A Software MPEG-1 Decoder for Internet TV 
Set-top Boxes

Yoonsik Cheon, Heung-Nam Kim and Kyusang Shin
Real-Time Computing Department
ETRI Computer & Software Technology Laboratories
161 Kajung-dong, Yusong-ku, Taejon 305-350 Korea
cheon@etri.re.kr.

Abstract—A recent advance on the Internet access technologies like CATV and ADSL has opened a new possibility in delivering multimedia services to homes. It becomes possible to watch movies or listen to music at home through the Internet, which demands a higher network bandwidth. In this paper, we report on our experience in developing such a system, focusing on software MPEG-1 decoders for Internet TV set-top boxes. A set-top box is a device for accessing the Internet using a television set and a regular telephone line, a CATV line, or a satellite dish. A prototype system has been developed on a commercial real-time operating system (RTOS), and will be ported to an in-house RTOS and set-top boxes. However, the performance of the prototype system turned out to be quite unsatisfactory; an MP3 audio decoder worked very well, but an MPEG-1 video decoder performed poorly, way below our expectation. Therefore, we are a bit pessimistic on realizing, in software, an MPEG-1 system for set-top boxes that are typically built with cheap and low-power processors. We are currently in the process of improving its performance, but at the same time we are also considering to take hardware-based solutions such as decoding chips or boards.

Keywords—Internet TV, MP3, MPEG-1, set-top box, software decoder

I. BACKGROUND

It becomes possible to watch movies or listen to music at home through the Internet due to a recent advance on the Internet access technologies such as ADSL, CATV and ISDN, supporting a higher network bandwidth. There are two noticeable trends on this. The first is a non-PC based Internet access like set-top boxes, and the second is a realization of multimedia services through software solutions. For example, as the speed of processors increases, real-time software decoding of compressed video becomes possible, and software MPEG decoders are replacing hardware decoders [1] [2]. In this paper, we explain our experience in developing software MPEG-1 decoders for Internet TV set-top boxes. A set-top box is a device that allows one to access the Internet by connecting it to an analog television set. There are three different types of set-top boxes: one is for the telephone-based set-top boxes; another is for cable-TV set-top boxes; and yet another is for satellite set-top boxes. The software decoders have been developed as a part of a project aimed to provide, through set-top boxes, Internet-based multimedia services to homes, e.g., video-on-demand and audio-on-demand.

In ETRI Computer & Software Technology Laboratories, we have been launching a project called I-TV, funded by the Korean Ministry of Information and Communications. The aim of the project is to develop an economical Internet access hardware and software system for home users. By the I-TV hardware, we mean set-top boxes that can be plugged into analog television sets. The software system includes a real-time operating system, a set of libraries, applications, and a support tool-set for the newly developed operation system. The I-TV applications include such programs as a web browser, an email client, and media players such as an MP3 player, an MPEG-1 player and an MPEG-4 player. This paper is concerned with both the MP3 player and the MPEG-1 player, collectively called an MPEG-1 system for set-top boxes. In particular, we are interested in realizing the MPEG-1 decoder in software.

The MPEG-1 is an international standard for coding audio-visual information in a digital compressed format [3] [4] [5]. It addresses the compression of synchronized video and audio at a total bit rate of about 1.5 megabits per second. The standard consists of three parts: system, audio, and video. The audio offers a choice of three independent layers of compression depending on codec complexity and compressed audio quality: layer I, II, and III [5] [6]. The video represents a pure video stream without accompanying audio [4] [7], and the system part is a standard for synchronizing and multiplexing elementary audio and video streams [3]. When we say an MPEG-1 video, we usually mean an MPEG-1 system stream in the MPEG terminology.

II. AN MPEG-1 SYSTEM FOR I-TV

An MPEG-1 system for I-TV enables one to play, at home and in real-time, MPEG-1 video and audio data stored on the server using I-TV set-top boxes. In addition, the system can synchronize textual information to MPEG-1 streams; teaching materials like slides can be synchronized to educational videos, or lyrics can be synchronized to music videos. The MPEG-1 system consists of three components: a media server, a transmission system, and a media player. The media server is responsible for storing and managing multimedia data such as MP3 audio files and MPEG-1 video files. The transmission system is in charge of streaming multimedia data from the server to the client set-top boxes [8]. The media player, residing on the client set-top boxes, decodes and plays the transmitted multimedia data. The media player supports both MP3 audio and MPEG-1 videos. The media server runs on LINUX machines and the transmission system conforms to the DMIF standard [9].

