System design for audio record and playback with a computer using FireWire

Michael SCHÜEPP
BridgeCo AG
michael.schuepp@bridgeco.net

Rolf "Day" KOCH
BridgeCo AG
rolf.koch@bridgeco.net

Rene Widtmann
BridgeCo AG
rene.widtmann@bridgeco.net

Klaus Buchheim
BridgeCo AG
klaus.buchheim@bridgeco.net

Abstract

This paper describes the problems and solutions to enable a solid and high-quality audio transfer to/from a computer with external audio interfaces and takes a look at the different elements that need to come together to allow high-quality recording and playback of audio from a computer.

Keywords

Recording, playback, IEEE1394

1 Introduction

Computers, together with the respective digital audio workstation (DAW) software, have become powerful tools for music creation, music production, post-production, editing. More and more musicians turn to the computer as a tool to explore and express their creative ideas. This tendency is observed for both, professional and hobby musicians. Within the computer music market a trend towards portable computers can be observed as well. Laptops are increasingly used for live recordings outside a studio as well as mobile recording platforms. And, with more and more reliable system architectures, laptops/computers are also increasingly used for live performances.

However making music on a computer faces the general requirement to convert the digital music into analogue signals as well as to digitize analogue music to be processed on a computer.

Therefore the need for external, meaning located outside of the computer housing, audio interfaces is increasing.

The paper describes a system architecture for IEEE1394 based audio interfaces including the computer driver software as well as the audio interface device.

1.1 System Overview

When discussing the requirements for an audio interfaces it is important to understand the overall system architecture, to identify the required elements and the environment in which those elements have to fit in.

The overall system architecture can be seen in the following figure:

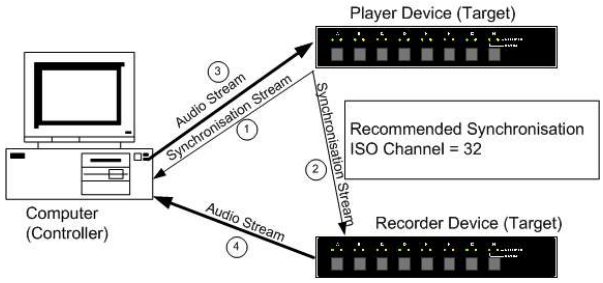

Illustration 1: Computer audio system overview

The overall system design is based on the following assumptions:

- A player device receives m audio channels (connection 3), from the computer, and plays them out. In addition it plays out data to i MIDI Ports. The data (audio and MIDI) sent from the computer are a compound stream.
- A recorder device records n audio channels and sends the data to the computer (connection 4). In addition it records data from j MIDI Ports and sends their data to the computer. The data (audio and MIDI) sent from the computer are a compound stream.
- A device can send or receive a synchronisation stream (connection 1 and 2). Typically one of the Mac/PC attached audio devices is the clock master for the synchronisation stream.

The player and recorder functions can be stand-alone devices or integrated into the same device.

1.2 What is there?

In the set-up above the following elements already exist and are widely used:

On computers:

- Digital audio workstation software such as Cubase and Logic with their respective audio APIs (ASIO and CoreAudio)
- Operating system (here Windows XP and Apple Mac OS X)
- Computer hardware such as graphic cards, OHCI controllers, PCI bus etc.

On audio interface:

- Analogue/Digital converters with I2S interfaces

All of the above elements are well accepted in the market and any solution to be accepted in a market place needs to work with those elements.

1.3 What is missing?

The key elements that are missing in the above system are the following:

1. The driver software on the computer that takes the audio samples to/from a hardware interface and transmits/receives them to/from the audio APIs of the music software.
2. The interface chip in the audio interface that transmits/receives the audio samples to/from the computer and converts them to the respective digital format for the converter chips.

The paper will now focus on these two elements and shows, what is required for both sides to allow for a high-quality audio interface. In a first step we will look at the different problems we face and then at the proposed solutions.

2 Issues to resolve

To allow audio streaming to/from a computer the following items have to be addressed:

2.1 Signal Transport

It has to be defined how the audio samples get to/from the music software tools to the audio interface. The transfer protocol has to be defined as well the transfer mode.

Additionally precautions to reduce the clock jitter during the signal transport have to be taken. Also the latency in the overall system has to be addressed.

