A Simple DSP Laboratory Project for Teaching Real-Time Signal Sampling Rate Conversions

by

Li Tan, Ph.D.
lizhetan@pnc.edu
Department of ECET
Purdue University North Central
Westville, Indiana

Jean Jiang, Ph.D.
jjiang@devry.edu
Department of ECET
DeVry University
Atlanta, Georgia

Abstract: In this paper, we present a simple laboratory project for teaching real-time sampling rate conversions of digital signals using low-cost digital signal processors. Usually, the topic of the sampling rate conversion is discussed in the second digital signal processing (DSP) course in undergraduate programs of engineering technology. The paper deals with a situation for teaching sampling rate conversions when the different sampling rates of analog to digital conversion (ADC) and digital to analog conversion (DAC) cannot be flexibly set during real-time processing. To overcome this barrier, we employ a compromised way in which we first request students to store audio segments sampled at different sampling rates in the processor memory and let students implement their designed DSP algorithm to covert each pre-sampled audio segment to produce an output at the DAC sampling rate. The method could be applied as an alternative when the sampling rates for ADC and DAC devices in the DSP system cannot be freely changed for sampling rate conversions.

Index Terms: Real-time digital signal processing, sampling rate conversion, decimation, interpolation, anti-aliasing filter, anti-image filter

I. Introduction

Recent advancement of digital signal processing (DSP) technology has a dramatic impact on the disciplines of electrical, computer, and biomedical engineering technology programs. Technologists are often desirable to be familiar with digital signals and systems, and basic

DSP techniques, and to possess DSP working knowledge towards applications in order to keep up with the industry trends. Many undergraduate programs in engineering technology not only offer a course to cover fundamentals of DSP, but also continue to provide a second elective DSP course in which real-time applications and corresponding advanced topics such as multi-rate signal processing and adaptive filtering are introduced [1]. In many DSP applications, there is often a requirement for lowering or rising of a sampling rate. Such applications include speech and audio systems where the sampling rate of original data in storage media does not match the sampling rate set in the DSP system. As an example in [1, 2], a DSP system may be required to convert an MP3 audio with a sampling rate of 48 kHz to a CD audio having a sampling rate of 44.1 kHz, or to digital voice at the 8 kHz sampling rate, vice versa.

Although we could teach basic principles of sampling rate conversions using MATLAB simulations, engineering technology students would prefer to have hands-on DSP coding experience for real time processing to enhance their understanding of the rate conversion principles. On the other hand, the sampling rate of a DSP processor may be set to be fixed along with the embedded analog anti-aliasing filter before analog to digital conversion (ADC) device and analog reconstruction filter after digital to analog conversion (DAC) device such as TX320TMS67C1x DSK [1, 3, 4, 5]. This limitation could render teaching real-time sampling rate conversions to be a challenging task due to the demand of one sampling rate required for ADC device and other different sampling rate for DAC device. To tackle this problem for teaching the subject, we employ a compromised approach described as follows. We first request students to store audio segments sampled at different sampling rates in memory of the DSP system and then let students implement their designed DSP algorithms to covert each pre-sampled audio segment to a processed digital output at the DSP system sampling rate. Practical implementations of real-time sampling rate conversions in different cases such as sampling rate reduction using an integer factor, sampling rate increase with an integer factor, as well as sampling rate change by a non-integer conversion factor are illustrated. The teaching method can be adopted as an alternative when the sampling rates for ADC and DAC devices in the DSP system could not be flexibly changed for the sampling rate conversion processing. In this paper, Section II describes the principles of sampling rate conversions and laboratory considerations for real-time processing, Section III depicts a laboratory project dealing with practical real-time implementations, and finally, Section IV shows summaries and conclusions.

II. Sampling Rate Conversions and Laboratory Considerations

A. Sampling rate conversion schemes

Figure 1 shows three typical schemes of sampling rate conversions. Reduction of a sampling rate by an integer factor M, referred to down-sampling process or decimation, is shown in Figure 1a, in which a reduced sampling rate $f_{sM} = f_s / M$ can be obtained, where f_s is the original sampling rate in Hz. Symbol $\downarrow M$ indicates down-sampling operation in which for every M samples, the operation will keep the first sample and discard the rest of $(M-1)$

samples. As an example of using the scheme, if the original sampling rate and reduction integer factor are required to be $f_s = 8$ kHz and $M = 4$, respectively, the scheme will produce a new sampling rate as $f_{sM} = f_s / M = 2$ kHz. Since the reduced sampling rate of 2 kHz has a new Nyquist limit of 1 kHz, any components with frequencies beyond 1 kHz in the original data samples will be aliased. In order to prevent possible aliasing noise after down-sampling process, a digital anti-aliasing filter with the low-pass type and stop frequency edge of 1 kHz must be designed and applied prior to the down-sampling operation. Notice that in Figure 1a, $x(n)$ and $w(n)$ are the filter input and output, respectively, and the filter operates at the original sampling rate f_s. Index n indicates the time index of the original samples while index m is the time index of the down-sampled output $y(m)$.

