Time Sensitive Networking Library

This library provides components for creating time sensitive networking and media transport applications. In particular, it supports the various standards for Ethernet AVB (Audio Video Bridging).

Features

- 1722 61883-6 audio Talker and Listener (simultaneous) support
- 1722 MAAP support for Talkers
- 802.1Q MRP, MVRP, MSRP protocols
- gPTP server and protocol
- Media clock recovery and interface to PLL clock source
- Support for 1722.1 AVDECC: ADP, AECP (AEM) and ACMP

Software version and dependencies

This document pertains to version 7.0.0 of this library. It is known to work on version 14.0.3 of the xTIMEcomposer tools suite, it may work on other versions.

This library depends on the following other libraries:

- lib_logging (>=2.0.0)
- lib_xassert (>=2.0.0)
- lib_i2c (>=3.0.0)
- lib_ethernet (>=3.0.3)
- lib_otpinfo (>=2.0.0)

Typical Resource Usage

This following table shows typical resource usage in some different configurations. Exact resource usage will depend on the particular use of the library by the application.

Configuration	Pins	Ports	Clocks	Ram	Logical cores
Standalone gPTP server	0	0	0	~10.1K	1
Combined gPTP and media clock server	1	1 (1-bit)	0	~13.7K	1
1722 Talker (1 stream, 8 channels, 48kHz)	0	0	0	~4.7K	1
1722 Listener (1 stream, 8 channels, 48kHz)	0	0	0	~8.6K	1
1722.1 and MAAP protocol stack	0	0	0	~13.6K	0/1
Stream Reservation Protocol stack	0	0	0	~15.7K	0/1

See the Ethernet MAC and I2S/TDM library documentation for their typical resource usage.

Related application notes

The following application notes use this library:

- AN00202 - XMOS Gigabit Ethernet AVB I2S demo app note
- AN00203 - XMOS Gigabit Ethernet AVB TDM demo app note

Copyright 2015 XMOS Ltd.

1 Ethernet AVB standards

Ethernet AVB consists of a collection of different standards that together allow audio, video and time sensitive control data to be streamed over Ethernet. The standards provide synchronized, uninterrupted streaming with multiple talkers and listeners on a switched network infrastructure.

1.1 802.1AS

802.1AS defines a Precision Timing Protocol based on the *IEEE 1558v2* protocol. It allows every device connected to the network to share a common global clock. The protocol allows devices to have a synchronized view of this clock to within microseconds of each other, aiding media stream clock recovery to phase align audio clocks.

The IEEE 802.1AS-2011 standard document[1] is available to download free of charge via the IEEE Get Program.

1.2 802.1Qav

802.1Qav defines a standard for buffering and forwarding of traffic through the network using particular flow control algorithms. It gives predictable latency control on media streams flowing through the network.

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

The 802.1Qav specification is available as a section in the IEEE 802.1Q-2011 standard document[2] and is available to download free of charge via the IEEE Get Program.

1.3 802.1Qat

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

The 802.1Qat specification is available as a section in the IEEE 802.1Q-2011 standard document[3].

1.4 IEC 61883-6

IEC 61883-6 defines an audio data format that is contained in *IEEE 1722* streams. The XMOS AVB solution uses *IEC 61883-6* to convey audio sample streams.

The IEC 61883-6:2005 standard document[4] is available for purchase from the IEC website.

1.5 IEEE 1722

IEEE 1722 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 1722*. In addition it can use the *802.1AS* timestamps to accurately recover the audio master clock from an input stream.

The IEEE 1722-2011 standard document[5] is available for purchase from the IEEE website.

[1] http://standards.ieee.org/getieee802/download/802.1AS-2011.pdf
[2] http://standards.ieee.org/getieee802/download/802.1Q-2011.pdf
[3] http://standards.ieee.org/getieee802/download/802.1Q-2011.pdf
[4] http://webstore.iec.ch/webstore/webstore.nsf/ArtNum_PK/46793
[5] http://standards.ieee.org/findstds/standard/1722-2011.html

1.6 IEEE 1722.1

IEEE 1722.1 is a system control protocol, used for device discovery, connection management and enumeration and control of parameters exposed by the AVB endpoints.

The IEEE 1722.1-2013 standard document[6] is available for purchase from the IEEE website.

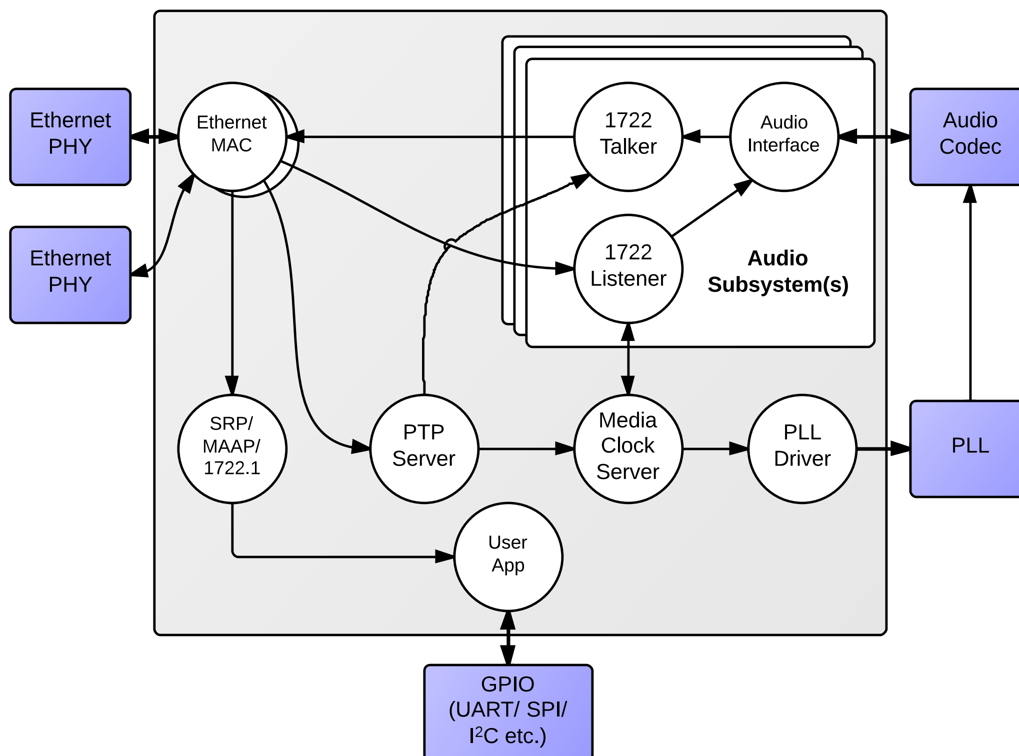
[6] http://standards.ieee.org/findstds/standard/1722.1-2013.html

2 Usage

An AVB/TSN audio endpoint consists of five main interacting components:

- The Ethernet MAC
- The Precision Timing Protocol (PTP) engine
- Audio streaming components
- The media clock server
- Configuration and other application components

The following diagram shows the top level structure of an AVB endpoint implemented on the xCORE architecture.



2.1 Ethernet MAC

The XMOS Ethernet MAC library provides the necessary standards-compliant AVB support for an endpoint.

If 10/100 Mb/s support is required only, the 10/100 Mb/s real-time Ethernet MAC should be used. Gigabit Ethernet is supported via the 10/100/1000 Mb/s real-time Ethernet MAC and will fallback to 10/100 Mb/s operation on 10/100 networks.

For full usage and API documentation, see the Ethernet MAC library user guide[7].

[7] https://www.xmos.com/published/lib_ethernet-userguide?version=latest

2.2 Precision Timing Protocol

The Precision Timing Protocol (PTP) enables a system with a notion of global time on a network. The TSN library implements the *IEEE 802.1AS* protocol. It allows synchronization of the presentation and playback rate of media streams across a network.

The PTP server requires a single logical core to run and connects to the Ethernet MAC. The library interprets PTP packets from the Ethernet MAC and maintains a notion of global time. The maintenance of global time requires no application interaction with the library.

The PTP library can be configured at runtime to be a potential *PTP grandmaster* or a *PTP slave* only. If the library 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 synchronize a global time from this source and the other grandmasters stop providing timing information. Depending on the intermediate network, this synchronization can be to sub-microsecond level resolution.

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

Client tasks can communicate with the server using the API described in Section §3.6.

- The PTP system in the endpoint is self-configuring, it runs automatically and gives each endpoint an accurate notion of a global clock.
- The global clock is *not* the same as the audio word clock, although it can be used to derive it. An audio stream may be at a rate that is independent of the PTP clock but will contain timestamps that use the global PTP clock domain as a reference.

2.3 Audio components

2.3.1 AVB streams, channels, talkers and listeners

Audio is transported in streams of data, where each stream may have multiple channels. Endpoints producing streams are called *Talkers* and those receiving them are called *Listeners*. Each stream on the network has a unique 64-bit stream ID.