2.2 Synchronization

In a typical audio application there are many different clock sources for the audio interface. Therefore we have the requirement to be able to synchronize to all of those clock sources and to have means to select the desired clock source.

2.3 Signal Processing

For low latency requirements and specific recording set-up, it is required to provide the capability for additional audio processing in the audio interface itself. An example would be a direct monitor mixer that mixes recorded audio onto the audio samples from the computer.

2.4 Device Management

Since we have the requirement to sell our product to various different customers as well as for various different products in a short time-to-market, it is necessary to provide a generic approach that reduces the customization efforts on the firmware and driver. Therefore it was necessary to establish a discovery process that allows the driver at least to determine the device audio channels and formats on-the-fly. This would reduce the customization efforts significantly. Therefore means to represent the device capabilities within the firmware and to parse this information by the driver have to be found.

2.5 User Interface

It must be possible to define a user interface on the device as well as the computer or a mix of both. Therefore it is required to provide means to supply control information from both ends of the system

2.6 Multi-device Setup

It is believed that it must be possible to use several audio interfaces at the same time to provide more flexibility to end-users. This puts additional requirements on all above issues. To avoid sample rate conversion in a multi-device setup it is mandatory to allow only a single clock source within the network. This requirement means to select the master clock within the multi-device setup as well as to propagate the clock information within the network so that all devices are slaved to the same clock.

3 Resolution

Very early in the design process it was decided to use the IEEE1394 (also called FireWire) standard

[6] as the base for the application. The IEEE1394 standard brings all means to allow isochronous data streaming, it is designed as a peer-to-peer network and respective standards are in place to transport audio samples across the network. It was also decided to base any solution on existing standards to profit from already defined solutions. However it was also clear that the higher layers of the desired standards were not good enough to solve all of our problems. Therefore efforts have been undertaken to bring our solutions back to the standardization organisations.

Overall the following standards are applied to define the system:

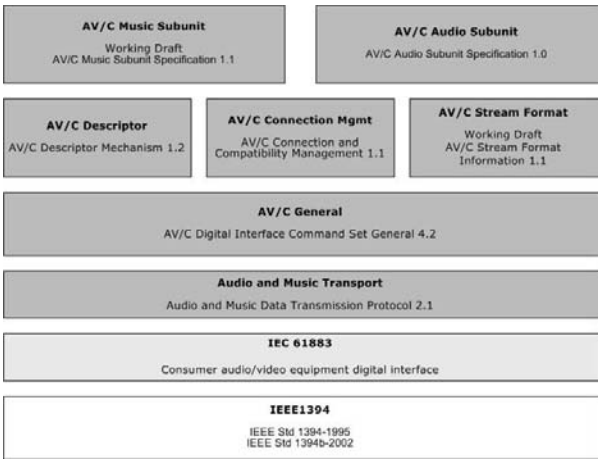

Illustration 2: Applied standards

As we can see, a variety of standards on different layers and from different organisations are used.

3.1 Signal Transport

The signal transport between the audio interfaces and the computer is based on the isochronous packets defined in the IEEE1394 standard. The format and structures of audio packets is defined in IEC61883-6 standard [6], which is used here as well. However a complex audio interface requires transmitting several audio and music formats at the same time. This could e.g. be PCM samples, SPDIF framed data and MIDI packets. An additional requirement is synchronicity between the different formats. Therefore it was decided to define a single isochronous packet, based on an IEC 61338-6 structure that contains audio and music data of different formats. Such a packet is called a compound packet. The samples in such a packet are synchronized since the time stamp for the packet applies to all the audio data within the packet.

IEC 61883-6 packets that contain data blocks with several audio formats are called compound packets. Isochronous streams containing compound packets are called compound streams. Compound streams are used within the whole system to transfer audio and music data to/from the audio interface.

The IEC 61883-6 standard defines the structure of an audio packet being sent over the IEEE1394 bus. The exact IEC 61883-6 packet structure can be found in [6].

