

AVB Reference Design Guide

(VERSION 0.9.0)



2009/02/16

Authors:

XMOS

Copyright © 2009, XMOS Ltd.
All Rights Reserved

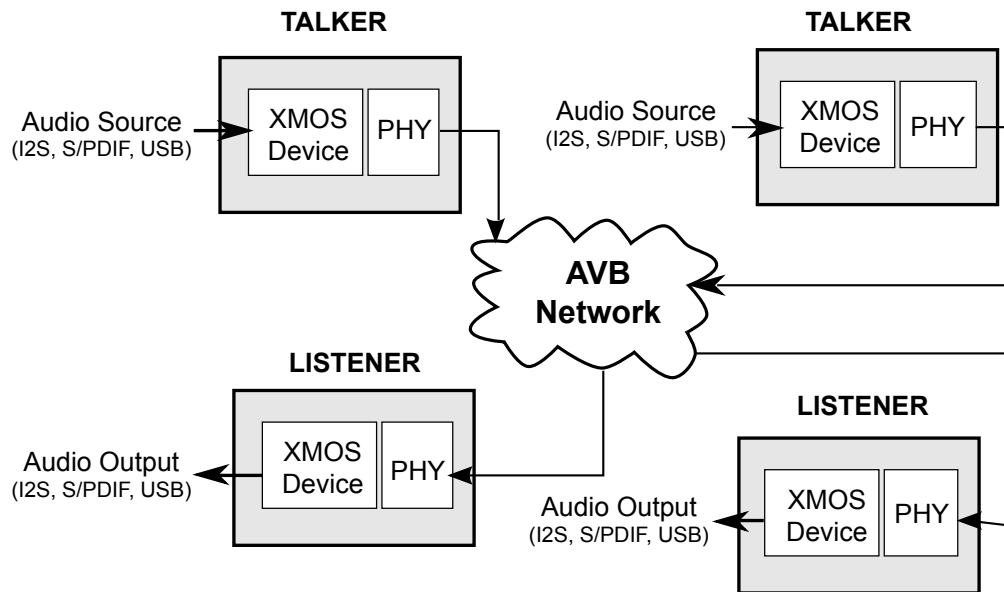
CONTENTS

1	Summary	3
2	AVB Reference Design Specification	4
3	Ethernet AVB Standards	5
3.1	802.1Qas	5
3.2	802.1Qav	5
3.3	802.1Qat	5
3.4	Emerging standards	6
3.4.1	IEEE P1722	6
3.4.2	IEC 61883-6	6
4	XS1 Architecture	7
4.1	XMOS XS1-G Programmable Devices	7
4.2	Software Libraries	8
4.3	XC Language	8
5	XMOS AVB Overview	9
5.1	Ethernet MAC Component	9
5.2	Precise Timing Protocol Component	10
5.3	Audio Stream Component	11
5.3.1	Audio Talker	12
5.3.2	Audio Listener	12
5.3.3	Audio clock recovery - hardware assistance	13
5.3.4	Audio clock recovery - digital resynchronisation	14
5.3.5	Audio Hardware Interface	15
5.4	Communication Between Components	15
6	System Design	17
6.1	Typical Configurations	18
6.1.1	Base System	18
6.1.2	Multiple Streams, Connected to External System	19
6.1.3	Multiple Device System	20
6.2	System Design, Implementation and Customisation	20

7	Development/Demonstration Platforms	23
7.1	Typical Development Setups	23
7.2	Building and running the demo	24
7.2.1	Debug output	26
8	System Reference	27
8.1	System Integration	27
8.2	Audio H/W Interface	27
8.2.1	Audio Component	28
8.3	Port Configuration	28
8.3.1	MAC port configuration	28
8.3.2	Audio port configuration	29
A	AVB Reference Design Software	31
A.1	Directory Structure	31
A.2	Important Reference Files	32
B	Licensing and Support Information	33
B.1	Reference Design License	33
B.2	Support	33
C	Further Reading	34
D	References	34

1 Summary

The XMOS Audio Visual Bridging (AVB) reference design can be used to stream synchronised audio over an ethernet network. The XMOS solution is based on software defined silicon programmable devices, which can transmit and receive multiple audio streams and implement both talker and listener functionality.



AVB Reference Design Features

- Supports all endpoint requirements: ethernet interface, packet processing, timing synchronisation, configuration/stream setup and media rate recovery can all be handled on device.
- Supports emerging AVB ethernet standards such as *IEEE 802.1as*, *IEEE 801.1Qav* and *IEEE P1722*.
- Flexible software based design implements both hardware interfaces and protocol layers in the same environment allowing, flexibility in system design and easy modification.
- Multiple audio hardware interfaces supported such as *I2S* and *S/PDIF*.

2 AVB Reference Design Specification

Functionality	
Provides ethernet interface, audio transport, precise timing protocol clock synchronisation and media clock recovery to streamed audio over ethernet.	
Supported Standards	
Ethernet	IEEE 802.3 (via MII)
AVB QoS	IEEE 802.1Qav
Precise Timing Protocol	IEEE 1588v2 or IEEE 802.1as
AVB Audio Over Ethernet	IEEE 1722
Audio Streaming	IEC 61883-6
Supported Devices	
XMOS Devices	XS1-G4
Resource Usage	
Thread Usage	12-14 depending on application
Memory Usage	110Kbytes
Required Ports	3×1-bit, 1×4-bit for each direction of ethernet MII layer Audio depends on CODEC(s)
Requirements	
Required Development Tools	XMOS Desktop Tools v9.2.0 or later
Ethernet	MI I compatible 100Mbit PHY
Audio	Audio input/output device (e.g. ADC/DAC audio CODEC)
Licensing and Support	
Reference code provided free of charge to XMOS customers for unrestricted use on XMOS devices only.	
Reference code is maintained by XMOS Limited.	

3 Ethernet AVB Standards

Ethernet AVB consists of a collection of different standards that together allow audio and video to be streamed over ethernet. The standards allow synchronised, uninterrupted streaming with multiple talkers and listeners on a switched network infrastructure.

3.1 802.1Qas

802.1Qas defines a precise timing protocol based on the *IEEE 1588v2* protocol. It allows every device connected to the network to share a common global clock. The protocol allows devices to have a synchronised view of this clock to within microseconds of each other, aiding media stream clock recovery and coordinated AVB traffic control.

This protocol is implemented in the XMOS AVB Reference Design.

3.2 802.1Qav

802.1Qav defines a standard for buffering and forwarding of traffic through the network using particular flow control algorithms. It uses the global clock provided by *802.1as* to synchronise traffic forwarding and gives predictable latency control on media streams going through the network.