A single endpoint can be a Talker, a Listener or both. In general each endpoint will have a number of *sinks* with the capacity to receive a number of incoming streams and a number of *sources* with the capacity to transmit a number of streams.

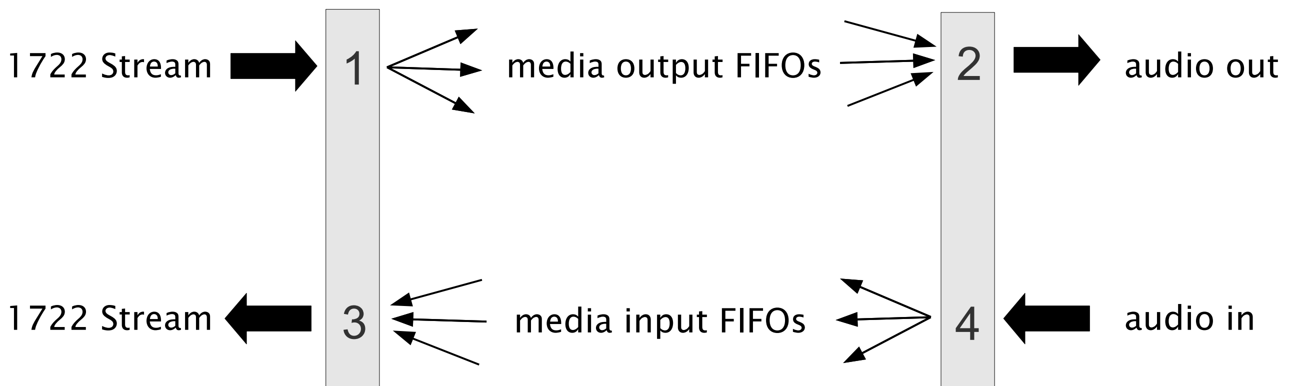
Routing is done using layer 2 Ethernet MAC addresses. The destination MAC address is a multicast address so that several Listeners may receive it. In addition, AVB switches can reserve an end-to-end path with guaranteed bandwidth for a stream. This is done by the Talker endpoint advertising the stream to the switches and the Listener(s) registering to receive it. If sufficient bandwidth is not available, this registration will fail.

Streams carry their own *presentation time*, the time that samples are due to be output, allowing multiple Listeners that receive the same stream to output in sync.

- Streams are encoded using the IEEE 1722 AVB transport protocol.
- All channels in a stream must be synchronized to the same sample clock.
- All the channels in a stream must come from the same Talker.
- Routing of audio streams uses Ethernet layer 2 routing based on a multicast destination MAC address
- Routing of channels is done at the stream level. All channels within a stream must be routed to the same place. However, a stream can be multicast to several Listeners, each of which picks out different channels.

- A single endpoint can be both a Talker and Listener.
- Information such as stream ID and destination MAC address of a Talker stream should be communicated to Listeners via 1722.1. (see Section §2.5).

2.3.2 Internal routing and audio buffering



As described in the previous section, an IEEE 1722 audio stream may consist of many channels. These channels need to be routed to particular audio I/Os on the endpoint. To achieve maximum flexibility the XMOS design uses intermediate audio buffering to route audio.

The above figure shows the breakdown of 1722 streams into local FIFOs. The figure shows four points where transitions to and from audio FIFOs occur. For audio being received by an endpoint:

1. When a 1722 stream is received, its channels are mapped to output audio FIFOs. This mapping can be configured dynamically so that it can be changed at runtime by the configuration component.
2. The digital hardware interface maps audio FIFOs to audio outputs. This mapping is fixed and is configured statically in the software.

For audio being transmitted by an endpoint:

1. The digital hardware interface maps digital audio inputs to a double buffer.
2. Several channels from this buffer can be combined into a 1722 stream. This mapping is dynamic.

The configuration of the mappings is handled through the API described in §3.4.

The audio buffering uses shared memory to move data between tasks, thus the filling and emptying of the buffers must be on the same tile.

2.3.3 Talker units

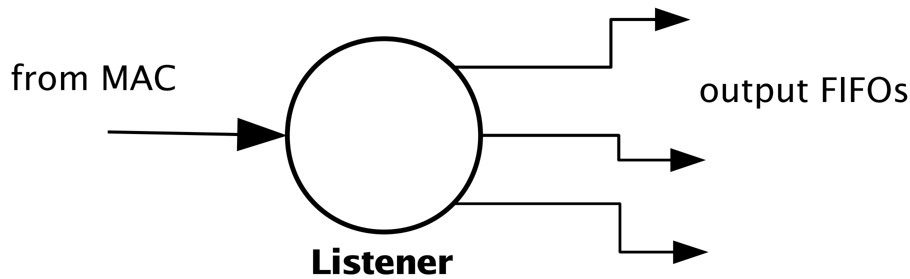
A Talker unit consists of one logical core which creates *IEEE 1722* packets and passes the audio samples onto the MAC. Audio samples are passed to this component via a double buffer. The Talker task copies a full buffer of samples into a 1722 packet while a different task implementing the audio hardware interface writes to a second buffer. Once the second buffer is full, the buffers are swapped.

Sample timestamps are converted to the time domain of the global clock provided by the PTP library, 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).

The instantiating of Talker units is performed via the API described in Section §3.5. Once the Talker unit starts, it registers with the main control task and is controlled via the main AVB API described in Section §3.4.

Copyright 2015 XMOS Ltd.

2.3.4 Listener units



A Listener unit takes *IEEE 1722* packets from the MAC and converts them into a sample stream to be fed into a media FIFO. Each audio Listener component can listen to several *IEEE 1722* streams.

A system may have several Listener units. The instantiating of Listener units is performed via the API described in Section §3.5. Once the Listener unit starts, it registers with the main control task and is controlled via the main AVB API described in Section §3.4.

2.3.5 Audio hardware interfaces

The audio hardware interface components drive external audio hardware, pull sample out of audio buffers and push samples into audio buffers.

Different interfaces may interact in different ways; some directly push and pull from the audio buffers, whereas some for performance reasons require samples to be provided over an XC channel.

2.4 Media clocks

A media clock controls the rate at which information is passed to an external audio device. For example, an audio word clock that governs the rate at which samples should be passed to an audio CODEC.

A media clock can be synchronized to one of two sources:

- An incoming clock signal on a port.
- The word clock of a remote endpoint, derived from an incoming *IEEE 1722* audio stream.

A hardware interface can be tied to a particular media clock, allowing the audio output from the XMOS device to be synchronized with other devices on the network.

All media clocks are maintained by the media clock server component. This component maintains the current state of all the media clocks in the system. It then periodically updates other components with clock change information to keep the system synchronized. The set of media clocks is determined by an array passed to the server at startup.

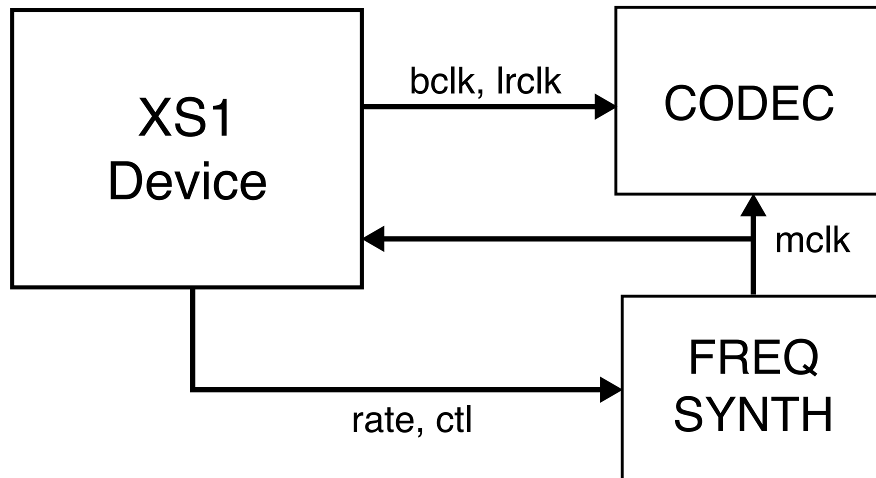
The media clock server component also receives information from the audio Listener component to track timing information of incoming *IEEE 1722* streams. It then sends control information back to ensure the listening component honors the presentation time of the incoming stream.

Multiple media clocks require multiple hardware PLLs or sample rate conversion.

2.4.1 Driving an external clock generator

A high quality, low jitter master clock is often required to drive an audio CODEC and must be synchronized with an AVB media clock. The xCORE architecture cannot provide the necessary quality of clock directly but can provide a lower frequency input source for a frequency synthesizer chip or external PLL chip. The frequency synthesizer chip must be able to generate a high frequency clock based on a lower frequency signal, such as the Cirrus Logic CS2100-CP. The recommended configuration is as in the block diagram

below:



The xCORE device provides control to the frequency synthesizer and the frequency synthesizer provides the audio master clock to the CODEC and xCORE device. The sample bit and word clocks are then provided to the CODEC by the xCORE device.