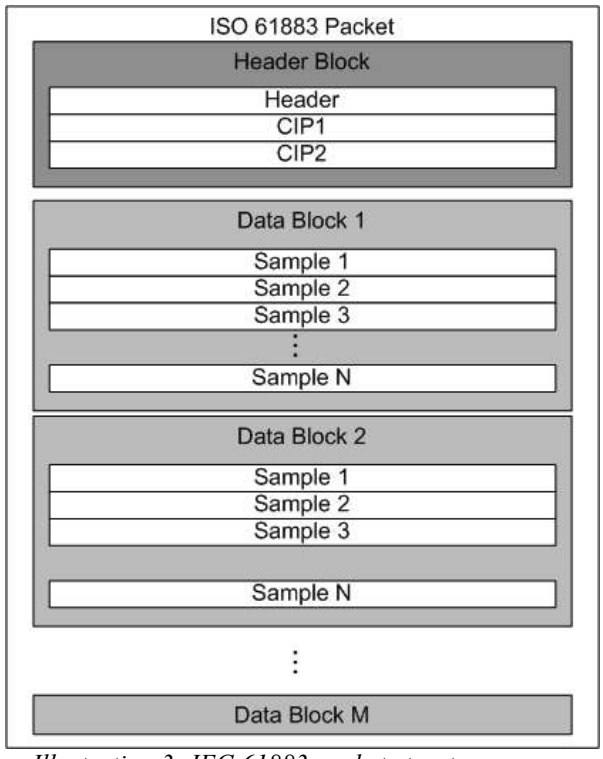

Illustration 3: IEC 61883 packet structure

The blocking mode is our preferred mode for data transmission on the IEEE1394 bus. In case data is missing to complete a full packet on the source side empty-packets are being sent. An empty-packet consists of only the header block and does not contain data. The SYT field in the CIP1 header is set to 0xffff.

The following rules to create an IEC 61883 compliant packet are applied:

- A packet always begins with a header block consisting of a header and two CIP header quadlets.
- (M) data blocks follow the header block.
  Table 1 defines M and its dependency on the stream transfer mode.
- In blocking mode, the number of data blocks is constant. If insufficient samples are available to fill

all the data blocks in a packet, an empty packet will be sent.

- In non-blocking mode, all the available samples are placed in their data blocks and sent immediately. The number of data blocks is not constant.

Sampling Frequency (FDF) [kHz]	Blocking Mode	Non-Blocking Mode
32	8	5-7
44.1	8	5-7
48	8	5-7
88.2	16	11-13
96	16	11-13
176.4	32	23-25
196	32	23-25

Table 1: Number of data blocks depending on the sampling frequency

The header information and structure for an isochronous IEC61883 packet is defined as follows:

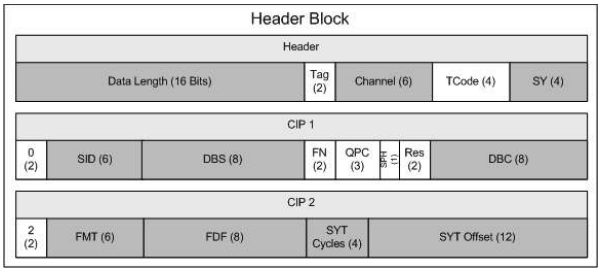

Illustration 4: IEC 61883 packet header

Table 2 describes the different elements and their definition within the packet header:

Field	Description
Data Length	Length in bytes of the packet data, including CIP1 and CIP2 header.
Channel	Isochronous channel to which the packet belongs.
SY	"System" Can be of interest if DTCP (streaming encryption) is used.
SID	"System Identification" Contains the IEEE1394 bus node id of the stream source.
DBS	"Data Block Size" Contains information about the number of samples belonging to a data block.
DBC	"Data Block Count" Is a counter for the number of data blocks that have already been sent. It can be used to detect multiply sent packets or to define the MIDI port to which the sample belongs.
FMT	"Format" The format of the stream. For an audio stream this field is always 0x10.
FDF	The nominal sampling frequency of the stream. See [3.1] for value definition.
SYT Cycles	This field, in combination with the SYT Offset field, defines the point in time when the packet should be played out. Value range: 0 – 15
SYT Offset	This field, in combination with the SYT Cycles field, defines the point in time when the packet should be played out. Value range: 0 – 0xBFF

Table 2: IEC 61883 packet header fields

Within an IEC 61883 packet, the data blocks follow the header. For the data block structure we applied the AM824 standard as defined in 6.

