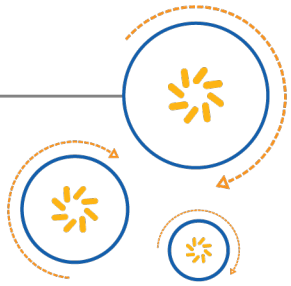
Qualcomm Technologies International, Ltd.



My Second DSP Application

Application Note

80-CT399-1 Rev. AP

October 23, 2017

Qualcomm Technologies International, Ltd. (formerly known as Cambridge Silicon Radio Limited) is a company registered in England and Wales with a registered office at: Churchill House, Cambridge Business Park, Cowley Road, Cambridge, CB4 0WZ, United Kingdom.
Registered Number: 3665875 | VAT number: GB787433096

Revision history

Revision	Date	Description
1	JUN 2007	Initial release. Alternative document number CS-00114287-AN.
2	AUG 2008	Cosmetic changes to update to latest style
3	AUG 2008	Optimized FIR filter library code
4	JUL 2010	Minor Editorial changes
5	JUL 2010	Updates to support the new SDK
6	FEB 2012	Major changes have been made on VM code to accommodate the changes in xIDE, ADK and hardware (CNS10001 Development board). Minor changes have been made on DSP code. Add step-by-step steps to build and run Qualcomm® Kalimba™ DSP on development boards. Rewrite the private DSP library build in Kalasm3. Update Appendices A ~ D. Remove the sections about ADC/DAC gain setting, operators, and FIR filter implementation. Updated document to latest style.
7	DEC 2012	References to specific ADK version updated to 2.5.
8	DEC 2013	Updated for ADK 3 and to new CSR™ style
9	JUL 2014	Updates for CSR8675
10	JUN 2015	Updates to DSP code.
11	JAN 2016	Updates for new FIR filter coefficients
12	JAN 2016	Added QTIL confidentiality statement
13	SEP 2016	Updated for ADK 4.1. Updated to conform to QTI standards.
AP	OCT 2017	Added to the Content Management System. DRN updated to use the Agile number. No change to technical content.

Contents

Figures

1 my_second_dsp_app

`my_second_dsp_app` is an example application, which uses the Kalimba DSP and the xIDE development environment. This application builds on the `my_first_dsp_app`, with the following changes:

- Added a FIR filter operator in left and right channels of the audio path
- Removed the DC remove operator and shift operator

Figure 1-1 provides a mark-up of the differences between the `my_first_dsp_app` and the `my_second_dsp_app`. The VM codes of both examples are the same, while the DSP code has been slightly modified.

The `my_second_dsp_app` describes how to generate the private DSP library that helps protect your intellectual property (IP).

This document does not describe how to generate the appropriate FIR filter coefficients for the required frequency response.

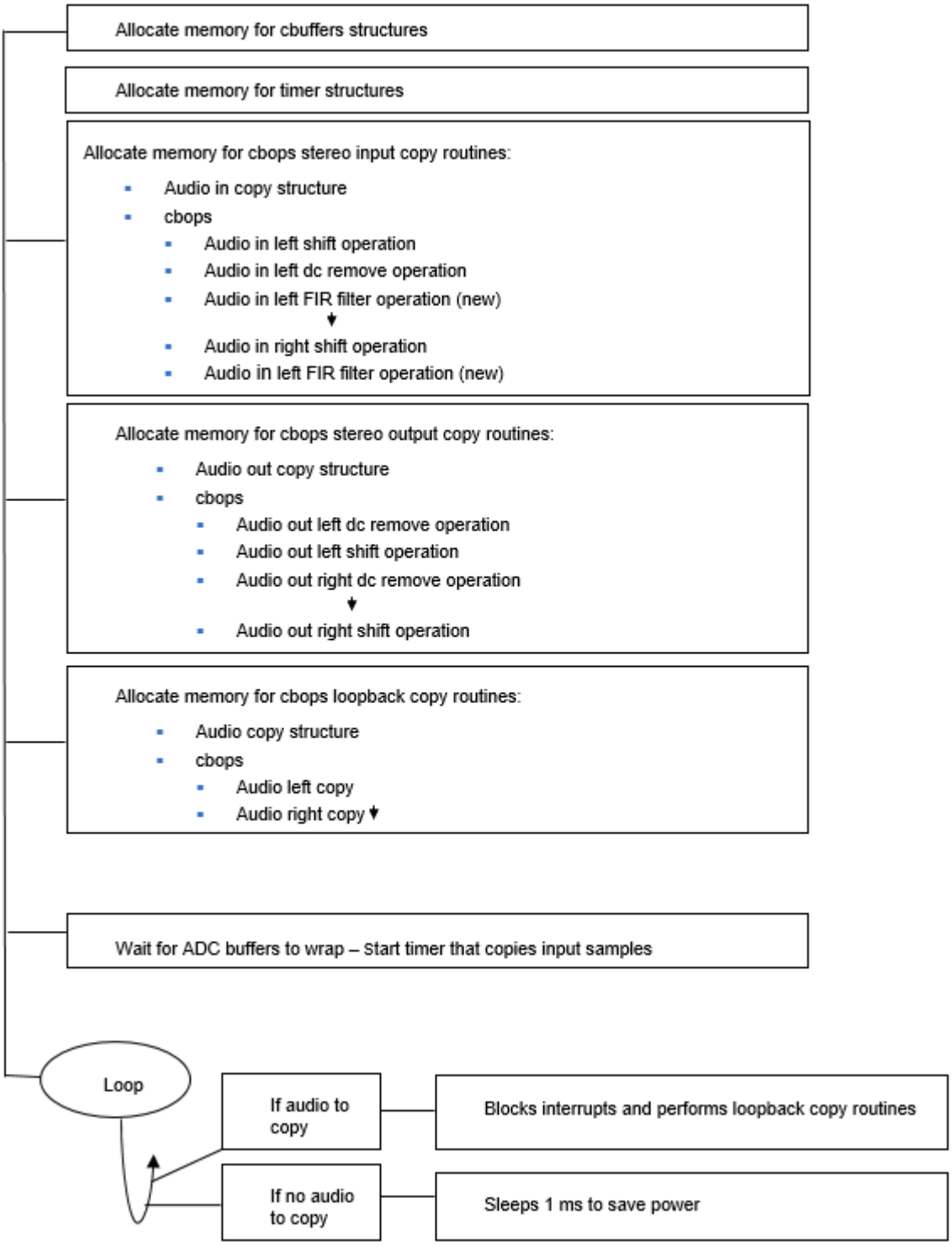


Figure 1-1 Changes between the _my_first_dsp_app and my_second_dsp_app

2 How to build and run a Kalimba DSP application on a CSR8670 board

Kalimba DSP applications are developed within the xIDE development environment ADK. Applications are structured as a workspace, which can be made up of a number of projects. When developing a DSP application the workspace should contain both a VM application and a DSP application. For instructions on how to create projects in xIDE, see the *BlueCore technology xIDE User Guide*.

Since the VM codes of both the `my_first_dsp_app` and the `my_second_dsp_app` are the same, the steps for these two example codes are also similar. The only difference is that `my_second_dsp_app` adds the private DSP library (FIR filtering code in this example code).

The instructions for building and running the Kalimba DSP application for CNS10001v4 development board (referred to as CSR8670 board in this document) and H13179v2 with H13374 module development board (referred to as CSR8675 board in this document) are slightly different.

1. Build the CSR8670 FIR filter private DSP library. At a Command Prompt, change the directory to <**ADK INSTALL>\apps\examples\my_second_dsp_app** or the directory where the FIR source code `fir_filter.asm` is located.
   Update the included batch file **buildPrivateLib_gdn.bat** with the installation folder of the ADK and use it to build the FIR filtering CSR8670 private DSP library. The subdirectory of */* **dsp_demo_app_2_fir_gdn**, contains the FIR CSR8670 scrambled private DSP library **fir_gdn.pa**. The batch file copies **fir_gdn.pa and fir_gdn.** > **link** to the xIDE CSR8670 library directory <**ADK INSTALL>\kalimba\lib_sets\sdk\gordon**. For more information on how to build the private DSP library in Kalasm3, see How to build and run a Kalimba DSP application on a CSR8675 development board.
2. Connect the CNS12002v1 interface daughter board to the CSR8670 board, as shown in Figure 2-1. Also connect an aerial and 3.7 V battery.

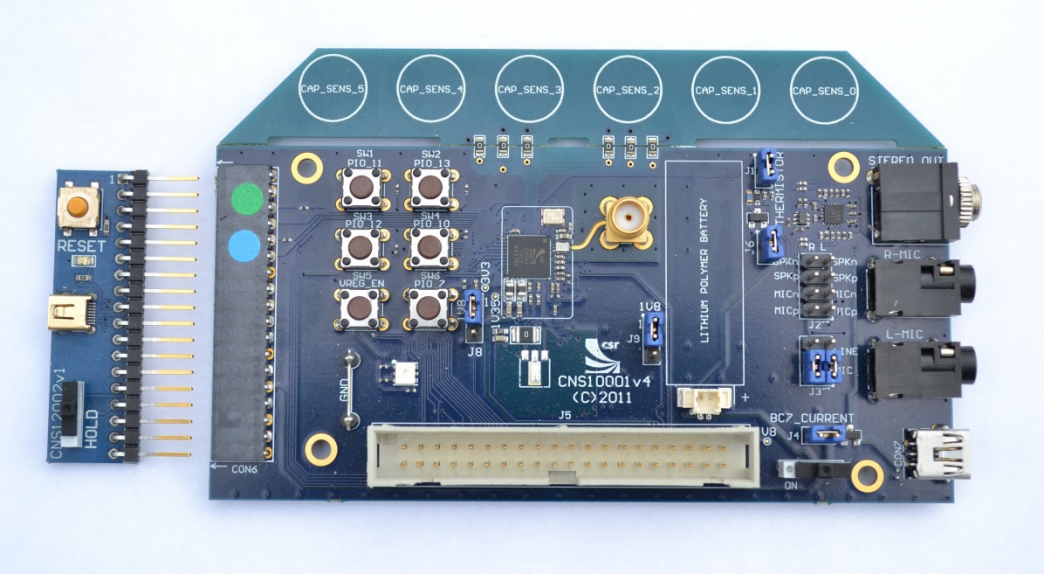


Figure 2-1 Connecting CNS12002v1 interface daughterboard to the CNS10001 (CSR8670) board

3. Connect the USB-SPI converter (CNS10020) to the CNS12002v1 interface daughter board using the supplied cable. Connect the other end of USB-SPI converter to the PC USB port using the USB cable, as shown below.

Figure 2-2 USB-SPI converter (CNS10020)

4. Connect the STEREO OUT jack to the speaker or headphone.
   For the single-ended stereo LINE input, connect the R-MIC and L-MIC jacks to the audio Line-out jack of PC or any portable media player through the Y-type audio cable. Set the **J3** jumper to **LINE**.
   For the 2-mic application, connect the R-MIC and L-MIC jacks to the stereo microphones through

the Y-type audio cable. For the 1-mic application, connect either R-MIC or L-MIC jack to the mono microphone. Set the **J3** jumper to **MIC**, as shown in Figure 2-3.