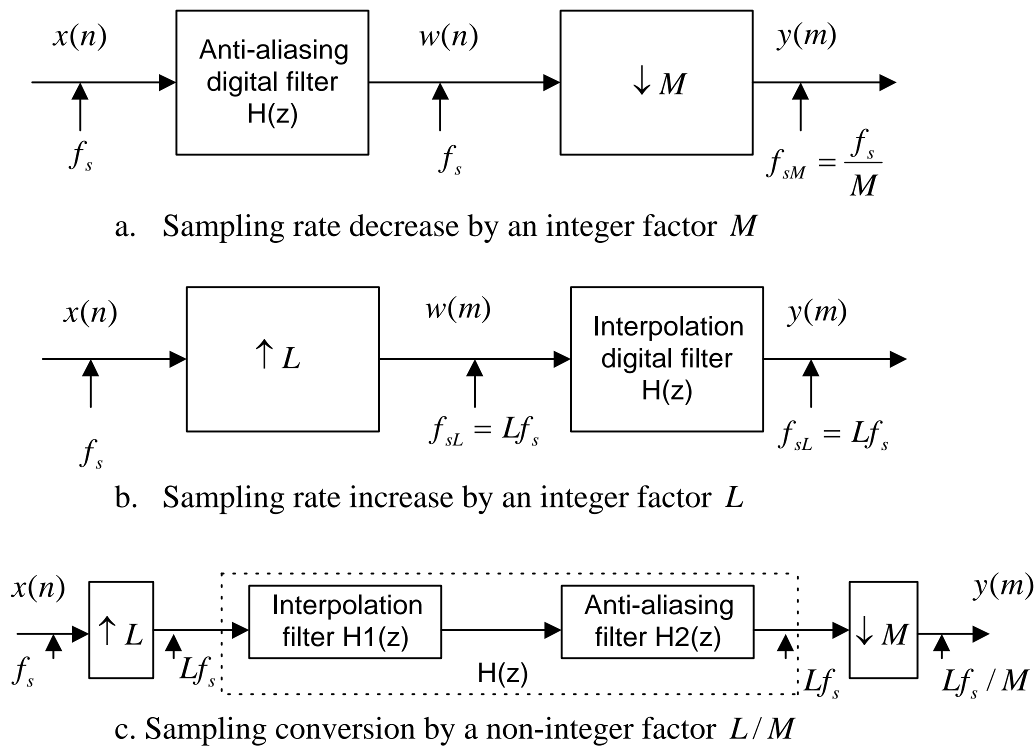

Fig. 1 Block diagrams for various sampling rate conversions.

Figure 1b depicts the second case for increasing a sampling rate by an integer factor L, referred as up-sampling process or interpolation. Symbol $\uparrow L$ designates the up-sampling operation in which for each input sample, the operation will append it with $(L-1)$ zeros. If the original sampling rate and increasing integer factor are $f_s = 8$ kHz and $L = 3$, an increased sampling rate will be $f_{sL} = Lf_s = 24$ kHz. Since the Nyquist limit of the original

samples, in this case, is 4 kHz and the up-sampling scheme results in a new Nyquist limit of 12 kHz, the image signals between 4 kHz and 12 kHz will be introduced, resulting in a distortion in the processed output. Hence, an anti-image filter (also called interpolation filter) with the low-pass type and stop frequency edge at 4 kHz should be designed and applied after a up-sampling operation removes the image noise. Again, since the anti-image filter is cascaded after the up-sampling operation, it operates at the increased rate, that is, $f_{sL} = Lf_s$, with the filter input and output designated as $w(m)$ and $y(m)$, respectively. Note that time indices n and m are the time indices at the original sampling rate and up-sampling rate, respectively.

