Audio Application Examples Quickstart Guide

REV A

Publication Date: 2013/2/20

XMOS®

Table of Contents

1 Initial Setup and Biqud Filter Demo **4**
1.1 Hardware Setup . . . 4
1.2 Import and Build the Application . . . 5
1.3 Run the Application . . . 5
1.4 Look at the Code . . . 6

2 Short Reverb Demo **8**
2.1 Hardware Setup . . . 8
2.2 Import and Build the Application . . . 8
2.3 Run the Application . . . 8
2.4 Look at the Code . . . 8

3 Long Delay Demo **10**
3.1 Hardware Setup . . . 10
3.2 Import and Build the Application . . . 10
3.3 Run the Application . . . 10
3.4 Look at the Code . . . 10

4 Longer Reverb Demo **12**
4.1 Hardware Setup . . . 12
4.2 Import and Build the Application . . . 12
4.3 Run the Application . . . 12
4.4 Look at the Code . . . 13

This document covers the setup and execution of various demonstrations of real time audio processing. They all require the XA-SK-AUDIO Slice Card, and the last two also need the XA-SK-SDRAM external memory Slice Card.

1 Initial Setup and Biqud Filter Demo

IN THIS CHAPTER

- Hardware Setup
- Import and Build the Application
- Run the Application
- Look at the Code

This is a demonstration of a simple audio application that uses a small number of software modules and slice cards, to produce an audio equalisation effect.

1.1 Hardware Setup

The following hardware components are required:

- XP-SKC-L2 (Slicekit L2 Core Board)
- XA-SK-AUDIO (Audio board)
- XA-SK-XTAG2 (Slicekit XTAG adaptor)
- XTAG-2 (XTAG Connector board)

XP-SKC-L2 Slicekit Core board has four slots with edge conectors: `SQUARE`, `CIRCLE`, `TRIANGLE` and `STAR`, and one chain connector marked with a `CROSS`.

To setup up the system:

1. Connect the XA-SK-AUDIO Audio board to Slicekit Core board using the connector marked with the `CIRCLE`.
2. Connect the XTAG Adapter board to Slicekit Core board, using the chain connector marked with a `CROSS`.
3. Connect XTAG-2 board to the XTAG adapter.
4. Set the `XMOS LINK` to `OFF` on the XTAG Adapter.
5. Connect the XTAG-2 to host PC with a USB cable (not provided with the Slicekit starter kit).
6. Connect analogue audio source (e.g. microphone) to minijack In_1/2
7. Connect analogue audio monitor (e.g. headphones) to minijack Out_1/2
8. Connect D/C barrel jack of XMOS power supply to Slicekit Core board.

9. Switch on the power supply to the Slicekit Core board.

1.2 Import and Build the Application

1. Open xTIMEcomposer and check that it is operating in online mode. Open the edit perspective (Window->Open Perspective->XMOS Edit).

2. Locate the `'BiQuad Filter Slicekit Audio Demo'` item in the xSOFTip pane on the bottom left of the window, and drag it into the Project Explorer window in the xTIMEcomposer. This will also cause the modules on which this application depends to be imported as well. These modules are: module_dsp_biquad, module_dsp_utils, module_i2s_master and module_i2c_single_port.

3. Click on the app_slicekit_biquad item in the Explorer pane then click on the build icon (hammer) in xTIMEcomposer. Check the console window to verify that the application has built successfully.

For help in using xTIMEcomposer, try the xTIMEcomposer tutorial, that can be found by selecting Help->Tutorials from the xTIMEcomposer menu.

Note that the Developer Column in the xTIMEcomposer on the right hand side of your screen provides information on the xSOFTip components you are using. Select the module_dsp_biquad component in the Project Explorer, and you will see its description together with API documentation. Having done this, click the *back* icon until you return to this quickstart guide within the Developer Column.

1.3 Run the Application

Now that the application has been compiled, the next step is to run it on the Slicekit Core Board using the tools to load the application over JTAG (via the XTAG2 and Xtag Adaptor card) into the xCORE multicore microcontroller.

1. Supply some analogue audio to the audio board (via mini jack In_1/2) by connecting a PC or phone audio output. Now try playing the XMOS introductory video found here[1]. Alternatively connect and speak into a microphone if you have one.

2. Monitor the analogue audio from the audio board (via mini jack Out_1/2) by connecting some headphones, or sending the audio to an active speaker.