The XMOS AVB solution implements the requirements for endpoints defined by *802.1av*. This is done by traffic flow control in the transmit arbiter of the ethernet MAC component.

3.3 802.1Qat

802.1Qat defines a stream reservation protocol that provides end-to-end reservation of bandwidth across an AVB network.

This protocol is not currently implemented in the AVB Reference Design. It could be implemented as part of the user application on an XMOS device or by a separate host processor communicating with an XMOS device.

3.4 Emerging standards

3.4.1 IEEE P1722

IEEE P1722 defines an encapsulation protocol to transport audio streams over ethernet. It is complementary to the AVB standards and in particular allows timestamping of a stream based on the *802.1as* global clock.

The XMOS AVB solution handles both transmission and receipt of audio streams using *IEEE P1722*. In addition it can use the *802.1as* timestamps to accurately recover the sample rate clock of the audio to match on the listener side.

3.4.2 IEC 61883-6

IEC 61883-6 defines an audio data format that is contained in *IEEE P1722* streams.

The XMOS AVB Reference Design uses *IEC 61883-6* to convey audio sample streams.

4 XS1 Architecture

The XS1 architecture consists of one or more processing cores, called XCores™, which have the following properties:

- Each XCore can execute up to eight threads concurrently, at a speed of up to 400 MIPS. Each thread has a dedicated register set enabling it to operate as a logical core.
- The eight threads share a single 64 KByte unified memory with no access collisions.
- Integer and fixed point operations are provided for efficient DSP and cryptographic operations.
- 64 I/O general purpose pins are provided, which can be programmed from software. Thread execution is deterministic and hence each thread can implement a hard real-time I/O task, regardless of the behaviour of other threads.
- I/O pins are grouped into logical ports of width 1, 4, 8, 16 and 32 bits. Each port incorporates serialisation/deserialisation, synchronisation with the external interface and precision timing.
- Each XCore incorporates eight timers that measure time relative to a 100 MHz reference clock.

The architecture is designed to support standard programming languages such as C. Extensions to standard languages, libraries, or the use of assembly language provide access to the full benefits of the instruction set.

4.1 XMOS XS1-G Programmable Devices

The XS1-G family is the first available implementation of the XS1 architecture. It includes the XS1-G4 device that integrates four XCore processors. Each device is connected to a high performance switch interconnect via four internal links. Each link is capable of transferring data at 800 Mbits/second. The switch provides full connectivity between the cores on the programmable device, and also

provides up to sixteen external links. Each external link is capable of transferring data at up to 400 Mbits/second.

Other members of the family include the XS1-G1 and XS1-G2. XS1 devices can be used standalone, or can be connected together using links to create a network of XCore processors.

4.2 Software Libraries

The XS1 architecture implements hardware as software, so that hardware solutions are just software libraries. These software solutions are compiled and linked like ordinary C code into a binary file which can be loaded and executed on the device.

The AVB Reference Design is a library of source files that can be included in a larger application (templates and demos are provided with the reference design). The application can then be compiled and deployed onto a device to provide your AVB hardware solution.

4.3 XC Language

The XMOS originated XC language [1] is based upon C. XC is a concurrent and realtime programming language designed to target the XS1-G architecture. XC programs are easy to write and debug—free from deadlocks, race conditions and memory violations—and can be compiled to produce high performance multicore designs.

XC uses channels to provide high-speed bidirectional communication and synchronisation between channel ends within threads on the same processor, a different processor on the same chip or a different chip.

5 XMOS AVB Overview

An XMOS AVB Reference Design consists of three main components:

- The ethernet MAC and MII
- The precise timing engine (PTP)
- Audio streaming components

These three components along with a user component for configuration and final application implementation make up an XMOS AVB solution.

5.1 Ethernet MAC Component

The MAC component, which provides ethernet connectivity to the AVB solution, includes a software MAC and MII interface. To use the component, an MII physical interface must be attached to the XCore ports. The component provides a robust error-checking interface for a 100Mbps connection (see Figure 1).

The component consists of six threads that must be run on the same core. These threads handle both the receiving and transmission of ethernet frames. The MAC component can be linked (via channels) to other components/threads in the system. Each link can set a filter to control which packets are conveyed to it via that channel.

All configuration of the channel is managed by a client C API, which configures and registers the filters. Details of the API used to configure MAC channels can be found in the header files in the `src/mac/client/` directory within the reference design software.

The MAC component supports two features that are necessary to implement upcoming AVB standards with precise timing and quality constraints.

- *Timestamping* - allows receipt and transmission of ethernet frames to be timestamped with respect to a clock (*i.e* a 100MHz reference clock can provide a resolution of 10ns).

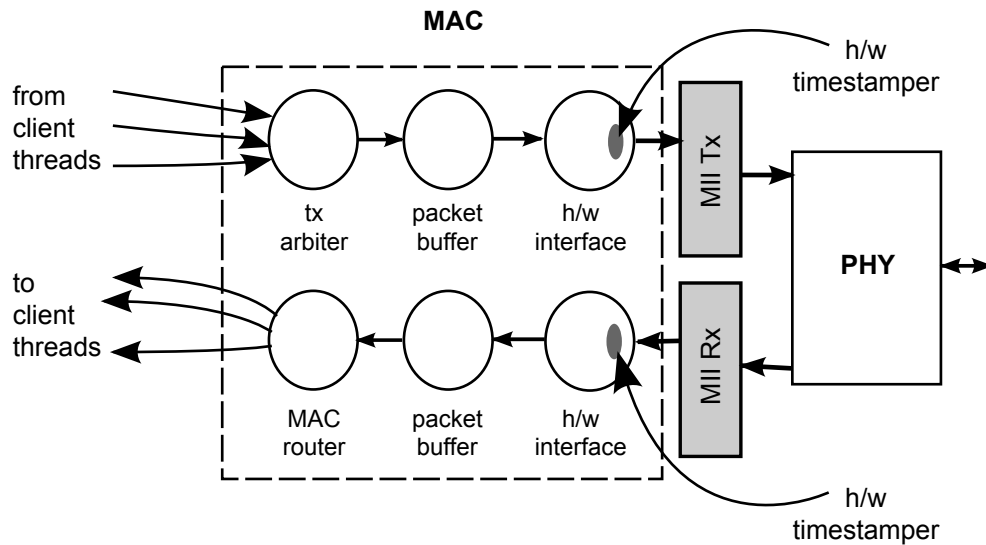


Figure 1 MAC component thread diagram

- *Flow control* - allows different channels to have different priorities and bandwidth restrictions to allow steady flow of outgoing media stream packets. The implementation provides flow control to satisfy the requirements of an AVB endpoint as specified in the upcoming 802.1Qav standard.