The third case for changing a sampling rate with a non-integer factor L/M is shown in Figure 1c in which if the original sampling rate $f_s = 8$ kHz, and integer factors $M = 4$ and $L = 3$ are given, the conversion scheme leads to a resultant sampling rate as $Lf_s / M = 6$ kHz. The process can be obtained via cascading the down-sampling scheme after the up-sampling scheme. As shown in Figure 1c, we notice that the anti-image filter (interpolation filter) $H_1(z)$ and anti-aliasing filter $H_2(z)$ operate at the same rate Lf_s. Hence, these two filters can be combined into a single filter $H(z) = H_1(z)H_2(z)$ to reduce the implementation complexity. The combined filter is also a low-pass type and designed by using the most demanding requirements extracted from both anti-image and anti-aliasing filters. In this laboratory project, all digital filters designed are FIR type for simplicity. Engineering technology students are capable of designing various FIR filters using the MATLAB toolbox because they have learnt digital filter design in the first required DSP course. Before starting this laboratory project, the principles of sampling rate conversions associated with FIR filters are taught using the textbook [1] at the engineering technology level. We focus on presenting a laboratory technique for students to experience real-time sampling rate conversions.

B. Laboratory Considerations

For some low-cost DSP systems such as the TMS320C6711 DSK, where ADC and DAC sampling rates are set to be 8 kHz and analog anti-aliasing and reconstruction filters are embedded, changing sampling rates for ADC and DAC is not flexible. In order to let engineering technology students concentrate on coding DSP algorithms, Figure 2 describes a simple laboratory model. As shown in Figure 2, the lab uses the existing DAC sampling rate f_s and analog reconstruction filter, and requires students to convert each pre-sampled audio segment sampled at different sampling rates $\overline{f_s}$ and stored in DSP memory. After sampling rate conversion, the processed output will be output at the existing sampling rate f_s, displayed, and played in real time. In this way, we avoid the problem of changing different sampling rates for ADC and DAC devices.

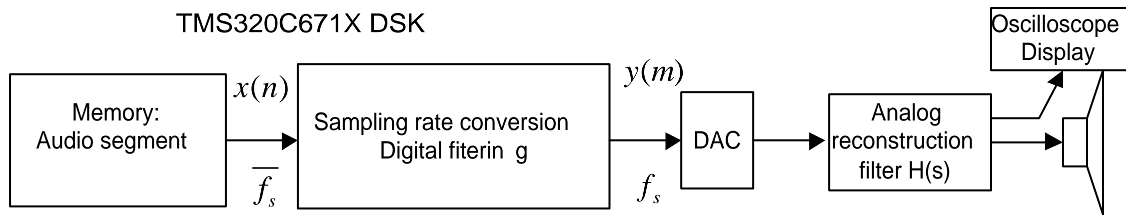

Fig. 2 Laboratory setup for sample rate conversions.

In this laboratory project, MATLAB offers a tool to generate audio data segments at different sampling rate $\overline{f_s}$ so that pre-processed audio data segments could be loaded into processor memory for future real-time conversion. Students also use MATLAB to design the anti-aliasing as well as anti-image filters used in rate conversion systems.

III. Laboratory Implementations

The laboratory project requires three phases to complete. First, students need to prepare audio data samples consisting of sinusoids and pre-processed audio data. Each size of the audio segments is limited to a size of 4 K bytes so that after loading data samples into the DSP memory, there is still enough memory for program codes. Each audio sample is encoded as 16 bits per sample and each audio segment is repetitively applied in real-time conversion processing. Table 1 lists the parameters used for generating the different sinusoidal waveforms used in this laboratory project. Note that the first sinusoidal segment contains frequency components of 1 kHz and 5 kHz sampled at 16 kHz. It can be demonstrated that the 5 kHz signal must be filtered (anti-aliasing filtering) before the down-sampling operation due to a fact that the DSP system at the 8 kHz sampling rate can only handle the component with the maximum frequency up to 4 kHz (Nyquist limit), otherwise, the aliasing signal of 3 kHz will be introduced instead. Segments 4 and 5 are the voice segments each containing a word of "we" (we.wav) obtained as follows. Students use a voice segment (we.wav) with duration of 0.5 second originally sampled at 48 kHz with each audio sample encoded by 16 bits. This voice segment will be further down-sampled to have sampling rates of 2 kHz and 3.2 kHz for experiments, respectively. Students will perform filter design and sampling rate conversion using MATLAB. The fundamentals for sampling rate conversions are covered in class as shown in the textbook [1].