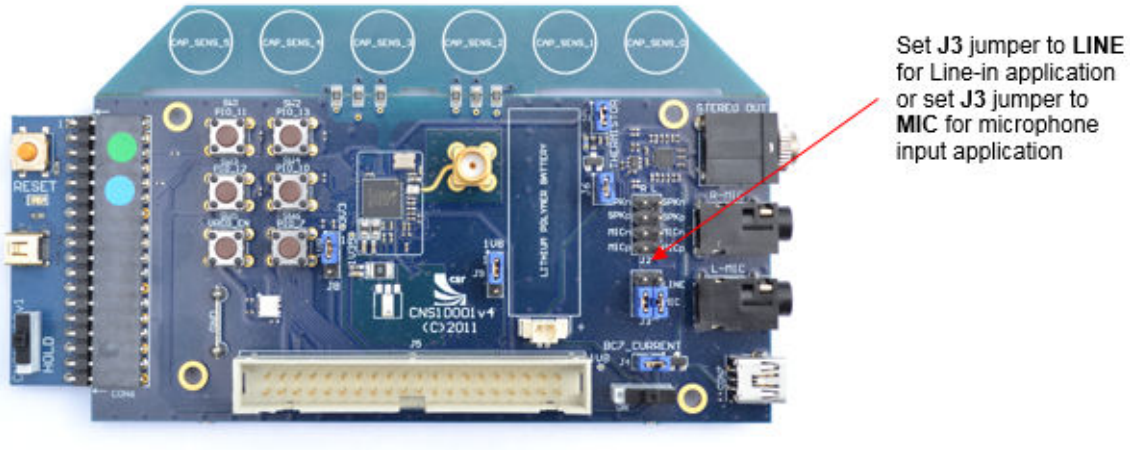


Figure 2-3 Setting J3 jumper

5. Connect the USB cable to the CSR8670 board to provide 5 V power supply.
6. Turn the Power switch **ON**.
7. Launch Audio Development Kit (ADK).
8. Locate the project workspace **my_second_dsp_app.xiw** by choosing P**roject | Open Workspace ...** within xIDE.

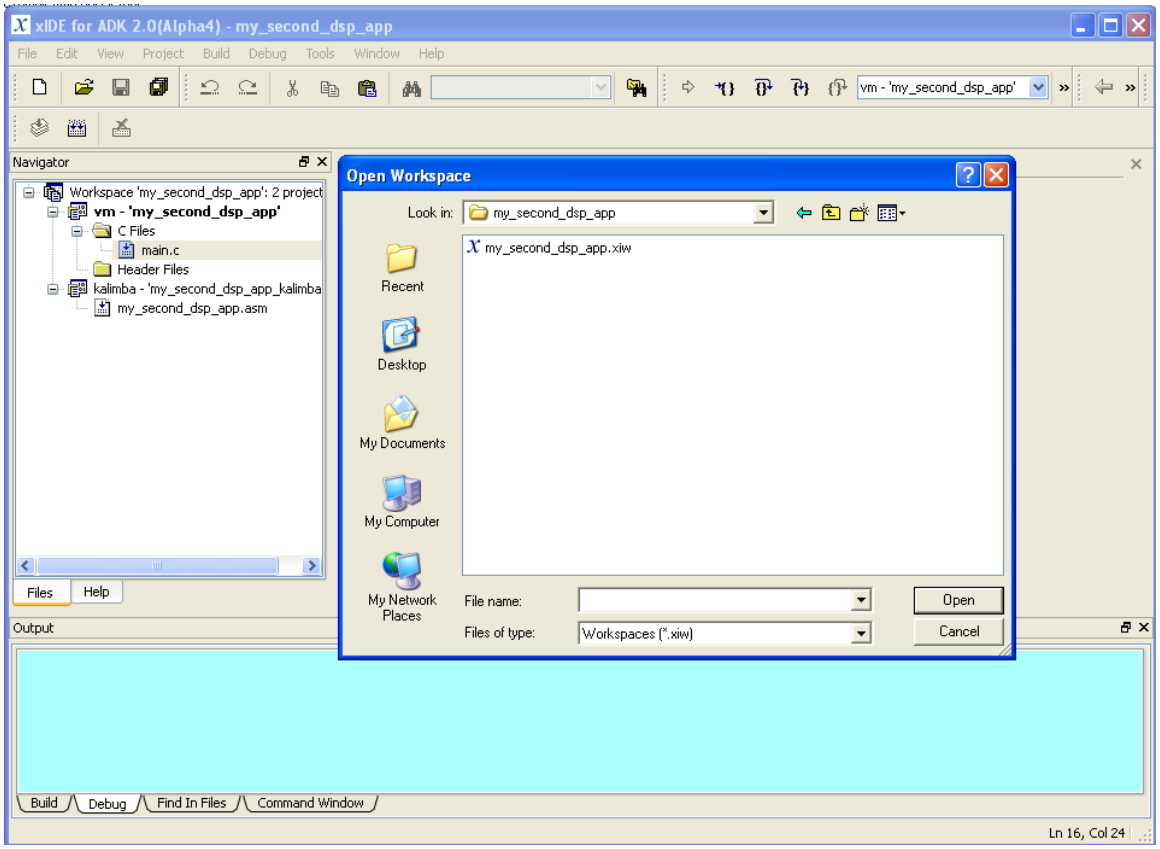


Figure 2-4 Open workspace

The project file **my_second_dsp_app.xiw** is located in:

<ADK INSTALL>\apps\examples\my_second_dsp_app

Your ADK is installed in **C:\ADK_INSTALL**. You can copy it to anywhere or create your own Kalimba DSP project. For instructions, see the *BlueCore technology xIDE User Guide*.

9. Check that the development board you are using is configured correctly in the project settings. To do this, right-click `vm – 'my_second_dsp_app` and select **Properties... | Build System,** as shown in Figure 2-5.

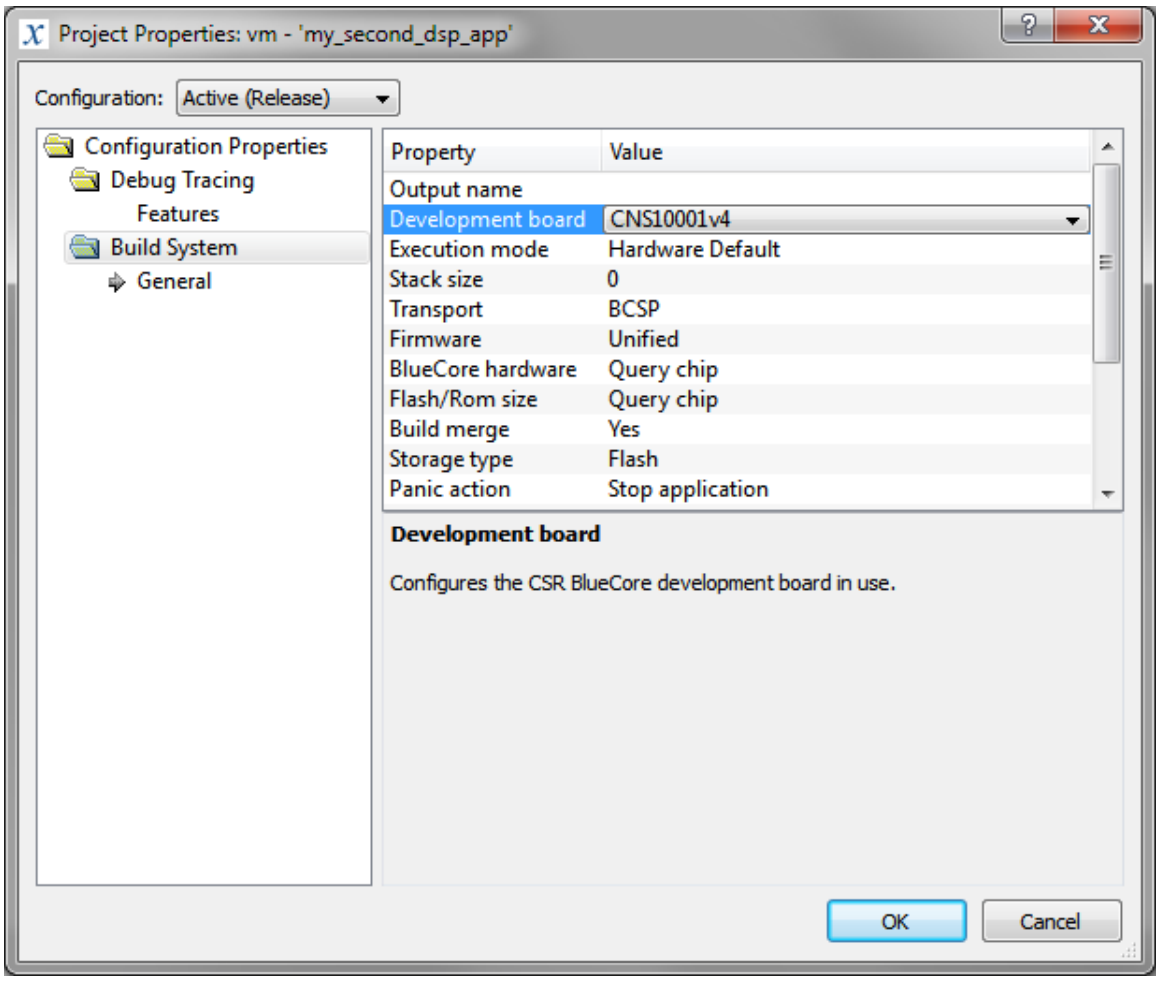


Figure 2-5 Project Properties: VM - my_second_dsp_app

10. Make sure the FIR library `fir_gdn` is included by right-clicking **kalimba – 'my_second_dsp_app_kalimba'** and then selecting **Properties.** To conditionally compile the FIR filtering code, include the symbol `FIR_FILTER` in **Extra defines**, see Figure 2-6.

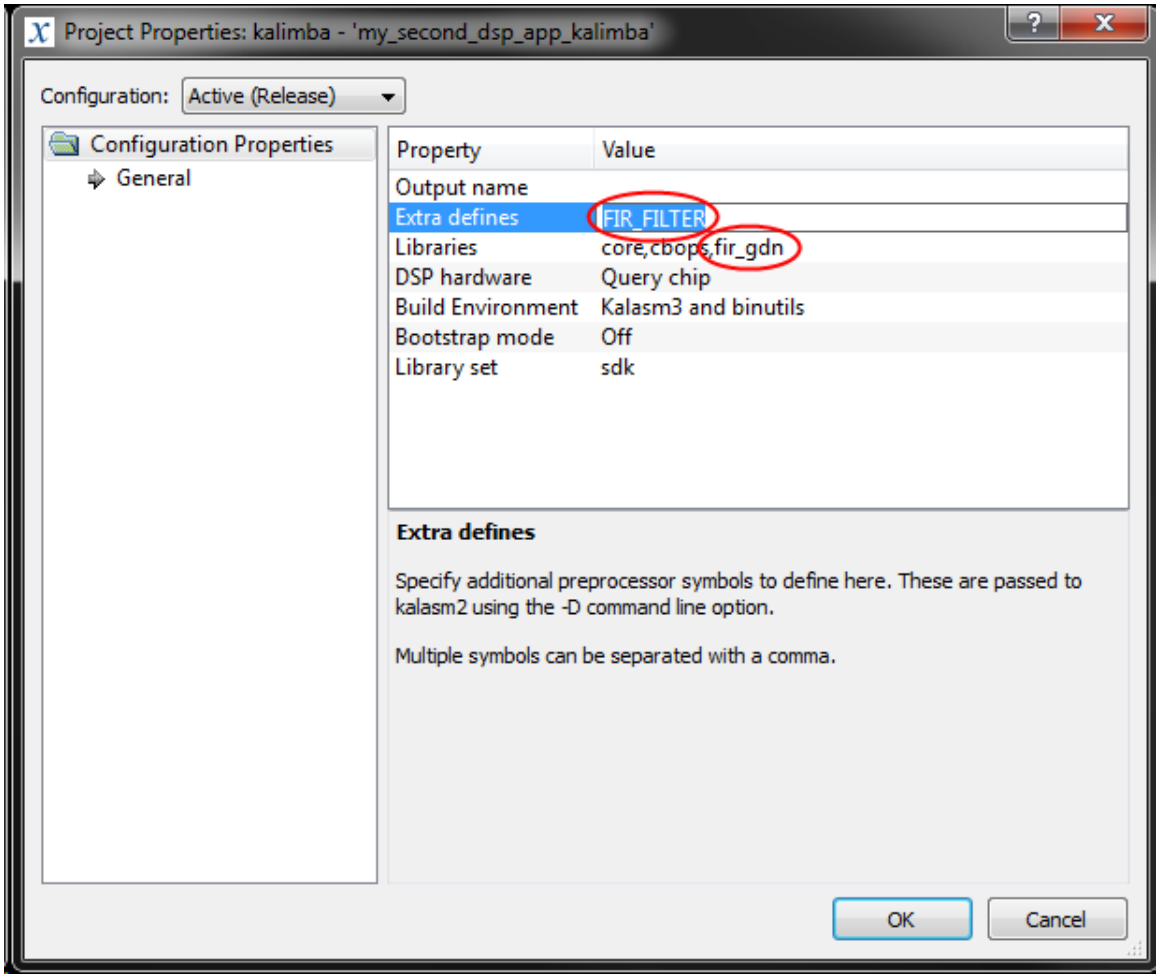


Figure 2-6 Kalimba - my_second_dsp_app_kalimba

11. Ensure that the correct Debug Transport is selected, the setting for this is under the **Debug | Transport...** menu in xIDE. Select the USB-SPI adapter connected to the development board, the serial number of the USB-SPI adapter is printed on the underside of the unit
Build the application by pressing the **F7** key, which compiles and links the code for both VM and DSP.

