Signal Processing In Embedded Systems

K. Dewald, Graduate Student Member, IEEE and D. Jacoby, Member, IEEE

Abstract— The implementation of algorithms by means of signal processors usually lacks a friendly user interface, mainly because of the difficulty involved in their creation and maintenance. One practical solution to this problem, both in education and industry, is the integration of signal processors within embedded systems. This configuration enables a designer to easily implement an interface that does not only allow the visualization of the process, but also admits the possibility of changing parameters in real time.

Keywords— beagleboard, digital signal processing, education, embedded systems.

I. INTRODUCTION

The appearance of cheap embedded devices has significantly extended their potential for industry and education. In particular, they facilitate the teaching of relatively complex topics, such as digital signal processing, by providing an easy-to-program interface to the relevant device, thus allowing the student to spend more time on the educational aspects of the task at hand, reducing the amount of unproductive administrative tasks required by the hardware. Additionally, as embedded devices have become increasingly widespread due to their practicality and great flexibility to deal with changing requirements, there has been a growing need for educational institutions to train people who are proficient in the development and implementation of such systems. In the following pages, an approach to such an educational scheme is presented.

II. HARDWARE AND SOFTWARE PLATFORM

In educational environments, the usage of open source hardware and software allows students to interact with great amounts of people involved in many important aspects of the development and implementation of a certain device in different fields, providing an overview of all the different applications where embedded systems can be used. In the scheme presented, the hardware platform used was a Beagleboard Revision C4 [1]-[2], a single-board computer designed by Texas Instruments. The most important aspect to emphasize is its main processor, an OMAP 3530, which contains an ARM Cortex-A8 running at 700 MHz, 256Mb of RAM and the Texas Instruments TMS320C64x+ Digital Signal Processor (DSP), which can run up to 430 MHz. The operating system chosen was Ubuntu Linux, using the XFCE desktop environment, although any other similar alternative works equally well for the educational purposes exposed.

III. EMBEDDED DSP SYSTEMS

A very common requirement during the development of digital signal processing algorithms on a hardware platform is the changing of working parameters in real time. This task is usually done by a computer connected to the DSP hardware, which provides a visual user interface to make these modifications, providing instant feedback to the student. Otherwise, the only alternative consists in making the changes directly in the source code, losing the real time feedback.

There are very well known tools available such as SYS/BIOS [3] -a real time operating system created by Texas Instruments- which tackle these deficiencies by providing a powerful interface that maximizes the capabilities of the available hardware, also facilitating the debugging and optimization process to the user. On the other hand, as most of these advanced systems usually run directly on the DSP, the available resources are diminished. Also, if additional devices are connected to the DSP, it becomes impractical to write the required drivers for that platform in comparison to an equivalent embedded system running Linux, as the user might not be acquainted with all the system internals of that platform. The embedded DSP system presented in this paper offers an alternative approach to DSP programming, preserving the same functionality as others, but simplifying the integration process with more complex systems.

This system consists of two main parts: The first one is a Linux operating system running on the main core of the embedded device, and the second one is a wrapper library running on the DSP. The advantage of having a known operating system as the main interface towards the student is the soft learning curve to gain full control of the device, as well as the availability of drivers and documentation to assist the process. At the same time, the wrapper library provides a simple API to access all the capabilities of the DSP from within a C-written program by handling all internal procedures on its own and only leaving to the user the addition of his code to the project. The final wrapper library presented was developed on top of one created within a competition organized by Texas Instruments [4]. This new
version maintained all the functionality of the original
wrapper library but had a much simpler internal organization
which made it easier for it to be modified.

This dual system allows, for example, a more visual and
clearer debugging process in comparison to the most
commonly used DSPs in educational environments, as the
behavior of the program can be monitored from a log file, a
screen connected to the embedded device or even remotely.
However, the main advantage comes from the creation of a
completely reconfigurable DSP environment running within
the same piece of hardware, in which both processors work
closely together doing the work each one of them is best
suited to do. One of the possibilities this reconfiguration
process provides is the in-situ modification in the working
parameters of a DSP function. This can be used, for example,
to alter the coefficients of a finite impulse response filter by
changing the filter response without having to stop the
execution of the program. Another possibility that this system
allows is the modification of the code being executed on the
DSP during runtime. Since the DSP is behaving under the
presented paradigm as a slave, the main processor can upload
new code to the DSP dynamically if a different function set
needs to be executed, which leads to an interactive and
educational programming model.

IV. EMBEDDED DSP PROGRAMMING MODELS

All programming models derived from the DSP system
presented before work on the same principle, in which the
main task running on the main processor sends data to be
processed by the DSP and retrieves the result afterwards.
Based on that idea, four models arise, each one more complex
than the previous one, which facilitate the educational
purpose of using DSPs on embedded systems. These models
also allow the main program to decide what to do next based
on the result obtained from the DSP, enabling the user to
program in higher level languages, such as C, Bash or Python.
In addition to that, it also facilitates the integration of the DSP
to a bigger and more complex system.

