



## lib\_board\_support: XMOS board support

Publication Date: 2026/2/3

Document Number: XM-015142-UG v1.5.0

## IN THIS DOCUMENT

1	Introduction	2
2	Supported Boards	3
2.1	xcore.ai Multi-Channel Audio Board	4
2.2	xcore-200 Multi-Channel Audio Board	8
2.3	xcore.ai Evaluation Kit	10
2.4	xcore-200 Evaluation Kit	13
2.5	xcore.ai Ethernet Development Kit	14
2.6	XK-VOICE-L71 Voice Reference Design Evaluation Kit	17
3	Usage	18
3.1	Using <b>lib_board_support</b>	18
4	Example Applications	19
4.1	Simple C Usage	19
4.2	XC Usage Example	19
4.3	Building the example	19
4.4	Running the example	20
5	Application Programmer Interface	21
5.1	Common API	21
5.2	XK_AUDIO_316_MC_AB API	22
5.3	XK_AUDIO_216_MC_AB API	26
5.4	XK_EVK_XU316 API	28
5.5	XK_EVK_XU216 API	30
5.6	XK_ETH_316_DUAL API	31
5.7	XK_VOICE_L71 API	33

---

## 1 Introduction

This repo contains board specific hardware configuration code for various *XMOS* evaluation and development kits. By keeping the board-specific code in a dedicated repository various applications need not replicate commonly used code such as initialisation of on-board peripherals and in addition any updates or fixes can easily be rolled out to all dependent applications.



## 2 Supported Boards

The following boards are supported by `lib_board_support` with interfaces provided in the languages shown in the table below.

Board	Supported Languages
XK_EVK_XU316	XC / C
XK_AUDIO_316_MC_AB	XC / C
XK_AUDIO_216_MC_AB	XC / C
XK_EVK_XE216	XC
XK_ETH_316_DUAL	XC
XK-VOICE-L71	XC / C

The following sections describe the features of each supported board.



## 2.1 xcore.ai Multi-Channel Audio Board

The XMOS *xcore.ai Multichannel Audio Board* (XK-AUDIO-316-MC) is a complete hardware and software reference platform targeted at up to 32-channel USB audio applications, such as DJ decks, mixers and other musical instrument interfaces. The board can also be used to prototype products with reduced feature sets or HiFi style products.

The XK-AUDIO-316-MC is based around the XU316-1024-TQ128-C24 multicore microcontroller; a dual-tile *xcore.ai* device with an integrated High Speed USB 2.0 PHY and 16 logical cores delivering up to 2400MIPS of deterministic and responsive processing power.

Exploiting the flexible programmability of the *xcore.ai* architecture, the XK-AUDIO-316-MC supports a USB audio source, streaming 8 analogue input and 8 analogue output audio channels simultaneously - at up to 192kHz. It also supports digital input/output streams (S/PDIF and ADAT) and MIDI. Ideal for consumer and professional USB audio interfaces. The board can also be used for testing general purpose audio DSP activities - mixing, filtering, etc.

For full details regarding the hardware please refer to [xcore.ai Multichannel Audio Platform Hardware Manual](#).

### Hardware Features

The location of the various features of the *xcore.ai Multichannel Audio Board* (XK-AUDIO-316-MC) is shown in [Fig. 1](#).

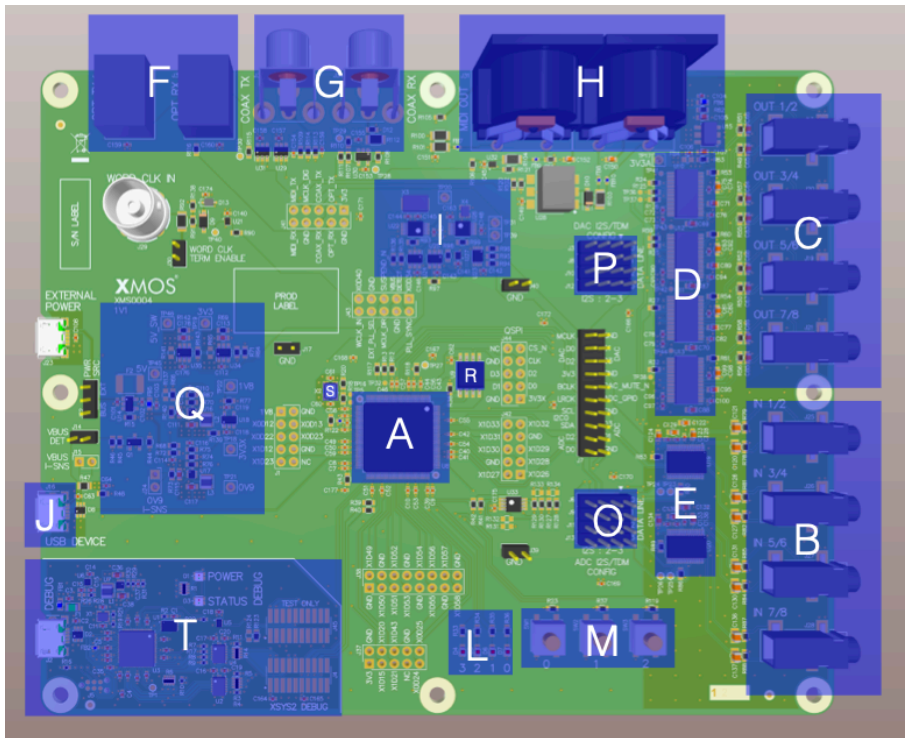


Fig. 1: xcore.ai Multichannel Audio Board hardware features



It includes the following features:

- ▶ A: *xcore.ai* (XU316-1024-TQ128-C24) device
- ▶ B: 8 line level analog inputs (3.5mm stereo jacks)
- ▶ C: 8 line level analog outputs (3.5mm stereo jacks)
- ▶ D: 384kHz 24 bit audio DACs
- ▶ E: 192kHz 24 bit audio ADCs
- ▶ F: Optical connections for digital interface (e.g. S/PDIF and ADAT)
- ▶ G: Coaxial connections for digital interfaces (e.g. S/PDIF)
- ▶ H: MIDI in and out connections
- ▶ I: Flexible audio master clock generation
- ▶ J: USB 2.0 micro-B jacks
- ▶ L: 4 general purpose LEDs
- ▶ M: 3 general purpose buttons
- ▶ O: Flexible I<sup>2</sup>S/TDM input data routing
- ▶ P: Flexible I<sup>2</sup>S/TDM output data routing
- ▶ Q: Integrated power supply
- ▶ R: Quad-SPI boot ROM
- ▶ S: 24MHz Crystal
- ▶ T: Integrated XTAG4 debugger

### Analogue Input & Output

A total of eight single-ended analog input channels are provided via 3.5mm stereo jacks. These inputs feed into a pair of quad-channel PCM1865 ADCs from Texas Instruments.

A total of eight single-ended analog output channels are provided. These are fed from four PCM5122 stereo DAC's from Texas instruments.