2.5 Device Discovery, Connection Management and Control

2.5.1 The control task

In addition to components described in previous sections, an AVB endpoint application requires a task to control and configure the system. This control task varies across applications but the protocol to provide device discovery, connection management and control services has been standardized by the IEEE in 1722.1.

2.5.2 1722.1

The 1722.1 standard defines four independent steps that can be used to connect end stations that use 1722 streams to transport media across a LAN. The steps are:

1. Discovery
2. Enumeration
3. Connection Management
4. Control

These steps can be used together to form a system of end stations that interoperate with each other in a standards compliant way. The application that will use these individual steps is called a *Controller* and is the third member in the Talker, Listener and Controller device relationship.

A Controller may exist within a Talker, a Listener, or exist remotely within the network in a separate endpoint or general purpose computer.

The Controller can use the individual steps to find, connect and control entities on the network but it may choose to not use all of the steps if the Controller already knows some of the information (e.g. hard coded values assigned by user/hardware switch or values from previous session establishment) that can be gained in using the steps. The only required step is connection management because this is the step that establishes the bandwidth usage and reservations across the AVB network.

The four steps are broken down as follows:

Copyright 2015 XMOS Ltd.

- Discovery is the process of finding AVB endpoints on the LAN that have services that are useful to the other AVB endpoints on the network. The discovery process also covers the termination of the publication of those services on the network.
- Enumeration is the process of the collection of information from the AVB endpoint that could help an 1722.1 Controller to use the capabilities of the AVB endpoint. This information can be used for connection management.
- Connection management is the process of connecting or disconnecting one or more streams between two or more AVB endpoint.
- Control is the process of adjusting a parameter on the endpoint from another endpoint. There are a number of standard types of controls used in media devices like volume control, mute control and so on. A framework of basic commands allows the control process to be extended by the endpoint.

  The XMOS endpoint provides full support for Talker and Listener 1722.1 services. It is expected that Controller software will be available on the network for handling connection management and control.

2.5.3 1722.1 Descriptors

The XMOS AVB reference design provides an AVDECC Entity Model (AEM) consisting of descriptors to describe the internal components of the Entity. For a complete overview of AEM, see section 7 of the 1722.1 specification.

An AEM descriptor is a fixed field structure followed by variable length data which describes an object in the AEM Entity model. The maximum length of a descriptor is 508 octets.

All descriptors share two common fields which are used to uniquely identify a descriptor by a type and an index. AEM defines a number of descriptors for specific parts of the Entity model. The descriptor types that XMOS currently provide in the reference design are listed in the table below.

2.5.4 Editing descriptors

The descriptors are declared in the a header configuration file named `aem_descriptors.h.in` within the `src/` directory of the application. The XMOS Reference column in the table refers to the array names of the descriptors in this file.

This file is post-processed by a script in the build stage to expand strings to 64 octet padded with zeros.

Name	Description	XMOS Reference
ENTITY	This is the top level descriptor defining the Entity.	desc_entity
CONFIGURATION	This is the descriptor defining a configuration of the Entity.	desc_configuration_0
AUDIO_UNIT	This is the descriptor defining an audio unit.	desc_audio_unit_0
STREAM_INPUT	This is the descriptor defining an input stream to the Entity.	desc_stream_input_0
STREAM_OUTPUT	This is the descriptor defining an output stream from the Entity.	desc_stream_output_0
JACK_INPUT	This is the descriptor defining an input jack on the Entity.	desc_jack_input_0
JACK_OUTPUT	This is the descriptor defining an output jack on the Entity.	desc_jack_output_0
AVB_INTERFACE	This is the descriptor defining an AVB interface.	desc_avb_interface_0
CLOCK_SOURCE	This is the descriptor describing a clock source.	desc_clock_source_0..1
LOCALE	This is the descriptor defining a locale.	desc_locale_0
STRINGS	This is the descriptor defining localized strings.	desc_strings_0
STREAM_PORT_INPUT	This is the descriptor defining an input stream port on a unit.	desc_stream_port_input_0
STREAM_PORT_OUTPUT	This is the descriptor defining an output stream port on a unit.	desc_stream_port_output_0
EXTERNAL_PORT_INPUT	This is the descriptor defining an input external port on a unit.	desc_external_input_port_0
EXTERNAL_PORT_OUTPUT	This is the descriptor defining an output external port on a unit.	desc_external_output_port_0
AUDIO_CLUSTER	This is the descriptor defining a cluster of channels within an audio stream.	desc_audio_cluster_0..N
AUDIO_MAP	This is the descriptor defining the mapping between the channels of an audio stream and the channels of the audio port.	desc_audio_map_0..N
CLOCK_DOMAIN	This is the descriptor describing a clock domain.	desc_clock_domain_0

2.5.5 Adding and removing descriptors

Descriptors are indexed by a descriptor list named aem_descriptor_list in the aem_descriptors.h.in file.

The format for this list is as follows:

Descriptor type
Number of descriptors of type (N)
Size of descriptor 0 (bytes)
Address of descriptor 0
...
Size of descriptor N (bytes)
Address of descriptor N

For example:

AEM_ENTITY_TYPE, 1, `sizeof(desc_entity)`, `(unsigned)desc_entity`

3 API

All AVB/TSN functions can be accessed via the avb.h header:

```
#include <avb.h>
```

You will also have to add lib_tsn to the USED_MODULES field of your application Makefile.

3.1 Audio subsystem defines

AVB applications using the TSN library must include a header configuration file named avb_conf.h within the src/ directory of the application and this file must set the following values with #defines.

Macro	**AVB_MAX_AUDIO_SAMPLE_RATE**
Description	The maximum sample rate in Hz of audio that is to be input or output.

Macro	**AVB_NUM_SOURCES**
Description	The total number of AVB sources (streams that are to be transmitted).

Macro	**AVB_NUM_TALKER_UNITS**
Description	The total number or Talker components (typically the number of tasks running the avb_1722_talker function).

Macro	**AVB_MAX_CHANNELS_PER_TALKER_STREAM**
Description	The maximum number of channels permitted per 1722 Talker stream.

Macro	**AVB_NUM_MEDIA_INPUTS**
Description	The total number of media inputs (typically number of I2S input channels).

Macro	**AVB_NUM_SINKS**
Description	The total number of AVB sinks (incoming streams that can be listened to).

Macro	**AVB_NUM_LISTENER_UNITS**
Description	The total number or listener components (typically the number of tasks running the avb_1722_listener function).

Macro	AVB_MAX_CHANNELS_PER_LISTENER_STREAM
Description	The maximum number of channels permitted per 1722 Listener stream.

Macro	AVB_NUM_MEDIA_OUTPUTS
Description	The total number of media outputs (typically the number of I2S output channels).

Macro	AVB_NUM_MEDIA_UNITS
Description	The number of components in the endpoint that will register and initialize media FIFOs (typically an audio interface component such as I2S).

Macro	AVB_NUM_MEDIA_CLOCKS
Description	The number of media clocks in the endpoint. Typically the number of clock domains, each with a separate PLL and master clock.

3.2 1722.1

Macro	AVB_ENABLE_1722_1
Description	Enable 1722.1 AVDECC on the entity.

Macro	AVB_1722_1_TALKER_ENABLED
Description	Enable the 1722.1 Talker functionality.

Macro	AVB_1722_1_LISTENER_ENABLED
Description	Enable the 1722.1 Listener functionality.

Macro	AVB_1722_1_CONTROLLER_ENABLED
Description	Enable 1722.1 Controller functionality on the entity.

Descriptor specific strings can be modified in a header configuration file named `aem_entity_strings.h.in` within the `src/` directory. It is post-processed by a script in the build stage to expand strings to 64 octet padded with zeros.

Define	Description
AVB_1722_1_ENTITY_NAME_STRING	A string (64 octet max) containing an Entity name
AVB_1722_1_FIRMWARE_VERSION_STRING	A string (64 octet max) containing the firmware version of the Entity
AVB_1722_1_GROUP_NAME_STRING	A string (64 octet max) containing the group name of the Entity
AVB_1722_1_SERIAL_NUMBER_STRING	A string (64 octet max) containing the serial number of the Entity
AVB_1722_1_VENDOR_NAME_STRING	A string (64 octet max) containing the vendor name of the Entity
AVB_1722_1_MODEL_NAME_STRING	A string (64 octet max) containing the model name of the Entity

3.3 1722.1 application hooks

These hooks are called on events that can be acted upon by the application. They can be overridden by user defined hooks of the same name to perform custom functionality not present in the core stack.

Function	**avb_talker_on_listener_connect**
Description	A Controller has indicated that a Listener is connecting to this Talker stream source.

Type	`void`
	`avb_talker_on_listener_connect(`
	`client interface avb_interface i_avb,`
	`int source_num,`
	`const_guid_ref_t listener_guid)`

Parameters	`i_avb` client interface of type `avb_interface` into avb_manager()
	
	`source_num`
	The local id of the Talker stream source
	
	`listener_guid`
	The GUID of the Listener entity that is connecting

Function	**avb_talker_on_listener_disconnect**
Description	A Controller has indicated that a Listener is disconnecting from this Talker stream source.

