Taking the 'Sting' Out of Evolving Digital Audio Networks

Gregory F. Shay
The Telos Alliance
Cleveland Ohio

Abstract - *Digital audio networking is here to stay. It is the state of the art, and it is the future. But with which standard? Today there are multiple competing systems, and work on a compatibility standard is underway. Knowing a digital audio networking standard is the future that will be, what should equipment vendors and user do in the meantime to get 'over the hump' of these next few years? This paper will examine this problem by breaking down where some of the real design issues are. Given the difference between the systems, what is the most that could need to be changed? How can an equipment manufacturer today designing to use one particular digital audio network interface potentially leave the door open to the future standard even though that is not yet settled? Broad categories of equipment design issues will be considered, like: network capability, synchronization, stability and design of local clocking hardware, stream formats, memory for buffering, mips for protocol parsing, real time requirements on hardware and software.*

GOAL STATEMENT

How can you design products today that will not need too extensive modifications to interoperate with different digital audio networks, and the future X192 AES standard whatever that settles to? There are signposts already pointing the way to the end goal, if you know how to see them.

We are at that point in the history of digital audio networks that will be looked back on as just before standardization happened.

The answer is to break down the implementation of an audio network interface design, and examine each part, and see how each part may vary. We will find that there is not too much variation, and with some careful planning and selection of design features, it will be possible to support many of the existing audio networks and be ready for X192.

BOTTOM UP APPROACH

There are two ways to break down and describe a system: top down or bottom up. There are many white papers [1], books [2] and current and pending standards documents [3] that describe the design of audio network from generally the top down. To draw a contrasting approach, I will describe a bottom up interface design. At the end, we will find that there are a collection of 'pieces of the puzzle' that when put together, form the end goal of a more future proof design.

PARTS OF AN AUDIO NETWORK INTERFACE DESIGN

- Timebase recovery (synchronization)
- Audio packet handling
- Audio data buffering
- Buffer synchronization
- Serial to parallel, parallel to serial
- Connection management, discovery and control.

TIMEBASE RECOVERY

Timebase recovery is the essential process of having all devices on the audio network use a common sample rate and common knowledge of sample phase, which in the case of packet buffered blocks, where the block boundaries (or 'frames') are at the same time. Examples of timebase sync are IEEE-1588 and the earlier Livewire sync method, which are similar in function, but vary in some details. All these methods use the mechanism of "clock packets" which contain a timestamp of the master clock at the moment they were sent, and are accurately timestamped with the local clock counter at the time of arrival.

The first thing to note is that accurate timebase recovery is directly related to, and essential for, low latency (low delay) of the audio going over the network. This is because the latency of an audio network is not the time of flight of the packets over the wire or packets though the switches (although over very long distances the speed of light does eventually dominate), the latency is determined by the amount of buffering used, and the minimum amount of buffering that can be used is determined by the time inaccuracy, the wander and jitter of the transfer of the data in time, which is all coordinated by the timebase as recovered at each device relative to each other. So less accurate timebase recovery implementation would require more buffering, and would be only able to serve a higher latency audio network, but an accurate timebase recovery can serve both low and high latency audio network designs.

If you want a very low latency audio network (for example the 750 microsecond "Livestream" mode of Livewire), this corresponds to an audio sample block of 12 samples per packet at 48Khz sample rate, and an audio frame / packet rate of 4Khz (250 microseconds). To do this in practice, we use a timebase recovery accuracy of +/- 5 microseconds, which is about ¼ of a sample. This very tight timebase accuracy is so the device can meet the schedule of taking input audio packets, performing required DSP, and transmit the output packets before the deadline of the next

250 microsecond frame. Any error in the local timebase relative to any other device takes away time that you have available to do processing of the audio. You can pipeline the audio frames' input / process / output sequence at the expense of added latency, but timebase error still threatens the ability to know the moment of time when all the input packets are present at the shifted execution time, and threatens making output transmissions by the frame deadline.

It is possible to make use of a less accurate local timebase, even a purely software operating system task scheduler, as for example in purely software IP audio drivers for Windows / Mac OS / Linux are available to replace hardware sound cards. But these software-only drivers cannot generate the lowest latency type of streams. Example: A typical OS task execution time accuracy is +/- 10 milliseconds (or more), with some of the better more modern OS's down to a few milliseconds. These require larger buffers at the receiving end to cover the transmit timing error, and this buffering causes the audio latency, in this the example just given, on the order of tens of milliseconds.

To reiterate: an audio network interface product with a more accurate timebase can always interface with and easily meet the wider frame deadline of a higher latency audio network with bigger buffers. But the converse is not true, a device with less accurate timebase cannot interface with a lower latency audio network.

Puzzle Piece #1: Design for high timebase accuracy, which will not limit you from interfacing to any audio network (high or low latency).

