Audio effects acceleration using the TMS320C67xx DSP

Philippe Bénabès (Advising professor)  
Sandrine Burriel  
Arnaud Cazaledes  
Benoît Derat  
Jean-Baptiste Mazon  
Christian Tremblay

ABSTRACT

This document was an entry in the Texas Instruments DSP Challenge 2000, an annual contest organized by TI to encourage students from around the world to find innovative ways to use DSPs. For more information on the TI DSP Challenge 2000, see TI’s World Wide Web site at www.ti.com/sc/dsp_challenge.

Even though the average computing power has known a sharp increase in the past few years, PC-based sound processing is still limited: for an application such as Steinberg’s Cubase, using too many effects overwhelms the processor. Therefore, our aim was to relieve the PC by sending the signal to be processed to the TI DSP. Within the framework of the TI DSP Challenge, we used a TMS320C67xx DSP on its EVM.

This document presents our work on the communication between the PC and the DSP, and more specifically the use of Steinberg VST, to embed DSP-based effects transparently in a commercial PC application. Thus we also present some of the sound effects that we have implemented in two parts: the DSP-based sound processing, and the Cubase plug-in.
Description of the project

General overview of the project

The computing power on a Personal Computer has dramatically increased in the past years. However, the use of applications such as Steinberg’s Cubase is still limited, because the use of any sound effect implies a large amount of PC computing. Therefore, one cannot use more than a few sound effects without reaching the PC’s computational limits.

Hence our aim was to relieve the PC’s processor while sending the signal to be processed to the DSP. Our work relied on implementing the communication protocol between the Cubase software and the DSP, and sound effects to be used on the DSP.

Cubase software

Cubase is a complete professional music recording system developed by Steinberg (www.steinberg.net). It allows audio and high resolution MIDI recording. Its most interesting feature for us is its VST plug-in system, an easy-to-use, open standard to implement add-in effects.
Specifications

The communication protocol between the Cubase software and the DSP involves the VST protocol provided by Steinberg.

On the other hand, the sound effects that we have implemented comply to the TMS320 DSP Algorithm Standard, in order to ensure a better integration of the system.

Interface between the PC and the DSP

The main goal of this project was to define and develop a communication protocol between the Cubase Software and the TMS320C6700 board. This protocol allows to add audio effects to Cubase that use the DSP's power to process audio effects.
Quick description of Cubase Plug-ins

To implement a plug-in for Cubase, we must use the VST protocol defined by Steinberg. This protocol defines a main object, AEfFectX, and all new audio effects must inherit from this object to be used by Cubase. All new effects are in their own library and Cubase has automatic access to all effects in its effects directory.

The four main functions that an effect must contain are: its initialization, the modification of a parameter, the process of audio samples and the suppression of this effect. These are the four functions that will have an impact on the DSP part of our system.

The communication between the PC and DSP

To let many plug-ins communicate with one single DSP, we have to build a single interface that all effects must use to access to the DSP. This interface, on the PC side, has been done in a single dynamic linking library (DLL) containing our communication object. A single instance of this object is created and as all effects belong to the same process, they are all able to access this unique instance to communicate with the DSP.

This communication object offers all the services required by audio effects to use the DSP: creation of a new effect, modification of a parameter, processing of audio samples and suppression of an effect. Inside the object, synchronization with the DSP and part of the data transfers are carried out with the DSP’s mailbox with the message functions of the EVM host support software. The 32 bits of a message are separated in 8 bits for the effect number, 8 bits for the message type and 16 bits for data associated with the message. All messages transit by the single link, but are re-routed by the interface to its destination effect.

Figure 1. The message structure

<table>
<thead>
<tr>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>24</td>
</tr>
<tr>
<td>23</td>
<td>16</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
</tr>
</tbody>
</table>

Message type | Instance number | Message data

The next figure shows a global view of the system. We see the single communication link between the PC and DSP. Each time a new audio effect is created on the PC, the equivalent effect is created on the DSP side to process the data. If an effect of the same nature is created more than once, the same code is used, but new data space is allocated.
When a new effect is created, the PC asks the DSP to allocate new space for the effect and fills it with the effects characteristics. When this is done, the DSP allocates all buffers associated with the effect and then tells the PC that the effect is ready to be used.

To process data samples, the PC copies the samples in the effect’s input buffers on the DSP side and tells the DSP to process the buffers. When the PC receives the message from the DSP indicating the end of the processing, it can retrieve audio samples from the effect’s output buffers back to the PC. While the DSP is processing the data, the PC is free to update graphical interfaces or to set up data for the next audio effect processing on the DSP.
Figure 3. Typical information exchange through the PCI bus

<table>
<thead>
<tr>
<th>Step in the process</th>
<th>Transfer direction</th>
<th>Detail of the message</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>VST</td>
<td>EVM</td>
</tr>
<tr>
<td>Initialization</td>
<td></td>
<td>Request to create an effect</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Authorization to create the effect</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Request for the structure address</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Acknowledgement and address</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Transfer of the effect structure</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Transfer complete</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Structure processed and initialized</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Request for space allocation for the code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Acknowledgement and address</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Transfer of the code</td>
</tr>
<tr>
<td>Processing</td>
<td></td>
<td>Transfer of buffer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Buffer ready for processing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Buffer has been processed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sending of the processed buffer</td>
</tr>
<tr>
<td>Destruction</td>
<td></td>
<td>Request for destruction of effect</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Acknowledgement : effect destroyed</td>
</tr>
</tbody>
</table>

Arrow definition:

- Standard message sent through the mailbox
- Memory transfer through the HPI
Creating a new effect

New effects for this system have two components: the PC side and the DSP side. On the PC side, implementation is very easy as it has only to deal with graphical interfaces and passing parameters and data to the DSP via the communication object.