Type	`void`
	`avb_talker_on_listener_disconnect(`
	`client interface avb_interface i_avb,`
	`int source_num,`
	`const_guid_ref_t listener_guid,`
	`int connection_count)`

Continued on next page

Copyright 2015 XMOS Ltd.

Parameters	
	i_avb — client interface of type avb_interface into avb_manager() source_num — The local id of the Talker stream source listener_guid — The GUID of the Listener entity that is disconnecting connection_count — The number of connections a Talker thinks it has on it's stream source, i.e. the number of connect TX stream commands it has received less the number of disconnect TX stream commands it has received. This number may not be accurate since an AVDECC Entity may not have sent a disconnect command if the cable was disconnected or the AVDECC Entity abruptly powered down.

Function	avb_listener_on_talker_connect
Description	A Controller has indicated to connect this Listener sink to a Talker stream.

Type	`avb_1722_1_acmp_status_t`
	`avb_listener_on_talker_connect(`
	`client interface avb_interface i_avb,`
	`int sink_num,`
	`const_guid_ref_t talker_guid,`
	`unsigned char dest_addr[6],`
	`unsigned int stream_id[2],`
	`unsigned short vlan_id,`
	`const_guid_ref_t my_guid)`

Parameters	i_avb — client interface of type avb_interface into avb_manager() sink_num — The local id of the Listener stream sink talker_guid — The GUID of the Talker entity that is connecting dest_addr — The destination MAC address of the Talker stream stream_id — The 64 bit Stream ID of the Talker stream vlan_id — The VLAN ID of the Talker stream my_guid — The GUID of this entity

Function	avb_listener_on_talker_disconnect
Description	A Controller has indicated to disconnect this Listener sink from a Talker stream.
Type	`void avb_listener_on_talker_disconnect( client interface avb_interface i_avb, int sink_num, const_guid_ref_t talker_guid, unsigned char dest_addr[6], unsigned int stream_id[2], const_guid_ref_t my_guid)`
Parameters	`i_avb` client interface of type avb_interface into avb_manager() `sink_num` The local id of the Listener stream sink `talker_guid` The GUID of the Talker entity that is disconnecting `dest_addr` The destination MAC address of the Talker stream `stream_id` The 64 bit Stream ID of the Talker stream `my_guid` The GUID of this entity

Type	avb_1722_1_aecp_aem_status_code
Description	The result status of the AEM command in the response field.
Values	AECP_AEM_STATUS_SUCCESS The AVDECC Entity successfully performed the command and has valid results. AECP_AEM_STATUS_NOT_IMPLEMENTED The AVDECC Entity does not support the command type. AECP_AEM_STATUS_NO_SUCH_DESCRIPTOR A descriptor with the descriptor_type and descriptor_index specified does not exist. AECP_AEM_STATUS_ENTITY_LOCKED The AVDECC Entity has been locked by another AVDECC Controller. AECP_AEM_STATUS_ENTITY_ACQUIRED The AVDECC Entity has been acquired by another AVDECC Controller.

Continued on next page

	AECP_AEM_STATUS_NOT_AUTHENTICATED The AVDECC Controller is not authenticated with the AVDECC Entity. AECP_AEM_STATUS_AUTHENTICATION_DISABLED The AVDECC Controller is trying to use an authentication command when authentication isn't enable on the AVDECC Entity. AECP_AEM_STATUS_BAD_ARGUMENTS One or more of the values in the fields of the frame were deemed to be bad by the AVDECC Entity (unsupported, incorrect combination, etc). AECP_AEM_STATUS_NO_RESOURCES The AVDECC Entity cannot complete the command because it does not have the resources to support it. AECP_AEM_STATUS_IN_PROGRESS The AVDECC Entity is processing the command and will send a second response at a later time with the result of the command. AECP_AEM_STATUS_ENTITY_MISBEHAVING The AVDECC Entity is generated an internal error while trying to process the command. AECP_AEM_STATUS_NOT_SUPPORTED The command is implemented but the target of the command is not supported. For example trying to set the value of a read-only Control. AECP_AEM_STATUS_STREAM_IS_RUNNING The Stream is currently streaming and the command is one which cannot be executed on an Active Stream.

Type	**avb_1722_1_control_callbacks**
Description	A callback interface for 1722.1 events.

Continued on next page

Type	avb_1722_1_control_callbacks (continued)
Functions	

Function	get_control_value
Description	This function events on a GET_CONTROL 1722.1 command received from a Controller.

Type	`unsigned char`
	`get_control_value(unsigned short control_index,`
	`    unsigned int &value_size,`
	`    unsigned short &values_length,`
	`    unsigned char values[])`

Parameters	`control_index` the index of the CONTROL descriptor `value_size` the size in bytes of the type of the value `values_length` a reference to the length in bytes of the values array `values` an array of values to return to the Controller The contents of this field are dependent on the Control being fetched.
Returns	an AEM status code of enum avb_1722_1_aecp_aem_status_code for the GET_CONTROL response

Continued on next page

Type	avb_1722_1_control_callbacks (continued)
	(nested table below)

Function	set_control_value
Description	This function events on a SET_CONTROL 1722.1 command received from a Controller. The response should always contain the current value (i.e. it contains the new value if the commands succeeds, or the old value if it fails)
Type	`unsigned char set_control_value(unsigned short control_index, unsigned short values_length, unsigned char values[])`
Parameters	`control_index` the index of the CONTROL descriptor `values_length` the length in bytes of the `values` array `values` an array of values to set from the Controller. The contents of this field are dependent on the Control being addressed.
Returns	an AEM status code of enum avb_1722_1_aecp_aem_status_code for the SET_CONTROL response

Continued on next page

Type	avb_1722_1_control_callbacks (continued)
	Function: get_signal_selector **Description:** This function events on a GET_SIGNAL_SELECTOR 1722.1 command received from a Controller. **Type:** `unsigned char get_signal_selector(unsigned short selector_index, unsigned short &signal_type, unsigned short &signal_index, unsigned short &signal_output)` **Parameters:** `selector_index` the index of the SIGNAL_SELECTOR descriptor `signal_type` a reference to the descriptor type of signal source for the selector `signal_index` a reference to the descriptor index of signal source for the selector `signal_output` a reference to the index of the output of the signal source of the selector **Returns:** an AEM status code of enum avb_1722_1_aecp_aem_status_code for the GET_SIGNAL_SELECTOR response

Continued on next page

Type	avb_1722_1_control_callbacks (continued)

Function	set_signal_selector
Description	This function events on a SET_SIGNAL_SELECTOR 1722.1 command received from a Controller.
Type	`unsigned char set_signal_selector(unsigned short selector_index, unsigned short signal_type, unsigned short signal_index, unsigned short signal_output)`
Parameters	`selector_index` the index of the SIGNAL_SELECTOR descriptor `signal_type` the descriptor type of signal source for the selector `signal_index` the descriptor index of signal source for the selector `signal_output` the index of the output of the signal source of the selector
Returns	an AEM status code of enum avb_1722_1_aecp_aem_status_code for the SET_SIGNAL_SELECTOR response

3.4 AVB Control API

Type	**avb_stream_format_t**
Description	The audio format of a 1722 Talker or Listener.
Values	AVB_FORMAT_MBLA_24BIT 24bit MBLA

Type	**avb_source_state_t**
Description	The state of an AVB source (Talker).
Values	AVB_SOURCE_STATE_DISABLED The source is disabled and will not transmit. AVB_SOURCE_STATE_POTENTIAL The source is enabled and will transmit if a listener requests it. AVB_SOURCE_STATE_ENABLED The source is enabled and transmitting.

Type	**avb_sink_state_t**
Description	The state of an AVB sink (Listener).
Values	AVB_SINK_STATE_DISABLED The sink is disabled. AVB_SINK_STATE_POTENTIAL The sink is enabled and will pass audio when a Talker requests it. AVB_SINK_STATE_ENABLED The sink is enabled and passing audio.

Type	**device_media_clock_type_t**
Description	The type of source to use as a media clock.
Values	DEVICE_MEDIA_CLOCK_INPUT_STREAM_DERIVED The clock is sourced from the media clock of an Input Stream.

Continued on next page

Copyright 2015 XMOS Ltd.

	DEVICE_MEDIA_CLOCK_LOCAL_CLOCK The clock is sourced from within the entity from the local crystal oscillator.

Type	**device_media_clock_state_t**
Description	The state of a media clock.
Values	DEVICE_MEDIA_CLOCK_STATE_DISABLED The media clock is disabled. DEVICE_MEDIA_CLOCK_STATE_ENABLED The media clock is enabled.

