Philippine Engineering Journal (2000) XXI (1): 28-37

MP3 DECODER USING TI TMS320C6X DSP CHIP

Bienvenido H. Galang Jr.
Lemuel Q. Quiwa
Computer Software Division
Advanced Science and Technology Institute
Department of Science and Technology

ABSTRACT

MPEG I Layer III, more commonly known as MP3, is the de facto standard of digital audio compression. It is an open standard for low bit rate coding of audio signals. It is able to reduce an audio file up to a factor of 12 without losing fidelity—CD quality.

This paper describes the theory behind MP3 and the implementation of ISO/IEC 11172-3 Audio Layer III decoding using the TMS320C6X Evaluation Module. The software was written in ANSI C and it was cross-compiled through Code Composer Studio.

The project is able to decode an MP3 file, store the result in a buffer and play the decoded segment using the codec of the evaluation module, or it can store the PCM samples as a raw file. This raw file can then be played using software such as Goldwave.

Introduction

MP3 is a shortcut for MPEG1 layer III. MPEG or the Moving Pictures Expert Group was formed to provide the standards for audio and video coding schemes at low data rates. These standards are the first international standards in the field of high-quality digital audio and video compression. MP3 is the most powerful compression technique but it is also the most complex. The compact disk (CD), today's de facto standard of audio representation, uses 16-bits per sample at 44.1 kHz and holds up to 72 minutes of audio. With the use of MP3, a single CD can hold up to 8 hours of audio without losing fidelity—"CD quality"[1]. That's basically it—an MP3 file is a *reduced audio file.*

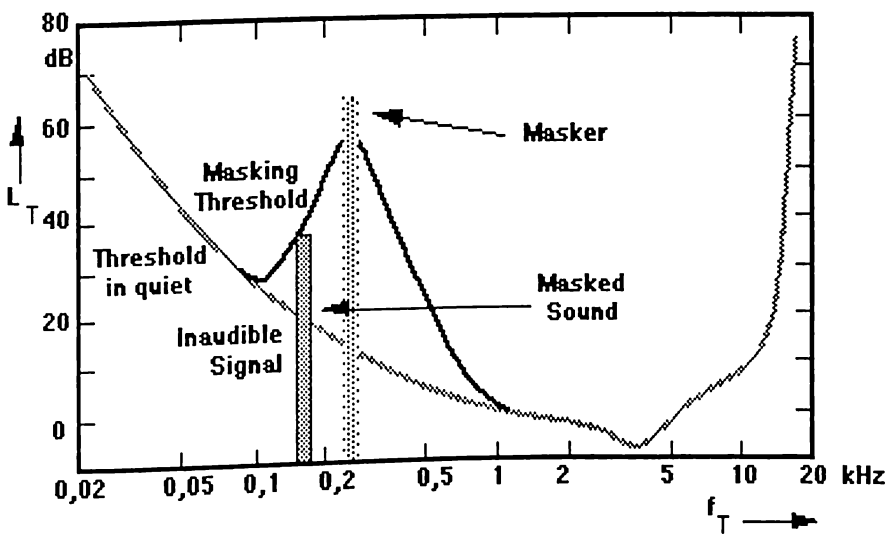
The classic way of size reduction is to reduce the information. For example, if we reduce the 16-bit PCM samples to 8-bit PCM samples, we reduce the file size by half but we lose half of the quality and probably gain more noise in the process. If we reduce the sampling frequency, we lose resolution[2].

Modest reductions in audio bit rates were done by instantaneous companding (μ-law/A-law)—reducing 16-bit PCM into 11-bit PCM; various forms of block companding (DSR, NICAM); and adaptive differential PDM (ADPCM)—getting the difference between two consecutive samples instead of the samples themselves [1].

Excellent audio coding performance was obtained by using various frequency coders—Subband Coding (SBC) and Adaptive Transform Coding (ATC). These coding techniques are the ones used by MPEG for file reduction. Other recent coding technologies employed by MPEG are: Perceptual Audio Coding, Frequency Domain Coding, Window Switching, and Dynamic Bit Allocation.

Basically, an efficient source-coding algorithm will remove redundant components of the source signal by exploiting correlations between its samples and remove components that are perceptually irrelevant to the ear [1]. This is how MP3 works—it strips away information that is not important. It decides what information is necessary and what is not, based on the research of human perception.

