Qualcomm Technologies International, Ltd.

My First 24-bit Qualcomm Kalimba DSP Application

Application Note

80-CT422-1 Rev. AG

October 18, 2017

Revision history

Revision	Date	Description
1	JUL 2014	Initial release. Alternative document number CS-00317863-AN.
2	JUL 2014	MMU audio bandwidth for 24-bit audio updated
3	NOV 2014	Source code sections updated
4	DEC 2014	Source code sections updated
5	DEC 2015	Source code sections updated
6	SEP 2016	Updated to conform to QTI standards; no technical content was changed in this document revision
AG	AUG 2017	Added to the Content Management System. DRN updated to Agile number. No change to the technical content.

Contents

Tables

Figures

1 my_first_24bit_dsp_app

`my_first_24bit_dsp_app` introduces 24-bit audio processing on the CSR8675.

The application uses the supplied Kalimba libraries. It uses connection buffers to read and write from the 24-bit audio ports and the `cbops` linked list of operators to copy and process audio.

This `cbops` framework provides the means to configure operators to achieve the required processing of the audio. In addition, new custom operators can be inserted as required.

`my_first_24bit_dsp_app` works on CSR8675.

2 Building and running a 24-bit DSP application

The example application `my_first_24bit_dsp_app` can be used to build and run a basic Qualcomm® Kalimba™ DSP application. It is relevant for CSR8675.

`my_first_24bit_dsp_app` is an example audio application. It uses the Kalimba DSP and the xIDE development environment. The application routes stereo audio through the DSP and describes how to use the Kalimba libraries.

Getting started

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*.

2.1 Building and running 24-bit Kalimba DSP application on CSR8675 development board

1. Connect the H13223 headphone amplifier to the H13179 board, as shown in Figure 2-1. Also connect an aerial and 3.7 V battery.

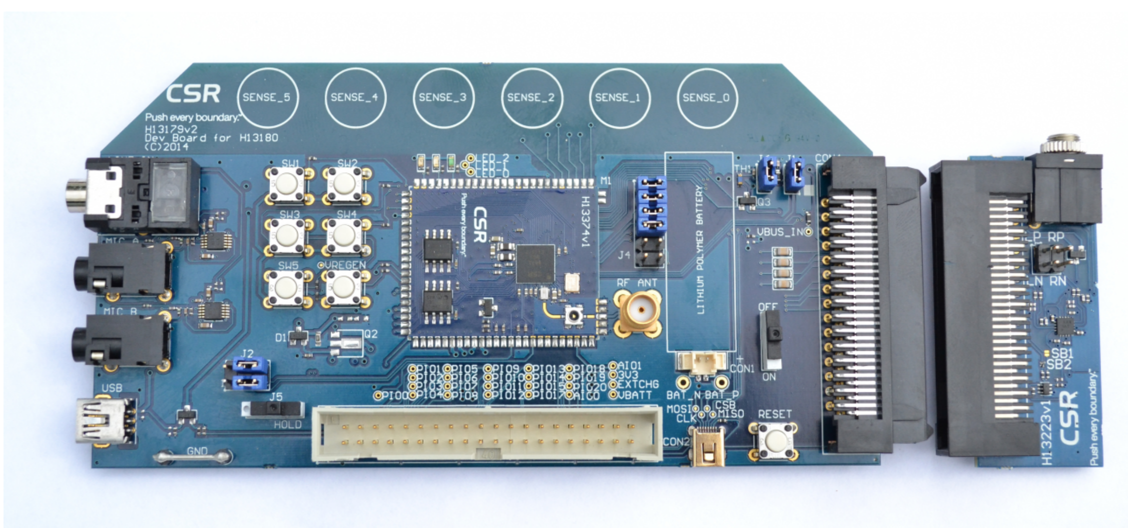

Figure 2-1 Connecting H13223 headphone amplifier board to the H13179/H13374 (CSR8675) board

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.

Figure 2-2 USB-SPI converter (CNS10020)

3. Connect the I^2S audio to the 40 Pin connector **J5**.

Table 2-1 Pin configuration for I^2S connections

J5 Pin	Shared PIO	I^2S Function
21	PIO[17]	Data In
22	PIO[18]	Data Out
23	PIO[19]	Word Clock / Sync
24	PIO[20]	Bit Clock

4. Connect a USB cable to the CSR8675 board to provide 5 V power supply.
5. Start PSTool and use it to change the value of PSKEY_MMU_BW_AUDIO if necessary. See for information on the CSR8675 MMU audio bandwidth.
6. Close PSTool and launch the ADK.

