OMAP 3 BASED SIGNAL PROCESSING FOR BIOMEDICAL ENGINEERING TEACHING

Matthias Klostermann, Olaf Christ, Kunal Mankodiya, Simon Vogt and Ulrich G. Hofmann

Institute for Signal Processing, University of Lübeck, Lübeck, Germany
Ratzeburger Allee 160, 23538 Lübeck, Germany
phone: + (49) 451.500.5803, fax: + (49) 451.500.5802, email: hofmann@isip.uni-luebeck.de
web: www.isip.uni-luebeck.de

ABSTRACT

When it comes to academic signal processing, students' experiences are sometimes spoilt by the dry nature of the topic, applying only mathematical methods. We propose in the following a more hands-on method to acquire signal processing experiences by utilizing state-of-the-art Digital Signal Processor hardware for biomedical engineering related tasks. In order to circumvent hindering setup problems we provide students with a complete and royalty free development environment based on Virtual Machines to program an OMAP3530 dual core processor. That way students can jump start implementing class relevant signal processing algorithms and gain a physical impression of realtime signal processing.

1. INTRODUCTION

When facing current needs of academic engineering teaching, the chinese proverb „I hear and I forget. I read and I remember. I do and I understand" is probably the shortest, but nevertheless best appraisal of its state. Teaching of Digital Signal Processing thus should be an application of the above statement attributed to *Lao Tse*.

In order to comply with the proverb's implications, students do not only attend a more or less complex lesson on DSP („I hear"), but have to study the underlying textbooks as well („I read"). However, when it comes to satisfy the „I do" criterion in teaching, the activity often is limited to exercising mathematical calculus to solve nice academic problems (see e.g. [1, 2]. This may or may not lead to an understanding of DSP worth recalling for the individual student, but we doubt that it is the best way of „doing" DSP. In particular, it seems not to meet the needs of more application oriented fields like biomedical engineering.

We consequently propose a less calculus oriented „learning by doing" approach to Digital Signal Processing by exposing students to a real world *digital signal processor* of the newest build. Thus, they are not only learning the concepts of Digital Signal Processing on a problem oriented bases, but also applying them in a cutting edge, portable and low-power implementation with biomedical engineering motivations.

Above mentioned teaching ideas are not easily met in reality, not even with current commercial DSP systems, although their evaluation boards come with a dedicated set of tools and examples [3]. Instead, even this support leads to an individual learning curve too steep for a student to be mastered during term-long problem solving. We therefore looked for a better supported and proven DSP system and found it in the Open Source community.

The following report will describe our efforts to integrate this state-of-the-art dual core microcomputer chip into our teaching for biomedical engineering classes at the masters level.

2. MATERIALS AND METHODS

The chip we recently started to use in a number of student development projects is the OMAP3530 from Texas Instruments (TI), which contains a fast ARM core, a DSP core, and a graphics accelerator all on one die. This new generation of OMAP processors is officially distributed by TI, but has a large and growing online community supporting it. As this generation of processors is still fairly new, it is probable that these chips can be used for another 3-5 years without losing software support.

The OMAP3530 processor can be ordered as part of an official full-scale evaluation module (OMAP3530EVM, Texas Instruments) that provides most peripherals such as a numeric keypad and a TFT display for completeness.

For our purposes more suitable is a different interesting approach to development boards and follows the idea of keeping the board as simple as possible and supplying only the most needed peripheral hardware mounted on the board itself (Fig.1). Such a system has been developed for the OMAP3530 by a open source community under the name *BeagleBoard* (BeagleBoard.org, USA).

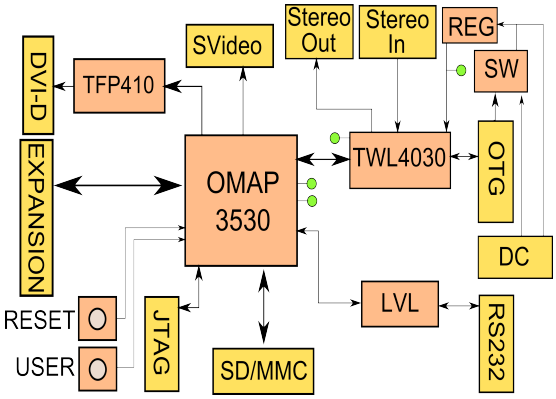

Fig.1: Diagram of the BeagleBoard with the OMAP3530 processor in the center and many useful peripherals on board [4].

2.1. Overview over the BeagleBoard

Unlike commercial development boards, the BeagleBoard has an open source and freely supported operating system coming with a growing repository of working applications. The large and advantageous development community of BeagleBoard developers can often solve a problem faster than the technical support of TI can do, although most community efforts are currently still aimed towards the Embedded Linux ARM part of the system.

