RTProc: Rapid Real-time Prototyping for Audio Signal Processing

Hauke Krüger, Thomas Schumacher, Thomas Esch, Bernd Geiser, Peter Vary
Institute of Communication Systems and Data Processing
RWTH Aachen University, D-52056 Aachen, Germany
{krueger,schumacher,esch,geiser,vary}@ind.rwth-aachen.de
www.ind.rwth-aachen.de

Abstract: In this contribution the RTProc system for the rapid development of real-time prototypes for digital audio signal processing algorithms is presented. RTProc enables even unexperienced programmers to transform the first implementation of a new algorithm in Matlab into a stand-alone real-time demonstrator written in C/C++ in a very efficient way. In order to achieve this goal, the RTProc software architecture is defined such that hardware and algorithm related programming issues are separated. All hardware related programming aspects are hidden so that the algorithm developer can focus on the implementation of the algorithm.
Different application scenarios are supported by RTProc to operate on two different platforms: General purpose PCs (RTProcPC) and the Analog Devices ADSP-21369 EZKIT [1] embedded DSP target (RTProcDSP). While RTProcPC enables real-time processing of algorithms realized in C/C++ with a minimum system latency of approximately 5 ms, even lower system latency can be achieved by RTProcDSP if necessary.
Compared to earlier versions of RTProc [3], the current version has been extended according to the needs of algorithm developers to guide users through all development phases in digital signal processing, starting from first Matlab simulations to the final highly efficient implementation in fixed point arithmetic.
Example real-time prototypes for noise reduction for mobile communication, simulation of speech and audio codecs, and Matlab based room acoustic measurements will be demonstrated to show the potential of RTProc.

1 Introduction

The conventional development of a product applying a new audio signal processing algorithm commonly can be divided into three development phases as depicted in Fig. 1.

Development Phase I: Algorithm Development
First investigations on a new algorithm in general involve the usage of high level programming languages such as Matlab. Existing libraries are employed for algorithm exploration. Since the algorithm is operated with audio samples in offline manner, aspects such as computational complexity do not play a significant role yet.

Development Phase II: Real-Time Evaluation
In phase II, the algorithm developed in phase I is tested under real-time conditions[1]. A highly efficient programming language such as C/C++ is chosen since processing efficiency is crucial

[1] Since most operating systems on general purpose PCs do not fulfill hard real-time constraints, the term *real-time* refers to soft real-time conditions in RTProcPC.

to reach the required program execution speed. In order to identify a suitable realization of the new algorithm, further aspects have to be taken into consideration such as portability to fixed point arithmetic, algorithmic delay, and further optimization of the computational complexity.

Development Phase III: Product Integration
In the final development phase, the new algorithm is realized on the target product platform, e.g., a low power fixed point DSP or microcontroller unit (MCU). This phase can be well prepared in phases I and II and, in most cases, is not in the main focus of the algorithm developer.

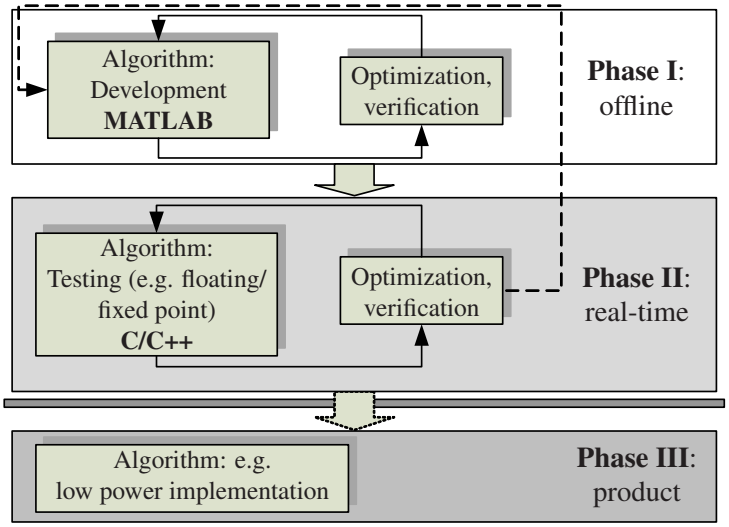


Figure 1 - Phases in the development of real-time audio signal processing prototypes.

Rapid real-time prototyping as the outcome of phase II is very important for the successful introduction of new algorithms: From the developer's point of view, it enables efficient algorithm optimization, evaluation, and verification in real-world scenarios and customer field tests. In addition to that, a real-time prototype is a highly valuable instrument to market new solutions since potential customers can get in touch with prototypes of future products before entering development phase III.

