Unit I: Audio Processing

Signal Processors Lab

Markus Quaritsch
markus.quaritsch@tugraz.at

SS 2017
Insitute for Technical Informatics

Typically, DSPs are used in applications in the field of digital audio processing. In this unit, you should implement and evaluate different audio processing algorithms using the C6713 DSP.

1 Software Framework

A framework which configures the audio codec of the DSK and implements a processing chain is provided. Your task is to implement the essential audio processing functions.

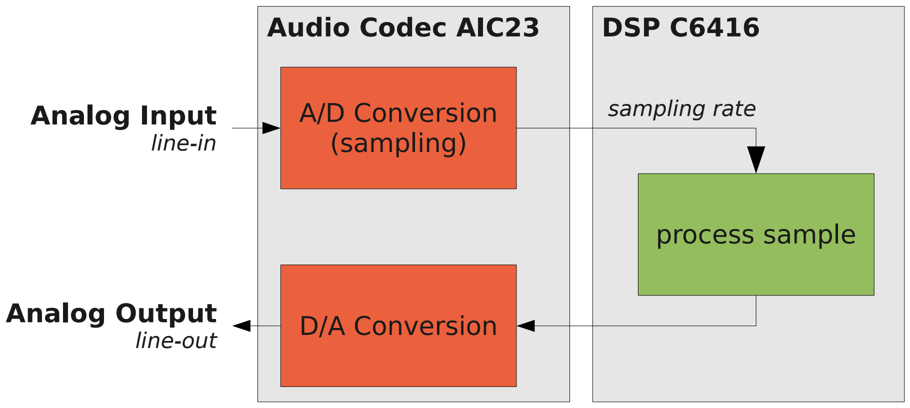


Figure 1: High-level view of the audio processing chain.

Figure 1 shows a high-level view of the processing chain. At a frequency of *sampling rate* (e.g. 8kHz), a 16-bit sample is read from the A/D converter of the audio codec, is processed and the result is written to the D/A converter. In the prepared skeleton this processing chain is implemented using an infinite loop and busy waiting. The core of the infinite loop is implemented similar to the following code snippet:

```
while(1) {
  DSK6713_AIC23_read(hCodec, &indata)     // grab sample
  outdata = c_process_sample(insample)    // process sample using C
  outdata = a_process_sample(insample)    // process sample using Asm
  DSK6713_AIC23_write(hCodec, outdata)    // output result
}
```

The functions `c_process_sample()` and `a_process_sample()` which process the input sample should be replaced by a call to your implementations in C and assembler, respectively. Either C or assembler should be activated, not both of them. You can define new processing functions to have the implementations of all tasks in your source at the same time. However, it is important that you do not change the list of arguments of your filter function. Use `static` variables to store preceding samples directly in the filter function. *Hint: The filter function is called for each sample separately, e.g. 8000 times in a second for a sampling rate of* $8kHz$.

2 Evaluation

Evaluation Equipment

DSK 6713: The DSK 6713 *DSP Starter Kit* is a DSP evaluation board, which integrates a Texas Instruments TMS320C6713 DSP, external memory, an audio codec, a JTAG emulator, peripheral interfaces, etc. on a single board. For evaluation, the DSP is used to run the audio processing algorithms, audio data is transferred via the input (A/D) and output (D/A) lines of the integrated AIC23 stereo audio codec. See chapter 2.2 and 3.4 of the DSK technical reference for details.

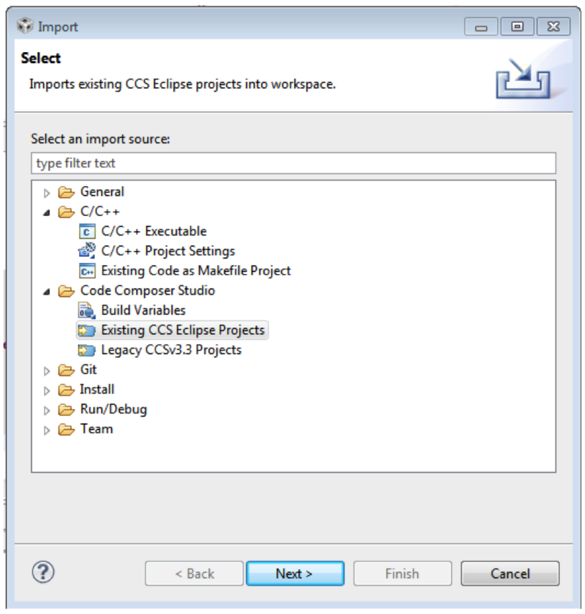
Headphones: In this lab headphones can be used for making the audio signals hearable and therefore interpretable in a subjective manner. **Take care of the volume!**

