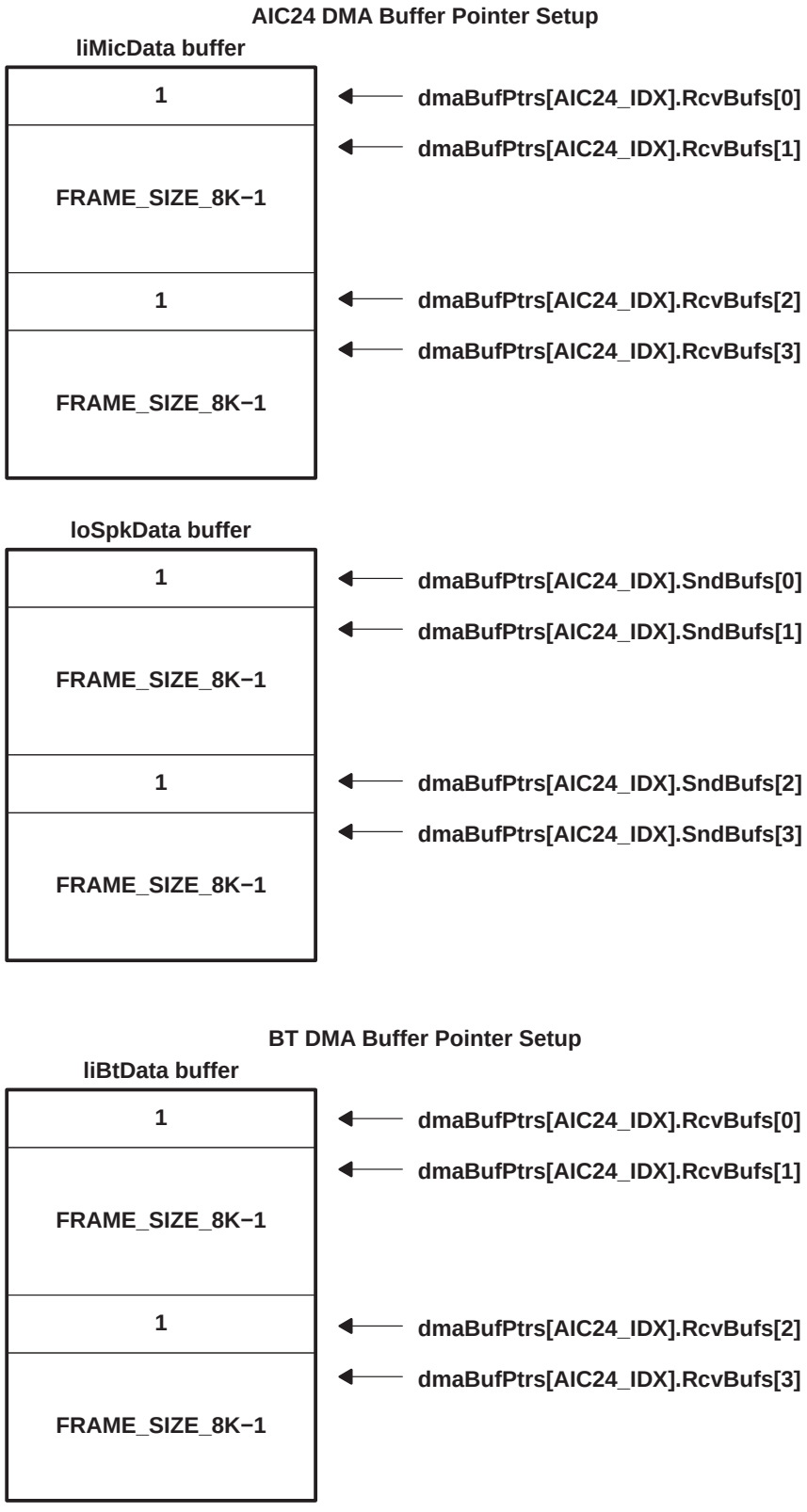


Figure 5. DMA Buffer Pointer Setup

5.2.3 Hfk5407_aic24_bsync.c

The file hfk5407_aic24_bsync.c implements the initialization function and the ISR of the AIC24 driver.

This file was created by modifying the file hfk5407_aic24.c used in the example hfk5407_aic24 and described in the application report *Hands-Free Kit integration of the AIC24 Driver in Reference Framework 3* (SPRA966). The reader is invited to open the two files and to inspect them. The following additions have been made to the original file:

- in the initialization function, *HFK5407_AIC24_init(),* the DMA controller registers are configured to perform a first transfer of FRAME_SIZE_8K frames whereas the DMA autoinitialization registers are configured to perform a transfer of one frame. Each frame is constituted of two words, one for each channel.
- the ISR *HFK5407_AIC24_DMA_isr()* has been modified to support the buffer synchronization.
- global function *ToggleBuffer()* used in the buffer synchronization. This function updates the *dmaState* variable and switches the ping/pong buffers by updating the *bufActive* global variable. The processing SWI is posted from this function.
- global debug variables *aicDovr* and *btDovr* added. These global variables are used in the ISRs to count the number of samples dropped by the interface.

5.2.4 Hfk5407_brf6100_bsync.c

The file hfk5407_brf6100_bsync.c implements the initialization function and the ISR of the BRF6100 driver.

The implementation of the BRF6100 driver is similar to the one of the AIC24 driver. The reader is invited to open the two files and inspect them. There are two main differences between the two drivers:

- the BRF6100 driver is a digital interface, there is no codec that transforms the audio signal into a digital one. Therefore there is no codec configuration function similar to the function *AIC24_setParams()* in the AIC24 codec.
- the BT audio interface has a single channel. There is no need to use the index register in the DMA controller.

The BRF6100 daughter card is connected to the the serial port McBSP 1. The implementation of the BRF6100 driver uses the DMA channels 4 and 5.

5.2.5 Aic24.c

The file aic24.c has not been modified.