Development phase I is related to the development of high level programming language software only. A realization of the same algorithm to be tested and optimized under real-time conditions, however, involves a lot of additional programming effort in order to connect the new algorithm to audio hardware as well as to create a control user interface which allows to adapt algorithm parameters during runtime. In practice, during phases I and II, due to the different programming languages, at least two versions of the algorithm exist. Each version is verified and optimized in each of the phases independently. The synchronization of both versions (indicated by the dotted arrow in Fig. 1) is difficult and time consuming. In summary, the conventional development is inefficient and prohibits short development intervals.

The goal of `RTProc` is to simplify the development process to reduce the time spent for the creation of real-time demonstrators. Due to a clear separation of functional components in `RTProc`, all hardware related programming aspects are hidden from the developer, and almost no additional effort is required to switch from the offline to the real-time development phase. In addition, easy-to-use mechanisms for the automatic generation of source code enable to, e.g., attach graphical control interfaces to prototypes or to realize common programming tasks in a very convenient way. Special attention has been payed to realize `RTProc` as an algorithm development system according to the needs of the algorithm developers: The integration of real-time audio processing functionality in Matlab enables to smoothly make the transition from offline

development to real-time processing by mixing Matlab and C/C++ code without significant additional programming effort due to RTProc code generation tools. Consequently, development phase I and II are merged such that problems related to different algorithm versions for offline and real-time processing are no longer present. Moreover, tools are provided to measure and optimize the timing behavior during real-time processing and algorithm efficiency in terms of computational complexity.

2 Principle of RTProc

RTProc is based on a software architecture realized in C/C++ that decomposes audio signal processing applications into three functional units, the **driver**, the **algorithm** and the **host** com-

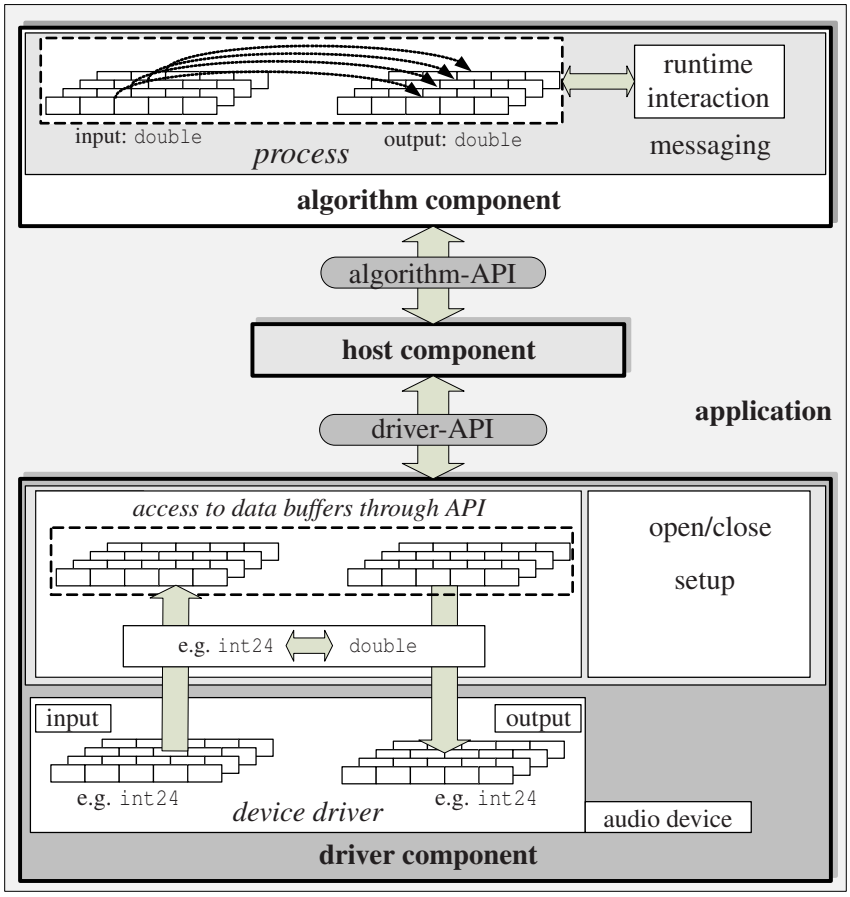


Figure 2 - Basic principle of RTProc.

ponent, as depicted in Figure 2.

The Driver Component
The driver component has access to the audio input/output device. In practice, the mechanism to interact with the hardware differs depending on the target platform and, on the PC, the chosen hardware access technology. In order to provide one unique interface for all applications, the driver component maps all basic hardware and system specific functionality to a simple interface, the RTProc driver application programmable interface (driver-API). During real-time processing, the driver component converts all hardware specific data types into hardware independent data types and grants access to the audio samples for further processing through the driver-API.