Table 1: Data sets and conversion requirements

No. Seg.	Frequency (Hz)	Original sampling rate	Size (samples)	Conversion type	Converted sampling rate
1	1 k and 5 k	16000 Hz	1600	M=2	8000 Hz
2	400 Hz	1600 Hz	1600	L=5	8000 Hz
3	1 k and 5 k	12000 Hz	1200	L=2, M=3	8000 Hz
4	"we"	2000 Hz	1000	L=4	8000 Hz
5	"we"	3200 Hz	1600	L=5, M=2	8000 Hz

The second phase begins with designing anti-aliasing and anti-image digital filters for each sampling rate conversion to be performed in real time. Students are required to design digital FIR (finite impulse response) filters (anti-aliasing, anti-image, and the combined filter) to meet pass-band ripples less than 0.1 dB, stop-band attenuation less than 45 dB, transition bandwidth of 800 Hz for segment 1 and segment 3, and transition bandwidth of 400 Hz for the rest of segments. Table 2 lists each FIR filter specifications including designs with Hamming window, cutoff frequency, and number of taps (coefficients).

Table 2: FIR Filter specifications
(All filters are low-pass FIR filters and designed using Hamming windows; the converted sampling rate is 8000 Hz)

No. Seg.	Original sampling rate	Transition band	Filter Operating frequency	Filter type, Cut-off frequency (fc) Number of taps (N)
1	16 kHz	3.2 - 4 kHz	16 kHz	Anti-aliasing filter, fc = 3.6 kHz, N=67
2	1.6 kHz	400 - 800 Hz	8 kHz	Anti-image filter, fc= 600 Hz, N=67
3	12 kHz	3.2 - 4 kHz	24 kHz	Combined filter, fc=3.6 Hz, N=99
4	2 kHz	0.6 - 1 kHz	8 kHz	Anti-image filter, fc=800 Hz, N=67
5	3.2 kHz	1.2 – 1.6 kHz	16 kHz	Combined filter, fc=1.4 kHz, N=133

According to the specifications listed in Table 2, students will complete each filter design using MATLAB functions. As an example for converting the sampling rate for segment 1, a simple MATLAB function **firwd()** for FIR filter design given in textbook [1] and MATLAB **freqz()** in the MATLAB signal processing toolbox can be applied as following:

```
» b=firwd(67,1,2*pi*3600/16000,0,4); % obtain filter coefficients
» freqz(b,1,16000,16000); %plot frequency responses
```

The obtained frequency responses of the above designed anti-aliasing filter are displayed in Figure 3. As shown in Figure 4, performance of the pass-band ranging from 0 to 3.2 kHz is satisfied while the attenuation of stop-band ranging from 4 to 8 kHz is 50 dB which satisfies the attenuation requirement of 45 dB. The phase response is linear in pass-band, which is

preferred for audio applications. Other filters could be designed accordingly. Furthermore, the designed filter coefficients are recorded and will be ported to the DSP processor for future real-time implementations in phase 3.