All ADC's and DAC's are configured via an I<sup>2</sup>C bus. Due to an clash of device addresses a I<sup>2</sup>C multiplexor is used.

The four digital I<sup>2</sup>S/TDM input and output channels are mapped to the xCORE input/outputs through a header array. These jumpers allow channel selection when the ADCs/-DACs are used in TDM mode.

### Digital Input & Output

Optical and coaxial digital audio transmitters are used to provide digital audio input output in formats such as IEC60958 consumer mode (S/PDIF) and ADAT. The output data streams from the *xcore* are re-clocked using the external master clock to synchronise the data into the audio clock domain. This is achieved using simple external D-type flip-flops.



## MIDI

MIDI input and output is provided on the board via standard 5-pin DIN connectors compliant to the MIDI specification. The signals are buffered using 5V line drivers and are then connected ports on the xCORE, via a 5V to 3.3V buffer. A 1-bit port is used for receive and a 4-bit port is used for transmit. A pull-up resistor on the MIDI output ensures there is no MIDI output when the *xcore* device is not actively driving the output.

## Audio Clocking

In order to accommodate a multitude of clocking options a flexible clocking scheme is provided for the audio subsystem.

Three methods of generating an audio master clock are provided on the board:

- ▶ A Cirrus Logic CS2100-CP PLL device. The CS2100 features both a clock generator and clock multiplier/jitter reduced clock frequency synthesizer (clean up) and can generate a low jitter audio clock based on a synchronisation signal provided by the *xcore*
- ▶ A Skyworks Si5351B PLL device. The Si5351 is an I<sup>2</sup>C configurable clock generator that is suited for replacing crystals, crystal oscillators, VCXOs, phase-locked loops (PLLs), and fanout buffers.
- ▶ *xcore.ai* devices are equipped with a secondary (or *application*) PLL which can be used to generate audio clocks.

Selecting between these methods is done via writing to bits 6 and 7 of PORT 8D on tile[0]. See [Control I/O](#).

### Note

**lib\_board\_support** currently only supports the *xcore.ai* secondary PLL and CS2100 device

## Control I/O

4 bits of PORT 8C are used to control external hardware on the board. This is described in [PORT 8C functionality](#).

Table 1: PORT 8C functionality

Bit(s)	Functionality	0	1
[0:3]	Unused		
4	Enable 3v3 power for digital (inverted)	Enabled	Disabled
5	Enable 3v3 power for analogue	Disabled	Enabled
6	PLL Select	CS2100	Si5351B
7	Master clock direction	Output	Input

### Note

To use the *xcore* application PLL bit 7 should be set to 0. To use one of the external PLLs bit 7 should be set to 1.



### LEDs, Buttons and Other IO

All programmable I/O on the board is configured for 3.3 volts.

Four green LED's and three push buttons are provided for general purpose user interfacing.

The LEDs are connected to PORT 4F and the buttons are connected to bits [0:2] of PORT 4E, both on tile 0. Bit 3 of this port is connected to the (currently unused) ADC interrupt line.

The board also includes support for an AES11 format Word Clock input via 75 ohm BNC. The software does not currently support any functionality related to this and it is provided for future expansion.

All spare I/O is brought out and made available on 0.1" headers for easy connection of expansion boards etc.

### Power

The board is capable of acting as a USB2.0 self or bus powered device. If bus powered, the board takes power from the **USB DEVICE** connector (micro-B receptacle). If self powered, board takes power from **EXTERNAL POWER** input (micro-B receptacle).

A power source select jumper (marked **PWR SRC**) is used to select between bus and self-powered configuration.

#### Note

To remain USB compliant the software should be properly configured for bus vs self powered operation

### Debug

For convenience the board includes an on-board xTAG4 for debugging via JTAG/xSCOPE. This is accessed via the USB (micro-B) receptacle marked **DEBUG**.



## 2.2 xcore-200 Multi-Channel Audio Board

The XMOS [xcore-200 Multi-channel Audio board](#) (XK-AUDIO-216-MC) is a complete hardware and reference software platform targeted at up to 32-channel USB and networked audio applications, such as DJ decks and mixers.

The XK-AUDIO-216-MC is based around the XE216-512-TQ128 multicore microcontroller; an dual-tile xcore-200 device with an integrated High Speed USB 2.0 PHY, RGMII (Gigabit Ethernet) interface and 16 logical cores delivering up to 2000MIPS of deterministic and responsive processing power.

Exploiting the flexible programmability of the *xcore-200* architecture, the XK-AUDIO-216-MC supports either USB or network audio source, streaming 8 analogue input and 8 analogue output audio channels simultaneously - at up to 192kHz.

For full details regarding the hardware please refer to [xcore-200 Multichannel Audio Platform Hardware Manual](#).

### Analogue Input & Output

A total of eight single-ended analog input channels are provided via 3.5mm stereo jacks. Each is fed into a CirrusLogic CS5368 ADC. Similarly a total of eight single-ended analog output channels are provided. Each is fed into a CirrusLogic CS4384 DAC.

The four digital I<sup>2</sup>S/TDM input and output channels are mapped to the *xcore* input/outputs through a header array. This jumper allows channel selection when the ADC/DAC is used in TDM mode

### Digital Input & Output

Optical and coaxial digital audio transmitters are used to provide digital audio input output in formats such as IEC60958 consumer mode (S/PDIF) and ADAT. The output data streams from the *xcore-200* are re-clocked using the external master clock to synchronise the data into the audio clock domain. This is achieved using simple external D-type flip-flops.

### MIDI

MIDI I/O is provided on the board via standard 5-pin DIN connectors. The signals are buffered using 5V line drivers and are then connected to 1-bit ports on the *xcore-200*, via a 5V to 3.3V buffer.

### Audio Clocking

A flexible clocking scheme is provided for both audio and other system services. In order to accommodate a multitude of clocking options, the low-jitter master clock is generated locally using a frequency multiplier PLL chip. The chip used is a Phaselink PL611-01, which is pre-programmed to provide a 24MHz clock from its CLK0 output, and either 24.576 MHz or 22.5792MHz from its CLK1 output.

The 24MHz fixed output is provided to the *xcore-200* device as the main processor clock. It also provides the reference clock to a Cirrus Logic CS2100, which provides a very low jitter audio clock from a synchronisation signal provided from the *xcore-200*.

Either the locally generated clock (from the PL611) or the recovered low jitter clock (from the CS2100) may be selected to clock the audio stages; the *xcore-200*, the ADC/DAC and Digital output stages. Selection is controlled via an additional I/O, bit 5 of PORT 8C.



### **LEDs, Buttons and Other IO**

An array of 4\*4 green LEDs, 3 buttons and a switch are provided for general purpose user interfacing. The LED array is driven by eight signals each controlling one of 4 rows and 4 columns.

A standard XMOS xSYS interface is provided to allow host debug of the board via JTAG.



### 2.3 xcore.ai Evaluation Kit

The XMOS *xcore.ai Evaluation Kit* (XK-EVK-XU316) is an evaluation board for the *xcore.ai* multi-core microcontroller from XMOS.

