Design and Performance of a Multi-Stream MPEG-I System Layer Encoder/Player[1]

J.A. Boucher, Z.Yaar, E.J. Rubin, J.D. Palmer, and T.D.C. Little

Department of Electrical, Computer and Systems Engineering
Boston University, Boston, Massachusetts 02215, USA
(617) 353-8042, (617) 353-6440 fax
tdcl@bu.edu

MCL Technical Report 01-07-1995

Abstract–Current efforts in the area of MPEG-I audio/video synchronization have been limited to single audio, single video applications. The MPEG-I specification includes provisions for the interleaving of up to 16 separate video streams with up to 32 distinct audio streams. This paper explores the possible uses of this capability as well as the design of a robust encoder and playback system. Perceived shortfalls within the specification are discussed including the usefulness of time stamps and the lack of sequence start and end codes within the audio stream format. We also describe our implementation of a software-only MPEG-I encoder/player set and describe its performance under various configurations.

Keywords: Video compression, MPEG-I System Layer, software video playout, multi-stream synchronization.

[1]In *Proc. Multimedia Computing and Networking*, IS&T/SPIE Symposium on Electronic Imaging Science and Technology, Vol. 2417, February 1995, pp. 435-446.

1 Introduction

The market demand for computer systems capable of rendering full-motion audio/video has resulted in a wide range of platform-specific audio/video file formats. To provide computer users with a common format to exchange this data, the International Standards Organization (ISO) generated ISO CD 11172, the MPEG-I specification [1]. Section II of the MPEG-I specification details a powerful video compression algorithm and its output video data stream format. Section III of the MPEG-I specification describes an equally powerful means of compressing audio data and its audio stream format. To provide mechanisms for synchronizing these streams, Section I of the MPEG-I specification details a system-layer format. The system-layer format provides a standard way of interleaving up to 16 video streams with up to 32 audio streams to produce a single interleaved stream. As more developers adopt the MPEG-I standard, the demand for a low cost means of playing out this stream is expected to grow. Currently, several hardware and software products exist to playback single audio/single video MPEG-I system-layer streams but, at the time of this submission, we are not aware of any software-only systems that have been generated to support multiple audio/video MPEG-I playback.

One of the goals of our development effort has been to explore issues and applications enabled by the multi-stream function described by the MPEG-I standard. Possibilities include any combination of audio and video streams including multiple video only, multiple audio only, multiple video with single audio, as well as multiple video streams multiplexed with multiple audio streams. Moreover, there is no requirement to start and stop streams together. This allows for the the option to staggered start audio and video clips as well as overlapping audio/video pairs of differing run-lengths. Applications include sporting events, training packages, and concert playback with video output on multiple participants with either a common audio stream or a separate audio stream for each instrument or commentator. Viewers would then have the option to either cancel, mute, pause, or enhance a specific audio or video stream based upon their level of interest. Inherent to a multiple audio stream context is the need for a high quality audio mixer to combine streams of various sampling rates and channel characteristics (i.e., stereo or mono). The MPEG-I standard also allows for the multiplexing of "private" streams into the same system layer stream as as the audio/video data. This allows for the creation of specialized applications such as embedded closed captioning, animation information, and MIDI tracks to be decoded synchronously with audio/video data.

The MPEG-I system-layer encoding encapsulates packets of data taken from the individ-

ual audio and video streams. Packets from differing input streams are interleaved together. To provide for synchronization between these streams, a system of time stamps is used. These time stamps are referred to as the System Clock Reference (SCR), the Presentation Time Stamp (PTS), and the Decoding Time Stamp (DTS). The SCR indicates the current time which should appear on the 90 KHz system clock while the stream is being played. The DTS indicates the time a given packet of data should be decoded. The PTS indicates the time at which the first frame contained within a packet should be played out. These structures and a theoretical basis for real-time inter-stream synchronization are presented by Lu et al [3]. However, the authors require knowledge of frame decode time to within microsecond accuracies. In our implementation we did not assume a real-time operating system but instead assumed the limitations of a generic software-only Unix solution.

Rowe, Smith, and Liu developed a highly functional audio/video MPEG-I application known as the CMPlayer [4]. The CMPlayer effectively provides real-time audio/video playback over a network using MPEG-I video streams. The CMPlayer does not use MPEG-I audio streams, but instead uses an 8 KHz, 8 bit stream of audio data. Without using an MPEG-I audio stream, the player cannot support the system-layer stream format. Moreover, the CMPlayer needs to enclose its MPEG video stream within a "clip-file" wrapper. This wrapper provides both frame rates and offsets to header structures within the stream. Our mechanism instead is designed in conformance to the MPEG-I System Layer encoding specification.

Szabo and Wallace [5] published an early paper which addresses the strengths and shortfalls of the MPEG-I system stream from the perspective of of audio/video synchronization. They describe interleaving as a technique to prevent disk seeks (thrashing) from interfering with real-time I/O performance, and assert that limitations of time stamping to achieve inter-stream synchronization in the MPEG-I standard. We investigate these implications in the discussion of our system implementation.