***Audacity* Audio Processing Software:** Audacity is an open-source audio processing software providing a huge range of functionality. Some features which are very useful for audio analysis of this lab are: (1) loading of audio waveform files in different formats, (2) looped playback, (3) audio recording, (4) simultaneous playback and recording, (5) spectrum plotting. The software is rather easy to use and the GUI is self-explaining. See the integrated help for more information.

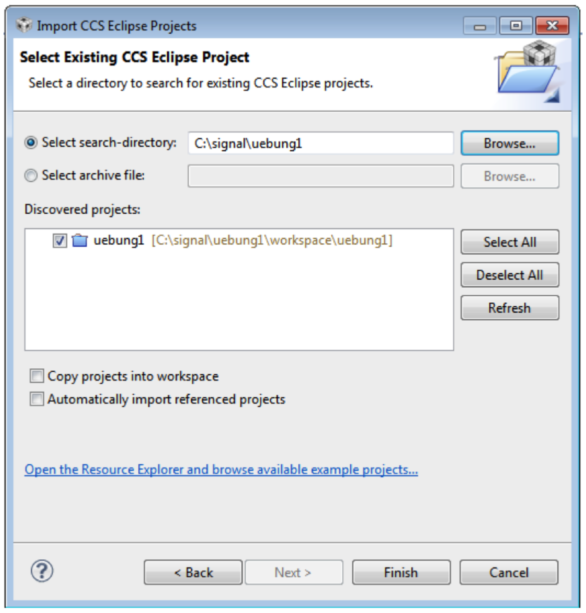
***SignalGen* Signal Generator Software:** SignalGen is a realtime signal generator software which uses the audio equipment of the workstation to output the signals. It supports the generation of sine, rectangular and triangular signals with online adaptable frequency and amplitude. You have to press the *unmute* checkbox in order to produce an output signal!

Project Skeleton

On the network share you find a skeleton for every assignment. The skeleton already contains basic functionality such as reding the input samples from the audio codec, writing the output samples to the audio codec etc. Your job is to fill this skeleton with meat and implement the functionality described in Section 3.



(a) Import existing project


Import CCS Eclipse Projects
Select Existing CCS Eclipse Project
Select a directory to search for existing CCS Eclipse projects.
Browse...
Browse...
Select All
Deselect All
Refresh
Open the Resource Explorer and browse available example projects...
< Back
Next >
Finish
Cancel

Select search-directory: C:\signal\uebung1
Select archive file:
Discovered projects:
☑ uebung1 [C:\signal\uebung1\workspace\uebung1]
☐ Copy projects into workspace
☐ Automatically import referenced projects

(b) Select project for import

Figure 2: Importing skeleton

In order to import the code skeleton into your workspace proceed as follows:

1. In the virtualized system mount the network drives. On the desktop you find a Batch-file named 'logon.bat'. Start this script and log in with your TU Graz credentials.
   Important: disconnect the network drives before shutting down the virtual machine via the 'logout.bat' script on the Desktop

