Hands-Free Kit: Integration of the Clarity Acoustic Echo Cancellation (AEC) Algorithm in RF3

Texas Instruments *HFK DP Software Applications*
Clarity Technologies

ABSTRACT

This application report describes the integration of 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 Clarity AEC algorithm in RF3. Second, the report describes the modifications required to operate the system with a 16kHz sampling frequency.

Contents

List of Figures

1 Introduction

The Hands-Free Kit (HFK) system presented in this application report was developed as an open system based on the DSP/BIOS RF3. 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 Clarity eXpressDSP-compliant AEC algorithm in RF3. The Example1 provided in the HFK SDK in the folder HFK\Src\Examples\referenceframeworks\apps\rf3\hfk5407_aic24 is used as a foundation for the integration of the AEC algorithm. The step-by-step process to build Example1 is described in the application report, *Hands-Free Kit: Integration of the AIC24 Driver in Reference Framework 3* (SPRA966).

The integration of the AEC algorithm is presented in a tutorial format to help you understand the process. The final result of the first section of this tutorial is provided in the folder \rf3\hfk5407_aec1 in the HFK SDK. This example will be called hfk5407_aec1 in the documentation. The second section of the tutorial describes the operation of the system with a 16kHz sampling frequency. The final result of the second section of this tutorial is provided in the folder \rf3\hfk5407_aec2 in the HFK SDK. This example will be called hfk5407_aec2 in the documentation.

