A PC BASED PLATFORM FOR MULTICHANNEL REAL-TIME AUDIO PROCESSING

Hauke Krüger, Thomas Lotter, Gerald Enzner and Peter Vary

Institute of Communication Systems and Data Processing
Aachen University, Templergraben 55, D-52056 Aachen, Germany
`e-mail: {krueger,lotter,enzner,vary}@ind.rwth-aachen.de`

ABSTRACT

In this contribution a new concept for the rapid development of real-time prototypes for digital signal processing algorithms on a General Purpose Personal Computer (GP-PC) based on the Windows operating system (Windows OS) is presented. In this approach, hardware and algorithm related programming issues are separated in order to liberate the designer from problems which are not related to the development of the algorithm. The described platform therefore admits the complete focus of the developer on algorithmic aspects.

We will present the system along with example applications from acoustic echo cancelation and multichannel noise control. A third example from the field of speech coding demonstrates the computational power available on a state-of-the-art PC.

1. INTRODUCTION

The development of a product applying a new algorithm in the field of digital audio signal processing commonly consists of three development phases as depicted in Figure 1.

- Research involving the usage of high level programming languages (Development Phase 1) such as Matlab [1] or C/C++ [2]
- Real-time evaluation of the resulting algorithm (Development Phase 2)
- System design and integration for the fully developed algorithm (Product)

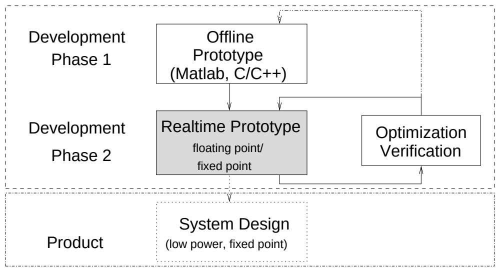

Fig. 1. Three phases during the development of a new algorithm

In phase 1 the basic functionality of the new algorithm is investigated. Phase 2 is important for the verification and optimization under real-time conditions. Phase 3 comprises the design of a product oriented system and a fixed point integration of the algorithm and in general is not in the scope of the algorithm developer.

For the real-time evaluation in phase 2, a system must be provided that on the one hand is capable to execute the new algorithm under real-time conditions as close to the final product as possible, and that on the other hand simplifies the transition from the offline prototype to the real-time implementation. While the final product usually requires a fixed point implementation of the algorithm, floating point arithmetic is sufficient for the evaluation platform.

Earlier, the real-time and computational requirements for the evaluation of new algorithms could only be satisfied by developing a new piece of hardware containing a powerful Digital Signal Processor (DSP) or designing a logical device such as a Field Programmable Gate Array (FPGA) [3]. Implementing a new algorithm on the developed hardware at least requires the knowledge of the hardware internals. Furthermore the algorithm may have to be reimplemented on the new device due to the programming in assembly language (DSP specific assembler) or a hardware description language (e.g. Very High Speed Integrated Circuit Hardware Description Language (VHDL), [4]). Often the hardware related audio setup highly depends on the algorithm prototype to be implemented (e.g. fixed number of audio channels). After the real-time implementation has been completed for one algorithm the next one may require a new hardware setup. Under certain circumstances the existing prototyping platform does not comply with the new setup and must therefore be developed from scratch.

User interaction at runtime is also an important feature that helps to optimize and verify algorithm parameters. For example in the field of hearing aids the perfect setup for a new algorithm can only be found by a person who himself is affected by hearing limitations (which most commonly is not the algorithm developer). Adding these interaction mechanisms requires the integration of additional hardware and software to the real-time prototyping platform.

The increase of performance of General Purpose Processors (GP-CPUs) recently helps to alternatively come up with PC based platforms that are capable to execute algorithms mostly under real-time conditions. The architectural advantages that DSPs provide can be outbalanced by the high clock frequencies of state-of-the-art GP-CPUs at the expense of a higher power consumption which can be tolerated during the real-time prototyping phase. The Windows operating system by definition is not a real-time operating system. The response time of the system however has decreased during the last years and is nowadays sufficiently short to fulfill real-time constraints with low latencies at least for most of the time (soft real-time). Using GP-PCs, an algorithm designer can now integrate new algorithms in the field of audio processing based on various Software Development Kits (SDK) such as ASIO [1] [5] or DirectX [6] and reach latencies even below 5 ms (ASIO).

However, we have experienced that even though the provided development kits are well documented, designers often desire to implement their algorithms without consideration of particular constraints of the underlying hardware and software. In the ASIO-SDK for example the user has to deal with hardware specific sample formats and the methodologies to open/close the device driver for supported soundcards.