3. Click on the `Run` icon (the white arrow in the green circle). After a 1 or 2 seconds filtered audio should be heard. The output switches between effect and dry signals about every 8 seconds, so the listener can compare the 2 modes. The effect itself cycles through the following 4 filter types: [LO_PASS, HI_PASS, BAND_PASS, BAND_STOP]. The currently active effect is displayed in the debug console window.
   - LO_PASS produces audio with no treble (high frequencies)
   - HI_PASS produces audio with no bass (low frequencies)

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

- BAND_PASS produces audio with no treble or bass, only some mid-range frequencies
- BAND_STOP produces audio with some mid-range frequencies removed.

1.4 Look at the Code

Now that the application has been run with the default settings, you could try and adjust the filter configuration parameters. Such as the 'significant frequency' (e.g. cut-off frequency) and 'quality factor' (e.g. resonance). Note well, some combinations may produce overload (clipping distortion), in which case turn down the input volume.

1. Examine the application code. In xTIMEcomposer, navigate to the `src` directory under app_slicekit_biquad and double click on the `main.xc` file within it. The file will open in the central editor window.

2. Find the `main.xc` file and note that main() runs 2 cores (processes) in parallel connected by one channel. The processes are distributed over the tiles available on the Slicekit Core board.
   - AUDIO_IO_TILE handles the Analogue-to-Digital and Digital-to-Analogue conversion.
   - c_aud_dsp is a bi-directional channel transferring 32-bit audio samples.
   - DSP_TILE handles the Digital Signal Processing required to filter the audio samples.

3. Find the app_global.h header. At the top are the tile definitions. Note that on the Slicekit Core Board there are only 2 physical tiles 0 and 1. All cores are placed on the same tile (1).

4. Find the dsp_biquad.xc file. The function `dsp_biquad()` handles the DSP processing for the biquad filter. It communicates with the other parallel core via channel c_dsp. Data from these channels is buffered, and the buffers are passed to the `use_biquad_filter()` function for processing. `use_biquad_filter()` and `config_biquad_filter()` can be found in directory `module_dsp_biquad\src`. Finally, there is a finite-state-machine which switches the output between the dry and effect signals.

5. The BiQuad algorithm uses 6 multiples/sample. It is estimated that 24 multiples are possible at a sample rate of 48 kHz. This would allow 4 channels of audio to be processed simultaneously.

2 Short Reverb Demo

IN THIS CHAPTER

- Hardware Setup
- Import and Build the Application
- Run the Application
- Look at the Code

This is a demonstration of a complex audio application that uses a number of software modules to produce a short 'reverb-like' effect on an audio stream.

2.1 Hardware Setup

Leave the setup is unchanged from the Biquad application demo.

2.2 Import and Build the Application

1. Locate the `'Short-Reverb Slicekit Audio Demo'` item in the xSOFTip pane on the bottom left of the window, and drag it into the Project Explorer window in the xTIMEcomposer. This will also cause the modules on which this application depends to be imported as well.

2. Click on the app_slicekit_short_reverb item in the Explorer pane then click on the build icon (hammer) in xTIMEcomposer to build it as before.

2.3 Run the Application

1. If the previous application is still running, click on the red square button in the debug console to stop it.

2. Leaving the hardware setup unchanged Click on the `Run` icon and ensure the XMOS video is playing. After a 1 or 2 seconds the reverb effect should be heard. The output switches between effect and dry signals about every 8 seconds, so the listener can compare the 2 modes. Since this application only uses internal SRAM for audio buiffering, the reverb effect is of limited length.

2.4 Look at the Code

1. Find the app_global.h header. At the top are the tile definitions. Note that on the SliceKit Core Board there are only 2 physical tiles 0 and 1. To maximise the amount of memory available for the process (core) controlling the delay

(*dsp_control()*‘), it is given it’s own tile. All other cores are placed on the other tile.

2. Find the dsp_reverb.xc file. The function `dsp_control()` handles the DSP processing for the reverb. It communicates with the other 3 parallel cores, via 3 channels: c_aud_dsp, c_dsp_eq, and c_dsp_gain. Data from these channels is buffered, and the buffers are passed to the `use_reverb()` function for processing. `use_reverb()` and `config_reverb()` can be found in directory `module_dsp_short_reverb\src`. Finally, there is a finite-state-machine which switches the output between the dry and effect signals.

3 Long Delay Demo

IN THIS CHAPTER

- Hardware Setup
- Import and Build the Application
- Run the Application
- Look at the Code

This application introduces additional processing over and above the simple SDRAM based delay demo, using the BiQuad filter, long delay and Non-linear gain modules to create a proper reverberation effect.