The more accurate the time stamps are, the better, and less time measurement error is introduced into the timebase recovery process. This is analogous to less noise input to a time recovery Phase Lock Loop (PLL). The timestamping can be done in software (part of the packet receive interrupt), but the interrupt response latency then becomes timing error. (You may not want a high priority interrupt per packet, anyway. There can be many, many packets, hundreds of thousands of packets per second coming to your device in a low latency multichannel audio network system.)

The solution is to use an FPGA to monitor the incoming packets, and either timestamp all of them or have a recognition filter that finds the clock packets by pattern matching and performs the time stamp, and makes this time stamp information available to software which will be running a software PLL timebase recovery algorithm.

IEEE-1588 enabled Ethernet hardware contains this same kind of timestamping mechanism. Although with the present state of the art, you must pay close attention to the hardware features that are implemented in the chip you choose, and the features that the PTP (Precision Time Protocol) stack expects. These must match. Also depending on the assumptions made, an existing PTP stack implementation may or may not produce for you a usable hardware clock that you can run the sample clock of analog/AES converters from (if you are interfacing).

Puzzle Piece #2: Use an FPGA based incoming packet timestamping module. Stamp all packets, or be able to have its recognition filter configured to accommodate any of the sync clock packet formats.

Sync recovery is essentially a PLL that steers a local programmable oscillator, in order to match the external clock master. This sync method using relatively infrequent clock packets means that you don't get a very large amount of sync 'steering' information. In PLL terms, this means the bandwidth of the PLL feedback loop is *low*. When your PLL feedback bandwidth is low you benefit from the most stable local oscillator that you are using. A PLL must steer itself to overcome 3 things: to track the remote master clock, to overcome the time noise (clock packet time jitter) and to compensate for the drift of the local oscillator.

Hitching a ride from the digital cell phone industry it seems, a class of inexpensive TCXO (Temperature compensated crystal oscillator) of about +/- 2ppm are available for very reasonable cost around $2. The net system performance this enables is worth this cost, to have a stable local timebase reference as compared to the difficulties created by the standard variety +/- 20 to 50ppm crystals. A clock generator design that give simultaneously both stability, low jitter, and a wide range of high resolution programmability, which will be a solution for any audio network solution at any sample rate, is the Direct Digital Synthesis (DDS) method. This can be a chip, or implemented in FPGA.

Note: if you are to have analog audio converters in your product, or have AES digital local inputs and outputs with sample rate converters (SRC's) to external AES sample rates, you use this master DDS generated timebase as your master overclock for the converters and the sample clock to the 'inside' of the AES SRC's.

Of course, the very nature of having an audio network allows you to eliminate analog or digital interfaces from the product design, if you so desire. In this case, of a pure audio network device, you still have the need of an accurate local timebase to generate the essential moment in time (an interrupt to the processor) to know when to take the audio packets that came in, perform signal processing, and generate audio packets out to the network. Any error in the moment of time that this process is triggered, represents being possibly late delivering output audio packets and in a low latency system, could underflow the small buffers of the next device in the audio path. The existing crystal associated with the CPU may drift too much, and the scheduling time resolution of the existing CPU resources may be too coarse, so that the accurate TCXO plus DDS may still be desired.

Puzzle Piece #3: Use a local clock oscillator design consisting of a stable TCXO with a DDS architecture.

The remaining part of audio network timebase recovery is essentially a "software PLL" that uses the differences between the received timestamps of the clock packets and the master clock time value carried inside the clock packet, to carefully speed up or slow down your steerable local clock (TXCO + DDS). The details of how this is done vary for different audio network sync schemes, but the point here is that this should be software. While it may be possible to implement more of the PLL control algorithm in hardware ASIC or FPGA, the observation is that the rate of clock packets is comparatively low (1 packets per second for IEEE-1588 up to 30 pps for Livewire) , so there is no real need not to handle this algorithmic complexity in software. In fact, the standard IEEE-1588 Precision Time Protocol (PTP) stack can do much of this work for you. The catch is the assumptions made by the PTP stack as to how the local timebase is generated and steered. A standard PTP stack may only be assuming that it is steering the kernel task scheduling mechanism, not a physical clock. But this can be made to work with the right modifications to the PTP hardware driver.

Puzzle Piece #4: Use a software PLL for timebase recovery.

AUDIO PACKET HANDLING

The assumption is being made that the actual network interface (Ethernet) is an embedded interface in a general purpose CPU. Ethernet interfaces themselves are highly evolved (multiple generations of design technology since the 1980's), very efficient using fast block DMA offloading practically all load from the CPU, and very low cost, practically zero cost, being included in the core platform of the CPU. For economical audio network interface design, we want to take advantage of all of the above factors and use the CPU embedded Ethernet interface, and avoid adding as much external special function hardware as possible.