On top of the low price and large and fast support, the BeagleBoard has another advantage: Its physical dimensions are just around 10x10 cm, making it more suitable for portable and student applications at a university.

The USB-powered BeagleBoard is a low-cost, fan-less single board computer based on a TI OMAP3530 dual core processor that is said to reach laptop-like performance and integrates a 600MHz ARM Cortex-A8 core with a high-end 430MHz DSP-TMS320C64x core [4].

Additional hardware can easily be connected via USB as mentioned above [4].

2.2. BeagleBoard tool chain

With the BeagleBoard every developer faces the opportunity as well as the challenge to write applications for both the ARM Cortex A8 Core and the TSMC320C64x+ DSP Core of its OMAP35x platform. But before the developer gets a chance to probe the hardware for its potential, he needs a suitable development environment to facilitate his endeavor.

The ARM Cortex A8 Core fulfills the role of a general purpose processor running various Embedded Linux Distributions like *Angström, Debian ARM, Ubuntu ARM* or even Google's *Android*. With every major *Linux Distribution* comes a repository of open source software, allowing the user to work "out of the box" with familiar applications.

In the context of signal processing the available Linux tools and APIs may be used for data acquisition and visualization by simply writing an application on a PC and subsequently compiling it for the ARM architecture.

However, for a student eager to learn the ways of digital signal processing the focus of this environment clearly lies on the DSP tool chain and less on the ARM side of the chip. One of these DSP tool chains, including a complete IDE, is TI's *Code Composer Studio* (CCS): Specialized on TI's line of DSP hardware, it provides a complete suite for code editing, generation and debugging (e.g. *JTAG*), needed to get started immediately. For industrial use it comes with a hefty price-tag, but an academic and teaching license is available.

For outside the classroom use, TI's free *C6x Code Generation* tool chain accompanied by the *DSP/BIOS 5.X.X* sources poses a fairly basic alternative to the full fledged CCS. These tools form the back-end of CCS anyways, however it is only possible to generate plain binaries for the DSP with them. A non-proprietary way to compensate the lack of an IDE, a debugging facility and a transfer of generated DSP-binaries to the DSP-core is needed.

An alternative to the IDE part of *Code Composer Studio* may be found in several different open source IDEs like *Code::Blocks* [5] or *Eclipse* [6], though the level of integration varies from a mere colored text-editor up to plugins for the transfer of binaries to the board. If there is no such plugin available, the transfer may be done with a USB stick, USB harddrive or an SD card. We chose the open source supported *Code::Blocks* as our IDE [5].

For data visualization and especially debug output we utilize the ARM Cortex itself: *CodeSourcery,* developer of a commercial GNU compiler collection [7] offers with *Sourcery*

G++Lite a free open source C/C++ tool chain for ARM processors, making it relatively easy to write applications for the Linux distribution running on the OMAP35x. The missing link between the ARM- and the DSP-core is supplied by TI's *DSP/BIOS-link API* in a master-slave constellation. Thus the *DSP/BIOS-link API* is controlling the DSP from the ARM side, starting it, stopping it and feeding it with the DSP binaries.

2.3. **Student‘s developing environment**

Provided with a completely free set of tools it is now possible to send the students off to install everything and then have them start experimenting, which may be done for advanced computer science students. But installing all the parts of the tool chain is time consuming and small mistakes during the setup lead to long searches for the cause, unnecessarily consuming even more time and hindering the learning curve.
We shorten this delay by using a Virtual Appliance approach [8, 9]. In the beginning of the DSP class, we create a platform independent package containing a Linux distribution and a finished setup of the tool chain. The virtual disk image thus contains:

- pre-installed and pre-configured *Ubuntu* Linux 8.10
- *Code::Blocks* IDE [5] 8.02
- the *Sourcery G++ Lite [7]* ARM development tool chain
- the DSP development tool chain:
    - *TI C6x Code Generation Tools*
    - *DSP/BIOS 5.33.03*
    - *DSP/BIOS-Link 1.61*

It can be downloaded at a student accessible site or is handed out as DVD-ROM. The student using this package is then able to start with development on the OMAP35x, just shortly after installing the Virtual Machine [9] on his PC.

Fig. 2: Physical setup of a BeagleBoard programming environment: The screen in the background shows the DVI output of the boxed BB in the center (arrow), here live audio stream and FFT results.

In fact, the necessary effort for the student is reduced to the following, simple steps: Install the Virtualization Software on the PC and add the Virtual Machine to it [9]. Connect the Beagle Board (BB) and the PC with a RS232 serial connection, and a monitor at the DVI output (see Fig.2). Plug a preconfigured SD card containing e.g. A*ngström Linux* into the BB. Start a terminal client on the PC connecting to the serial port. Connect the power supply (either a mini-USB-B to USB-A cable or a regular 5V power supply) to the BB and watch it boot on the SSH-like terminal window.

