Streaming Structured Audio

Marek Claussen
Technical University of Hamburg-Harburg
Arbeitsbereich Verteilte Systeme
21071 Hamburg, Germany
M.Claussen@tuhh.de

Ludger Solbach
Philips Semiconductors
Multimedia Competence Center
Sunnyvale, CA 94088, U.S.A.
Ludger.Solbach@philips.com

ABSTRACT

Structured Audio (SA) is an algorithmic low-bitrate coding standard for high-quality audio. Being a part of the MPEG-4 standard, it is expected to play an increasing role in multimedia applications such as computer games and interactive internet presentations. While first implementations have been developed for standard PCs using file-I/O, this paper is concerned with issues arising when the design goal is a streaming decoder running on an embedded multimedia processor with comparatively small external memory.

1 INTRODUCTION

MPEG-4 is the latest multimedia standard released by the Moving Pictures Experts Group (MPEG). The main aim of MPEG is to define state-of-the-art tools for the representation, transmission, and decoding of multimedia data. In contrast to the MPEG-1 and MPEG-2 standard employing perceptual coding methods, the scope of MPEG-4 is extended to object-based, parametric representations of synthetic content. Among these codecs is a new type of audio standard called "Structured Audio" (SA) (Scheirer and Kim, 1999; Scheirer, 1999; Vercoe et al., 1998), allowing for efficient and flexible coding of synthetic music and sound effects at a guaranteed reproduction quality. SA is the latest outcome of over 40 years of research in the field of synthetic music, building and improving on properties inherited from algorithmic sound description languages such as MUSIC-N (Roads, 1996), Csound (Bainbridge, 1997) and NetSound (Casey and Smaragdis, 1996). For an historic overview, see (Claussen, 2000).

This paper is structured as follows: Section 2 gives a brief overview of the Structured Audio Standard. An introduction of the TriMedia processor and the TriMedia Software Streaming Architecture (TSSA) is the content of Section 3. Section 4 describes a streaming SA implementation on the TriMedia processor. Section 5 concludes the paper and makes suggestions for future improvements.

2 STRUCTURED AUDIO

The data stream in SA consists of separate parts: an orchestra section written in SAOL (Structured Audio Orchestra Language) defines the instruments for sound generation and processing. The orchestra can be controlled through event lists written in SASL (Structured Audio Score Language) or alternatively through MIDI controls or dynamic instructions of the MPEG-4 system layer. For the incorporation of natural sound, the MPEG-4 standard provides the Structured Audio Sample Bank Format (SASBF) (Scheirer and Ray, 1998).

The MPEG-4 audio standard defines three different SA profiles: (ISO, 1999; Scheirer, 1998):

1. The score-based synthesis profile is defined for applications which require only minimal control over synthetic sound. This profile provides equivalent synthesis functionality to General MIDI.

2. The wavetable synthesis profile allows for the delivery of SASBF data and MIDI control, but not SAOL or SASL code. This profile is similar to the current MIDI DLS standard and is intended for applications, which DLS is targeting today: karaoke systems and simple multimedia presentations. The primary advantage of using MPEG-4 rather than MIDI DLS in these applications comes from the additional functionalities provided by MPEG-4.

3. The full profile incorporates all of SAOL, SASL, MIDI and SASBF for performing algorithmic and sample-based synthesis and effect processing algorithms. The full profile is the default for MPEG-4 and the majority of implementations is expected to provide this functionality.

3 IMPLEMENTATION PLATFORM

For the implementation of an efficient, full-profile streaming SA decoder several conditions must be met.
by the software environment and the host platform:

- The host platform has to offer sufficient computation power for executing complex SA performances in real-time. In a complex multimedia application, it is virtually impossible to meet real-time constraints without a realtime operating system underneath. Moreover, the host platform should support typical DSP operations like multiply-add efficiently, since they are an important part of most signal-processing algorithms used in SAO.LC orchestras.

