Software defined ultra-low latency Video-over-IP system with compression

Siegfried Foessel, Thomas Richter

Fraunhofer IIS, Fraunhofer Institute for Integrated Circuits IIS, Germany

Written for presentation at the

SMPTE 2021 Annual Technical Conference & Exhibition

Abstract. *Traditional Video-over-IP implementations in software either result in higher latency due to the processing time required for compression, or in high bandwidth required when transmitted as an uncompressed stream. With UHD-1 and UHD-2 video, this is even more of a challenge, as the uncompressed stream requires high-performance Ethernet networks or dedicated hardware implementing the compression. In some cases however, a software implementation and standard COTS equipment are beneficial to allow higher flexibility. With JPEG XS, a mezzanine compression codec was developed that can also be implemented as an ultra-low latency system in software. However, special attention must be paid how the data for processing is distributed across multiple threads, how large the number of threads is, in order to achieve optimal latency. Using a case study for UHD-1 Video-over-IP, this paper explains how such a system can be implemented with COTS components, which software architecture is necessary and how far the latency can be reduced.*

Keywords. *Video over IP; Compression; Ultra-low latency; Software defined codec; JPEG-XS; SMPTE ST 2110; 10GbE*

Takeaways.

- Video over IP systems with software compression can have a very low latency
- Multi-threading for compression allows adaptation to target processor
- JPEG XS can guarantee CBR and a balanced load on the line

Type of paper. Case Study

Target audience. Senior Managers, Engineers

Introduction

In recent years, video production facilities have moved from SDI lines to IP-based networks. This enables a higher flexibility in the workflow and the connection between devices, allowing also to directly connect workflows to cloud services. To preserve this high flexibility COTS equipment and software-based workflows are recommended.

The challenge of using Video-over-IP in combination with COTS equipment and software-based workflows is to ensure low latency along with high throughput in software-based compression engines. This is especially important for live productions, where monitoring, control and remote operations are taken place. As long as the data pipeline is not completely switched to IP-based workflow, also hybrid systems with SDI input, output and IP transmission have to be considered that add additional delays to the workflow. In this paper, we cover both challenges, the switch between hybrid workflows and the latency of software based compression systems.

JPEG XS as new mezzanine codec

With JPEG XS a tailored new mezzanine image codec was standardized at ISO [1] that allows an algorithmic latency of typically 16 lines during encoding or decoding, in some special modes even lower. In real world this latency can be realized with special hardware implementations (ASIC, FPGA), but such approaches eliminate the advantage of using standard computer equipment.

The low algorithmic latency of JPEG XS in general is achieved by an asymmetrical wavelet transformation and a “rolling buffer” rate allocation with a limited look-ahead window. For software implementations, however, such a design alone is not sufficient as it defines essentially a serial codec that is unable to achieve the throughput for UHD-1 or UHD-2 images. Thus, the architecture requires additional refinement to allow parallelization. Luckely, JPEG XS is an inherently parallel design as it allows splitting the source images after the wavelet transform into slices of 16 lines high and calculating the rate distortion for these slice independently. A more detailed explanation of the features and operation of JPEG XS can be found in [3].

For this paper, it is important to note that slices can be processed independently by individual software threads. Taking an UHD-1 video (3840x2160 Pixel) as example, the image can be splitted into 2160/16 = 135 slices. In order to avoid a high level of administration effort due to the thread management, several consecutive slices can be combined into slicegroups that are processed by the same thread, thereby reducing the number of threads. This paper explains the right balance between number of slices in a slicegroup, the number of threads and related latency.

Case Study Setup

For this case study, we use a typical setup with components integrated in servers for a video processing pipeline.

Figure 1 shows the typical standard design for a software based processing pipeline.