Multichannel uncompressed audio involves a relatively large amount of data throughput. If the data word and bit format of the audio samples and sample blocks do not exactly match the internal format of your processor, or if the audio channels are not packed or interleaved in a format compatible with your processor, this may represent a very significant CPU MIPS load, to unpack, possibly bit shuffle, and repack the audio network packet format.

A CPU is very efficient moving blocks of data, either using a single cycle repeated 'movem' type opcode, or with DMA hardware assistance. FPGA's are very efficient at byte and bit manipulation, packing and unpacking.

So the solution is to let the CPU be limited to doing block moves of audio data to and from the payloads of audio packets. The audio data block moves go through a high bandwidth interface to and from an FPGA. The FPGA does all of the bit level manipulation as needed, and channel interleaving, adapting to or from the details of the audio network data format.

Puzzle Piece #5: Let the host CPU do only block moves of audio data, the CPU never touches audio samples individually. Use an FPGA to adapt / convert / repack or construct the detailed audio network packet payload format.

Must there be an FPGA?

No, but consider the following: If your product has analog or AES local I/O, it is going to need multichannel parallel to serial / serial to parallel conversion, which is best performed in an FPGA. The timestamping mechanism described above wants to live in an FPGA (unless you are strictly using IEEE-1588 enabled hardware for IEEE-1588 sync only). The high accuracy timebase requires logic. This is a tradeoff between more cost for more CPU cycles as opposed the FPGA cost. If your design needs an FPGA for any of the other reasons, then partition the audio sample bit and packing functionality into the FPGA as well.

AUDIO DATA BUFFERING

Buffering may be done in the host CPU's memory and in the FPGA. Or alternatively all buffering in the FPGA. Remember for low latency audio, the buffers are small, so not so much FPGA RAM is needed for buffering. For high latency larger buffers, CPU host memory is more economical, holding audio in blocks in the format of the audio network packet payload.

Puzzle Piece #6: Audio data buffering can be in the host CPU memory (as packet payload formatted blocks), in the FPGA, both, or only in FPGA.

BUFFER SYNCHRONIZATION

Buffer synchronization means knowing where in the buffer to put the latest arriving input data, and from where in the buffer to take out audio data to be sent at present instant of time, as a function of absolute time and /or the immediate relative time to network frames.

Buffer sync can be tricky. Pointers have to be initialized, the data transfer process has to be initiated at just the correct time, then increment the pointers exactly to match the data flow. Finally the buffer pointer logic has to be able to recover from abnormal conditions such as missing packets, momentary network disconnect or malfunction, and transient conditions of timebase recovery (either initially or transfer of clock mastership events). If you want to give the acid test to a audio network product design, in the middle of audio streaming, unplug and replug the network cable a couple of times in quick succession, and see if the audio recovers completely and cleanly in a relatively short time (a fraction of a second, ideally). If not, this is a sign of buffer synchronization logic problems.

While it is possible to implement buffer sync logic in FPGA, the number of special cases point to a software implementation, that guides or command FPGA buffer pointers as needed.

It is to be noted here that there can be relatively major differences between different audio networks in this seemingly straightforward topic of buffer sync logic. The differences stem from the fundamental policy defined by the audio network, of when the samples of a given audio stream are to be played out.

One policy is to begin to play out the audio stream as soon as possible, with as little buffering as possible as determined by the timebase accuracy of the receiving device. This has the advantages of simplicity of definition, and the ability to automatically adapt to long distance transmission where the time of flight of the packets is not negligible, but has the disadvantage that the absolute time that a given audio sample is used (or output) at the destination is not precisely defined. In practice this latter disadvantage is unimportant if the latency is low, and audio streams closely related to one another in phase (i.e. stereo pairs or surround channel sets) share the same packets in the same stream.

An alternate audio network policy is to play out the audio stream samples according to the correct absolute time as defined by the timebase information in the audio packets themselves. This has the advantage of precise definition of the use or output time of all audio samples and channels, regardless of how they are grouped together in shared packets or not.

Precise differences in these two policies, amounts to different audio buffer synchronization logic. For an audio network interface design to not be limited to one policy or the other, and to be able to handle the more complex special cases, the audio buffer logic should be in software.

Puzzle Piece #7: Implement audio buffer synchronization logic in software.

SERIAL / PARALLEL INTERFACING

If the audio network device contains local analog or digital AES interfaces, or contains DSP functions that are best served their input/output data on multichannel serial ports, multichannel serialization is most straightforward to implement in FPGA logic.

Note that at all of these serial interfaces the timebase is still defined by the audio network. In other words, the bit clocks and frame signals are derived from the master DDS, which is steered by the software PLL to come into sync with the audio network master clock. A local interface, or DSP device cannot be the master clock for any of the interfaces which end up going to an audio network. An audio network requires all interfaces to be synchronous to its timebase.

