EE 396: DESIGN LAB
PROJECT REPORT

VibeShift : Real-Time Audio Effects Using the TI C6416T DSP

Under the Supervision of:

Prof. Srinivasan Krishnaswamy

By:

Bhavik Jain
Roll No.: 220102023
Manikanta Krishnamurty
Roll No.: 220102059

Department of Electronics and Electrical Engineering
Indian Institute of Technology
Guwahati, Assam, India
April, 2025

Abstract

This report presents the implementation of various voice effects using the TMS320C6416T DSP board and Code Composer Studio. The effects include robotic voice (WALL-E Effect), radiophone voice, a dark phonk-style audio effect and a bonus effect of 3D Audio. These were achieved by applying different signal processing techniques such as amplitude modulation and filtering on real-time audio inputs. The project demonstrates the capabilities of real-time audio manipulation on a DSP platform.

1 Introduction

Real-time audio processing is a key application of digital signal processors (DSPs), especially in embedded audio systems. This project aims to explore creative voice effects by utilizing the TMS320C6416T DSP board. The goal was to generate distinctive voice effects in real-time, showcasing both the performance and flexibility of DSP-based processing.

2 Objectives

- Implement a robotic voice effect using modulation and bit crushing..
- Simulate a radio voice using bandpass filtering and noise overlay.
- Produce a phonk effect involving rhythmic chopping, reverb and tremolo.
- Create a 3D audio spatialization effect using basic audio controls.

3 Hardware and Software Tools

The following table lists all the components required for this project.

Component Name	**Quantity**
TMS320C6416T DSP Starter Kit	1
3.5 mm Male-to-Male TRS Audio Cable	2
Headphones	1
Audio player	1

Table 1: List of components required for the project.

The programming was done on **Code Composer Studio v5** using **C Programming Language**.

4 Description of the TMS320C6416T Board

The TMS320C6416T is a high-performance DSP development board from Texas Instruments, widely used for real-time signal processing tasks. It features a powerful VLIW architecture capable of executing multiple instructions per cycle, making it ideal for demanding audio processing applications.

Key features that make the TMS320C6416T well-suited for audio processing include:

- High clock speed and parallel processing capabilities for real-time audio computation.
- On-board 24-bit stereo audio codec for high-quality audio input and output.
- Multiple memory interfaces and expansion options for handling large audio data streams.
- Comprehensive development tools and debugging support for efficient implementation of audio algorithms.

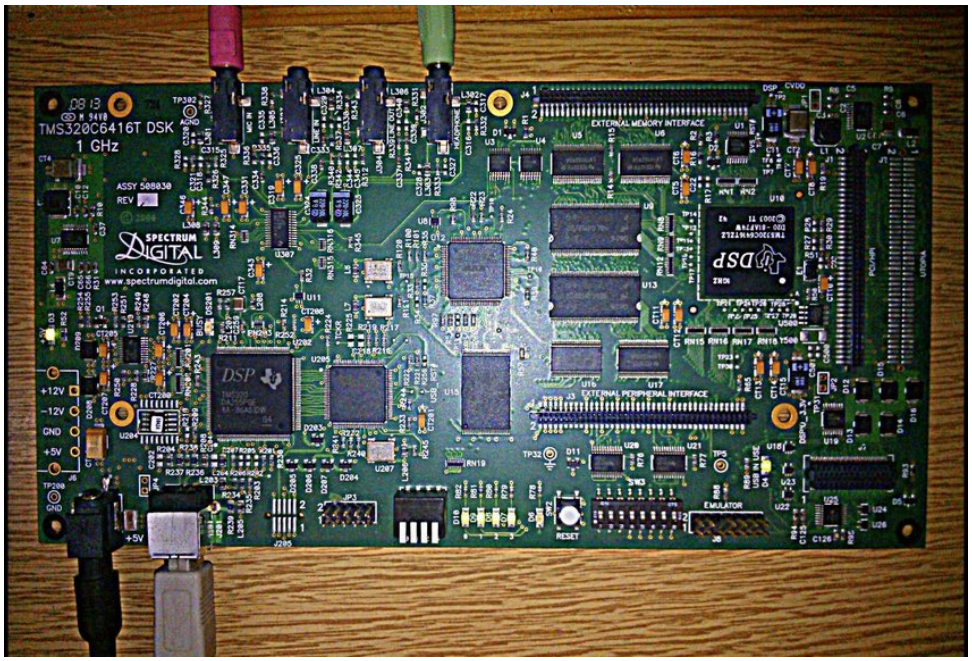

Figure 1: TI's TMS320C6416T DSK BOARD

5 Audio Codec on the TMS320C6416T Board

The TMS320C6416T board integrates the **TLV320AIC23B** audio codec, a critical component for high-fidelity audio input and output in real-time processing applications. This 24-bit stereo codec combines a high-performance analog front-end with flexible digital interfaces, making it ideal for speech, music, and acoustic signal processing.

Key Features

- **High-Resolution Conversion**:
  - ADC (Analog-to-Digital Converter): 24-bit resolution, 8–96 kHz sampling rate.
  - DAC (Digital-to-Analog Converter): 24-bit resolution, 8–96 kHz sampling rate.
  - Supports common audio standards (e.g., 44.1 kHz for CD-quality audio).
- **Signal Quality**:
  - ADC SNR: 90 dBA (minimizes noise in recordings).
  - DAC SNR: 100 dBA (ensures clean audio output).
  - Integrated programmable gain amplifiers (microphone/line input).
- **Interfacing**:
  - Connects to the DSP via McBSP (Multi-Channel Buffered Serial Port) for low-latency data transfer.
  - Supports I^2C or SPI control for configuring sampling rates, gains, and filters.
- **Physical Connectivity**:
  - Dedicated 3.5 mm jacks for microphone input, line input, line output, and headphone output.
  - On-board anti-aliasing and reconstruction filters.