Before we hear anything, our brain analyses the incoming data and it interprets the sound and filters irrelevant information. MP3 just does the job earlier—this is called Perceptual Audio Coding. PAC uses simultaneous masking and temporal masking. Simultaneous masking is a frequency domain phenomenon where a low-level signal (maskee) can be made inaudible (masked) by a simultaneously occurring stronger signal (masker) as long as the masker and the maskee are close enough to each other in frequency. A masking threshold can be measured and low-level signals below this threshold will be inaudible. If there is no masker, a low-level signal will be inaudible if it is below the threshold in quiet, which depends on the frequency. Temporal masking is a time domain phenomenon and it occurs when two sounds appear within a small interval of time. Depending on the individual Sound Pressure Levels (SPL's), the stronger signal may mask the weaker signal even if the maskee precedes the masker. MP3 reduces file size by removing these inaudible signals [3].



Perceptual Audio Coding (Masking)[4]

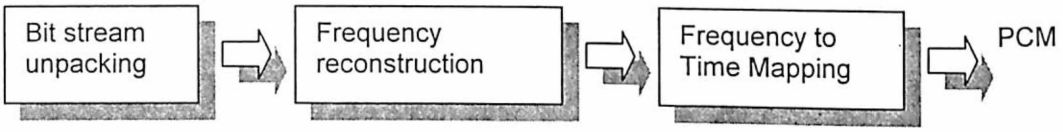
In Frequency Domain Coding, redundancy (the non-flat short-term spectral characteristic of the source signal) and irrelevancy (signals below the psychoacoustical thresholds) are exploited to reduce the transmitted data rate with respect to PCM. This is achieved by splitting the source spectrum into frequency bands to generate nearly uncorrelated spectral components and by quantizing these components separately. There are two coding categories: Transform Coding and Subband Coding. In TC, a block of input samples is linearly transformed via a discrete transform into a set of nearly uncorrelated transform coefficients. These coefficients are then quantized. In SBC, the source signal is fed into an analysis filerbank consisting of M subband filters. Each decimated filter output is quantized separately. The quantization error is limited for each band [1].

These are the techniques used in the encoding algorithm to effectively reduce the size of an audio file without losing fidelity.

The purpose of this project was to implement the MP3 decoding algorithm to the TI TMS320C6X Evaluation Module, to learn the fundamentals of MP3, and to learn the basics of Code Composer Studio. The knowledge learned from this project may be used to make a stand-alone MP3 player prototype in the future.

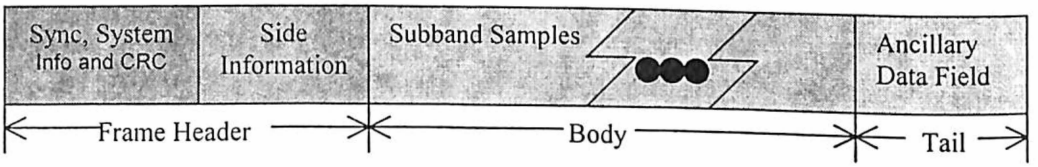
MP3 Decoding

The decoding process is just the reverse of the encoding process. A simple block diagram of the decoding process is shown below [5]:

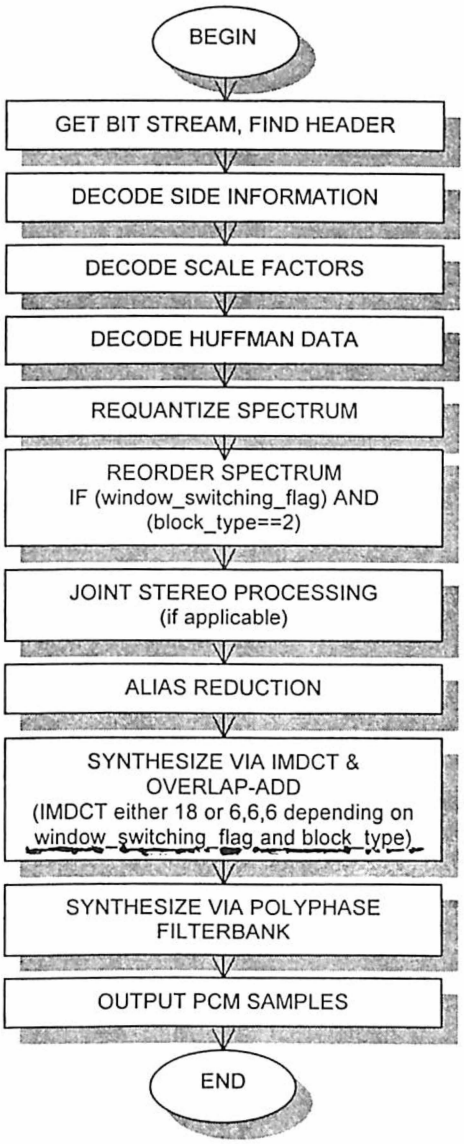


An MP3 file is a bit stream of ones and zeroes divided into frames. First, the necessary information is extracted from the header of the frame. Then the samples are used to reconstruct the source spectrum. Finally, we convert from frequency domain to time domain.

An MP3 frame consists of three parts: header, body, and "tail". The header contains all the necessary information about the MP3 file, the body contains the sample or the main data, and the "tail" contains the ancillary data. The frame header consists of two parts. The first part contains 12 synchronization bits, 20-bit system information, and an optional 16-bit cyclic redundancy check (CRC) code. The second part consists of the side information—about the bit allocation and the scale factors. The length of each frame is not constant, due to the fact that the length of the main information field depends on the bitrate and the sampling frequency. Each frame is also autonomous. You do not need the previous frame, to decode the next frame. The frame structure is shown below [1]:



The MP3 decoding algorithm consists of eleven blocks. The decoding flowchart is shown below [6].



The first step is to find the 12-bit syncword, 1111 1111 1111, to synchronize the incoming bitstream. In some cases, the first 16 bits of the header becomes the 16-bit syncword since the ID, layer, and protection status are already known. The first 32 bits make up the header, and each bit or group of bits contain information about the MP3 file. Please see reference [6] for the complete details, Huffman tables, etc.

The next step is to extract the side information from the bitstream and store it as the current frame is being decoded. The table select information is used to identify which Huffman table is going to be used and how many ESC-bits (linbits) will be used.

In layer III, the body containing the main data is not necessarily located next to the side information due to the Huffman coding. The beginning of the main data is located by using the

main_data_begin pointer of the current frame. The main data is allocated in a way that all main data are resident in the input buffer when the header of the next frame is arriving in the input buffer. The decoder skips header and side information when decoding the main data because their position is already known from the bitrate_index and padding_bit. The header is always made up of 4 bytes; the side information is always made up of 17 bytes in mode single_channel and 32 bytes in the other modes. The main data can span more than one block of header and side information [6].

After decoding the side information, the scalefactors are decoded using slen1 and slen2. The values of slen1 and slen2 are determined from scalefac_compress. The decoded values can be used to calculate the factors for each scalefactor band or they can be used in lookup tables. When decoding the second granule, the scfsi has to be considered. When scfsi is set to 1, the scalefactors of the first granule are also used for the second granule therefore they are not transmitted for the second granule [6].

The number of bits used to encode scalefactors is called part2_length and is calculated as follows [6]:

For block_type==0, 1, or 3 (long blocks)
part2_length=11*slen1+10*slen2

For block_type==2 (short blocks) and mixed_block_flag==0;
part2_length=18*slen1+18*slen2

For block_type==2 (short blocks) and mixed_block_flag==1;
part2_length=17*slen1+18*slen2

The main data is then fed to a Huffman decoder. All the information we need to generate the Huffman code tree can be found from the 32 Huffman tables. First, the big_values are decoded. The frequency lines in region 0, region 1, and region 2 are Huffman decoded in pairs until big_values number of line pairs has been decoded. The remaining Huffmancodebits are decoded. Decoding is done until all Huffman code bits have been decoded or until quantized values representing 576 frequency lines have been decoded, whichever comes first. If there are more Huffman code bits than necessary to decode 576 values, they are regarded as stuffing bits and are discarded [6].