The remainder of this paper is organized as follows. Section 2 provides an overview of MPEG-I video compression and presents the design of the multistream encoder. Section 3 discusses the modifications required to convert a "best effort" video decoder into a real-time player. Section 4 presents the design of our multistream player called SLAP (System Layer Player) including the mechanism used to synchronize multi-stream audio/video playback. Player performance results are detailed in Section 5. Section 6 concludes the paper.

2 Multi-Stream Encoder Design

2.1 MPEG-I Video Compression

The stream of the MPEG-I video compression layer is characterized by techniques for both intra-frame and inter-frame compression. For intra-frame compression, the MPEG-I standard specifies the use of a hybrid of lossy Discrete Cosine Transform-based compression and lossless Huffman and run-length encoding. Inter-frame compression is achieved by finding redundant information in neighboring frames and avoiding its duplication. Each video frame can be either an Intra-frame (I), Bi-directional (B) or Predictive (P) frame. An I-frame contains all the information needed to decode itself while a P-frame can draw upon information provided within a previous I or P-frame. This inter-frame encoding process reduces redundancies between image frames providing greater compression. B-frames can draw upon the immediately following I-frame or P-frame as well as the immediately preceding I or P-frame when either encoding or decoding. As expected, B-frames provide greater compression than either I or P-frames. These structures are explained in detail by LeGall[2] and are not the focus of our paper. However, this compression approach yields high compression ratios but also results in frames which differ, not only in their size, but also the processing time needed to decode them.

2.2 Multi-Stream Encoder Design

The encoder provides the functionality to multiplex multiple audio and video streams into a single stream in accordance with ISO CD 11172. An MPEG-I system-layer encoder can be implemented as a 1 or 2-pass encoder. A 1-pass encoder is used for real-time compression because it must estimate some stream parameters before reading and encoding the input stream [3]. A 2-pass encoder can first scan each stream and calculate suitable stream parameters such as buffer size. Due to the non-real-time nature of this part of the application, a 2-pass design was chosen because of its added flexibility.

2.2.1 First Pass

In the first pass, the encoder creates a list of the different audio and video streams that will be included and finds the necessary header information. In addition, the type and location of each frame in the individual streams are found and initial presentation times for the

first frame of each stream are set. In the second pass, the encoder reads data from each of the various streams and writes them into a single system stream along with the various time-stamp and header information. The primary structure used for control of the encoder's second pass is the Multiplex Timing Array (MTA). The MTA maintains the state of all streams so the encoder process can perform an optimal sequencing algorithm whereby the packet with the oldest Decoding Time Stamp (DTS) time code gets encoded next. This allows for the best possible sequencing in the output stream.

A preprocessor comprises the first pass of the encoder and is primarily concerned with correctly identifying the streams (audio or video) and finding the location, size, and type of the frames in the stream. The module follows a process of going from stream to stream and creating a list of all the frames in the particular stream. This list is created by going through the entire stream looking for the frame start codes. A frame starts at the first start code and ends at the byte just before the next start code. The encoder also determines the frame type and frame size. Finally, if the frame type is determined to be an I or P frame, the encoder will keep track of how many B frames are passed until the next I or P frame and stores this number in a separate list. This is important because some I or P frames must be decoded before B frames, but presented afterwords. In addition to the frame data, the preprocessor also generates buffering and timing information based on header information such as frame rate, as well as processed information such as largest frame size.

2.2.2 Second Pass

An encoding mode comprises the second pass of the encoder during which the various audio and video streams along with an optional padding stream are multiplexed in a single system level stream. The controlling mechanism of the encoding module is the Multiplex Timing Array (MTA), a stack-like data structure with pointers to the highest priority streams. The MTA starts by searching the streams list for the highest priority active stream and all other active streams with equal priority. At any one time, all the streams in the MTA have equal priority, but the closer the stream is to the top, the sooner it gets encoded. The encoding is done on a packet by packet basis, meaning that a packet of the highest priority stream is encoded and that stream is then removed from the MTA. Once a packet is written, an updating routine calculates how many frames of that stream have been written in one packet. For every frame written, the priority of that stream is decreased. If the last frame has been written, the stream is made inactive. The cycle is then repeated. This continues until the MTA cannot find any active streams in the stream list.