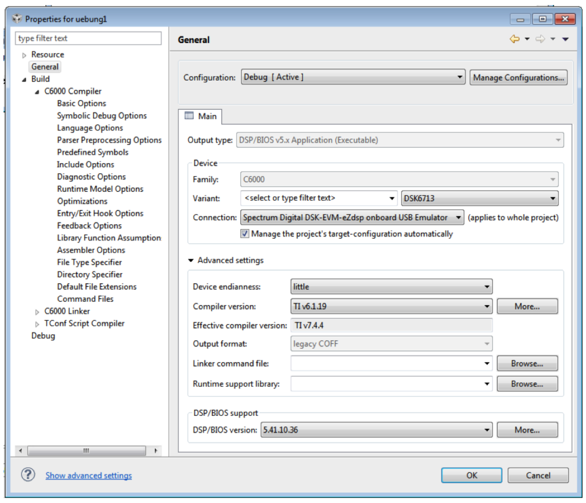
2. The code skeleton is located in the folder `M:\ITI\LABOR\LV448053 Signalprozessoren`

3. In the virtualized system launch the batch-file `uebung1.cmd` in the folder mentioned above. This will copy the sources from the shared network drive to your local drive `C:\signal`.

4. In Code Composer Studio go to **Project** → **Import...** and select **Existing CCS Project** (see Figure 2a).

5. In the following dialog choose **C:\signal** as search-directory and select the desired existing project from the list of discovered projects (see Figure 2b). Make sure to tick the checkbox **Copy projects into workspace**.

6. After importing the project make sure to select the correct target (`DSK6713`) and correct connection type (`Spectrum Digital DSK-EVM-eZdsp onboard USB Emulator`) – see Figure 3. Also select the most recent compiler version and DSP/BIOS version.


Properties for uebung1

General

Configuration: Debug [Active]

Output type: DSP/BIOS v5.x Application (Executable)

Device
Family: C6000
Variant: <select or type filter text> DSK6713
Connection: Spectrum Digital DSK-EVM-eZdsp onboard USB Emulator (applies to whole project)
[✓] Manage the project's target-configuration automatically

Advanced settings
Device endianness: little
Compiler version: TI v6.1.19
Effective compiler version: TI v7.4.4
Output format: legacy COFF
Linker command file:
Runtime support library:

DSP/BIOS support
DSP/BIOS version: 5.41.10.36

Figure 3: Project properties

Evaluation Set-Up

The hardware and software equipment as introduced above is used to evaluate audio processing algorithms on the DSP. In general, three steps are involved: (1) Generation of audio signals. (2) Processing of signals. (3) Analysis of the processed signals.

Different set-ups using the provided equipment are possible, but the following proved as very functional when preparing the course material. The DSK is connected via *line-in* and *line-out* to the *line-out* and *line-in* of the audio codec of the workstation, respectively. **Note: Use the line-in/line-out connectors at the back-side of the workstation.** This allows to generate audio signals using any audio software (Audacity, SignalGen), and allows to record the processed signals directly using Audacity. Furthermore, by connecting the headphones to the *headphone* connector of the DSK, the processed signal can be evaluated acoustically in parallel. Figure 4 shows the described set-up. **Note: Run Audacity and the signal generator application on the host system, *not* the virtualized system! Therefore you may need to mount the network drives on the host system as well.**

Instead of using audio recording software for visualizing the processed signals, an oscilloscope can be used. Furthermore, stand-alone signal generators are available to generate the input signals. **When using the signal generator, take care of the maximum amplitude of the signal!**

Hint: The evaluation set-up from the workstation to the DSK and back to the workstation involves different hardware devices, drivers and tools. It is advisable to check each module and device involved step by step for faulty configuration before starting the evaluation. Some important things to check:

- Connection DSK $\leftrightarrow$ workstation: Are the correct jacks and plugs connected?
- On the workstation: Are line-in/line-out unmuted and is the volume set appropriately?
- On the workstation: Is the local echo of line-in deactivated to avoid backcoupling?