The data is then requantized using a non-uniform quantizer. For each value "is", from the Huffman decoder, "$|is|^{4/3}$" is calculated or its value is determined from a lookup table. If short blocks are used, the rescaled data shall be reordered in subband order prior to the IMDCT operation [6].

After requantization, the reconstructed values are processed for MS Mode, Intensity Mode, or both; before going to the synthesis filterbank. If MS_stereo is enabled but intensity stereo is not, the entire spectrum is decoded in MS_stereo. If they are both enabled, the upper bound of the scalefactor bands decoded in MS_stereo is derived from the "zero-part" of the difference (right) channel. In this case, the scalefactor band in which the last non-zero (right channel) frequency line occurs is the last scalefactor band to which the MS_stereo equations apply. In MS_stereo, the values of the normalized middle/side channels M_i/S_i are transmitted instead of the left/right channel values L_i/R_i. Thus L_i/R_i are reconstructed using [6]:

$$L_i = \frac{M_i + S_i}{\sqrt{2}} \qquad R_i = \frac{M_i - S_i}{\sqrt{2}}$$

The values M_i are transmitted in the left, S_i values are transmitted I the right channel [6].

In layer III, intensity stereo is not done using a pair of scalefactors as in layers I and II, but by specifying the magnitude; and a stereo position is transmitted instead of scalefactors for the right channels. The stereo position is used to derive the left and right channel signals according to the formula [6]:

$$L_i = L_i * \frac{is\_ratio}{1 + is\_ratio} \qquad \text{for all indices i within sb}$$

$$R_i = R_i * \frac{is\_ratio}{1 + is\_ratio} \qquad \text{for all indices i within sb}$$

For long block type granules, alias reduction is performed prior to IMDCT. Alias reduction in not performed with block_type==2 (short blocks) [6].

To calculate for the Inverse Modified Discrete Cosine Transform, we use the formula [6]:

$$x_i = \sum_{k=0}^{n/2-1} x_k \cos\left[\frac{\pi}{2n}\left(2i + 1 + \frac{n}{2}\right)(2k+1)\right] \qquad \text{for } i = 0 \text{ to } n-1$$

where n is the number of windowed samples (n=12 for short blocks; n=36 for long blocks)

In the synthesis filterbank, the frequency lines are preprocessed by the alias reduction scheme and fed into the IMDCT matrix, each 18 into one transform block. The first half of the output values are added to the stored overlap values from the last block. These values are new output values and are input values for the polyphase filterbank. The second half of the output values are stored for overlap with the next data granule. For every second subband of the polyphase filterbank every second input value is multiplied by –1 to correct for the frequency inversion of the polyphase filterbank [6].

Depending on the block_type different shapes of windows are used [6]:

block_type=0 (normal window)

$z_i = x_i \sin\left[\frac{\pi}{36}(i + \frac{1}{2})\right]$ for i = 0 to 35

block_type=1 (start block)

$z_i = x_i \sin\left[\frac{\pi}{36}(i + \frac{1}{2})\right]$ for i = 0 to 17

$z_i = x_i$ for i = 18 to 23

$z_i = x_i \sin\left[\frac{\pi}{12}(i - 18 + \frac{1}{2})\right]$ for i = 24 to 29

$z_i = 0$ for i = 30 to 35

block_type=3 (stop block)

$z_i = 0$ for i = 0 to 5

$z_i = x_i \sin\left[\frac{\pi}{12}(i - 6 + \frac{1}{2})\right]$ for i = 6 to 11

$z_i = x_i$ for i = 12 to 17

$z_i = x_i \sin\left[\frac{\pi}{36}(i + \frac{1}{2})\right]$ for i = 18 to 35

block_type=2 (short block)

Each of the three short blocks is windowed separately

$y_i^{(j)} = x_i^{(j)} \sin\left[\frac{\pi}{36}(i + \frac{1}{2})\right]$ for i = 0 to 11, j = 0 to 2

The windowed short blocks must be overlapped and concatenated [6]. Please see the reference for more technical details. The encoding algorithm and the psychoacoustic model can also be found in the reference.