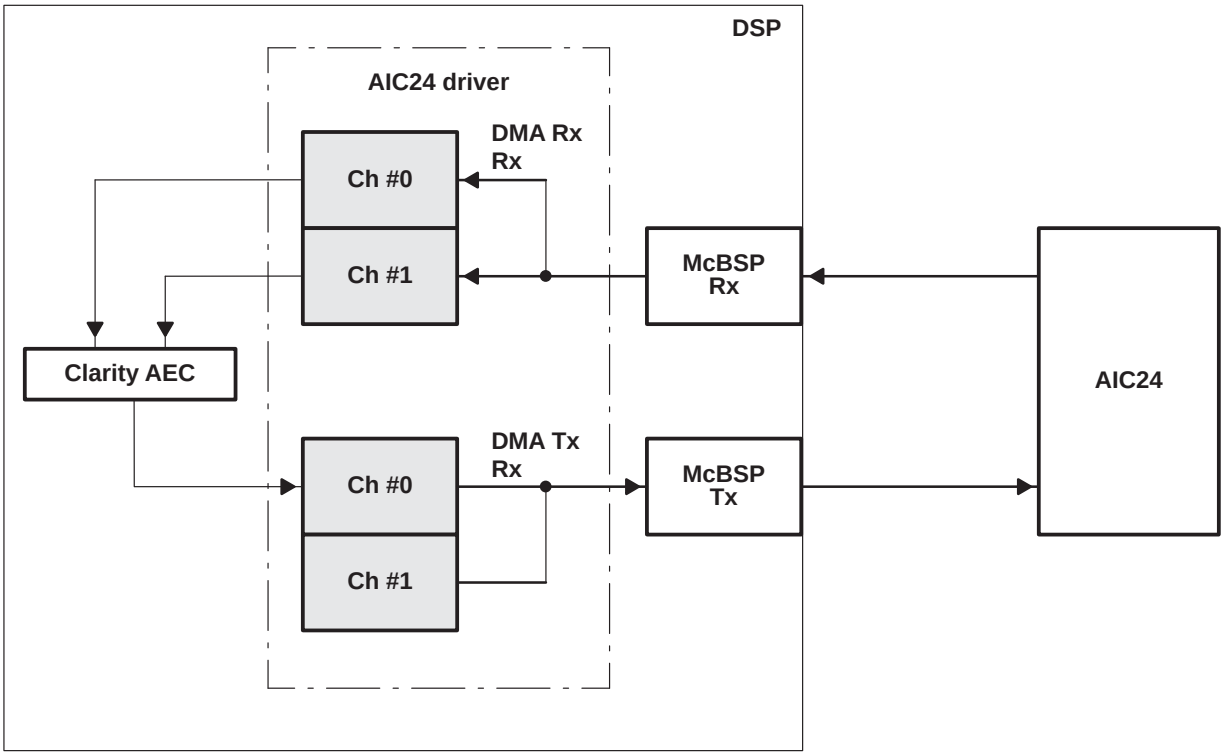


Figure 1. Block Diagram of the HFK AEC Example hfk5407_aec1

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 necessary steps to run the 8kHz acoustic echo cancellation example, hfk5407_aec1, provided with the HFK SDK.

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. 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. Play the audio samples received with the HFK SDK in the folder HFK\Src\AEC\audioSamples.
9. Calibrate the CD player as described in section 3.3 in *Clarity CVC-HFK-sys2.1.0 Software Integration, Testing, and Validation* (SPRA959).
10. Connect an amplified speaker to the speaker output of the HFK EVM.
11. Power-up the HFK EVM.

Running the acoustic echo cancellation example hfk5407_aec1

12. Start Code Composer Studio and open the project called app.pjt located in HFK\Src\Examples\referenceframeworks\apps\rf3\hfk5407_aec1.by using the Project → Open Menu command.
13. Load the GEL file provided in HFK\Src\GEL.
14. Load the executable app.out from .\Debug subdirectory with File → Load Program.
15. Start the audio source.
16. Run the code: Select Debug à Run from the Debug menu or press F5.
17. The audio signal will be heard on the amplified speaker. The audio signal heard should be similar to the signal provided on track2 in the audio samples folder.

Inspecting the project files

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

    Notice that in the ISR HFK5407_AIC24_DMA_isr(line 328), the swiHfkAudio SWI object is posted.

21. Set a breakpoint in the HFK5407_AIC24_DMA_isr() function and select Debug à Animate from the Debug menu, or press F12. The ISR will be called every time there are buffers available for processing
22. Remove the breakpoint and close the file hfk5407_aic24.c.
23. 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.
24. 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.
25. Close the file hfk5407_ drvGbl.c
26. From the View menu in the Code Composer Studio window, select View → Graph → Time/Frequency. A Graph Property Dialog window will open. Modify the following fields:

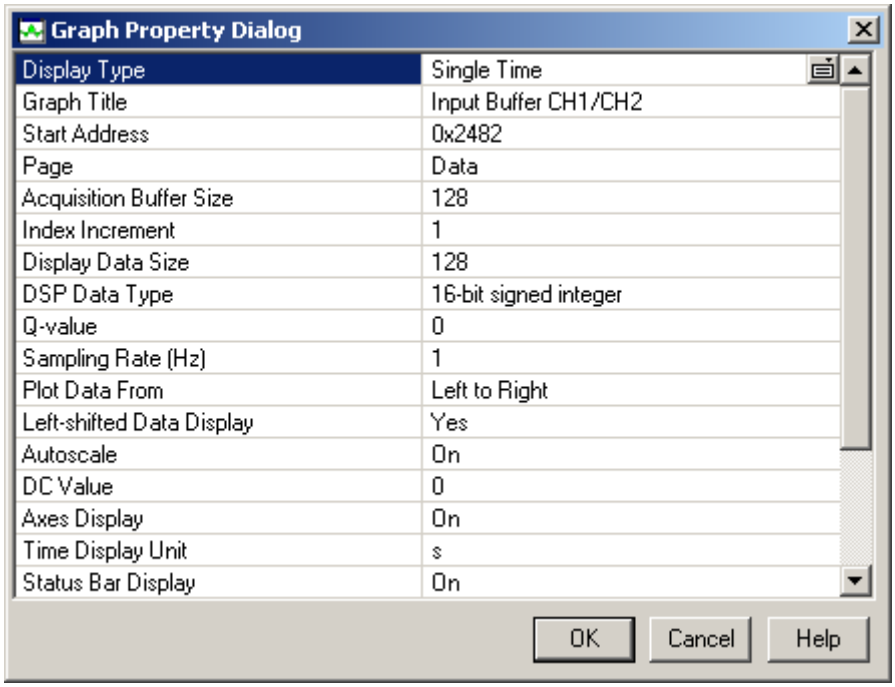
    Start Address: address of the liMicData[] buffer