Type	**avb_interface**
Description	The core AVB interface API for interacting with the endpoint.
Functions	

Function	**_get_source_info**
Description	Intended for internal use within client interface get and set extensions only.
Type	`avb_source_info_t _get_source_info(unsigned source_num)`

Function	**_set_source_info**
Description	Intended for internal use within client interface get and set extensions only.
Type	`void _set_source_info(unsigned source_num, avb_source_info_t info)`

Function	**_get_sink_info**
Description	Intended for internal use within client interface get and set extensions only.
Type	`avb_sink_info_t _get_sink_info(unsigned sink_num)`

Continued on next page

Type	avb_interface (continued)

Function	_set_sink_info
Description	Intended for internal use within client interface get and set extensions only.
Type	`void _set_sink_info(unsigned sink_num, avb_sink_info_t info)`

Function	_get_media_clock_info
Description	Intended for internal use within client interface get and set extensions only.
Type	`media_clock_info_t _get_media_clock_info(unsigned clock_num)`

Function	_set_media_clock_info
Description	Intended for internal use within client interface get and set extensions only.
Type	`void _set_media_clock_info(unsigned clock_num, media_clock_info_t info)`

Function	get_source_format
Description	Get the format of an AVB source.
Type	`int get_source_format(unsigned source_num, enum avb_stream_format_t &format, int &rate)`
Parameters	`source_num` the local source number `format` the format of the stream `rate` the sample rate of the stream in Hz

Continued on next page

Type	avb_interface (continued)

Function	set_source_format
Description	Set the format of an AVB source. The AVB source format covers the encoding and sample rate of the source. Currently the format is limited to a single encoding MBLA 24 bit signed integers. This setting will not take effect until the next time the source state moves from disabled to potential.
Type	`int set_source_format(unsigned source_num, enum avb_stream_format_t format, int rate)`
Parameters	`source_num` the local source number `format` the format of the stream `rate` the sample rate of the stream in Hz

Function	get_source_channels
Description	Get the channel count of an AVB source.
Type	`int get_source_channels(unsigned source_num, int &channels)`
Parameters	`source_num` the local source number `channels` the number of channels

Continued on next page

Type	avb_interface (continued)

Function	set_source_channels
Description	Set the channel count of an AVB source. Sets the number of channels in the stream. This setting will not take effect until the next time the source state moves from disabled to potential.
Type	`int set_source_channels(unsigned source_num, int channels)`
Parameters	`source_num` the local source number `channels` the number of channels

Function	get_source_sync
Description	Get the media clock of an AVB source.
Type	`int get_source_sync(unsigned source_num, int &sync)`
Parameters	`source_num` the local source number `sync` the media clock number

Function	set_source_sync
Description	Set the media clock of an AVB source. Sets the media clock of the stream.
Type	`int set_source_sync(unsigned source_num, int sync)`
Parameters	`source_num` the local source number `sync` the media clock number

Continued on next page

Type	avb_interface (continued)

Function	get_source_presentation
Description	Get the presentation time offset of an AVB source.
Type	`int get_source_presentation(unsigned source_num, int &presentation)`
Parameters	`source_num` the local source number to set `presentation` the presentation offset in ms

Function	set_source_presentation
Description	Set the presentation time offset of an AVB source. Sets the presentation time offset of a source i.e. the time after sampling that the stream should be played. The default value for this is 2ms. This setting will not take effect until the next time the source state moves from disabled to potential.
Type	`int set_source_presentation(unsigned source_num, int presentation)`
Parameters	`source_num` the local source number to set `presentation` the presentation offset in ms

Function	get_source_vlan
Description	Get the destination vlan of an AVB source.
Type	`int get_source_vlan(unsigned source_num, int &vlan)`
Parameters	`source_num` the local source number `vlan` the destination vlan id

Continued on next page

Type	avb_interface (continued)

Function	set_source_vlan
Description	Set the destination vlan of an AVB source. Sets the vlan that the source will transmit on. This defaults to 2. This setting will not take effect until the next time the source state moves from disabled to potential.
Type	`int set_source_vlan(unsigned source_num, int vlan)`
Parameters	`source_num` the local source number `vlan` the destination vlan id

Function	get_source_state
Description	Get the current state of an AVB source.
Type	`int get_source_state(unsigned source_num, enum avb_source_state_t &state)`
Parameters	`source_num` the local source number `state` the state of the source

Function	set_source_state
Description	Set the current state of an AVB source. Sets the current state of an AVB source. You cannot set the state to ENABLED. Changing the state to AVB_SOURCE_STATE_POTENTIAL turns the stream on and it will automatically change to ENABLED when connected to a listener and streaming.
Type	`int set_source_state(unsigned source_num, enum avb_source_state_t state)`
Parameters	`source_num` the local source number `state` the state of the source

Continued on next page

Type	avb_interface (continued)

Function	get_source_map
Description	Get the channel map of an avb source.
Type	`int get_source_map(unsigned source_num, int map[], int &len)`
Parameters	`source_num` the local source number to set `map` the map, an array of integers giving the input FIFOs that make up the stream `len` the length of the map; should be equal to the number of channels in the stream

Function	set_source_map
Description	Set the channel map of an avb source. Sets the channel map of a source i.e. the list of input FIFOs that constitute the stream. This setting will not take effect until the next time the source state moves from disabled to potential.
Type	`int set_source_map(unsigned source_num, int map[len], unsigned len)`
Parameters	`source_num` the local source number to set `map` the map, an array of integers giving the input FIFOs that make up the stream `len` the length of the map; should be equal to the number of channels in the stream

Continued on next page

Type	avb_interface (continued)

Function	get_source_dest
Description	Get the destination address of an avb source.
Type	`int get_source_dest(unsigned source_num, unsigned char addr[], int &len)`
Parameters	`source_num` the local source number `addr` the destination address as an array of 6 bytes `len` the length of the address, should always be equal to 6

Function	set_source_dest
Description	Set the destination address of an avb source. Sets the destination MAC address of a source. This setting will not take effect until the next time the source state moves from disabled to potential.
Type	`int set_source_dest(unsigned source_num, unsigned char addr[len], unsigned len)`
Parameters	`source_num` the local source number `addr` the destination address as an array of 6 bytes `len` the length of the address, should always be equal to 6

Function	get_source_id
Description	
Type	`int get_source_id(unsigned source_num, unsigned int id[2])`

Continued on next page

Type	avb_interface (continued)

Function	get_sink_id
Description	Get the stream id that an AVB sink listens to.
Type	`int get_sink_id(unsigned sink_num, unsigned int stream_id[2])`
Parameters	`sink_num` the number of the sink `stream_id` int array containing the 64-bit of the stream

Function	set_sink_id
Description	Set the stream id that an AVB sink listens to. Sets the stream id that an AVB sink listens to. This setting will not take effect until the next time the sink state moves from disabled to potential.
Type	`int set_sink_id(unsigned sink_num, unsigned int stream_id[2])`
Parameters	`sink_num` the number of the sink `stream_id` int array containing the 64-bit of the stream

Function	get_sink_format
Description	Get the format of an AVB sink.
Type	`int get_sink_format(unsigned sink_num, enum avb_stream_format_t &format, int &rate)`
Parameters	`sink_num` the local sink number `format` the format of the stream `rate` the sample rate of the stream in Hz

Continued on next page

Copyright 2015 XMOS Ltd.

Type	avb_interface (continued)

Function	set_sink_format
Description	Set the format of an AVB sink. The AVB sink format covers the encoding and sample rate of the sink. Currently the format is limited to a single encoding MBLA 24 bit signed integers. This setting will not take effect until the next time the sink state moves from disabled to potential.
Type	`int set_sink_format(unsigned sink_num, enum avb_stream_format_t format, int rate)`
Parameters	`sink_num` the local sink number `format` the format of the stream `rate` the sample rate of the stream in Hz

Function	get_sink_channels
Description	Get the channel count of an AVB sink.
Type	`int get_sink_channels(unsigned sink_num, int &channels)`
Parameters	`sink_num` the local sink number `channels` the number of channels

Function	set_sink_channels
Description	Set the channel count of an AVB sink. Sets the number of channels in the stream. This setting will not take effect until the next time the sink state moves from disabled to potential.
Type	`int set_sink_channels(unsigned sink_num, int channels)`
Parameters	`sink_num` the local sink number `channels` the number of channels

Continued on next page

Type	avb_interface (continued)

Function	get_sink_sync
Description	Get the media clock of an AVB sink.
Type	`int get_sink_sync(unsigned sink_num, int &sync)`
Parameters	`sink_num` the local sink number `sync` the media clock number

Function	set_sink_sync
Description	Set the media clock of an AVB sink. Sets the media clock of the stream.
Type	`int set_sink_sync(unsigned sink_num, int sync)`
Parameters	`sink_num` the local sink number `sync` the media clock number

Function	get_sink_vlan
Description	Get the virtual lan id of an AVB sink.
Type	`int get_sink_vlan(unsigned sink_num, int &vlan)`
Parameters	`sink_num` the number of the sink `vlan` the vlan id of the sink

Function	set_sink_vlan
Description	Set the virtual lan id of an AVB sink. Sets the vlan id of the incoming stream. This setting will not take effect until the next time the sink state moves from disabled to potential.
Type	`int set_sink_vlan(unsigned sink_num, int vlan)`
Parameters	`sink_num` the number of the sink `vlan` the vlan id of the sink

Continued on next page

Type	avb_interface (continued)

Function	get_sink_addr
Description	Get the incoming destination mac address of an avb sink.
Type	`int get_sink_addr(unsigned sink_num, unsigned char addr[], int &len)`
Parameters	`sink_num` The local sink number `addr` The mac address as an array of 6 bytes. `len` The length of the address, should always be equal to 6.

