Application Note: AN00220

Microphone array phase-aligned capture example

This example demonstrates how to use the microphone array library to capture samples from the microphone array and present them simultaneously with equal phase delay on each sample. 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 ??? 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 the minimum code to interface to the microphone array.

APPENDIX A - Demo Hardware Setup

Ensure that an XTAG is connected along with the microphone array via USB.

APPENDIX B - Launching the demo application

Build the application then run using `xrun app_phase_aligned_example.xe`.

APPENDIX C - Task setup

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

```
streaming chan c_pdm_to_dec[2];
streaming chan c_ds_output[2];

par{
      pdm_rx(p_pdm_mics, c_pdm_to_dec[0], c_pdm_to_dec[1]);
      decimate_to_pcm_4ch(c_pdm_to_dec[0], c_ds_output[0]);
      decimate_to_pcm_4ch(c_pdm_to_dec[1], c_ds_output[1]);
      example(c_ds_output);
}
```

Note that the decimators have to be on the same tile as the application due to shared frame memory.

APPENDIX D - 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.

APPENDIX E - 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_16kHz_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_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
                },
                {
                        &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
                }
        };
        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, DECIMATOR_NO_FRAME_OVERLAP);
```

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, 2);
```

The return value of the above is a pointer to the frame that the application (`example()`) 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.

APPENDIX F - 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

APPENDIX G - Full source code listing

G.1 Source code for main.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_1;

//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];

void example(streaming chanend c_ds_output[2]){

    unsigned buffer;
    unsafe{
        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_16kHz_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_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
                },
                {
                        &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
                }
        };
        decimator_configure(c_ds_output, 2, dc);
    }

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

    while(1){

        //The final argument is set to two to reflect that frame_audio audio[2]; is size 2 also.
        frame_audio *  current = decimator_get_next_audio_frame(c_ds_output, 2, buffer, audio, 2);

        //buffer and audio should never be accessed.

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


    }
}

int main(){
    par{
        on tile[0]:{
            streaming chan c_pdm_to_dec[2];
```

```
            streaming chan c_ds_output[2];

            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);

            par{
                pdm_rx(p_pdm_mics, c_pdm_to_dec[0], c_pdm_to_dec[1]);
                decimate_to_pcm_4ch(c_pdm_to_dec[0], c_ds_output[0]);
                decimate_to_pcm_4ch(c_pdm_to_dec[1], c_ds_output[1]);
                example(c_ds_output);
            }
        }
    }
    return 0;
}
```