Actual programming and code prototyping may now start in the Virtual Machine with the PC as the built target. This allows fast and easy debugging of algorithms. The resulting sourcecode can then be ported to the ARM Architecture by a simple re-compile with the ARM tool chain. With the now obtained proof that the algorithms work, there is enough free time to learn the intricacies of DSP/BIOS-link [10].

3. DSP TEACHING EXAMPLES

Out of the large number of possible signal processing algorithms taught, we selected a small set corresponding to our lecture and to avoid negating the effect of "learning by doing". A given task needs to be done in a restricted amount of time, limiting the choices by the complexity of the theory behind it and the necessary effort for the implementation. If it is not possible to finish the task in time, the intended goal to further the understanding of digital signal processing cannot be reached.
As a consequence we have a small, but growing catalogue of example tasks, requiring not more than a BB and some adequate audio source (microphone or electronic stethoscope):

- live display of acquired audio signal (ARM core only)
- live inversion and thresholding (e.g. ECG monitor)
- live convolution with a predefined kernel (simple low pass filtering, e.g. speech analysis)
- live filtering (50 Hz notch filter, band pass, e.g. remove power line noise)
- live spectral analysis by discrete Fourier transform (e.g. auscultation)
- live auto- or cross correlation of stereo signals (e.g. intra-aural time difference)

More advanced and therefore under construction in longer student projects are:

- live Hilbert transform and phase shifting (ambient noise canceling)
- live empirical mode decomposition [11]

- live wavelet transform by the à-trous algorithm [12, 13] and by lifting [14]
- blind source separation of two audio channels [15] [16]
- heart rate variability (needs a custom made physiological amplifier board [3])

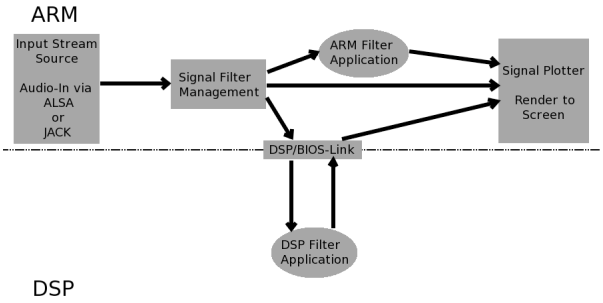

Fig. 3: The diagram shows the signal path through the OMAP following the reference framework 6 (RF6) processes. Incoming audio data from the ARM is sent through the DSP/BIOS link to the DSP, where processing is performed. Process results are then sent back to the ARM for display and audio output [10].

Any audio task setup is created using the BB's audio input capabilities and a suitable microphone as a signal source and a TFT display connected to the DVI output as a signal sink, plotting the source and filtered signal (see Fig. 3). With this approach it is possible to evaluate different algorithms in a real-life signal processing scenario with a clearly visible link between cause and effect. The design of the application is kept simple and, to a certain degree, flexible. Data from the audio-device may be obtained block-wise via ALSA (Advanced Linux Sound Architecture) [12] or JACK [13] with the possibility to add further APIs, opening the application to a wide variety of streaming data sources. Blocks of signal data are then filtered directly on the ARM core or better handed over to the DSP for this task.

Filtering is performed by implementing a simple discrete convolution with a choice of predefined or in MATLAB designed FIR filter kernels. Depending on the level of studies this application may be further advanced by requesting IIR filters to be designed and characterized e.g. in MATLAB first.

After filtering the signal by applying the signal processing algorithms in question, it is handed back to the ARM to a plotting algorithm which renders the signal visible on the screen. Optional audio output is performed via speakers or headphones connected to the line-out jack on the BB.

To illustrate the teaching procedure, we present in the following example the teaching task of live audio filtering on the DSP.

As stated above, the student has access to a working BeagleBoard development environment (see Fig. 2) and an instruction manual. The instructions will guide the student through the few steps to bring up the Virtual Machine and the BeagleBoard online and enables him to compile an example DSP/BIOS link application created specifically for the following task.

He is then familiarising himself with the capabilities of the example application by learning from the audio data acquisition and plotting routines. With this example, he is executing his first signal processing task, a simple sign inversion of the signal values (see Fig. 4, second top trace).

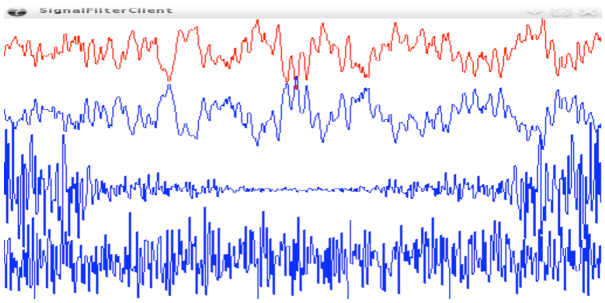

Fig. 4: Output screenshot of 1024 data points with (from top) live raw signal, inverted signal, real part of FFTed signal and an auto-convolved signal.