- A video source generates a video stream on a SDI interface.
- Server 1 is equipped with a SDI capture card and a 10GbE Ethernet interface at the ouput. The data transfer for IP transmission is based on RTP packets with RFC 9134 [6] and SMPTE ST 2110-22 [4]. Inside there will be some video processing and encoding to further transmit the data to the next processing node. As we only want to evaluate the I/O and coding latencies, the video processing unit is omitted.
- The reverse happens at Server 2: data are received on an IP interface, decoded and then output via an SDI playout card.

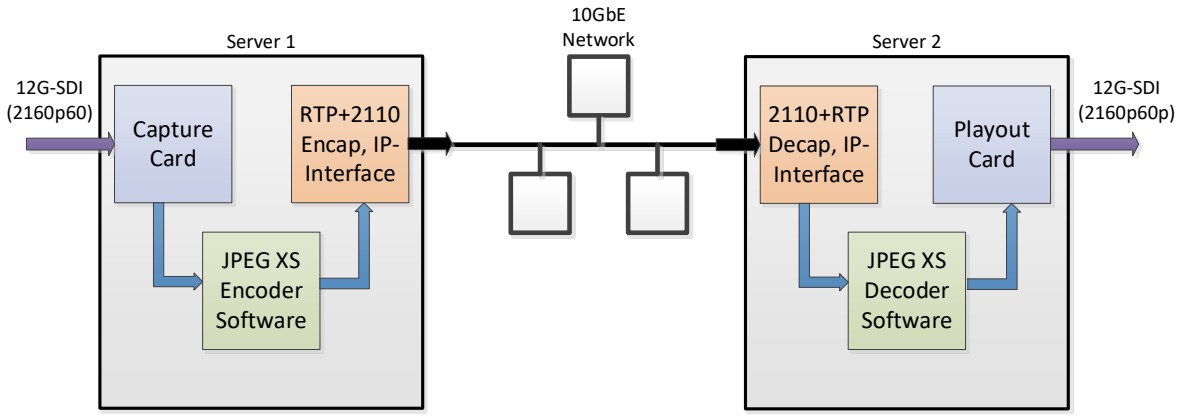

Figure 1. Typical I/O pipeline for video processing

In a first step (Design 1) we calculate the latency using previous I/O cards and a sequential processing, in a second step we propose a new design (Design 2) with recent I/O cards and a staggered processing pipeline.

Design 1 (Standard design)

The typical operation is frame based and sequential (Figure 2) [2]. That means, the SDI input card captures first one complete video frame before a data transfer to the PC memory is initiated. The data transfer is typically done via Direct Memory Transfer (DMA) methods. For a UHD-1 image the data transfer on a PCIe 2.0 x8 interface can last up to a third of a frame at 60Hz. Then an encoding is initiated, which can also last up to one frame dependent on the performance of the computer. It cannot last longer as otherwise the thoughput of the server is not sufficient. At the end of the data processing in server 1, the data are packetized and sent over an IP stack on the Ethernet interface to the network. The overall latency is about 5 frames.