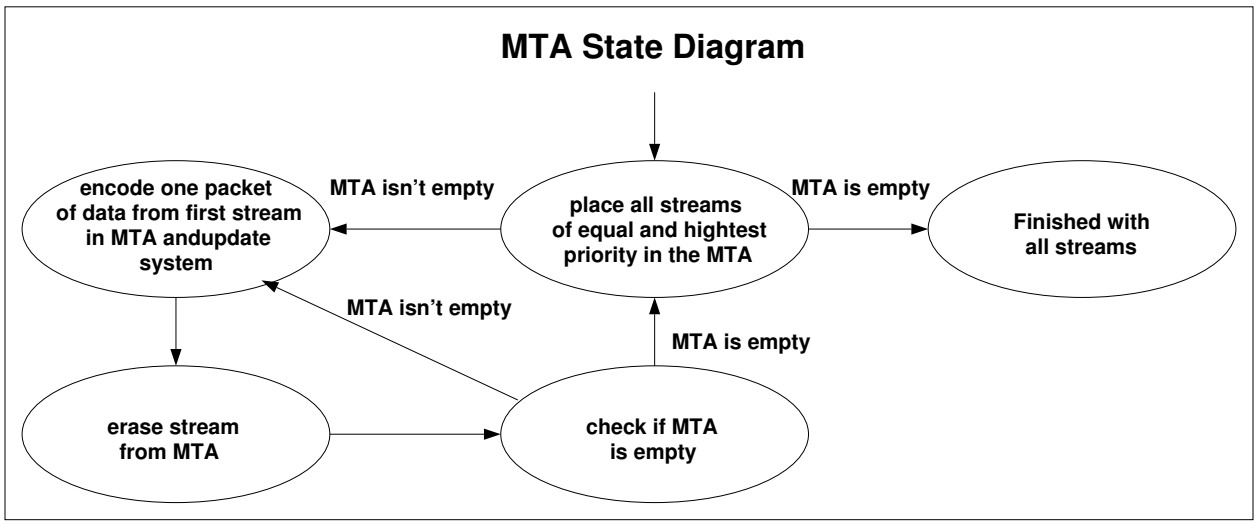

Figure 1: MTA State Diagram

3 Real-Time Video Player

Although the Berkeley `mpeg_play` program provides all functionality needed to decode and display an MPEG-I video stream, it is not capable of real-time playback. The player is only capable of playing-out data frames within the best effort of the system. To achieve real-time playout, it was necessary to modify the existing player to adaptively drop frames to match the required real-time frame rate of the data to the platform's processing capability. In the context of our system layer player (called SLAP), an adaptive video player process attempts to act as a data sink with an input data rate that exactly matches the requirements established by the stream headers. To ensure the smooth playback of any given MPEG-I video stream, the player requires the ability to alter its internal configuration to adapt to specific video stream characteristics such as frame rate and the mix of frame types.

3.1 A Real-Time Clock

The first step in establishing any real-time system is to generate a stable real-time clock mechanism. While hardware systems can rely on microsecond accuracy timers, any real-time clock generated in software under Unix has an accuracy based on the reliability of interval timer signals and the process scheduling mechanism. Through experimentation it was found that a 0.5 to 1.0 second accuracy was the best that could be expected from a software interrupt-driven real-time clock. Since this is not an acceptable synchronization

mechanism for dealing with frame rates of at least 24 frames per second, a higher resolution time-source was needed. For this purpose, a frame clock is maintained. For example, given the MPEG-I 90 KHz system clock, a video frame rate of 30 frames per second increments the frame clock by 3000 ticks (90000/30) each time a frame is seen. By tracking the difference between the frame clock and the software real-time clock, one can determine whether the player is going too fast or too slow. Fig. 2 shows the overall video decoder architecture.

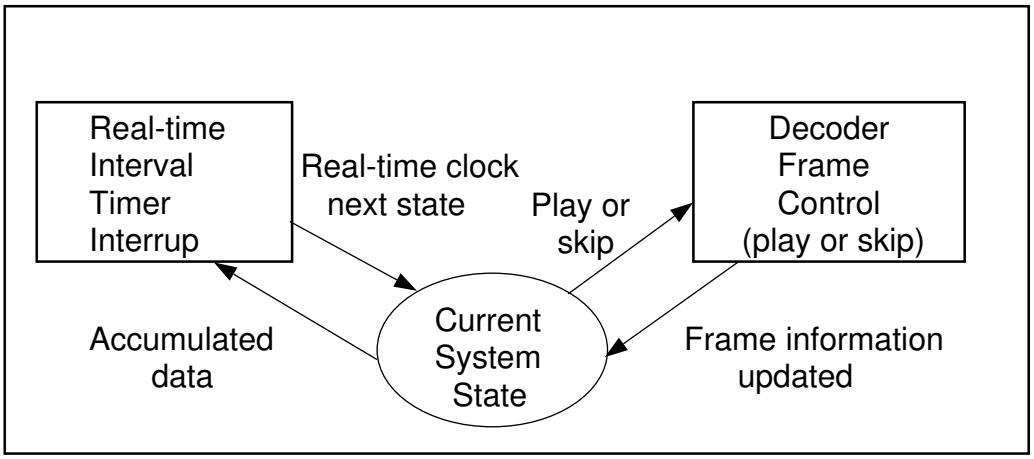

Figure 2: Video Player Architecture

3.2 Frame Dropping

The simplest mechanism available to adjust frame times to support real-time playout is to drop groups of frames. If the system is unable to play all the frames then all of the B-frames can be discarded. If the system is still too slow then all of the P-frames can be dropped. Finally, I-frames can be dropped in any quantity. For example, if it is expected that 6 I-frames, 6 P-frames, and 18 B-frames will be seen over a given one second interval, playing all the frames would provide 30 frames per second. Dropping all the B-frames means we will be playing only 12 frames per second. If "step-resolution" is defined as the smallest increment of control between two playback states in frames per second (i.e., the difference between playing 30 frames per second or 12 as in the above example), the example equates to a step-resolution of 18 frames per second. This approach results in high jitter when trying to play out MPEG-I video on platforms capable of playing, for example, 20 frames per second. This is because the system state will oscillate between trying to play 30 frames per second to trying to play 12 frames per second.