5.2 Precise Timing Protocol Component

The precise timing protocol component (PTP) provides a system with a notion of global time on a network. The component supports the *IEEE 1588v2* timing protocol and the upcoming *AVB 802.1as* timing protocol. It allows synchronisation of the presentation and playback rate of media streams across a network (see Figure 2).

The timing component, which consists of two threads, connects to the ethernet MAC component and provides channel ends for clients to query for timing information. The component interprets PTP packets from the MAC and maintains a notion of global time. The maintenance of global time requires no application interaction with the component.

The PTP component can be configured at runtime to be a grandmaster or slave.

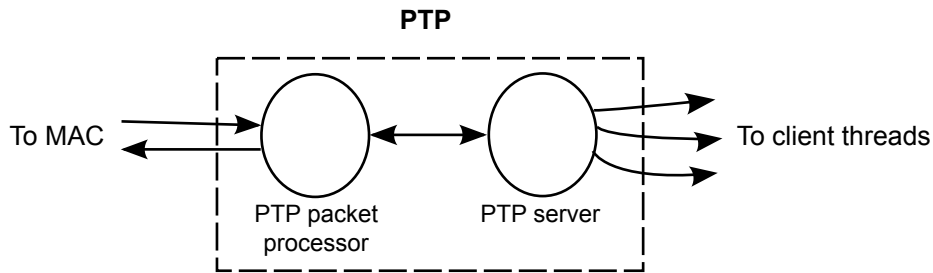


Figure 2 *PTP component thread diagram*

If the component is configured as a grandmaster, it supplies a clock source to the network. If the network has several grandmasters, the potential grandmasters negotiate between themselves to select a single grandmaster. Once a single grandmaster is selected, all units on the network synchronise a global time from this source and the other grandmasters stop providing timing information. Depending on the intermediate network, this synchronisation can be to sub-microsecond level resolution.

Client threads connect to the timing component via channels. The relationship between the local reference counter and global time is maintained across this channel, so that a client can timestamp with a local timer very accurately and then convert it to global time, giving highly accurate global timestamps.

Client threads can communicate with the server using the API described in the file `src/ptp/client/ptp_client.h`.

5.3 Audio Stream Component

The audio stream component connects to the MAC component and performs one of two functions:

- Samples an audio stream from an audio device and transmits it over ethernet (talker functionality).
- Samples an audio stream from ethernet and conveys it to an audio device (listener functionality).

Audio samples are split into packets and sent or received over ethernet using the *IEEE 1722* protocol (see Figure 3).

5.3.1 Audio Talker

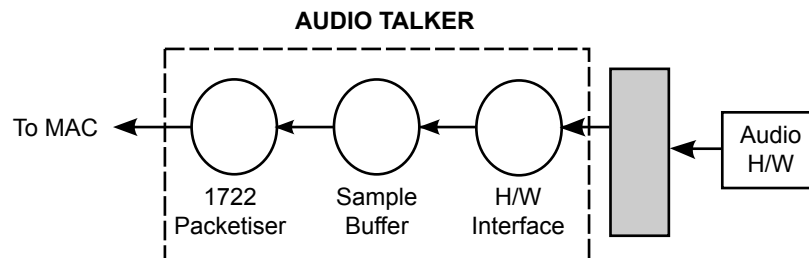


Figure 3 *Audio talker thread diagram*

The talker component consists of two threads, an *IEEE P1722* packetiser, an audio sample buffer and an audio hardware interface. The sample buffer connects to the audio hardware interface and timestamps samples on entry to the buffer. The packetiser removes the samples and splits them into *IEEE P1722* ethernet packets to be broadcast via the MAC component.

When the packets are created the timestamps are converted to the time domain of the global clock provided by the PTP component, and a fixed offset is added to the timestamps to provide the *presentation time* of the samples (*i.e* the time at which the sample should be played by a listener).

5.3.2 Audio Listener

The audio listener component takes *IEEE P1722* packets from the MAC and converts them into a sample stream to be fed into a buffer thread.

Since the audio stream will have been created by a system on a different oscillator to the one on the listener, there will be a difference in the audio clocks that determine the sample rate of the stream. This difference has to be accounted for by the listener component. To this end, there is an audio clock recovery thread that uses the following pieces of information:

- The current rate of the audio hardware interface clock (provided over a channel from the audio hardware interface thread).
- The rate of the incoming samples (provided by the *IEEE P1722* timestamps).
- The current PTP clock.

These three pieces of information allow the clock recovery thread to determine the ratio of the incoming rate to the local hardware rate and therefore the amount of adjustment required. It is noted that the difference in rate should not be large depending on the oscillator difference. A typical rate difference for 48Khz audio is up to 10 samples per second for typical oscillators.

5.3.3 Audio clock recovery - hardware assistance

Figure 4 shows the component layout for an audio listener when using hardware assisted audio clock recovery.

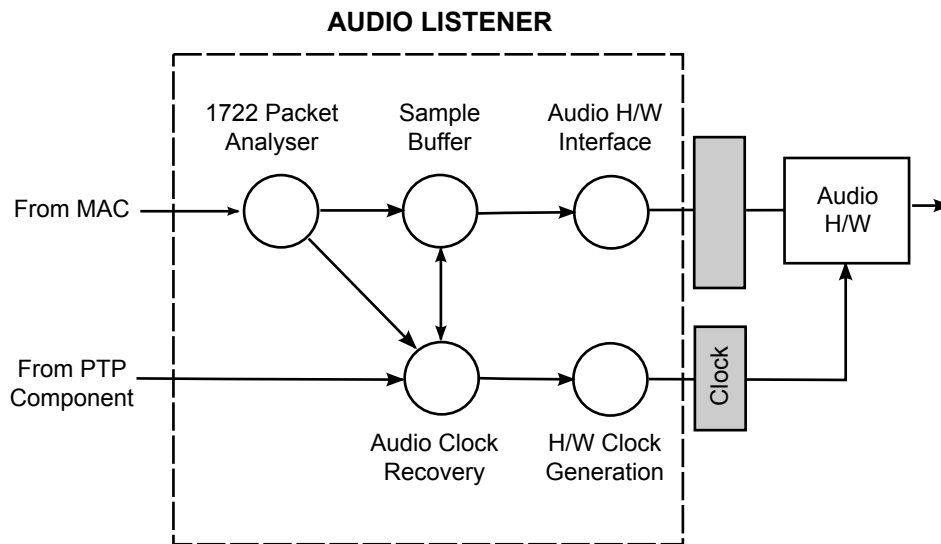


Figure 4 Listener thread diagram (h/w assisted audio clock recovery)

Hardware assisted clock recovery requires the audio hardware interface to support varying of the sample rate clock to meet the incoming stream. The clock

recovery thread calculates the length of time per sample of the incoming stream and the ratio of the incoming sample to the current local sample rate. It then communicates this information to a hardware clock generation thread that alters the hardware clock to match the incoming rate. With the rates matched, the listener can automatically set an appropriate amount of buffering to ensure that samples are played at their presentation time, allowing the audio output to be synchronised precisely with other devices on the AVB network with a very high degree of accuracy.

