A VERSATILE DSP-SYSTEM FOR STUDENT-PROJECTS ON EMBEDDED REAL-TIME AUDIO SIGNAL PROCESSING

Hauke Krüger, Thomas Lotter, and Peter Vary

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

ABSTRACT

In this contribution a new software environment for an embedded system is presented that is part of a laboratory for teaching students in Digital Audio Signal Processing. This environment enables even students with only a limited background in programming embedded technology to achieve remarkable implementations of real-time projects including runtime user interaction. Hardware and algorithm related programming issues are separated for audio processing and interactive messaging. While the hardware related part is provided along with the software environment the students can completely focus on algorithmic aspects. A well defined interface builds the bridge between the two programming components.
Examples are presented to demonstrate the flexibility and efficiency of the given software platform. The underlying projects were defined, implemented and finally demonstrated by students in an interactive real-time demonstration.

1. INTRODUCTION

Specifying a laboratory course that on the one hand comprises an interesting challenge and on the other hand considers the large variety of personnel interest and background knowledge of students inheres an educational burden. One way to achieve this goal is to offer a project oriented laboratory: The students are grouped into small teams of two or three persons and each group defines its own educational program, a project. This project will be planned and accomplished by the students themselves with support from supervisors.
In this context we offer a laboratory "Digital Signal Processing". This laboratory is based on projects that groups of students define and finally complete in a given time period of one semester. Prior to the project phase the students are given an introduction into Digital Signal Processing.
The final goal of the project is to demonstrate an algorithm from the field of Digital Signal Processing on an embedded real-time platform containing a real-world Digital Signal Processor (DSP) that is well suited for two-channel audio processing with up to high fidelity quality. In order to have a flexible demonstration a PC can be used for interactive messaging to e.g. switch the demonstrated algorithm on or off or to control parameters at runtime. During the demonstration the signal from e.g. a CD-player or two microphones (Source) is processed by the DSP and the resulting audio signal played back by a loudspeaker (Playback). Figure 1 illustrates the setup for the real-time demonstration.

Creating the software to run the embedded board from scratch requires a lot of programming of functionalities that have administrative character and thus is not relevant for the educational goal of teaching Digital Signal Processing. In order to liberate the students from these programming issues, we have created a software environment that separates the complete real-time program into two components: The first component comprises the implementation of the basic functionality (operating system) for running the embedded system and deals with functional parts that every algorithm will require to function properly. This for example includes the initialization of the hardware components of the embedded board.
The second component contains the algorithm of the student's project. Each algorithm comprises processing and messaging related functionalities. This component is implemented by the students.
In order to create the bridge between the two components a well defined interface is used.
For educational purposes besides learning the basic functionality to run a DSP (machine-intimate programming, assembly language) the students will be introduced to the usage of the provided interface.
In this paper we present the functionality of the software environment that builds the basis for the student's projects. Besides using the environment for teaching purposes the whole system also is a good starting point for creating real-time prototypes for new algorithms based on embedded DSP technology.
In section 2 the basic concept and educational program of the laboratory is described. In section 3 the hardware setup for the real-time demonstration is introduced.The software environment and in special the interfacing functionality the student's projects are based on are explained in section 4. Section 5 gives examples of student-projects during the last semesters. Conclusions are drawn in section 6.

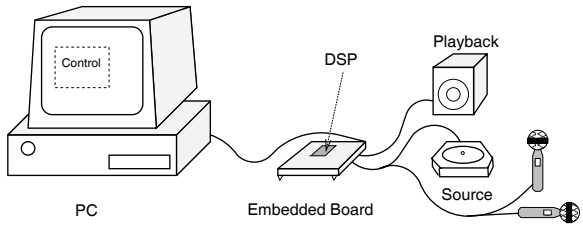

Fig. 1. System for interactive real-time Demonstration.

2. LABORATORY FOR DIGITAL SIGNAL PROCESSING

Our laboratory "Digital Signal Processing" aims at students who have finished the undergraduate courses and have basic knowledge about signal processing. In general they are in the third or fourth year of studies. Groups of two or three persons are combined for each project, the duration of the laboratory is roughly 13 weeks (one semester), while each group works for four hours each week. The laboratory consists of two phases to give an introduction to Digital Signal Processing first and to accomplish the self defined projects afterwards:

2.1. Laboratory Phase 1: The Introduction

In the first phase of the laboratory the students are introduced to the fundamentals of Digital Audio Signal Processing: The high-level programming languages MATLAB and SIMULINK are available to the students, and exercises such as quantization/sampling and Discrete Fourier Transform are solved with these languages as a practical application. Furthermore the students learn the basics in designing and realizing Digital Filters, comprising Finite Impulse Response (FIR) as well as Infinite Impulse Response (IIR) Filter implementations. In order to build the bridge to the real-time implementation the students study real-time programming with focus on embedded technology and Digital Signal Processing. This comprises principles of interrupt driven machine intimate programming, assembly syntax and also efficient implementations of the filter structures from previous exercises. Finally the software platform for the embedded real-time system is introduced. Exercises help the students to understand the fundamentals of the system and the integration of their own components.