In this paper we present a new technique that enables the designer in the field of digital audio processing to fully concentrate on the development of algorithms without wasting too much energy in dealing with the audio hardware in order to create a real-time prototype. With this approach a high level programming language can be used also for the real-time prototype. Under certain circumstances software components from the offline development phase can even be reused without any modifications. The new technique furthermore comprises mechanisms for a real-time messaging mechanism that, especially in combination with a Graphical User Interface (GUI), will allow even the non-expert to optimize the parameters of the algorithm by simply clicking buttons or moving sliders.

The whole system is based on state-of-the-art soundcards. Setting up the computer for new hardware conditions only means to purchase a new audio device for a PC and to install the driver.

2. PRINCIPLES OF AUDIO PROCESSING

Real-time audio processing is usually based on a double-buffering architecture: While in one task the audio samples from the Analog-To-Digital Converter (ADC) are stored in one input buffer and played back from one output buffer via the Digital-To-Analog Converter (DAC), the audio samples in the other input buffer are transferred into the other output buffer in a parallel task applying algorithm related processing of samples. After the collection of samples has been completed for one buffer length the buffers are switched.

It is desirable that the only task for the algorithm developer is to implement the routine processing the data in the input buffer in order to transfer it to the output buffer applying the algorithm related functionality. The algorithm will become active whenever a complete buffer has been filled (buffers are switched) and must terminate before the next buffer will be complete (buffers are switched). When using the mentioned SDKs the approach is more complicated due to the variety of underlying software concepts, e.g. opening a driver or dealing with hardware specific sample formats.

[1] ASIO is a trademark of the Steinberg Media Technologies AG

3. NEW APPROACH

In order to support the design of the real-time application we have created a system that separates the hardware related topics from those related to the algorithm. Therefore we have divided each real-time application into three entities: The *hardware*, the *host* and the *algorithm* module. Figure 2 depicts the three-modules-architecture to be explained in the following sections.

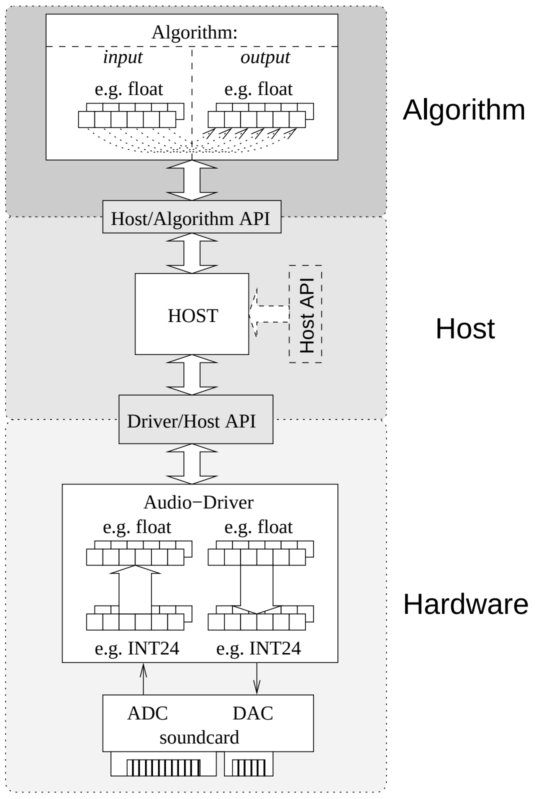

Fig. 2. Hardware, host and algorithm module.

3.1. Hardware Module

The hardware module is based on one of the SDKs mentioned. It is only linked to the host. Via an Application Programmable Interface (API) the host has access to the hardware. Functionality such as opening a driver and obtaining information on the hardware setup is provided. Most device drivers use 16-, 24 or 32-bit fixed point datatypes for the audio input and output. In order to prepare the processing of the audio input during real-time execution the hardware module converts the hardware specific input buffers into buffers of a hardware independent datatype (floating point datatype). After the conversion the host is notified that a new buffer is ready for being processed.

3.2. Algorithm Module

The algorithm module processes the data in the hardware independent buffers that the host provides. This is where the algorithm designer will implement the application. The algorithm module is linked to the host via an API. Besides the processing the algorithm module has the opportunity to negotiate samplerate, buffersize and amount of input and output channels with the host. Messages for a runtime user interaction can be handled in the algorithm-module internally or can be received from the host-module. By setting new values for parameters that are used in the main algorithm-procedure from within the message receiving routine, the user interaction will directly have an impact on the audio processing.

3.3. Host Module

The host module implements the bridge between the hardware and the algorithm components. It can be an application itself or may be used as a component controlled by an application (Host-API). Before processing the hardware must be instructed to open the driver and to describe the hardware capabilities for processing (available amount of input/output channels, sample-rates and buffersizes: *hardware* constraints). The algorithm module is registered with the host and provides information on the audio setup it requires (expected amount of input/output channels, sample-rates and buffersizes: *software* constraints). A match between hardware and software can be found by the user and is passed to the hardware and the software module before the processing is initiated. At runtime the hardware independent buffers are transferred from the hardware module to the algorithm for processing and back. In parallel the host may exchange messages with the algorithm module in order to enable user interaction.