The hardware clock generation can take several forms depending on the hardware interface including:

- The audio hardware has a clock recovery module that can recover an accurate audio system clock from an approximate word clock. In this case the hardware thread outputs the word clock.
- The audio hardware has a system clock that is controlled by a voltage controlled oscillator (VCXO). In this case the hardware thread outputs adjustments to the VCXO.
- The audio hardware takes an *I2S* signal with the audio hardware running in slave mode. The XMOS device then generates the word clock and bit clock based on the incoming sample rate. *This means that the word clock of the audio device will not be synchronised to the system clock of the audio device. It is important that you are sure the audio device can handle this to use this method.*

5.3.4 Audio clock recovery - digital resynchronisation

Figure 5 shows the component layout of an audio listener component using digital resynchronisation in the XMOS device to provide audio clock recovery.

Using this method the clock recovery thread calculates the ratio of the incoming sample rate to the local hardware sample rate. It then communicates this information to an asynchronous rate controller thread. This thread takes a sample stream from the audio sample buffer at the incoming rate and outputs a stream to the audio hardware at the outgoing rate. The number of samples may vary between input to the thread and output. This is achieved using a digital filter to interpolate samples in the stream.

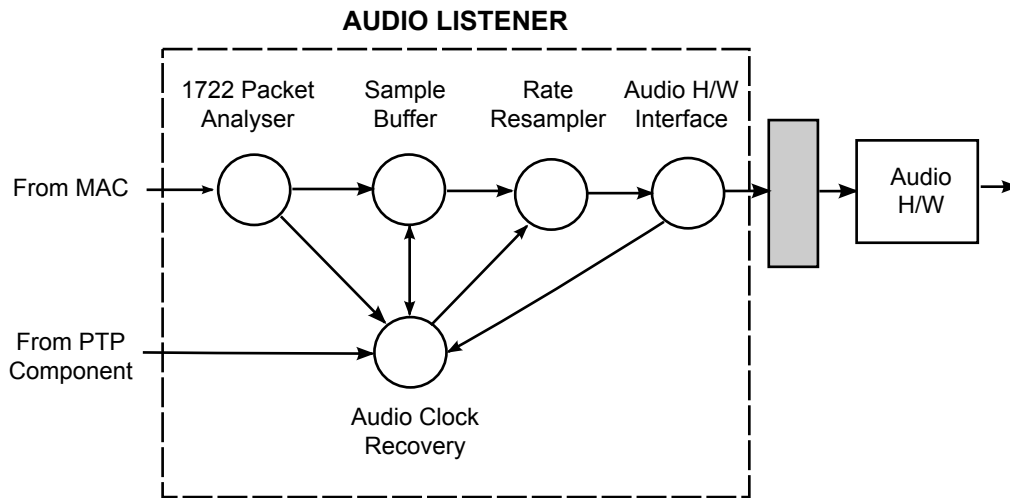


Figure 5 Listener thread diagram (software audio clock recovery)

5.3.5 Audio Hardware Interface

The audio hardware interface that the audio component connects to can be configured to specific hardware. Details on implemented hardware interfaces can be found in Section 8.2. It is important to use the correct clock recovery method for the specific hardware.

The source file for each hardware interface documents the compatible clock recovery strategies.

5.4 Communication Between Components

The components are connected together using XC communication channels on the processor. The channels are integrated in both the hardware and software and can be used by calling XC functions with a common channel argument (see Figure 6).

Once the components are connected, communication is made by a client software API provided by the component. Using these links and APIs one can easily design an AVB system on a programmable device and modify it to fit the end application requirements. Full details on system integration of the components can be found in Section 6.2. All functionality is handled in software and including any

```
par { // execute i parallel
    ptp_server(ptp_chan0, ..
    mac_server(mac_chan0, ..
    ...
    avb_1722_stream(ptp_chan0, ..
                    mac_chan0, ..
}
```

Figure 6 *Using channels to link components*

hardware interfaces required between components.

6 System Design

Figure 7 shows the generic layout of an AVB audio streaming system implementation. The XMOS system could be a single XMOS programmable device (e.g. XS1-G4) or multiple XMOS devices connected by links.

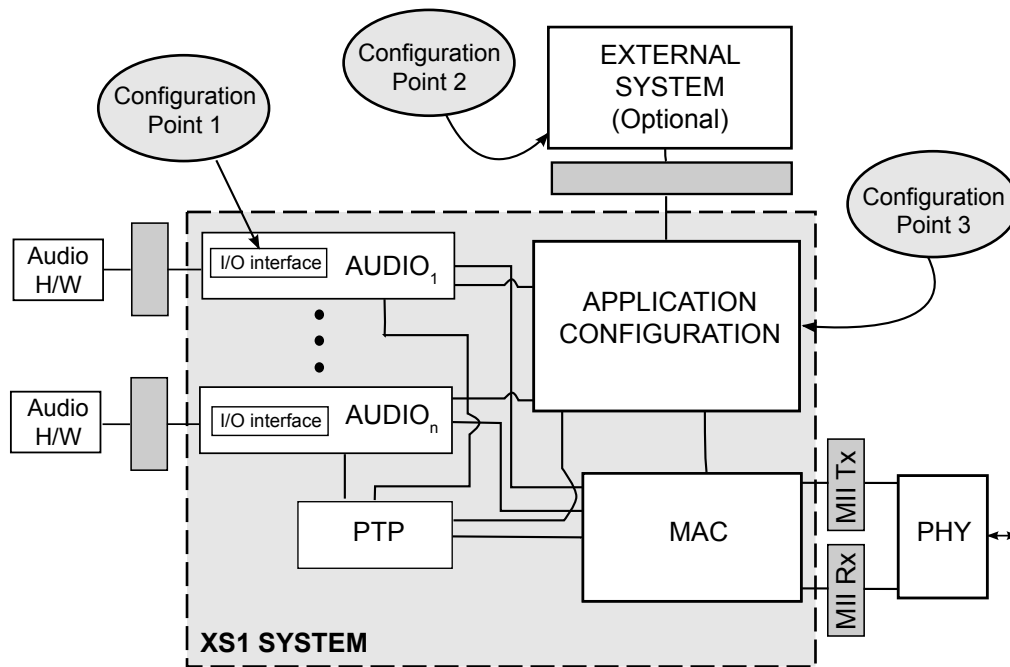


Figure 7 AVB system layout

A separate instantiation of an audio component is required for each stream and hardware interface, but each stream may be multi-channel (e.g. stereo) if the hardware interface provides this capability (see Section 5.3.1).