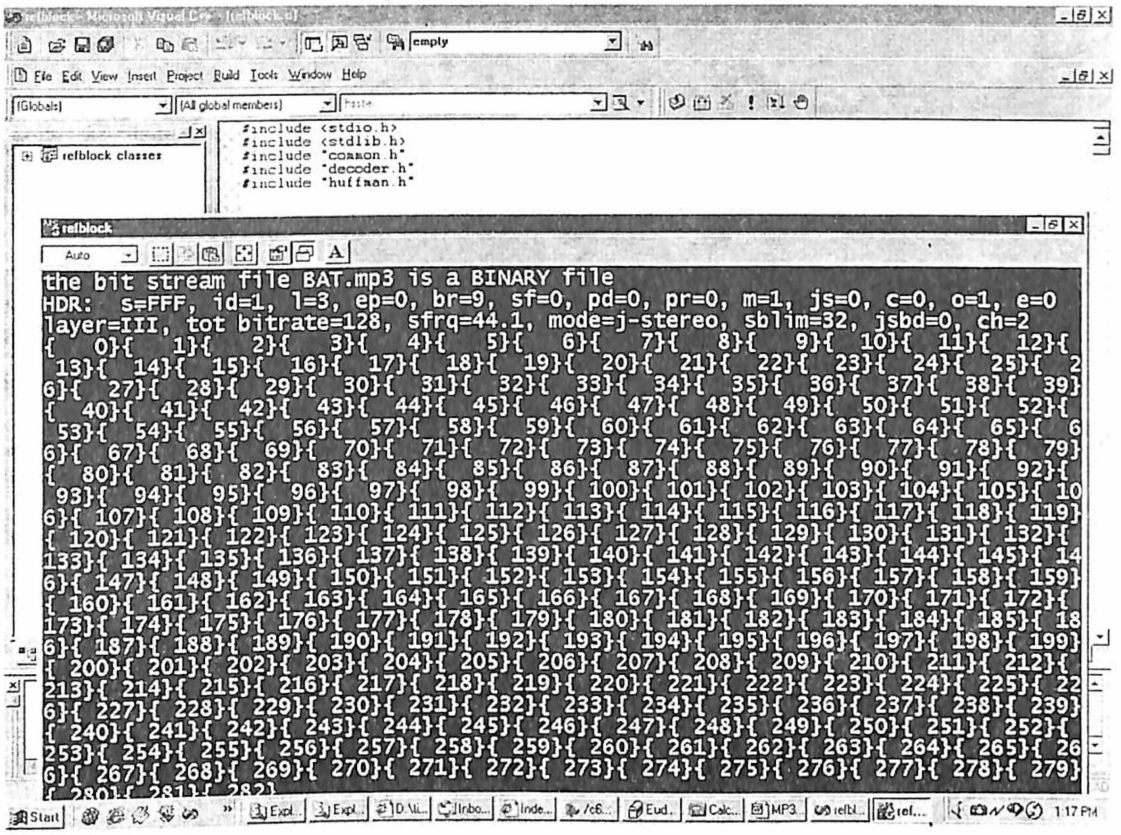
Methodology

To implement the MP3 decoding algorithm, we took the necessary files and functions from 32 files downloaded from the Internet. From these functions, we built our code. The code was written in ANSI C and it was cross-compiled using Code Composer Studio. Code Composer Studio is the development software for the TMS320C6X Evaluation Module. It contains the GUI, Assembler, Compiler, Debugger, and it loads the COFF file to the target DSP chip. We implemented the decoding algorithm by block, starting with the first block, then adding the next block, until all eleven blocks are complete. It was quite hard implementing the code because the behavior of the code was quite different from its behavior when compiled in C. To debug the code, we had to run the code using Visual C++, trace it and compare the results when running the code using Code Composer Studio.