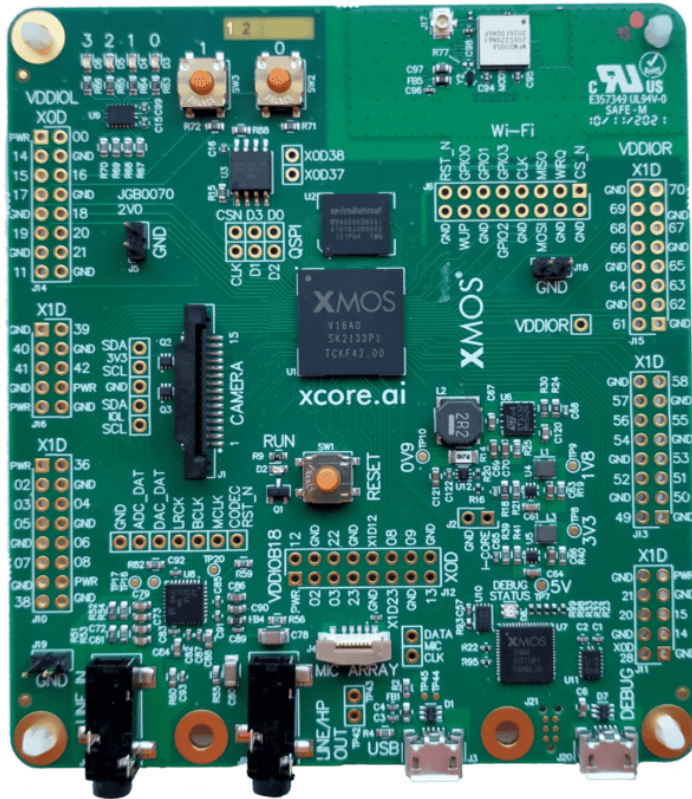


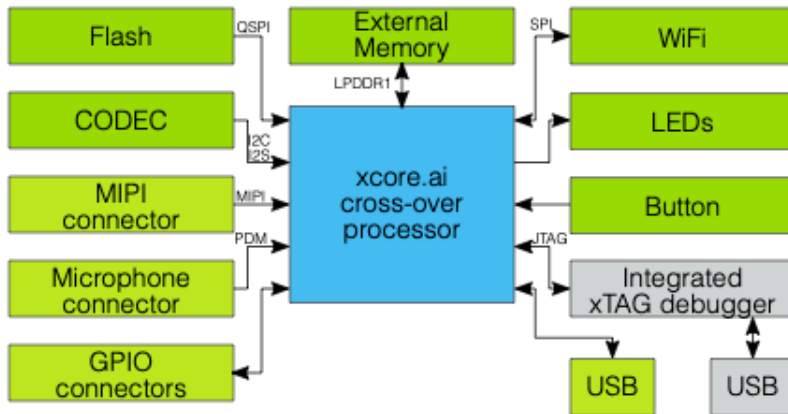
Fig. 2: *xcore.ai* Evaluation Kit

The XK-EVK-XU316 allows testing in multiple application scenarios and provides a good general software development board for simple tests and demos. The XK-EVK-XU316 comprises an *xcore.ai* processor with a set of I/O devices and connectors arranged around it, as shown in [Fig. 3](#).

External hardware features board include, four general purpose LEDs, two general purpose push-button switches, a PDM microphone connector, audio codec with line-in and line-out jack, QSPI flash memory, LPDDR1 external memory 58 GPIO connections from tile 0 and 1, micro USB for power and host connection, MIPI connector for a MIPI camera, integrated xTAG debug adapter and a reset switch with LED to indicate running.

For full details regarding the hardware please refer to [XK-EVK-XU316 xcore.ai Evaluation Kit Manual](#).



Fig. 3: *xcore.ai* Evaluation Kit block diagram

### **Warning**

The *xcore.ai Evaluation Kit* is a general purpose evaluation platform and should be considered an “example” rather than a fully fledged reference design.

### **Analogue Audio Input & Output**

A stereo CODEC (TLV320AIC3204), connected to the *xcore.ai* device via an I<sup>2</sup>S interface, provides analogue input/output functionality at line level.

The audio CODEC is configured by the *xcore.ai* device via an I<sup>2</sup>C bus.

### **Audio Clocking**

*xcore.ai* devices are equipped with a secondary (or *application*) PLL which is used to generate the audio clocks for the CODEC.

### **LEDs, Buttons and Other IO**

Four green LED’s and two push buttons are provided for general purpose user interfacing.

The LEDs are connected to PORT 4C and the buttons are connected to bits [0:1] of PORT 4D.

All spare I/O is brought out and made available on 0.1” headers for easy connection of expansion boards etc.

### **Power**

The XK-EVK-XU316 requires a 5V power source that is normally provided through the micro-USB cable J3. The voltage is converted by on-board regulators to the 0V9, 1V8 and 3V3 supplies used by the components.

The board should therefore be configured to present itself as a bus powered device when connected to an active USB host.



## Debug

For convenience the board includes an on-board xTAG4 for debugging via JTAG/xSCOPE. This is accessed via the USB (micro-B) receptacle marked **DEBUG**.



## 2.4 xcore-200 Evaluation Kit

The XCORE-200 Evaluation Kit features the XE216-512-TQ128 device with sixteen logical cores delivering up to 2000MIPS deterministically. The high-speed USB interface, 10/100/1000 Mbps Ethernet interface and 53 high-performance GPIO make it ideal for a wide range of applications, including networking and digital audio. The board also includes six servo interfaces for rapid prototyping of motor and motion control projects.

The kit also contains an XTAG debug adapter and is fully supported by the XTC Tools development environment.

For further information and detailed documentation see [XK-EVK-XE216](#)



## 2.5 xcore.ai Ethernet Development Kit

The *xcore.ai Ethernet Development Kit* (XK-ETH-316-DUAL) is an development board for the *xcore.ai* multi-core microcontroller from XMOS.

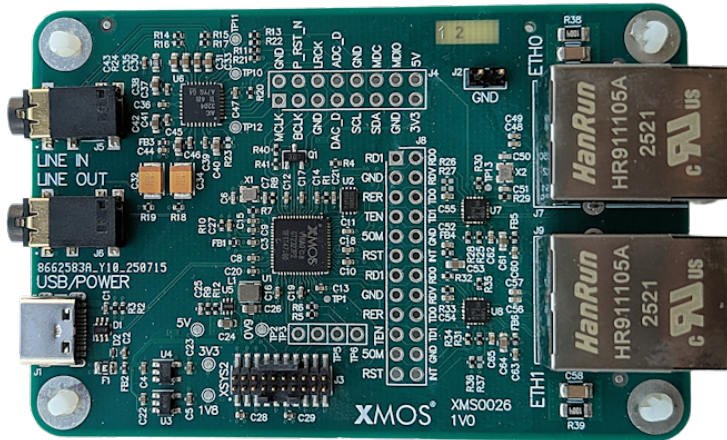


Fig. 4: *xcore.ai* Ethernet Development Kit

The XK-ETH-316-DUAL allows testing in multiple application scenarios and provides a good general software development board for simple tests and demos. The XK-ETH-316-DUAL comprises an *xcore.ai* processor with a set of I/O devices and connectors arranged around it, as shown in [Fig. 5](#).