    Acquisition Buffer Size: 128

    Display Data Size: 128

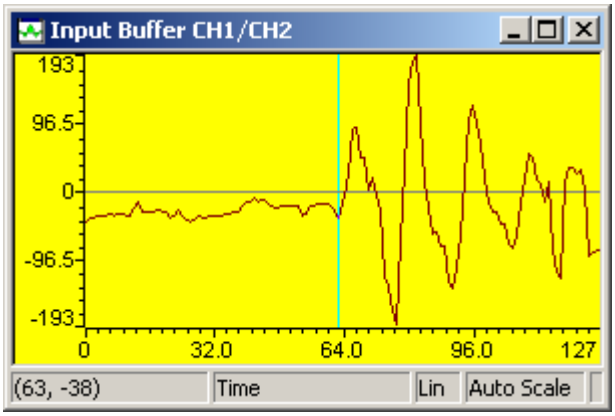
    DSP Data Type: 16-bit signed integer

    You should see something similar to:



27. Select OK.

28. 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. The screen shot above was taken at a point in time when there was no signal on Channel1.

29. Open the file thrHfkAudio_aec1.c. This file defines two functions: thrHfkAudioInit() (line 19) is used to initialize the Hfk Audio thread, and thrHfkAudioRun() (line 32) is the thread processing function associated with the swiHfkAudio SWI object.

30. thrHfkAudioRun() is called by the DSP/BIOS kernel after the swiHfkAudio SWI object is posted in the ISR. In the example hfk5407_aec1, this function calls the AEC algorithm processing function.

3 The Acoustic Echo Cancellation Algorithm

The AEC algorithm provided in the HFK software development kit is available in the folder HFK\Src\AEC. The algorithm includes:

- Far mode and near mode libraries

  HFK\Src\AEC\lib\hfkv20100010005_clarity.l54

  HFK\Src\AEC\lib\hfkv20100010005_clarity.l54f

- Documentation

  HFK\Src\AEC\doc

- XDAIS related files

  HFK\Src\AEC\hfkv20100010005_clarity.h

  HFK\Src\AEC\ihfk.c

  HFK\Src\AEC \ ihfk.h

4 Integrating the AEC Algorithm in RF3

This section describes how to integrate the AEC algorithm in RF3. The example built 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. This example is available in the folder, HFK\Src\Examples\referenceframeworks\apps\rf3\hfk5407_aic24.

The final project of the following LAB1 is available in the HFK software development kit in the folder, HFK\Src\Examples\referenceframeworks\apps\rf3\hfk5407_aec1.

Copy the content of the folder, HFK\Src\misc\Labs\example_aic24\RF3_myHFK, to a new working folder, RF3_myHFK. Then, copy the folder, RF3_myHFK\referenceframeworks\apps\rf3\hfk5407_aic24 in the same directory and rename the copy hfk5407_aec1.

The folder RF3_myHFK\referenceframeworks\apps\rf3\hfk5407_aec1 will be the working folder for this example.

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

LAB1: Integrating 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

Adding the AEC libraries to the project

3. Copy the files:
   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

4. 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_aec1.c
5. Open the file thrHfkAudio_aec1.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.h>   /* driver global varaibles */
#include <hfk5407.h>        /* hfk5407  board info */
#include <hfk5407_aic24.h>   /* aic24 driver info */
#include <xdas.h>    /* XDAIS types definition */
```

7. Add the AEC global variables and the AEC related data memory sections using the #pragma DATA_SECTION directive before the definition of the *thrHfkAudioInit()*function.

