

Audio framework

High-performance audio streaming framework for MOST

The ever increasing demand for in-vehicle infotainment and the wide recognition of MOST as the automotive multimedia standard, paved the way for the design of a low cost streaming framework. This comprehensive streaming framework is a strategic solution for the diverse OEM specific requirements. In this paper, a new audio-streaming framework for the MOST network is proposed.

By Sabeen A. and Manju Venugopal

The demand for development of a streaming platform arises mainly from the various requirements posed by OEM manufactures. A common framework which is easily portable across different platforms is an advantage for the system developer.

This common framework focuses on development of streaming applications independent of the NetServices layer and the underlying platform. The framework shall support the addition of various stream formats and multiple channel support with enough flexibility and easiness as requested by the OEMs.

By incorporating the proposed solution, the above requirements can be handled very effectively.

The following are the major advantages of this framework:

- ▶ Easily configurable with a high level of reusability.
- ▶ Easily adoptable to different platforms.
- ▶ Independent of NSL.
- ▶ Independent of the MOST speed-grade.
- ▶ Efficient memory utilization and low computing power demand.
- ▶ MISRA-C compliant.

Challenge

On various projects third party audio streaming platforms have been used.

This platform helped to show the prototype model easily. But when we progressed to the actual product, the following issues arised.

With the existing platform it was possible to add support for multiple channels. But the performance degraded and memory utilization increased. Furthermore support for compressed audio streams was needed, and the framework was so rigid that, to support this, parts of the code had to be rewritten. The final source code was too complex and made it difficult to

maintain. The MIPS and memory was almost entirely utilized. Finally, it was not possible to add new functions without considerably affecting the current performance.

So a different streaming platform with focus on easily configurable framework to add support for multiple stream types and channels with minimum computing power demand and memory utilization had to be thought of. The secondary focus was to make the framework independent of the underlying NetServices layer and MOST speed-grades.

There are partially implemented audio streaming platforms available in the market. But the issue with using such platforms is that they tend to consume a large amount of memory. It also causes heavy load for the system. The high cost is also a factor to decline the use of such platforms. For using some of the available platforms, one would have to change the system design to enhance the support provided by the framework.

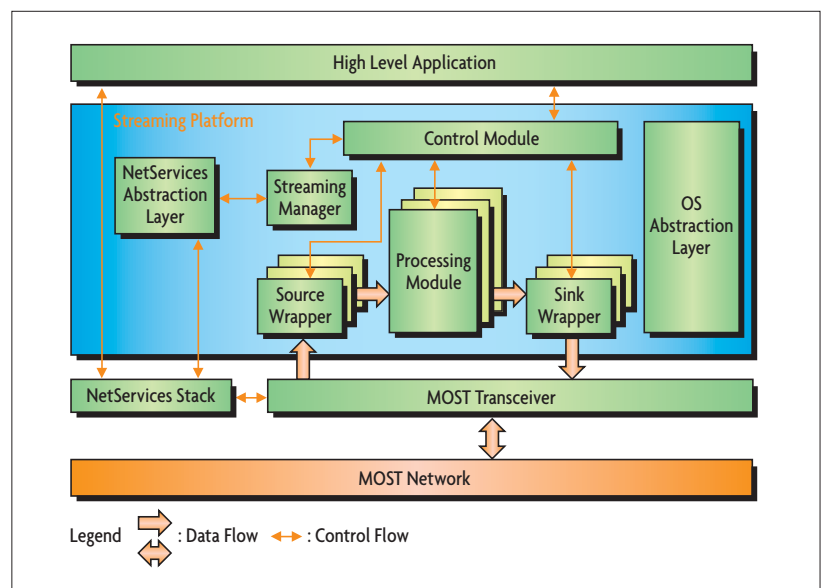


Figure 1. Architecture of a high-performance infotainment framework.

If the system requirement is to support multiple channels simultaneously, the existing platform should be scaled to a higher level. The scalability is not so easy in case of the existing platforms. This paper concentrates on deriving a solution for the above listed problems by designing a new lightweight platform, which could be used in multiple platforms without much scope for scalability related issues.

■ Architecture of the framework

A high-performance in-vehicle infotainment framework is required to support high media quality, with low delay. The proposed framework can be used along with a source or a sink device, for a slave node (**figure 1**). The operating system abstraction layer facilitates easy migration between various operating systems. The framework is designed in a way so that it can

be deployed in all MOST systems, independent of the speed grade.

The different logical layers are identified based on the functions performed.

The streaming platform with focus on least memory utilization is specifically designed for embedded system applications. The light weight framework minimizes the processing overheads and copy operations, usually related to stream transmission.