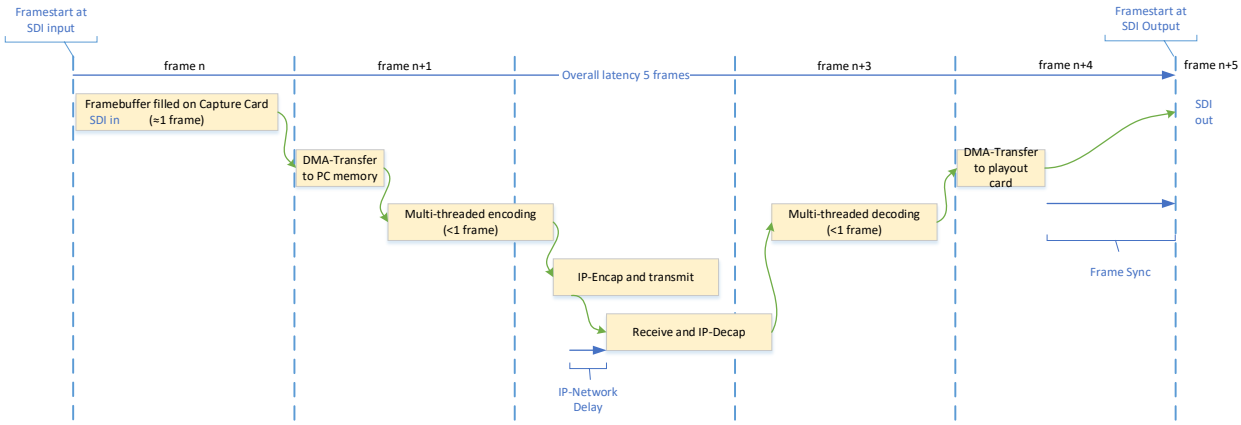

Figure 2. Dataflow and timing in Design 1

The latencies are summarized in table 1:

Orign	Source	Delay
SDI Capture Card	Framebuffer filled before DMA transfer is initiated	≈1 frame
DMA transfer to PC memory	Data per UHD-1 frame: 20MByte Example: BMD Decklink 4k extreme 12G PCIe 2.0x8 theoretical 4GByte/s	Frame duration @ 60p: 16,6ms Transfer time min: 20/4000s = 5ms ≈1/3 frame
Frame based encoding	Multi-threading, depend on performance of computer	< 1frame
IP transmission	Data collecting, Packetizing, Traffic Shaping	≈1/8 frame
IP-Network	Transport, Routing Depend on distance	Very low in LAN
IP reception	Packet collecting, Sorting, Data Buffer filling	≈1/8 frame + 1 frame due to traffic shaping on sender side
Frame based Decoding	Multi-threading, depend on performance of computer	< 1frame
DMA transfer from PC memory to SDI card	Same as capture card	≈1/3 frame
Sync at output	Waiting for frame sync	<1 frame

Table 1. Latencies in Standard Design

Design 2 (Ultra low latency Design)

As mentioned before, JPEG XS has the concept of slice based processing, with slices being coded independently of each other. This can be used for parallelized coding and decoding of the image [5]. For an ultra-low latency design, the SDI input and output cards should allow subframe DMA transfers. A subframe is ideally an integer number of slices, e.g. 3 slices with 3*16=48 lines. This allows transferring first lines (e.g. one slicegroup with three slices) to the PC memory while capturing the rest of the frame.

A first encoding thread (Thread 1) can start if the first two slicegroups were transferred to the PC memory. The next thread (Thread 2) can be started after the next DMA transfer is finished and so on. After the first slicegroup is coded by Thread 1 it can be packetized and sent to the IP interface. One or more separate threads for IP interface handling allow parallel working of encoding and transmission. This realizes a kind of data waterfall concept or staggered design.

Note: Some encoding threads can be faster than others and may finish their work head of time, even if started later. If an ordered transport of packets is desired, a synchronization point or data reordering has to be established for the packetizing, data transmission and traffic shaping to the IP interface. The JPEG XS RTP interface allows, however, also out-of-order transport that puts the burden of re-assembling the data stream in order to the receiver. This is possible as JPEG XS RTP packets carry in their header their target designation in the bitstream.