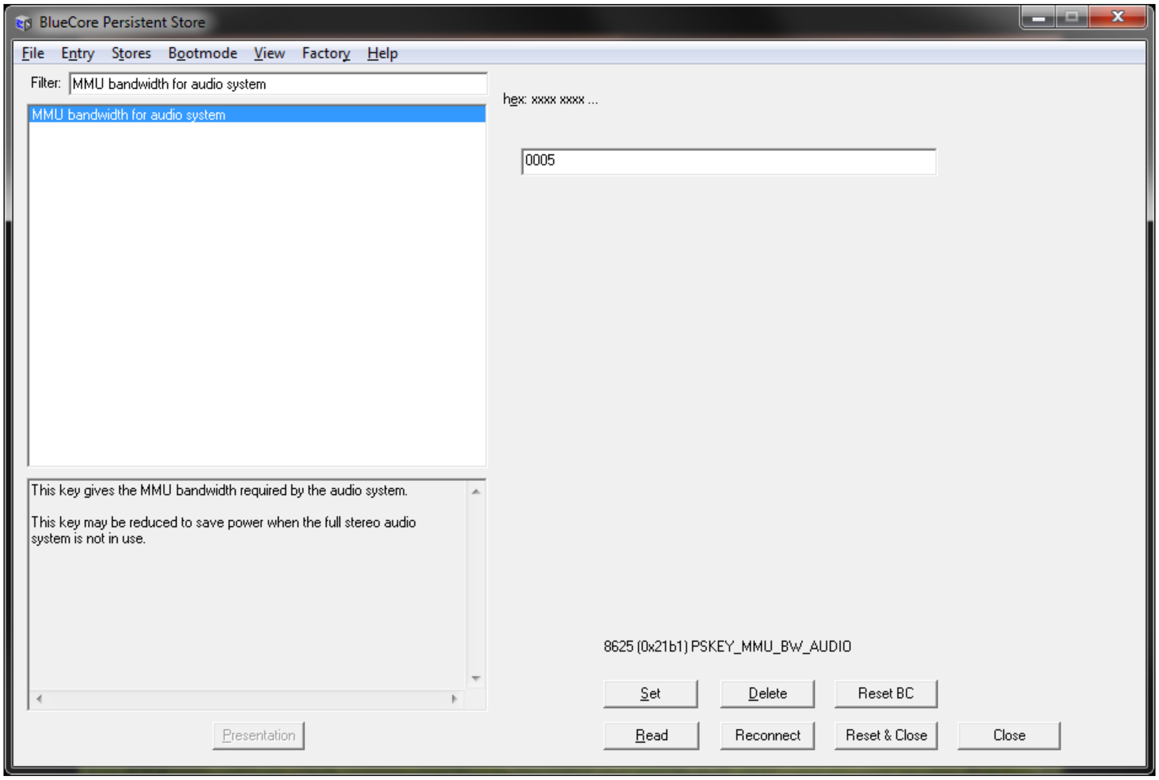

Figure 2-3 Configuring the MMU bandwidth for audio PS Key.

7. Locate the project workspace **my_first_24bit_dsp_app.xiw** by selecting **Project | Open Workspace ...** within xIDE.
   The project file `my_first_24bit_dsp_app.xiw` is in:

   `<ADK_INSTALL>\apps\examples\my_first_24bit_dsp_app`

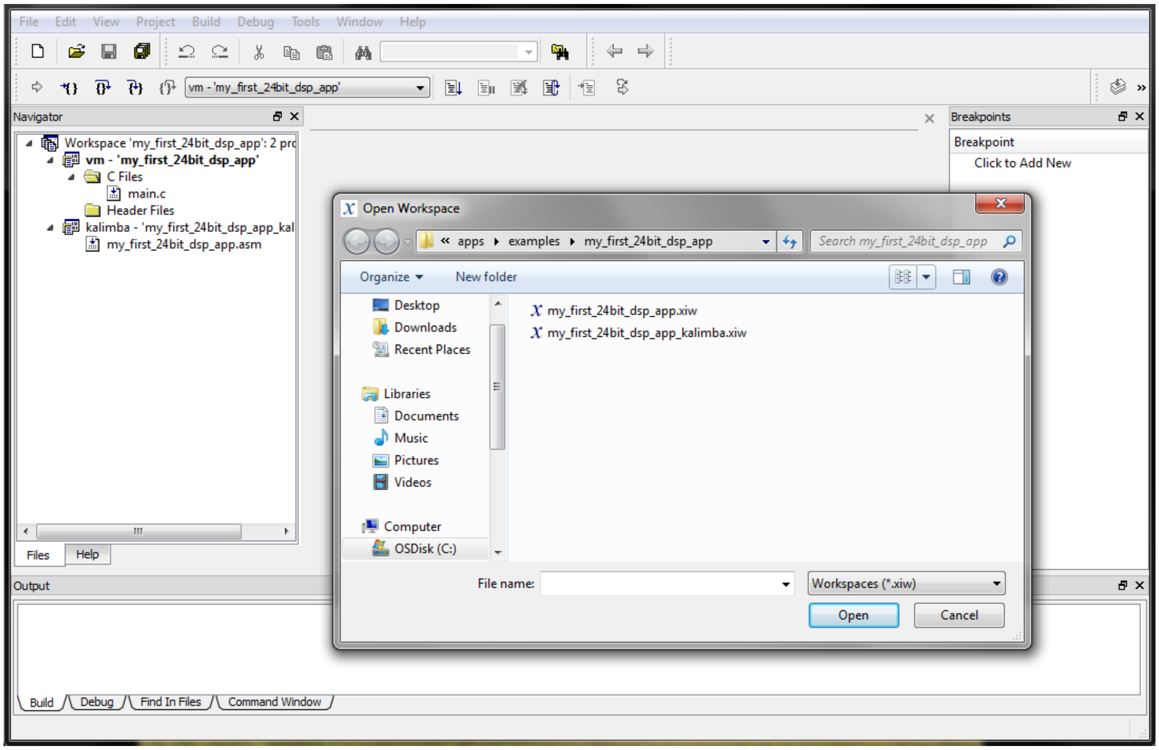

Figure 2-4 Open workspace

8. Ensure the correct Debug Transport is selected, the setting for this is under the **Debug | Transport...** menu in xIDE.
9. 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.
10. Build the application by pressing the **F7** key, which compiles and links the code for both VM and DSP.
11. Alternatively, right-click on `kalimba – 'my_first_24bit_dsp_app_kalimba`, select **Build** and then right-click on `vm – 'my_first_24bit_dsp_app` and select **Build.**
12. Run the Kalimba DSP application by pressing the **F5** key.
13. This step flashes the executable binary image to the hardware and runs the application in debugging mode.
14. 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.
    You must set the **Build merge** to **Yes** in the VM **Project Properties** window or the executable image `merge.xpv (.xdv)` does not generate, see Figure 2-5.

Figure 2-5 BlueFlash

After completing Step 11, an I²S amplifier or DAC can play a PCM audio stream. This example code routes the audio from I²S input to DSP and then from DSP to the I²S output.

As an exercise, you can bypass the DSP and route audio from I²S input to output directly without passing the DSP. To do this, comment out Line 19: `#define BYPASS_KALIMBA`.

You can also add the symbol `BYPASS_KALIMBA` in **Define symbols** and repeat Steps 9 and 10 or Step 11. If there is more than one symbol (pre-processors), separate all symbols with a comma.

To change whether the CSR8675 is I²S master or slave, change the define on line 22 of `main.c`.

2.2 DSP build options for 24-bit audio

The defaults are for copying 24-bit audio data through the DSP from input ports to output ports using a shift operator and DC removal operator.

- 16-bit mode may use used instead of 24-bit mode, controlled by `AUDIO_24BIT`.
- The DC removal operator may be enabled / disabled using `USE_DC_REMOVE_OPERATOR`.
- The copy operator may be used instead of the shift operator, controlled by `USE_SHIFT_OPERATOR`.

```
#define AUDIO_24BIT              // Enable 24 bit audio option
#define USE_DC_REMOVE_OPERATOR   // Enable DC removal operator
#define USE_SHIFT_OPERATOR       // Use shift operator instead of copy
operator
```

`$SHIFT_AMOUNT_IN` and `$SHIFT_AMOUNT_OUT`, used with the shift operator, can be seen as simple gain controls with a step size of 6 dB.