- As the SA decoder comprises an in-time compiler and a dynamic execution unit, it has a rather complex structure that is most conveniently implemented in a high-level language like C or C++. Efficient compilers are required to map the high-level code onto the host processor for real-time performance.

- The RAM space accessible by the host processor should be large enough to allow for the storage of SA orchestras making extensive use of wavetable instruments.

A platform fulfilling these conditions is the TriMedia processor family (Philips Semiconductors, 1999). It provides a very long instruction word (VLIW) core with five instructions per clock cycle as well as interfaces and custom blocks for audio/video-I/O and data processing. It runs a realtime operating system and is programmable in C and C++. A powerful compiler exists for exploiting fine grain parallelism resulting in machine code that is highly optimized for the VLIW architecture. Typical signal-processing operations are supported by custom operators. SDRAM connected to the TriMedia processor by high-speed data and address busses allows for the storage of the whole decoder configuration including memory-extensive wavetables.

A large library of streaming multimedia components exists for this platform. These components are implemented according to the guidelines of the TriMedia Software Streaming Architecture (TSSA). It guarantees a conformance of the streaming TriMedia components regarding their I/O, error, setup and run-time configuration interface. For enabling a software component to cooperate flawlessly with others, a TriMedia software component should be implemented according to the TSSA rules.

Seven API functions need to be implemented for a TSSA-compliant software component: GetCapabilities(), Open(), GetInstanceSetup(), InstanceSetup(), Start(), InstanceConfig(), Stop(), Close(). The application calls GetCapabilities() to retrieve the component’s capabilities structure containing the information required to connect it to other components, such as supported input and output data formats. The Open() function is used to install an instance of the component, followed by GetInstanceSetup() for retrieving a template containing the component’s initialization defaults. The application may alter the setup structure and pass it back to the component via an InstanceSetup() call. Now that the component library is configured, the actual data processing can start by calling Start(). While running in an infinite loop, a component synchronizes itself with connected components through the availability of data packets on the connection queues as shown in Fig.1. The application accesses the components via control queues by calling InstanceConfig() in order to get status information or to trigger runtime changes. The components are terminated by calling the Stop() function. When the processing has finished and the instance of a component is no longer used, the instance handle can be returned to the library by calling the component’s Close() function, which also performs a cleanup of allocated resources. The communication between the application and each software component is bidirectional. While the application uses the aforementioned functions to address the component, it may give the component the opportunity to call error, progress, memAlloc, memFree and completion function callbacks implemented by the application.

For a more detailed introduction to the TriMedia hardware and software platform, the reader is referred to (Peplinski and Fink, 1999; Claussen, 2000).

4 STREAMING IMPLEMENTATION

The basis for the TriMedia streaming implementation is the MPEG-4 SA reference decoder SAOLC. It was the only SA decoder available when the project started. The main design goal for SAOLC was clarity in coding
style, not high efficiency. For our design objective of a streaming implementation, SAOLC has several shortcomings in terms of I/O, memory and error handling, which will be addressed below.

A Philips IREF board with a TriMedia TMI1000 DSP (100MHz) and 8MB of external SDRAM is used as the host platform. The main differences between a file-I/O based decoder and an embedded streaming decoder are in the handling of data-I/O, memory and errors.

4.1 I/O Handling

In a data-driven, non-preemptive multitasking environment, task scheduling is synchronized at function calls for data I/O. For this reason, a streaming component should retrieve and output data in a block-wise manner. One of the first steps for optimizing the SAOLC reference was replacing each one of the many fread/fwrite commands by calls to functions accessing the component's I/O buffers. These buffers are retrieved in the component's main processing loop from the FIFO queues provided by the operating system.

4.2 Memory Handling

Memory handling plays a minor role on host platforms offering a large amount of RAM with additional virtual memory. In case of embedded systems, it becomes a major aspect, especially when other components share the same platform in parallel. This is even more critical for SA, since as opposed to perceptual audio decoders like AC-3 or AAC, the memory requirements for a streaming SA decoder cannot be known in advance. As the decoding algorithm is not fixed but data-dependent, memory must be dynamically allocated and freed at runtime.