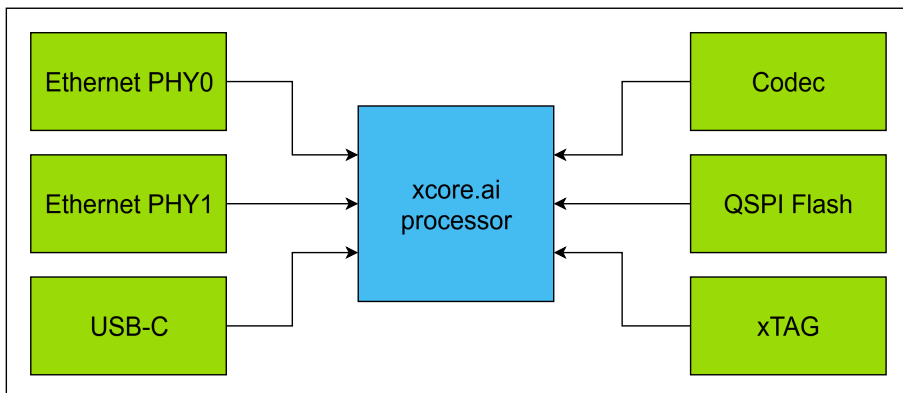


Fig. 5: *xcore.ai* Ethernet Development Kit block diagram

External hardware features board include, audio codec with line-in and line-out jack, QSPI flash memory and a USB-C connector for power and data.

The kit also contains an XTAG debug adapter and is fully supported by the XTC Tools development environment.

For full details regarding the hardware please refer to [XK-ETH-316-DUAL xcore.ai Ethernet Development Kit](#).

### Warning

The *xcore.ai Ethernet Development Kit* is a general purpose evaluation platform and should be considered an “example” rather than a fully fledged reference design.

## Hardware Features

The location of the various features of the *xcore.ai Ethernet Development Board* is shown in [Fig. 6](#).

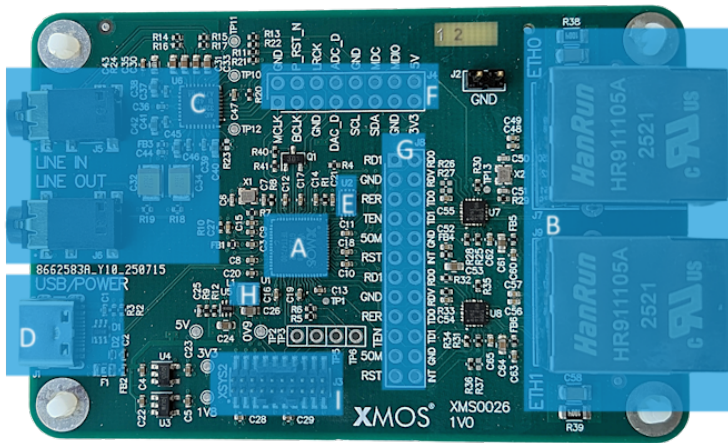


Fig. 6: *xcore.ai Ethernet Development Kit* hardware features

It includes the following features:

- ▶ A: *xcore.ai* (XU316-1024-QF60B-C24) device
- ▶ B: Dual 100 Base-T Ethernet ports
- ▶ C: Audio codec with line-in and line-out jacks



- ▶ D: USB-C jack
- ▶ E: Quad-SPI flash memory
- ▶ F: Audio signal breakout header 2.54mm (0.1")
- ▶ G: Ethernet signal breakout header 2.54mm (0.1")
- ▶ H: 25 MHz Crystal
- ▶ I: XTAG4 debugger connector

### Analogue Audio Input & Output

A stereo CODEC (TLV320AIC3204), connected to the *xcore.ai* device via an I<sup>2</sup>S interface, provides analogue input/output functionality at line level.

The audio CODEC is configured by the *xcore.ai* device via an I<sup>2</sup>C bus.

### Audio Clocking

*xcore.ai* devices are equipped with a secondary (or *application*) PLL which is used to generate the audio clocks for the CODEC.

### LEDs, Buttons and Other IO

Due to the development kit using the QFN60 package these are no IO allocated for LEDs or buttons.

The Ethernet IO signals are available on connector J8, and audio Codec IO signals are available on J4, both 0.1" headers for easy connection.

### Power

The *XK-ETH-316-DUAL* requires a 5V power source that is normally provided through the USB-C cable J1. The voltage is converted by on-board regulators to the 0V9, 1V8 and 3V3 supplies used by the components.

The board should therefore be configured to present itself as a bus powered device when connected to an active USB host.

### Debug

For convenience the kit includes an xTAG4 for debugging via JTAG/xSCOPE. The debugger connects via ribbon connector J3 (marked XSYS2). The debugger is accessed via the USB (micro-B) receptacle of the xTAG.



## 2.6 XK-VOICE-L71 Voice Reference Design Evaluation Kit

### OVERVIEW

The voice reference design board can be used as a Raspberry Pi HAT for integration with host based applications or used standalone as a USB accessory to a host system.

Features of the XK-VOICE-L71 include:

- ▶ XU316-1024-QF60A-C24 xcore.ai processor
- ▶ Raspberry Pi HAT connector
- ▶ 2 x Infineon IM69D130 MEMS mics
- ▶ 71mm inter-mic spacing
- ▶ Microphone mute switch
- ▶ Speaker output (Line level)
- ▶ USB / I2S host interface support
- ▶ Tri-colour LED
- ▶ Single push button

For further information and detailed documentation please follow this link [XK-VOICE-L71](#)

The API included supports the following features:

- ▶ Setting up of the DAC for either 16 kHz or 48 kHz. The DAC is always configured as I2S target (slave).
- ▶ The DAC may be configured to accept data from either I2S\_IN or I2S\_OUT signals on the board.
- ▶ Setting up of the Application PLL to drive a master clock at 12.288 MHz, 24.576 MHz or 49.152 MHz. Alternatively, it may be configured as an input to accept an external master clock.
- ▶ Setting up of the I2C expander which controls signal connection to the Raspberry Pi style header, allowing connection to external systems. MCLK, I2S, SPI and INT connections may be individually enabled or disabled.
- ▶ Provides a remote I2C master control task on tile[0] (I2S audio is on tile[1]) which may be exited once configuration is complete.



## 3 Usage

### 3.1 Using lib\_board\_support

**lib\_board\_support** is intended to be used with [XCommon CMake](#), the XMOS application build and dependency management system.

To use **lib\_board\_support** in an application, add it to the list of dependent modules in the application's *CMakeLists.txt* file and shown below. XMOS dependant modules should be pinned to release versions where possible, otherwise the latest commit on the *develop* branch will be used. The current release version of this library can be found on [XMOS Libraries](#)

```
set(APP_DEPENDENT_MODULES "lib_board_support")
```

#### Note

For further details on managing modules, pinning to a release version and other options, please see the page [xcommon-cmake Dependency Management](#).