An audio channel is assigned to a slot within the data block:

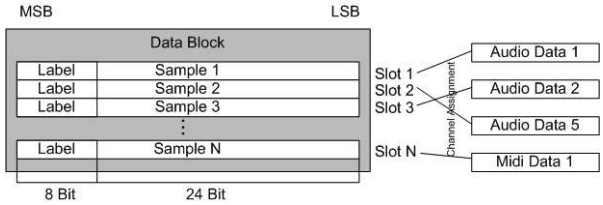

Illustration 5: Data block structure

The following rules apply to assemble the data blocks:

1. The number of samples (N) within a data block of a stream is constant.
2. The number of samples should be even (padding with ancillary no-data samples see 6)
3. The label is 8 bits and defines the sample data type
4. The sample data are MSB aligned
5. The channel to slot assignment is constant

The channel to data block slot assignment is user defined. To create a compound packet, a data structure had to be defined to place the different audio and music formats within the data blocks. No current standard defines the order in which such user data must be placed within a data block. The current standard 6 simply provides a recommended ordering. We applied this recommendation for our application and made the following data block structure mandatory to stream audio and music information:

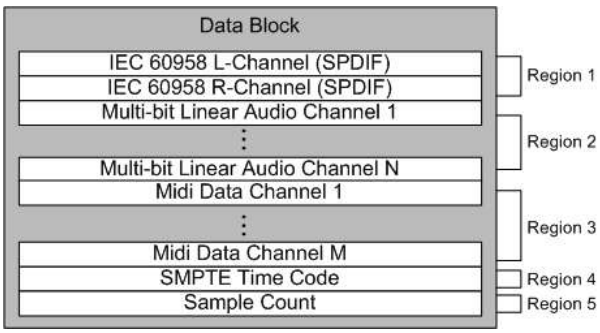

Illustration 6: User data order

The following rules are applied to create the data blocks of a compound packet:

1. A region within a data block always contains data with the same data type
2. Not every region type must exist in a packet

The following region order is used:

1. SPDIF: IEC 60958 (2 Channels)
2. Raw Audio: Multi-Bit Linear Audio (N Channels)
3. MIDI: MIDI Conformant Data
4. SMTPE Time Code
5. Sample Count

MIDI data is transferred, like audio data, within channels of a compound data block. Because of the low transfer rate of one MIDI port, data of 8 MIDI ports, instead of just one, can be transferred in one channel. As shown in Illustration 7, one data part of a MIDI port will be transferred per data block and channel. This method of splitting data is called multiplexing.

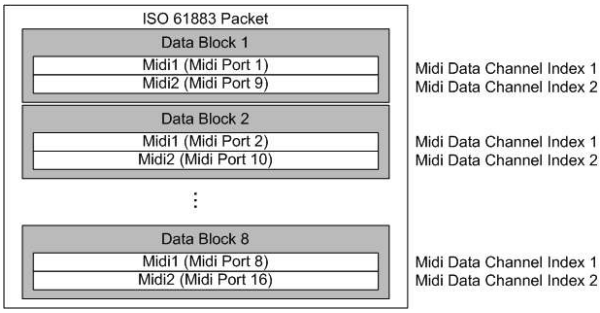

Illustration 7: MIDI data multiplexing

For the two main elements in this system, the driver and the interface processor, it is required to assemble the data packet correctly when sending data as well as to receive and disassmble the packets. Based on the dynamics of the system with different channel counts and formats the final packet structure has to be derived from the configuration from the interface such as number of channels per format and sample rate. Overall in the system it is required to keep the latency low so the framing and deframing processes have to be done as efficiently as possible.

3.2 Synchronization

The system synchronization clock for an IEEE1394 based audio interface can normally be retrieved from four different sources:

1. The time information in the SYT header field of an incoming isochronous stream.
2. The 8KHz IEEE1394 bus Cycle Start Packet (CSP).
3. The time information from an external source like Word Clock or SPDIF.
4. A clock generated by a XO or VCXO in the device.

