Application Note: AN00217

High Resolution Delay Example

This example demonstrates how to use the microphone array library with the high resolution delay lines to capture samples from the microphone array. The example is designed to show up to 8 channel array processing.

Required tools and libraries

The code in this application note is known to work on version 14.2.0 of the xTIMEcomposer tools suite, it may work on other versions.

The application depends on the following libraries:

- lib_mic_array

Required hardware

The example requires the XMOS microphone array reference design v1.

Prerequisites

- This document assumes familiarity with the XMOS xCORE architecture, the XMOS tool chain and the xC language. Documentation related to these aspects which are not specific to this application note are linked to in the references appendix.
- For descriptions of XMOS related terms found in this document please see the XMOS Glossary.
- The lib_mic_array user guide should be thoroughly read and understood.

1 Overview

This demo application shows how to interface to the microphone array in high resolution mode.

2 The high resolution delay example

The example in this note uses the microphone array library and shows a simple program that interfaces to the PDM microphones, decimates the PDM and finally packs it into frames.

To start using the microphone array library you need to add `lib_mic_array` to your `Makefile`:

```
USED_MODULES = ... lib_mic_array ...
```

You can then access the microphone array library functions in your source code via the `mic_array.h` header file:

```
#include "mic_array.h"
```

3 Demo Hardware Setup

1. Connect the xTAG-2 or xTAG-3 to Microphone Array Ref Design v1.
2. Connect the xTAG-2 or xTAG-3 USB connector to the USB connector on your development PC using a USB cable.
3. Connect the Microphone Array Ref Design v1 to your development PC using another USB cable.

4 Launching the demo application

Once the application source code is imported into the tools you can then build the project which will generate the binary file required to run the demo application.

Once the application has been built you need to download the application binary code onto the Microphone Array Ref Design v1. Here you use the tools to load the application over JTAG onto the xCORE multicore microcontroller.

Then click **Run**.

Alternatively, from the command line build the application, by doing a call to xmake, then run using `xrun app_high_resolution_delay_example.xe`. Nothing will happen.

5 Task setup

The PDM microphones interface task, high resolution delay task and the decimators have to be connected together and to the application (`example()`). There needs to be one `decimate_to_pcm_4ch()` task per four channels that need processing. There needs to be one `hires_delay` task per eight channels. The PDM interface task, `pdm_rx()` can process eight channels so only one is needed per eight channels. The PDM interface needs to be connected to the high resolution interface via two streaming channels and connected to the two decimators via streaming channels. Finally, the decimators have to be connected to the application. This gives the following task diagram:
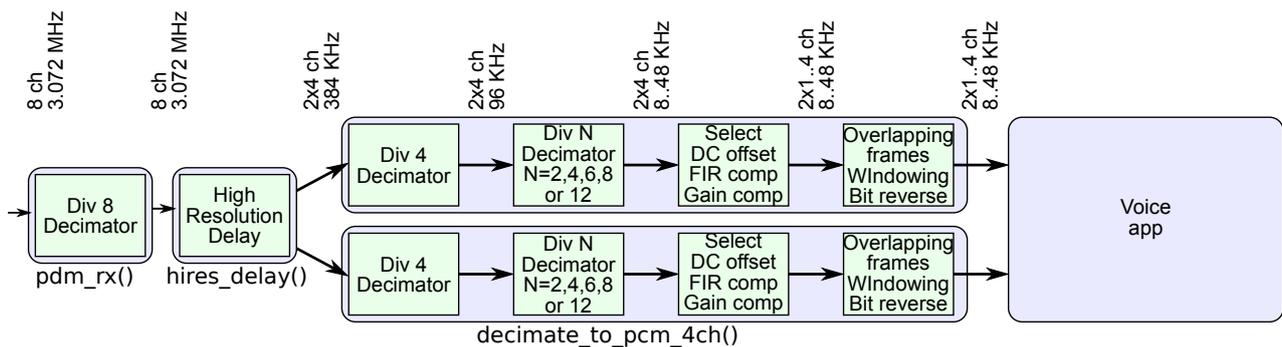