The application must provide a relevant **xn** file or *target*. Example **xn** files are provided in this library (see *xn\_files* directory). If using an **xn** file, copy it into the application project and add the *CMake* configuration, example given for the *XK-AUDIO-316-MC*:

```
set(APP_HW_TARGET "xk-audio-316-mc.xn")
```

When using *XK-EVK-XU316* and *XK-EVK-XE216* boards, instead of an **xn** file, the *target* should be specified as the board name as shown below. As for general purpose evaluation boards the **xn** file provided with the XTC tools.

```
set(APP_HW_TARGET XK-EVK-XU316)
```

The application must use the APIs for its target board. To ensure only the correct sources are compiled, set the preprocessor symbol *BOARD\_SUPPORT\_BOARD* to one of the boards listed in *api/boards/boards\_utils.h*. This can be done in the application with the following *CMake* of configuration:

```
set(APP_COMPILER_FLAGS -DBOARD_SUPPORT_BOARD=XK_AUDIO_316_MC_AB)
```

From the application where board initialisation of configuration is done it is necessary to include the relevant header file. For example:

```
#include "xk_audio_316_mc_ab/board.h"
```

From then onwards the code may call the relevant API functions to setup and configure the board hardware. Examples are provided in the *examples* directory of this repo.

Note that in some cases, the *xcore* tile that calls the configuration function (usually from I<sup>2</sup>S initialisation) is different from the tile where I<sup>2</sup>C controller is placed. Since I<sup>2</sup>C controller is required by most audio CODECs for configuration and *xcore* tiles can only communicate with each other via channels, a remote server is needed to provide the I<sup>2</sup>C setup. This usually takes the form of a task which is run on a thread placed on the I<sup>2</sup>C tile and is controlled via a channel from the other tile where I<sup>2</sup>S resides. The cross-tile channel must be declared at the top-level XC main function. The included examples provide a reference for this using both XC and C.



## 4 Example Applications

Some simple example applications are provided in order to show how to use `lib_board_support`.

### 4.1 Simple C Usage

The applications `app_evk_316_simple_c` and `app_xk_audio_316_mc_simple_c` provide a bare-bones application where the hardware setup is called from C.

These applications run on the `XK-EVK-XU316` and `XK-AUDIO-316-MC` boards respectively.

They show how to use the cross-tile communications in conjunction with the I<sup>2</sup>C controller (master) server. The applications only setup the hardware and then exit the I<sup>2</sup>C server.

### 4.2 XC Usage Example

The application `app_xk_audio_316_mc_simple_xc` demonstrates calling the hardware setup API from C. It runs on the `XK-AUDIO-316-MC` board.

### 4.3 Building the example

This section assumes that the [XMOS XTC Tools](#) have been downloaded and installed. The required version is specified in the accompanying [README](#).

Installation instructions can be found [here](#).

Special attention should be paid to the section on [Installation of Required Third-Party Tools](#).

The application is built using the [xcommon-cmake](#) build system, which is provided with the XTC tools and is based on [CMake](#).

The `lib_board_support` software ZIP package should be downloaded and extracted to a chosen working directory.

To configure the build, the following commands should be run from an XTC command prompt. In the following command-line snippets `<app-name>` is used to refer to the `examples` application sub-folder selected to be built:

```
cd examples/<app-name>
cmake -G "Unix Makefiles" -B build
```

If any dependencies are missing they will be retrieved automatically during this step.

The application binaries can be built using `xmake`:

```
xmake -j -C build
```

Binary artifacts (.xe files) will be generated under the appropriate subdirectories of the `examples/<app-name>/bin` directory — one for each supported build configuration.

For subsequent builds, the `cmake` step may be omitted. If `CMakeLists.txt` or other build files are modified, `cmake` will be re-run automatically by `xmake` as needed.



## 4.4 Running the example

From an XTC command prompt, the following command should be run from the **examples/<app-name>** directory:

```
xrun --io bin/<app-name>.xe
```

Alternatively, the application can be programmed into flash memory for standalone execution:

```
xflash bin/<app-name>.xe
```

Full command-line process to build and run the *app\_xk\_audio\_316\_mc\_simple\_xc* example:

```
cd examples/app_xk_audio_316_mc_simple_xc
cmake -G "Unix Makefiles" -B build
xmake -C build
xrun --io bin/app_xk_audio_316_mc_simple_xc.xe
```



## 5 Application Programmer Interface

This section contains the details of the API support by *lib\_board\_support*. The API is broken down into 2 sections:

1. *Boards*: This includes subdirectories for each supported board which need to be included in the application.
2. *Drivers*: This includes sources for configuring peripheral devices which may be on one or more of the supported boards.

### 5.1 Common API

This section contains the list of supported boards, one of which needs to be globally defined as `BOARD_SUPPORT_BOARD` in the project.

#### **NULL\_BOARD**

Define representing Null board i.e. no board in use

#### **XK\_AUDIO\_216\_MC\_AB**

Define representing XK-AUDIO-216-MC, xcore-200 Multi-channel AudioBoard

#### **XK\_AUDIO\_316\_MC\_AB**

Define representing XK-AUDIO-316-MC, xcore.ai Multi-channel Audio Board

#### **XK\_EVK\_XU316**

Define representing XK-EVK-XU316, xcore.ai Explorer Evaluation Kit board

#### **XK\_EVK\_XE216**

Define representing XK-EVK-XU216, xcore-200 Explorer Evaluation Kit board

#### **XK\_ETH\_316\_DUAL**

Define representing XK-ETH-316-DUAL, xcore.ai Ethernet Development Kit board

#### **XK\_VOICE\_L71**

Define representing XK\_VOICE\_L71, xcore.ai Two Mic Development board

#### **BOARD\_SUPPORT\_N\_BOARDS**

Total number of boards supported by the library

#### **BOARD\_SUPPORT\_BOARD**

Define that should be set to the current board type in use

Default value: `NULL_BOARD`



## 5.2 XK\_AUDIO\_316\_MC\_AB API

struct **xk\_audio\_316\_mc\_ab\_config\_t**

Configuration struct type for setting the hardware profile.

### Public Members

*xk\_audio\_316\_mc\_ab\_mclk\_modes\_t* **clk\_mode**

See *xk\_audio\_316\_mc\_ab\_mclk\_modes\_t* for available clock mode options.

char **dac\_is\_clock\_master**

Boolean setting for whether the DAC or the xcore.ai is I2S clock master. Set to 0 to make the xcore.ai master.

unsigned **default\_mclk**

Nominal clock frequency in MHz. Standard rates are supported between 11.2896 MHz and 49.152 MHz.

unsigned **pll\_sync\_freq**

When the CLK\_CS2100 is used, this defines the nominal reference clock frequency for multiplication by the PLL. This value is ignored when the CS2100 is not used.