```
/* 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.
 */
```

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_audioBuffer_init();

}
```

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

```
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;
```

10. Initialize the AEC pointers and call the processing function.

```
off = bufActive*FRAME_SIZE_8K*2;

aicSrcBufL                  = &liMicData[off];     //INP3
aicDstBufL                  = &loSpkData[off];     //OUT2
off += FRAME_SIZE_8K;
aicSrcBufR                  = &liMicData[off];     // INP2
aicDstBufR                  = &loSpkData[off];     // OUT3

             // Initialize AEC Pointers

outSpkr_Ptr = aicDstBufL; //OUT3  SPKR OUT
outSnd_Ptr  = aicDstBufR; //OUT2  SEND OUT (processed signal
inMic_Ptr   = aicSrcBufR; //INP3  SEND IN
inRef_Ptr   = aicSrcBufL; //INP2  REF IN

             // Process Frame
HFK_CLARITY_process(hfk_handle,inMic_Ptr,inRef_Ptr,outSnd_Ptr);
```

11. Prepare the output buffers. Only the processed signal will be sent to out.

12. Save and close the file

```
/*
 * 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] = outSnd_Ptr[i] & 0xfffe;
   outSpkr_Ptr[i] = inRef_Ptr[i] & 0xfffe;
   }
```

Changing the linker command file

13. Open the linker command file link.cmd in the Code Composer Studio Project View window.

14. Add the AEC far mode library to the linker command file.

```
/* 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

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

15. Add the AEC memory sections in the linker command file.

16. 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

}
```

Changing the build options

17. 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

18. Remove the file thrHfkAudio_aic24.c from the project
19. Add the file thrHfkAudio_aec1.c to the project_
20. Add the file ihfk.c to the project
21. Copy the file ihfk.h to the include folder RF3_myHFK\referenceframeworks\include

Running the code

22. In the Code Composer Studio menu, select *Project\Build* to build the project.
23. Load the Gel file provided in the HFK\Src\Gel folder
24. 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.
25. In the Code Composer Studio menu select *File\Load Program*… to run the code
26. Connect a CD player to the HFK microphone input. With the CD player, play the audio samples provided with the HFK (they are located in the HFK\Src\AEC\ausioSamples folder). This CD contains real audio stereo samples recorded in a car.
27. Calibrate the CD player using the directions given in section 6 in the application report, *Clarity CVC-HFK-sys2.1.0 Software Integration, Testing, and Validation* (SPRA959).
28. Play track 1 of the CD. The output heard should be similar to the sample output provided on track 2 of the CD.
29. Connect the speaker output to an amplified speaker.
30. In the Code Composer Studio menu select *Debug\Run* to run the code.

5 Using a 16kHz Sampling Frequency

In order to get a better frequency response, it is possible to use a 16kHz sampling frequency for the AIC24 codec. However, since the AEC algorithm operates at 8kHz, 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_aec1 described in the previous section will be used as a starting point.

Copy the content of the folder HFK\Src\misc\Labs\example_aec1\RF3_myHFK to a new working folder RF3_myHFK. Then, the folder RF3_myHFK\referenceframeworks\apps\rf3\hfk5407_aec1 should be copied in the same directory and the copy renamed hfk5407_aec2.

The folder RF3_myHFK\referenceframeworks\apps\rf3\hfk5407_aec2 will be the working folder for this example.

LAB2: Using a 16kHz Sampling Frequency with the AEC algorithm

In order to use a 16kHz sampling frequency you must make the following changes:

- Modify the AIC24 configuration for a 16kHz sampling frequency.
- Increase the size of the DMA buffers to double size.
- Modify the DMA configuration accordingly.
- Add the decimation and interpolation filters.

The first three modifications affect the drivers. These changes have been included in the drivers available in the HFK SDK folder: HFK\Src\drivers\drv16kHz_nobsync.

The last modification in the list affects the HFK audio thread definition file.

1. Open the folder RF3_myHFK\referenceframeworks\src\drivers\drv16kHz_nobsync. If this folder is not available copy the folder HFK\Src\drivers\drv16kHz_nobsync to RF3_myHFK\referenceframeworks\src\drivers.