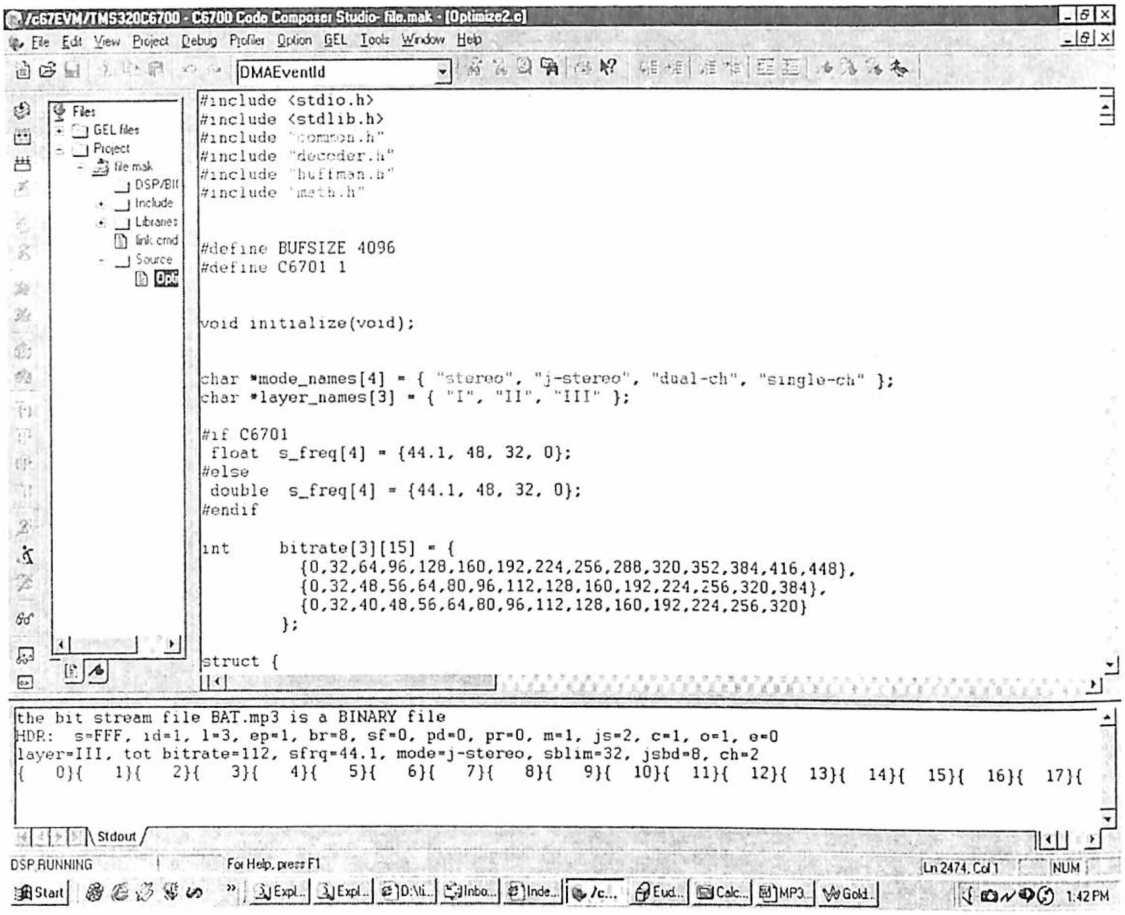
Results

We were able to implement all eleven blocks. We were able to decode an MP3 file, and store the result in a raw file. Decoding a 4-5 minute song took about a day and the result was a 35-40 MB raw file. The raw file can then be played using Goldwave. We were also able to use the codec of the evaluation module. Instead of storing the result in a raw file, we stored it in a buffer, and transmitted it to the codec. It took the buffer 20-30 minutes to fill up and transmit the result to the codec. It was able play 3-5 seconds of audio.