12. Alternatively, right-click **kalimba – 'my_second_dsp_app_kalimba'** and select **Build, t**hen right-click **vm – 'my_second_dsp_app'** and choose **Build**.
Run the Kalimba DSP application by pressing the **F5** key. This step flashes the executable binary image to the hardware and runs the application in debugging mode.

13. Alternatively use the xIDE to generate the executable binary image. To do this, press the **F7** key and use the flash tool **BlueFlash** to flash and run the application.

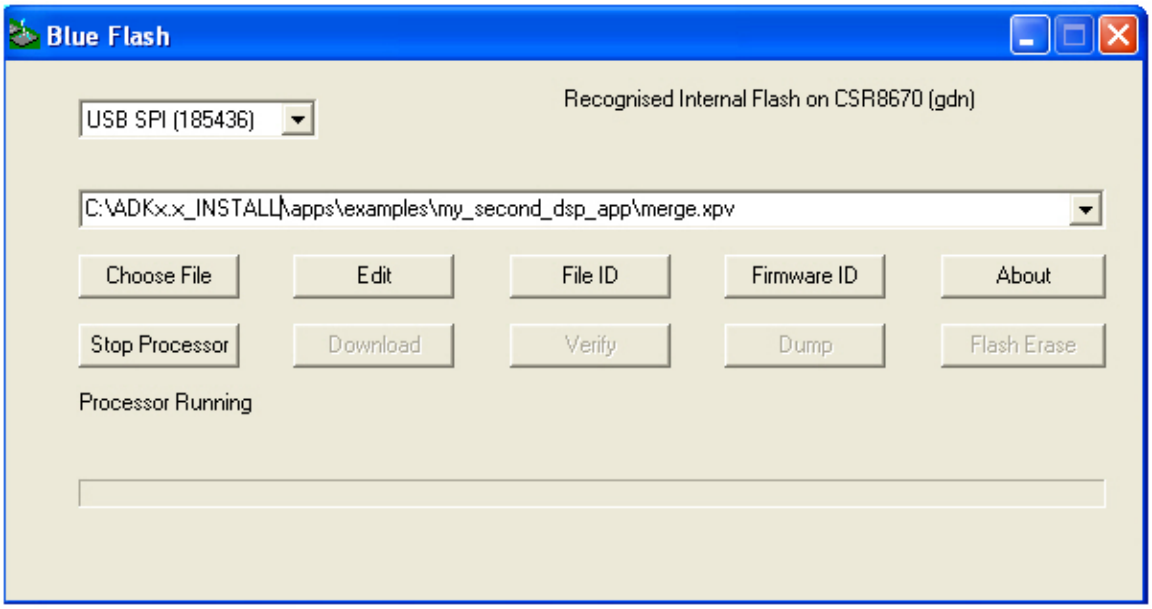


Figure 2-7 Blue Flash

For this alternative, you must set the **Build merge** to **Yes** in the VM **Project Properties** window. Otherwise, the executable image `merge.xpv (.xdv)` does not generate. QTIL recommends this approach because it is more efficient and productive.
After Step 14, the speaker or headphone should play the music. This example code routes the audio from the ADC to the DSP and then from the DSP to the DAC.

NOTE As an exercise you may bypass the DSP and route audio from the ADC to the DAC directly, i.e. without passing through the DSP. To do this, re-insert the macro definition on line 30 of main.c: `#define BYPASS_KALIMBA`. Alternatively, the symbol `BYPASS_KALIMBA` can be added to the **Project Properties** in **Define symbols** described in Step 11 (separate multiple symbols with commas). Steps 13 and 14 should then be repeated to rebuild the application.

To use a microphone input, re-insert the macro on Line 32 of main.c: `#define MIC_INPUT` and follow steps 1-14. Alternatively, this macro can be added via the **Project Properties** in **Define symbols**. You must also set Jumper **J3** to MIC, as described in Step 5.
This example FIR filter, filters the frequency contents of the input signal up to 3 kHz. If the FIR filtering is not added in the DSP audio path, you may remove the `FIR_FILTER` symbol in **Extra defines** as described in Step 11.

3 How to build and run a Kalimba DSP application on a CSR8675 development board

Kalimba DSP applications are developed within the xIDE development environment ADK. Applications are structured as a workspace, which can be made up of a number of projects. When developing a DSP application the workspace should contain both a VM application and a DSP application. For instructions on how to create projects in xIDE, see the *xIDE User Guide*.

Since the VM codes of both the `my_first_dsp_app` and the `my_second_dsp_app` are the same, the steps for these two example codes are also similar. The only difference is that `my_second_dsp_app` adds the private DSP library (FIR filtering code in this example code).

The instructions for building and running the Kalimba DSP application for CNS10001v4 development board (referred to as CSR8670 board in this document) and H13179v2 with H13374 module development board (referred to as CSR8675 board in this document) are slightly different.

1. Build the CSR8675 FIR filter private DSP library. At a Command Prompt, change the directory to **<ADK INSTALL>\apps\examples\my_second_dsp_app** or the directory where the FIR source code `fir_filter.asm` is located.
   Update the included batch file **buildPrivateLib_rck.bat** with the installation folder of the ADK and use it to build the FIR filtering CSR8675 private DSP library. The subdirectory of **/ dsp_demo_app_2_fir_rck**, contains the FIR CSR8675 scrambled private DSP library **fir_rck.pa**. The batch file copies **fir_rck.pa and fir_rck.** > **link** to the xIDE CSR8675 library directory <**ADK INSTALL>\kalimba\lib_sets\sdk\rick**. For more information on how to build the private DSP library in Kalasm3, see section 3.1.
   Connect the H13223 headphone amplifier to the H13179 board. Also connect an aerial and 3.7 V battery. SPI interface is built into the development board.

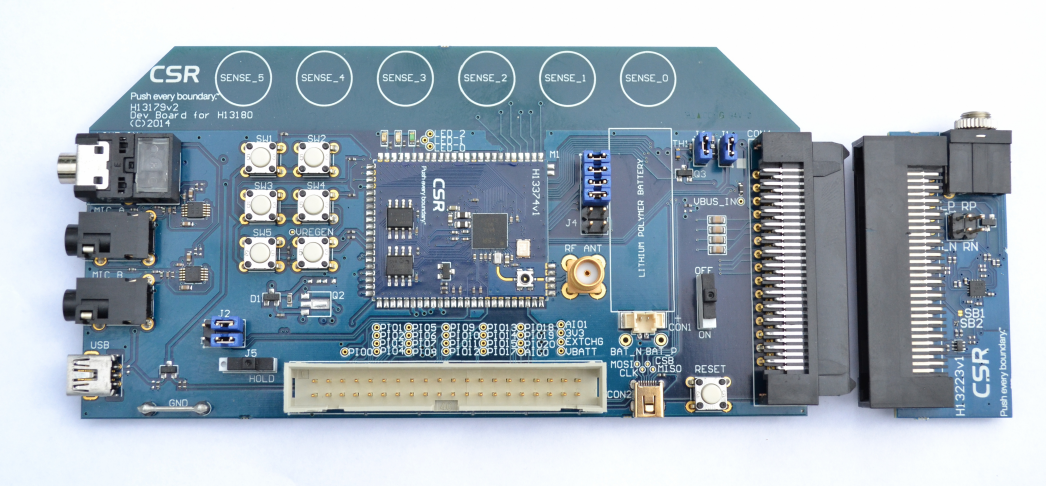


Figure 3-1 Connecting H13223 headphone amplifier board to the H13179/H13374 (CSR8675) board

2. Connect the USB-SPI converter (CNS10020) to CON2 on the H13179 board using the cable provided. Connect the other end of USB-SPI converter to the PC USB port using the USB cable. See Figure 2.2.
3. Connect the AUDIO OUT jack on the H13223 headphone amplifier to the headphones.
4. For the single-ended stereo LINE input, connect the LINE IN jack to the audio Line-out jack of a PC or any portable media player.
5. For the 2-mic application, connect the MIC A and MIC B jacks to the stereo microphones through a Y-type audio cable.
6. For the 1-mic application, connect either MIC A or MIC B jack to the mono microphone.
   a. Connect the USB cable to the board to provide 5 V power supply.
   b. Turn the Power switch **ON**.
7. Launch ADK.
8. Locate the project workspace **my_second_dsp_app.xiw** by choosing **Project | Open Workspace** ... within xIDE:

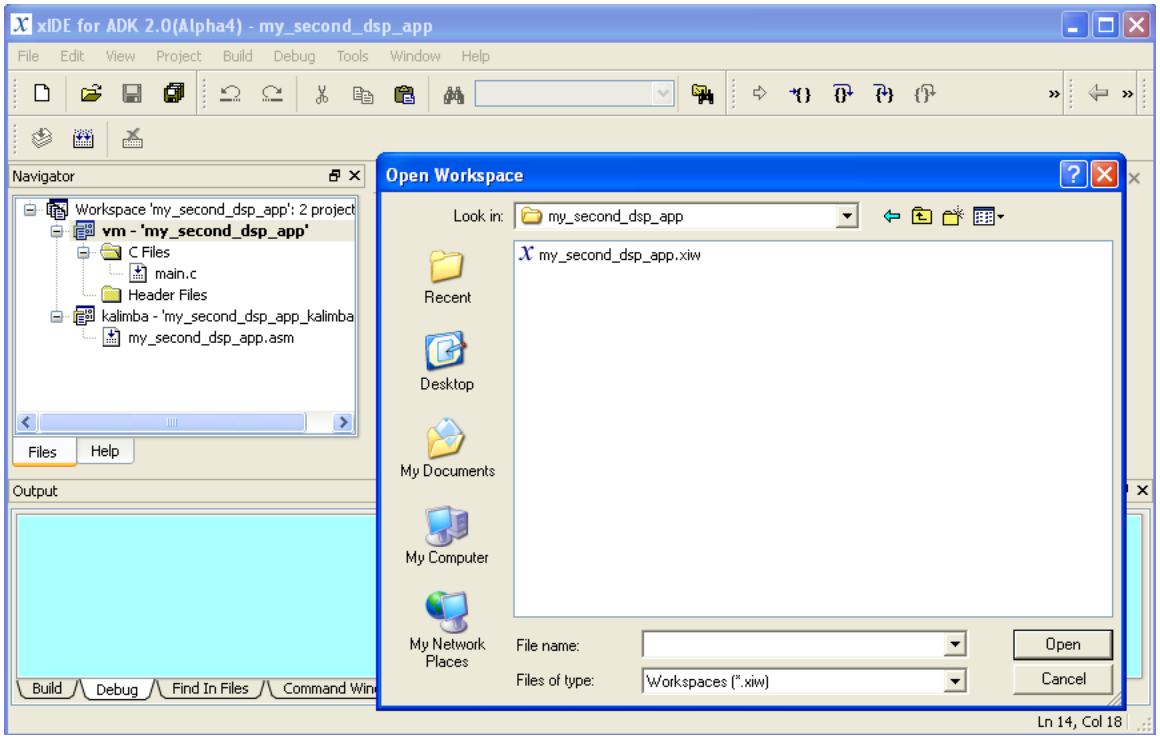


Figure 3-2 Open workspace

The project file **my_second_dsp_app.xiw** is located in:

```
C:\ADK_INSTALL\apps\examples\my_second_dsp_app
```

Your ADK is installed in **C:\ADK_INSTALL**. However, you can copy it to anywhere or create your own Kalimba DSP project. For instructions, see the *BlueCore technology xIDE User Guide*.