3.2.1 Inter-Frame Dropping

Skipping individual B- and P-frames is not a simple matter due to inter-frame dependencies. If the B-frames depend on the P-frames, then one cannot drop one of the P-frames and still play back all of the B-frames. This matter is further complicated when two P-frames depend on one another. For example, in the pattern IBBPBBPBBI, if the system dropped all the B-frames and the first P-frame, there would be insufficient information available to decode the second P-frame, so the system would be forced to drop it. These inter-frame complexities force the dropping of frames to be a dynamic decision based on the current real-time state as well as past state decisions.

3.2.2 Adaptive Performance Requirements

To ensure that a given hardware platform provides smooth video playback, the player must ensure the following:

- The frame dropping algorithm must provide a step-resolution of 1 frame per second for all phases of operation (explained below);
- The mechanism must be able to adapt to sporadic processor scheduling and to I/O contention without slipping irreversibly from real-time;
- The player's internal configuration must be calibrated to meet the needs of the specific video sequence being played.

To cover the full range of possible hardware platforms, it is necessary to consider not only the case where the hardware can decode and display the video stream faster than desired, but also the case where only a fraction of the desired frames per second are being displayed. Four phases of operation can be identified:

1. **Delay Phase:** In this phase, the system can playback frames at speeds greater than the required rate. Delay loops provide for suitable delays between frames.
2. **B-Phase:** The B-phase covers a range of playout schedules, from one in which all the B-frames are being played, to one of playing a single B-frame and skipping all the others within a given second of time. By their nature, if the B-frames are being played, then all P- and I-frames are being played as well. By incrementing the number

of frames skipped by one, and decrementing the number of frames played by one, the logical resolution provided has an accuracy of one frame per second.

3. **P-Phase:** In the P-phase, the same "range of values" approach is used as was in the B-phase. The decision to either play or skip a P-frame is dependent on the total number of frames played and skipped during the past time interval. Even frames that had to be skipped due to inter-frame dependencies are included in the decision.

4. **I-Phase and Degraded Phase:** I-phase functions the same as both B- and P-phase, but flexibility is lost when playing just one I-frame of all the frames seen during a given second. To address this shortfall, a Degraded Phase was added. In the Degraded phase, only one I-frame per interval is played and all others are skipped. As the Degraded Phase progresses, the timer interval is stretched to accommodate the fact that the number of frames to be seen will exceed the number of frames expected per second.

The MPEG-I stream can legitimately contain any pattern of I, B, or P-frames. To ensure a smooth transition between phases, one needs to calibrate the real-time state parameters for the specific stream being read. By counting the number of frames of each frame type seen from the beginning of the stream and comparing it to the frame rate, it is possible to tell when one second of frames has been received. This information can be used to identify the frame type mix and set the start and end limits of each phase.

3.2.3 Real-Time Video Player Performance

An example of the results achieved is shown in Fig. 3. This figure shows the number of frames seen (i.e., frames played + frames skipped) as well as the actual number of frames played out in relation to the system real-time clock. Figure 3 shows the I-phase operation with a 24 frames per second sequence of frames with dimensions of 320x240 pixels in an SGI Indigo.

4 System Layer Decoder/Player

The decoder/player has three major functions: demultiplexing and decompressing the system-layer stream, providing for both intra and inter-stream synchronization, and controlling user interaction. After demultiplexing the system-layer input stream, each compression-layer