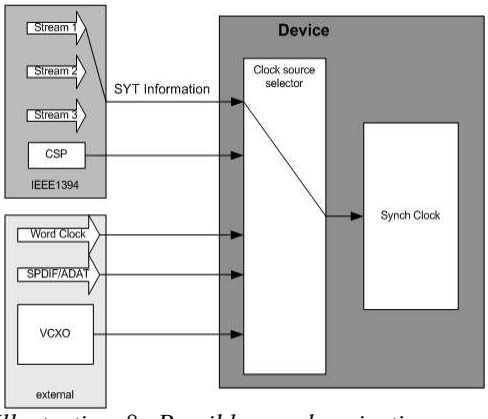

Illustration 8: Possible synchronization sources for an IEEE1394 based audio interface

3.3 Signal Processing

The specific architecture of the BridgeCo DM1x00 series is designed to enable signal processing of the audio samples once they have been deframed:

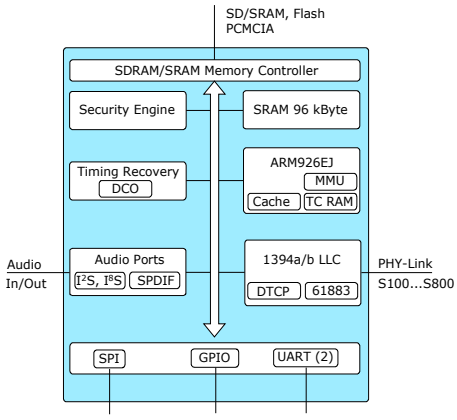

Illustration 9: Architecture of the BridgeCo DM1000 processor

Since the on-board ARM processor core can access every audio sample before it is either sent to the audio ports or sent to the IEEE1394 link layer,

it is possible to enable signal processing on the audio interface. Typical applications used in this field are direct monitor mixers, which allow mixing the inputs from the audio ports with the samples from the IEEE1394 bus before they are played out over the audio ports:

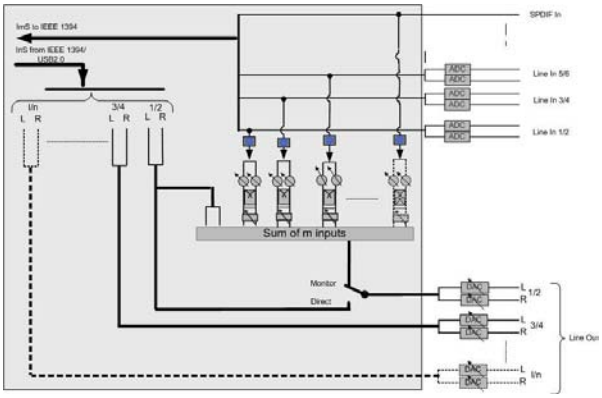

Illustration 10: Direct monitor mixer

3.4 Device Management

The device management plays a key role within the overall concept. To implement a generic approach it is absolutely necessary for the driver software to determine the device capabilities such as number of audio channels, formats and possible sample rates on-the-fly. Based on that information, the driver can expose the respective interfaces to the audio software APIs. The device management and device discovery is normally defined in the AV/C standards from the 1394TA. To achieve our goals, several audio and music related AV/C standards have been used:

- AV/C Music Subunit Specification V1.0
- AV/C Stream Format Information Specification V1.0
- AV/C Audio subunit 1.0

However the standards did not provide all means to describe and control devices as intended. Therefore two of above standards, AV/C Music Subunit and AV/C Stream Format Information are currently updated within the 1394TA organization, based on the experience and implementations from Apple Computer and BridgeCo.

Using AV/C, the driver has the task to determine and query the device for its capabilities whereas the device (meaning the software running on the device) needs to provide all the information requested by the driver to allow to stream audio between the driver and the audio interface. As soon as the device is connected to the driver via IEEE1394, a device discovery process is started. The discovery process is based on a sequence of AV/C commands exchanged between the driver and the device. Once this sequence is executed and the device is recognized as an AV/C device, the driver starts to query the device for the specific device information. This can either be done using proprietary AV/C commands or by parsing an AV/C descriptor from the device.

Within the device, the control flow and different signal formats are described with an AV/C model. The AV/C model is a representation of the internal control flow and audio routing:

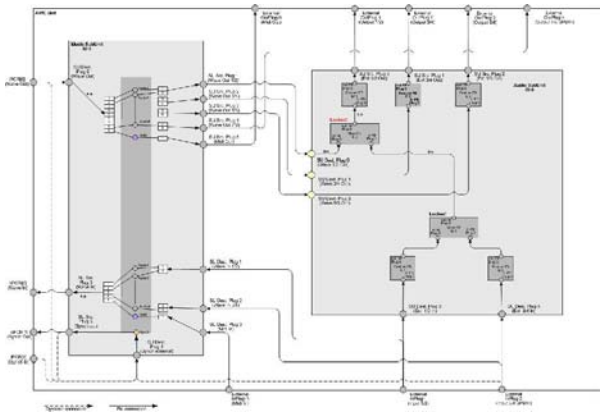

Illustration 11: Typical AV/C model

The following rules are applied to an AV/C model:

1 Fixed connections must not be changed. Every control command requesting such a change must be rejected.
2 Every unclear requested connection (like Ext. IPlug0 → Destination Plug) must be rejected.