2.2. Laboratory Phase 2: The Project

In the second phase of the laboratory the students define the project that they wish to implement in the following weeks. They either provide their own ideas or select from a list of predefined projects. After the selection they start to develop and investigate the chosen algorithm with MATLAB or SIMULINK first. Afterwards the algorithm is implemented on the embedded real-time system in order to achieve the interactive real-time program for the demonstration of the chosen algorithm. This real-time demonstration will be accompanied by a short presentation in front of all participating students at the end of the laboratory course.
The programming is completely based on assembly language due to the achievable performance.

2.3. Learning Goals

Besides learning the basics in Digital Audio Signal Processing the students also learn soft skills due to the project oriented character of the laboratory. Starting with the beginning of laboratory phase 2 the projects will have to be planned and worked on by the whole group and finally a presentation will have to be held in front of all participating students. Thus the students learn..

- to plan their project.
- to understand and apply Digital Signal Processing.
- to work as a team.
- to integrate their software in a given software project.
- to present the achieved results in front of a group.

3. SYSTEM FOR REAL-TIME DEMONSTRATION

The system for the real-time demonstration comprises an embedded board including a DSP for Signal Processing and a PC for software development and runtime control of the embedded platform.

3.1. Embedded platform

The platform used is called ADSP-21065L EZ-LAB from Analog Devices[1]. The appendant ADSP-21065L SHARC processor supports floating and fixed point arithmetic and has a 60 MHz clock. Thus, considering the maximum of 3 operations available during each clock cycle, the processor reaches a computational power of 180 Million Floating Point Operations Per Second (MFLOPS). The processor can be controlled using a JTAG connection [2]. For Analog-To-Digital and Digital-To-Analog Conversion (ADC, DAC) a hardware component can be programmed to support sampling rates in between 7 and 48 kHz, adjustable in steps of 1 kHz, with a quantization accuracy corresponding to 16 Bit for each sample (AD1819A Soundport Codec from Analog Devices, [3]). For the runtime messaging a hardware component from National Semiconductor can be setup to establish a full duplex RS232-connection (Universal Asynchronous Receiver/Transmitter, UART 16550D [4]). Additionally to 544 kBit on-chip-memory the EZ-LAB provides 1 MWord external Random Access Memory (RAM) with each word being 32 bits wide.

3.2. Controlling PC

The PC is used for two purposes:

- Development of the program that runs the embedded target
- Control the real-time processing during runtime

For the development of the program to run the embedded target, Analog Devices provides an Integrated Development Environment (IDE) called VisualDSP++. This software comprises an editor to type the source code, compiler and linker to generate object-code for the program and mechanisms to load the object-code of a program to the DSP and to emulate the downloaded software. The object-code is loaded to the DSP via a JTAG emulation card that is plugged into the PC (Mountain-ICE [2]) on one side and connected to the DSP on the other side. After downloading the program it can also be debugged and run instruction after instruction using this link.
As a second link from the PC to the DSP an RS232 link is connected from the PC to the embedded board. This link is used for the runtime messaging. On the PC side an example application lets the user send and receive DSP messages. This application is based on a simple object oriented software development kit and can be modified by the students for their project specific functionality. Figure 2 illustrates the hardware setup for real-time processing.

4. SOFTWARE ENVIRONMENT FOR STUDENTS

The software environment provided for students to accomplish the project contains all the programming that is not related to the algorithm. The additional algorithmic related programming components are project specific and can easily be integrated into the environment by the students. The interface to integrate the student's

[1] Analog Devices has provided the hardware for the laboratory in the context of their University Donation Program [1]

[2] ICE: In Circuit Emulation. [5]

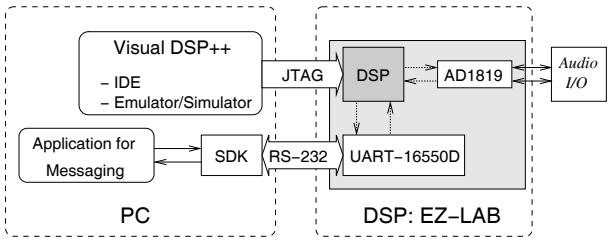

Fig. 2. Setup for real-time processing.

software corresponds to the functionality of the software environment and can be grouped into two main topics:

- Digital Signal Processing: The main algorithmic functionality is the Signal Processing routine that the students implement.
- Messaging: Messages for runtime user interaction are exchanged in between DSP and PC. The students have the possibility to send messages on one side, receive them on the other side, and incorporate the information obtained into the main Signal Processing routine.