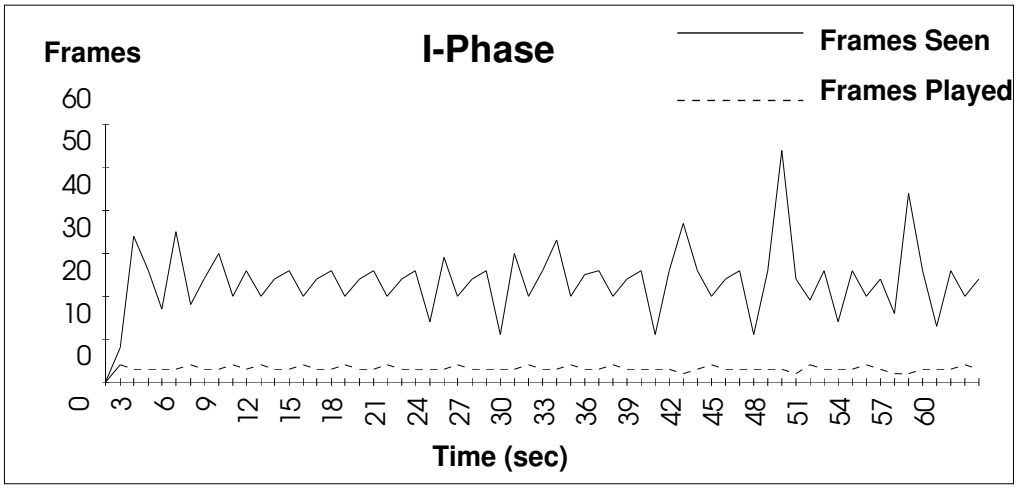

Figure 3: I-Phase

stream packet of data is forwarded to the appropriate audio or video decoding process. Audio decoding processes output streams of 16-bit Pulse Code Modulation (PCM) samples. These samples are read by the audio mixer/interface process, where multiple audio streams are combined. The task structure for the decoder is depicted in Fig. 4. Processes are connected via Unix pipes for simplicity. Later versions could use shared memory FIFOs to avoid unnecessary copying of data. Shared memory is used for inter-process communication of timing data. If audio streams are present, all video processes will track an audio based time mechanism. If no audio streams are currently playing, a real-time interrupt will be used to maintain a software real-time clock for use by the video processes. Due to the multi-stream nature of this project, the decoder must be capable of dynamically changing from audio based synchronization to the real-time interrupt based synchronization mechanism and back.

4.1 Synchronization

Traditional tight coupling synchronization mechanisms are difficult to adapt to the context of multiple video and audio streams in a software-only system. This is in part due to the fact that MPEG-I does not define relationships between individual streams. This means one cannot tell which video stream should match which audio stream when multiple streams are presented. We chose to implement relative synchronization [1] through a decentralized control mechanism. Each video process will be responsible for adapting to an acceptable

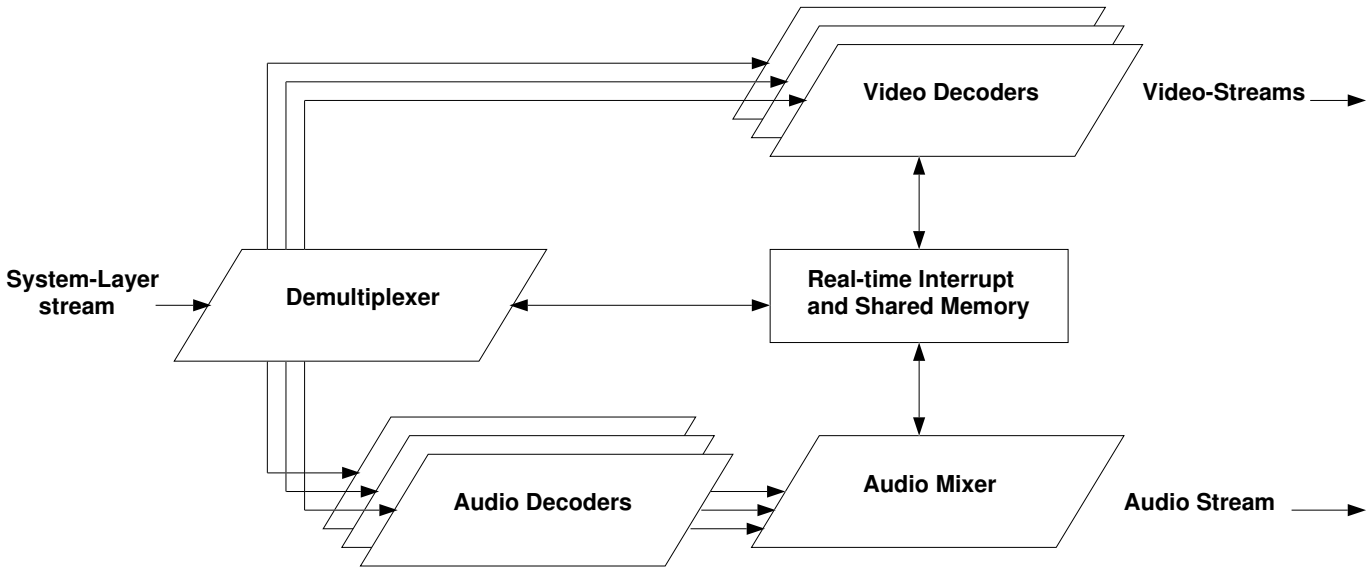

Figure 4: Decoder Task Structure

playback state to achieve real-time playback in the intra-stream context.