5.3 Integration of the BT Stack With the Audio Drivers in RF3

This section describes how to integrate the BT stack with the audio drivers in RF3. The starting point of this integration is the example hfk5407_bt1 developed in the previous section. The information in this section is presented as a Lab that will help you to understand the integration process.

First, copy the content of the folder HFK\Src\misc\Labs\example_bt1\RF3_myHFK to a new working folder RF3_myHFK. Then, the folder RF3_myHFK\referenceframeworks\apps\rf3\hfk5407_bt1 should be copied in the same directory and the copy renamed hfk5407_bt2.

The folder RF3_myHFK\referenceframeworks\apps\rf3\hfk5407_bt2 will be the working folder for this example.

The information in this section is presented as a Lab.

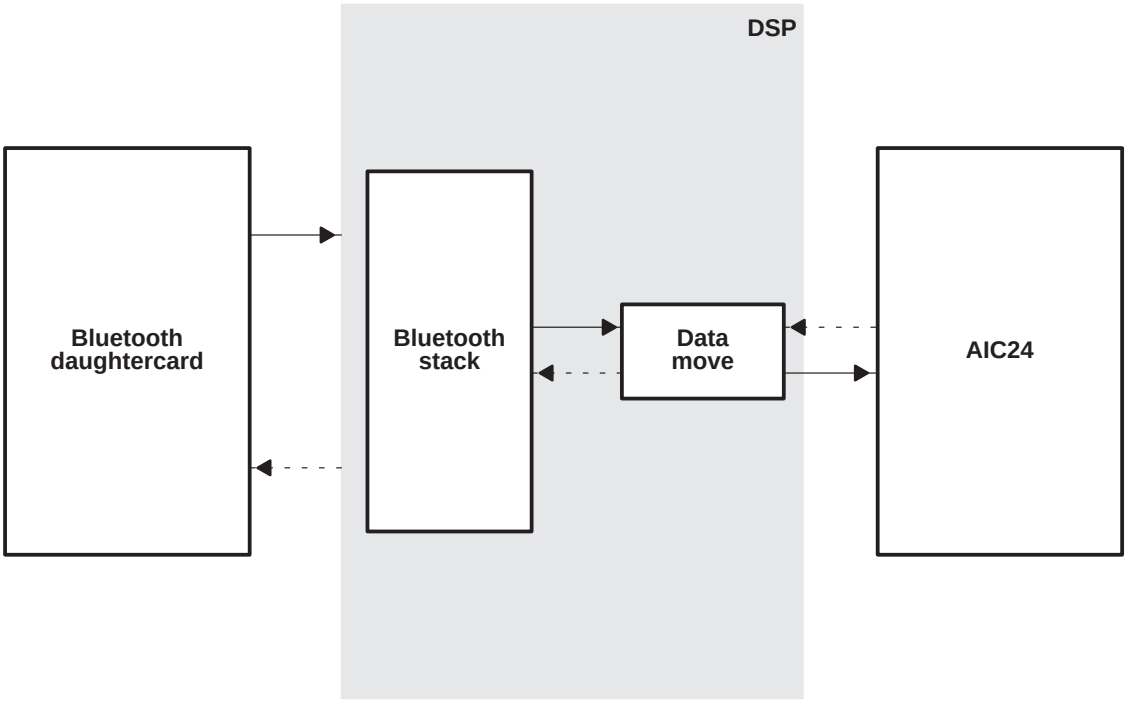


Figure 6. Block Diagram for LAB2, Example hfk5407_bt2

LAB 2: Integrating the BT stack with the audio drivers in RF3

Adding the driver files to the project

1. Open in Code Composer Studio the project called app.pjt located in RF3_myHFK\referenceframeworks\apps\rf3\hfk5407_bt2\app.pjt
2. In the Code Composer Studio Project View window open the *Source* folder. Remove the files hfk5407_aic24.c and hfk5407_drvGbl.c from the project (right-click, select Remove).
3. Add the following driver files to the project:
   \drivers\drv8kHz_bsync\hfk5407_aic24_bsync.c
   \drivers\drv8kHz_bsync\hfk5407_brf6100_bsync.c
   \drivers\drv8kHz_bsync\hfk5407_drvGbl_bsync.c

4. Copy the following header files to the include folder of the project:

   \drivers\drv8kHz_bsync\hfk5407_brf6100_bsync.h

   \drivers\drv8kHz_bsync\hfk5407_drvGbl_bsync.h

Modifying the audio processing thread

5. Open the file RF3_myHFK\referenceframeworks\apps\rf3\appModules\thrHfkAudio_aic24.c. Make a copy of this file in the same folder. Rename the copy thrHfkAudio_bt2.c
6. Open the file thrHfkAudio_bt2.c.
7. Include the header file hfk5407_drvGbl_bsync.h instead of hfk5407_drvGbl.h
8. Include the header file hfk5407_brf6100_bsync.h.

```
#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_bsync.h>   /* driver global variables */
#include <hfk5407_brf6100_bsync.h>  /* brf6100 driver info */
#include <hfk5407.h>                /* hfk5407  board info */
#include <hfk5407_aic24.h>          /* aic24 driver info */

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

9. Add in the thrHfkAudioInit() function the call to the function that takes the BT daughter card out of reset.

```
/*
 *  ========= thrHfkAudioInit ========
 *  Initialization of data structures for the thread, called from
 *  appThreads.c:thrInit() at init time.
 */
Void thrHfkAudioInit( Void )
{
HFK5407_AIC24_init();
HFK5407_BRF6100_init();
HFK5407_audioBuffer_init();
}
```

10. In the thrHfkAudioRun() function define the pointers to the BT DMA buffers and perform the data move.
11. In the Code Composer Studio Project View window open the *Source* folder. Remove the file thrHfkAudio_aic24.c from the project (right-click, select Remove).

12. Add the file RF3_myHFK\referenceframeworks\apps\rf3\appModules\thrHfkAudio_bt2.c to the project.