The resource requirement for each component is described in the following table:

Component	Threads	Memory
MAC	6	20k
PTP	2	37k
Audio Interface	5	20k

Note: The threads for each component must be placed on the same core. The component resource usage can be compared to resources available on specific devices:

Device	Threads	Memory
XS1-G4	32	256k

This table shows that for an XS1-G4, up to 60% of the device’s resources are available for the application threads, which may include network configuration protocols, network service negotiation/connection, user interface code and audio DSP code.

6.1 Typical Configurations

The generic AVB design can be configured in many different ways depending on the required capability of the application.

6.1.1 Base System

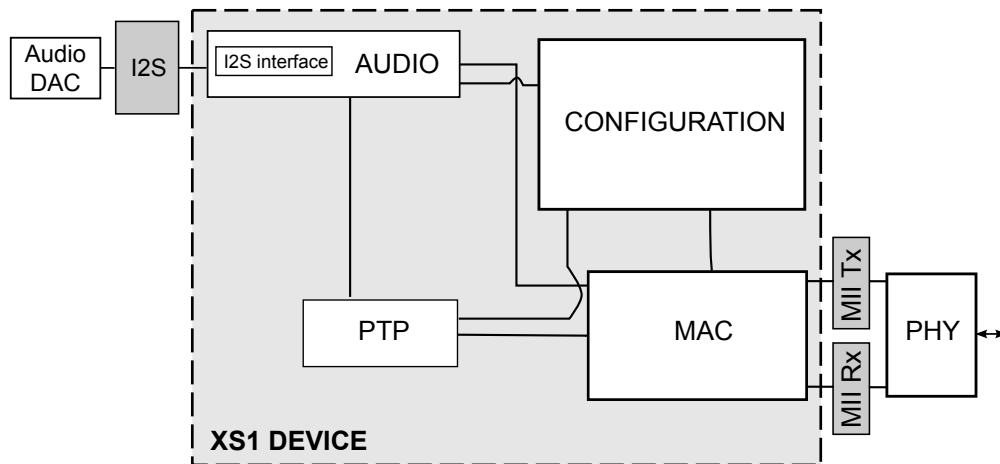


Figure 8 An minimal AVB system layout

A typical base system is shown in Figure 8 that could be used, for example, in a network connected speaker. The program takes a single stream over ethernet through the MII interface and outputs it via an I2S interface to a stereo DAC (a different audio interface could be used). In this case the configuration thread simply initialises the other components and configures the audio component to look for a specific stream.

Assuming that system configuration requires only one thread and a small amount of memory (<10k), the application would require 14 threads and 110k of memory, which would fit on two cores of an XS1-G4 device.

6.1.2 Multiple Streams, Connected to External System

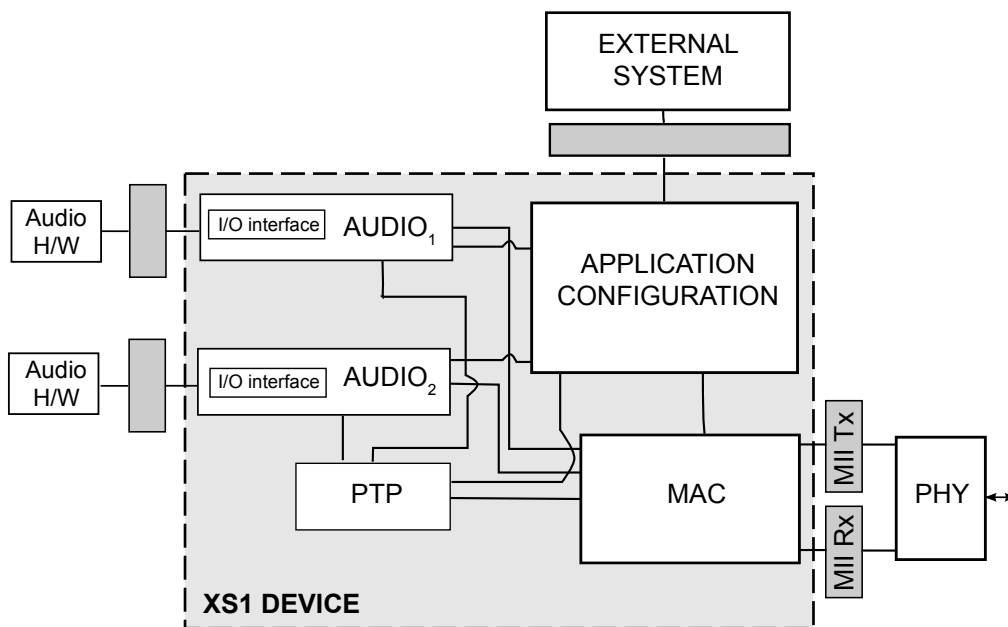


Figure 9 An AVB system connecting to a larger external system

Figure 9 shows a more complex system that is connected to a larger external system with two audio streams. The application code is likely to contain at least two threads: one for configuration and one for communicating with the rest of the system. Altogether this requires 20 threads and 150k of memory, it would fit onto an XS1-G4 device with additional space for audio streams and functionality.

6.1.3 Multiple Device System

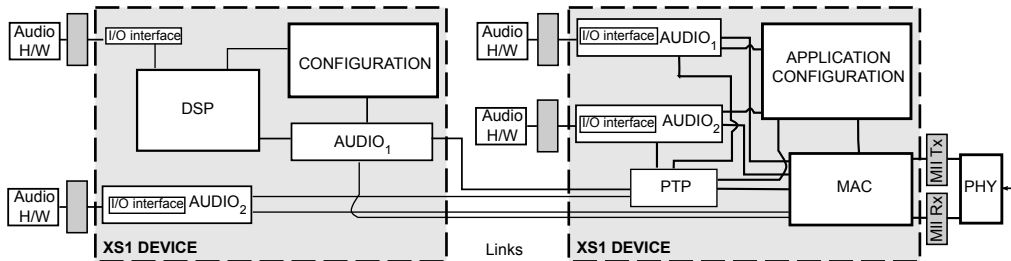


Figure 10 A multi-device AVB system

Figure 10 shows an example of a system spread across multiple XS1 devices, connected using links. Each link can be configured as standard XC channels.