While investigating multi-stream applications, we identified two different contexts where different synchronization paradigms should be used. The synchronization of audio inclusive streams can be effected at the audio mixer by forcing the process to read from all active audio streams before mixing. This approach allows one to utilize the number of audio frames written to the hardware device to generate an audio frame clock. Using shared memory, the single audio frame clock can be passed to all the video processes. The video processes will attempt to follow and maintain synchronization between their internal video frame clocks and the external audio frame clock. Since there is only one audio frame clock, and that clock is shared by all video processes, the quality of audio/video synchronization will be dependent on how well the video players perform their adaptive real-time playback function. When only video streams are being played, a real-time interrupt handler can be used to maintain the real-time clock, normally updated by the audio mixer.

4.2 Audio Decoding/Mixing

The `maplay` MPEG-I audio decoder program provided by Tobias Bading of the Berlin University of Technology required only minor changes to be suitable for this application. One change was to have the player write a 32-bit value through `stdout` as soon as the input

stream's sample rate has been calculated. This 32-bit value arrives ahead of any of the PCM samples and is read by the audio mixer process so as to inform it what sampling rate of data will be arriving from the pipe. Since MPEG-I supports three valid sampling rates (32, 44.1, and 48 KHz), actual rates must be converted to a common frequency before mixing.

To ensure that audio data remains synchronized, a blocking read from each of the PCM streams is done when only one audio stream is present. When more than one audio stream is present, polling must be done to ensure the system does not deadlock. Since the demultiplexer must perform a blocking write to avoid the loss or corruption of data, the possibility existed for the demultiplexer to block writing data to audio stream 1's decoding process at the same time the audio mixer blocks reading from audio stream 2's decoding process. This would cause a system-wide deadlock.

The mixing of MPEG-I audio streams in a multi-stream context is complicated when dealing with the cases where two streams either start or stop at differing times. A design decision was made to force the video playback rates to keep up with output rate of the audio mixer and device. This allows a single audio frame clock to be maintained which is valid for all audio streams. The problem, which occurs when starting at differing times, is that the mixer knows that there will be N audio streams in the system-layer stream, but it does not know which ones will start at time 0.0 seconds and which will start later in the sequence. This problem was handled by preventing the reading of the audio mixer input pipes until the audio frame clock is either equal to or exceeds the first recorded PTS value. The same approach was used to ensure offset video data began its displaying on schedule. Note this was one of only two useful purposes assigned to the system-layer time codes in this application.

4.3 Audio Layer Limitations

Since the MPEG-I audio format does not include an end-code to the sequence of audio frames, there is no efficient method in software to determine when to cease blocking for a mixer-read of a given stream when the audio stream has terminated. Using empty pipes or blocked decoding processes does not work, since processes are continuously blocking and pausing in a multi-stream context. The demultiplexer has no immediate way of detecting when an audio packet contains the last frame of audio data. In short, the lack of audio sequence end codes makes it impossible to cleanly terminate an audio decoding process once all data has been read. This is a serious shortfall in the MPEG-I specification if multi-stream applications are to be used in flexible applications. This issue is addressed by maintaining

an audio frame count that is reset between timer intervals. If no frames have been written to the audio device since the last interrupt, the currently blocked pipe is considered "frozen" and the stream is canceled. A problem occurs when the playback system is heavily loaded. The possibility arises for a zero audio frame count when we do not wish to terminate. It can be reasoned that if the audio stream has truly ended, the current SCR value will be greater than the current audio frame clock value. This relationship is tested each time a stream is considered frozen. If true, the stream is canceled. If false, the stream is kept active. One shortcut that helped resolve this problem was the fact that in the system stream, any audio stream can be padded with a series of zero bytes. This creates a pseudo-end-code and causes the decoder process to terminate. This results in a closed pipe to the audio mixer and an end-of-file message. Although this approach was useful, MPEG-I in no way forces one to pad the end of audio streams with zeros.

5 Results

Synchronization quality was tested using system layer streams of differing video frame sizes and audio sampling rates. Quality was evaluated based upon the drift measured between the audio frame clock (i.e., real-time) and the video frame time measured at the video decoding process (i.e., video stream time) as follows:

Excellent	< 0.3 s
Good	0.3 – 0.6 s
Acceptable	0.6 – 1.5 s
Poor	> 1.5 s

Each test stream run produces two graphs. The first is a graph showing the number of frames played vs. the total number of frames whether played or skipped over a particular interval. The gap between these two lines can be used to judge the relative difference between the platform's capabilities and those required to playback the full MPEG-I stream without dropping video frames. The second graph shows the time drift between the audio frame clock and the video frame clock. If the system was perfectly synchronized, this line would be horizontal and always equal to zero. Any jumping at the very end of a test run is indicative of the ending of the video sequence and can be ignored.