The two components will be described separately in the following:

4.1. Integration of Algorithm Components

Audio processing works based on a double buffering architecture and is part of the real-time kernel provided with the software environment: Input and output buffers for each channel exist twice. While input samples for each channel are collected in one input frame and played back from one output frame on an interrupt driven time basis, the second buffer is used for processing in parallel. Once the complete input buffer has been filled the buffers are switched. In the trivial case of a talkthrough implementation the samples in the second buffer for input are simply copied into the second buffer for output. After the next buffer switch the values are transmitted to the DAC and played back.
In order to integrate their own algorithm, the students provide a function *projectStep* to process the second input buffer filling the second output buffer for each channel as depicted in Figure 3.
This function is called whenever a new frame has completely

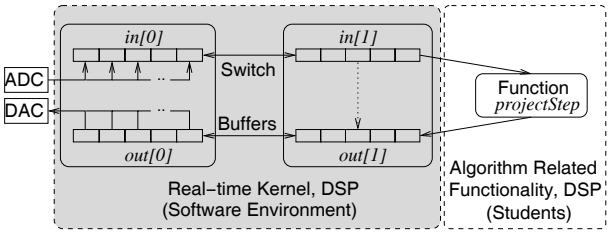

Fig. 3. Integration of the *projectStep* function into the software environment for signal processing.

been collected in one input buffer and the buffers have just been switched. At the same time a reference to the field that is now ready for being processed and a reference to the field that is expected to be filled for output by the processing routine are passed as functional arguments. A function prototype is provided as a starting point along with the software environment that implements a talkthrough functionality. Besides the *projectStep* function that will be implemented by the students, there are also initialization and termination functions that will be called before and after audio processing in case the algorithm requires initialization/termination.

4.2. Integration of Messaging Components

The messaging functionality is based on an RS232 connection. Figure 4 demonstrates the principles for runtime messaging on the DSP side. According to the open characteristic of the applied algo-

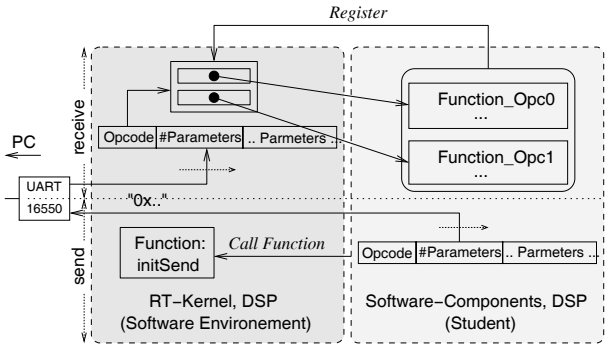

Fig. 4. Principles for sending and receiving runtime messages on the DSP side.

rithms implemented by the students, the format of the exchanged messages is also open: In general each message consists of three parts: The Opcode, the Amount of Parameters and the Parameters The opcode is a specifier of 32 bit length, the amount of parameters is also a 32 bit value while the parameters are sequential 32 bit values, the amount according to the value transmitted as second argument. The meaning of the opcode and the associated parameters can be freely adapted by the students
For the transmission over the RS232 link the message has to be divided into bytes on the sender side and composed of received bytes on the receiver side in the kernel of the software environment. The connection is full duplex and always between PC and DSP. Thus the PC and the DSP can both be sender and receiver. To send and receive messages the students have to undertake the following steps:

4.2.1. DSP: Sending

In order to send a message from the DSP to the PC, a field containing the complete message must be prepared and filled. A function is called that is part of the software environment implementation to initiate the transfer to the PC. The field is passed as reference and will be read from the kernel of the software environment automatically. A message that has been completely sent is acknowledged.

4.2.2. DSP: Receiving

For the reception of a message on the DSP side the student specifies an opcode specific function for each opcode used. This function is registered with the software environment at the beginning of the processing and internally assigned to the specified opcode. Whenever a message has been completely received and decoded by the kernel of the software environment, the registered function is called to indicate that a new message is available. The message itself is passed as functional argument so that the parameters can be read.

4.2.3. PC: Sending and Receiving

On the PC side sending and receiving are implemented in the high level programming language C++. A small application is provided that demonstrates the usage in the context of a graphical user interface. In order to send a message a function is implemented that expects opcode and parameters that will be sent to the DSP.
In order to receive a message a function must be provided by the student. The underlying software environment then calls the function whenever a message has been completely received, passing the opcode and parameters of the message as functional parameters. The students combine the messaging functionality with a graphical user interface to incorporate user interaction into a project specific application.