The default of `$SHIFT_AMOUNT_OUT` is set to `-1` in 24-bit mode, which has the effect of reducing the amplitude by 6 dB at the output port.

To use the example application as a DSP pass through, disable the DC remove operator and either disable the shift operator to use the copy operator, or set `$SHIFT_AMOUNT_IN` and `$SHIFT_AMOUNT_OUT` to zero.

2.3 MMU audio bandwidth for 24-bit audio

On CSR8675, PSKEY_MMU_BW_AUDIO needs to be set according to the total audio rate requirement of the end product. This PS Key gives the MMU bandwidth required by the audio system. Its value depends upon the maximum audio rate (that is, highest of total audio IN rate and total audio OUT rate) requirement of the end product. USB audio rate is not included in the maximum audio rate. USB is treated as Host subsystem. The recommended values for this PS Key are:

1. `0x0005`: If the maximum audio rate is less than 768 Kbytes/sec or all channels are running in 16-bit mode. This value is set as default from firmware version Unified-27d onwards.
2. `0x0007`: Usually required when running use cases in 24-bit mode and maximum audio rate is greater than 768 Kbytes/sec.

Examples of when to set this PS Key:

- **Example 1:** For a product with 5 audio IN streams each configured for 16-bit audio and 48 KHz sampling rate and no audio OUT streams, the total audio rate is 480 Kbytes/sec. So the default value of `0x0005` is sufficient.
- **Example 2:** For a product with 5 audio OUT streams each configured for 24-bit audio(1) and 48 Khz sampling rate and no audio IN streams, the total audio rate is 960 Kbytes/sec. For this product the PSKEY_MMU_BW_AUDIO needs to be set `0x0007`.
- **Example 3**: For a product with 5 audio IN streams each configured for 16-bits audio and 48 KHz sampling rate and 5 audio OUT streams each configured for 24-bit audio(1) and 48 Khz sampling rate. The maximum of the two rates is used to decide the PS Key value. In this case, IN rate is 480 Kbytes/sec. OUT rate is 960 Kbytes/sec. Maximum audio rate is highest of 480 and 960 which is 960 Kbytes/sec. For this product the PSKEY_MMU_BW_AUDIO needs to be set `0x0007`.
- **Example 4**: For a product with 2 audio OUT streams each configured for 24-bit audio(1) and 96 Khz sampling rate and no audio IN streams, the total audio rate is 768 Kbytes/sec. So the default value of `0x0005` is sufficient.

NOTE (1) 24 bits per sample are counted as 4 bytes per sample while calculating the audio IN and OUT rate.

A my_first_24bit_dsp_app VM application code

The following C source code is provided in the:

`<ADK Installation folder>\Examples\my_first_24bit_dsp_app folder`

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

```
/*
Copyright (c) 2006 - 2016 Qualcomm Technologies International, Ltd.
  Basic example app for routing I2S audio through the DSP.
*/
#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 <led.h>
#include <connection.h>
/* Define the macro "BYPASS_KALIMBA" to bypass Kalimba DSP otherwise direct
I2S->I2S */
#define BYPASS_KALIMBAx
/* I2S Master / Slave switch. 1 = Master, 0 = Slave */
#define I2S_MASTER       0
/* I2S Interface frame rate*/
#define I2S_RATE     96000
/* Audio Bit Depth for use with the STREAM_AUDIO_SAMPLE_SIZE key */
#define BIT_DEPTH       24
/* Location of DSP kap file in the file system */
static const char kal[] = "my_first_24bit_dsp_app_kalimba/
my_first_24bit_dsp_app_kalimba.kap";
void start_kalimba(void);
void connect_streams(void);
void led(void);
/* Main VM routine */
int main(void)
{
    /* Load the Kalimba */
    start_kalimba();
```

```
    /* Connect up audio ports */
    connect_streams();
    /* Turn on LED */
    led();
    /* 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)
{
    /* Audio Interfaces */
    Source I2S_IN_0;
    Source I2S_IN_1;
    Sink I2S_OUT_0;
    Sink I2S_OUT_1;
    /* I2S Setup */
    /* Source StreamAudioSource (audio_hardware hardware, audio_instance
instance, audio_channel channel); */
    I2S_IN_0 = StreamAudioSource(AUDIO_HARDWARE_I2S, AUDIO_INSTANCE_0,
AUDIO_CHANNEL_SLOT_0);
    I2S_IN_1 = StreamAudioSource(AUDIO_HARDWARE_I2S, AUDIO_INSTANCE_0,
AUDIO_CHANNEL_SLOT_1);
    /* Sink StreamAudioSink (audio_hardware hardware, audio_instance
instance, audio_channel channel); */
    I2S_OUT_0 = StreamAudioSink(AUDIO_HARDWARE_I2S, AUDIO_INSTANCE_0,
AUDIO_CHANNEL_SLOT_0);
    I2S_OUT_1 = StreamAudioSink(AUDIO_HARDWARE_I2S, AUDIO_INSTANCE_0,
AUDIO_CHANNEL_SLOT_1);
    /* SinkConfigure (Sink sink, stream_config_key key, uint32 value) */
    PanicFalse(SinkConfigure(I2S_OUT_0, STREAM_I2S_MASTER_MODE, I2S_MASTER));
    PanicFalse(SinkConfigure(I2S_OUT_1, STREAM_I2S_MASTER_MODE, I2S_MASTER));
    PanicFalse(SinkConfigure(I2S_OUT_0, STREAM_I2S_SYNC_RATE, I2S_RATE));
    PanicFalse(SinkConfigure(I2S_OUT_1, STREAM_I2S_SYNC_RATE, I2S_RATE));
    PanicFalse(SinkConfigure(I2S_OUT_0, STREAM_AUDIO_SAMPLE_SIZE, BIT_DEPTH));
    PanicFalse(SinkConfigure(I2S_OUT_1, STREAM_AUDIO_SAMPLE_SIZE, BIT_DEPTH));
    PanicFalse(SinkConfigure(I2S_OUT_0, STREAM_I2S_MASTER_CLOCK_RATE, 0));
```

```
    PanicFalse(SinkConfigure(I2S_OUT_1, STREAM_I2S_MASTER_CLOCK_RATE, 0));
    PanicFalse(SinkConfigure(I2S_OUT_0, STREAM_I2S_JSTFY_FORMAT, 0));
    PanicFalse(SinkConfigure(I2S_OUT_1, STREAM_I2S_JSTFY_FORMAT, 0));
    PanicFalse(SinkConfigure(I2S_OUT_0, STREAM_I2S_LFT_JSTFY_DLY, 1));
    PanicFalse(SinkConfigure(I2S_OUT_1, STREAM_I2S_LFT_JSTFY_DLY, 1));
    PanicFalse(SinkConfigure(I2S_OUT_0, STREAM_I2S_CHNL_PLRTY, 1));
    PanicFalse(SinkConfigure(I2S_OUT_1, STREAM_I2S_CHNL_PLRTY, 1));
    PanicFalse(SinkConfigure(I2S_OUT_0, STREAM_I2S_BITS_PER_SAMPLE, 24));
    PanicFalse(SinkConfigure(I2S_OUT_1, STREAM_I2S_BITS_PER_SAMPLE, 24));
    /* SourceConfigure (Source source, stream_config_key key, uint32 value) */
    PanicFalse(SourceConfigure(I2S_IN_0, STREAM_I2S_SYNC_RATE, I2S_RATE));
    PanicFalse(SourceConfigure(I2S_IN_1, STREAM_I2S_SYNC_RATE, I2S_RATE));
    /* Set Audio bit depth */
    PanicFalse(SourceConfigure(I2S_IN_0, STREAM_AUDIO_ SAMPLE_SIZE,
BIT_DEPTH));
    PanicFalse(SourceConfigure(I2S_IN_1, STREAM_AUDIO_ SAMPLE_SIZE,
BIT_DEPTH));
    /* Synchronise sink and source*/
    PanicFalse(SinkSynchronise(I2S_OUT_0, I2S_OUT_1));
    PanicFalse(SourceSynchronise(I2S_IN_0, I2S_IN_1));
#ifdef BYPASS_KALIMBA
    /* I2S loopback without DSP */
    PanicFalse( StreamConnect(I2S_IN_0, I2S_OUT_0) );
    PanicFalse( StreamConnect(I2S_IN_1, I2S_OUT_1) );
#else
    /* Plug I2S in slot 0 into port 0 */
    PanicFalse( StreamConnect(I2S_IN_0, StreamKalimbaSink(0)) );
    /* Plug I2S in slot 1 into port 1 */
    PanicFalse( StreamConnect(I2S_IN_1, StreamKalimbaSink(1)) );
    /* Plug port 0 into I2S out slot 0 */
    PanicFalse( StreamConnect(StreamKalimbaSource(0), I2S_OUT_0) );
    /* Plug port 1 into I2S out slot 1 */
    PanicFalse( StreamConnect(StreamKalimbaSource(1), I2S_OUT_1) );
#endif
}
void led(void)
{
    /* LED indication */
    if(I2S_MASTER)
    {
        LedConfigure(LED_0, LED_ENABLE, 1);
    }
    else
    {
        LedConfigure(LED_1, LED_ENABLE, 1);
    }
}
```