```
/*
 *  ========= thrHfkAudioRun ========
 *  The ”body” of the swiHfkAudio thread.
 *
 */
Void thrHfkAudioRun( Void )
{
    XDAS_Int16 *aicSrcBufL, *aicDstBufL;
    XDAS_Int16 *aicSrcBufR, *aicDstBufR;
    XDAS_Int16 *brf6100SrcBuf, *brf6100DstBuf;

       XDAS_Int16 i,off;

       off = bufActive*FRAME_SIZE_8K;
       brf6100SrcBuf = (XDAS_Int16*)&liBtData[off];
       brf6100DstBuf = (XDAS_Int16*)&loBtData[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++)
    {
       /* BT RX is sent to AIC TX R Channel */
       aicDstBufR[i] = brf6100SrcBuf[i] & 0xfffe;
       /* AIC RX R Channel is sent to BT TX*/
       brf6100DstBuf[i] = aicSrcBufR[i];

       }


}
```

Prepare the Hardware setup

Before executing the code the following setup is required:

13. The BT daughter card plugged in the HFK EVM.
14. Amplified speaker connected to HFK EVM speaker output.
15. Audio source connected to the HFK Mic input. The on board microphones available on the HFK EVM can be used as an alternative audio source.

Executing the code

16. In the Code Composer Studio menu select *Project\Build* to build the project.
17. Load the GEL file provided in the HFK software development kit. Perform *CPU_Reset* and *C5407_Init* in the Code Composer Studio GEL menu.
18. Load the code.
19. In the Code Composer Studio DSP/BIOS menu select *Message Log* to open a message log window.
20. Run the code.

21. The following message should appear in the message log window:

```
0   Heap INTERNALHEAP: size = 768
1        used: 4 (0%)
2   Heap EXTERNALHEAP: size = 768
3        used: 0 (0%)
4   Application started.
5   Bluetooth Device Initialized successfully.
```

22. Pair the BT phone with the HFK EVM.

    Even if the pairing process is dependant on the BT phone used the following steps should be present:

    - Select Bluetooth in the phone menu
    - Search for the device
    - Provide the pin 0x0000
    - Connect to the HFK5407 device

23. After the pairing process is completed the following should be displayed in the message log window:

```
0   Heap INTERNALHEAP: size = 768
1        used: 4 (0%)
2   Heap EXTERNALHEAP: size = 768
3        used: 0 (0%)
4   Application started.
5   Bluetooth Device Initialized successfully.
6   DeviceState: hsDisconnected -> hsHandsFreeConnected.
7   Service Level Connection Indication
```

24. Press a phone key. A sound should be heard on the speaker. In the message log window a message will signal that the BT stack started an audio connection. This connection will be terminated after some time if the user does not press a key again.

```
0   Heap INTERNALHEAP: size = 768
1        used: 4 (0%)
2   Heap EXTERNALHEAP: size = 768
3        used: 0 (0%)
4   Application started.
5   Bluetooth Device Initialized successfully.
6   DeviceState: hsDisconnected -> hsHandsFreeConnected.
7   Service Level Connection Indication.
8   Audio Connect Indication, Status: 0.
9   Audio Disconnect Indication.
```

25. Dial the BT phone.
26. When the phone rings press button one on the HFK EVM (the button the furthest from the power supply connector) to accept the call.
27. The signal transmitted by the dialing phone should be heard on the amplified speaker.
28. The signal transmitted by the HFK EVM should be heard on the dialing phone.

6 Integrating the BT Stack With the AEC

This section describes how to integrate the BT stack with the AEC algorithm in RF3. The starting point of this integration is the example hfk540_bt2 developed in the previous section.

The integration of the AEC algorithm in RF3 was described in the application report *Hands-Free Kit: Integration of the Clarity Acoustic Echo Cancellation (AEC) Algorithm in Reference Framework 3* (SPRA969). Since the BT stack acts as an audio interface the integration steps of the AEC algorithm will be identical to the ones described in the previously mentioned application report.

The information in this section is presented as a Lab that will help the reader to understand step by step the integration process.

The reader should copy the content of the folder HFK\Src\misc\Labs\example_bt2\RF3_myHFK to a new working folder RF3_myHFK. Then, the folder RF3_myHFK\referenceframeworks\apps\rf3\hfk5407_bt2 should be copied in the same directory and the copy renamed hfk5407_bt3.

The folder RF3_myHFK\referenceframeworks\apps\rf3\hfk5407_bt3 will be the working folder for this example.

The information in this section is presented as a Lab that will help the reader to understand step by step the integration process.

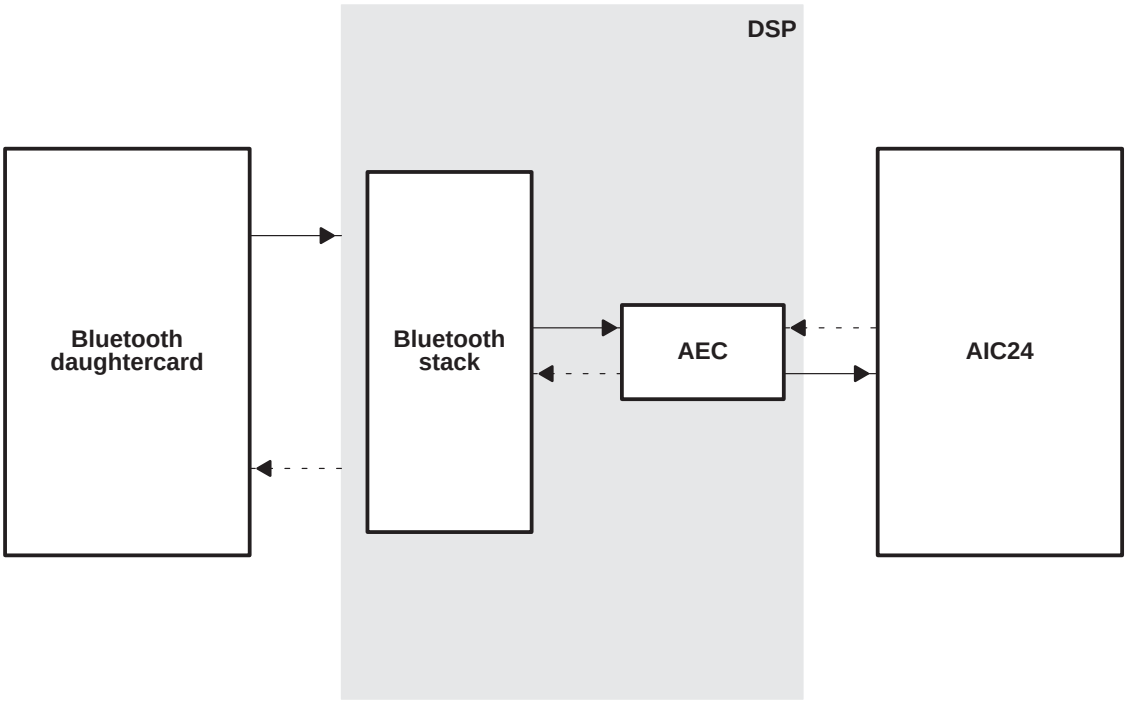


Figure 7. Block Diagram for LAB3, Example hfk5407_bt3

LAB 3: Integrating the BT stack with the AEC algorithm in RF3

Creating the AEC application folder

1. Create the folder RF3_myHFK\referenceframeworks\apps\rf3\algHFK.
2. Copy to this folder the files
   HFK\Src\AEC\xdais\ hfkv20100010005_clarity.h
   HFK\Src\AEC\xdais\ ihfk.c
   HFK\Src\AEC\xdais\ ihfk.h

Add the AEC libraries to the project

3. Copy the AEC libraries:

   HFK\Src\AEC\lib\ hfkv20100010005_clarity.l54

   HFK\Src\AEC\lib\ hfkv20100010005_clarity.l54f

   to the folder

   RF3_myHFK\referenceframeworks\lib

Modifying the audio processing thread

The following steps are identical to the ones described in *Hands-Free Kit Integration of the Clarity Acoustic Echo Cancellation (AEC) Algorithm in Reference Framework 3* (SPRA969). The reader is referred to this application note for samples of the modified code.

4. Open the file RF3_myHFK\referenceframeworks\apps\rf3\appModules\thrHfkAudio_bt2.c. Make a copy of this file in the same folder. Rename the copy thrHfkAudio_bt3.c
5. Open the file thrHfkAudio_bt3.c
6. Include the AEC header file hfkv20100010005_clarity.h