The media player, containing MPEG decoders to be discussed in this paper, has a layered architecture and consists of three layers: a presenter, decoders for MP3 and MPEG-1, and a DMIF client. The DMIF client at the bottom layer is the client side...
of the transmission system; it receives media streams from the server and passes them to the decoders. The decoders in the middle layer decode the received media streams so that the upper layer presenter can be able to display them. The middle layer in fact consists of two independent decoders: an MP3 decoder to decode MPEG-1 layer III audio streams, and an MPEG-1 decoder to decode MPEG-1 system streams. The presenter, as the name implies, is responsible for displaying the decoded audio and video samples to the output devices such as TV screens and speakers. It also manages and coordinates the whole player system in that it interacts with the user and handles all sorts of requests by dispatching proper operations. The media player has been integrated with an I-TV web browser [10].

From the beginning of the I-TV project, five major tasks — building set-top box hardware, developing a real-time operating system (RTOS), a set of libraries, application programs, and a support tool-set for the new RTOS — have been being performed concurrently by four different teams. As thus, the first prototype MPEG decoders have been implemented using VxWorksTM [11], a commercial RTOS, on Intel 80x86 boards. Later, they will be ported to VxWorks-based set-top boxes when the hardware becomes available. Eventually, they will be ported the ETRI-developed RTOS when it is released.

III. A SOFTWARE MPEG-1 DECODER FOR SET-TOP BOXES

To operate in a special environment like set-top boxes, the decoder has to satisfy several requirements. The following is some of the important requirements.

- **Light-weighted.** Since the target systems are set-top boxes with limited resources, the decoder must be small and light-weighted. For example, the size of execution file should be small and resource requirements (e.g., for dynamic memories) should be minimal.

- **Support for streaming and dynamic adjustment.** The decoder should be able to dynamically reconfigure or adjust to the changing environment (e.g., unstable network bandwidth), and should also support streaming in cooperation with the underlying media transmission system.

- **Synchronization of textual information.** In addition to audio and video synchronization, the decoder should be able to synchronize and decode auxiliary textual information — e.g., slides for lectures, captions for movies, lyrics for music videos, etc.

- **Portability.** The decoder should provide a flexible and well-defined interface, and should be portable to different hardware and software systems.

- **Support for control.** In addition to the basic decoding capability, the decoder should provide a set of control functions to manage the decoding process. It should support at least such control functions as pause, resume, jump, and stop.

A. Architecture of MPEG-1 Decoder

Fig. 1 shows the architecture of MPEG-1 decoder [3]. Conceptually, the decoder consists of three components: system decoder, clock control, and elementary decoders. The system decoder de-multiplexes MPEG-1 system streams into elementary streams such as MPEG-1 video streams and MPEG-1 audio streams, which are decoded by elementary decoders such as a video decoder, an audio decoder, and a text decoder. The clock control module implements a system clock to synchronize the decoding of elementary decoders and the display of decoded samples. An MP3 audio decoder is not separately developed but as an extension of the elementary audio decoder by adding decoding capability for MPEG-1 layer III audio streams.

B. Application Programming Interfaces (APIs)

The MPEG-1 decoder interacts directly with the presenter and indirectly with the media transmission system through the presenter. In the view point of data, the raw stream flows into the decoder through the input buffer and the decoded samples flow out from the decoder to the presenter through a set of output buffers (see Fig. 1). In the control aspect, the decoder interacts with the presenter through a set of well-defined interface functions. The major interface functions supplied by the decoder can be classified as follows.2

- **Registration functions:** registerInput(), registerAudioOutput(), registerVideoOutput(), registerTextOutput()
- **Decoding function:** play()
- **Control functions:** pause(), resume(), stop(), preJump(), jump(), init(), trash(),
- **Time functions:** setSTC(), getSTC()

The registration functions register input and output callback functions, thereby allowing the decoder to access the input buffer and the output buffers. The decoder invokes a registered input callback function when it needs samples to decode, thereby, reading the input buffer. It calls a registered output callback function when a unit of samples has been decoded (e.g., an audio frame and a video picture), thereby, storing the decoded frame into the corresponding output buffer. By using callback functions, we can encapsulate and hide the implementation details of I/O buffers from the decoder.

It is necessary to synchronize the decoding and display of audio, video, and text samples. For this, the decoder uses a logical clock, called a system time clock (STC), and it stores to the output buffers not only the decoded samples but also their presentation time stamps (PTS). The PTS’s represents the time on which the corresponding samples must be presented to the output devices such as TV screens and speakers. The presenter outputs the samples stored in the output buffers on the output devices according to their PTS’s. For this, the presenter refers to the STC

---

1The first release of set-top box hardware will host only Intel Strong ArmTM SA110 processors.