9. The default setting for the VM code in file `main.c` is for line-in input. To run it in other modes (such as microphone input or bypass DSP), comment out related lines or add the related symbols in **Define symbols**. To do this, right-click **vm – 'my_second_dsp_app'** and choose **Properties... | Build System** as shown in the figure in Step 10.
10. Make sure the FiR library `fir_rck` is included by right clicking **kalimba – 'my_second_dsp_app_kalimba**' and selecting **Properties.** To conditionally compile the FIR filtering code, include the symbol `FIR_FILTER` in **Extra defines**. See Figure 2.6.
11. Ensure that the correct Debug Transport is selected, the setting for this is under the **Debug | Transport...** menu in xIDE. Select the USB-SPI adapter connected to the development board, the serial number of the USB-SPI adapter is printed on the underside of the unit.
12. Build the application by pressing **F7** key, which compiles and links the code for both VM and DSP. Alternatively, right-click **kalimba – 'my_second_dsp_app_kalimba**' and select **Build, then** right-click **vm – 'my_second_dsp_app'** and select **Build**.
13. Run the Kalimba DSP application by hitting the **F5** key. This step flashes the executable binary image to the hardware and run the application in debugging mode.

Alternatively, use xIDE to generate the executable binary image. To do this, press the **F7** key and use the BlueFlash to flash and run the application.

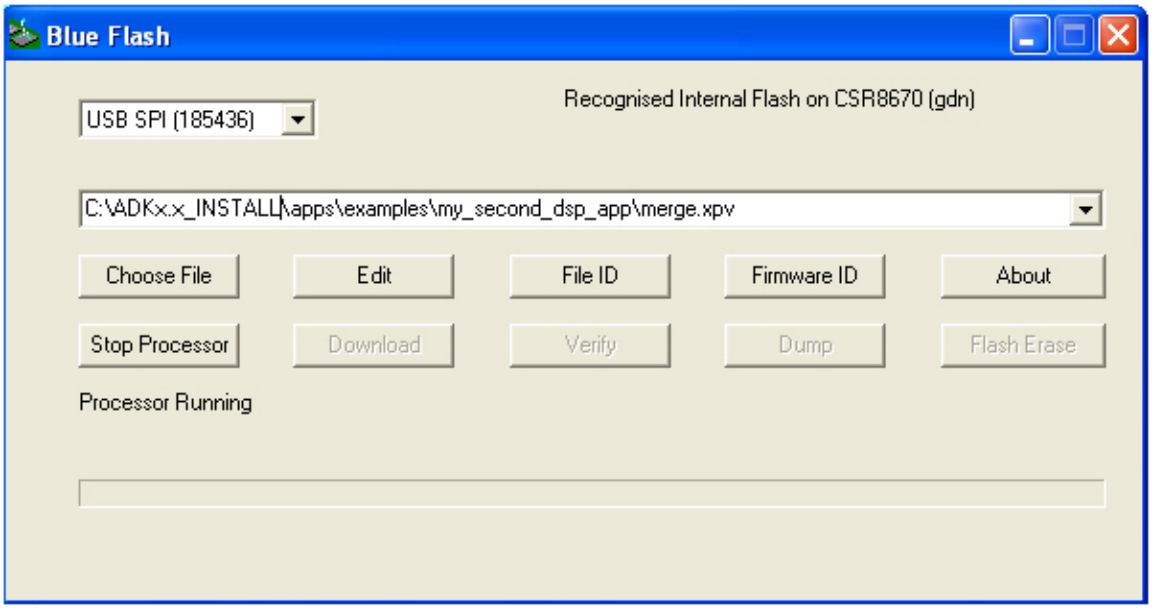


Figure 3-3 Blue Flash

14. For this alternative, you must set the **Build merge** to **Yes** in the VM **Project Properties** window. Otherwise, the executable image `merge.xpv` (`.xdv`) does not generate. QTIL recommends this approach because it is more efficient and productive.

After Step 14, the speaker or headphone can play the music. This example code routs the audio from the ADC to the DSP and then from the DSP to the DAC.

NOTE As an exercise you may bypass the DSP and route audio from the ADC to the DAC directly, i.e. without passing through the DSP. To do this, re-insert the macro definition on line 30 of main.c: `#define BYPASS_KALIMBA`. Alternatively, the symbol `BYPASS_KALIMBA` can be added to the **Project Properties** in **Define symbols** described in Step 11 (separate multiple symbols with commas). Steps 13 and 14 should then be repeated to rebuild the application.

To use a microphone input, re-insert the macro on Line 32 of main.c: `#define MIC_INPUT` and follow steps 1-14. Alternatively, this macro can be added via the **Project Properties** in **Define symbols**. You must also set Jumper **J3** to MIC, as described in Step 5.

This example FIR filter, filters the frequency contents of the input signal up to 3 kHz. If the FIR filtering is not added in the DSP audio path, remove the symbols `FIR_FILTER` in **Extra defines** as Step 11.

4 Understanding the VM and Kalimba application

The VM application used in the `my_second_dsp_app` is same as in the VM application used in `my_first_dsp_app`. The `my_second_dsp_app` only adds a FIR filtering in the left/right channels in DSP audio path. Refer to the *My First Kalimba DSP Application Note* for information about ADC/DAC gain setting and others.

This section describes how to build the private DSP library in Kalasm3, and how to hide and protect your intellectual property codes.

4.1 Creating private kalimba DSP libraries

To hide proprietary code from the `.kmap` file, create a private library. To create a private library, first the directives `PRIVATE` and `PUBLIC` should be added in the private DSP code.

1. Add the `.PRIVATE` directive to the appropriate module. Multiple modules can be declared `.PRIVATE`:

```
.MODULE $M.user_code.fir_filter.main;
        .PRIVATE;
        .CODESEGMENT PM;
        .DATASEGMENT DM;
        $user_code.fir_filter.main:
   // user defined code here
.ENDMODULE;
```

NOTE The `.PRIVATE` directive only works when creating libraries. Including the `.PRIVATE` directive in the source code generates errors when compiling in xIDE.

Declare any symbols in the private DSP code that will be accessible from outside private DSP code to be PUBLIC. For example, in the private code `fir_filter.asm`, the variable `user_code.fir_filter` will be called from the DSP application code `my_second_dsp_app.asm`, which is outside private DSP library code. Declare this variable symbol `user_code.fir_filter` to be PUBLIC in private DSP code `fir_filter.asm`:

```
.PUBLIC $user_code.fir_filter;
```

To build the private Kalimba DSP libraries in Kalasm3:

1. Compile the private `*.asm` files and generate the `*.o` object files:
   kas.exe *`.asm -o *.o -DBLD_PRIVATE -DKAL_ARCH3 -DGORDON -DKALASM3 -g -mchip=Ggordon -Iinclude_dir`
   The above command is for CSR8670. If the code is for CSR8675, change the above command to:
   `kas.exe *.asm -o *.o -DBLD_PRIVATE -DKAL_ARCH5 -DRICK -DKALASM3 -g -mchip=rick -Iinclude_dir`
2. Generate the *.a library file from the *.o object files:
   `kar.exe cr *.a *.o`
3. Scramble the `*.a` library file to generate the `*.pa` private library files:
   `kalscramble *.a -o *.pa`

Figure 4-1 shows the command line to build a private DSP library for CSR8670 in Kalasm3. The batch file `buildPrivateLib_gdn.bat` is written based on this.

```
<ADK Installation Path>/tools/bin/kas.exe fir_filter.asm  -o fir_gdn.o
-DBLD_PRIVATE -DGORDON -DKAL_ARCH3 -DKALASM3 -g -mchip=Gordon
-I<ADK Installation Path>/kalimba
-I<ADK Installation Path>/kalimba/architecture
-I<ADK Installation Path>/kalimba/lib_sets/sdk/include
-I<ADK Installation Path>/kalimba/external_includes/vm
-I<ADK Installation Path>/kalimba/external_includes/firmware

<ADK Installation Path>/tools/bin/kar.exe cr fir_gdn.a fir_gdn.o

<ADK Installation Path>/tools/bin/kalscramble.exe fir_gdn.a -o
fir_gdn.pa
```

Figure 4-1 Building a private DSP library in Kalasm3 from the command line

The command line shown in Figure 4-1generates CSR8670 versions of `fir_gdn.a` and `fir_gdn.pa` from `fir_filter.asm`. After the private DSP `.pa` library is built, it must be copied to the xIDE library directory. For example, for the CSR8670, the private library `fir_gdn.pa` must is copied to:

<ADK Installation Path>\\kalimba\lib_sets\sdk\gordon

When building the application DSP project in which the private DSP library will be called, you must add the private DSP library to the project, as shown in Figure 4-2.

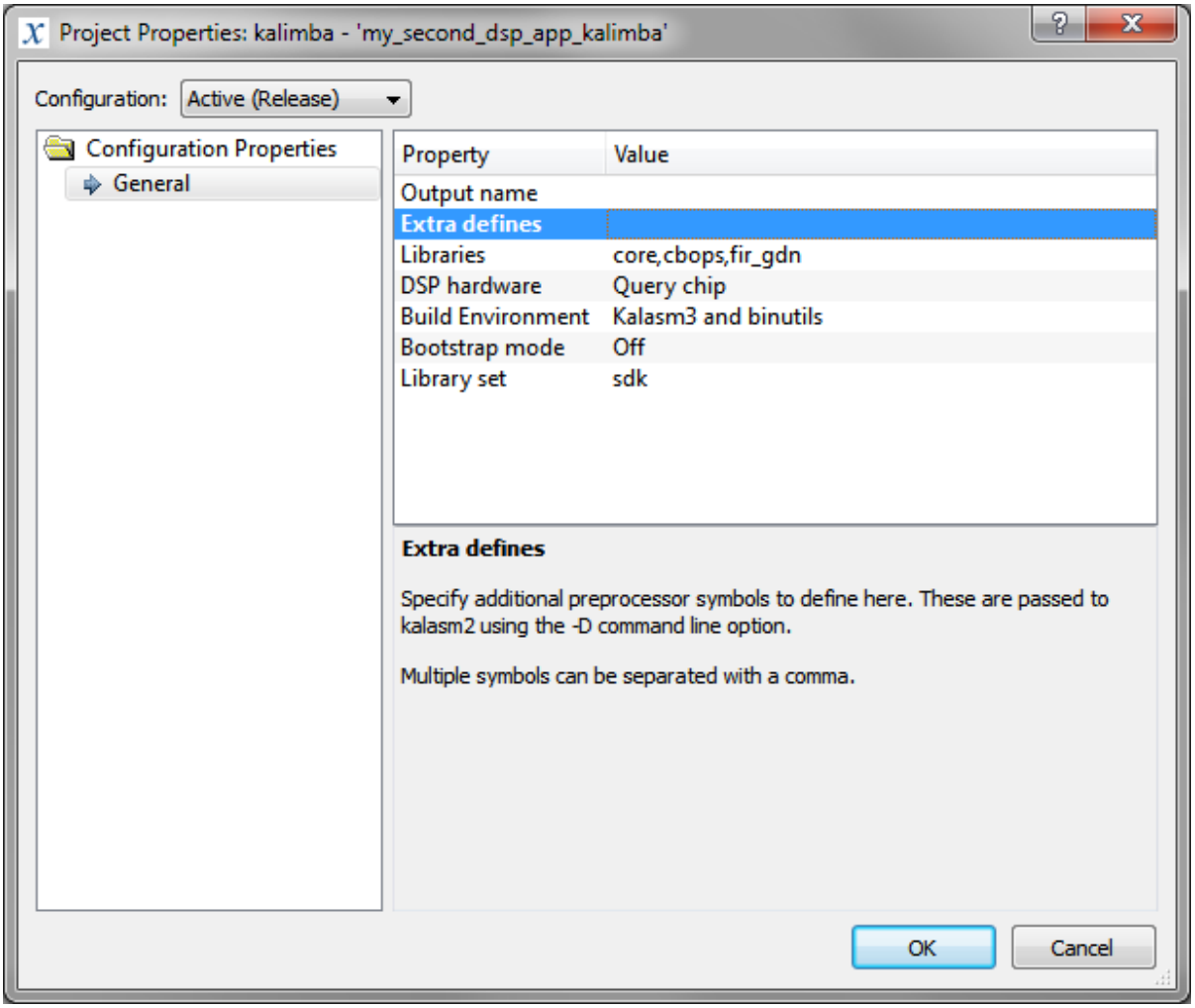


Figure 4-2 Project properties window - FIR filter library

Creating private kalimba DSP libraries shows the differences of the private code `fil_filter` in the `.kmap` file when using private and non-private libraries. The private library code `fir_filter` is not shown in the private library built `.kmap` file. The contents from the `0x02d4` to `0x02f9` in the

memory are not displayed. Unlike the public DSP library, the private DSP code is scrambled and difficult to disassemble.