In the memory allocation scheme employed by the SAOLC reference, blocks as small as 8 bytes are allocated dynamically. This is very inefficient, since for such small blocks the additional memory required for memory management is in the same order of magnitude. Many of these small allocation calls could be removed in the streaming implementation by using local variables (i.e. stack space) instead. In cases where the available memory on the host platform is still not large enough, the streaming decoder may actually have to stop decoding the current stream. However, it should never exit its processing loop.

4.3 Error Handling and Reporting

Error handling and reporting for the streaming SA decoder had to be adapted to match the requirements of the streaming architecture. As the appropriate reactions to an error differ from one application to the other, the only thing a component should do is reporting it to the application by calling the error callback function. It is then up to the application to decide how to react. Besides printing an error message, it could invoke a stop routine, call instanceConfig() for changing the properties of the component, or simply return without any reaction.

Whenever the SAOLC reference decoder detects a fatal error, it exits and leaves it to the operating system to perform the cleanup. In an embedded system, a component should never exit its processing loop except on request by the application. All memory that was allocated by a component must also be freed by the component. This implies, that the design of the memory management has to be done with a great deal of care to prevent memory leakage. The component has to guarantee that all steps for freeing memory and resetting the component-specific values are taken without affecting concurrent components.

4.4 Computation Load

Similar to the case of memory requirements, information about the complexity of the SA content is not available when the streaming decoder is initialized. The fact that a worst-case scenario is difficult to anticipate makes it necessary to find solutions in cases when the CPU load is too high to guarantee real-time decoding. The SA standard offers two ways for graceful degradation of the synthesis process in case that the terminals performance is not high enough to guarantee real-time output. One possibility is using the cpuload parameter (ISO, 1999, 5.8.6.8.17), which allows for dynamic voice-stealing in an orchestra. To make use of this parameter, the host processor needs to monitor the current CPU load as a percentage of its real-time capability. In an overload situation, it may drop certain components of the orchestra as indicated by the content authors. The second possibility is using the high-priority flag (ISO, 1999, 5.7.3.3.7) for SASL score events. Sound events with the high-priority flag cleared may be prematurely terminated if resources are not available for scheduling an event for which the high-priority flag is set.

As long as there is no de-facto decoder reference that both content authors and decoder implementors agree on, such measures cannot be circumvented, even though they contradict SA's objective to achieve normative reproduction quality. As long as computation power is not as unbounded as artistic expression, the implementation of mechanisms for graceful degrada-
tion is mandatory.

The SAOLC decoder offers low computational efficiency. It builds a syntax tree from the SAOL data, which is evaluated for every sample during the SASL decoding phase. For future streaming implementations, optimizations with respect to computational efficiency should be applied like support of block-by-block processing and the usage of local registers for storing temporal values which do not need to be evaluated at each sample. Also, optimization of the core opcodes, representing the building-blocks for processing algorithms, would lead to more efficient decoding.

Based on the SAOLC reference code, real-time decoding on the TriMedia platform can be achieved for simple example streams only. Depending on the complexity of a given example stream, the duration of the decoding process exceeds real-time by a factor of 1 to 10. As of today, other decoders offering improved decoding efficiency are available. For example, the SAINT decoder developed at the ISL/LSI (Lausanne) [Zoia and Alberti, 2000], reportedly gains speed-up factors in the order of 20 by implementing advanced preprocessing of the syntax tree.

5 CONCLUSION

The implementation of a streaming SA decoder on an embedded multimedia processor brings up some problems which have not been discussed so far. The unparalleled flexibility in conjunction with the requirement of a guaranteed reproduction quality makes the SA decoder a component difficult to implement in efficient embedded systems with their high standards for runtime reliability and stability.

Although the SAOLC reference does not allow for real-time decoding, several findings for a streaming implementation on an embedded multimedia processor could be derived from the port. These findings should be useful for future implementations of streaming SA decoders based on more efficient decoding architectures like SAINT.

References