2In the actual implementation, the word “mp3” or “mpeg1” is prefixed to the names of interface functions. For example, two play functions are defined for the decoding function: mp3Play() for MP3 and mpeg1Play() for MPEG-1.
through time functions like getSTC. In short, the time functions allow the presenter to play the audio, video, and text streams at the time intended by the encoder.

The control functions allow the presenter to control — e.g., pause, resume, jump, stop, etc. — the decoders. A state transition diagram as shown in Fig. 2 represents the meanings of control functions and their inter-relationships.

It is worthwhile to explain the design of jump operation that is performed in two steps (p_jump and jump). This is a side effect of supporting streaming. It is necessary for the decoder to interact with the presenter, the coordinator of MPEG-1 media player — for example, to clear out internal buffers and restart decoding at a new position. We can describe the protocol between the decoder and the presenter for the jump operation by a message sequence diagram.

First, the presenter informs a jump intention to the decoder by sending a prepare_jump signal (step 1). During this time, the stream is still being transmitted to the decoder by the DMIF-based media transmission system so that the decoding of current frame can be completed. As soon as the decoder completes decoding of the current frame, it replies an acknowledgement to the presenter and waits for a further instruction (step 2 and 3). On receiving an acknowledgement from the decoder, the presenter, in cooperation with the DMIF, clears the internal buffer and starts to transmit the stream at a new position (step 4, 5, and 6). And then it sends a jump signal to the decoder so that the decoder can start decoding with the newly transmitted data (step 7). The decoder resumes the decoding process (at the new position) as soon as it receives a jump signal (step 8 and 9). In the message sequence diagram, the steps 1-3 are implemented by the interface function preJump(), and the steps 7-8 are implemented by the function jump().

C. Synchronizing Textual Information

The MPEG-1 decoder should support for displaying synchronized auxiliary textual information such as slides, lyrics, and captions. For the synchronization of textual information, we develop a synchronization editor and add a text decoder to the MPEG-1 decoder. The synchronization editor inserts textual data into the existing MPEG-1 system streams. The textual data constitutes a separate elementary stream, called a text stream, that is multiplexed to system streams. A text decoder that is an elementary decoder included in the MPEG-1 decoder, decodes the text stream (see Fig. 1). Conceptually, a text stream is a sequence of pairs of a slide and a PTS. Slide is a unit of presentation for text streams — e.g., one page of presentation slides, a line of captions, etc. The PTS indicates when the corresponding slide should be presented to the output device. The synchronization editor first divides text streams into packets and then multiplexes them to system streams nondestructively. As a result, any existing MPEG-1 viewer can play the new MPEG-1 streams, but only with ignoring the synchronized data.

IV. Implementation

An early prototype has been implemented upon Intel 80X86 boards running VxWorks, a commercial RTOS from Wind River Systems, Inc [11]. The prototype has been integrated into the presenter and the media transmission subsystem, and can play MP3 audio data and MPEG-1 video data stored on a remote media server (see Fig. 3). The whole system was written in C, and developed in the VxWorks development environment, called Tornado. We chose the programming language C as our implementation language because our ultimate target system — the ETRI RTOS and its support tool-set — will initially support only the C language. We used a low-level graphics library called a Universal Graphics Library (UGL), and a commercial

3Technically, text streams are multiplexed to system streams as MPEG-1 private streams.
audio driver called Open Sound SystemTM by 4Front Technologies Inc.

For the elementary video decoder we ported an open source MPEG decoder by the MPEG Software Simulation Group (MSSG). Later, we found out it was a major mistake to take the MSSG video decoder (refer to Section V for a detailed discussion). For the MP3 audio decoder, we referenced an existing MP3 decoder written in C++, and we improved and translated it into C on VxWorks. The MPEG-1 system decoder and the audio/video/text synchronization mechanism have been implemented from the scratch. For the synchronization editor, only the engine part has been completed by now, and a GUI version will be implemented in the future.

The implementation of MPEG-1 decoder reflects its conceptual structure (see Fig. 1) and consists of five modules: a system decoder, a clock control, a video decoder, an audio decoder and a control module. The control module implements the public interface of decoder, that is, control functions such as pause, resume, jump, stop, etc. (refer to Section III-B). The text decoder is implemented as a functionality of the system decoder. The system decoder, the audio decoder and the video decoder are all implemented as VxWorks tasks, i.e., independent processes. The internal buffer for transmitting de-multiplexed streams from the system decoder to the elementary decoders such as the video decoder and the audio decoder is implemented as a ring buffer. The ring buffer supports both synchronous and asynchronous read and write.