5. EXAMPLES FROM PREVIOUS SEMESTERS

In this section examples will be given from the field of Digital Audio Signal Processing developed by students during the last semesters:

5.1. LPC Vocoder

In this project a group of three students developed a vocoder based on linear prediction coefficients (LPC) as depicted in 5. Linear prediction coefficients are a representation of the characteristics of the vocal tract for human speech generation (e.g. [6]). In the project a tenth-order LP-filter is estimated in an autocorrelation based approach for short term audio segments of an input signal $x(k)$ applying the levinson durbin algorithm (LPC-Analysis). The input segment is afterwards filtered by the LP-Analysis filter to determine the signals residual signal $e(k)$. From this signal the power is estimated in a final step. In order to synthesize the analysed speech, the residual signal is approximated by either sparse pulses or noise that has the same power as the original residual signal. Signal $\tilde{e}(k)$. $\tilde{e}(k)$ is then filtered with the LP-Synthesis Filter, the inverse to the analysis filter to create the artificial speech signal $\tilde{x}(k)$.
During a real-time demonstration the LP-order and the excitation source can be selected using the runtime messaging environment.

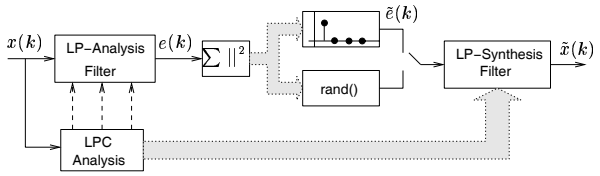

Fig. 5. Vocoder algorithm in students project.

5.2. Noise Reduction

During the last semesters various student groups realized a real-time noise reduction system on the DSP platform. One of these projects was a superdirective beamformer, which is depicted in Figure 6 as block diagram. The filtering is performed in the frequency domain as complex multiplication of DFT coefficients with filter weights $W(\theta, f)$ depending on a desired spatial angle θ and DFT frequency f. The coefficients are generated using a superdirective design criterion [7] and stored in the DSP on-chip-memory for a given resolution. During runtime the PC communication interface is used to change the set of coefficients and thus the spatial directivity of the two-microphone array. In order to minimize

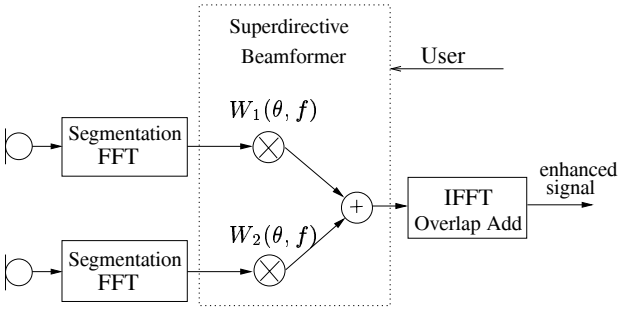

Fig. 6. Superdirective Beamformer realized on DSP platform

distortions caused by cyclic convolution of the frequency domain filter, the students included an overlap add synthesis.

6. CONCLUSION

In this paper we presented a system that is used for a project oriented laboratory for students called "Digital Signal Processing". In that laboratory the students are first introduced to fundamentals of signal processing and second, a self defined project is planned, realized and finally presented as a real-time demonstration running on an embedded board containing a Digital Signal Processor.
In order to free the students from hardware related programming issues a software environment enables the students to fully focus on algorithmic programming due to the separation of hardware and algorithmic components of the real-time program. Example projects from the past semesters demonstrate the efficiency of the system to enable the students to implement even complex algorithms including runtime user interaction in a limited timeframe.

7. REFERENCES

[1] Analog Devices Inc., "University Donation Program," http://forms.analog.com/Form_Pages/dsp/universityDonation.asp, 2003.

[2] IEEE, "1149.1-2001 IEEE Standard Test Access Port and Boundary-Scan Architecture 2001,," Tech. Rep., IEEE, 2001.

[3] ADI DSP Applications John Tomarakos, "Interfacing the ADSP-21065L SHARC DSP to the AD1819A 'AC-97' Soundport Codec, v2.4a," 1999.

[4] National Semiconductor, "PC16550D Universal Asynchronous Receiver/Transmitter with FIFOs," 1995.

[5] White Mountain DSP, "Mountain-ICE Emulator Hardware Users Guide," 2000.

[6] N.S. Jayant and P. Noll, *Digital Coding of Waveforms*, Prentice Hall, 1984.

[7] 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.