Following is the description of the different modules in the system:

- ▶ **Control module:** Resource management and communication handling between the streaming layer and higher level applications.
- ▶ **Streaming manager:** This module supervises the streaming channels and also handles the connection management related to the MOST transceiver.
- ▶ **Streaming layer:** A scalable streaming layer especially designed to support streaming audio. It comprises

of a source wrapper which handles the audio input from multiple sources, a processing module which implements the audio signal chain and a sink wrapper which routes the output audio data to speakers, headphones or back to the MOST transceiver.

- ▶ **NetServices abstraction layer:** A generic wrapper for communication with the NetServices layer.
- ▶ **OS abstraction layer:** The module which provides an interface to all basic functions of the OS.

In the following sections the functions of all layers are described.

Control module

The control module controls the functions and data flow between the source wrapper, the processing module and the sink wrapper. The control module controls the connection establishment between the source (sources)

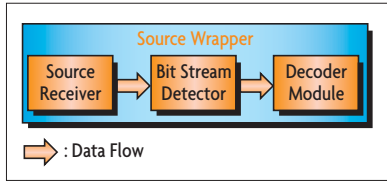


Figure 2. The source wrapper, as part of the streaming layer, can be separated in serveral sub-layers.

and sink (sinks). It receives the allocation, deallocation, connect and disconnect requests received from MOST via the higher level application and routes those messages to the streaming manager. The functions related to audio streaming including, but not limited to play, pause, stop signals etc. are handled by this module. The behavior of the source and sink wrappers are thus guided by the control module.

Streaming manager

The low level allocation of channels and routing functions are encapsulated within the network interface controller. The resources of the MOST transceiver are effectively administered by the streaming manager. This module processes the allocation and deallocation requests from the MOST node in the network which hosts the connection manager. This in return processes the channel allocation and deallocation requests as well as connection to and disconnection from the network.

Streaming layer

The basic streaming layer composes of the source wrapper, the processing module and the sink wrapper. The user can create multiple instances of the basic layer to provide support for numerous channels of audio data transmission.

The source wrapper layer can be split into different sub-levels as (figure 2):

- ▶ Source receiver
- ▶ Bit stream detector
- ▶ Decoder module

As the name suggests, the source receiver is capable of receiving data from various sources. The bit stream detector judges the incoming data and differentiates the various bit streams, i.e. AC3, DVD Audio, MPEG2 Audio, SACD and MPEG1/2 Audio. The signal is then routed to the corresponding decoders.

The output of the decoder module is fed to the processing module (figure 3). The processing module can be designed according to the systems requirements. The processing module comprises of different audio processing blocks, which implement the required signal chain.

The scratch buffers need to be designed by the user for audio processing requirements. The scratch buffer allocation shall be minimum with focus on reusability across the audio processing blocks.

The whole process can be designed by the user and seamlessly integrated to the streaming framework.

The sink wrapper is an interface layer between the processing module and the sink device. The data modified through the processing module is routed to the sink via the interfaces provided by the sink wrapper.

The streaming layer utilizes buffers for routing the audio stream. Double buffers are used at the input and output of the source and sink wrappers. For the processing, double buffers of requisite size need to be preallocated in the framework.

If a typical example of a 32-bit sample audio stream input is considered, a total of three buffers is required, each of size $4 \times 32 \times 2$.

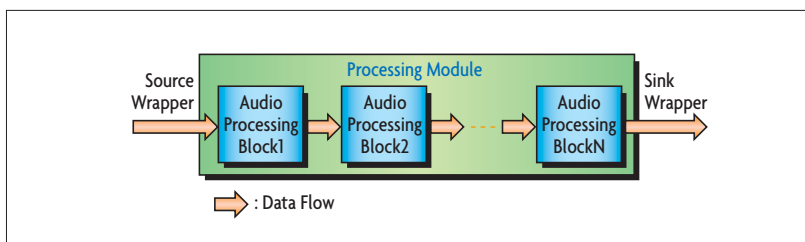


Figure 3. The processing unit can be easily adopted to the system requirement.

NetServices abstraction layer

The network service stack provides APIs to simplify the streaming mechanisms. The framework should be adaptable to the different versions of MOST stack available in the market. This is realized by the NetServices abstraction layer included in the framework.

OS abstraction Layer

The OS abstraction layer aims at the abstraction of operating system functions. This facilitates the integration of the framework into different platforms. sj



Sabeen A.

is working as Associate Technical Lead for embedded software products at Network Systems and Technologies (P) Ltd. He has over eleven years of professional experience in embedded software and has assisted in the design and implementation of various mobile and automotive products. sabeen.a@nestgroup.net



Manju Venugopal

started her career with Network Systems and Technologies (P) Ltd. soon after she completed her graduation. She is working as software designer at Continental Automotive, Singapore. She was ranked second in the university when she graduated from Cochin University of Science & Technology. itzmanju@gmail.com