We used VxWorks semaphores to synchronize among control functions and the system and elementary decoders. For example, a separate decoding task is spawned when the presenter invokes the mp3Play() control function. The control functions to be called by the presenter, communicate with the decoder through semaphores. It is instructive to look into the implementation of the control function mp3Stop() as a simple example. We use two shared variables.\footnote{The SEM_ID is a VxWorks built-in type for semaphores.}

```c
int stopFlag; /* 1 if stop requested; 0 otherwise */
SEM_ID semStop; /* semaphore for sync */
```

The control function mp3Stop() is implemented as follows.

```c
void mp3Stop()
{
  stopFlag = 1;
  semTake(semStop,FOREVER);
}
```

It first sets the variable stopFlag to inform the presenter the intention of stop, and waits for a response, i.e., tries to take the semaphore semStop. The presenter is expected to release the semaphore as soon as it completes its part of stop process.

The decoder process that is spawned by the control function mp3Play() has the following structure.

```c
void mp3Decode(){
  initialize data structures;
  while (getHeader()){
    decode current frame;
    if (stopFlag)
      break;
    process other control flags;
}
  free data structures;
  if (stopFlag)
    semGive(semStop);
}
```

On the completion of decoding each frame, the function checks whether any control flag such as stopFlag are set. If any flags are set, it acts accordingly to process the actions requested by the presenter. For example, if the variable stopFlag is set, the function breaks out of the decoding loop, frees internal data structures if any, and finally informs the presenter completion of its part of stop process by releasing the semaphore semStop. The presenter is waiting for the semaphore semStop after setting the stopFlag (refer to the definition of function mp3Stop). In summary, the presenter’s control requests are propagated to the decoder through binary flags, and the presenter task and the decoder task synchronize with each other using binary semaphores to process the control requests.

In MPEG-1, the control function mpeg1Play() spawns a system decoder as a separate task. The spawned system decoder process first creates an internal ring buffer, and then spawns the elementary decoders such as an audio decoder and a video decoder as independent tasks and coordinates them. For example, the control information such as control requests from the presenter is passed to the audio decoder and the video decoder by the system decoder. Thus, the system decoder is responsible not only for de-multiplexing system streams and synchronizing elementary decoders, but also for brokering control information between the presenter and the elementary decoders.

### A. Implementation Experience

In this section, we shortly report on our experience in implementing an early prototype, and explain the lessons we learned. We were not familiar with real-time systems and their cross development environments, in particular, VxWorks and Tornado by Wind River Systems, Inc [11] [12]. The difficulty was resolved as the development progresses and we get acquainted with the systems.

The most distinguished feature of our development is the pursuit of rigor and exactness from the start of the project, and the realization of object-oriented programming in the conventional imperative programming language C. We employed UML notations and their support tools to rigorous design modules, objects, and their interactions. In particular, we precisely described protocols for interfacing with other modules as shown in Section III-B. Some parts of protocols have been formalized in LOTOS [13] [14], and partly validated and verified with the help of a support tool-set [15] [16]. We also formally specified the interfaces and behavior of several modules, especially control API’s. We took a Hoare-style pre- and postconditions to specify the behavior [17] [18].

The application of formal methods was attempted to be rigorous rather than completely formal. The architecture, protocols and interfaces were partially formalized and some of their properties were proved. We needed an assurance that the proposed architectures and protocols work. The formalization ef-
forts focused on the behavioral aspect and the dynamic semantics of the protocols. For interface specifications, some modules were formally specified at the design stage before actual implementations began, but others were specified afterward as formal documentation of the code. As mentioned before, we used a combination of formal specification languages such as LOTOS and Larch. There is no universal specification language best suited for specifying the protocols and behavioral semantics of the system. It seems that the best approach is through a composite approach that uses several techniques suggested so far. The formalization process deepened our insight and knowledge on the design and increased our confidence on the design results even before the implementation began. The formal specification made it easy for the engineers constructing and using the modules to communicate with each other, and reduced the effort required to integrate them. In particular, the specifications of control API’s become one of the most important and effective referential materials during the implementation, testing, and system integration. We also achieved a large reduction of efforts at the integration stage of development life cycle, as the component interactions have been formally specified and verified within the design cycle.