The Algorithm Component

The algorithm component has access to the audio buffers prepared by the driver component. Its main purpose is to process the input audio samples according to the algorithm to be realized. The resulting output samples are written to the reserved output buffers. The algorithm component is completely independent from the rest of the audio processing application and the hardware. The *runtime interaction* block in Fig. 2 enables to modify processing parameters and therefore the algorithmic behavior during audio processing.

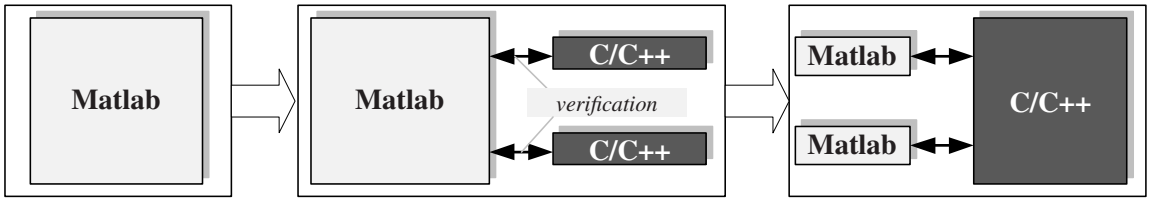
The Host Component
The host component controls the driver and the algorithm component through the algorithm- and the driver-API. It is the key element of all audio processing applications and routes all user interaction to the other two components. During audio signal processing, it transfers all audio data between driver and algorithm component.

In `RTProc` aided development, the developer realizes the new algorithm as an algorithm component independently from any hardware aspects. Since driver and host components have been implemented as part of `RTProc` already, it is sufficient to focus on the algorithmic functionality only.
`RTProcPC` is currently realized for the Windows Operating System. Driver components based on the **ASIO** [8]and the **directKS** [5] technology are supported to enable multichannel audio processing with system latencies of less than 5 ms. Enhanced `RTProc` driver components to support functionality such as additional real-time audio file routing, automatic resampling and asynchronous audio processing are available to cover the whole bandwidth of applications. Two different host components are available, one to integrate real-time processing in Matlab (the **Matlab host**), and a generic library to offer the opportunity to add `RTProc` to customized applications (the **generic host**). **GUI host** applications make use of the **Matlab** and the generic host component, respectively, to provide graphical user interfaces (GUI) for real-time audio processing control.
The embedded DSP version `RTProcDSP` targets applications which require even lower system latency and hard real-time. The driver and the host components are both part of a proprietary lightweight `RTProc` DSP operating system in this case. An optional connection to a general purpose PC can be established on the basis of an onboard hardware component to realize a serial communication link.

3 `RTProc` Aided Algorithm Development

`RTProc` aided algorithm development significantly benefits from the merge of the development phases I and II. In that context, the different operation modes of the `RTProc` Matlab host application, the *offline*, the *hook up*, and the *high efficiency* mode, play an important role: In general, the developer starts algorithm exploration in the *offline* operation mode of the host. In this mode, audio files stored on the harddrive or in the workspace are processed in a convenient way and stored back to harddrive or workspace. In order to realize a specific algorithm, the algorithm developer provides a callback function in analogy to the algorithm component main signal processing function. This callback function is called by the Matlab host application on a buffer-by-buffer basis. All other functionality, e.g., the decomposition of the input signal into buffers is managed by the Matlab host application.
If the buffer-by-buffer processing callback function execution time is sufficiently short, the Matlab implementation of the algorithm can also be executed in real-time. For this purpose, the `RTProc` Matlab host application can be operated in the *hook up* mode: The `RTProc` software architecture including the `RTProc` driver component is loaded in the background of Matlab and synchronizes Matlab to the soundcard I/O. Compared to the *offline* mode, the Matlab signal processing callback provided by the algorithm developer reads and writes audio samples from

and to the audio device.
Due to the required synchronization overhead, this purely Matlab based real-time audio signal processing approach is applicable only in case of algorithms with a low computational complexity and for applications with moderate latency constraints. In all other situations, the realization of an algorithm component in C/C++ according to the `RTProc` software architecture is the more efficient approach to follow. In order to enable a smooth transition from the offline version of the algorithm in Matlab to a C/C++ implementation, the `RTProc` Matlab host application enables to mix Matlab functions and functions which are part of the C/C++ algorithm component arbitrarily in the *offline* mode. Consequently, the code transition process of an algorithm from Matlab to C/C++ can be realized very efficiently as follows:
At the beginning, most algorithm functionality has been realized and verified in Matlab as shown in Fig. 3 a). In order to convert the Matlab code into a C/C++ based real-time version of the algorithm, all functional parts are slowly ported to C/C++ in a step-by-step procedure and from inside to outside. Two typical situations in the transition process are demonstrated by the examples in Fig. 3: In part b) of the figure, the first functions have been converted to C/C++ in