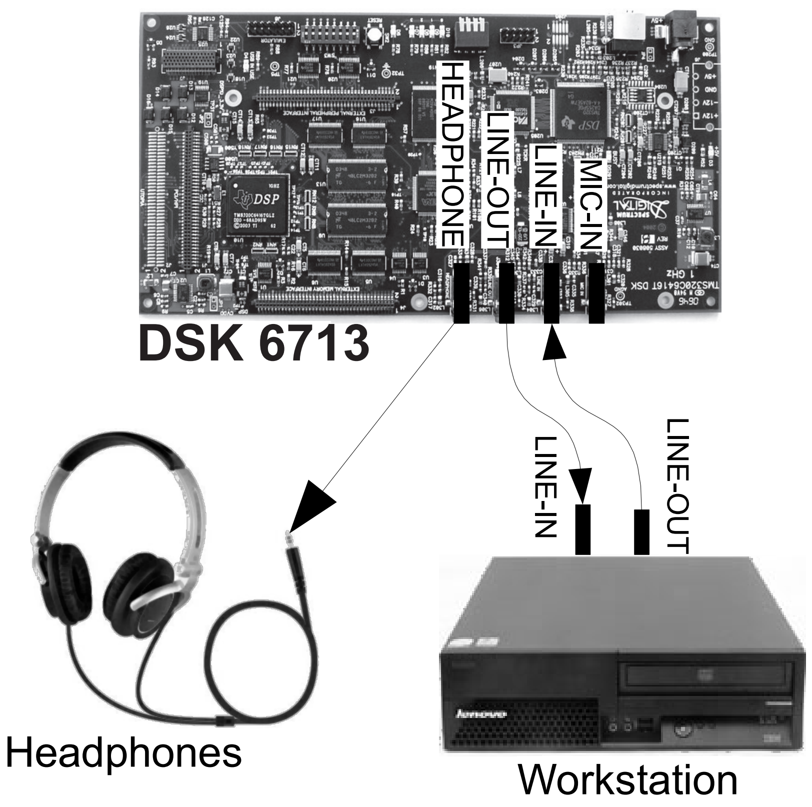


Figure 4: Set-up for evaluating digital audio algorithms using the DSK6713

3 Tasks

Task 1 – Amplifier: Develop an amplifier to adapt the gain of the input signal.

1. Implement the amplifier in C and assembler. Try different gains, at least 0.5, 1.33, 5.0 and 10.0.
2. Use music and a sine as an input, and your headphones and Audacity for analysis.

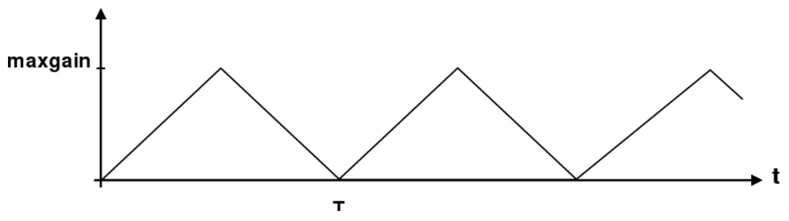
Task 2 – Fixed-Point Arithmetic: Although the C6713 supports floating-point operations, this is not true for the majority of DSPs. Hence, in this task we avoid floating-point operations and implement our solution using fixed-point arithmetic.

1. Implement the Amplifier from the previous task using only fixed-point arithmetic (both, in C and assembler). You need to change the signature of the `process_sample` function to take a `short` as argument and return a `short`: add a new function with the mentioned signature). In your implementation use only multiplication and shift operations. Divisions should be realized by using multiplication and shifting. Implement different gains, at least 0.5, 1.33, and 5.0 (error $\leq 0.1\,\%$, e.g., 1.33 ± 0.00133).

Task 3 – Basic Filters: Implement filter functions, which adapts the audio signals in various ways. Implement all filters in both, C and assembler language. Test your implementations using a well-defined signal, e.g., a sine wave, rectangle etc.

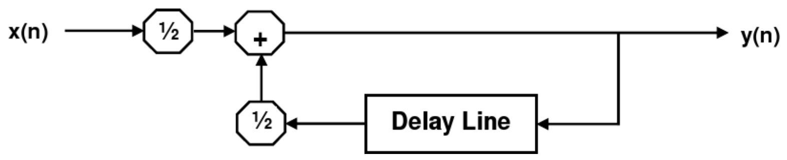
1. Clipping: Clip values greater and smaller than ± 5000.
2. Half-wave rectification: Cut off the negative half-wave of the signal.
3. Full-wave rectification: Convert both polarities of the signal to be positive.
4. Differentiate the signals: $out(n) = f_n - f_{n-1}$. For evaluation use the signal generator (sine) and vary the frequency of the sine or a triangular signal.