```
#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 ”hfkv20100010005_clarity.h”    /* AEC include file */

#include <hfk5407_drvGbl_bsync.h>   /* driver global varaibles */
#include <hfk5407_brf6100_bsync.h>  /* brf6100 driver info */
#include <hfk5407.h>                /* hfk5407  board info */
#include <hfk5407_aic24.h>          /* aic24 driver info */

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

/* Begin AEC Memory Declaration */

#pragma DATA_SECTION(hfkObject,”.HFK_MEM1”);
short                hfkObject[HFK_CLARITY_MEM1_size];
#pragma DATA_SECTION(cbuf_si,”.HFK_MEM2”);
short                cbuf_si[HFK_CLARITY_MEM2_size];
#pragma DATA_SECTION(cbuf_ri,”.HFK_MEM3”);
short                cbuf_ri[HFK_CLARITY_MEM3_size];
#pragma DATA_SECTION(cbuf_so,”.HFK_MEM4”);
short                cbuf_so[HFK_CLARITY_MEM4_size];
#pragma DATA_SECTION(buff0,”.HFK_MEM5”)
short                buff0[HFK_CLARITY_MEM5_size];
#pragma DATA_SECTION(buff1,”.HFK_MEM6”)                    // must be aligned on 256
short                buff1[HFK_CLARITY_MEM6_size];
#pragma DATA_SECTION(buff2,”.HFK_MEM7”)
short                buff2[HFK_CLARITY_MEM7_size];

/* AEC global variables */
IALG_Handle                      hfk_handle = (IALG_Handle) &hfkObject;
IALG_MemRec                      hfkMem[7];

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

7. Add the AEC global variables and the AEC related data memory sections using the #pragma DATA_SECTION directive

8. In the *thrHfkAudioInit()* function initialize the AEC XDAIS object and the bases of the memory sections that the algorithms requires from the framework.

```
/*
 *  ========= thrHfkAudioInit ========
 *  Initialization of data structures for the thread, called from
 *  appThreads.c:thrInit() at init time.
 */
Void thrHfkAudioInit( Void )
{
        hfkMem[0].base = hfk_handle;
        hfkMem[1].base = cbuf_si;
        hfkMem[2].base = cbuf_ri;
        hfkMem[3].base = cbuf_so;
        hfkMem[4].base = buff0;    //buffer for intermediate result
        hfkMem[5].base = buff1;    //buffer for intermediate result
        hfkMem[6].base = buff2;    //filter buffer

        /* initialize algorithm */
        HFK_CLARITY_initObj(hfk_handle, hfkMem, NULL,(IALG_Params *)&IHFK_PARAMS);

   HFK5407_AIC24_init();
   HFK5407_BRF6100_init();
   HFK5407_audioBuffer_init();

}
```

9. In the *thrHfkAudioRun()* function add the definition of the pointers that will be used with the AEC algorithm.

```
/*
 *  ========= thrHfkAudioRun ========
 *  The ”body” of the swiHfkAudio thread.
 *
 */
Void thrHfkAudioRun( Void )
{
    XDAS_Int16 *aicSrcBufL, *aicDstBufL;
    XDAS_Int16 *aicSrcBufR, *aicDstBufR;
    XDAS_Int16 *brf6100SrcBuf, *brf6100DstBuf;

        XDAS_Int16 i,off;

            /* AEC related local variables */
        XDAS_Int16 *inRef_Ptr, *inMic_Ptr, *outSnd_Ptr, *outSpkr_Ptr;
```