The example has four audio streams spread across two devices. On the second device, the audio stream component has been altered to send the samples through a DSP filter before output. The second device does not need a MAC and therefore requires only 11 threads for configuration and audio streaming, leaving space for DSP on a programmable device such as an XS1-G4.

NOTE: For a multi-device system, both devices must use the same oscillator so that the system timing used by the PTP component information is accurate across devices.

6.2 System Design, Implementation and Customisation

To adapt the reference design to a particular application:

1. Alter the top-level source file of the application to call the relevant audio components and application threads. The reference design includes a template for the top-level source file.
2. Write functions to implement the application.
3. Write external system code to configure the XMOS device (optional)
4. Re-build the application and load onto an XMOS device.

This process alters the *Configuration Points* shown in Figure 7 so that the platform meets the application requirements.

Configuration Point 1: Audio Interface

The audio streaming component can be used with many different hardware interfaces. To configure the interface you need to:

- Add port variables to reference the hardware ports the audio hardware is connected to.
- Add the relevant function calls to the top-level source file of the application (using the port variables).

Figure 11 shows an example snippet of the configuration source code. Details of these interfaces can be found in Section 8.2.

```
par
{
...
on stdcore[3]: {audio_TI_TLV320AIC23B_init(p_aud_sclk, p_aud_sdin, 0);
                audio_i2s_dac_slave(p-aud_bclk,
                                   p-aud_lrcout,
                                   p_aud_dout;
                                   aud_bit_clock,
                                   clk_ctl,
                                   audio_data);}
on stdcore[3]: avb_1722_listener(ptp_link0,
                                rx_link1,
                                audio_data,
                                clk_ctl,
                                audio_stream_config);
...
}
```

Figure 11 *An audio configuration*

The component can be further modified by adding a bespoke audio hardware driver.

Configuration Point 2: Application/Configuration Threads

The application threads implement code for a particular application. Given all possible applications, the contents of the exact application code is beyond the scope of this document.

The application is likely to include a configuration section that sets up all the AVB components. An example template is included in the reference design. The configuration thread takes channel inputs from the AVB components and the configuration is made by calls to component APIS (see Section 8).

Once the application threads are developed, calls to the implemented functions need to be added to the top level source file.

Configuration Point 3: External System Configuration (optional)

The XMOS AVB application may be connected to other parts of a larger system. The following steps must be taken to allow communication between the host processor and the XMOS system:

- Implement an application thread on the XMOS system that communicates over a port protocol with the host system.
- Integrated with an external system that communicates with the XMOS system.

The reference design includes an example host library API that could be implemented (in the `host/` sub-directory). For details on possible hardware interfaces and their implementation please contact XMOS or visit www.xmos.com.

7 Development/Demonstration Platforms

To develop applications using the AVB Reference Design, you need to use the XMOS XS1-G Development Kit (XDK) which includes an ethernet PHY and a stereo audio CODEC to enable AVB development.

7.1 Typical Development Setups

The minimum setup required to demonstrate audio streaming is:

- 2 x XS1-G Development Kits
- Audio source (to 3.5mm audio jack)
- Audio listening equipment (from 3.5mm audio jack)
- 1 x CAT-5 patch or crossover cable

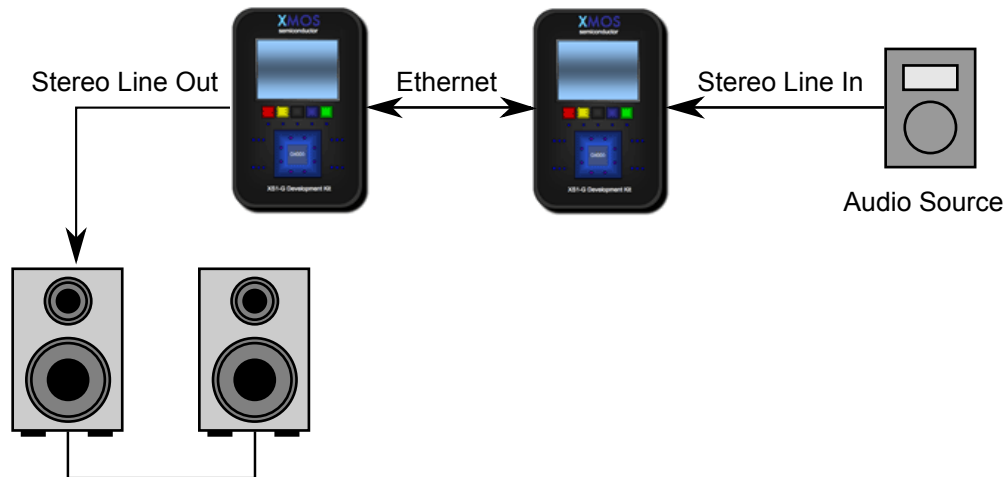


Figure 12 *Minimum XDK demo setup*

This setup can be used to demonstrate audio over ethernet. It does not allow testing with multiple talkers/listeners and may therefore make it hard to develop code for all situations.

A more flexible development platform is to use four XDKs, which allows testing with different combinations of talkers and listeners.

Figure 13 shows one configuration of a four-XDK setup. In addition to the XDKs, the setup shows a computer connected to the ethernet hub, which can be used to analyse ethernet traffic using a packet sniffer/analyser program. The full requirements for this setup are:

- 4 x XS1-G Development Kits
- Multiple audio sources (to 3.5mm audio jack)
- Multiple audio listening equipment (from 3.5mm audio jack)
- 4-5 x CAT-5 patch or crossover cables
- 1 x ethernet hub with 5 ports
- 1 x PC with packet sniffing software installed

NOTE: XMOS is developing an audio breakout board for the XS1-G Development Kit that attaches to the development kit and provides multiple audio interfaces to develop code on.

7.2 Building and running the demo

The AVB Reference Design provides a demo application to run on a development setup. To build the demo, you need to have the XMOS Desktop Tools Version 9.2.0 or later installed on a supported OS.

The demo implements streams played on the XDK's stereo CODEC via an *I2S* interface. The application consists of a configuration thread that sets the talkers transmitting and the listeners listening to the first stream they receive. In addition, the demo adds threads to display a frequency spectrum analysis and a vertical bar display representing the *802.1as* clock on the XDK's LCD screen.