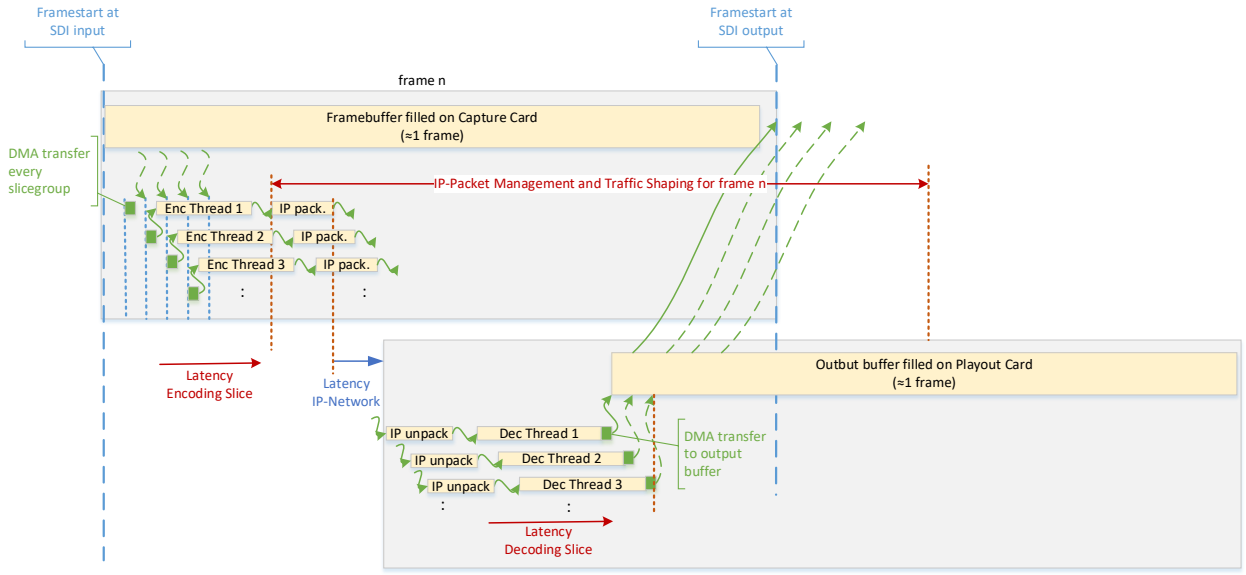

Figure 3. Dataflow and timing in staggered design

Latencies can be calculated as defined in table 2 (example with slicegroup of three slices):

Orign	Source	Delay
SDI Capture Card	Two slicegroups before DMA transfer starts	Lines received/total lines/fps= 2*48/2250/60s = 0,71ms
DMA transfer to PC memory	Data per slicegroup: 460kByte Example: Deltacast 12G PCIe 3.0x8 theoretical 7,8GByte/s	Transfer time min: 0.461/7800s = 0,06ms
Slicegroup based encoding	Per Slicegroup at 2bpp target rate Example: AMD Threadripper 3970X (3.7GHz)	Average Encoding time per slicegroup: 1,72ms
IP transmission	Data collecting, Packetizing, Traffic Shaping	
IP-Network	Transport, Routing Depend on distance	Depends on transmission channel, very low in LAN
IP reception	Packet collecting, Sorting, Data Buffer filling	
Slicegroup based decoding	Per Slicegroup at 2bpp target rate Example: AMD Threadripper 3970X (3.7GHz)	Average Decoding time per slicegroup: 1,15ms
DMA transfer from PC memory to SDI card	Same as capture card	Transfer time min: 0.461/7800s = 0,06ms
Sync at output	Waiting for frame sync	<1 frame

Table 2. Latencies in Staggered Design

The delay from the SDI input to the decoder output without IP handling can be added up to: 0,71ms+0,06ms+1,72ms+1,15ms+0,06ms= 3,7ms which offers enough reserve for the IP Handling and the synchronization up to the next frame.

Experimental results

Performance tests were executed on an AMD Threadripper 3970X to calculate the average throughput per core for encoding and decoding.

For the Fraunhofer JPEG XS SDK 4.1 this ended up in the following values:

AMD Threadripper per core with **virtual 1 GHz**			
Average coding time per slice in ms for **422** content			
type	$t_{encoding}$ **(ms)**	$t_{decoding}$ **(ms)**	$Number_{Slices}$
2160p/2bpp	2,122	1,411	135
2160p/3bpp	2,384	1,688	135
2160p/4bpp	2,534	1,916	135

Table 3. Processing times forJPEG XS (Fraunhofer SDK4.1)

A calculation shows that one core on an AMD Threadripper with 3.7Ghz clock needs a processing time per slice for 2bpp target rate of 2.122ms/3.7(GHz)=574us, for decoding 1.411ms/3.7(GHz)=382us. Compared to the input data rate on a 12G-SDI interface (16/2250/60s=118us), this results in a processing factor to real-time uncompressed transmission of 574/118 = 4.9 for encoding and 382/118=3.3 for decoding.