Figure 1: Task diagram of the example

Or represented in code as:

```
streaming chan c_pdm_to_hires[2];
streaming chan c_hires_to_dec[2];
streaming chan c_ds_output[2];
streaming chan c_cmd;

par {
    pdm_rx(p_pdm_mics, c_pdm_to_hires[0], c_pdm_to_hires[1]);
    hires_delay(c_pdm_to_hires, c_hires_to_dec, 2, c_cmd);
    decimate_to_pcm_4ch(c_hires_to_dec[0], c_ds_output[0]);
    decimate_to_pcm_4ch(c_hires_to_dec[1], c_ds_output[1]);
    example(c_ds_output, c_cmd);
}
```

Note that the decimators have to be on the same tile as the application due to shared frame memory. Finally, there needs to be a channel between the `hires_delay` and the application in order to issue the commands to change the taps on each delay line.

6 Frame memory

For each decimator an block of memory must be allocated for storing FIR data. The size of the data block must be:

```
Number of channels for that decimator * THIRD_STAGE_COEFS_PER_STAGE * Decimation factor * sizeof(int)
```

bytes. The data must also be double word aligned. For example:

```
int data_0[4*THIRD_STAGE_COEFS_PER_STAGE*DF] = {0};
int data_1[4*THIRD_STAGE_COEFS_PER_STAGE*DF] = {0};
```

Also the frame memory must also be a double word aligned array of length of at least 2. Note that on the xCORE-200 all global arrays are guaranteed to be double-word aligned.

7 Configuration

Configuration for the example is achieved through:

```
decimator_config_common dcc = {
        0, // Frame size log 2 is set to 0, i.e. one sample per channel will be present in each frame
        1, // DC offset elimination is turned on
        0, // Index bit reversal is off
        0, // No windowing function is being applied
        DF,// The decimation factor is set to 6
        g_third_stage_div_6_fir, // This corresponds to a 16kHz output hence this coef array is used
        0, // Gain compensation is turned off
        0, // FIR compensation is turned off
        DECIMATOR_NO_FRAME_OVERLAP, // Frame overlapping is turned off
        2  // There are 2 buffers in the audio array
};
decimator_config dc[2] = {
        {
                &dcc,
                data_0,     // The storage area for the output decimator
                {INT_MAX, INT_MAX, INT_MAX, INT_MAX},  // Microphone gain compensation (turned off)
                4           // Enabled channel count (currently must be 4)
        },
        {
                &dcc,
                data_1,     // The storage area for the output decimator
                {INT_MAX, INT_MAX, INT_MAX, INT_MAX}, // Microphone gain compensation (turned off)
                4           // Enabled channel count (currently must be 4)
        }
};
decimator_configure(c_ds_output, 2, dc);
```

All configuration options are enumerated in the Microphone array library. Once configured then the decimators require initialization via:

```
decimator_init_audio_frame(c_ds_output, 2, buffer, audio, dcc);
```

The the decimators will start presenting samples in the form of frames that can be accessed with:

```
frame_audio *current = decimator_get_next_audio_frame(c_ds_output, 2, buffer, audio, dcc);
```

The return value of the above is a pointer to the frame that the application (hires_DAS_fixed()) is allowed to access. The current structure contains the frame data in the data member. data is a 2D array with the first index denoting the channel number and the second index denoting the frame index. The frame index used 0 for the oldest samples and 2 to the power of dcc.frame_size_log2 minus one as the newest samples.

8 Updating the delays

The high resolution delay task is initially set to delay by a zero 384kHz clock cycles. Each channel can be delayed by up to HIRES_MAX_DELAY (default 32) samples at a sample rate of 384kHz. Setting of the taps is done through the function `hires_delay_set_taps();` which will do an atomic update of all the active delay lines tap positions.