Example of Non-Private Library Output

```
0x02D3      D002F0DD            jump $M.main.copy_loop;


 <$user_code.fir_filter.main>
0x02D4      0000D1F3            push rLink;
0x02D5      00002AD1            r0 = M[r8 + 0];
0x02D6      2F0038D0            r1 = M[r6 + r0];
0x02D7      3F000050            I0 = Null + r1;
0x02D8      2F0039D0            r1 = M[r7 + r0];
0x02D9      3F00C050            L0 = Null + r1;
0x02DA      01002AD1            r0 = M[r8 + 1];
0x02DB      2F0038D0            r1 = M[r6 + r0];
0x02DC      3F004050            I4 = Null + r1;
0x02DD      2F0039D0            r1 = M[r7 + r0];
0x02DE      3F00E050            L4 = Null + r1;
0x02DF      02002AD1            r0 = M[r8 + 2];
0x02E0      2F001050            I1 = Null + r0;
0x02E1      05005AD1            r3 = M[r8 + 5];
0x02E2      03002AD1            r0 = M[r8 + 3];
0x02E3      2F005050            I5 = Null + r0;
0x02E4      04002AD1            r0 = M[r8 + 4];
0x02E5      2F00F050            L5 = Null + r0;
0x02E6      2F00D050            L1 = Null + r0;
0x02E7      F202F0E5            do $M.user_code.fir_filter.main.loop;
0x02E8      00210003            Null = Null + Null, r0 = M[I0,1];
0x02E9      00002593            r0 = r0 ASHIFT r3;
0x02EA      A4000003            Null = Null + Null, M[I5,0] = r0;
0x02EB      FF007044            r5 = Null + L5;
0x02EC      00001001            rMAC = Null + 0;
```

Example of Private Library Output

```
0x02D3      D002F0DD    jump $M.main.copy_loop;
---


 <$loopback_copy>
0x02FA      0000D1F3    push rLink;
```

Figure 4-3 KMAP file output – difference between non-private and private library

4.2 Routing audio

The example application `my_second_dsp_app` routes stereo audio from the ADC to the DAC. The ADC and DAC are free running and controlled by hardware. Their buffers are managed by the MMU. The DSP has its own input and output circular connection buffers known as cbuffers.

The example Kalimba application has four parts as shown in Figure 4-4.

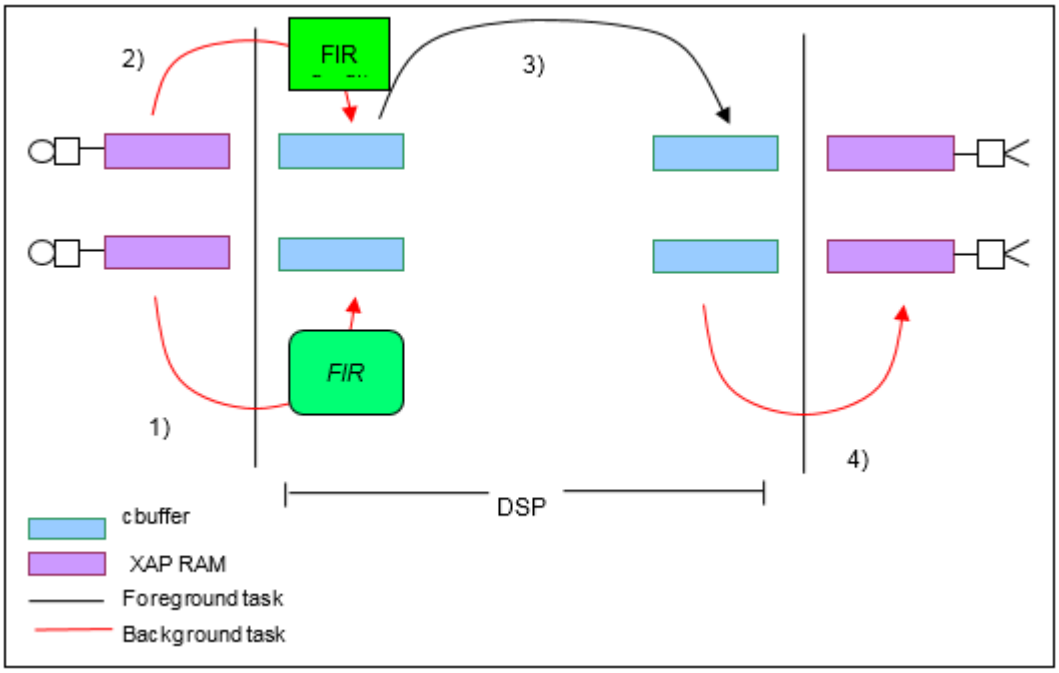


Figure 4-4 Audio buffering

1. Copy audio from the MMU to input `cbuffers`
2. Route audio through the FIR filter operator
3. Copy audio from the input `cbuffers` to the output `cbuffers`
4. Copy audio from the output `cbuffer` to the MMU

4.3 Application details

The input and output processes, labeled as Steps 1, 2 and 4 in Figure 4-4, are handled in interrupt routines in the background.

Each part of the application involves the copying of audio between buffers. Each step is a separate linked list of operations that copy and process the audio.

This framework, a linked list of operations (or cbops), is used to specify how to process the audio. The operators that appear in the list are configurable and are used to add custom processes by writing new operations as necessary.

Steps 1 and 2 in Figure 4-4 add a configurable `fir_filter` operator to the left/right input channel.

More operations can be carried out during this input copy, such as amplifying or attenuating the signal by a further shift or checks for overflow for example.

In the foreground, the main application loop calls a function to copy audio from the input `cbuffer` to the output `cbuffer` (Step 3 in Figure 4-4). It then sleeps for 1ms if there is time as a power-saving mechanism. During the copy routine call, the interrupts must be blocked to prevent memory corruption. As with the input and output steps, this copying is carried out as part of a linked list of operators.

NOTE There is the scope, as with the input and output steps, to insert processing tasks as required into the list.

The `fir_filter` used in this example is 111-tap high pass FIR filter and its frequency response is shown in Figure 4-5.

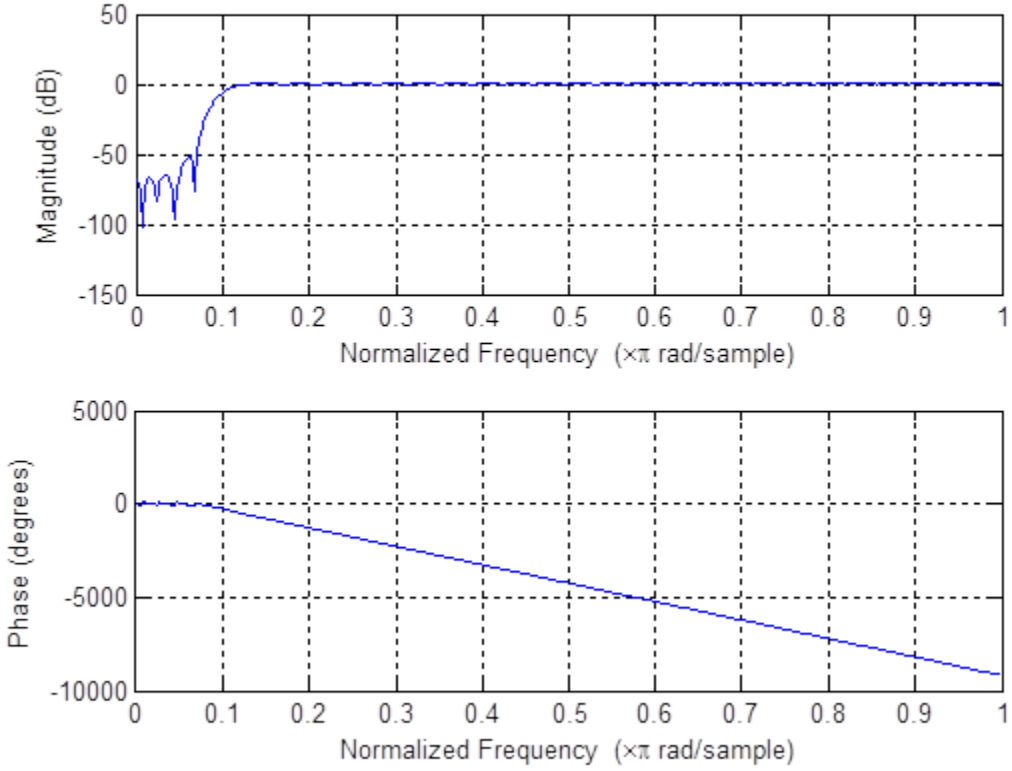


Figure 4-5 Frequency response of FIR high pass filter used in example

Figure 4-6 describes the structure of the Kalimba application, the code for which is contained in `my_second_dsp_app.asm` and `fir_filter.asm`.

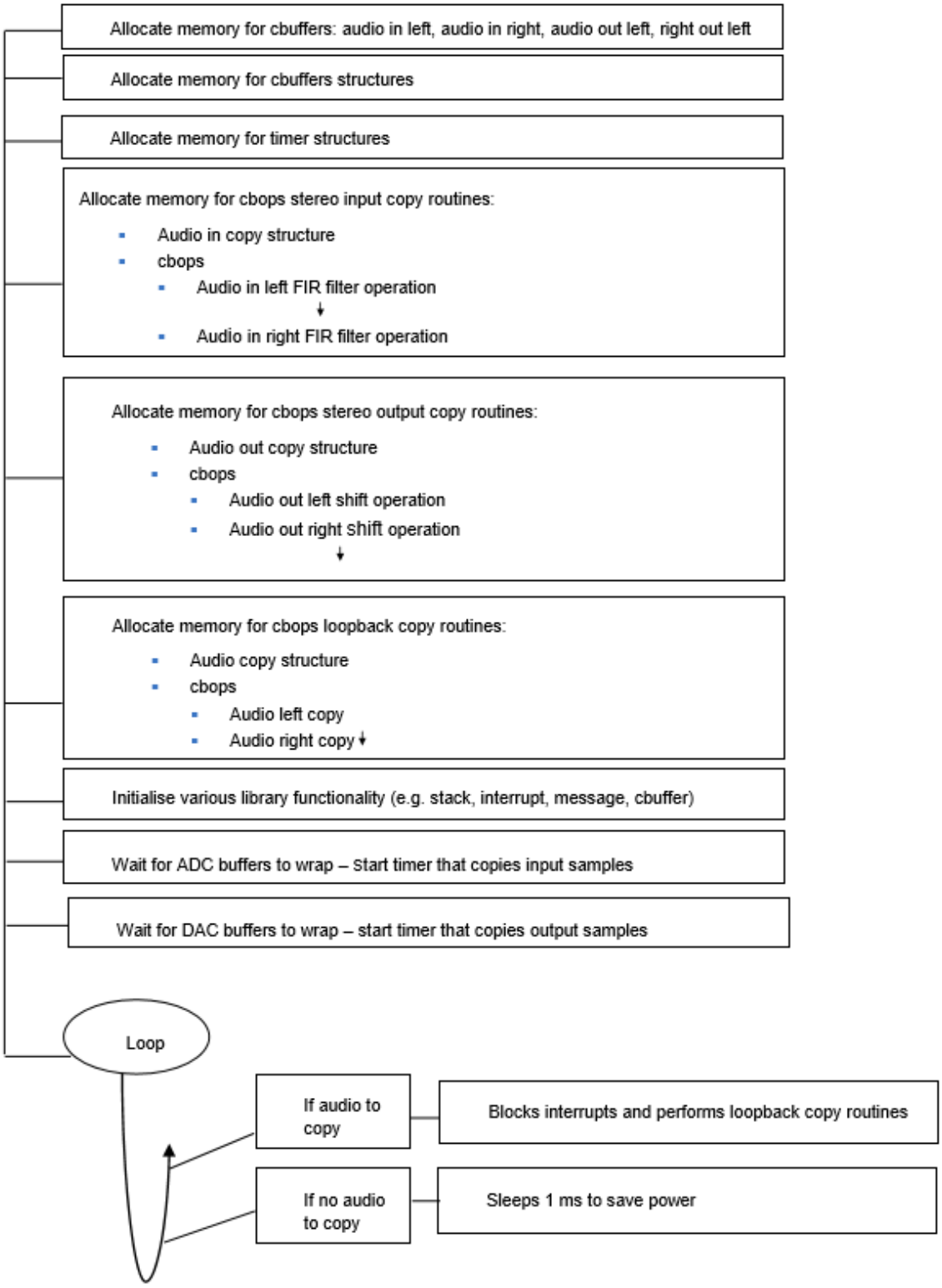


Figure 4-6 my_second_dsp_app

A my_second_dsp_app VM code

The following code is provided in:

<ADK Installation folder>\Examples\my_second_dsp_app

Do not cut and paste the following sample code for use.

```
/*
Copyright (c) 2010 - 2015 Qualcomm Technologies International, Ltd.
  An example app for routing audio through the Kalimba DSP from ADC to DAC
*/
#include <kalimba.h>
#include <kalimba_standard_messages.h>
#include <file.h>
#include <string.h>
#include <panic.h>
#include <source.h>
#include <sink.h>
#include <stream.h>
#include <connection.h>
#include <micbias.h>
#include <pio.h>
void PioSetPio (uint16 pPIO , bool pOnOrOff);
/* Select Amp PIO depending on board used.  If not defined, assume the
CNS10001v4 board is assumed. */
#ifdef H13179V2
    #define POWER_AMP_PIO 14
#else  /* Assume CNS10001v4 */
    #define POWER_AMP_PIO 4
#endif
/* Define the macro "BYPASS_KALIMBA" to bypass Kalimba DSP otherwise
direct ADC->DAC */
/* #define BYPASS_KALIMBA */
/* Define the macro "MIC_INPUT" for microphone input otherwise line-in
input */
/* #define MIC_INPUT */
/* Location of DSP kap file in the file system */
static const char kal[] = "my_second_dsp_app_kalimba/
my_second_dsp_app_kalimba.kap";
```

```
uint16 sampleRate = 48000;
void start_kalimba(void);
void connect_streams(void);
/* Main VM routine */
int main(void)
{
    /* Load the Kalimba */
    start_kalimba();
    /* Connect up the ADCs and DACS */
    connect_streams();
    /* Start the Kalimba */
    PanicFalse( KalimbaSendMessage(KALIMBA_MSG_GO,0,0,0,0) );
    /* Remain in MessageLoop (handles messages) */
    MessageLoop();
    return 0;
}
void start_kalimba(void)
{
    /* Find the codec file in the file system */
    FILE_INDEX index = FileFind( FILE_ROOT, (const char *)kal,
strlen(kal) );
    /* Did we find the desired file? */
    PanicFalse( index != FILE_NONE );
    /* Load the codec into Kalimba */
    PanicFalse( KalimbaLoad( index ) );
}
void connect_streams(void)
{
    /* Access left and right ADC and DAC */
    Source audio_source_a = StreamAudioSource( AUDIO_HARDWARE_CODEC,
AUDIO_INSTANCE_0, AUDIO_CHANNEL_A );
    Source audio_source_b = StreamAudioSource( AUDIO_HARDWARE_CODEC,
AUDIO_INSTANCE_0, AUDIO_CHANNEL_B );
    Sink audio_sink_a = StreamAudioSink( AUDIO_HARDWARE_CODEC,
AUDIO_INSTANCE_0, AUDIO_CHANNEL_A );
    Sink audio_sink_b = StreamAudioSink( AUDIO_HARDWARE_CODEC,
AUDIO_INSTANCE_0, AUDIO_CHANNEL_B );
    /* Configure sampling rate for both channels and synchronise left and
right channels */
    PanicFalse( SourceConfigure(audio_source_a, STREAM_CODEC_INPUT_RATE,
sampleRate) );
    PanicFalse( SourceConfigure(audio_source_b, STREAM_CODEC_INPUT_RATE,
sampleRate) );
    PanicFalse( SourceSynchronise(audio_source_a, audio_source_b) );
    PanicFalse( SinkConfigure(audio_sink_a, STREAM_CODEC_OUTPUT_RATE,
sampleRate) );
```

```
    PanicFalse( SinkConfigure(audio_sink_b, STREAM_CODEC_OUTPUT_RATE,
sampleRate) );
    PanicFalse( SinkSynchronise(audio_sink_a, audio_sink_b) );
    /* Set up codec gains */
#ifdef MIC_INPUT
    PanicFalse( SourceConfigure(audio_source_a,
STREAM_CODEC_MIC_INPUT_GAIN_ENABLE, 1) );
    PanicFalse( SourceConfigure(audio_source_b,
STREAM_CODEC_MIC_INPUT_GAIN_ENABLE, 1) );
    PanicFalse(MicbiasConfigure(MIC_BIAS_0, MIC_BIAS_ENABLE,
MIC_BIAS_FORCE_ON));
    PanicFalse(MicbiasConfigure(MIC_BIAS_1, MIC_BIAS_ENABLE,
MIC_BIAS_FORCE_ON));
#else
    PanicFalse( SourceConfigure(audio_source_a,
STREAM_CODEC_MIC_INPUT_GAIN_ENABLE, 0) );
    PanicFalse( SourceConfigure(audio_source_b,
STREAM_CODEC_MIC_INPUT_GAIN_ENABLE, 0) );
#endif
    PanicFalse( SourceConfigure(audio_source_a, STREAM_CODEC_INPUT_GAIN,
9) );
    PanicFalse( SourceConfigure(audio_source_b, STREAM_CODEC_INPUT_GAIN,
9) );
    PioSetPio(POWER_AMP_PIO, TRUE);
    PanicFalse( SinkConfigure(audio_sink_a, STREAM_CODEC_OUTPUT_GAIN,
15) );
    PanicFalse( SinkConfigure(audio_sink_b, STREAM_CODEC_OUTPUT_GAIN,
15) );
#ifdef BYPASS_KALIMBA
    /* Plug Left ADC directly into left DAC */
    PanicFalse( StreamConnect(audio_source_a, audio_sink_a) );
    /* Plug Right ADC directly into right DAC */
    PanicFalse( StreamConnect(audio_source_b, audio_sink_b) );
#else
    /* Plug Left ADC into port 0 */
    PanicFalse( StreamConnect(audio_source_a, StreamKalimbaSink(0)) );
    /* Plug Right ADC into port 1 */
    PanicFalse( StreamConnect(audio_source_b, StreamKalimbaSink(1)) );
    /* Plug port 0 into Left DAC */
    PanicFalse( StreamConnect(StreamKalimbaSource(0), audio_sink_a) );
    /* Plug port 1 into Right DAC */
    PanicFalse( StreamConnect(StreamKalimbaSource(1), audio_sink_b) );
#endif
}
void PioSetPio (uint16 pPIO , bool pOnOrOff)
{
```

```
    uint16 lPinVals = 0 ;
    uint16 lWhichPin  = (1<< pPIO) ;
    if ( pOnOrOff )
    {
        lPinVals = lWhichPin  ;
    }
    else
    {
        lPinVals = 0x0000;/*clr the corresponding bit*/
    }
      /*(mask,bits) setting bit to a '1' sets the corresponding port as an
output*/
    PioSetDir32( lWhichPin , lWhichPin );
      /*set the value of the pin*/
    PioSet32 ( lWhichPin , lPinVals ) ;
}
```

B my_second_dsp_app Kalimba DSP code

The following Kalimba code is provided in:

<ADK Installation folder>\Examples\my_second_dsp_app

Do not cut and paste the following sample code for use.

```
//
*****************************************************************************
****************
// Copyright (c) 2003 - 2015 Qualcomm Technologies International, Ltd.
//
*****************************************************************************
****************
//
// DESCRIPTION
// An example app for routing audio through the Kalimba DSP from ADC to
DAC
//
// NOTES
//
// What the code does:
//    Sets up cbuffers (circular connection buffers) for reading audio
from
//    the ADC and routing to the DAC. Cbuffers are serviced by timer
//    interrupts.
//
*****************************************************************************
****************
// 1.5ms is chosen as the interrupt rate for the audio input/output
// because: ADC/DAC MMU buffer is 512 bytes = 256samples
//                               = 256 samples = 5.0 ms @ 48kHz (approx)
// Assume absolute worst case jitter on interrupts = 1ms
// Hence 1.5ms (much less than 5 ms) would be OK for audio input/output
interrupts
#define $TMR_PERIOD_AUDIO_COPY         1500
#define $AUDIO_CBUFFER_SIZE            512
#define $DATA_COPIED                   0
#define $DATA_NOT_COPIED               1
```

```
#define $TONES_BLOCK_SIZE              $AUDIO_CBUFFER_SIZE/2
// Standard includes
#include "core_library.h"
#include "cbops_library.h"
#include "user_code.h"
.MODULE $M.main;
   .CODESEGMENT PM;
   .DATASEGMENT DM;
   $main:
   // ** Setup ports that are to be used **
   .CONST  $AUDIO_LEFT_IN_PORT    ($cbuffer.READ_PORT_MASK +
$cbuffer.FORCE_SIGN_EXTEND + 0);
   .CONST  $AUDIO_RIGHT_IN_PORT   ($cbuffer.READ_PORT_MASK +
$cbuffer.FORCE_SIGN_EXTEND + 1);
   .CONST  $AUDIO_LEFT_OUT_PORT   ($cbuffer.WRITE_PORT_MASK +
$cbuffer.FORCE_SIGN_EXTEND + 0);
   .CONST  $AUDIO_RIGHT_OUT_PORT  ($cbuffer.WRITE_PORT_MASK +
$cbuffer.FORCE_SIGN_EXTEND + 1);
   // ** Allocate memory for cbuffers **
   // cbuffers are 'circular connection buffers'
   .VAR/DMCIRC $audio_in_left[$AUDIO_CBUFFER_SIZE];
   .VAR/DMCIRC $audio_in_right[$AUDIO_CBUFFER_SIZE];
   .VAR/DMCIRC $audio_out_left[$AUDIO_CBUFFER_SIZE];
   .VAR/DMCIRC $audio_out_right[$AUDIO_CBUFFER_SIZE];
   // ** Allocate memory for user_code **
   // Initialize filter coefficients
   .VAR/DMCIRC $user_code.fir_filter.filter_coefs[$FILTER_LENGTH] =

#include "fir_coefs.dat"
                                                                    ;
   // Initialize a circular buffer for filter coefficients
   .VAR/DMCIRC
$user_code.fir_filter.filter_data_buffer_left[$FILTER_LENGTH];
   .VAR/DMCIRC
$user_code.fir_filter.filter_data_buffer_right[$FILTER_LENGTH];
   // ** Allocate memory for cbuffer structures **
   .VAR $audio_in_left_cbuffer_struc[$cbuffer.STRUC_SIZE] =
       LENGTH($audio_in_left),         // Size
       &$audio_in_left,                // Read pointer
       &$audio_in_left;                // Write pointer
   .VAR $audio_in_right_cbuffer_struc[$cbuffer.STRUC_SIZE] =
       LENGTH($audio_in_right),        // Size
       &$audio_in_right,               // Read pointer
       &$audio_in_right;               // Write pointer
   .VAR $audio_out_left_cbuffer_struc[$cbuffer.STRUC_SIZE] =
       LENGTH($audio_out_left),        // Size
```

```
        &$audio_out_left,               // Read pointer
        &$audio_out_left;               // Write pointer
    .VAR $audio_out_right_cbuffer_struc[$cbuffer.STRUC_SIZE] =
        LENGTH($audio_out_right),       // Size
        &$audio_out_right,              // Read pointer
        &$audio_out_right;              // Write pointer
    // ** Allocate memory for timer structures **
    .VAR $audio_in_timer_struc[$timer.STRUC_SIZE];
    .VAR $audio_out_timer_struc[$timer.STRUC_SIZE];
    // Input:
    // ------
    // ** Allocate memory for cbops stereo input copy routines **
    .VAR $audio_in_copy_struc[] =
      $audio_in_left_op,
       2,                               // Number of inputs
       $AUDIO_LEFT_IN_PORT,             // Input
       $AUDIO_RIGHT_IN_PORT,            // Input
       2,                               // Number of outputs
       &$audio_in_left_cbuffer_struc,   // Output
       &$audio_in_right_cbuffer_struc;  // Output
   #ifdef FIR_FILTER
    // FIR Filter Operator
    .BLOCK $audio_in_left_op;
       .VAR $audio_in_left_op.next = &$audio_in_right_op;
       .VAR $audio_in_left_op.func = &$user_code.fir_filter;
       .VAR $audio_in_left_op.param[$user_code.fir_filter.STRUC_SIZE] =
          0,                                          // Input index
          2,                                          // Output index
          &$user_code.fir_filter.filter_coefs,        // Coefficient
Buffer Address
          &$user_code.fir_filter.filter_data_buffer_left,    // History
Buffer Address
          $FILTER_LENGTH,                             // Filter Length
          0;                                          // Shift amount
    .ENDBLOCK;
    .BLOCK $audio_in_right_op;
       .VAR $audio_in_right_op.next = $cbops.NO_MORE_OPERATORS;
       .VAR $audio_in_right_op.func = &$user_code.fir_filter;
       .VAR $audio_in_right_op.param[$user_code.fir_filter.STRUC_SIZE] =
          1,                            // Input index
          3,                            // Output index
          &$user_code.fir_filter.filter_coefs,        // Coefficient
Buffer Address
          &$user_code.fir_filter.filter_data_buffer_right,   // History
Buffer Address
          $FILTER_LENGTH,                             // Filter Length
```

```
      0;                              // Shift amount
 .ENDBLOCK;
#else
 .BLOCK $audio_in_left_op;
    .VAR $audio_in_left_op.next = &$audio_in_right_op;
    .VAR $audio_in_left_op.func = &$cbops.shift;;
    .VAR $audio_in_left_op.param[$cbops.shift.STRUC_SIZE] =
       0,                                          // Input index
       2,                                          // Output index
       0;                                            // Shift amount
 .ENDBLOCK;
 .BLOCK $audio_in_right_op;
    .VAR $audio_in_right_op.next = $cbops.NO_MORE_OPERATORS;
    .VAR $audio_in_right_op.func = &$cbops.shift;
    .VAR $audio_in_right_op.param[$cbops.shift.STRUC_SIZE] =
       1,                              // Input index
       3,                              // Output index
       0;                              // Shift amount
 .ENDBLOCK;
#endif
 // Output:
 // ------
 // ** allocate memory for cbops stereo output copy routines **
 .VAR $audio_out_copy_struc[] =
    &$audio_out_left_shift_op,       // First operator block
    2,                               // Number of inputs
    &$audio_out_left_cbuffer_struc,  // Input
    &$audio_out_right_cbuffer_struc, // Input
    2,                               // Number of outputs
    $AUDIO_LEFT_OUT_PORT,            // Output
    $AUDIO_RIGHT_OUT_PORT;           // Output
 .BLOCK $audio_out_left_shift_op;
    .VAR $audio_out_left_shift_op.next = $audio_out_right_shift_op;
    .VAR $audio_out_left_shift_op.func = &$cbops.shift;
    .VAR $audio_out_left_shift_op.param[$cbops.shift.STRUC_SIZE] =
       0,                              // Input index.
       2,                              // Output index
       0;                              // Shift amount
 .ENDBLOCK;
 .BLOCK $audio_out_right_shift_op;
    .VAR $audio_out_right_shift_op.next = $cbops.NO_MORE_OPERATORS;
    .VAR $audio_out_right_shift_op.func = &$cbops.shift;
    .VAR $audio_out_right_shift_op.param[$cbops.shift.STRUC_SIZE] =
       1,                              // Input index.
       3,                              // Output index
       0;                              // Shift amount
```

```
   .ENDBLOCK;
   // Input to Output:
   // ----------------
   // ** allocate memory for cbops stereo loopback copy routines **
   .VAR $audio_loopback_copy_struc[] =
      &$audio_loopback_left_copy_op,   // First operator block
      2,                               // Number of inputs
      &$audio_in_left_cbuffer_struc,   // Input
      &$audio_in_right_cbuffer_struc,  // Input
      2,                               // Number of outputs
      &$audio_out_left_cbuffer_struc,  // Output
      &$audio_out_right_cbuffer_struc; // Output
   .BLOCK $audio_loopback_left_copy_op;
      .VAR $audio_loopback_left_copy_op.next = &
$audio_loopback_right_copy_op;
      .VAR $audio_loopback_left_copy_op.func = &$cbops.copy_op;
      .VAR $audio_loopback_left_copy_op.param[$cbops.copy_op.STRUC_SIZE] =
         0,                            // Input index
         2;                            // Output index
   .ENDBLOCK;
   .BLOCK $audio_loopback_right_copy_op;
      .VAR $audio_loopback_right_copy_op.next = $cbops.NO_MORE_OPERATORS;
      .VAR $audio_loopback_right_copy_op.func = &$cbops.copy_op;
      .VAR $audio_loopback_right_copy_op.param[$cbops.copy_op.STRUC_SIZE]
=
         1,                            // Input index
         3;                            // Output index
   .ENDBLOCK;
   // Initialise the stack library
   call $stack.initialise;
   // Initialise the interrupt library
   call $interrupt.initialise;
   // Initialise the message library
   call $message.initialise;
   // Initialise the cbuffer library
   call $cbuffer.initialise;
   // Tell vm we're ready and wait for the go message
   call $message.send_ready_wait_for_go;
   // left and right audio channels from the mmu have been synced to each
   // other by the vm app but are free running in that the dsp doesn't
tell
   // them to start.  We need to make sure that our copying between the
   // cbuffers and the mmu buffers starts off in sync with respect to left
   // and right channels.  To do this we make sure that when we start the
   // copying timers that there is no chance of a buffer wrap around
   // occuring within the timer period. The easiest way to do this is to
```

```
   // start the timers just after a buffer wrap around occurs.
   // Wait for ADC buffers to have just wrapped around
   wait_for_adc_buffer_wraparound:
      r0 = $AUDIO_LEFT_IN_PORT;
      call $cbuffer.calc_amount_data;
      // If the amount of data in the buffer is less than 32 bytes then a
      // buffer wrap around must have just occurred.
    Null = r0 - 32;                    // calc_amount_data r0 returns
number of words/samples
   if POS jump wait_for_adc_buffer_wraparound;
   // Start timer that copies input samples
   r1 = &$audio_in_timer_struc;
   r2 = $TMR_PERIOD_AUDIO_COPY;
   r3 = &$audio_in_copy_handler;
   call $timer.schedule_event_in;
   // Wait for DAC buffers to have just wrapped around
   wait_for_dac_buffer_wraparound:
      r0 = $AUDIO_LEFT_OUT_PORT;
      call $cbuffer.calc_amount_space;
      // If the amount of space in the buffer is less than 32 bytes then a
      // buffer wrap around must have just occurred.
      Null = r0 - 32;                  // calc_amount_data r0 returns
number of words/samples
   if POS jump wait_for_dac_buffer_wraparound;
   // Start timer that copies output samples
   r1 = &$audio_out_timer_struc;
   r2 = $TMR_PERIOD_AUDIO_COPY;
   r3 = &$audio_out_copy_handler;
   call $timer.schedule_event_in;
   // Start a loop to copy the data from the input through to the output
buffers
   copy_loop:
   call $loopback_copy;
   Null = r0 - $DATA_NOT_COPIED;
   if Z call $timer.1ms_delay;
   jump copy_loop;
.ENDMODULE;
//
*****************************************************************************
***************
// MODULE:
//    $audio_in_copy_handler
//
// DESCRIPTION:
//    Function called on an interrupt timer to copy samples from MMU
//    input ports to internal cbuffers.
```

```
//
//
*****************************************************************************
****************
.MODULE $M.audio_in_copy_handler;
   .CODESEGMENT PM;
   .DATASEGMENT DM;
   $audio_in_copy_handler:
   // Push rLink onto stack
   $push_rLink_macro;
   // Copy data whatever mode we are in to keep in sync
   // transfer data from mmu port to internal cbuffer
   r8 = &$audio_in_copy_struc;
   call $cbops.copy;
   // Post another timer event
   r1 = &$audio_in_timer_struc;
   r2 = $TMR_PERIOD_AUDIO_COPY;
   r3 = &$audio_in_copy_handler;
   call $timer.schedule_event_in;
   // Pop rLink from stack
   jump $pop_rLink_and_rts;
.ENDMODULE;
//
*****************************************************************************
****************
// MODULE:
//    $audio_out_copy_handler
//
// DESCRIPTION:
//    Function called on an interrupt timer to copy samples from internal
//    cbuffers to output MMU ports.
//
//
*****************************************************************************
****************
.MODULE $M.audio_out_copy_handler;
   .CODESEGMENT PM;
   .DATASEGMENT DM;
   $audio_out_copy_handler:
   // Push rLink onto stack
   $push_rLink_macro;
   // Transfer data from internal cbuffer to mmu port
   r8 = &$audio_out_copy_struc;
   call $cbops.copy;
   // Post another timer event
   r1 = &$audio_out_timer_struc;
```

```
   r2 = $TMR_PERIOD_AUDIO_COPY;
   r3 = &$audio_out_copy_handler;
   call $timer.schedule_event_in;
   // Pop rLink from stack
   jump $pop_rLink_and_rts;
.ENDMODULE;
//
*****************************************************************************
***************
// MODULE:
//    $loopback_copy
//
// DESCRIPTION:
//    Routine to copy data from both input channels into the corresponding
//    output buffer.  Routine is on a long delay between calls so need to
//    ensure we copy enough data.
//
// INPUTS:
//    none
//
// OUTPUTS:
//    r0 = DATA_COPIED / DATA_NOT_COPIED
//
// TRASHED REGISTERS:
//    r8
//    Called buffer routines also trash:
//    r1, r2, r3, r4, I0, L0, I1, L1, r10, DO LOOP
//
//
*****************************************************************************
***************
.MODULE $M.loopback_copy;
   .CODESEGMENT PM;
   .DATASEGMENT DM;
   $loopback_copy:
   // push rLink onto stack
   $push_rLink_macro;
   // Check if there is enough data in the input buffer
   r0 = &$audio_in_left_cbuffer_struc;
   call $cbuffer.calc_amount_data;
   Null = r0 - $TONES_BLOCK_SIZE;
   if NEG jump dont_copy;
   r0 = &$audio_in_right_cbuffer_struc;
   call $cbuffer.calc_amount_data;
   Null = r0 - $TONES_BLOCK_SIZE;
   if NEG jump dont_copy;
```

```
    // Check if there is enough data in the output buffer
    r0 = &$audio_out_left_cbuffer_struc;
    call $cbuffer.calc_amount_space;
    Null = r0 - $TONES_BLOCK_SIZE;
    if NEG jump dont_copy;
    r0 = &$audio_out_right_cbuffer_struc;
    call $cbuffer.calc_amount_space;
    Null = r0 - $TONES_BLOCK_SIZE;
    if NEG jump dont_copy;
    // Block interrupts when copying sample data
    call $interrupt.block;
    // Copy the data between the buffers
    r8 = &$audio_loopback_copy_struc;
    call $cbops.copy;
    // Now unblock interrupts
    call $interrupt.unblock;
    // Indicate DATA_COPIED and return
    r0 = $DATA_COPIED;
    // pop rLink from stack
    jump $pop_rLink_and_rts;
    // Indicate DATA_NOT_COPIED and return
    dont_copy:
    r0 = $DATA_NOT_COPIED;
    // pop rLink from stack
    jump $pop_rLink_and_rts;
.ENDMODULE;
```