*xk\_audio\_316\_mc\_ab\_pcm\_format\_t* **pcm\_format**

See *xk\_audio\_316\_mc\_ab\_pcm\_format\_t* for available data frame options.

unsigned **i2s\_n\_bits**

Number of bits per data frame in I2S.

unsigned **i2s\_chans\_per\_frame**

This defines the number of audio channels per frame (a frame is a complete cycle of FSYNC or LRCLK).

enum **xk\_audio\_316\_mc\_ab\_mclk\_modes\_t**

Type of MCLK to be generated.

This may be:

- ▶ a fixed clock generated using the secondary PLL,
- ▶ an adjustable clock generated by the external CS2100 PLL,
- ▶ an adjustable clock generated using the secondary PLL, or
- ▶ no clock generated.

*Values:*

enumerator **CLK\_FIXED**

enumerator **CLK\_CS2100**

enumerator **CLK\_PLL**



enumerator **CLK\_NONE**

enum **xk\_audio\_316\_mc\_ab\_pcm\_format\_t**

Formats supported by the DAC and ADC. Either I2S using multiple data lines or TDM supporting multi-channel using a single data line.

Values:

enumerator **AUD\_316\_PCM\_FORMAT\_I2S**

enumerator **AUD\_316\_PCM\_FORMAT\_TDM**

enum **xk\_audio\_316\_mc\_ab\_xcore\_voltage\_t**

Voltage settings supported for the xcore core supply. Set by [xk\\_audio\\_316\\_mc\\_ab\\_core\\_voltage\\_set\(\)](#).

Values:

enumerator **AUD\_316\_XCORE\_VOLTAGE\_0\_925V**

enumerator **AUD\_316\_XCORE\_VOLTAGE\_0\_922V**

enumerator **AUD\_316\_XCORE\_VOLTAGE\_0\_9V**

enumerator **AUD\_316\_XCORE\_VOLTAGE\_0\_854V**

enumerator **AUD\_316\_XCORE\_VOLTAGE\_0\_85V**

port **p\_scl**

I2C interface ports

port **p\_sda**

void **xk\_audio\_316\_mc\_ab\_i2c\_master**(SERVER\_INTERFACE(i2c\_master\_if,  
i2c[1]))

Starts an I2C master task. Must be started from tile[0] after [xk\\_audio\\_316\\_mc\\_ab\\_board\\_setup\(\)](#) and before and tile[1] HW calls.

#### Parameters

- **i2c** – client side of I2C master interface connection.

void **xk\_audio\_316\_mc\_ab\_board\_setup**(  
const REFERENCE\_PARAM([xk\\_audio\\_316\\_mc\\_ab\\_config\\_t](#), config),  
)

Performs the required port operations to enable and the audio hardware on the platform. Must be called from tile[0] and before [xk\\_audio\\_316\\_mc\\_ab\\_AudioHwInit\(\)](#) is called.

#### Parameters



- ▶ **config** – Reference to the [xk\\_audio\\_316\\_mc\\_ab\\_config\\_t](#) configuration struct.

```
void xk_audio_316_mc_ab_core_voltage_set(
    const xk\_audio\_316\_mc\_ab\_xcore\_voltage\_t voltage_setting,
)
```

Allows control of the xcore.ai core voltage independantly. Warning - use with caution. This is only supported when the xcore is significantly clocked down. Please consult the datasheet for Operating Conditions / DC Characteristics. Must be called from tile[0].

#### Parameters

- ▶ **voltage\_setting** – See [xk\\_audio\\_316\\_mc\\_ab\\_xcore\\_voltage\\_t](#) for options

```
void xk_audio_316_mc_ab_AudioHwInit(
    CLIENT_INTERFACE(i2c_master_if, i2c), const REFERENCE_PARAM(xk\_audio\_316\_mc\_ab\_config\_t, config),
)
```

Initialises the audio hardware ready for a configuration. Must be called once *after* [xk\\_audio\\_316\\_mc\\_ab\\_board\\_setup\(\)](#).

#### Parameters

- ▶ **i2c** – Client side of I2C master interface connection.
- ▶ **config** – Reference to the [xk\\_audio\\_316\\_mc\\_ab\\_config\\_t](#) hardware configuration struct.

```
void xk_audio_316_mc_ab_AudioHwShutdown(CLIENT_INTERFACE(i2c_master_if, i2c))
```

Shuts down the audio hardware via I2C commands but keeps power rail on. Use [xk\\_audio\\_316\\_mc\\_ab\\_AudioHwShutdown\(\)](#) to remove power afterwards for minimum power.

```
void xk_audio_316_mc_ab_AudioHwPowerdown(void)
```

Powers down the audio hardware. Call [xk\\_audio\\_316\\_mc\\_ab\\_AudioHwShutdown\(\)](#) first to avoid clicks/pops [xk\\_audio\\_316\\_mc\\_ab\\_board\\_setup\(\)](#) and [xk\\_audio\\_316\\_mc\\_ab\\_AudioHwInit](#) must be called once *after* this before attempting to configure the hardware with [xk\\_audio\\_316\\_mc\\_ab\\_AudioHwConfig\(\)](#) again. Must be called from tile[0].

```
void xk_audio_316_mc_ab_AudioHwConfig(
    CLIENT_INTERFACE(i2c_master_if, i2c), const REFERENCE_PARAM(xk\_audio\_316\_mc\_ab\_config\_t, config), unsigned sam-
    Freq, unsigned mClk, unsigned dsdMode, unsigned sampRes_DAC, unsigned
    sampRes_ADC,
)
```

Configures the audio hardware following initialisation. This is typically called each time a sample rate or stream format change occurs.

#### Parameters

- ▶ **i2c** – Client side of I2C master interface connection.
- ▶ **config** – Reference to the [xk\\_audio\\_316\\_mc\\_ab\\_config\\_t](#) hardware configuration struct.
- ▶ **samFreq** – The sample rate in Hertz.
- ▶ **mClk** – The master clock rate in Hertz.



- ▶ **dsdMode** – Controls whether the DAC is to be set into DSD mode (1) or PCM mode (0).
- ▶ **sampRes\_DAC** – The sample resolution of the DAC output in bits. Typically 16, 24 or 32.
- ▶ **sampRes\_ADC** – The sample resolution of the ADC input in bits. Typically 16, 24 or 32.

```
void xk_audio_316_mc_ab_i2c_master_exit(CLIENT_INTERFACE(i2c_master_if,  
i2c)
```

Causes the tile[0] to exit, freeing up a thread. Must be called from tile[1]. Once called, HW config calls from tile[1] will block forever. It is possible to re-start [xk\\_audio\\_316\\_mc\\_ab\\_i2c\\_master\(\)](#) on tile[0] if needed to re-enable this service.

#### **Parameters**

- ▶ **i2c** – Client side of I2C master interface connection.



### 5.3 XK\_AUDIO\_216\_MC\_AB API

struct **xk\_audio\_216\_mc\_ab\_config\_t**

Configuration struct type for setting the hardware profile.