The next independent step is to implement a „brute force" convolution, convolving the signal with a given small kernel (e.g. a 16 sample rectangular signal). At this point the student may experiment with the effect of different convolution-kernels on the audio signal.

To demonstrate performance limitations of the DSP hardware, the next task is to increase the length of the convolution-kernel until the hardware can not keep up with the amount of incoming audio data anymore, displaying erroneous curves. As a follow-up the student is now requested to perform the convolution in the Fourier domain. Thus he has to utilize a discrete Fourier transform (see Fig. 4, second from bottom) and its inverse such that again the convolution of the signal with the kernel is obtained.

Finally, the student runs the DFT based convolution of the audio-signal with a kernel of increasing size, finding the hardware limitations with this algorithm. The newly obtained result may then be compared to the result of the previous convolution algorithm.

4. CONCLUSIONS AND PRACTICAL USE

At the end of the lab class, the student has got first hand experience in programming DSP hardware, the effect of a specific signal processing algorithm on a live audio signal and the practical impact of different algorithmic strategies of digital signal processing.

At the time of this writing, several teams of master students in biomedical engineering at our institute are actively testing above explained jump start to signal processing on the OMAP3 based BeagleBoard. Although their experience is not completely analysed, it is clear, that our approach with Virtual Machines does speed up the learning curve on a digital signal processor greatly.

Of course, one could argue, that most of the above stated teaching examples are introduced in MATLAB or Python as well. However to have instructional algorithms run in real-time on a dedicated and industrially used DSP hardware almost gives a physical impression to the student, which seems more advantageous than plain programming. At the same time students experience clearly the limitations of the technology and what to expect from these processors in real world applications. But even more, the use of the BeagleBoard produces a strong motivation to reflect on the otherwise dry topics of academic signal processing and simply are fun to use.

We look forward to expose our next term signal processing students to this type of „learning by doing“.

5. ACKNOWLEDGEMENTS

We want to acknowledge valuable input and help from the BeagleBoard community and Robert Owen from Texas Instruments, Europe. This work is in part funded by the German Research Ministry‘s grant „BiCIRTS“ 13N9190.

6. REFERENCES

1. Proakis, J.G. and D.G. Manolakis, *Digital Signal Processing*. 4th ed. ed. 2007, Upper Saddle River, NJ: Pearson Education.
2. Lüke, H., *Signalübertragung*. 6. ed. 1995, Berlin: Springer. 381.
3. Mankodiya, K., et al. Portable electrophysiologic monitoring based on the OMAP-family processors from a beginners prospective. in submitted to DSP 2009. 2009. Santorini.
4. http://www.beagleboard.org, BeagleBoards Software Reference Manual, Rev. B.5. 2008.
5. http://www.codeblocks.org/. Code::Blocks -The open source, cross platform, free C++ IDE. . 2009 [cited 2009 2.3.2009].
6. http://www.eclipse.org/. Eclipse website: Open source IDE. 2009 [cited 2009 2.3.2009].
7. http://www.codesourcery.com. *Sourcery G++ light*. 2009 [cited 2009 2.3.2009].
8. Camargos, F. and G. Girard. Virtualization of Linux servers. in Proceedings of the Linux Symposium 2008. 2008. Ottawa, Ontario, Canada.
9. http://www.virtualbox.org. VirtualBox - a general-purpose full virtualizer 2008 [cited 2009 2.3.2009].
10. Mullanix, T., et al., Reference Frameworks for eXpressDSP Software: RF6, A DSP/BIOS Link-Based GPP-DSP System, T.I. Inc., Editor. 2004.
11. Huang, N.E., et al., The empirical mode decomposition and the Hilbert Spectrum for nonlinear and nonstationary time series analysis. Proceedings of the Royal Society London, 1998. **Ser. A**(454): p. 903–95.
12. Dutillieux, P. An implementation of the algorithm `a trous to compute the wavelet transform. in Wavelets: Time-Frequency Methods and Phase Space. 1989. Marseille 1987: Springer.
13. Holschneider, M., et al. A real- time algorithm for signal analysis with the help of the wavelet transform. in Wavelets: Time-Frequency Methods and Phase Space. 1989. Marseille 1987: Springer.
14. Sweldens, W. The lifting scheme: A new philosophy in biorthogonal wavelet constructions. in Wavelet Applications in Signal and Image Processing III. 1995: SPIE.
15. Mei, T., et al., Blind source separation based on time-domain optimizations of a frequency-domain independence criterion. IEEE Trans. Audio, Speech, and Language Processing, 2006. **14**(6): p. 2075-2085.
16. Mazur, R. and A. Mertins. Reducing reverberation effects in convolutive blind source separation. in European Signal Processing Conference. 2006. Florence.