In the AV/C model we also see the control mechanism for the direct monitor mixer which can be controlled over AV/C e.g. to determine the levels of the different inputs into the mixer.

Based on this information, the driver software can now determine the number of audio channels to be sent to the device, the number of audio channels received from the device, the different formats and expose this information to the audio streaming APIs such as ASIO or CoreAudio and expose all available sample rates to a control API.

3.5 User Interface

In the described system of an audio interface connected to a computer there are two natural points to implement a user interface:

- A control panel on the computer
- Control elements on the audio interface

Depending on customer demands and branding, different OEMs have different solutions/ideas of a control interface. In our overall system architecture we understand that it is impossible to provide a generic approach to all possible configurations and demands. Therefore we decided to provide APIs that can easily be programmed. Those APIs have to be present on both sides, on the driver as well as in the device software:

- On the driver side we expose a control API that allows direct controlling the device as well as to sent/use specific bus commands.
- On the device we have several APIs that allow to link in LEDs, knobs, buttons and rotaries.

The commands from the control API of the driver are send as AV/C commands or vendor specific AV/C commands to the device. The control API provides the correct framing of those commands whereas the application utilizing the control API needs to fill in the command content.

On the device side, those AV/C commands are received and decoded to perform the desired action. This could e.g. be different parameters for the mixer routines or being translated into SPI sequences to control and set the external components.

Next to the UI information, the device might need to send additional information to a control panel, e.g. peak level information of different audio samples, rotary information for programmable functions etc. For those high-speed data transfers, the AV/C protocol can be too slow since it e.g. allows a timeout of about 10 msec before sending retries. Within that time frame, useful and important information might already be lost. Therefore we deployed a high-speed control interface (HSCI) that allows the device to efficiently provide information to the driver. With the HSCI, the device writes the desired information into a reserved space of the IEEE1394 address space of the device. This allows the application on the computer doing highly efficient asynchronous read requests to this address space to obtain the information. Since the information structure is such, that no information gets lost, the PC application can pull the information when needed.

3.6 Multi-device Setup

A multi-device setup is normally needed when users like to use multiple device to gain a higher channel count or use different formats that are not all available in a single audio interface:

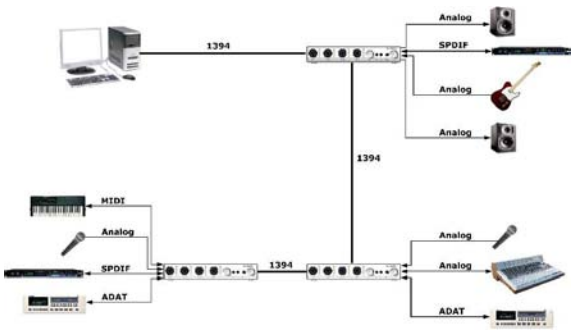

Illustration 12: Multi-device configuration

If multiple audio interfaces are connected to a computer, we face certain limitations, either imposed by the operating system, the audio software on a computer, APIs etc. :

1. ASIO and CoreAudio based audio software (e.g. Cubase) can only work with a single device.
2. To avoid sample rate conversion, only a single clock source can be allowed for all devices.
3. All devices need to be synchronised over the network

To overcome those limitations the driver software has to provide the following capabilities:

1. Concatenate multiple IEEE1394 devices into a single ASIO or CoreAudio device for the audio software application.
2. Allow selecting the clock source for each device.
3. Ability to transmit and send several isochronous streams.
4. Ability to supply the same SYT values to all transmitted isochronous streams

The device itself needs to provide the following functions:

1. Expose available clock sources to the driver software
2. Generate correct SYT values for outgoing isochronous streams

To synchronise multiple devices on a single clock source, which might be an external clock source for one of the devices, the following clocking scheme is used:

1. A device must be selected as clock master. This can be the computer as well.
2. If an external device is the clock master, the driver software synchronizes to the SYT time stamps within the isochronous stream from the clock master device.
3. The driver copies the received SYT time stamps from the clock master stream to its outgoing stream for all other devices.