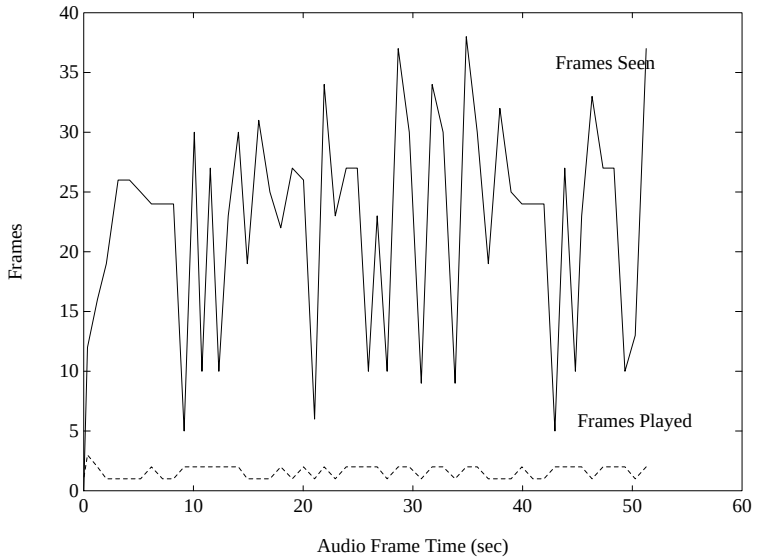

Figure 5: Full Sized Video Stream on Indigo-2

Fig. 5 can be used to show how the player functions in a degraded mode where the system is being asked to playback a stream far beyond its capabilities. A "full" frame is defined as 640x480 in our tests. The curves show erratic jumps in the total number of frames while the actual number of frames played per second is oscillating relatively smoothly between the range of 2 to 4 frames per second. The maximum drift on the Indigo-2 platform was approximately 1.2 seconds (Fig. 6). Based upon the group's subjective quality standards, the system's performance for this stream on the Indigo-2 was acceptable.

Figs. 7 and 8 show the playback characteristics for an Indigo playing back at "normal" frame size (320x240) sequence. An average of five frames per second was played out with a maximum drift of 0.7 seconds This playback was not adequate to read lips, but on the Indigo-2, an average of 11 frames per second was played out for the same stream with a maximum drift of 0.4 seconds (good by our standards). This run did produce playback with which one could visually match the lip movement with the words spoken.

System playback quality for "small" and "mini" sized frames where the platform performance capabilities were nearer to the actual requirements needed to playback the complete stream demonstrated both good and excellent synchronization results.

This model did not include a sophisticated buffer management mechanism. It is believed that many of the peaks of the playback curves are due to buffer overflows and underflows.

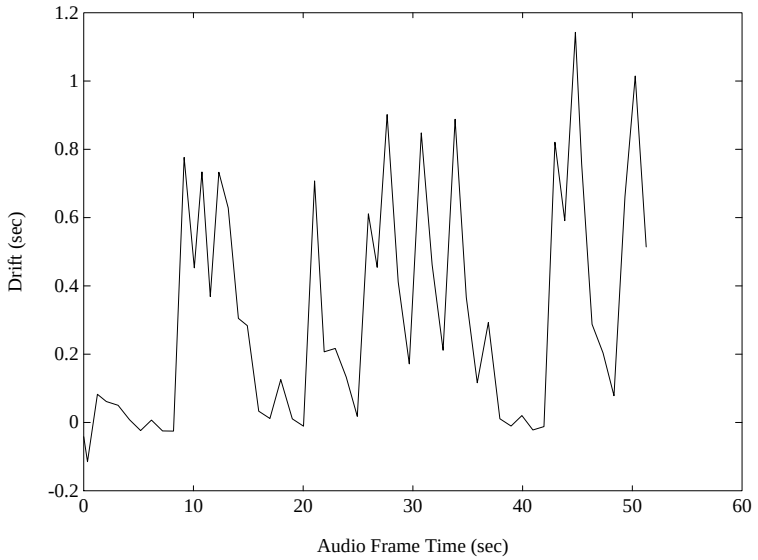

Figure 6: Full Sized Video Stream Drift on Indigo-2

Unfortunately, adding extra buffer management comes at the cost of an added processor load. In the case of our model, this added processor load was deemed prohibitive. This would not be the case on higher performance systems.

6 Conclusion

The quality of single stream audio/single stream video synchronization was deemed very good for small and mini sized video frame sequences, while normal sized sequences played at good quality. Synchronization quality was less acceptable for the playback of either full sized image sequences or multiple audio/ video stream files. It is expected that as platform performance increases and more elaborate buffer management and inter-process communications are implemented, this limitation will be less significant.

The time stamp mechanism contained within the MPEG-I system layer streams was found to be useful for two purposes. It allowed for streams encoded with an offset to avoid being played back too soon as a result of rapid data movement through input buffers with respect to that of the steady state data rate of pipes which have been already filled. As well, the current system SCR was useful in establishing whether or not an audio stream has ended or is just played-out slower than real-time. It was agreed that, for software-only