For the implementation of MP3 audio decoder, we referenced an existing MP3 decoder written in C++. We had to translate it into C. For this, we have defined a set of translation rules, and decided to realize object-oriented programming in C. For example, we modularized the program by implementing abstract data types in terms of files. We supported information hiding and encapsulation of implementation details by separating the header files into public and private headers. We realized objects in terms of C pointers, and established systematic rules and conventions for object construction and destruction, parameter passing, and implementation of dynamic binding. In short, we have realized object-orientation in C through a well-defined set of rules and conventions. The resulting system turned out to be modularized, comprehensible, and easy to modify and maintain.

V. PERFORMANCE EVALUATION

The following table shows a quick summary of our prototype implementation. In the table, the audio represents an MP3 audio decoder while the video means an elementary MPEG-1 video decoder, i.e., a video decoder for pure video streams.

<table>
<thead>
<tr>
<th></th>
<th>source</th>
<th>binary</th>
<th>mem req</th>
<th>perf</th>
</tr>
</thead>
<tbody>
<tr>
<td>audio</td>
<td>7.4K</td>
<td>167K</td>
<td>80K</td>
<td>good</td>
</tr>
<tr>
<td>video</td>
<td>5.6K</td>
<td>41K</td>
<td>422K</td>
<td>poor</td>
</tr>
</tbody>
</table>

We ran a series of experiments to evaluate the performance of our prototype, and calculated its decoding capability on several processors. The MP3 audio decoder shows a satisfactory result on all processors. However, the MPEG-1 video decoder had quite a different performance on different processors because software MPEG-1 video decoding is CPU intensive. For example, on an Intel Pentium processor of clock rate 166Mhz, the decoder showed a decoding rate of 15 frames per second (fps), and about 17 fps on a Pentium 200 processor. The decoder can decode at about 40fps on a Pentium III 450 processor. These numbers are for decoding pure video streams; the decoding rate drops to 30 fps on the Pentium III 450 when both video and audio are decoded at the same time, i.e., when a system stream is decoded. As explained, the performance of MPEG-1 decoder was very poor. Considering the fact that our target systems are set-top boxes with Intel Strong Arm (SA) 110 processors, the result was quite frustrating. As thus, we quickly ported the video decoder to an Intel SA 110 processor-based board, and we found out the performance was about 18 20 fps.

At the beginning we worried about network bandwidth because the MPEG-1 standard requires a total bit rate of about 1.5Mbps. But it turned out that the real bottleneck was the processing power of set-top box CPUs. As a partial solution to this, we have implemented a method to dynamically drop video frames — e.g., a portion of B-pictures are dropped depending on the network bandwidth. The frame drop rate can be calculated statically once at the beginning of decoding or dynamically during whole decoding process. However, the frame drop approach could not provide an ultimate solution to the video decoder’s performance problem.

Later, we found out that it was a fatal mistake to take the public decoder written by MSSG. The MSSG version, that supports both MPEG-1 and MPEG-2, shows a performance that is two times slower than the popular MPEG-1 video decoder published by the University of California at Berkeley. At the beginning of the project, we did not worry about the performance at all and chose the MSSG decoder simply because the engineer in charge of video decoders was familiar with it. We are currently in the process of porting and integrating the Berkeley decoder to our system. If the Berkeley version should not perform to our expectation, we have at our last resort a commercial MPEG-1 decoder called an MPEG Expert System (MPX), a commercial-quality MPEG-1 decoder by SUN Microsystems, Inc., that is included in the SUN’s Java Media Framework (JMF).

In summary, we are very disappointed with the performance of our prototypes. The MP3 decoder performs very well, but the performance of the MPEG-1 decoder is unsatisfactory and way below our expectation. An MP3 audio decoder is realizable on the I-TV set-top boxes, but a software solution for MPEG-1 video services on the I-TV set-top boxes might be beyond our reach.

VI. CONCLUSION

A high-bandwidth Internet access service makes it possible to provide Internet-based multimedia services to home. We reported on our experience developing software MPEG-1 decoders for Internet TV set-top boxes, which are hooked to analog television sets for accessing the Internet. The decoders can be able to play in real-time MP3 audio and MPEG-1 video data stored on a media server, and they provide flexible and well-defined set of interfaces. The most distinguished feature of our development is the pursuit of rigor and exactness by applying formal methods to the development process, and the realization of object-orientation in C. In spite of this effort, our first prototypes performs poorly. An MP3 audio decoder lives up to our

5The Intel SA 110 processor of 233 Mhz is published to have a computing power of 2.1 MIPS.
expectation, but an MPEG-1 video decoder shows a terrible performance. We are currently improving the decoding rate of the MPEG-1 video decoder, but at the same time a bit pessimistic on providing software MPEG-1 decoders for I-TV set-top boxes.

REFERENCES