#### Public Members

*xk\_audio\_216\_mc\_ab\_clk\_mode\_t* **clk\_mode**

See *xk\_audio\_216\_mc\_ab\_clk\_mode\_t* for clock mode available options.

char **codec\_is\_clk\_master**

Boolean setting for whether the DAC or the xcore-200 is I2S clock master. Set to 0 to make the xcore-200 master.

*xk\_audio\_216\_mc\_ab\_usb\_sel\_t* **usb\_sel**

USB port selection - see *xk\_audio\_216\_mc\_ab\_usb\_sel\_t* for options.

*xk\_audio\_216\_mc\_ab\_pcm\_format\_t* **pcm\_format**

See *xk\_audio\_216\_mc\_ab\_pcm\_format\_t* for available *pcm\_format* options.

unsigned **pll\_sync\_freq**

When the external PLL is used, this defines the nominal reference clock frequency for multiplication by the PLL.

enum **xk\_audio\_216\_mc\_ab\_clk\_mode\_t**

Type of clock to be instantiated. This may be a fixed clock using an external generator or an adjustable clock using an external PLL (CS2100) in either digital Rx clock recovery or USB clock recovery using synchronous mode.

*Values:*

enumerator **AUD\_216\_CLK\_FIXED**

enumerator **AUD\_216\_CLK\_EXTERNAL\_PLL**

enumerator **AUD\_216\_CLK\_EXTERNAL\_PLL\_USB**

enum **xk\_audio\_216\_mc\_ab\_pcm\_format\_t**

Formats supported by the DAC and ADC. Either I2S using multiple data lines or TDM supporting multi-channel using a single data line.

*Values:*

enumerator **AUD\_216\_PCM\_FORMAT\_I2S**

enumerator **AUD\_216\_PCM\_FORMAT\_TDM**

enum **xk\_audio\_216\_mc\_ab\_usb\_sel\_t**

Selects which USB port to use - either type A or type B.

*Values:*



enumerator **AUD\_216\_USB\_A**

enumerator **AUD\_216\_USB\_B**

```
void xk_audio_216_mc_ab_AudioHwInit(  
    const REFERENCE_PARAM(xk_audio_216_mc_ab_config_t, config),  
)
```

Initialises the audio hardware ready for a configuration. Must be called once *after* *xk\_audio\_316\_mc\_ab\_board\_setup()*.

#### Parameters

- ▶ **config** – Reference to the *xk\_audio\_216\_mc\_ab\_config\_t* hardware configuration struct.

```
void xk_audio_216_mc_ab_AudioHwConfig(  
    const REFERENCE_PARAM(xk_audio_216_mc_ab_config_t, config), unsigned sam-  
    Freq, unsigned mClk, unsigned dsdMode, unsigned sampRes_DAC, unsigned  
    sampRes_ADC,  
)
```

Configures the audio hardware following initialisation. This is typically called each time a sample rate or stream format change occurs.

#### Parameters

- ▶ **config** – Reference to the *xk\_audio\_216\_mc\_ab\_config\_t* hardware configuration struct.
- ▶ **samFreq** – The sample rate in Hertz.
- ▶ **mClk** – The master clock rate in Hertz.
- ▶ **dsdMode** – Controls whether the DAC is to be set into DSD mode (1) or PCM mode (0).
- ▶ **sampRes\_DAC** – The sample resolution of the DAC output in bits. Typically 16, 24 or 32.
- ▶ **sampRes\_ADC** – The sample resolution of the ADC input in bits. Typically 16, 24 or 32.



## 5.4 XK\_EVK\_XU316 API

struct **xk\_evk\_xu316\_config\_t**

Configuration struct type for setting the hardware profile.

### Public Members

unsigned **default\_mclk**

Default MCLK frequency in Hz to generate using the secondary PLL at initialisation. Set to 0 to not generate MCLK; the application is responsible for providing MCLK in that case.

enum **audioHwCmd\_t**

Command enumeration for channel based commands to I2C master server on other tile.

Values:

enumerator **AUDIOHW\_CMD\_REGWR**

enumerator **AUDIOHW\_CMD\_REGRD**

enumerator **AUDIOHW\_CMD\_EXIT**

void **xk\_evk\_xu316\_AudioHwRemote**(chanend c)

Starts an I2C master server task. Must be started *before* the tile[1] `xk_evk_xu316_AudioHwInit` calls. In the background this also starts a combinable channel to interface translation task so the API may be used over a channel end however it still only occupies one thread. May be exited after config by sending `AUDIOHW_CMD_EXIT` if dynamic configuration is not required.

### Parameters

- ▶ **c** – Server side of channel connecting I2C master server and HW config functions.

void **xk\_evk\_xu316\_AudioHwChanInit**(chanend c)

Initialises the client side channel for remote communications with I2C. Must be called on tile[1] *before* `xk_evk_xu316_AudioHwInit()`.

### Parameters

- ▶ **c** – Client side of channel connecting I2C master server and HW config functions.

void **xk\_evk\_xu316\_AudioHwInit**(  
const REFERENCE\_PARAM(`xk_evk_xu316_config_t`, config),  
)

Initialises the audio hardware ready for a configuration. Must be called once *after* `xk_evk_xu316_AudioHwRemote()` and `xk_evk_xu316_AudioHwChanInit()`.

### Parameters

- ▶ **config** – Reference to the `xk_evk_xu316_config_t` hardware configuration struct.



```
void xk_evk_xu316_AudioHwConfig(  
    unsigned samFreq, unsigned mClk, unsigned dsdMode, unsigned sam-  
    pRes_DAC, unsigned sampRes_ADC,  
)
```

Configures the audio hardware following initialisation. This is typically called each time a sample rate or stream format change occurs.

#### Parameters

- ▶ **samFreq** – The sample rate in Hertz.
- ▶ **mClk** – The master clock rate in Hertz.
- ▶ **dsdMode** – Controls whether the DAC is to be set into DSD mode (1) or PCM mode (0).
- ▶ **sampRes\_DAC** – The sample resolution of the DAC output in bits. Typically 16, 24 or 32.
- ▶ **sampRes\_ADC** – The sample resolution of the ADC input in bits. Typically 16, 24 or 32.



## 5.5 XK\_EVK\_XU216 API

```
void xk_eth_xe216_phy_driver(  
    CLIENT_INTERFACE(smi_if, i_smi), CLIENT_INTERFACE(ethernet_cfg_if, i_eth),  
)
```

Task that connects to the SMI master and MAC to configure the ar8035 PHY and monitor the link status. Note this task is combinable (typically with SMI) and therefore does not need to take a whole thread. This task must be run from tile[1].

### Parameters

- ▶ **i\_smi** – Client register read/write interface
- ▶ **i\_eth** – Client MAC configuration interface



## 5.6 XK\_ETH\_316\_DUAL API

enum **port\_timing\_index\_t**

Index value used with `get_port_timings()` to refer to which PHY is in operation. The timings change according to which PHY is active in the hardware configuration of the dual PHY dev-kit.