B my_first_24bit_dsp_app kalimba DSP application code

The following Kalimba code is provided in the:

`<ADK Installation folder>\Examples\my_first_24bit_dsp_app folder`

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

```
//
*****************************************************************************
// Copyright (c) 2006 - 2016 Qualcomm Technologies International, Ltd.
//
****************************************************************************
//
****************************************************************************
// DESCRIPTION
// Basic example app for routing I2S audio through the DSP.
//
// NOTES
//
// What the code does:
//    Sets up cbuffers (circular connection buffers) for reading audio from
//    the I2S interface and routing back to the I2S interface.
//    Cbuffers are serviced by timer interrupts.
//    DC remove operator included defining the USE_DC_REMOVE_OPERATOR symbol.
//    Shift operator used instead of copy operator by defining the
//    USE_SHIFT_OPERATOR symbol.
//
//
****************************************************************************
#define $TMR_PERIOD_AUDIO_COPY         500
#define $AUDIO_CBUFFER_SIZE            512
#define $DATA_COPIED                   0
#define $DATA_NOT_COPIED               1
#define $AUDIO_BLOCK_SIZE              $AUDIO_CBUFFER_SIZE/2
// Application defines
// -----
#define AUDIO_24BIT                   // Enable 24 bit audio option
#define xUSE_DC_REMOVE_OPERATOR       // Enable DC removal operator
#define xUSE_SHIFT_OPERATOR           // Use shift operator instead of copy
```

```
operator
#ifdef AUDIO_24BIT
   #define $SHIFT_AMOUNT_IN    0      // No shift required for 24 Bit audio
   #define $SHIFT_AMOUNT_OUT  -1      // -6db output example (0 = no shift,
-8 = -48dB)
#else // AUDIO_24BIT
   #define $SHIFT_AMOUNT_IN    8      // Shift 8 required for 16 Bit audio
   #define $SHIFT_AMOUNT_OUT  -8      // Shift 8 required for 16 Bit audio
#endif // AUDIO_24BIT
// Standard includes
#include "core_library.h"
#include "cbops_library.h"
.MODULE $M.main;
   .CODESEGMENT PM;
   .DATASEGMENT DM;
   $main:
   // ** Setup ports that are to be used **
#ifdef AUDIO_24BIT
   .CONST  $AUDIO_LEFT_IN_PORT     (($cbuffer.READ_PORT_MASK |
$cbuffer.FORCE_24B_PCM_AUDIO ) + 0);
   .CONST  $AUDIO_RIGHT_IN_PORT    (($cbuffer.READ_PORT_MASK |
$cbuffer.FORCE_24B_PCM_AUDIO ) + 1);
   .CONST  $AUDIO_LEFT_OUT_PORT    (($cbuffer.WRITE_PORT_MASK |
$cbuffer.FORCE_24B_PCM_AUDIO ) + 0);
   .CONST  $AUDIO_RIGHT_OUT_PORT   (($cbuffer.WRITE_PORT_MASK |
$cbuffer.FORCE_24B_PCM_AUDIO ) + 1);
#else // AUDIO_24BIT
   .CONST  $AUDIO_LEFT_IN_PORT     (($cbuffer.READ_PORT_MASK |
$cbuffer.FORCE_PCM_AUDIO ) + 0);
   .CONST  $AUDIO_RIGHT_IN_PORT    (($cbuffer.READ_PORT_MASK |
$cbuffer.FORCE_PCM_AUDIO ) + 1);
   .CONST  $AUDIO_LEFT_OUT_PORT    (($cbuffer.WRITE_PORT_MASK |
$cbuffer.FORCE_PCM_AUDIO ) + 0);
   .CONST  $AUDIO_RIGHT_OUT_PORT   (($cbuffer.WRITE_PORT_MASK |
$cbuffer.FORCE_PCM_AUDIO ) + 1);
#endif // AUDIO_24BIT
   // ** 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 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:
   // ------
   // Use copy or shift operator to copy from port to audio in cbuffer.
   // DC offset removal optional.
   // ** Allocate memory for cbops stereo input copy routines **
   .VAR $audio_in_copy_struc[] =
#ifdef USE_SHIFT_OPERATOR
      &$audio_in_left_shift_op,        // First operator block
#else // USE_SHIFT_OPERATOR
      &$audio_in_left_copy_op,         // First operator block
#endif // USE_SHIFT_OPERATOR
      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
   // Shift operator blocks
   // ------