3.4. Practical Aspects

We have experienced two different approaches to use the host: In one approach the host is implemented as a Graphical User Interface. The algorithm-module comes along with an additional graphical user dialog window and handles the runtime messaging mechanism internally. In another approach the complete host is controlled via a powerful API. The real-time algorithm module is registered and can be controlled using the messaging mechanism provided by the host during runtime. The idea is to integrate the real-time processing functionality into any kind of application and graphical environment, e.g. a web browser.

The strongest feature of the developed platform is the interchangeability of the components. The driver in general is interchangeable at runtime in the Windows operating system (OS). The algorithm module is implemented as a Dynamic Link Library (DLL) and can be loaded and unloaded at runtime. To have the components interchangeable at runtime means that different algorithms can be sequentially executed on different soundcards without the need to restart the host application.

Due to the interchangeability of the components the host in combination with the algorithm can also be used as an offline audio processor. This will enable the designer to verify the algorithm with predefined audio signals. No modification will be required to switch from the offline algorithm development phase (phase 1) to the real-time prototyping phase (phase 2).

4. DEMONSTRATION

The proposed technology will be presented on a portable PC with state-of-the-art audio devices used for analog-to-digital conversion. We will present the following example applications of real-time audio processing, switching from one to the other by simply loading and unloading different algorithm modules:

4.1. Acoustic Echo Canceler

Acoustic echo in hands-free communication systems is due to the echo path from the loudspeaker to the microphone as shown in Figure 3. Given the received signal, an echo canceler reconstructs and removes the echo signal from the microphone signal before transmission [7]. Real-time simulation of acoustic echo cancelers is necessary to prove the robustness of the adaptation in realistic acoustic environments. As shown in Figure 3, the acoustic echo canceler has two input and two output channels. The corresponding set of sample buffers is a very user-friendly interface for the algorithm developer. Very little care is required about hardware related issues (ADC, DAC, audio playback). A multichannel setup is of course possible, provided that enough hardware channels and sufficient processing power are available.

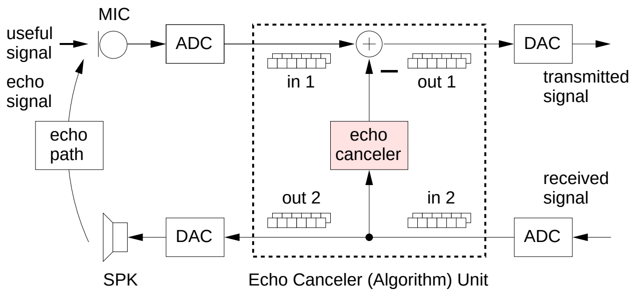

Fig. 3. Schematic of an acoustic echo canceler embedded in the real-time setup.

On a commercial PC (Pentium IV, 1800 MHz) we have implemented an acoustic echo control unit with two input and two output channels and the following features (not shown in Figure 3):

- Sampling rate conversion from 32kHz (soundcard) to 8kHz (algorithm).
- Frame-based adaptation of an echo canceler and a postfilter for residual echo suppression in the Discrete Fourier Transform (DFT) domain [8].
- A frame-shift (algorithmic delay) of 64 samples at a sampling rate of 8 kHz and 6 DFTs/IDFTs of length 512 per frame-shift.
- A computational complexity of about $2 \cdot 10^6$ multiply/accumulate instructions per second.
- Compensation of 20 ms hard- and software latency in the echo canceler path of the system.

4.2. Superdirective Beamformer

A beamformer for noise reduction with multiple microphones as depicted in Figure 4 is embedded in the real-time setup. The beamformer consists of M time domain filters.

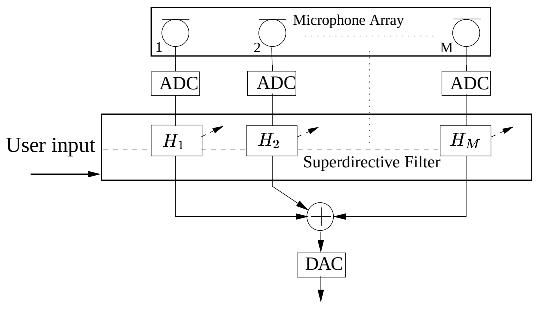

Fig. 4. Superdirective beamformer embedded in real-time setup.