9 References

XMOS Tools User Guide

http://www.xmos.com/published/xtimecomposer-user-guide

XMOS xCORE Programming Guide

http://www.xmos.com/published/xmos-programming-guide

XMOS Microphone Array Library

http://www.xmos.com/support/libraries/lib_mic_array

10 Full source code listing

10.1 Source code for app_high_resolution_delay_example.xc

```
// Copyright (c) 2016, XMOS Ltd, All rights reserved
#include <platform.h>
#include <xs1.h>

#include "mic_array.h"

on tile[0]: in port p_pdm_clk               = XS1_PORT_1E;
on tile[0]: in buffered port:32 p_pdm_mics  = XS1_PORT_8B;
on tile[0]: in port p_mclk                  = XS1_PORT_1F;
on tile[0]: clock pdmclk                    = XS1_CLKBLK_2;

// This sets the FIR decimation factor.
// Note that the coefficient array passed into dcc must match this.
#define DF 6

int data_0[4*THIRD_STAGE_COEFS_PER_STAGE*DF] = {0};
int data_1[4*THIRD_STAGE_COEFS_PER_STAGE*DF] = {0};
frame_audio audio[2];     //double buffered

void example(streaming chanend c_ds_output[2], streaming chanend c_cmd) {

    unsafe{
        unsigned buffer;

            decimator_config_common dcc = {
                    0, // Frame size log 2 is set to 0, i.e. one sample per channel will be present in each
                       ↪ frame
                    1, // DC offset elimination is turned on
                    0, // Index bit reversal is off
                    0, // No windowing function is being applied
                    DF,// The decimation factor is set to 6
                    g_third_stage_div_6_fir, // This corresponds to a 16kHz output hence this coef array is
                       ↪ used
                    0, // Gain compensation is turned off
                    0, // FIR compensation is turned off
                    DECIMATOR_NO_FRAME_OVERLAP, // Frame overlapping is turned off
                    2  // There are 2 buffers in the audio array
            };
            decimator_config dc[2] = {
                    {
                            &dcc,
                            data_0,      // The storage area for the output decimator
                            {INT_MAX, INT_MAX, INT_MAX, INT_MAX},  // Microphone gain compensation (turned off
                               ↪ )
                            4            // Enabled channel count (currently must be 4)
                    },
                    {
                            &dcc,
                            data_1,      // The storage area for the output decimator
                            {INT_MAX, INT_MAX, INT_MAX, INT_MAX}, // Microphone gain compensation (turned off)
                            4            // Enabled channel count (currently must be 4)
                    }
            };
            decimator_configure(c_ds_output, 2, dc);

        decimator_init_audio_frame(c_ds_output, 2, buffer, audio, dcc);

        while(1){
            frame_audio *current = decimator_get_next_audio_frame(c_ds_output, 2, buffer, audio, dcc);

            // Buffer and audio should never be accessed.

            int ch0_sample0 = current->data[0][0];
            int ch1_sample0 = current->data[1][0];

            // Update the delays. Delay values must be in range 0..HIRES_MAX_DELAY
            unsigned delays[7] = {0, 1, 2, 3, 4, 5, 6};
            hires_delay_set_taps(c_cmd, delays, 7);
```

```
        }
    }
}

int main() {

    par {
        on tile[0]: {
            configure_clock_src_divide(pdmclk, p_mclk, 4);
            configure_port_clock_output(p_pdm_clk, pdmclk);
            configure_in_port(p_pdm_mics, pdmclk);
            start_clock(pdmclk);

            streaming chan c_pdm_to_hires[2];
            streaming chan c_hires_to_dec[2];
            streaming chan c_ds_output[2];
            streaming chan c_cmd;

            par {
                pdm_rx(p_pdm_mics, c_pdm_to_hires[0], c_pdm_to_hires[1]);
                hires_delay(c_pdm_to_hires, c_hires_to_dec, 2, c_cmd);
                decimate_to_pcm_4ch(c_hires_to_dec[0], c_ds_output[0]);
                decimate_to_pcm_4ch(c_hires_to_dec[1], c_ds_output[1]);
                example(c_ds_output, c_cmd);
            }
        }
    }

    return 0;
}
```