10. Initialize the AEC pointers

11. The incoming BT signal needs to be divided by two in order to be used as input for the AEC algorithm.

12. Call the processing function.

```
        off = bufActive*FRAME_SIZE_8K;
        brf6100SrcBuf = (XDAS_Int16*)&liBtData[off];
        brf6100DstBuf = (XDAS_Int16*)&loBtData[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
                    Initialize AEC Pointers
         outSpkr_Ptr = aicDstBufL; //OUT2  SPKR OUT
        outSnd_Ptr  = brf6100DstBuf; //  SEND OUT (processed signal)
        inMic_Ptr   = aicSrcBufL; //INP3  SEND IN
        inRef_Ptr   = brf6100SrcBuf;  //  REF IN
/*      Dividing the reference signal by 2 because the AEC algorithm requires
                    1: Rcv_in > Snd_In   2: Snd_in <= 8000 in case of BT
 */
        for(i=0; i< FRAME_SIZE_8K;i++)
        {
                    brf6100SrcBuf[i] >>= 1;
}
                    // Process Frame
        HFK_CLARITY_process(hfk_handle,inMic_Ptr,inRef_Ptr,outSnd_Ptr);
```

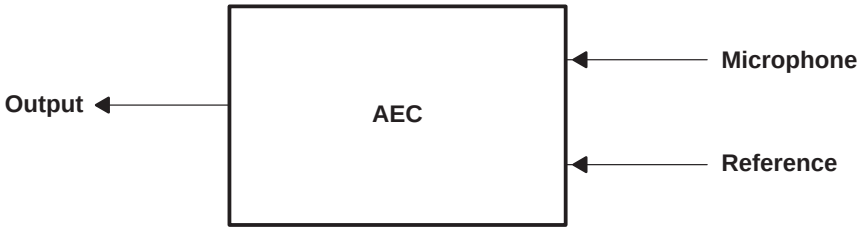


Figure 8. AEC Signals

The AEC algorithm processes two inputs, the Microphone (Mic) input and the Reference (Ref) input. In the context of an HFK system, the Mic signal is the signal provided by the microphone that is placed in the car whereas the Ref signal is the incoming phone call.

In the example described in Lab3 the Ref signal is the audio input from the on board EVM microphones and the Ref signal is the audio Bluetooth input.

13. Perform the data move.

```
       /*
        * 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++)
           {
           outSnd_Ptr[i]&=0xFFFE;   // sending out the processed signal
           outSpkr_Ptr[i]=brf6100SrcBuf[i]&0xFFFE;  //sending the BT input to speaker
           }
```

14. Save and close the file.
15. One of the on−board microphones provided on the HFK EVM is used as audio source in this example. These microphones require a 36 dB gain. The AIC24 configuration file aic24.h needs to be modified in order to reflect this gain.

    Open the file RF3_myHFK\referenceframeworks\include\aic24.h. Set #define BT3BT4CONFIG 1

16. A 36dB gain will be applied to the channel #1 signal.
17. Save and close the file aic24.h

Changing the linker command file

18. Open the linker command file link.cmd in the Code Composer Studio Project View window.
19. Add the AEC far mode library to the linker command file.

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

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

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

/* include the SS1 Bluetopia Library */
-l bluetopia54XX.l54f

/* include the SS1 Headset Profile Library*/
-l headset54XX.l54f

/* include the SS1 Hands-Free Profile Library*/
-l handsfree54XX.l54f

/* include the AEC library */
-l hfkv20100010005_clarity.l54f
```

20. Add the AEC memory sections in the linker command file.
21. Allocate the .text sections of the AEC algorithm and of the drivers related files in internal memory.
22. Allocate the .const section of the AEC algorithm in internal memory. The following memory sections are now defined in the linker command file.

23. Save and close the linker command file.

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

        .HFK_MEM1                                   : {} > IDATA PAGE 1 ALIGN (2)
        .HFK_MEM2                                   : {} > IDATA PAGE 1 ALIGN (128)
        .HFK_MEM3                                   : {} > IDATA PAGE 1 ALIGN (128)
        .HFK_MEM4                                   : {} > IDATA PAGE 1 ALIGN (128)
        .HFK_MEM5                                   : {} > IDATA PAGE 1
        .HFK_MEM6                                   : {} > IDATA PAGE 1 ALIGN (256)
        .HFK_MEM7                                   : {} > IDATA PAGE 1

        .text_internal:
                        {
              hfkv20100010005_clarity.l54f <hfk.o54> (.text)
              thrHfkAudio_bt3.o54f (.text)
              hfk5407_aic24_bsync.o54f (.text)
              hfk5407_drvGbl_bsync.o54f (.text)
              hfk5407_brf6100_bsync.o54f (.text)
              aic24.o54f (.text)
                    } > IPROG PAGE 0

        .const_internal:
                        {
               hfkv20100010005_clarity.l54f <hfk.o54> (.const)
                    } > IDATA PAGE 1

   .bss:     {}  > EDATA               PAGE 1
   .cio:     {} >  IDATA               PAGE 1
   .data:    {} >  IDATA               PAGE 1
   .const:   {} >  EDATA               PAGE 1
   .cinit:   {} >  EXTRAM1             PAGE 0
   .pinit:   {} >  EXTRAM1             PAGE 0
   .switch:  {} >> EXTRAM1 | EXTRAM2   PAGE 0
   .text:    {} >> EXTRAM1 | EXTRAM2 | EXTRAM3 | EXTRAM4 | EXTRAM5 | EXTRAM6 |
EXTRAM7     PAGE 0

}
```

Modifying the memory map

24. In the **System:MEM** folder open the IPROG memory section and modify the following properties.

    Base: 0x4000

    len: 0x3400

25. Select OK.

Changing the build options.