4. All external devices expect the clock master use the SYT time stamps of their incoming isochronous stream as a clock source.

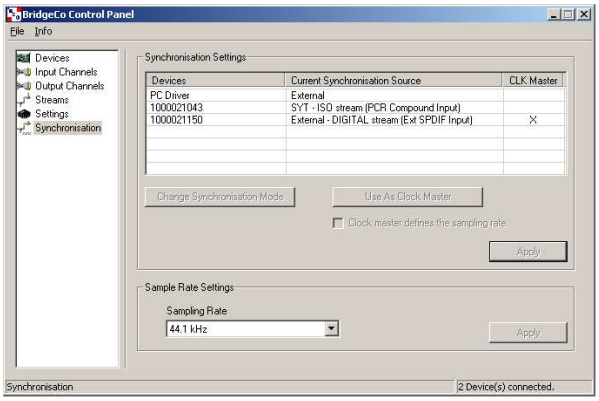

Illustration 13: Example for a multi-device clock setup

Now, all devices are slaved across the IEEE1394 network to a single clock source. This avoids to use word clock or similar schemes to synchronize multiple devices.

Based on above configurations, each device needs to be able to synchronize on the SYT time stamps of the isochronous packets to work in such an environment.

Therefore the following features are required for the driver software:

1. Concatenate multiple devices into a single device on the audio API (ASIO or CoreAudio)
2. Allow synchronizing on the SYT time stamps from a selected isochronous stream
3. Generate correct SYT time stamps for all isochronous streams based on the received SYT time stamps
4. Parse AV/C model to determine all available clock sources on a device
5. Allow to set the clock source for each device

For the chip/firmware combination on the audio interface the following requirements must be met:

1. Allow to synchronize to SYT time stamps
2. Expose all available clock sources on the device in the AV/C Model
3. Allow external control of the clock sources via AV/C

4 FreeBob Project

Currently, there only exists drivers for the Windows and MacOS X platform, which are of course not free. The FreeBob project is trying to implement a complete free and generic driver for Linux based systems. The project is still in early stages, though first (hardcoded) prototypes are working. For further informatin please visit the website of the project (http://freebob.sf.net).

5 Conclusion

Due to the wide spectrum of interpretation within the available standards a very tight cooperation between all elements in the system is necessary. In developing such a system, it is not enough just to concentrate on and develop one element within the system. Instead it is rather required to start from a system perspective, to design an overall system concept that is initially independent of the different elements. Then, in a second step the individual tasks for each element can be defined and implemented. BridgeCo has chosen this approach and with over 20 different music products shipping today has proven that the concept and the system design approach leads to a success story. BridgeCo also likes to express its gratitude to Apple Computer, which has been a great partner throughout the design and implementation process and has provided valuable input into the whole system design concept.

6 Reference

[1] IEEE Standard 1394-1995, IEEE Standard for a High Performance Serial Bus, IEEE, July 22 1996

[2] IEEE Standard 1394a-2000, IEEE Standard for a High Performance Serial Bus—Amendment 1, IEEE, March 30 2000

[3] IEEE Standard 1394b-2002, IEEE Standard for a High-Performance Serial Bus—Amendment 2, IEEE, December 14 2002

[4] TA Document 2001024, “Audio and Music Data Transmission Protocol” V2.1, 1394TA, May 24 2002

[5] TA Document 2001012, “AV/C Digital Interface Command Set General Specification”, Version 4.1, 1394TA, December 11, 2001

[6] TA Document 1999031, “AV/C Connection and Compatibility Management Specification”, Version 1.0, 1394TA, July 10, 2000

[7] TA Document 1999025, “AV/C Descriptor Mechanism Specification”, Version 1.0, 1394TA, April 24 2001

[8] TA Document 2001007, “AV/C Music Subunit”, Version 1.0, 1394TA, April 8 2001

[9] TA Document 1999008, “AV/C Audio Subunit Specification”, Version 1.0, 1394TA, October 24 2000

[10] IEC 61883-6, Consumer audio/video equipment - Digital interface - Part 6: Audio and music data transmission protocol, IEC, October 14 2002