Function	set_sink_addr
Description	Set the incoming destination mac address of an avb sink. Set the incoming destination mac address of a sink. This needs to be set if the address is a multicast address so the endpoint can register for that multicast group with the switch. This setting will not take effect until the next time the sink state moves from disabled to potential.
Type	`int set_sink_addr(unsigned sink_num, unsigned char addr[len], unsigned len)`
Parameters	`sink_num` The local sink number `addr` The mac address as an array of 6 bytes. `len` The length of the address, should always be equal to 6.

Function	get_sink_state
Description	Get the state of an AVB sink.
Type	`int get_sink_state(unsigned sink_num, enum avb_sink_state_t &state)`
Parameters	`sink_num` the number of the sink `state` the state of the sink

Continued on next page

Copyright 2015 XMOS Ltd.

Type	avb_interface (continued)

Function	set_sink_state
Description	Set the state of an AVB sink. Sets the current state of an AVB sink. You cannot set the state to ENABLED. Changing the state to POTENTIAL turns the stream on and it will automatically change to ENABLED when connected to a talker and receiving samples.
Type	`int set_sink_state(unsigned sink_num, enum avb_sink_state_t state)`
Parameters	`sink_num` the number of the sink `state` the state of the sink

Function	get_sink_map
Description	Get the map of an AVB sink.
Type	`int get_sink_map(unsigned sink_num, int map[], int &len)`
Parameters	`sink_num` the number of the sink `map` array containing the media output FIFOs that the stream will be split into `len` the length of the map; should equal to the number of channels in the stream

Continued on next page

Copyright 2015 XMOS Ltd.

Type	avb_interface (continued)

Function	set_sink_map
Description	Set the map of an AVB sink. Sets the map i.e. the mapping from the 1722 stream to output FIFOs. This setting will take affect immediately.
Type	`int set_sink_map(unsigned sink_num, int map[len], unsigned len)`
Parameters	`sink_num` the number of the sink `map` array containing the media output FIFOs that the stream will be split into `len` the length of the map; should equal to the number of channels in the stream

Function	get_device_media_clock_rate
Description	Get the rate of a media clock.
Type	`int get_device_media_clock_rate(int clock_num, int &rate)`
Parameters	`clock_num` the number of the media clock `rate` the rate of the clock in Hz

Function	set_device_media_clock_rate
Description	Set the rate of a media clock. Sets the rate of the media clock.
Type	`int set_device_media_clock_rate(int clock_num, int rate)`
Parameters	`clock_num` the number of the media clock `rate` the rate of the clock in Hz

Continued on next page

Type	avb_interface (continued)

Function	get_device_media_clock_state
Description	Get the state of a media clock.
Type	`int get_device_media_clock_state(int clock_num, enum device_media_clock_state_t &state)`
Parameters	`clock_num` the number of the media clock `state` the state of the clock

Function	set_device_media_clock_state
Description	Set the state of a media clock. This function can be used to enabled/disable a media clock.
Type	`int set_device_media_clock_state(int clock_num, enum device_media_clock_state_t state)`
Parameters	`clock_num` the number of the media clock `state` the state of the clock

Function	get_device_media_clock_source
Description	Get the source of a media clock.
Type	`int get_device_media_clock_source(int clock_num, int &source)`
Parameters	`clock_num` the number of the media clock `source` the output FIFO number to base the clock on

Continued on next page

Type	avb_interface (continued)

Function	set_device_media_clock_source
Description	Set the source of a media clock. For clocks that are derived from an output FIFO. This function gets/sets which FIFO the clock should be derived from.
Type	`int set_device_media_clock_source(int clock_num, int source)`
Parameters	`clock_num` the number of the media clock `source` the output FIFO number to base the clock on

Function	get_device_media_clock_type
Description	Get the type of a media clock.
Type	`int get_device_media_clock_type(int clock_num, enum device_media_clock_type_t &clock_type)`
Parameters	`clock_num` the number of the media clock `clock_type` the type of the clock

Function	set_device_media_clock_type
Description	Set the type of a media clock.
Type	`int set_device_media_clock_type(int clock_num, enum device_media_clock_type_t clock_type)`
Parameters	`clock_num` the number of the media clock `clock_type` the type of the clock

Copyright 2015 XMOS Ltd.

3.5 Core components

Function	avb_manager
Description	Core AVB API management task that can be combined with other AVB tasks such as SRP or 1722.1.

Type	`[[combinable]]`
	`void`
	`avb_manager(server interface avb_interface i_avb[num_avb_clients],`
	`   unsigned num_avb_clients,`
	`   client interface srp_interface ?i_srp,`
	`   chanend c_media_ctl[],`
	`   chanend(& ?c_listener_ctl)[],`
	`   chanend(& ?c_talker_ctl)[],`
	`   client interface ethernet_cfg_if i_eth_cfg,`
	`   client interface media_clock_if ?i_media_clock_ctl)`

Parameters	`i_avb[]` array of avb_interface server interfaces connected to clients of avb_manager `num_avb_clients` number of client interface connections to the server and the number of elements of i_avb[] `i_srp` client interface of type srp_interface into an `srp_task()` task `c_media_ctl[]` array of chanends connected to components that register/control media FIFOs `c_listener_ctl[]` array of chanends connected to components that register/control IEEE 1722 sinks `c_talker_ctl[]` array of chanends connected to components that register/control IEEE 1722 sources `i_eth_cfg` a client interface of type ethernet_cfg_if for Ethernet MAC configuration `i_media_clock_ctl` client interface of type media_clock_if connected to the media clock server

Type	avb_srp_info_t
Description	Struct containing fields required for SRP reservations.

Continued on next page

Copyright 2015 XMOS Ltd.

Fields	

	`unsigned stream_id`
	64-bit Stream ID of the stream
	
	`unsigned char dest_mac_addr`
	Stream destination MAC address.
	
	`short vlan_id`
	VLAN ID for Stream.
	
	`short tspec_max_frame_size`
	Maximum frame size sent by Talker.
	
	`short tspec_max_interval`
	Maximum number of frames sent per class measurement interval.
	
	`unsigned char tspec`
	Data Frame Priority and Rank fields.
	
	`unsigned accumulated_latency`
	Latency at ingress port for Talker registrations, or latency at end of egress media for Listener Declarations.
	
	`unsigned char failure_bridge_id`
	Bridge ID of bridge that changed Talker Advertise to Talker Failed.
	
	`unsigned char failure_code`
	Failure code associated with the failure bridge.

Type	**srp_interface**
Description	An interface used to register and deregister stream reservations via MSRP.
Functions	

Function	**register_stream_request**
Description	Used by a Talker application entity to issue a request to the MSRP Participant to initiate the advertisement of an available Stream.
Type	`short register_stream_request(avb_srp_info_t stream_info)`
Parameters	`stream_info` Struct of type avb_srp_info_t containing parameters of the stream to register

Continued on next page

Type	srp_interface (continued)

Function	deregister_stream_request
Description	Used by a Talker application entity to request removal of the Talker's advertisement declaration, and thus remove the advertisement of a Stream, from the network.
Type	`void deregister_stream_request(unsigned stream_id[2])`
Parameters	`stream_id` two int array containing the Stream ID of the stream to deregister

Function	register_attach_request
Description	Used by a Listener application entity to issue a request to attach to the referenced Stream.
Type	`short register_attach_request(unsigned stream_id[2], short vlan_id)`
Parameters	`stream_id` two int array containing the Stream ID of the stream to register `vlan_id` the VLAN ID associated with the stream. If 0 the current VID from the SRP domain will be used.

Function	deregister_attach_request
Description	Used by a Listener application entity to remove the request to attach to the referenced Stream.
Type	`void deregister_attach_request(unsigned stream_id[2])`
Parameters	`stream_id` two int array containing the Stream ID of the stream to deregister

Function	avb_srp_task
Description	SRP task that implements MSRP and MVRP protocols. Can be combined with other combinable tasks.