26. In the Code Composer Studio menu open *Project\Build Options*… Select the Preprocessor Category. In the *Include Search Path* window add the path for the AEC folder: ..\algHFK;

Preparing the project

27. Remove the file thrHfkAudio_bt2.c from the project.

28. Add the file thrHfkAudio_bt3.c to the project.

29. Add the file ihfk.c to the project.

30. Copy the file ihfk.h to the project include folder RF3_myHFK\referenceframeworks\include.

Executing the code

31. In the Code Composer Studio menu select *Project\Build* to build the project.
32. Load the Gel file provided in the HFK\Src\Gel folder
33. 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.
34. In the Code Composer Studio menu select *File\Load Program…* to load the code
35. Connect the speaker output to an amplified speaker.
36. In the Code Composer Studio menu select *Debug\Run* to run the code.
37. The following message should appear in the message log window:

```
0   Heap INTERNALHEAP: size = 768
1        used: 4 (0%)
2   Heap EXTERNALHEAP: size = 768
3        used: 0 (0%)
4   Application started.
5   Bluetooth Device Initialized successfully.
```

38. Pair the BT phone with the HFK EVM.
39. Dial the BT phone.
40. When the BT phone rings press Button One (the furthest button from the power supply connector) on the HFK EVM to accept the call.
41. The connection is established.
42. Speak in the on-board microphone.
43. The caller of the BT phone should not hear the echo.

Using the FM transmitter as output

In order to use the FM transmitter as output the aic24.h file needs to be modifed as follows:

− select in Channel #1 Control Register 6B, OUTP1

```
#define AIC24_REG6B_CH1_DEFAULT
    AIC24_8BITWORD(      /* CONTROL REGISTER 6B */                    \
        /* Reserved */ 1,   /* Do not change -- signifies 6B */       \
        /* Reserved */ 0,     /* Do not use */                        \
        /* ASTO3 */ 0,        /* Analog sidetone output for OUTP3 */ \
        /* ASTO2 */ 0,        /* Analog sidetone output for OUTP2 */ \
        /* Reserved */ 0,     /* Reserved */                          \
        /* OUTP1 */ 1,        /*OUTP1: DAC out to 600-Ohm line driver*/  \
        /* OUTP2 */ 0,          /*OUTP2: DAC out to 150-Ohm OUTP2 driver*/\
        /* OUTP3 */ 0          /*OUTP3: DAC out to 150-Ohm OUTP3 driver*/\
    )
```

− set in Channel #1 Control Register 5B the D/A Gain to −12db

```
#define AIC24_REG5B_CH1_DEFAULT                                       \
    AIC24_8BITWORD(        /* CONTROL REGISTER 5B */                 \
        /* D7 */ 0, 1,     /* Do NOT change -- signifies 5B */   \
        /* DAPGA */ 0, 0, 1, 0, 0, 0      /* D/A PGA Gain -12dB */  \
    )
```

The switch SW1 located on the HFK board (same side as the SRAM memory, under the speaker connector) needs to be in position on. The radio should be tuned around 90Mhz.

7 16kHz Sampling Frequency

In order to get a better frequency response it is possible to use a 16-kHz sampling frequency. However since the AEC algorithm operates at 8 kHz it is necessary to down sample the input signals and to up sample the output signal.

This section of the application report will provide the step by step process to accomplish that. Example hfk5407_bt3 described in the previous section will be used as a starting point. The resulting project of this tutorial is provided in the HFK software development kit in the folder Examples\referenceframeworks\apps\rf3\hfk5407_bt4.

LAB4: Using a 16kHz Sampling Frequency with the AEC algorithm.

NOTE: The following steps are similar to the ones described in *Hands Free Kit Integration of the Clarity Acoustic Echo Cancellation algorithm in the Reference Framework 3* (SPRA969). However, there are some differences related to the signals which are decimated and interpolated.

In this example only the Send In signal and the Speaker Out signal need to be transformed. Therefore only one decimation and one interpolation are required.

The reader should copy the content of the folder HFK\Src\misc\Labs\example_bt3\RF3_myHFK to a new working folder RF3_myHFK. Then, the folder RF3_myHFK\referenceframeworks\apps\rf3\hfk5407_bt3 should be copied in the same directory and the copy renamed hfk5407_bt4.

The folder RF3_myHFK\referenceframeworks\apps\rf3\hfk5407_bt4 will be the working folder for this example.

Adding the 16kHz specific drivers to the project

1. Copy from the SDK the folder

   HKF\Src\drivers\drv16kHz_bsync

   to the folder

   RF3_myHFK\referenceframeworks\src\drivers
2. Open in Code Composer Studio the project called app.pjt located in RF3_myHFK\referenceframeworks\apps\rf3\hfk5407_bt4\app.pjt
3. In the Code Composer Studio Project View window open the Source folder. Remove the files hfk5407_aic24_bsync.c, hfk5407_drvGbl_bsync.c, aic24.c from the project (right-click, select Remove).
4. Add the following driver files to the project:

   \drivers\drv16kHz_bsync\ hfk5407_aic24_bsync_16kHz.c

   \drivers\drv16kHz_bsync\ hfk5407_drvGbl_bsync_16kHz.c

   \drivers\drv16kHz_bsync\ aic24_16kHz.c
5. Copy the following header files to the include folder of the project:

   \drivers\include\ aic24_16kHz.h

   \drivers\include\ hfk5407_aic24_16kHz.h.

Adding the C54x DSP Library to the project

The decimation and interpolation filters will be added in the file RF3_myHFK\referenceframeworks\apps\rf3\appModules\thrHfkAudio.c. The interpolation and decimation functions used are the firdec() and firinterp() functions from the C54x DSP Library. This library is provided in the HFK SDK. It is located in HFK\Src\16kHz\dsplib.