The top-level directory contains a `Makefile` which builds the demo by default. Use the following command to build the demo:

```
xmake all
```

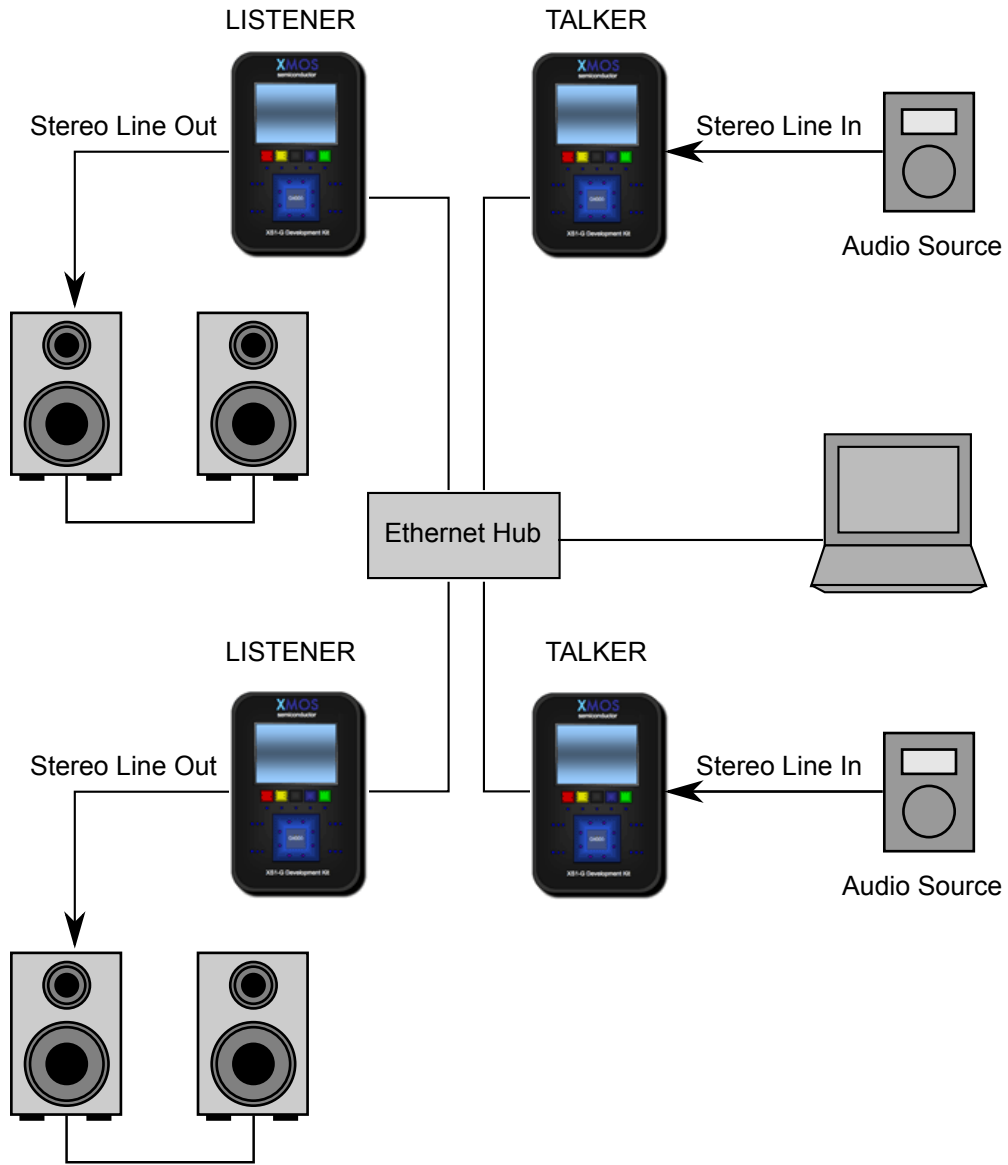


Figure 13 XDK development setup

This creates two XE and two XB binary files in the `bin` directory.

The XE binaries include debug information and can be loaded and debugged on the XDKs via JTAG:

Binary	Description
AVB_L.XE	Listener endpoint for loading over JTAG
AVB_T.XE	Talker endpoint for loading over JTAG

The XB binaries can be loaded onto the SD card provided with the XDK and run through the menu screen.

Binary	Description
AVB_L.XB	Listener endpoint for loading on SD card
AVB_T.XB	Talker endpoint for loading on SD card

7.2.1 Debug output

To debug the program you must load the programs onto your XDKs using the JTAG connector, and log output via UART on the connector. You need a connection to a terminal on the COM port provided by the driver on your OS. The speed of the UART is 115200 baud (N-8-1).

You can also use the debugger in XMOS Desktop Tools. See the *Desktop Tools Quick Start Guide* for further details.

8 System Reference

8.1 System Integration

The top-level source file is the main point for customisation of the system design. This file contains the port declarations used by the audio components and the application, along with a top level `main` function that declares which threads run on each core and which channels connect these threads.

Templates for a top level application can be found in the `app_template/` sub-directory of the reference design software. The files are described in the following table:

File	Description
<code>system_template.xc</code>	Template for a top level source file describing the system design
Makefile	Example Makefile for building an application

The code in each file is documented, explaining how to adapt it to a particular implementation.

8.2 Audio H/W Interface

The reference design includes various audio interfaces included in the `src/audio/audio_hw_interfaces` directory. Each interface has its own documentation in the source directory. There are two types of interface files:

- specific CODEC configuration implementations
- interface implementations

In the top level source file, a thread can be created that initialises the CODEC and then implements an interface talking to that CODEC. For example, Figure 11 shows an example where the Texas Instruments TI_TLV320AC23B is initialised and then communicated with using I2S with the XMOS device in I2S slave mode.

8.2.1 Audio Component

As well as an audio hardware interface, you need to specify an audio streaming component for each interface. The function to implement this depends on whether the audio component is a talker or listener:

Audio component type	Function Name
Talker	<code>avb_1722_talker</code>
Listener	<code>avb_1722_listener</code> <i>[postfix]</i>

The functions are documented in the file `src/audio/avb_1722.h`. The postfix to the listener function represents the type of clock recovery to use. The following table lists the postfixes and the clock recovery type.

File	Clock Recovery Type
<i>no postfix</i>	Digital sample rate synchronisation in the XCore
<code>_clock_recovery_ext</code>	External audio system clock adjustment
<code>_clock_recovery_i2s</code>	I ² S bus word-clock generation

Note: Not all clock recovery methods can be used with all hardware interfaces. See the individual function documentation for details.

8.3 Port Configuration

8.3.1 MAC port configuration

Each system has a single MAC component. The ports used to connect to the MII PHY are fixed and set in the file `src/mac/server/mii.xc`.