3.1 Hardware Setup

1. Leave the setup unchanged from the Long Delay demo.

3.2 Import and Build the Application

1. Locate the `'Long-Reverb Slicekit Audio Demo'` item in the xSOFTip pane on the bottom left of the window, and drag it into the Project Explorer window in the xTIMEcomposer.

2. Click on the app_slicekit_long_reverb item in the Explorer pane then click on the build icon (hammer) in xTIMEcomposer.

3.3 Run the Application

1. Restart the audio source.

2. Click on the `Run` icon (the white arrow in the green circle). After a 1 or 2 seconds the reverb effect should be heard. The output switches between effect and dry signals about every 8 seconds, so the listener can compare the 2 modes.

3.4 Look at the Code

1. Examine the application code. In xTIMEcomposer, navigate to the `src` directory under app_slicekit_long_reverb and double click on the `main.xc` file within it. The file will open in the central editor window.

2. Find the `main.xc` file and note that main() runs 5 cores (processes) in parallel. These are distributed over the tiles available on the SliceKit Core board.

3. Find the app_global.h header. At the top are the tile definitions. Note that on the SliceKit Core Board there are only 2 physical tiles 0 and 1. Long-reverb is a time critical application. It is important that the SDRAM (Memory-slice) is connected to the same tile as the one running the delay functions (dsp_sdram_reverb).

4. Find the dsp_sdram_reverb.xc file. The function `dsp_sdram_reverb()` controls the DSP processing for the reverb. It communicates with the other 4 parallel cores, via 4 channels: c_aud_dsp, c_dsp_eq, c_dsp_gain, and c_dsp_sdram. Data from these channels is buffered, and the buffers are passed to the `use_sdram_reverb()` function for processing. `use_sdram_reverb()` and `config_sdram_reverb()` can be found in directory `module_dsp_long_reverb\src`. Finally, there is a finite-state-machine which switches the output between the dry and effect signals.

4 Longer Reverb Demo

IN THIS CHAPTER

- Hardware Setup
- Import and Build the Application
- Run the Application
- Look at the Code

This is a demonstration of an simple audio application that adds the XA-SK-SDRAM Slice Card to the existing setup using the XA-SK-AUDIO Slice Card, enabling larger amounts of audio buffering and hence more long lived and easily audible audio effects.

4.1 Hardware Setup

To setup up the system:

1. Turn off the power to the Slicekit Core Board.
2. Connect XA-SK-SDRAM Memory-Slice to the XP-SKC-L2 Slicekit Core board using the connector marked with the `SQUARE`.
3. Leave the rest of the setup unchanged from before.

4.2 Import and Build the Application

1. Locate the `'Long-Delay Slicekit Audio Demo'` item in the xSOFTip pane on the bottom left of the window, and drag it into the Project Explorer window in the xTIMEcomposer.
2. Click on the app_slicekit_long_delay item in the Explorer pane then click on the build icon (hammer) in xTIMEcomposer. Check the console window to verify that the application has built successfully. There will be quite a number of warnings that `bidirectional buffered port not supported in hardware`. These can be safely ignored for this component.

4.3 Run the Application

1. Restart the audio source.
2. Click on the `Run` icon (the white arrow in the green circle). After a 1 or 2 seconds the delay effect should be heard. The output switches between effect and dry signals, so the listener can compare the 2 modes: 16 seconds of the effect, followed by 8 seconds of the dry signal.

4.4 Look at the Code

1. Examine the application code. In xTIMEcomposer, navigate to the `src` directory under app_slicekit_long_delay and double click on the `main.xc` file within it. The file will open in the central editor window.

2. Find the `main.xc` file and note that main() runs 3 cores (processes) in parallel. These are distributed over the tiles available on the Slicekit Core board.

3. Find the app_global.h header. At the top are the tile definitions. Note that on the Slicekit Core Board there are only 2 physical tiles 0 and 1. Long-Delay is a time critical application. It is important that the SDRAM (Memory-slice) is connected to the same tile as the one running the delay functions (dsp_sdram_delay).

4. Find the dsp_sdram_delay.xc file. The function `dsp_sdram_delay()` controls the DSP processing for the delay. It communicates with the other 2 parallel cores, via 2 channels: c_aud_dsp and c_dsp_sdram. Data from these channels is buffered, and the buffers are passed to the `use_sdram_delay()` function for processing. `use_sdram_delay()` and `config_sdram_delay()` can be found in directory `module_dsp_long_delay\src`. Finally, there is a finite-state-machine which switches the output between the dry and effect signals.