a) Start of transition from Matlab to C/C++

b) Early transition stage

c) Advanced transition stage

Figure 3 - Porting from Matlab to C/C++ in `RTProc`: Examples for situations at the beginning (a), in an early (b) and in an advanced (c) development stage.

an early development stage, whereas in part c), nearly the full algorithm has been converted to C/C++ in an advanced development stage.
Since in this smooth transition process, each single function can be easily verified against its Matlab counterpart, implementation failures, which are often very time consuming to find in conventional development, are avoided. The effort to call C/C++ functions from within Matlab is only minimal since all required code is generated automatically by an `RTProc` tool based on simple meta information provided by the algorithm developer.
Once all functionality is completely implemented in C/C++ and verified, the Matlab Host GUI can be switched into the *high efficiency* operation mode. In this mode, the time-critical signal processing tasks are executed completely in the background of Matlab without interfering with any Matlab functionality. No modification of source code or recompilation of the algorithm component is required to run the algorithm in real-time in the *high efficiency* operation mode. And even though signal processing is done in the background, the algorithm developer benefits from the `RTProc` Matlab integration in the *high efficiency* mode: A non-time-critical bidirectional data link between Matlab and the C/C++ based algorithm component in the background enables to, e.g., compute processing parameters such as fixed filter coefficients in Matlab and to send those parameters to the algorithm component. Here, the parameters instantaneously have an impact on the signal processing algorithm.
In the last phase of the development, the real-time prototype should be prepared for deployment to customers. In this context, an operation independently from Matlab as a stand-alone real-time demonstrator can be realized based on the *deployable host* application. Purely C/C++ based algorithm components which have been developed with the Matlab host application can

be loaded without any source code modification or recompilation.
If an embedded solution is desired, the RTProc software architecture allows for a convenient conversion of an algorithm from RTProcPC to RTProcDSP since both versions are highly source code compatibility. A step-by-step procedure to convert Matlab functions into C/C++ code for the embedded DSP in analogy to the methodology described in the context of the PC version of RTProc is realized by means of the serial communication link described in Section 2.

3.1 RTProc Runtime User Controls, Tools and Extensions

Once an algorithm component has the desired functionality, a very important task is to create a runtime user control in order to modify signal processing parameters while the algorithm operates (runtime user interaction). Different mechanisms are supported by RTProc to reduce the programming effort in different development phases:

Generic User Controls
Simple user controls are sufficient for real-time algorithm exploration, verification and optimization. The RTProc generic runtime configuration mechanism is based on a description of parameters to be adapted during runtime in a software construct, in Fig. 2 shown as the *runtime interaction* block. Based on this description, graphical user interfaces are *generated* dynamically by the host. Consequently, the specification of the runtime control is independent from the host and no knowledge about GUI programming is required. All source code required to manage the generic runtime configuration is generated automatically by the RTProc Generic Runtime Compiler.

QT based User Controls
More sophisticated user controls can be realized based on QT [4] and the RTProc QT Extension. A framework involving code generation and the QT designer enables a drag-and-drop design methodology in which only very little portions of source code have to be added by hand. Due to the powerful GUI features of QT, very attractive graphical user interfaces can be designed.

In addition to the basic RTProc functionality, tools and extensions are provided with RTProcPC to support the algorithm developer in optimizing the real-time operation of newly created prototypes, e.g.,

- the RTProc Full Speed Data Logger to store data produced during real-time processing on hard drive without interfering with real-time processing
- the RTProc Matlab Data Reader to read the stored data for timing accurate algorithm analysis afterwards in Matlab
- The RTProc Communication Extension to connect the audio signal processing algorithm to real communication networks such as ISDN, GSM and Ethernet
- The RTProc Codec Library as a collection of standardized and non-standardized speech and audio codecs to be used in combination with the communication extensions for real-time simulations of state-of-the-art and future communication systems.