   .BLOCK $audio_in_left_shift_op;
      .VAR $audio_in_left_shift_op.next = &$audio_in_right_shift_op;
      .VAR $audio_in_left_shift_op.func = &$cbops.shift;
      .VAR $audio_in_left_shift_op.param[$cbops.shift.STRUC_SIZE] =
         0,                            // Input index
         2,                            // Output index
         $SHIFT_AMOUNT_IN;             // Shift amount
   .ENDBLOCK;
   .BLOCK $audio_in_right_shift_op;
#ifdef USE_DC_REMOVE_OPERATOR
      .VAR $audio_in_right_shift_op.next = &$audio_in_left_dc_remove_op;
#else // USE_DC_REMOVE_OPERATOR
      .VAR $audio_in_right_shift_op.next = $cbops.NO_MORE_OPERATORS;
```

```
#endif // USE_DC_REMOVE_OPERATOR
      .VAR $audio_in_right_shift_op.func = &$cbops.shift;
      .VAR $audio_in_right_shift_op.param[$cbops.shift.STRUC_SIZE] =
         1,                             // Input index
         3,                             // Output index
         $SHIFT_AMOUNT_IN;              // Shift amount
   .ENDBLOCK;
    // Copy operator blocks
    // ------
   .BLOCK $audio_in_left_copy_op;
      .VAR $audio_in_left_copy_op.next = &$audio_in_right_copy_op;
      .VAR $audio_in_left_copy_op.func = &$cbops.copy_op;
      .VAR $audio_in_left_copy_op.param[$cbops.copy_op.STRUC_SIZE] =
         0,                             // Input index
         2;                             // Output index
   .ENDBLOCK;
   .BLOCK $audio_in_right_copy_op;
#ifdef USE_DC_REMOVE_OPERATOR
      .VAR $audio_in_right_copy_op.next = &$audio_in_left_dc_remove_op;
#else // USE_DC_REMOVE_OPERATOR
      .VAR $audio_in_right_copy_op.next = $cbops.NO_MORE_OPERATORS;
#endif // USE_DC_REMOVE_OPERATOR
      .VAR $audio_in_right_copy_op.func = &$cbops.copy_op;
      .VAR $audio_in_right_copy_op.param[$cbops.copy_op.STRUC_SIZE] =
         1,                             // Input index
         3;                             // Output index
   .ENDBLOCK;
   // DC Remove operator block
   // ------
   .BLOCK $audio_in_left_dc_remove_op;
      .VAR $audio_in_left_dc_remove_op.next = &$audio_in_right_dc_remove_op;
      .VAR $audio_in_left_dc_remove_op.func = &$cbops.dc_remove;
      .VAR $audio_in_left_dc_remove_op.param[$cbops.dc_remove.STRUC_SIZE] =
         2,                             // Input index
         2,                             // Output index
         0;                             // DC estimate
   .ENDBLOCK;
   .BLOCK $audio_in_right_dc_remove_op;
      .VAR $audio_in_right_dc_remove_op.next = $cbops.NO_MORE_OPERATORS;
      .VAR $audio_in_right_dc_remove_op.func = &$cbops.dc_remove;
      .VAR $audio_in_right_dc_remove_op.param[$cbops.dc_remove.STRUC_SIZE] =
         3,                             // Input index
         3,                             // Output index
         0;                             // DC estimate
   .ENDBLOCK;
   // Output:
   // ------
```

```
   // Use copy or shift operator to copy from audio out cbuffer to port.
   // DC offset removal optional.
   // ** Allocate memory for cbops stereo output copy routines **
   .VAR $audio_out_copy_struc[] =
#ifdef USE_DC_REMOVE_OPERATOR
      &$audio_out_left_dc_remove_op,    // First operator block
#else // USE_DC_REMOVE_OPERATOR
   #ifdef USE_SHIFT_OPERATOR
      &$audio_out_left_shift_op,       // First operator block
   #else // USE_SHIFT_OPERATOR
      &$audio_out_left_op,             // First operator block
   #endif // USE_SHIFT_OPERATOR
#endif // USE_DC_REMOVE_OPERATOR
      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
   // DC removal operator blocks
   // ------
   .BLOCK $audio_out_left_dc_remove_op;
      .VAR $audio_out_left_dc_remove_op.next = &$audio_out_right_dc_remove_op;
