Audio Decoding on the C54X

Alec Robinson, Chuck Lueck, Jon Rowlands

Abstract*—Fueled by the excitement over music distribution on the Internet, audio decompression has become a popular topic. There are several different algorithms available, which makes the programmable DSP a nice choice for systems supporting multiple formats. This paper discusses our efforts in porting two audio compression algorithms, MPEG-1 Layer 3 (MP3) and MPEG-2 Advanced Audio Coding (AAC), to the fixed-point C54X DSP.*

Introduction

Within the last year, the popularity of downloadable, compressed audio formats via the internet has skyrocketed. This paper discusses our efforts in porting the decoders of two such compressed audio formats, MPEG-1 Layer 3 (MP3) and MPEG-2 Advanced Audio Coding (AAC), to the C54X fixed-point DSP. MPEG-1 Layer 3 is currently perhaps the most popular compressed audio format, while MPEG-2 AAC offers better audio quality at the same compression ratios.

Platform Description

The target processor for development was the C54X, TI's low-power, 16-bit fixed-point DSP. Our initial design goal was to have both decoders running on the C5410 processor, which has 64 kwords of RAM available and up to 100 MIPS of computation power, while our ultimate goal is to get the same functionality running on the C5409, which has 32 kwords of RAM along with a 16 kword ROM for holding tables, constants, etc.

For code development, we made significant use of the Spectrum Digital C54X EVM. With this, it was possible to profile the software, perform functional and diagnostic testing on the C54X, etc. For real-time code development, the TI C5410 Internet Audio EVM, developed within the Portable Audio Group at TI, was used. This board is a microless design with a compact flash for holding music and a user interface for controlling playback. Support is provided for audio decompression, sample rate conversion, graphic equalization, and digital volume control. The DSP also handles user interface operations and compact flash I/O.

Algorithms Overview

Our efforts have been focused on two standard MPEG audio compression algorithms: MPEG-1 Layer 3 and MPEG-2 Advanced Audio Coding. MPEG-1 Layer 3, now commonly referred to as MP3, was standardized by MPEG in 1991 [1]. Recently, it has gained great popularity for downloading audio content via the internet. MP3 has the capability of compressing high-quality audio at an 11:1 compression ratio (or 128 kbps) with little loss in sound quality.

MPEG-2 Advanced Audio Coding, or AAC, was developed as a non-backward compatible (with respect to MPEG-1) addition to MPEG-2 Audio, promising better quality without the burden of having to support backward compatibility. Standardized in April of 1997 [2], AAC became the first codec to achieve transparent quality audio (by ITU definition) at 64 kbps per audio channel [3]. AAC has also been adopted as the high-quality audio codec in the new MPEG-4 compression standard, and has also been selected for use in the future Japanese HDTV standard.

A block diagram of the MP3/AAC audio decoder is shown in Figure 1. Both decoders are transform based, have nonlinear quanitization, and use extensive variable-length coding. Temporal noise shaping (TNS), a technique unique to AAC, provides better performance for signals with highly-varying time envelopes, such as speech, which are traditionally difficult to compress using a transform-based codec.

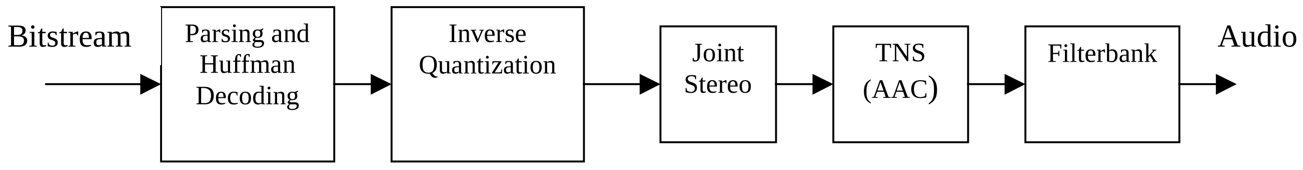

Figure 1 - Block diagram of MP3/AAC Audio Decoder

Implementation

For both decoders we started the C54X implementation with a floating-point code base running on the PC. Being confident in the quality of this code, we used it as a reference for all the changes we made.

Compiling this code directly with the C54X C compiler, we were able to get a *very* non-real time version running on the Spectrum Digital EVM. This served two purposes. First, it gave us a working decoder on C54X hardware, and second, it gave us an opportunity us to create configuration files for our build environment and EVM boards. With these components in place, the rest of the development was able to move more quickly and smoothly.