6. Copy the file HFK\Src\16kHz\dsplib\54xdspf.lib to RF3_myHFK\referenceframeworks\lib.
7. Copy the file

   HFK\Src\16kHz\dsplib\include\TMS320.H

   to

   RF3_myHFK\referenceframeworks\include.
8. Include the 54xdspf.lib in the linker command file. Since the name of this library starts with a number, quotes are required to be used around the name.

```
/* include the C54x DSP Library */
-l   ”54xdspf.lib”
```

Adding the decimation and interpolation filters

The filter coefficients are provided in the HFK SDK in the file HFK\Src\16kHz\idFilters\id_filters.c

9. Copy the folder HFK\Src\16kHz\idFilters to the folder RF3_myHFK\referenceframeworks\src.
10. Add the file id_filters.c to the Code Composer Studio project
11. Copy the id_filters.h file to the project include folder RF3_myHFK\referenceframeworks\include

Modifying the audio processing thread

12. Make a copy of the file RF3_myHFK\referenceframeworks\apps\rf3\appModules\thrHfkAudio_bt3.c. in the same folder. Rename the copy thrHfkAudio_bt4.c
13. Open the file thrHfkAudio_bt4.c
14. include the file TMS320.h .
15. include the file id_filters.h
16. include the file hfk5407_aic24_16kHz.h instead of hfk5407_aic24.h

```
#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 ”hfkv20100010005_clarity.h”    /* AEC include file */

#include <hfk5407_drvGbl_bsync.h>   /* driver global varaibles */
#include <hfk5407_brf6100_bsync.h>  /* brf6100 driver info */
#include <hfk5407.h>                /* hfk5407  board info */
#include <hfk5407_aic24_16kHz.h>          /* aic24 driver info */


#include <TMS320.h>      /* DSPLIB types definition */
#include <id_filters.h>
```

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

17. Add the references to the filter buffers

```
/* Filter buffers */
extern DATA    dec_buf1[];
extern DATA    dec_buf2[];
extern DATA   interp_buf1[];
extern const DATA d_coeffs[];
extern const DATA i_coeffs[];

DATA *dp1 = dec_buf1;
DATA *dp2 = dec_buf2;
DATA *dp3 = interp_buf1;
```

18. In the *thrHfkAudioRun()* function add the definition of the local buffer pointers for the 8KHz buffers.

```
Void thrHfkAudioRun( Void )
{
      XDAS_Int16 *aicSrcBufL, *aicDstBufL;
      XDAS_Int16 *aicSrcBufR, *aicDstBufR;
      XDAS_Int16 i,off;

          /* AEC related local variables */
                   XDAS_Int16 *inRef_Ptr, *inMic_Ptr, *outSnd_Ptr, *outSpkr_Ptr;
        /* 8KHz buffer pointers used with 16KHz  related local variables */
          DATA *inMic_Ptr_8Khz, *inRef_Ptr_8Khz, *outSnd_Ptr_8Khz;
```

19. In the function *thrHfkAudioRun()* comment out the occurrence of *FRAME_SIZE_8K* related to the AIC24 buffers and replace it with *FRAME_SIZE_16K.* Do not modify the occurrence of *FRAME_SIZE_8K* related to the BT buffers because these buffers are unchanged. The modifications for the 16kHz sampling frequency affect only the AIC24 related buffers.

```
       off = bufActive*FRAME_SIZE_8K;
       brf6100SrcBuf = (XDAS_Int16*)&liBtData[off];
       brf6100DstBuf = (XDAS_Int16*)&loBtData[off];


//     off = bufActive*FRAME_SIZE_8K*2;
      off = bufActive*FRAME_SIZE_16K*2;

       aicSrcBufL               = (XDAS_Int16*)&liMicData[off];       //INP3 Tip
       aicDstBufL               = (XDAS_Int16*)&loSpkData[off];       //OUT2 Tip
//     off += FRAME_SIZE_8K;
   off += FRAME_SIZE_16K;
       aicSrcBufR               = (XDAS_Int16*)&liMicData[off];       // INP2 Ring
       aicDstBufR               = (XDAS_Int16*)&loSpkData[off];       // OUT3 Ring
```

20. Initialize the 8-kHz buffer pointers

21. The incoming BT signal needs to be divided by two in order to be used as input for the AEC algorithm.

22. Call the processing function.

```
       inMic_Ptr_8Khz = (DATA *)&inMic8KHz[0];
       inRef_Ptr_8Khz =  brf6100SrcBuf;
       outSnd_Ptr_8Khz = brf6100DstBuf;

                    // Initialize AEC Pointers
       outSpkr_Ptr = aicDstBufL;               //OUT2  SPKR OUT
//    outSnd_Ptr  = brf6100DstBuf;             //OUT2  SEND OUT (processed signal)
       inMic_Ptr   = aicSrcBufL;               //INP3  SEND IN
//     inRef_Ptr   = brf6100SrcBuf;            //INP2  REF IN
```

23. In the *thrHfkAudioRun()* function add the calls to the *firdec()* and *firinterp()* functions

```
/*      Dividing the reference signal by 2 because the AEC algorithm requires
                    1: Rcv_in > Snd_In   2: Snd_in <= 8000 in case of BT
 */
        for(i=0; i< FRAME_SIZE_8K;i++)
        {
                    brf6100SrcBuf[i] >>= 1;
        }

// 16kHz to 8kHz inMic buffer
firdec(inMic_Ptr,(DATA*)d_coeffs,inMic_Ptr_8Khz,&dp1,NUM_DI_COEFFS,FRAME_SIZE_16K,2
);

// Process Frame with 8Khz buffers
        HFK_CLARITY_process(hfk_handle,inMic_Ptr_8Khz,inRef_Ptr_8Khz,brf6100DstBuf);

// 8Hz to 16Hz outSpkr buffer
firinterp(brf6100SrcBuf,(DATA*)i_coeffs,outSpkr_Ptr,&dp3,NUM_DI_COEFFS,FRAME_SIZE_8
K,2);
```

24. Do not forget to modify the size of the data move loop.

```
/*
 * 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++) {
 for (i = 0; i < FRAME_SIZE_16K; i++)
 {
 outSpkr_Ptr[i]&=0xFFFE;   //sending the BT input to speaker
 }
```

25. Save and close the thrHfkAudio_bt4.c file.

26. Add the file thrHfkAudio_bt4.c to the Code Composer Studio project

27. Remove the file thrHfkAudio_bt3.c from the Code Composer Studio project.

Resetting the delay buffers

Each instance of the decimation or interpolation function requires a delay buffer. Double pointers are used to point to the delay buffers. In this example dp1,and, dp3 are the double pointers used to point to the delay buffers. The delay buffers must be cleared before the functions are invoked the first time.