      .VAR $audio_out_left_dc_remove_op.func = &$cbops.dc_remove;
      .VAR $audio_out_left_dc_remove_op.param[$cbops.dc_remove.STRUC_SIZE] =
         0,                            // Input index
         0,                            // Output index
         0;                            // DC estimate
   .ENDBLOCK;
   .BLOCK $audio_out_right_dc_remove_op;
#ifdef USE_SHIFT_OPERATOR
      .VAR $audio_out_right_dc_remove_op.next = &$audio_out_left_shift_op;
#else // USE_SHIFT_OPERATOR
      .VAR $audio_out_right_dc_remove_op.next = &$audio_out_left_op;
#endif // USE_SHIFT_OPERATOR
      .VAR $audio_out_right_dc_remove_op.func = &$cbops.dc_remove;
      .VAR $audio_out_right_dc_remove_op.param[$cbops.dc_remove.STRUC_SIZE] =
         0,                            // Input index
         0,                            // Output index
         0;                            // DC estimate
   .ENDBLOCK;
   // Shift operator blocks
   // ------
   .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
      $SHIFT_AMOUNT_OUT;            // 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
      $SHIFT_AMOUNT_OUT;            // Shift amount
.ENDBLOCK;
// Copy operator blocks
// ------
.BLOCK $audio_out_left_op;
   .VAR $audio_out_left_op.next = &$audio_out_right_op;
   .VAR $audio_out_left_op.func = &$cbops.copy_op;
   .VAR $audio_out_left_op.param[$cbops.copy_op.STRUC_SIZE] =
      0,                            // Input index.
      2;                            // Output index
.ENDBLOCK;
.BLOCK $audio_out_right_op;
   .VAR $audio_out_right_op.next = $cbops.NO_MORE_OPERATORS;
   .VAR $audio_out_right_op.func = &$cbops.copy_op;
   .VAR $audio_out_right_op.param[$cbops.copy_op.STRUC_SIZE] =
      1,                            // Input index
      3;                            // Output index
.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
// occurring 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 $cbuffer.calc_amount_data;
   // If the amount of data in the buffer is less than 32 bytes then a
   // buffer wrap around must have just occurred.
   Null = r0 - 32;
if POS jump wait_for_adc_buffer_wraparound;
// Start timer that copies input samples
r1 = &$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 $cbuffer.calc_amount_space;
   // If the amount of space in the buffer is less than 32 bytes then a
   // buffer wrap around must have just occurred.
   Null = r0 - 32;
if POS jump wait_for_dac_buffer_wraparound;
// Start timer that copies output samples
r1 = &$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 called 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 - $AUDIO_BLOCK_SIZE;
   if NEG jump dont_copy;
   r0 = &$audio_in_right_cbuffer_struc;
   call $cbuffer.calc_amount_data;
   Null = r0 - $AUDIO_BLOCK_SIZE;
   if NEG jump dont_copy;
   // Check if there is enough data in the output buffer
   r0 = &$audio_out_left_cbuffer_struc;
   call $cbuffer.calc_amount_space;
   Null = r0 - $AUDIO_BLOCK_SIZE;
   if NEG jump dont_copy;
   r0 = &$audio_out_right_cbuffer_struc;
   call $cbuffer.calc_amount_space;
   Null = r0 - $AUDIO_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;
```

Document references

Document	Reference
xIDE User Guide	80-CT405-1/CS-00101500-UG

Terms and definitions

Term	Definition
ADC	Analog to Digital Converter
ADK	Audio Development Kit
BC	Qualcomm® BlueCore™
BlueCore	Group term for the range of QTIL 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
I^2S	Inter IC Sound
IC	Integrated Circuit
MMU	Memory Management Unit
QTIL	Qualcomm Technologies International, Ltd.
VM	Virtual Machine
xIDE	BlueCore Integrated Development Environment