Put in another way, more than 5 cores should be used for encoding of UHD-1 content to 2bpp and more than 4 cores for decoding. These are optimal values, in real systems enough headroom should be given for processing. With three slices per slicegroup 45 threads have to be processed for a UHD-1 image. By pinning multiple threads to the same core (in the example with 9 processing cores), e.g. Thread 1, 10, 19, 28, 37 to Core 1; Thread 2, 11, 20, 29, 38 to Core 2 and so on, a good load balancing can be achieved.

Implementation

Fraunhofer implemented a ULL JPEG XS system based on the above principles, using AMD threadrippers for both encoding and decoding. Source material is currently generated by a UHD camera. To ensure proper synchronization of encoder and decoder, a clock generator creates a tri-level sync signal the camera and the decoder genlocks to. JPEG XS implementations at encoder and decoder use both a single slice per slicegroup, and utilize 16 threads both for encoding and decoding. This design ensures lowest possible latency as each slice is processed by its own processor core, though creates some additional overhead due to the overlapping wavelet transformation. For transport, an out-of-order RTP based JPEG XS stream is used, following the latest IETF specification. That is, encoder threads push encoded data out to the network as soon as it becomes available, and the decoder needs to re-assemble the packets in proper order.

As the decoder needs to run behind the encoder, an additional delay is added to the received tri-level sync signal and the start of frame generated by the play-out card. It is easy to see that this delay must be at least as large as the end-to-end (encoder-in to decoder-out) latency.

Measurements with an IP analyzer (Tektronic prism) showed that this system reaches an end-to-end latency of 180 lines.

Conclusion

We prototyped a UHD-1 JPEG-XS software-based ultra-low latency system. By careful design and allocating of processing threads on a multicore CPU, latencies far below 1 frame can be achieved.

This can be realized by splitting the workload on a slice or slicegroup granularity to multiple threads. The number of threads and the latency can be adapted to the available number of CPU cores and the performance per core.

For the data transfer on the IP network, a slice based packaging and out-of-order transmission was choosen. This enables direct packaging in IP packets and immediate dispatch after the encoding task is completed. At the receiver side, the slices will be reordered and ingested into the image frame.

As a result, in a synchronized system (with a frame synchronizer on the output of the decoder), one frame delay between input and output of the transmission chain is achievable.

References

[1] ISO/IEC 21122, “Information technology — JPEG XS low-latency lightweight image coding system”

[2] K. Itakura, M. Miyazaki, S. Fößel and M. V. Dorpe, "JPEG-XS Codec Adapted to 8K and ST 2110," SMPTE 2020 Annual Technical Conference and Exhibition, 2020, pp. 1-8, doi: 10.5594/M001904.

[3] A. Descampe et al., "JPEG XS--A New Standard for Visually Lossless Low-Latency Lightweight Image Coding," in Proceedings of the IEEE, doi: 10.1109/JPROC.2021.3080916.

[4] "ST 2110-22:2019 - SMPTE Standard - Professional Media Over Managed IP Networks: Constant Bit-Rate Compressed Video," in ST 2110-22:2019 , vol., no., pp.1-6, 14 Aug. 2019, doi: 10.5594/SMPTE.ST2110-22.2019.

[5] Thomas Richter, Joachim Keinert, Siegfried Fößel, "Parallelization and multi-threaded latency constrained parallel coding of JPEG XS," Proc. SPIE 11137, Applications of Digital Image Processing XLII, 111370J (6 September 2019); https://doi.org/10.1117/12.2526917

[6] IETF, "RTP Payload Format for ISO/IEC 21122 (JPEG XS)", https://datatracker.ietf.org/doc/rfc9134/