Since the C54X is a fixed-point part the next major hurdle was converting the floating-point C into fixed-point. With the help of a fixed-point math library, we did this conversion on the PC where we could quickly reference changes to the floating-point code. This allowed us to see how changes in Q-points at various stages affected precision in the PCM output. After we had this optimized on the PC, we compiled it for the C54X with little effort.

Having all the Q-point issues worked out, we profiled the code on the C54X EVM extensively with the TI software tools (Code Composer and TI Emulator.) This gave us a picture of how far off real-time we were, and told us where to focus our assembly optimizations. After every optimization, we went back to profiling to find the next area to target. We used this recursive procedure until we had the code running at our MIPS goal.

Filterbank

Both MP3 and AAC get their data reductions by transforming a time domain audio signal into the frequency domain, and quantizing the spectral data. Transform coders are able to remove perceptually irrelevant information and thus reduce the number of bits required to represent the signal. In the encoder, a filter bank performs this transformation, and in the decoder, the inverse filter bank is used to get back to a time signal.

The filter banks in MP3 and AAC are different. MP3 uses a hybrid approach by first filtering the signal into 32 bands then performing a small MDCT on the output of each band. AAC uses one large MDCT. Another difference is the size of the transformation. MP3 splits the frequency domain into 576 spectral lines while AAC uses 1024. In order to control the spread of quantization noise in the time domain, both coders use a technique called “window switching” which allows for some control over time-frequency resolution.

Our filter bank implementations for both decoders use 32-bit precision for the best PCM precision at the output. Since memory usage is a concern our filter banks were designed to minimize state storage. Unfortunately, for MP3, there are two filter banks which means we need two state buffers, one for each bank. Despite only having half the number of spectral lines as AAC, MP3’s state buffer requirements are about the same. The low-level functions, which perform the required signal processing, were hand-coded in assembly.

Parser

Both MP3 and AAC require significant decoding of variable length codes. Since the computational requirements for such codes are somewhat dependent on the bitrate, we have made a considerable effort to reduce the complexity of the parser and associated

Huffman decoder. Fast algorithms for Huffman decoding have been implemented and the core processing routines have been coded in assembly language. For AAC, the Huffman codebooks have been restructured based on prefix lengths to take advantage of a special C54X instruction which is useful for counting strings of zeros or ones. Audible fast-forward/reverse have also been implemented, a task which is non-trivial due to the large degree of Huffman coding in the bitstream and the fact that the syncword in the transport layer is not unique (can be emulated by Huffman code patterns).

Quantization

MP3 uses a nonlinear inverse quantizer to reconstruct the spectral values from the quantized values transmitted in the bitstream. To handle the most common values, a table lookup has been implemented. For other values, a fast implementation of the nonlinear function has been written in assembly to minimize computational requirements. The end result is a trade off between MIPS and memory: a larger table requires more memory but fewer MIPS. For AAC, a similar procedure is also used.

Temporal Noise Shaping

Temporal noise shaping, unique to AAC, provides better quality for audio signals which have large variations in their time-domain envelopes, such as speech. In the encoder, temporal noise shaping requires the application of an FIR filter to the frequency-domain spectral components prior to quantization. In the decoder, the inverse filter, an all-pole filter, must be applied to the reconstructed spectral components prior to inverse transformation. In our implementation, the all-pole filter has been implemented in double-precision, and special precautions have been taken to avoid potential overflow problems.

Conclusion:

This paper discussed our efforts in porting two audio compression algorithms, MPEG-1 Layer 3 (MP3) and MPEG-2 Advanced Audio Coding (AAC), to the fixed-point C54X DSP. Implementing audio decoders on the C54X requires careful planning to minimize memory and MIPS, and to preserve precision at the output.   The high-level software structure of the decoder can be written in C, but many low-level signal processing functions must be hand-coded in assembly to meet real-time performance.

References:

[1] ISO/IEC JTC1/SC29/WG11 MPEG. International Standard IS 11172-3 Information Technology – Generic Coding of Moving Pictures and Associated Audio, Part 3: Audio, 1991.

[2] ISO/IEC JTC1/SC29/WG11 MPEG. International Standard IS 13818-7 Information Technology – Generic Coding of Moving Pictures and Associated Audio, Part 7: Advanced Audio Coding, 1997.

[3] D. Meares, K. Watanabe, E. Schreirer, "Report on the MPEG-2 AAC Stereo Verification Tests", ISO/IEC JTC1/SC29/WG11 MPEG document N2006, February 1998.