Inspecting the 16kHz configuration of the AIC24

2. The configuration of the AIC24 with 16kHz sampling frequency is defined in the file \drivers\include\ aic24_16kHz.h.
3. The sampling frequency of the AIC24 is defined in the AIC24 control register 4 by the divider values of M,N,P. These values will depend on the frequency of the external oscillator available on the HFK EVM.
4. For more information about the M, N, P divider values refer to the *TLV320AIC24 Data Manual* (SLAS366A), section 3.1.

Increasing the size of the DMA buffers

5. Since the value of the sampling frequency has been increased by a factor of two, the size of the DMA buffers needs to be adjusted accordingly (i.e., increased by a factor of two).
6. In the file \drivers\drv16kHz_nobsync\ hfk5407_drvGbl_16kHz.c the references to *FRAME_SIZE_8K* were commented out The value *FRAME_SIZE_16K* is used. The memory sections for the 16kHz buffers are used.

```
/* sections for 8KHz buffers used with 16KHz sampling */
#pragma DATA_SECTION(inMic8KHz,".HFK_AUDIO");
#pragma DATA_SECTION(inRef8KHz,".HFK_AUDIO");
#pragma DATA_SECTION(outSnd8KHz,".HFK_AUDIO");

// Codec Buffers - 16kHz
int liMicData[FRAME_SIZE_16K*4];
int loSpkData[FRAME_SIZE_16K*4];
int inMic8KHz[FRAME_SIZE_8K];
int inRef8KHz[FRAME_SIZE_8K];
int outSnd8KHz[FRAME_SIZE_8K];

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

7. The *HFK5407_audioBuffer_init()* function has been modified:

```
void HFK5407_audioBuffer_init(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++){
       for (i=0; i< FRAME_SIZE_16K*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];
}
```

Modifying the DMA configuration

8. Open the file \drivers\drv16kHz_nobsync\ hfk5407_aic24_16kHz.c.
9. In the functions *HFK5407_AIC24_init() and HFK5407_AIC24_DMA_isr(),* the value *FRAME_SIZE_16K* is used instead of *FRAME_SIZE_8K*.

Adding the C54x DSP Library to the project

The interpolation and decimation functions used are the *firdec()* and *firinterp*() functions from the C54x DSP Library. This library, provided in the HFK SDK, is located in HFK\Src\16kHz\dsplib.

10. Copy the file HFK\Src\16kHz\dsplib\54xdspf.lib to RF3_myHFK\referenceframeworks\lib.
11. Copy the file HFK\Src\16kHz\dsplib\include\TMS320.H to RF3_myHFK\referenceframeworks\include.
12. 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

13. Copy the folder HFK\Src\16kHz\idFilters to the folder RF3_myHFK\referenceframeworks\src.
14. Add the file id_filters.c to the Code Composer Studio project
15. Copy the id_filters.h file to the project include folder RF3_myHFK\referenceframeworks\include
16. In the Code Composer Studio menu open *Project\Build Options*… Select the Preprocessor Category. In the *Include Search Path* window add the path: ..\..\include;

    This path will enable the file id_filters.c to find the file TMS320.h that defines the DSPLIB data types.

Modifying the audio processing thread

17. Make a copy of the file RF3_myHFK\referenceframeworks\apps\rf3\appModules\thrHfkAudio_aec1.c in the same directory, rename it thrHfkAudio_aec2.c
18. Open the file RF3_myHFK\referenceframeworks\apps\rf3\appModules\thrHfkAudio_aec2.c
19. include the file TMS320.h.
20. include the file id_filters.h and the 16kHz versions of the driver headers

```
#include <hfk5407_drvGbl.h>              /* driver global varaibles */
#include <hfk5407.h>                     /* hfk5407  board info */
#include <hfk5407_aic24_16kHz.h>         /* aic24 driver info */

#include "hfkv20100010005_clarity.h"     /* AEC header file */

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

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

21. 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;
```

22. In the *thrHfkAudioRun()* function add the definition of the local buffer pointers for the 8-kHz 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;
```

23. In the function *thrHfkAudioRun()* comment out the occurrences of *FRAME_SIZE_8K* and replace them with *FRAME_SIZE_16K.*

```
//    off = bufActive*FRAME_SIZE_8K*2;
      off = bufActive*FRAME_SIZE_16K*2;
      // DMA method rotates outputs by one channel so swap Output Channels

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

24. Initialize the 8-kHz buffer pointers.