Continued on next page

<table>
<tr><td>Type</td><td><pre>[[combinable]]
void
avb_srp_task(client interface avb_interface i_avb,
   server interface srp_interface i_srp,
   client interface ethernet_rx_if i_eth_rx,
   client interface ethernet_tx_if i_eth_tx,
   client interface ethernet_cfg_if i_eth_cfg)</pre></td></tr>
<tr><td>Parameters</td><td>
<table>
<tr><td>i_avb</td><td>client interface of type avb_interface into the avb_manager() for API control of the stack</td></tr>
<tr><td>i_srp</td><td>server interface of type srp_interface that offers client tasks access to SRP reservation functionality</td></tr>
<tr><td>i_eth_rx</td><td>a client receive interface into the Ethernet MAC</td></tr>
<tr><td>i_eth_tx</td><td>a client transmit interface into the Ethernet MAC</td></tr>
<tr><td>i_eth_cfg</td><td>a client interface for Ethernet MAC configuration</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td>Function</td><td>avb_1722_1_maap_task</td></tr>
<tr><td>Description</td><td>A task that runs MAAP and 1722.1 ADP, ACMP and AECP protocols and interacts with the rest of the AVB stack.
Can be combined with other combinable tasks.</td></tr>
<tr><td>Type</td><td><pre>[[combinable]]
void
avb_1722_1_maap_task(otp_ports_t & ?otp_ports,
   client interface avb_interface i_avb,
   client interface avb_1722_1_control_callbacks i_1722_1_entity,
   fl_QSPIPorts & ?qspi_ports,
   client interface ethernet_rx_if i_eth_rx,
   client interface ethernet_tx_if i_eth_tx,
   client interface ethernet_cfg_if i_eth_cfg,
   chanend c_ptp)</pre></td></tr>
</table>

Continued on next page

Parameters		
	otp_ports	reference to an OTP ports structure of type otp_ports_t
	i_avb	client interface of type avb_interface into avb_manager()
	i_1722_1_entity	client interface of type avb_1722_1_control_callbacks
	qspi_ports	a reference to a Quad SPI flash ports structure
	i_eth_rx	a client receive interface into the Ethernet MAC
	i_eth_tx	a client transmit interface into the Ethernet MAC
	i_eth_cfg	a client interface for Ethernet MAC configuration
	c_ptp	chanend into the PTP server

Function	gptp_media_clock_server
Description	The media clock server.
Type	`void gptp_media_clock_server(server interface media_clock_if media_clock_ctl, chanend ?ptp_svr, chanend(& ?buf_ctl)[num_buf_ctl], unsigned num_buf_ctl, out buffered port:32 p_fs[], client interface ethernet_rx_if i_eth_rx, client interface ethernet_tx_if i_eth_tx, client interface ethernet_cfg_if i_eth_cfg, chanend c_ptp[num_ptp], unsigned num_ptp, enum ptp_server_type server_type)`

Continued on next page

Parameters	
	`media_clock_ctl` server interface of type media_clock_if connected to the avb_manager() task `ptp_svr` chanend connected to the PTP server `buf_ctl[]` array of links to listener components requiring buffer management `num_buf_ctl` size of the buf_ctl array `p_fs` output port to drive PLL reference clock `i_eth_rx` a client receive interface into the Ethernet MAC `i_eth_tx` a client transmit interface into the Ethernet MAC `i_eth_cfg` a client interface for Ethernet MAC configuration `c_ptp[]` an array of chanends to connect to clients of the ptp server `num_ptp` The number of PTP clients attached `server_type` The type of the PTP server (PTP_GRANDMASTER_CAPABLE or PTP_SLAVE_ONLY)

Function	**avb_1722_listener**
Description	An AVB IEEE 1722 audio listener thread. This thread implements a listener. It takes IEEE 1722 packets from the ethernet MAC and splits them into output FIFOs. The buffer fill level of these streams is set in conjunction with communication to the media clock server. This thread is dynamically configured using the AVB control API.

Type	`void`
	`avb_1722_listener(streaming chanend c_eth_rx_hp,`
	`chanend ?c_buf_ctl,`
	`chanend ?c_ptp_ctl,`
	`chanend c_listener_ctl,`
	`int num_streams,`
	`client push_if audio_output_buf)`

Continued on next page

Parameters	`c_eth_rx_hp` a high priority client receive interface into the Ethernet MAC `c_buf_ctl` buffer control link to the media clock server `c_ptp_ctl` PTP server link for retrieving PTP time info `c_listener_ctl` channel to configure the listener (given to `avb_init()`) `num_streams` the number of streams the unit will handle `audio_output_buf` a client interface to get a handle to push to the audio output buffer

Function	**avb_1722_talker**
Description	An AVB IEEE 1722 audio talker thread. This thread implements a talker, taking media input FIFOs and combining them into 1722 packets to be sent to the ethernet component. It is dynamically configured using the AVB control API.

Type	`void avb_1722_talker(chanend c_ptp,`
	`streaming chanend c_eth_tx_hp,`
	`chanend c_talker_ctl,`
	`int num_streams,`
	`client pull_if audio_input_buf)`

Parameters	`c_ptp` link to the PTP timing server `c_eth_tx_hp` a high priority client transmit interface into the Ethernet MAC `c_talker_ctl` channel to configure the talker `num_streams` the number of streams the unit controls `audio_input_buf` a client interface to get a handle to pull from the audio input buffer

3.6 Creating a gPTP server instance

All gPTP functions can be accessed via the gptp.h header:

```
#include <gptp.h>
```

Type	ptp_server_type
Description	The type of a PTP server. Can be passed into the ptp_server() function.
Values	PTP_GRANDMASTER_CAPABLE The port is capable of being both PTP Grandmaster and Slave role. PTP_SLAVE_ONLY The port is capable of PTP Slave role only.

Function	ptp_server
Description	This function runs the PTP server. It takes one logical core and runs indefinitely.
Type	void ptp_server(client interface ethernet_rx_if i_eth_rx, client interface ethernet_tx_if i_eth_tx, client interface ethernet_cfg_if i_eth_cfg, chanend ptp_clients[], int num_clients, enum ptp_server_type server_type)
Parameters	i_eth_rx a receive interface connected to the Ethernet server i_eth_tx a transmit interface connected to the Ethernet server i_eth_cfg a client configuration interface to the Ethernet server ptp_clients an array of channel ends to connect to clients of the PTP server num_clients The number of clients attached server_type The type of the server (PTP_GRANDMASTER_CAPABLE or PTP_SLAVE_ONLY)

3.7 Time data structures

Type	ptp_timestamp
Description	This type represents a timestamp in the gPTP clock domain with respect to the epoch.

Fields	`unsigned int seconds`
	The integer portion of the timestamp in units of seconds.
	
	`unsigned int nanoseconds`
	The fractional portion of the timestamp in units of nanoseconds.

3.8 Getting PTP time information

Type	ptp_time_info
Description	This type is used to relate local xCORE time with gPTP time. It can be retrieved from the PTP server using the ptp_get_time_info() function.

Type	ptp_time_info_mod64
Description	This structure is used to relate local xCORE time with the least significant 64 bits of gPTP time. The 64 bits of time is the PTP time in nanoseconds from the epoch. It can be retrieved from the PTP server using the ptp_get_time_info_mod64() function.

Function	ptp_get_time_info
Description	Retrieve time information from the PTP server. This function gets an up-to-date structure of type `ptp_time_info` to use to convert local time to PTP time.
Type	`void` `ptp_get_time_info(chanend ptp_server, ptp_time_info &info)`
Parameters	`ptp_server` chanend connected to the ptp_server `info` structure to be filled with time information

Function	ptp_get_time_info_mod64
Description	Retrieve time information from the PTP server. This function gets an up-to-date structure of type `ptp_time_info_mod64` to use to convert local time to PTP time (modulo 64 bits).

Continued on next page

Type	`void`
	`ptp_get_time_info_mod64(chanend ?ptp_server,`
	`ptp_time_info_mod64 &info)`

Parameters	`ptp_server` chanend connected to the ptp_server `info` structure to be filled with time information

Function	**ptp_request_time_info**
Description	This function requests a ptp_time_info structure from the PTP server. This is an asynchronous call so needs to be completed later with a call to ptp_get_requested_time_info().
Type	`void` `ptp_request_time_info(chanend ptp_server)`
Parameters	`ptp_server` chanend connecting to the ptp server

Function	**ptp_request_time_info_mod64**
Description	This function requests a ptp_time_info_mod64 structure from the PTP server. This is an asynchronous call so needs to be completed later with a call to ptp_get_requested_time_info_mod64().
Type	`void` `ptp_request_time_info_mod64(chanend ptp_server)`
Parameters	`ptp_server` chanend connecting to the PTP server

Function	**ptp_get_requested_time_info**
Description	This function receives a ptp_time_info structure from the PTP server. This completes an asynchronous transaction initiated with a call to ptp_request_time_info(). The function can be placed in a select case which will activate when the PTP server is ready to send.