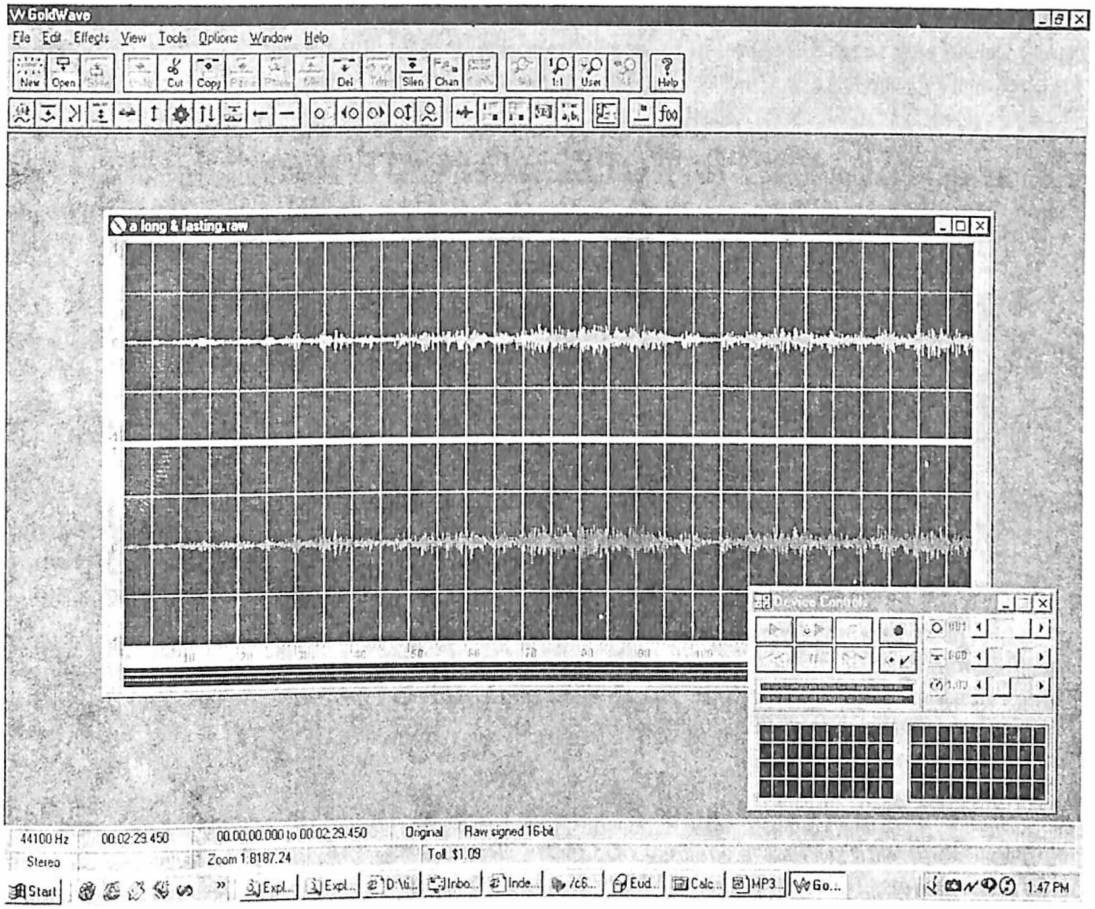
The code was compiled using Visual C++ to test its workability. This was then used as our reference in implementing the decoding algorithm to the evaluation module. Decoding an MP3 file took about 8-10 minutes. The code prints out the information about the MP3 file, the frames decoded, and it stores the result in a raw file. Decoding an MP3 file using Visual C++ is shown below:



Using Code Composer Studio, we were able to cross compile the code and load the COFF file to the target DSP. The code would also print the information about the MP3 file, print the frames decoded, and store the result in a raw file. Decoding an MP3 file using TMS320C6X Evaluation Module, is shown below:



The raw file can then be played using Goldwave. The settings are: 16-bit, signed, stereo, 44.1 sampling frequency. In some cases, you need to swap the bytes (select byte-swapped). The raw file decoded is shown below:



Conclusion

The MP3 decoding algorithm was successfully implemented using Code Composer Studio and TMS320C6X Evaluation Module. The MP3 file was decoded and it was played using the codec of the evaluation module but it was not played in real time. The next step would be to optimize and speed up the operation of the code to make it real time. This would utilize the DSP BIOS of the TMS320C6X Evaluation Module. We tried loading the MP3 file to the memory of the evaluation module hoping to speed up the decoding but there was only a little improvement. The decoding speed increased by less than 10%.

References

[1] Peter Noll. **"MPEG Digital Audio Coding"**. IEEE Signal Processing. Vol. 14. No. 5
[2] http://www.mpeg.org
[3] Marina Bosi. **"Perceptual Audio Coding"**. IEEE Signal Processing. Vol. 14. No. 5
[4] http://www.iis.fhg.de/amm/techinf/basics.html
[5] Davis Pan. **"A Tutorial on MPEG/Audio Compression"**. IEEE Multimedia Journal. 1995
[6] MPEG. **"ISO/IEC 11172-3"**. 1993