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 14.1.1 of the xTIMEcomposer tools suite, it may work on other versions.

The application depends on the following libraries:

- lib_mic_array

Required hardware

This is an example only with no hardware requirements.

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.
- The `lib_mic_array` user guide should be thoroughly read and understood.
- For a description of XMOS related terms found in this document please see the XMOS Glossary[1].

[1] http://www.xmos.com/published/glossary

1 Overview

1.1 Introduction

This demo application shows the minimum code required to setup the microphone array. It outlines configuring the decimators and illustrates task structure.

1.2 Block diagram

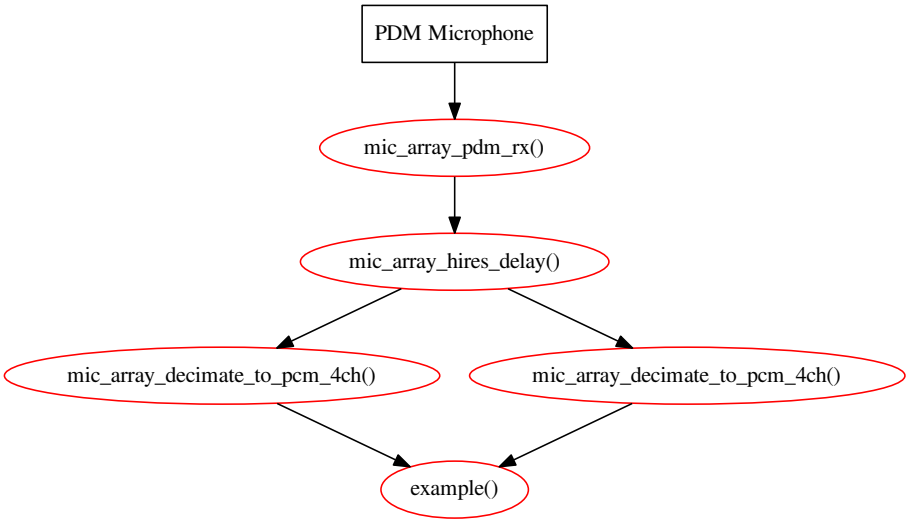

Figure 1: Application block diagram

2 How to use lib_mic_array

2.1 The Makefile

To start using the microphone array library with high resolution delay lines, you need to add `lib_mic_array` to you Makefile:

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

2.2 Includes

This application requires the system headers that defines XMOS xCORE specific defines for declaring and initialising hardware:

```
#include <platform.h>
#include <xs1.h>
#include <string.h>
```

The microphone array library functions are defined in `lib_mic_array.h`. This header must be included in your code to use the library. The support functions for the board are defined in `mic_array_board_support.h` and the logging functions are provided by `debug_print.h`.

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

2.3 Allocating hardware resources

A PDM microphone requires a clock and a data pin. For eight PDM mocrophones a single clock can be shared between all microphones and the data can be sampled on a single 8 bit port. On an xCORE the pins are controlled by `ports`. The application therefore declares one 1-bit port and one 8-bit port:

```
on tile[0]: in port p_pdm_clk               = XS1_PORT_1E;
on tile[0]: in buffered port:32 p_pdm_mics  = XS1_PORT_8B;
```

To generate the PDM clock a 24.576MHz master clock is divided by 8 using a clock block. These two hardware resources are declared with:

```
on tile[0]: in port p_mclk                  = XS1_PORT_1F;
on tile[0]: clock pdmclk                    = XS1_CLKBLK_1;
```

and are configured with:

```
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);
```

The result begin a 3.072MHz PDM clock is used for clocking the microphone data into the xCORE.

3 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 mic_array_decimate_to_pcm_4ch() task per four channels that need processing. There needs to be only one mic_array_hires_delay task for up to 16 channels. The PDM interface task, mic_array_pdm_rx() can process eight channels so only one is needed for this application. 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.

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

par {
    mic_array_pdm_rx(p_pdm_mics, c_pdm_to_hires[0], c_pdm_to_hires[1]);
    mic_array_hires_delay(c_pdm_to_hires, c_hires_to_dec, 2, c_cmd);
    mic_array_decimate_to_pcm_4ch(c_hires_to_dec[0], c_ds_output[0]);
    mic_array_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. Also, there needs to be a channel between the mic_array_hires_delay and the application in order to issue the commands to change the taps on each delay line.

4 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[DECIMATOR_COUNT*DECIMATOR_CH_COUNT]
         [THIRD_STAGE_COEFS_PER_STAGE*DECIMATION_FACTOR];
```

Note that on the xCORE-200 all global arrays are guaranteed to be double-word aligned.

5 Configuration

Configuration of the microphone array for the example is achieved through:

```
mic_array_decimator_conf_common_t 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
        DECIMATION_FACTOR,// 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
        FIR_COMPENSATOR_DIV_6, // FIR compensation is set to the corresponding coefficients
        DECIMATOR_NO_FRAME_OVERLAP, // Frame overlapping is turned off
        FRAME_BUFFER_COUNT  // The number of buffers in the audio array
};

mic_array_decimator_config_t dc[DECIMATOR_COUNT] = {
    {
            &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[4],     // 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)
    }
};
mic_array_decimator_configure(c_ds_output, DECIMATOR_COUNT, dc);
```

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

```
mic_array_init_time_domain_frame(c_ds_output, DECIMATOR_COUNT, buffer, audio, dc);
```

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

```
mic_array_frame_time_domain *  current =
                    mic_array_get_next_time_domain_frame(c_ds_output, DECIMATOR_COUNT, buffer, audio, dc);
```

The return value of `mic_array_get_next_time_domain_frame()` is a pointer to the frame that the application is allowed to access. The `current` frame 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 increasing indicies for newer samples.

5.1 Changing the sample rate

The sample rate chane be change easily with the example code by modifying:

```
#define DECIMATION_FACTOR    6    //Corresponds to a 16kHz output sample rate
```

The supported DECIMATION_FACTORS that come as stadard from `lib_mic_array` are 2, 4, 6, 8 and 12. These correspond to 48kHz, 24kHz, 16kHz, 12kHz, 8kHz and 6kHz. In order to change the define successfully you must also ensure that the coefficients to the decimators are correct for the selected decimation factor. The coefficients are declared in the header `fir_coefs.h` which is included in `mic_array.h`. The `coefs` member of `mic_array_decimator_config_common` must match the `output_decimation_factor` member. Also, the FIR compensation must be made to match. For example, to change to 24kHz output, the config should look like:

```
mic_array_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
            4,// The decimation factor is set to 6
            g_third_stage_div_4_fir, // This corresponds to a 24kHz output hence this coef array is used
            0, // Gain compensation is turned off
            FIR_COMPENSATOR_DIV_4, // FIR compensation is set to the corresponding coefficients
            DECIMATOR_NO_FRAME_OVERLAP, // Frame overlapping is turned off
            FRAME_BUFFER_COUNT  // The number of buffers in the audio array
};
```

5.2 Changing the frame buffer count

The number of frame buffer in use can be modifide through the define

```
#define FRAME_BUFFER_COUNT  2   //The minimum of 2 will suffice for this example
```

within this application. Increasing this would allow sample to be known for a longer period of time before the memory is resued for the present samples at the cost of increased memory usage.

5.3 Changing the frame size

The length of a frame is always a power of two. The maximum allowed at run time is given by two to the power of `MIC_ARRAY_MAX_FRAME_SIZE_LOG2`. At run time the length can be dynamically configured by setting the `frame_size_log2` member of `mic_array_decimator_config_common`.

5.4 Changing the buffering type

The buffering type can be either `DECIMATOR_NO_FRAME_OVERLAP` or `DECIMATOR_HALF_FRAME_OVERLAP`. This is configured through the `buffering_type` member of `mic_array_decimator_config_common`. In half frame overlap mode the rate that the `mic_array_get_next_time_domain_frame()` function must be serviced is doubled.

5.5 Disabling the DC offset elimination

The DC offset elimination can be disbaled by configuring the `mic_array_decimator_config_common` with 0 in the `apply_dc_offset_removal` member. Setting `apply_dc_offset_removal` to non-zero will enable it.

5.6 Enabling microphone gain compensation

To enable the microphone gain compensation first the `apply_mic_gain_compensation` member of `mic_array_decimator_config_common` must be set to non-zero. Then for each `mic_array_decimator_config` structure used to configure a 4 channel decimator the array member `mic_gain_compensation` must be set. If the gain a microphone i is g_i, then the compensation factor should be `INT_MAX` / min(g_i) for each microphone i.

6 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

7 Full source code listing

7.1 Source code for app_high_resolution_delay_example.xc

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

#define DECIMATION_FACTOR   6   //Corresponds to a 16kHz output sample rate
#define DECIMATOR_COUNT     2   //8 channels requires 2 decimators
#define FRAME_BUFFER_COUNT  2   //The minimum of 2 will suffice for this example

#define DECIMATOR_CH_COUNT 4    //Just to be clear

int data[DECIMATOR_COUNT*DECIMATOR_CH_COUNT]
         [THIRD_STAGE_COEFS_PER_STAGE*DECIMATION_FACTOR];

void example(streaming chanend c_ds_output[DECIMATOR_COUNT], streaming chanend c_cmd) {
    unsafe{
        mic_array_frame_time_domain audio[FRAME_BUFFER_COUNT];

        unsigned buffer;    //No need to initialize this.
        memset(data, 0, DECIMATOR_COUNT*DECIMATOR_CH_COUNT*
                THIRD_STAGE_COEFS_PER_STAGE*DECIMATION_FACTOR*sizeof(int));

        mic_array_decimator_conf_common_t 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
                DECIMATION_FACTOR,// 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
                FIR_COMPENSATOR_DIV_6, // FIR compensation is set to the corresponding coefficients
                DECIMATOR_NO_FRAME_OVERLAP, // Frame overlapping is turned off
                FRAME_BUFFER_COUNT  // The number of buffers in the audio array
        };

        mic_array_decimator_config_t dc[DECIMATOR_COUNT] = {
            {
                    &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[4],     // 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)
            }
        };
        mic_array_decimator_configure(c_ds_output, DECIMATOR_COUNT, dc);

        mic_array_init_time_domain_frame(c_ds_output, DECIMATOR_COUNT, buffer, audio, dc);

        while(1){
            mic_array_frame_time_domain *  current =
                            mic_array_get_next_time_domain_frame(c_ds_output, DECIMATOR_COUNT, buffer,
                              ↪ audio, dc);

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

```
    }
}

int main() {
    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[DECIMATOR_COUNT];
    streaming chan c_hires_to_dec[DECIMATOR_COUNT];
    streaming chan c_ds_output[DECIMATOR_COUNT];
    streaming chan c_cmd;

    par {
        mic_array_pdm_rx(p_pdm_mics, c_pdm_to_hires[0], c_pdm_to_hires[1]);
        mic_array_hires_delay(c_pdm_to_hires, c_hires_to_dec, 2, c_cmd);
        mic_array_decimate_to_pcm_4ch(c_hires_to_dec[0], c_ds_output[0]);
        mic_array_decimate_to_pcm_4ch(c_hires_to_dec[1], c_ds_output[1]);
        example(c_ds_output, c_cmd);
    }

    return 0;
}
```