A. Master/Slave Single-Threaded Behavior Model

The first and most simple of the programming models
presented is the Master/Slave Single-Threaded Behavior
Model, in which the main processor holds complete control of
the DSP, sending data to be processed and waiting for it to
return before continuing with the execution of the main
program, as diagrammed in Fig. 1.

The main advantage of this model is its simplicity, which
serves as a first step for students to understand the process of
programming an embedded DSP. It facilitates the learning of
the DSP assembly code and the most used algorithms, while
also allowing further understanding of the internal functioning
of the hardware, such as dynamic memory allocation or the
pointer exchange between processors.

B. Master/Slave Multi-Threaded Behavior Model

The Master/Slave Multi-Threaded Behavior Model is an
extended version of the Single-Threaded Behavior Model, in
which the main program keeps executing other tasks while the
DSP is performing the requested processing, as diagrammed in
Fig. 2.

The main difference with the previous model is the
increased performance, as more processing time is available
by using the previously unused idle time of the main
processor. The educational value comes from teaching the
student how to implement an embedded DSP system on
projects which require a higher processing power or parallel
computing.

C. Task Queue Model

The Task Queue Model takes advantage of a feature
implemented by the original wrapper library, which allowed
multiple tasks to be sent to the DSP without the need of
waiting for the first one to finish before the second one is
sent. The main idea behind this model consists of an idle loop
running on the DSP which puts the incoming tasks in a queue
and sends the results back only when a certain amount of them have been processed, as diagrammed in Fig. 3.

![Program Flow for Task Queue Model](image)

Fig. 3. Program Flow for Task Queue Model.

The main benefit of this model compared to the previous ones is a higher performance when there is a big difference in task execution time or when the tasks are finished in a relatively short time. Data is exchanged between processors only when a certain amount of tasks have been processed. The overhead generated by quickly-processed tasks is shared among many more of them, thus increasing the overall performance.

D. Adaptive Behavior Model

The Adaptive Behavior Model takes advantage of the fact that the DSP can be entirely controlled by the main processor, and therefore, the code running in it can be modified during runtime, making it much more flexible to respond to the needs of the system it is interacting with.

The main advantage of this added feature is the increased flexibility of any given application, as different algorithms can be implemented in the DSP according to what is required by external factors, creating a fully reconfigurable DSP environment running within the main program, using the benefits of all the previously defined programming models. It should be noted that the process of modifying the DSP code is slow, which may lead to important performance drawbacks. However, under some circumstances the increased functionality might compensate the additional overhead.

V. IMPLEMENTATION EXAMPLE

In order to show the capabilities of this system, a bandpass FIR filter is implemented using the Master/Slave Multi-Threaded Behavior Model. To test this filter, the audio input is provided by the operating system’s audio driver (ALSA).

This implementation of the filter allows the coefficients sent along with the input data to be modified during runtime. The additional overhead does not affect the overall performance of the program as that information is loaded while the DSP is applying the filter to the previous batch of input data. Despite the latter issue, the simplicity of the code allows a student to easily introduce himself into digital signal processing by being able to study and modify such examples.

Even though DSP code is usually written in assembly language in order to improve performance, when the latter is not critical, the C programming language is another option, thus making the algorithms clearer to be understood. In the example below, the DSP function which applies the FIR filter is presented in Fig. 4:

```c
void apply_FIR(struct dsp_global_t* global, 
    void* in, 
    void* out, 
    uint32_t coeFirstListSize, 
    uint32_t FrameListSize) 
{
    int16_t* output = (int16_t*)out; 
    int16_t* coeFirstList = (int16_t*)in; 
    int16_t* FrameList = &coefficient[coeFirstListSize]; 
    int i;
    for( i = 0; i < FrameListSize; i++) 
    {
        int32_t sum = 0; 
        for( j = 0; j < coeFirstListSize; j++) 
        { 
            sum += (coeFirstList[j] * (FrameList[i+j]))<<1; 
        }
        output[i] = sum >> 16; 
    }
    return;
}
```

Fig. 4. Sample FIR Filter Function.

Data batches to be processed by the DSP are sent in so-called messages, containing all the required information for the DSP to do the requested processing. The used wrapper library functions allow some additional parameters to be included, as in this case the amount of coefficients and frames being sent to the DSP.

VI. CONCLUSION

Considering all the above, an embedded DSP system proves to be an excellent tool for industry and education, highly accelerating the development process in both sectors. The integration of signal processors within embedded systems has made their range of applications grow significantly, while at the same time, development costs were reduced. By having each of the processors of an embedded system working on the task they are best suited for, efficiency and flexibility are maximized.

ACKNOWLEDGMENT

K. Dewald thanks D. Jacoby for his support on this project.

REFERENCES

Kevin Dewald is an Electrical Engineering student at the Buenos Aires Institute of Technology (ITBA), where he received a Merit Scholarship. Currently, he works in the Digital Electronics and Signal Processing Laboratory.

Daniel Jacoby is a Professor of Electrical Engineering at the Buenos Aires Institute of Technology (ITBA). Currently, he works as the Director of the Digital Electronics and Signal Processing Laboratory.