Task 4 – Amplitude modulation: Develop a function which modulates the input signal onto a isosceles triangle function. The maximal gain *maxgain* must be reached at the peak of the triangle. Check the proper function of your code by using Audacity, breakpoints, and the watch window.



1. Implement the filter function in C.
2. Use various periods: $\tau \in \{500, 1000, 4000, 8000, 16000\}$ sampling cycles
3. Use Audacity to visualize the modulated audio signal.

Task 5 – Digital reverberation: Develop a *Digital Reverberation* filter by using a circular buffer. The filter is defined as follows:

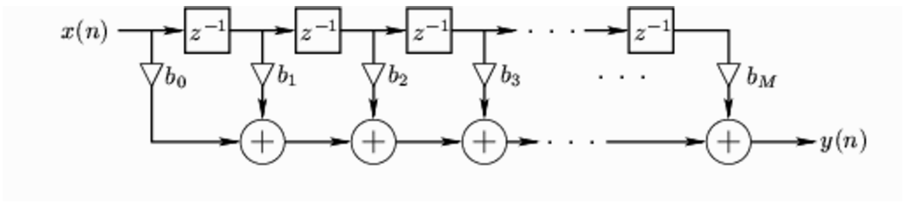


1. Implement the filter function in C in a simple and efficient way.
2. Intialize the buffer values with 0 before the algorithm starts.
3. Evaluate the filter by using different delay periods: $T = \{1000, 2000, 4000, 8000\}$ samples. Try different input frequencies, e.g., by using music as a source.

Task 6 – FIR Filter: Develop a function realising the *Finite Impulse Response* (FIR). The FIR filter is based on convolution, whereby the audio signal $x(k)$ is convoled by pre-calculated filter coefficients $b(k)$. Examples for FIR filters are low-pass, high-pass and band-pass filters.

$$y(n) = \sum_{k=0}^{N-1} b(k) \cdot x(n-k)$$

The coeffients of various filter functions $b(k)$ needed for this task are pre-calculated and stored as arrays in the header file *filter.h* (designed for 8kHz sampling rate). The coefficients are stored in arrays of floats. Note: for demonstration purpose and for better understanding the order of the filters in this task is very high. In real life usually a rather small magnitude, e.g. 2, is sufficient.



1. Implement a FIR function by using a circular buffer with depth N, using C language.
2. Intialize the buffer values with 0 before the algorithm starts.
3. Use the filter coefficients from file *filter.h* for applying different filters to the input samples. Test the proper function by varying the input frequency. Use the signal generator and music as source signal.
4. What characteristics of the filter are changed, when the sampling rate is changed?
5. Measure the signal delay between input and output.
6. Measure the execution time for one output sample using the Simple Clock Profile (Run $\rightarrow$ Clock)
7. **Optional Task:** convert coefficient to int at runtime to improve runtime

4 Questionaire

Task 1 & 2

(1) What problem occurs if a loud signal is amplified even more? Show the resulting signal with a sine wave as input. How can this problem be solved?

(2) How is the *Total Harmonic Distortion* (Klirrfaktor) influenced by high gains (subjective feeling + explanation)?

(3) How can the multiplication by a floating-point constant, e.g. 1.75, be realised on a fixed-point processor such as the C64x? What is the accuracy and the resolution of this method?

Task 3

In which way is the output affected, when the signal is differentiated? . . . .

Task 4

Show the modulated output signal using a sine-wave as input.

Task 5

Runtime for processing one sample: . . . . *cycles*

Task 6

(1) What happens, when the sampling rate is multiplied by 0.5 and 2, respectively? Which characteristics of the filters change?

(2) Runtime of a FIR function of order 256: . . . . *cycles*

(3) Signal delay of a FIR function of order 128:

delay t= . . . . *ns* | sine freq f= . . . . kHz | filter coeffs= . . . .