Type	`void`
	`ptp_get_requested_time_info(chanend ptp_server,`
	`ptp_time_info &info)`

Continued on next page

Parameters	`ptp_server` chanend connecting to the PTP server `info` a reference parameter to be filled with the time information structure

Function	**ptp_get_requested_time_info_mod64**
Description	This function receives a `ptp_time_info_mod64` structure from the PTP server. This completes an asynchronous transaction initiated with a call to ptp_request_time_info_mod64(). The function can be placed in a select case which will activate when the PTP server is ready to send.
Type	`void ptp_get_requested_time_info_mod64(chanend ptp_server, ptp_time_info_mod64 &info)`
Parameters	`ptp_server` chanend connecting to the PTP server `info` a reference parameter to be filled with the time information structure

3.9 Converting Timestamps

Function	**local_timestamp_to_ptp**
Description	Convert a timestamp from the local xCORE timer to PTP time. This function takes a 32-bit timestamp taken from an xCORE timer and converts it to PTP time.
Type	`void local_timestamp_to_ptp(ptp_timestamp &ptp_ts, unsigned local_ts, ptp_time_info &info)`
Parameters	`ptp_ts` the PTP timestamp structure to be filled with the converted time `local_ts` the local timestamp to be converted `info` a time information structure retrieved from the PTP server

Function	**local_timestamp_to_ptp_mod32**

Continued on next page

Description	Convert a timestamp from the local xCORE timer to the least significant 32 bits of PTP time. This function takes a 32-bit timestamp taken from an xCORE timer and converts it to the least significant 32 bits of global PTP time.
Type	`unsigned local_timestamp_to_ptp_mod32(unsigned local_ts, ptp_time_info_mod64 &info)`
Parameters	`local_ts` the local timestamp to be converted `info` a time information structure retrieved from the PTP server
Returns	the least significant 32-bits of PTP time in nanoseconds

Function	**ptp_timestamp_to_local**
Description	Convert a PTP timestamp to a local xCORE timestamp. This function takes a PTP timestamp and converts it to a local 32-bit timestamp that is related to the xCORE timer.
Type	`unsigned ptp_timestamp_to_local(ptp_timestamp &ts, ptp_time_info &info)`
Parameters	`ts` the PTP timestamp to convert `info` a time information structure retrieved from the PTP server.
Returns	the local timestamp

Function	**ptp_timestamp_offset**
Description	Calculate an offset to a PTP timestamp. This function adds and offset to a timestamp.
Type	`void ptp_timestamp_offset(ptp_timestamp &ts, int offset)`
Parameters	`ptp_timestamp` the timestamp to be offset; this argument is modified by adding the offset `offset` the offset to add in nanoseconds

Copyright 2015 XMOS Ltd.

APPENDIX A - Known Issues

- Firmware images made with xTIMEcomposer 14.0.3 will corrupt the factory boot image on firmware upgrade due to a bug in the quad flash library.

APPENDIX B - TSN library change log

B.1 7.0.0

- Library changed to new structure and tools 14 compatibility added
- Support added for new version 3 of Ethernet library and Gigabit Ethernet on xCORE-200
- Support added for new version 2 of I2S/TDM library
- Audio buffering performance improvements for higher channel count applications
- Support added for 1722.1 Enitity Firmware Upgrade (EFU) using new Quad SPI flash library
- Support added for 1722.1 ACMP Fast Connect
- Support added for 1722.1 AECP sample rate change via GET/SET_SAMPLING_RATE and GET/SET_STREAM_FORMAT commands
- Support added for 1722.1 AECP GET/SET_SIGNAL_SELECTOR commands
- Current value fields in 1722.1 descriptors are now updated to reflect the current set value
- Bug fix for gPTP number of lost reponses not being reset on link up event
- Unimplemented 1722.1 commands now return the correct NOT_IMPLEMENTED status response
- Resolved bug in 1722.1 ACMP disconnection caused by stream info not being zeroed

B.2 6.3.1

- Bug fix for excessive Talker AVTP presentation time being absorbed in the FIFOs for a short period at start
- Fixes regression in bad gPTP pdelay follow up detection
- Bug fix for reported base audio clusters in AEM stream descriptors

B.3 6.3.0

- MEDIA_CLOCK_SOURCE bit now set in 1722.1 ADP Talker Capabilities
- 1722.1 GET_COUNTERS command added for CLOCK_DOMAIN descriptor
- Minor bug fix in gPTP where multiple pdelay responses were not triggering AVnu specific behaviour
- Change to SRP interface to allow SRP to control the joining of VLANs via MVRP
- Max frame size reported by SRP changed to reflect the current set sample rate instead of the max supported
- Changes to dependencies:
  - sc_ethernet: 2.3.2rc0 -> 2.3.3beta0
    - Change to rounding of Qav slope calculation

B.4 6.2.2

- PTP clock accuracy is now reported to be within 25 ns by BMCA
- PTP offset scaled log variance is now set to the correct unkown value (0x436A) per IEEE P802.1AS-Cor-1
- Grandmaster timeBaseIndicator and lastGmFreqChange parameters are now set in the PTP sync follow up TLV
- Pdelay exchanges are marked invalid and asCapable reset if the delay is measured as negative
- Fixed issue with lost PTP messages being counted twice, causing a premature asCapable reset

B.5 6.2.1

- Fix potential parallel usage violation on PTP client function

Copyright 2015 XMOS Ltd.

B.6 6.2.0

- Ethernet AVB server now configures auto-negotiation on the PHY
- State of MAAP and PTP now reset on link up of single port configuration
- Minor bug fixes to 1722.1 descriptors and commands

B.7 6.1.2

- Various minor SRP compliance fixes

B.8 6.1.1

- Various gPTP AVnu compliance fixes (June 2014 report)

B.9 6.1.0

- Support added for sw_avb_lc single port reference design
- gptp.c moved to XC
- Misc M*RP AVnu compliance fixes
- gPTP AVnu compliance fixes
- Changes to dependencies:
  - sc_ethernet: 2.3.1rc0 -> 2.3.2rc0
    - Updated timestamp adjustements for LAN8710A PHY to realistic values

B.10 6.0.7

- Changes to dependencies:
  - sc_ethernet: 2.3.0rc0 -> 2.3.1rc0
    - Fix invalid inter-frame gaps.

B.11 6.0.6

- Reverted change to 1722 introduced in 6.0.3 that caused media clock to unlock

B.12 6.0.5

- Bug fix to prevent compile error when Talker is disabled
- Update to 1722 MAAP to fix non-compliance issue on conflict check

B.13 6.0.4

- Updates design guide documentation to include AVB-DC details
- SPI task updated to take a structure with ports
- Bug fix on cd length of acquire command response
- Added EFU mode and address access flags to ADP capabilities

B.14 6.0.3

- Firmware upgrade functionality changed to support START_OPERATION commands to erase the flash
- Several SRP bug fixes that would cause long connect/disconnection sequences to fail

Copyright 2015 XMOS Ltd.

B.15 6.0.2

- Interim release for production manufacture

B.16 6.0.1

- VLAN ID is now reported via 1722.1 ACMP
- Fixed XC pointer issue for v13.0.1 tools

B.17 6.0.0

- First release supporting daisy chain AVB
- Refactoring sw_avb modules into sc_avb

B.18 5.2.0

- Numerous updates to support xTIMEcomposer v12 tools, including updated sc_ethernet
- 1722.1 Draft 21 support for ADP, ACMP and a subset of AECP including an AEM descriptor set
- Old TCP/IP based Attero Tech application replaced with a 1722.1 demo
- Added ability to arbitrarily map between channels in sinked streams and audio outputs
- 1722 MAAP rewritten to optimise memory and improve compliance to standard
- AVB status API replaced with new weak attribute hooks
- Support added for CS2100 variant of PLL
- sc_xlog printing removed, replaced with XScope
- Support removed for XDK/XAI, XC-2 and XC-3 dev kits
- Application support removed for Open Sound Control

B.19 5.1.2

- PTP fix to correct step in g_ptp_adjust (commit #1548fa5ce7)
- Software support added for CS2100 PLL.
- Media clock recovery PID tuned to decrease settle time and amplitude of oscillations
- Fixes to app_xr_avb_lc_demo to work with channel counts < 8
- Transport stream interface
- 1722/61883-4 packet encapsulation
- Update to ethernet and tcp package dependencies

B.20 5.1.1

- Field update module added
- I2S slave functionality added

B.21 5.1.0

- 802.1Qat support
- Partial (beta) 1722.1 support
- Clock recovery corrections for 8kHz and >48kHz
- 1722 packet format corrections
- 1722 timestamp corrections
- Stream lock/unlock more predictable
- Test harnesses for various features

Copyright 2015 XMOS Ltd.

- SRP state machine corrections
- SRP state machine drives stream transmission

B.22 5.0.0

- New control API
- 1722 MAAP support
- Standard updates
- Optimizations
- See design guide for new release details

B.23 4.1.0

- Move to new build system

B.24 4.0.0

- Fixed missing functionality in media clock server
- Small changes media server API - see demos for examples
- Optimized audio transport for local listener streams
- Major rewrite, many internal APIs changed, overall performance improvements
- Added gigabit ethernet support
- Added flexible internal routing (local streams) with simplified API, framework is much more powerful for many-channel applications
- Rewritten audio_clock_recovery as more flexible media_clock_server
- Added demos for audio interface board
- Added 8-channel TDM audio interface
- Added uip IP/UDP/TCP server for adding configuration layer
- Various bug fixes

XMOS®

Copyright © 2015, 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.