If an interface or a device is fundamentally on its own sample rate or timebase, and cannot be made to use the audio network sample rate and timebase, audio sample rate converters (SRCs) will have to be used. Every SRC has two clock inputs, one for each side of the converter. In the case of an audio network, one side of the SRC will be connected to the audio network interface logic, and be a slave to the network interface logic master clock. The other side of the SRC can face the outside, non-network interface, and be a slave to that interface's master clock.

Puzzle Piece #8: Use sample rate converters (SRCs) to interface all local audio interfaces or devices that cannot be slaved to the overall audio network sample rate and timebase.

CONNECTION MANAGEMENT, DISCOVERY AND CONTROL

An audio network accomplishes the connection and transport of many thousands of channels of audio in an efficient and timely manner. Imagine a giant, low cost, easy to automate, patch panel. But as soon as you have a complex operation, using thousands of channels of audio, and you care about what the end result sounds like, you are going to have to practically manage all of those channels.

Some brief definitions:

Connection management is the process used to make audio routes from point to point, once the user decides which input channel of audio is desired to appear at each output. The different connection management methods may range from using network broadcast addressing (no management at all, flood the entire network), to multicast (freely transmit all channels, receiver selects which channel), to unicast (arrange each transmit / receive pair on a one to one basis).

Discovery is finding out what audio channels are present, and what material they contain. This can range from a fixed prearranged order, on the fly data gathering, or a central registration and database lookup scheme.

Control is initiating or ending the audio connections if the endpoints need to be turned on and off, optional gain controls, initiating channel changes, etc.

The good news is that an audio network, being based on a data network, easily handles all of these required management tasks without any additional cables, wires, or interfaces. The bandwidth for connection management, discovery and control is a small fraction of the network bandwidth consumed by the audio, so with proper use of Quality of Service (QOS), can coexist. The bad news is that these methods are farther in the future from standardization than the audio part of the audio network itself.

Puzzle Piece #9: Be prepared to be flexible with software implementations of connection management, discovery and control.

CONCLUSION

By breaking down the implementation of an audio network interface design into these flexible parts as we have examined here, they can be seen to be capable of not needing too extensive modifications to interoperate with many of the existing audio digital audio networks, and the future X192 AES standard whatever that settles to. By managing the design risk, products can be brought to market today without having to wait for future standards to settle, and with confidence that the future standards will be able to be supported.

APPENDIX

CASE EXAMPLE: WHAT IT TOOK TO EVOLVE LIVEWIRE EQUIPMENT TO BE COMPLIANT WITH RAVENNA

Livewire is a digital audio network system from Telos Systems, established over 10 years and in use in over 3000 radio studios worldwide. In the spring of 2012, it was announced that Livewire devices would embrace and support Ravenna, a contemporary digital audio network, because the value for all present and future Livewire customers is increased with interoperability. Support is also given by for the AES X192 initiative to define a interoperable digital audio network standard. Examining the details of what it took to make Livewire equipment able to fully support Ravenna, is informative and instructive to the subject of this paper.

Livewire and Ravenna both used a common IETF RTP (Real Time Protocol) L24 audio stream packet format, which defines all of the contents of the audio packets. What could be wrong? There were three main hurdles:

Hurdle #1: Ravenna relied on a specific use of the source timestamp in the audio packet, Livewire did not.

Solution: 1a) For compatibility with existing Livewire devices, Ravenna defined a new 'Livewire compatible relaxed mode', ignoring this value.

1b) The software driver in newer designed Livewire products will fill in the source timestamp value, to be fully Ravenna compliant.

Hurdle #2 : Ravenna uses RTCP messages in the audio multicast channel to control audio streams. Livewire assumes streams are 'always on' and does not use nor expect RTCP packets.

Solution 2) a software update for Livewire devices to safely filter out and ignore the extra RTCP packets.

Hurdle #3 : Different sync methods. Ravenna uses IEEE-1588, Livewire uses a pre-1588 sync method.

Solution 3a) As long as both systems externally slaved to the same master timebase, both are in sync with each other.

3b) Newer designed Axia products now can use IEEE-1588 directly, consisting of modifications to the FPGA clock packet timestamp filter, and modifications to the timebase recovery synchronization software PLL routine.

Summary: The required modifications to make the Livewire interface equipment also compatible with Ravenna were relatively minor, able to be changes in FPGA design and software drivers. This bodes well for being able to adapt to future standards like AES X192, when it settles.

REFERENCES

[1] Various authors. White papers at www.axiaaudio.com/tech

[2] Church, Steve G. (2009). *Audio Over IP: Building Pro AoIP Systems with Livewire.* Focal Press.

[3] Audio Engineering Society, *X192 High-performance streaming audio over IP interoperability*. www.x192.org