The filters are designed with constrained superdirectivity [9], i.e. the tradeoff between degree of superdirectivity and susceptibility toward microphone or amplifier mismatch can be regulated. Due to the versatility of the real-time system, all parameters of the superdirective beamformer can be adjusted on the fly. First, the number of microphones, the samplerate and the number of microphones used can be adapted. Second, the degree of superdirectivity and the desired spatial direction of the beamformer can be changed during runtime. For that purpose the filter design procedure is implemented as background task, which updates the coefficients as soon as the design process is completed. At a sample rate of $f_s = 32$kHz the filter-and-sum operation can easily be realized with $M = 8$ signals and filter lengths of 128 taps using a standard PC and a state-of-the-art multichannel soundcard.

4.3. Two Channel Real-time AMR Speech Codec

In order to demonstrate the computational power available we present the application of a two channel (stereo) speech codec: In this application the left and right channel input signal is processed identically as depicted in Figure 5: The incoming audio signal is processed by the Narrow Band Adaptive Multirate (NB-AMR) Speech Encoder first and the resulting bit-stream decoded in the NB-AMR Decoder, both in floating point implementation following the ETSI-standardization [10]. The audio signal obtained from the soundcard (samplerate $f_S = 32$ kHz) must be lowpass filtered and resampled prior to processing due to the narrow band characteristic expected by the speech codec ($f_S = 8$ kHz). Before passing the decoder output ($f_S = 8$ kHz) to the soundcard for playback ($f_S = 32$ kHz) the samplerate must be converted by resampling and lowpass filtering.

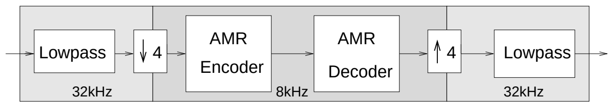

Fig. 5. NB-AMR Codec Simulation for each channel.

The codec's worst case performance is specified as 16.75 WMOPS (Weighted Million Operations Per Second: Measurement for complexity using a set of basic operations defined by ETSI in fixed point arithmetic [11]). Thus the complexity for the stereo setup is 33.5 WMOPS. Combining the two instances of the codec for left and right channel with the rate conversion filters (60 Taps FIR-filter) for up- and down-sampling, a portable PC, containing a 500 MHz Pentium III and running an external audio device connected via Universal Serial Bus (USB), can execute the algorithm in real-time.

5. CONCLUSIONS

In this paper, we introduced a platform that aims at rapid real-time prototyping in the field of digital audio signal processing. The capabilities of GP-PCs and especially the Windows OS can be exploited to avoid time consuming development of new hardware. In that context we proposed a technique that helps to simplify the process of algorithm implementation for the algorithm designer. The designer can completely focus on the algorithm implementation without the need to deal with getting in contact with hardware related routines. A powerful messaging mechanism supports runtime user interaction for parameter optimization and verification, even controlled by a person who has no background in the field of digital signal processing. A very interesting feature of the applied technology is the possibility to use the same software component for offline processing and the interactive real-time prototype of a new algorithm. In the context of three real-time prototyping example applications we demonstrated that a commercial PC and soundcard setup is capable to execute even complex algorithms in real-time.

6. REFERENCES

[1] K.D. Kammeyer, V. Kühn, *Matlab in der Nachrichtentechnik*, J. Schlembach Fachverlag, 2001.

[2] B.W. Kernighan, D.M. Ritchie, *The C-programming-language*, Prentice-Hall, 1990.

[3] R. C. Restle, "Choosing between DSPs, FPGAs, μP and ASICs to Implement Digital Signal Processing", www.eg3.com, Jan. 2000.

[4] Y. Sudhakar, *VHDL Starter's Guide*, Prentice-Hall, 1998.

[5] Steinberg Media Technologies AG, "ASIO 2.0 Audio Streaming Input Output 2.0 Software Development Kit, Release 1", www.steinberg.net.

[6] Microsoft Corporation, "DirectX SDK", msdn.microsoft.com, 2003.

[7] C. Breining, E. Hänsler, "Acoustic echo control, an application of very–high–order adaptive filters", *IEEE Signal Processing Magazine*, pp. 42–69, September 1999.

[8] G. Enzner, P. Vary, "Robust and Elegant, Purely Statistical Adaptation of Acoustic Echo Canceler and Postfilter", Proc. Intl. Workshop on Acoustic Echo and Noise Control (IWAENC) 2003, Kyoto, September 2003.

[9] M. Dörbecker, "Multi-channel algorithms for the enhancement of noisy speech for hearing aids", *Ph.D. Thesis (in German)*, Aachener Beiträge zu digitalen Nachrichtensystemen, edited by P.Vary, vol. 10 (ISBN 3-86073-439-3), 1998.

[10] ETSI, *GSM 06.71: Digital cellular telecomunications system (Phase 2+); Adaptive Multi-Rate (AMR); Speech Processing Functions; General Description*, European Telecommunications Standards Institute, 1998.

[11] K. Järvinen, "Standardisation of the adaptive multi-rate codec", in *European Signal Processing Conference (Eusipco)*, Tampere, Finland, 2000, IEEE.