```
       inMic_Ptr_8Khz = (DATA *)&inMic8KHz[0];
       inRef_Ptr_8Khz = (DATA *)&inRef8KHz[0];
       outSnd_Ptr_8Khz = (DATA *)&outSnd8KHz[0];

                    // Initialize AEC Pointers
       outSpkr_Ptr = aicDstBufL; //OUT3  SPKR OUT
       outSnd_Ptr  = aicDstBufR; //OUT2  SEND OUT (processed signal)
       inMic_Ptr   = aicSrcBufR; //INP3  SEND IN
       inRef_Ptr   = aicSrcBufL; //INP2  REF IN
```

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

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

// 16kHz to 8kHz inRef buffer
firdec(inRef_Ptr,(DATA*)d_coeffs,inRef_Ptr_8Khz,&dp2,NUM_DI_COEFFS,FRAME_SIZE_16K,2
);

// Process Frame with 8Khz buffers
     HFK_CLARITY_process(hfk_handle,inMic_Ptr_8Khz,inRef_Ptr_8Khz,outSnd_Ptr_8Khz);

// 8Hz to 16Hz  outSnd buffer
firinterp(outSnd_Ptr_8Khz,(DATA*)i_coeffs,outSnd_Ptr,&dp3,NUM_DI_COEFFS,FRAME_SIZE_
8K,2);
```

26. Modify the size of the data move loop.

27. Save and close the file thrHfkAudio_aec2.c.

```
/*
 * 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++) {
   /*  */
   outSnd_Ptr[i] = outSnd_Ptr[i] & 0xfffe;
   outSpkr_Ptr[i] = inRef_Ptr[i] & 0xfffe;
   }
```

Preparing the project

28. Remove the file thrHfkAudio_aec1.c from the project.
29. Add the file thrHfkAudio_aec2.c to the project.
30. Remove the files aic24.c, hfk5407_aic24.c, hfk5407_drvGbl.c from the project
31. Add the files \drivers\drv16kHz_nobsync\aic24_16kHz.c;hfk5407_aic24_16kHz.c; hfk5407_drvGbl_16kHz.c to the project
32. Copy the files \drivers\include\aic24_16kHz.h;hfk5407_aic24_16kHz.h to the project include folder RF3_myHFK\referenceframeworks\include
33. Copy the file ihfk.h to the include folder RF3_myHFK\referenceframeworks\include

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,dp3, 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.

34. 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;
                    dec_buf2[i] = 0;
                    interp_buf1[i] = 0;
       }
```

Adding the filter related memory sections in the linker command file

35. Open the linker command file and add the following filter related memory sections. The filter related memory sections require memory alignments because the filter buffers are used as circular buffers. For more information about the interpolation and decimation functions in the C54x DSPLIB refer to the *TMS320C54x DSP Library Programmer's Reference* (SPRU518). The *firdec()* and *firinterp()* functions are the ones used in this project.

```
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)

}
```

36. Build the project and run the code following the steps provided in LAB 1 (steps 22−29)

6 Conclusion

This application report described the integration of the Clarity AEC algorithm in RF3. The integration was presented in a tutorial format in order to help you understand the process. The modifications required to use 16kHz signals were also presented. The result of this integration is available in the HFK SDK in the folders, Examples\referenceframeworks\apps\rf3\hfk5407_aec1 and Examples\referenceframeworks\apps\rf3\hfk5407_aec2.

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