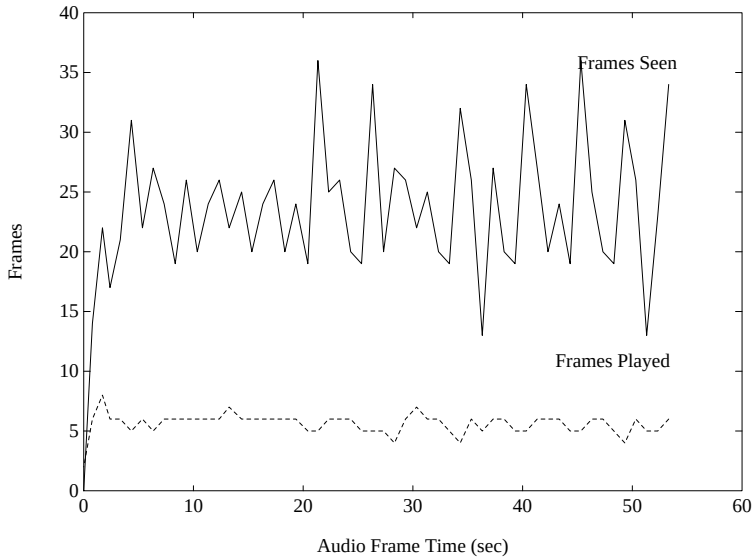

Figure 7: Normal Sized Video Stream on Indigo

systems, audio frame time and video frame time were more useful than the time stamps for maintaining audio/video synchronization.

The need for an audio end code for use in multi-stream operations was explained. Without an end code, there is no sure way of immediate identification of the termination of an audio stream. Future specifications that wish to include multi-stream functionality should address this current shortfall.

7 Future Work

The software package generated as a result of this project is suitable as a basis for an MPEG-I engine for audio/video applications. Future versions of this player could include the use of shared memory FIFO's instead of pipes for inter-process communications. The implementation of more sophisticated buffer management strategies should also improve playback quality. The current encoder and decoder fail to address the Constrained Parameter Mode settings and operations of the MPEG-I stream. This should also be addressed in future packages. The following areas are all interesting extensions to this work:

1. Porting the encoder/decoder set to other operating systems and/or platforms;

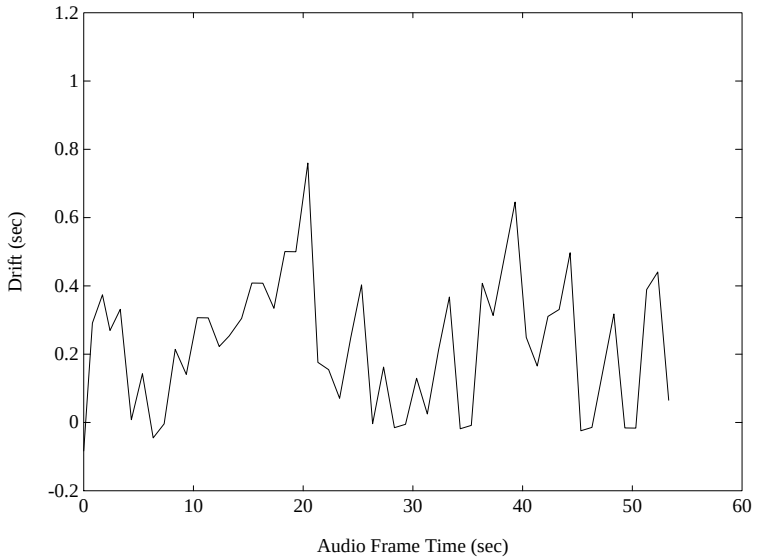

Figure 8: Normal Sized Video Stream Drift on Indigo

2. Developing multi-stream MPEG-I authoring tools;

3. The use and support of private stream applications such as closed captioning for the hearing-impaired, animation script streams, and MIDI stream support.

The introduction of this package opens the door for many new and exciting multi-stream MPEG-I applications. It is expected that further work in this area will continue as long as platform performance can grow to meet its highly demanding processing requirements.

8 Acknowledgements

Two existing public domain software packages are used as a basis for our implementation. These are the Berkeley `mpeg_play` version 2.0 which is used to decode and display each video stream, and the `maplay` version 1.2 program written by Tobias Bading of the Berlin University of Technology, which is used to decode the audio streams.

References

[1] ISO CD 11172 – Coding of Moving Pictures and Associated Audio for Digital Storage Media Up to 1.5 Mbit/s.

[2] D. LeGall, "MPEG, A Video Compression Standard for Multimedia Applications," *Communications of the ACM*, April 1991, Vol. 34, No. 4, pp. 46-58.

[3] G. Lu, H. Pung, and T. Chua, "Mechanisms of MPEG-I Stream Synchronization," *Computer Communications Review*, Vol. 24, No. 1, January 1994.

[4] L.A. Rowe, K.D. Patel, B.C. Smith, and K. Liu, "MPEG Video in Software: Representation, Transmission, and Playback," *Proc. IS&T/SPIE Symposium on Electronic Imaging Science & Technology*, San Jose, CA, February 1994.

[5] B.I. Szabo, G.K. Wallace, "Design Considerations for JPEG Video and Synchronized Audio in a Unix Workstation," *Proc. Summer 1991 Usenix Conf.*, Nashville, TN, June 1991, pp. 353-368.