The following port definitions can be changed to map to particular ports on the device:

Port Variable	Description
p_mii_rxclk	RX Clock
p_mii_rxd	RX Data (4 bit port)
p_mii_rxdv	RX Valid
p_mii_rxer	RX Error
p_mii_txclk	TX Clock
p_mii_txd	TX Data (4 bit port)
p_mii_txen	TX Enable
p_mii_txer	TX Error

8.3.2 Audio port configuration

```

on stdcore[3]:port in p_aud_bclk          = XS1_PORT_1C;
on stdcore[3]:port in p_aud_lrcin       = XS1_PORT_1D;
on stdcore[3]:port in p_aud_lrcout      = XS1_PORT_1H;
on stdcore[3]:buffered in port:8 p_aud_din = XS1_PORT_1J;
on stdcore[3]:buffered out port:8 p_aud_dout = XS1_PORT_1I;

par
{
...
    audio_i2s_dac_slave(p_aud_bclk,
                        p_aud_lrcout,
                        p_aud_dout,
                        aud_bit_clock,
                        clk_ctl,
                        audio_data);}
...
}
    
```

Figure 14 *Audio component port configuration*

Audio ports are configured by parameters to the audio hardware interface functions. Each hardware interface function documents its port requirements.

The ports used should be declared at the start of the top level source file and passed as arguments to the hardware interface function. Figure 14 shows an

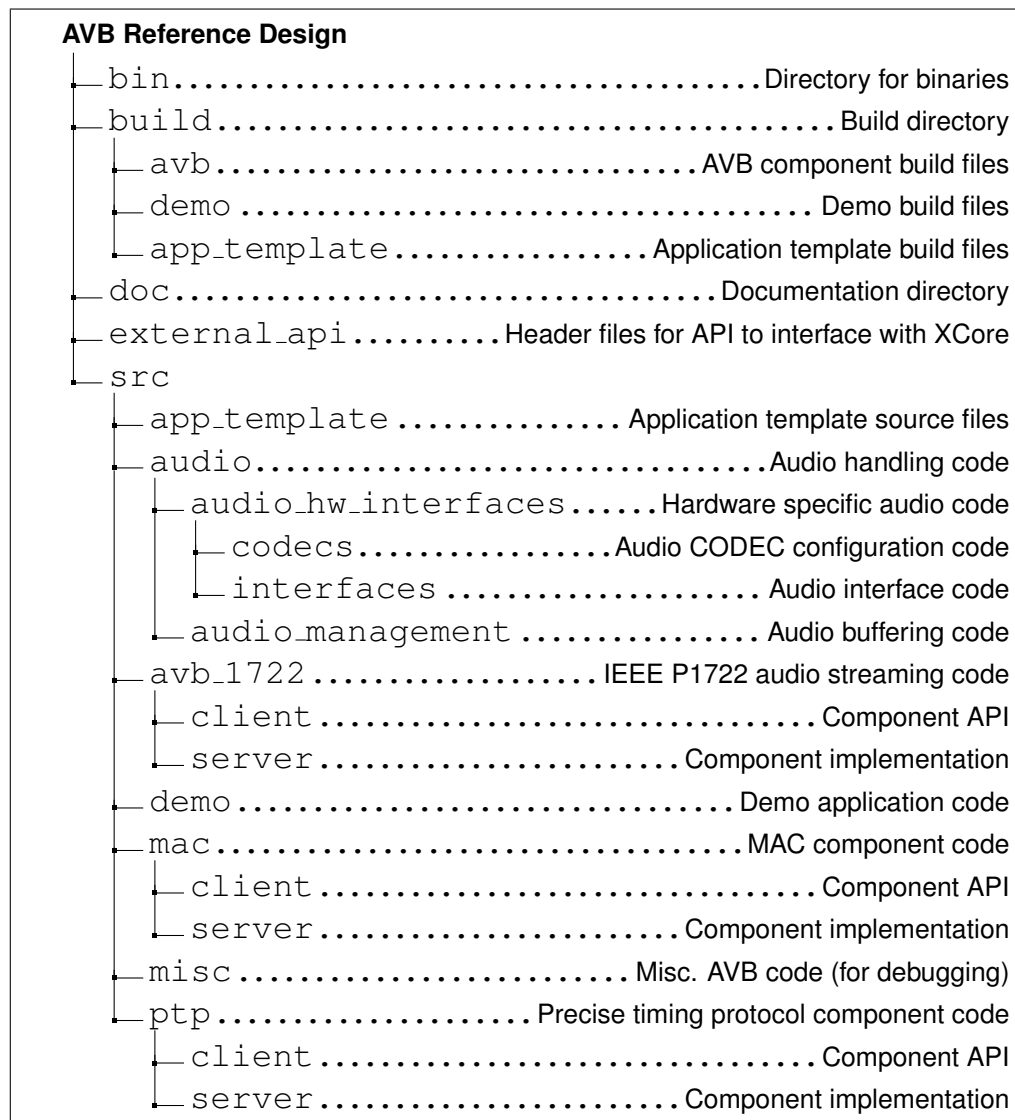
example of this configuration in the top level source file of the system.

See the top level application template file for more details.

A AVB Reference Design Software

A.1 Directory Structure

The AVB Reference Design is organised into a directory hierarchy described below:



A.2 Important Reference Files

File and Description	
README	Reference design README file
doc/release_notes	Reference design release notes
doc/CHANGELOG	Reference design change log
src/app_template/app_template.xc	Template for a top level source file describing the system design
build/app_template/Makefile	Example Makefile for building an application
src/mac/server/mii.xc	Source file containing MII port configuration
src/mac/client/ethernet*_client.h	MAC component API
src/ptp/client/ptp_client.h	PTP component API
src/avb_1722/client/avb_1722_client.h	Audio component API
external/*	Example API for use on external chip communicating with XMOS system

B Licensing and Support Information

B.1 Reference Design License

The XMOS AVB Reference Design software is distributed under the terms and conditions published at:

www.xmos.com/end-user-licence-agreement

B.2 Support

The reference design is maintained by XMOS and is provided “as is” under the terms described in Section [B.1](#). Although no formal support is provided with the design, the reporting of bugs and/or enhancement requests is appreciated. Any bugs and/or enhancement requests may be reported with a support ticket at www.xmos.com/support.

C Further Reading

Additional information can be found in the following documents:

- The XS1 System [2] document.

See also:

- www.xmos.com/documentation

D References

- [1] Douglas Watt and Richard Osborne and David May. XC Reference Manual (8.7). Website, 2008. <http://www.xmos.com/published/xc87>.
- [2] David May and Ali Dixon and Ayewin Oung and Henk Muller. XS1 System Specification. Website, 2008. <http://www.xmos.com/published/xsystem>.

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.

(c) 2009 XMOS Limited - All Rights Reserved