Values:

enumerator **NULL\_PHY\_TIMINGS**

enumerator **PHY0\_PORT\_TIMINGS**

enumerator **PHY1\_PORT\_TIMINGS**

void **dual\_ethernet\_phy\_driver**(  
 CLIENT\_INTERFACE(smi\_if, i\_smi), NULLABLE\_CLIENT\_INTERFACE(ethernet\_cfg\_if,  
 i\_eth\_phy\_0), NULLABLE\_CLIENT\_INTERFACE(ethernet\_cfg\_if, i\_eth\_phy\_1),  
 )

Task that connects to the SMI master and MAC to configure the DP83825I PHYs and monitor the link status. Note this task is combinable (typically with SMI) and therefore does not need to take a whole thread.

### Note

It is not necessary to use both PHYs. If only one PHY is needed, the other should be set to **null**. PHY0 is the clock master so will always be configured regardless of which PHYs are in use.

### Parameters

- ▶ **i\_smi** – Client register read/write interface
- ▶ **i\_eth\_phy\_0** – Client MAC configuration interface for PHY\_0. Set to NULL if unused.
- ▶ **i\_eth\_phy\_1** – Client MAC configuration interface for PHY\_1. Set to NULL if unused.

void **reset\_eth\_phys**(void)

Sends hard reset to both PHYs. Both PHYs will be ready for SMI communication once this function has returned. This function must be called from Tile[1].

### Warning

This function will reset both PHYs and the audio codec. To reset one of these devices after start-up, use the `smi_phy_reset()` function to set the reset bit in PHY Basic Control register.

rmii\_port\_timing\_t **get\_port\_timings**(*port\_timing\_index\_t* phy\_idx)

Returns a timing struct tuned to the XK-ETH-316-DUAL hardware. This struct should be passed to the call to `rmii_ethernet_rt_mac()` and will ensure setup and hold times are maximised at the pin level of the PHY connection. `rmii_port_timing_t` is defined in `lib_ethernet`.



### Parameters

- ▶ **phy\_idx** – The index of the PHY to get timing data about.

### Returns

The timing struct to be passed to the PHY.



## 5.7 XK\_VOICE\_L71 API

struct **xk\_voice\_l71\_config\_t**

Configuration struct type for setting the hardware profile.

### Public Members

*xk\_voice\_l71\_mclk\_modes\_t* **clk\_mode**

*xk\_voice\_l71\_config\_t::clk\_mode* See *xk\_voice\_l71\_mclk\_modes\_t* for available clock mode options.

*xk\_voice\_l71\_rpi\_enable\_t* **oe\_enables**

*xk\_voice\_l71\_config\_t::oe\_enables* See *xk\_voice\_l71\_rpi\_enable\_t* for available clock mode options.

*xk\_voice\_l71\_dac\_pin\_t* **dac\_pin**

*xk\_voice\_l71\_config\_t::dac\_pin* See *xk\_voice\_l71\_dac\_pin\_t* for available clock mode options.

unsigned **default\_mclk**

The initial MCLK frequency in Hz to output before *xk\_voice\_l71\_AudioHwConfig()* is called

enum **xk\_voice\_l71\_mclk\_modes\_t**

Type of clock to be instantiated. This may be an external clock or an adjustable or fixed clock using the on-chip application PLL.

Values:

enumerator **CLK\_FIXED**

Generate fixed MCLK from XCORE using APP\_PLL

enumerator **CLK\_EXTERNAL**

Expect an externally provided MCLK. Note you will need to set ENABLE\_MCLK in *xk\_voice\_l71\_rpi\_enable\_t* during *xk\_voice\_l71\_AudioHwInit()*

enum **xk\_voice\_l71\_dac\_pin\_t**

Which DAC I2S data pin to use.

Values:

enumerator **DAC\_DIN\_PRI**

Use default DAC data pin (primary)

enumerator **DAC\_DIN\_SEC**

Use secondary DAC data pin (secondary)

enum **xk\_voice\_l71\_rpi\_enable\_t**

Which of the I2C expander enable lines to set. These can be ORed together. Setting these will connect the xcore chip to the raspberry PI expansion connector.

Values:



enumerator **ENABLE\_NO\_PINS**

All OEs disabled

enumerator **ENABLE\_MCLK**

enumerator **ENABLE\_SPI**

enumerator **ENABLE\_I2S**

enumerator **ENABLE\_INT**

void **xk\_voice\_171\_AudioHwRemote**(chanend c)

Starts an I2C master server task. Must be started on tile[0] *before* the tile[1] `xk_voice_171_AudioHwInit` calls. In the background this also starts a combinable channel to interface translation task so the API may be used over a channel end however it still only occupies one thread. May be exited after config by calling `xk_voice_171_AudioHwRemoteKill()` if dynamic configuration is not required.

#### Parameters

- ▶ **c** – Server side of channel connecting I2C master server and HW config functions.

void **xk\_voice\_171\_AudioHwChanInit**(chanend c)

Initialises the client side global channel end for remote communications with I2C. Must be called on tile[1] *before* `xk_voice_171_AudioHwInit()`.

#### Parameters

- ▶ **c** – Client side of channel connecting I2C master server and HW config functions.

void **xk\_voice\_171\_AudioHwInit**(  
const REFERENCE\_PARAM(`xk_voice_171_config_t`, config),  
)

Initialises the audio hardware ready for a configuration. Must be called once *after* `xk_voice_171_AudioHwRemote()` and `xk_voice_171_AudioHwChanInit()`.

#### Parameters

- ▶ **config** – Reference to the `xk_voice_171_config_t` hardware configuration struct.

void **xk\_voice\_171\_AudioHwConfig**(  
const REFERENCE\_PARAM(`xk_voice_171_config_t`, config), unsigned sample\_rate,  
unsigned mClk,  
)

Configures the audio hardware following initialisation. This is typically called each time a sample rate or stream format change occurs.

#### Parameters

- ▶ **config** – Reference to the `xk_voice_171_config_t` hardware configuration struct.
- ▶ **sample\_rate** – The sample rate in Hertz.
- ▶ **mClk** – The master clock rate in Hertz.



void **xk\_voice\_171\_AudioHwRemoteKill**(void)

Kills the remote I2C task. No further DAC or board config will be possible unless [xk\\_voice\\_171\\_AudioHwRemote\(\)](#) is restarted.

**Parameters**

- ▶ **c** – Server side of channel connecting I2C master server and HW config functions.





Copyright © 2026, All Rights Reserved.

---

XMOS Ltd. is the owner or licensee of this design, code, or Information (collectively, the "Information") and is providing it to you "AS IS" with no warranty of any kind, express or implied and shall have no liability in relation to its use. XMOS Ltd makes no representation that the Information, or any particular implementation thereof, is or will be free from any claims of infringement and again, shall have no liability in relation to any such claims.

XMOS, XCORE, VocalFusion and the XMOS logo are registered trademarks of XMOS Ltd. in the United Kingdom and other countries and may not be used without written permission. Company and product names mentioned in this document are the trademarks or registered trademarks of their respective owners.