The bigger work is then on the DSP side where we have to implement the algorithm and deal with memory allocation for all effects. To ensure good memory usage and make code more reusable, all algorithm must be implemented according to the eXpressDSP standard developed by Texas Instruments. This standard allows a centralised dynamic memory management for all effects.

To keep the “plug-in spirit”, effects should allow to be loaded and unloaded as the software is running without having to reset the DSP card. More than that, we should be able to add new effects without having to recompile all the code on the DSP side. This needs to have a system allowing to upload dynamically new code and execute this new code without restarting the system.

We have studied a method to upload code dynamically and made some tests for it, but it is not operational for now. For this method, we manually get binary executable data into the compiled COFF file and load it into a dynamically allocated memory space onto the DSP card. From the symbol table, we can get all functions offset inside the binary data block and then access them by telling the interface all new addresses of functions. The main problem with this method is that all system functions (like memory allocating function, for example) would be duplicated and such a system cannot work. We must take advantage of the eXpressDSP standard and compile the standard interface IAlg only once with the communication interface, and then dynamically give a vector table to effects for their calls to IAlg functions.

Actually, effects must be compiled all together with the main function and the communication interface. We have great hope that the dynamic code allocating method will work pretty soon.

Performance analysis

To allow Cubase software to function properly, a data frame must be processed in 15 ms. As the DSP card is on the PCI bus, we had hope that transfers would be fast enough to use many effects simultaneously. Theoretically, a stereo effect at 44.1 kHz should need around 5.6MBps while the PCI bus limit is about 1GBps. The problem we have to deal with is that to communicate with the DSP, we have to do it through the host port interface (HPI). As the link between the HPI and the DSP is a lot slower than the PCI bus, it limits the performances. Actually, we can process only one effect at a time.

This limitation is unfortunate, but could be solved with new versions of C67xx DSP processors where the DSP would be directly accessible via the PCI bus without having to use the HPI.
Effects and filters

*The reverberation effect*

We first coded the reverberation effect which can simulate the acoustical behaviour of a room. Since all recording is made in a quite damping environment, what musicians call “reverb” is one of the most useful effects.

Concerning our digital reverb, it has been programmed following the Schroeder/Moorer’s algorithm which is recognized to be as simple as efficient.

One of the goals we had was to implement this algorithm by following the eXpress DSP rules, so as to obtain the clearest result. The program was first created for a TMS320C6711 DSK which allowed us to test the reverb before inserting it into the global project working on the EVM.

The final program is due to be called and controlled from the host through the typical graphic interface of a Cubase plug-in:

![The reverberation interface for Cubase](image)

We can see on the figure the different real-time parameters the user can control:

1. preamp and post gain are simple amplifications of the input and output signal.
2. room size is obviously related to the time of reverberation.
3. damping simulates the air and obstacles absorption.
4. dry level gives the quantity of the original sound arriving to the listener.
5. wet level gives the quantity of the reverberated sound arriving to the listener.
6. and width is linked with the position of the listener in the room (it influences the stereo parameters of the reverb).
The equalization effect

We also coded the equalization effect which allows to increase or decrease the gain of some frequencies. It is used by musicians to obtain the same sound in different rooms by equalization of the answer sound.

This algorithm, like the first one, is eXpress DSP compliant and has been coded to function on a TMS320C6711DSK and on the EVM.

The graphic interface of the algorithm as a Cubase plug in is:

We can see that the user can change the real time parameters for each band of the equalizer:

1. the type of filter applied to the signal
2. the main frequency
3. the gain of the filter
4. the quality factor which allows to compute the width of the filter
Conclusion

With this project, we demonstrated that we can interface a TI DSP with a commercial software that uses plug-ins. Steinberg Cubase was a perfect example to use, because DSP are especially well adapted to process audio effects.

Unfortunately, the EVM DSP board we were working with got down before the end of the contest and we currently are in touch with TI technical support to make it repaired. However, our development work is still in progress, and we have good hopes to have a fully operational system as soon as our technical problems are solved.

By using our communication protocol, it is now very easy to implements new plug-ins and to increase the possibilities of our system, without interfering with already existing ones. There are still some difficulties to go over, like the speed of the communication link between the PC and the DSP, but the main idea remains and could even be adapted to other kind of computer applications that use plug-ins, such as multimedia web-based applications.