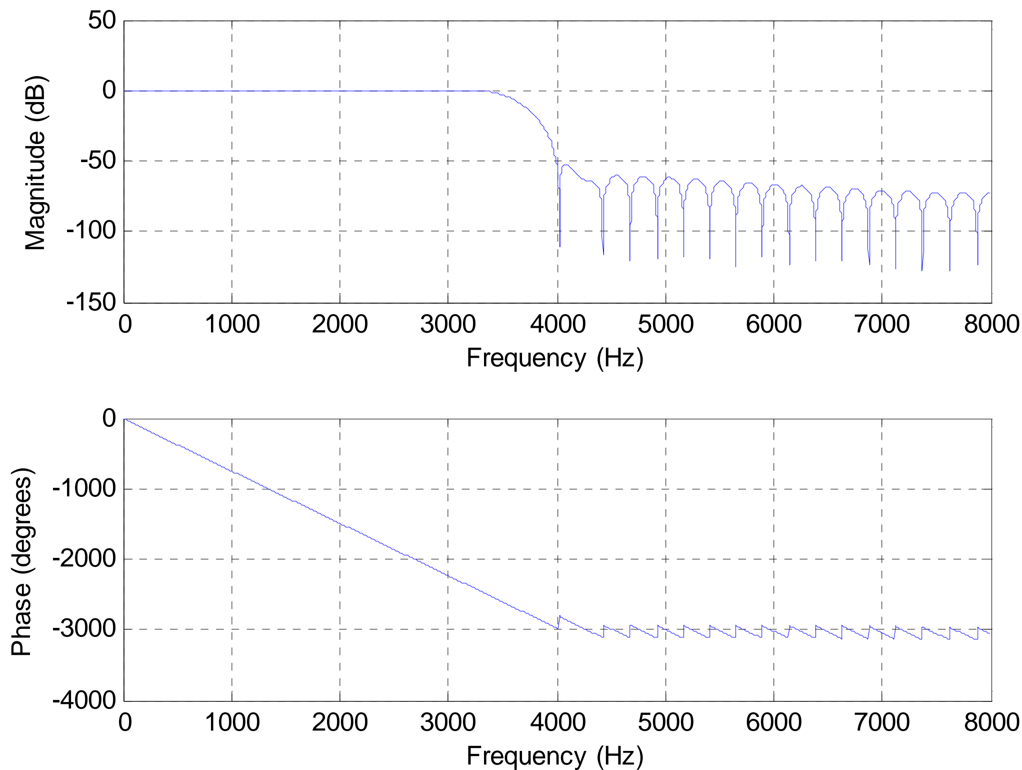

Fig. 3 Frequency responses of the anti-image filter for converting segment 1.

Phase 3 is a stage of real-time implementation and testing. Students will port each of the acquired data segment to an array with 16 bit size per sample (short type), port the designed FIR coefficients (floating-point format), and initialize input buffer (floating-point format) in the header file. As an example, a partial list of the header file for processing segment 1 is shown in Figure 4. Note that the objective in this paper is to show the laboratory pedagogy for teaching sampling rate conversions. Hence, the program listed here is made as more generic as possible and the algorithm is coded in the way that engineering technology students would be able to understand easily and perform coding. Hence, the advance coding of the DSP algorithm using the pointer and circular buffer is not considered in this paper. Although the DSP program here is written for TMS320C671X DSK platform, the idea can be used for other DSP platforms.

```
#define NUM_DATA  1600  // 1600 samples
/* sinusoids with 1000 Hz and 5000 Hz sampled at 16000 Hz */
short xin[NUM_DATA]={
    0,5226,0,2165,8000,2165,0,5226,0,-5226,0,-2165,-8000,-2165,0,-5226,
     0,5226,0,2165,8000,2165,0,5226,0,-5226,0,-2165,-8000,-2165,0,-
5226,
       ...
     (not continuously listed)

/* FIR filter coefficients */
float b[67]={
   0.0004,0.0008,-0.0001,-0.0010,-0.0002,0.0014,0.0008,-0.0018, 0.0019,
      ...
     (not continuously listed)

/* input buffer initialization*/
float x[67]={0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
       ...
    (not continuously listed)
```

Fig. 4 Partial list of data, filter coefficients, and initialized input buffer in the header file.

Next, students will modify the interrupt routine (**AtoD()** shown in Figure 5 for coding their developed DSP algorithms. The sample program for processing segment 1 in CCS [6] is depicted in Figure 5, where for each input data, anti-aliasing filtering is performed; and program sends the last processed output stored in variable **sample** (volatile type) to DAC for every M processed outputs. Note that the program will restart the input data array once data in the array is used up. To keep track of data elements used in the array, variable **cnt,** which is a counter, is incremented each time when loading a new element in the data array and reset to zero after the last element in the data array is processed.

```
interrupt void AtoD()
{
   int i,j;
   float sum, tmp;
   for (j=0;j<M;j++)
   {
     for(i=66; i>0; i--)   // update input buffer
     { x[i]=x[i-1]; }
      x[0]=(float) xin[cnt];  // load new sample
      cnt++;
      sum=0.0;
     for(i=0;i<67;i++)     // FIR filtering
     { sum=sum+x[i]*b[i]; }
     if (j== 0)
     { tmp=sum; }  // update DAC with processed sample (decimation)
     if(cnt==NUM_DATA)
     {  cnt=0;  }      // repeat inputting data segment
   }
   sample= (int) tmp;    // send down-sampled data to DAC
}
```

Fig. 5 Down-sampling conversion by an integer factor of M .

The sample program for implementing up-sampling conversion for segment 2 is shown_in Figure 6. As illustrated in the code, the program sets first $(L-1)$ inputs to be zeros and loads the Lth one from the data array. Next, FIR filtering is operated and the program sends each processed output sample to DAC. Note that variable **Lcount** is incremented and reset to zero to count for $(L-1)$ inserted zeros during the interpolation process. Finally, the output sample is scaled up by a factor of L before it is sent to DAC.

```
interrupt void AtoD()
{
   int i;
   float sum;
   Lcount++;
   for(i=66; i>0; i--)    // update input buffer with zeros
   {
     x[i]=x[i-1];
   }
   x[0]=0;
   if (Lcount==L)
   {
     x[0]=(float) xin[cnt];  // load new sample for every L samples
     cnt++;
     Lcount =0;
   }
   if (cnt==NUM_DATA) //check the data in the buffer
   {
     cnt=0;
    }
   sum=0.0;
   for(i=0;i<67;i++)      // FIR filtering
   {
     sum=sum+x[i]*b[i];
   }
   sum = ((float) L)*sum; // scaled up by a factor L
   sample= (int) sum;    // send the up sampled data to DAC
}
```

Fig. 6 Up-sampling conversion by an integer factor of L.