Clear the delay buffers in the thrHfkAudioInit() function.

```
Void thrHfkAudioInit( Void )
{
        int i;

        hfkMem[0].base = hfk_handle;
        hfkMem[1].base = cbuf_si;
        hfkMem[2].base = cbuf_ri;
        hfkMem[3].base = cbuf_so;
        hfkMem[4].base = buff0;    //buffer for intermediate result
        hfkMem[5].base = buff1;    //buffer for intermediate result
        hfkMem[6].base = buff2;    //filter buffer

        /* clear the delay buffers */

        for (i=0; i<NUM_DI_COEFFS; i++) {

                    dec_buf1[i] = 0;
                    interp_buf1[i] = 0;
        }
```

Modifying the AIC24 configuration file

28. One of the on-board microphones provided on the HFK EVM is used as audio source in this example. These microphones require a 36 dB gain. The 16-kHz AIC24 configuration file aic24_16kHz.h needs to be modified in order to reflect this gain.

    Open the file RF3_myHFK\referenceframeworks\include\aic24_16kHz.h. Set #define BT3BT4CONFIG 1

29. A 36-dB gain will be applied to the channel #1 signal.

30. Save and close the file aic24_16kHz.h

Adding the filter-related memory sections in the linker command file

31. Open the linker command file and add the following filter related memory sections. The filter related memory sections require memory alignments. The reason for these alignments is that these buffers are used for circular addressing. For more information about the interpolation and decimation functions in the C54x DSPLIB the reader is referred to the *TMS320C54x DSP Library Programmer's Reference* (SPRU518). The *firdec()* and *firinterp()* functions are used in this project.

32. In the linker command file update the name of the HFK processing thread function. Replace thrHfkAudio_bt3.o54f with thrHfkAudio_bt4.o54f

33. In the linker command file update the .text_internal section with the names of the 16kHz drivers.

34. In the linker command file add to the .const_internal section hfk5407_aic24_bsync_16kHz.o54f (.const). The reason for this is to be able to fit the rest of the .const section in the EDATA memory section.

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

       .HFK_MEM1                 : {} > IDATA PAGE 1 ALIGN (2)
       .HFK_MEM2                 : {} > IDATA PAGE 1 ALIGN (128)
       .HFK_MEM3                 : {} > IDATA PAGE 1 ALIGN (128)
       .HFK_MEM4                 : {} > IDATA PAGE 1 ALIGN (128)
       .HFK_MEM5                 : {} > IDATA PAGE 1
       .HFK_MEM6                 : {} > IDATA PAGE 1 ALIGN (256)
       .HFK_MEM7                 : {} > IDATA PAGE 1


       /* Data Interp/Decim Memory */
       .interp_buf1              : {} > IDATA PAGE 1 ALIGN (64)
       .dec_buf1                 : {} > IDATA PAGE 1 ALIGN (64)
       .dec_buf2                 : {} > IDATA PAGE 1 ALIGN (64)
       .i_coeffs                 : {} > IDATA PAGE 1 ALIGN (64)
       .d_coeffs                 : {} > IDATA PAGE 1 ALIGN (64)


       .text_internal:
       {
       hfkv20100010005_clarity.l54f <hfk.o54> (.text)
       thrHfkAudio_bt4.o54f (.text)

       hfk5407_aic24_bsync_16kHz.o54f (.text)

       hfk5407_drvGbl_bsync_16kHz.o54f (.text)
       hfk5407_brf6100_bsync.o54f (.text)
       aic24_16kHz.o54f (.text)              } > IPROG PAGE 0 PAGE 0
```

```
.const_internal:
{

hfkv20100010005_clarity.l54f <hfk.o54> (.const)
hfk5407_aic24_bsync_16kHz.o54f (.const)
} > IDATA PAGE 1
```

- - - - - - -

35. Build the project and run the code. Follow the same steps provided in Lab 3 (steps 27−39).

Using the FM transmitter as output

In order to use the FM transmitter as output the aic24.h file needs to be modifed as follows:

− select in Channel #1 Control Register 6B, OUTP1

```
#define AIC24_REG6B_CH1_DEFAULT
    AIC24_8BITWORD(      /* CONTROL REGISTER 6B */                    \
        /* Reserved */ 1,   /* Do not change -- signifies 6B */      \
        /* Reserved */ 0,     /* Do not use */                       \
        /* ASTO3 */ 0,        /* Analog sidetone output for OUTP3 */ \
        /* ASTO2 */ 0,        /* Analog sidetone output for OUTP2 */ \
        /* Reserved */ 0,     /* Reserved */                         \
        /* OUTP1 */ 1,        /*OUTP1: DAC out to 600-Ohm line driver*/  \
        /* OUTP2 */ 0,          /*OUTP2: DAC out to 150-Ohm OUTP2 driver*/\
        /* OUTP3 */ 0          /*OUTP3: DAC out to 150-Ohm OUTP3 driver*/\
    )
```

− set in Channel #1 Control Register 5B the D/A Gain to −12db

```
#define AIC24_REG5B_CH1_DEFAULT                                      \
    AIC24_8BITWORD(        /* CONTROL REGISTER 5B */                \
        /* D7 */ 0, 1,     /* Do NOT change -- signifies 5B */   \
        /* DAPGA */ 0, 0, 1, 0, 0, 0      /* D/A PGA Gain -12dB */  \
    )
```

The switch SW1 located on the HFK board (same side as the SRAM memory, under the speaker connector) needs to be in position on. The radio should be tuned around 90Mhz.

8 Conclusion

This application report presented the integration of the BT stack with the AEC algorithm in RF3.

The buffer synchronization required by the two audio interfaces was described in detail to help the reader understand its implementation.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

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

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI 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 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. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

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

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

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

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