C fir_filter DSP library code

The following code is provided in:

<ADK Installation folder>\Examples\my_second_dsp_app

Do not cut and paste the following sample code for use.

```
//
*****************************************************************************
****************
// Copyright (c) 2003 - 2015 Qualcomm Technologies International, Ltd.
//
// FIR Filter Operator performs FIR filter
//
// When using this operator, the following data structure is used:
//    - $user_code.fir_filter.INPUT_START_INDEX_FIELD = Pointer to the
input buffer
//    - $user_code.fir_filter.OUTPUT_START_INDEX_FIELD = Pointer to the
output buffer
//    - $user_code.fir_filter.COEFFICIENT_BUFFER_ADDR = Pointer to the
coefficient buffer
//    - $user_code.fir_filter.HISTORY_BUFFER_ADDR = Pointer to the
history buffer
//    - $user_code.fir_filter.FILTER_LENGTH = Length of FIR filter
//    - $user_code.fir_filter.SHIFT_AMOUNT = Shift amount
//
//
*****************************************************************************
****************
#include "stack.h"
#include "user_code.h"
.PUBLIC $user_code.fir_filter;
.MODULE $M.user_code.fir_filter;
   .PRIVATE;
   .CODESEGMENT PM;
   .DATASEGMENT DM;
   // ** Function vector **
   .VAR $user_code.fir_filter[$user_code.function_vector.STRUC_SIZE] =
      $user_code.function_vector.NO_FUNCTION,         // Reset function,
```

```
not used
      $user_code.function_vector.NO_FUNCTION,         // Amount to use
function, not used
      $user_code.fir_filter.main;                     // Main function
.ENDMODULE;
//
*****************************************************************************
***************
// MODULE:
//    $user_code.fir_filter.main
//
// DESCRIPTION:
//    Single channel FIR filter function
//
// INPUTS:
//    - r6 = Pointer to the list of input and output buffer pointers
//    - r7 = Pointer to the list of buffer lengths
//    - r8 = Pointer to operator structure
//    - r10 = Number of samples to process
//
// OUTPUTS:
//    - none
//
// TRASHED REGISTERS:
//    r0-5, r10, rMAC, I0, I1, I4, I5, L0, L1, L4, L5, DoLoop
//
//
*****************************************************************************
***************
.MODULE $M.user_code.fir_filter.main;
   .PRIVATE;
   .CODESEGMENT PM;
   .DATASEGMENT DM;
   $user_code.fir_filter.main:
   // push rLink onto stack
   $push_rLink_macro;
   // Get the offset to the read buffer to use
   r0 = M[r8 + $user_code.fir_filter.INPUT_START_INDEX_FIELD];
   // Get the input buffer read address
   r1 = M[r6 + r0];
   // Store the value in I0
   I0 = r1;
   // Get the input buffer length
   r1 = M[r7 + r0];
   // Store the value in
   L0 = r1;
```

```
   // Get the offset to the write buffer to use
   r0 = M[r8 + $user_code.fir_filter.OUTPUT_START_INDEX_FIELD];
   // Get the output buffer write address
   r1 = M[r6 + r0];
   // Store the value in I4
   I4 = r1;
   // Get the output buffer length
   r1 = M[r7 + r0];
   // Store the value in L4
   L4 = r1;
   // Set coefficient buffer's base address
   r0 = M[r8 + $user_code.fir_filter.COEFFICIENT_BUFFER_ADDR];
   I1 = r0;
   // Load shift amount
   r3 = M[r8 + $user_code.fir_filter.SHIFT_AMOUNT];
   // Load history buffer pointer
   r0 = M[r8 + $user_code.fir_filter.HISTORY_BUFFER_ADDR];
   I5 = r0;
   r0 = M[r8 + $user_code.fir_filter.FILTER_LENGTH];
   L5 = r0;
   L1 = r0;
   do loop;
      r0 = M[I0,1];
      r0 = r0 ASHIFT r3;
      // Store sample into memory address I5 circular buffer;
      M[I5, 0] = r0;
      // Start loop for summing (filter length samples * filter
coefficients)
      r5 = L5;
      rMAC = 0.0; // Clear rMAC
      r1 = M[I1,0], r2 = M[I5,0]; // r1 = h(0) r2 = x(0)
sum_loop1:
      r1 = M[I1,1], r2 = M[I5,1], // Fetch the next sample while doing
MAC in parallel.
      rMAC = rMAC + r1 * r2;
      r5 = r5 - 1;
      if NZ jump sum_loop1;
      // Dummy read to decrement I5
      r2 = M[I5, -1];
      // Output sample from delay loop buffer
      M[I4,1] = rMAC;
   loop:
   // Update history buffer pointer
   r0 = I5;
   M[r8 + $user_code.fir_filter.HISTORY_BUFFER_ADDR] = r0;
   // Zero the length registers and write the last sample
```

```
   L0 = 0;
   L1 = 0;
   L5 = 0;
   L4 = 0;
   // pop rLink from stack
   $pop_rLink_macro;
   rts;
.ENDMODULE;
//
*****************************************************************************
***************
// MODULE:
//    $user_code.fir_filter.main
//
// DESCRIPTION:
//    Single channel FIR filter function
//
// INPUTS:
//    - r6 = Pointer to the list of input and output buffer pointers
//    - r7 = Pointer to the list of buffer lengths
//    - r8 = Pointer to operator structure
//    - r10 = Number of samples to process
//
// OUTPUTS:
//    - none
//
// TRASHED REGISTERS:
//    r0-5, r10, rMAC, I0, I1, I4, I5, L0, L1, L4, L5, DoLoop
//
//
*****************************************************************************
***************
.MODULE $M.user_code.fir_filter.main;
   .PRIVATE;
   .CODESEGMENT PM;
   .DATASEGMENT DM;
   $user_code.fir_filter.main:
   // push rLink onto stack
   $push_rLink_macro;
   // Get the offset to the read buffer to use
   r0 = M[r8 + $user_code.fir_filter.INPUT_START_INDEX_FIELD];
   // Get the input buffer read address
   r1 = M[r6 + r0];
   // Store the value in I0
   I0 = r1;
   // Get the input buffer length
```

```
   r1 = M[r7 + r0];
   // Store the value in
   L0 = r1;
   // Get the offset to the write buffer to use
   r0 = M[r8 + $user_code.fir_filter.OUTPUT_START_INDEX_FIELD];
   // Get the output buffer write address
   r1 = M[r6 + r0];
   // Store the value in I4
   I4 = r1;
   // Get the output buffer length
   r1 = M[r7 + r0];
   // Store the value in L4
   L4 = r1;
   // Set coefficient buffer's base address
   r0 = M[r8 + $user_code.fir_filter.COEFFICIENT_BUFFER_ADDR];
   I1 = r0;
   // Load shift amount
   r3 = M[r8 + $user_code.fir_filter.SHIFT_AMOUNT];
   // Load history buffer pointer
   r0 = M[r8 + $user_code.fir_filter.HISTORY_BUFFER_ADDR];
   I5 = r0;
   r0 = M[r8 + $user_code.fir_filter.FILTER_LENGTH];
   L5 = r0;
   L1 = r0;
   do loop;
      r0 = M[I0,1];
      r0 = r0 ASHIFT r3;
      // Store sample into memory address I5 circular buffer;
      M[I5, 0] = r0;
      // Start loop for summing (filter length samples * filter
coefficients)
      r5 = L5;
      rMAC = 0.0;                          // Clear rMAC
      r1 = M[I1,0], r2 = M[I5,0]; // r1 = h(0) r2 = x(0)
      sum_loop1:
         r1 = M[I1,1],   r2 = M[I5,1],     // Fetch the next sample while
doing MAC in parallel.
         rMAC = rMAC + r1 * r2;
         r5 = r5 - 1;
      if NZ jump sum_loop1;
      // Dummy read to decrement I5
      r2 = M[I5, -1];
      // Output sample from delay loop buffer
      M[I4,1] = rMAC;
   loop:
   // Update history buffer pointer
```

```
    r0 = I5;
    M[r8 + $user_code.fir_filter.HISTORY_BUFFER_ADDR] = r0;
    // Zero the length registers and write the last sample
    L0 = 0;
    L1 = 0;
    L5 = 0;
    L4 = 0;
    // pop rLink from stack
    $pop_rLink_macro;
    rts;
.ENDMODULE;
```

D user_code.h in my_second_dsp header file

This file includes the required definitions for building the `fir_filter` library as well as declaring FIR specific variables in the main DSP application code. These definitions are included in the `fir_filter` operator code as well as the main DSP application by including `user_code.h`.

NOTE The following code is provided in:

<ADK Installation folder>\Examples\my_second_dsp_app

Do not cut and paste the following sample code for use.

```
/
*****************************************************************************
*****
Copyright (c) 2015 Qualcomm Technologies International, Ltd.
*****************************************************************************
*****/
#ifndef USER_CODE_HEADER_INCLUDED
#define USER_CODE_HEADER_INCLUDED
    // ** Function vector parameters **
    .CONST  $user_code.function_vector.RESET_FIELD                0;
    .CONST  $user_code.function_vector.AMOUNT_TO_USE_FIELD        1;
    .CONST  $user_code.function_vector.MAIN_FIELD                 2;
    .CONST  $user_code.function_vector.STRUC_SIZE                 3;
    .CONST  $user_code.function_vector.NO_FUNCTION                0;
    // FIR Filter operator structure definition
    .CONST  $user_code.fir_filter.INPUT_START_INDEX_FIELD         0;
    .CONST  $user_code.fir_filter.OUTPUT_START_INDEX_FIELD        1;
    .CONST  $user_code.fir_filter.COEFFICIENT_BUFFER_ADDR         2;
    .CONST  $user_code.fir_filter.HISTORY_BUFFER_ADDR             3;
    .CONST  $user_code.fir_filter.FILTER_LENGTH                   4;
    .CONST  $user_code.fir_filter.SHIFT_AMOUNT                    5;
    .CONST  $user_code.fir_filter.STRUC_SIZE                      6;
     // Defines for FIR filter
    #define $FILTER_LENGTH                                       111
#endif // USER_CODE_HEADER_INCLUDED
```

Document references

Document	Reference
Qualcomm BlueCore technology xIDE User Guide	80-CT405-1/CS-00101500-UG
My First Kalimba DSP Application	80-CT398-1/CS-00101420-AN
Qualcomm BlueCore Classic vs Native VM vs Assisted Native VM Application Note	80-CT403-1/CS-00122636-AN

Terms and definitions

Term	Definition
ADC	Analog to Digital Converter
ADK	Audio Development Kit
BlueCore	Group term for the range of Qualcomm Bluetooth wireless technology ICs
Bluetooth	Set of technologies providing audio and data transfer over short-range radio connections.
CODEC	COder DECoder
DAC	Digital to Analog Converter
DSP	Digital Signal Processor
f_c	Cut-off Frequency
FIR	Finite Impulse Response
f_s	Sample Frequency
IC	Integrated Circuit
IP	Intellectual Property
Kalimba	Qualcomm's on-chip DSP processor.
MMU	Memory Management Unit
PCM	Pulse Code Modulation
QTIL	Qualcomm technologies Internaona
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
UART	Universal Asynchronous Receiver/Transmitter
VM	Virtual Machine
XAP	Low-power silicon-efficient RISC microprocessor.
xIDE	Integrated Development Environment for ADK