Figure 7 illustrates an implementation for the case of processing segment 3, which requires up-sampling and down-sampling processes. Note that the combined anti-aliasing and anti-image filter operates at 24 kHz rate. For each sample in the interpolation process, whether it is a data sample from the array or from an inserted zero, the filter continues process M times and keeps one of the M processed samples to achieve the sampling rate $f_s L/M$. As shown in the code, the last sample for every processed M samples is sent to DAC. Variable **Lcount** counts $(L-1)$ inserted zeros in the up-sampling process.

```
interrupt void AtoD()
{
   int i,j;
   float sum;
/* implement interpolation*/
/* process M times (at Lxfs kHz) and keep one to get the M/L kHz
samples*/
 for(j=0;j<M;j++)
 {
   Lcount++;   // count for input the new sample
   for(i=98; i>0; i--)   // update input buffer with zeros
   {
     x[i]=x[i-1];
   }
   x[0]=0;
   if (Lcount==1)
   {
     x[0]=(float) xin[cnt];  // load new sample for every L samples
     cnt++;
   }
   if (Lcount == L)
   {
     Lcount =0;  // ensure load new sample for next sample processing
   }
   if (cnt==NUM_DATA) //check the data in the buffer
   {
     cnt=0;
   }
   sum=0.0;
   for(i=0;i<99;i++)     // FIR filtering
   {
     sum=sum+x[i]*b[i];
   }
   sum = ((float) L)*sum; // scaled up by a factor L
 }
 /* down sampled by a factor of M */
     sample= (int) sum;    // send the up-sampled data to DAC
}
```

Fig. 7 Sampling rate conversion by a non-integer factor of L/M .

Test of converting segments 1, 2, and 3 can be conducted either by connecting the configured DAC line-out to an oscilloscope or connecting the configured DAC voice output to a speaker. Students should also use an oscilloscope to measure the frequency to ensure the accurate sampling rate conversion. For segments 4 and 5, students could listen to each converted voice segment from a speaker and compare their listening to the original voice segment to ensure the sampling rate conversions.

This laboratory project has been used in the advance DSP course in the electrical and computer engineering technology program at Purdue University North Central and DeVry University, Atlanta, Georgia. Students find that the approach is very attractive and stimulating in real-time implementation of sampling rate conversions. The real-time experiment also encourages students to pursue advanced implementations such as

implementing polyphase structures, subband audio coding, ECG (electrocardiograph) signal decimation and processing, and other related applications.

IV. Conclusions

We proposed a simple and efficient laboratory project for teaching real-time signal sampling rate conversions. The laboratory method employs a compromised approach in which the DAC output sampling rate is always fixed while the pre-stored audio data segments with the different sampling rates stored in the processor memory are converted, respectively. Each converted signal can be verified via measurement from its oscilloscope display meanwhile played using a speaker in real time. This method could be adopted as an alternative when the sampling rates cannot be flexibly changed in the low-cost DSP processors, or the DSP processor that has a limitation to set and run different sampling rates for its ADC and DAC devices.

References

[1] Tan, L., Digital Signal Processing: Fundamentals and Applications, Elsevier/Academics Press, 2007.

[2] Li, Z. N., and Drew, M. S., Fundamentals of Multimedia, Prentice Hall, Upper Saddle River, NJ 07458, 2004.

[3] Kehtaranavaz, N., Simsek, B.,C6x-Based Digital Signal Processing, Prentice Hall, Upper Saddle River, New Jersey 07458, 2000.

[4] Texas Instruments, TMS320C6x CPU and Instruction Set Reference Guide, Literature ID# SPRU 189C, Texas Instruments, Dallas, Texas, 1998.

[5] Spectrum Digital, Inc., TMS320C6713 Technical Reference, 2003.

[6] Texas Instruments, Code Composer Studio: Getting Started Guide, Texas Instruments, Dallas, Texas, 2001.