4 Example Applications

RTProc was successfully used to develop various audio signal processing prototypes. The following are example projects which will be presented at the 20th conference ESSV:

Noise Reduction in Mobile Communication: When a speech communication device is used

in environments with high levels of ambient noise, the noise picked up by the microphone significantly impairs the quality and/or the intelligibility of the transmitted speech signal. In order to get a reliable separation from the noise signal, noise reduction algorithms have become part of modern communication devices. Based on RTProc, a speech enhancement demonstrator was developed to demonstrate several state-of-the-art noise suppression techniques based on (advanced) statistical approaches [6] as well as psychoacoustical techniques [7]. In order to conduct field tests in realistic scenarios, the RTProc communication extensions enable to evaluate the performance of the speech enhancement algorithms in the context of real GSM based telephone calls.

Real-time Simulation of a Codec Candidate for Standardization: At the authors' department, an embedded speech and audio codec has been developed which is suitable for "super-wideband" (50 Hz – 14 kHz) audio communication over packet-switched networks. This codec has been successfully submitted to ITU-T by Huawei and ETRI as a candidate [2] for the upcoming super-wideband extensions of Rec. G.729.1 and G.718. For the purpose of real-time evaluation and demonstration of this codec, an RTProc real-time demonstrator has been created which comprises the encoder and the decoder components. Using a graphical interface, the user can adjust the sent and the received bit rate in real-time. Furthermore, the instantaneous packet loss rate can be controlled to assess the codec performance under realistic channel conditions.

Matlab tool for the measurements of room acoustics: Based on the RTProc Matlab host, a tool was developed to measure room acoustics. For each measurement, a test signal is played back by a loudspeaker, recorded by a microphone and finally evaluated in order to compute the room impulse response. Signal generation and measurement evaluation are realized most efficiently in Matlab whereas simultaneous and reproducible playback and recording of audio signals must be realized by RTProc operating in the background since this functionality is not available in Matlab. The tool is an open platform in which signal generation and data evaluation functionality can be exchanged. The RTProc communication extensions allow to enhance the functionality to measure the characteristics of communication systems such as GSM or ISDN.

5 Conclusion

In this contribution RTProc has been presented as a powerful system for the rapid development of audio signal processing prototypes on different platforms. Compared to the conventional way, RTProc can simplify the development process and shortens the development time.
Key element of RTProc is a software architecture according to which each application is decomposed into different functional blocks. This architecture enables to liberate the developer of a new algorithm from all programming aspects which are not related to the actual algorithmic functionality. Compared to other component based approaches for audio signal processing, RTProc has been designed according to the needs of algorithm developers and offers guidance throughout all common development phases. Besides this, tools are provided with RTProc to support the developer to create runtime user controls, to verify the proper functionality even under real-time conditions, and to optimize the new algorithm.
Currently, with RTProcPC and RTProcDSP two platforms are supported based on Microsoft Windows PCs and the ADSP-21369 EZKIT from Analog Devices respectively[2].

References

[1] ANALOG DEVICES, INC.: *EZ-KIT Lite for Analog Devices ADSP-21369 SHARC Proces-*

[2] Further information about RTProc is available at http://www.ind.rwth-aachen.de/~rtproc.

sor. http://www.analog.com, 2008.

[2] BERND GEISER, HAUKE KRÜGER, HEINRICH LÖLLMANN, PETER VARY, DEMING ZHANG, HUALIN WAN, HAI TING LI, AND LI BIN ZHANG: *Candidate Proposal for ITU-T Super-Wideband Speech and Audio Coding*. In *Proc. of Intl. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, Taipei, Taiwan, Apr. 2009.

[3] HAUKE KRÜGER, THOMAS LOTTER, GERALD ENZNER, AND PETER VARY: *A PC based Platform for Multichannel Real-time Audio Processing*. In *Proceedings of International Workshop on Acoustic Echo and Noise Control (IWAENC)*, Kyoto, Japan, Sept. 2003.

[4] JASMIN BLANCHETTE AND MARK SUMMERFIELD: *C++ GUI Programming with Qt 4*. Prentice Hall, 2006.

[5] MICROSOFT CORPORATION: *Windows Driver Development Kit*. http://msdn.microsoft.com.

[6] PETER VARY AND RAINER MARTIN: *Digital Speech Transmission - Enhancement, Coding and Error Concealment*. Wiley, Chichester, 2006.

[7] STEFAN GUSTAFSSON, RAINER MARTIN, PETER JAX, AND PETER VARY: *A Psychoacoustic Approach to Combined Acoustic Echo Cancellation and Noise Reduction*. IEEE Transactions on Speech and Audio Processing, 10(5):245–256, July 2002.

[8] STEINBERG SOFT- UND HARDWARE GMBH: *ASIO Interface Specification, v2.0, 1997-1999*. http://www.steinberg.de, 1999.