The TLV320AIC23B's programmability allows fine-tuning for specific audio tasks, such as noise reduction or spectral analysis, while its direct integration with the DSP enables real-time processing of audio streams. This combination of precision and flexibility makes it a cornerstone of audio-centric applications on the C6416T platform.

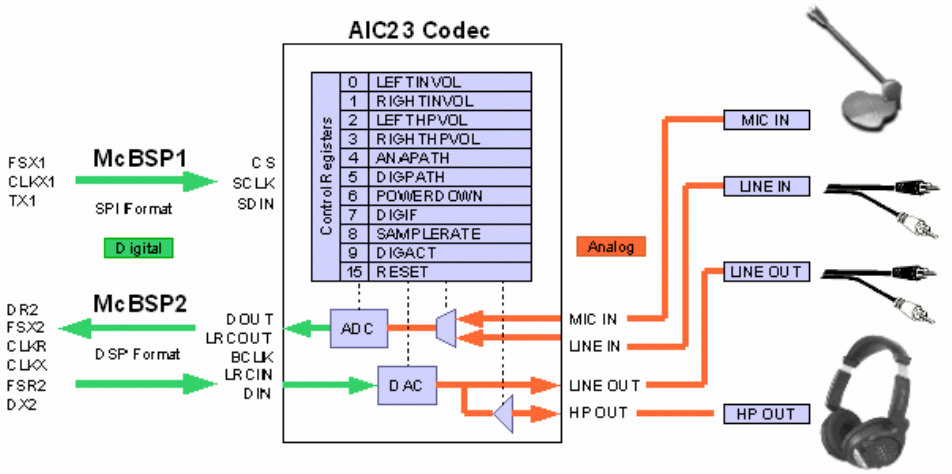

Figure 2: AIC23B CODEC

6 Methodology

This section involves the design followed in order to create the effects in real time.

6.1 WALL-E: Robotic Voice

To emulate the iconic **WALL-E** voice, three audio processing techniques were applied in sequence. Each step was selected to target a distinct auditory characteristic of the original voice—namely its mechanical tonality, lo-fi texture, and subdued expressiveness.

1. **Ring Modulation**
   A 200 Hz sine wave carrier was used to modulate the input voice signal. Ring modulation eliminates the original frequency components and replaces them with *sum and difference frequencies*. This technique introduces inharmonic sidebands that create a **metallic and robotic** quality, which is central to WALL-E's synthetic voice. The 200 Hz carrier was chosen specifically to preserve some low-frequency intelligibility while still pushing the voice into an artificial timbral space. The operation is represented as:

   $$y(t) = x(t) \cdot \cos(2\pi \cdot 200\, t)$$

   where $x(t)$ is the input signal. The resulting signal has no spectral overlap with the original, ensuring complete timbral transformation.

2. **Bit Crushing with TPDF Dithering (5-bit)**
   To simulate the limited resolution of low-fidelity digital systems, the ring-modulated signal was quantized to 5 bits, yielding only 32 discrete amplitude levels. However, direct quantization without any additional processing can introduce *correlated quantization noise*, especially when the signal amplitude is low. This results in distortion

that manifests itself as **static tonal artifacts**—undesirable characteristics that sound unnatural and repetitive.

To mitigate this, **triangular probability density function (TPDF) dither** was added before quantization. Dithering works by introducing low-level, deliberately randomized noise to the signal prior to quantization, which serves to *decorrelate* the quantization error from the input signal. TPDF dithering, in particular, is widely regarded as one of the most effective methods as it produces a noise profile that is both statistically unbiased and spectrally neutral.

TPDF dithering makes the resulting bit-crushed signal **perceptually smoother**. This helps preserve intelligibility and expressiveness while still maintaining the **crunchy, digital character** appropriate for the WALL-E aesthetic.

3. **3rd Order IIR Low-Pass Filter**
   To attenuate high-frequency artifacts resulting from ring modulation and bit crushing, a 3rd-order Infinite Impulse Response (IIR) low-pass filter was applied. The filter is defined by the following set of coefficients:

   - **Numerator:** $\{0.0181,\ 0.0543,\ 0.0543,\ 0.0181\}$
   - **Denominator:** $\{1.0000,\ -1.7600,\ 1.1829,\ -0.2781\}$

   This filter structure corresponds to a *Direct Form I implementation* of a causal IIR system. The coefficients implement a Butterworth-like low-pass filter with a smooth roll-off and no ripples in the passband, favoring a natural decay over abrupt cutoff. The low-pass characteristics were carefully selected to preserve low-frequency speech features while suppressing high-frequency noise introduced by earlier processing stages. It ensures that the final audio output remains **intelligible, tonally balanced, and consistent with the WALL-E aesthetic**.

6.1.1 Filter analysis

The frequency response of the filter (see figure below) reveals a smooth low-pass behavior with a cutoff frequency around 4804.7 Hz (–3 dB point in linear magnitude). This value was determined using MATLAB and marks the frequency where the output magnitude begins to significantly attenuate.

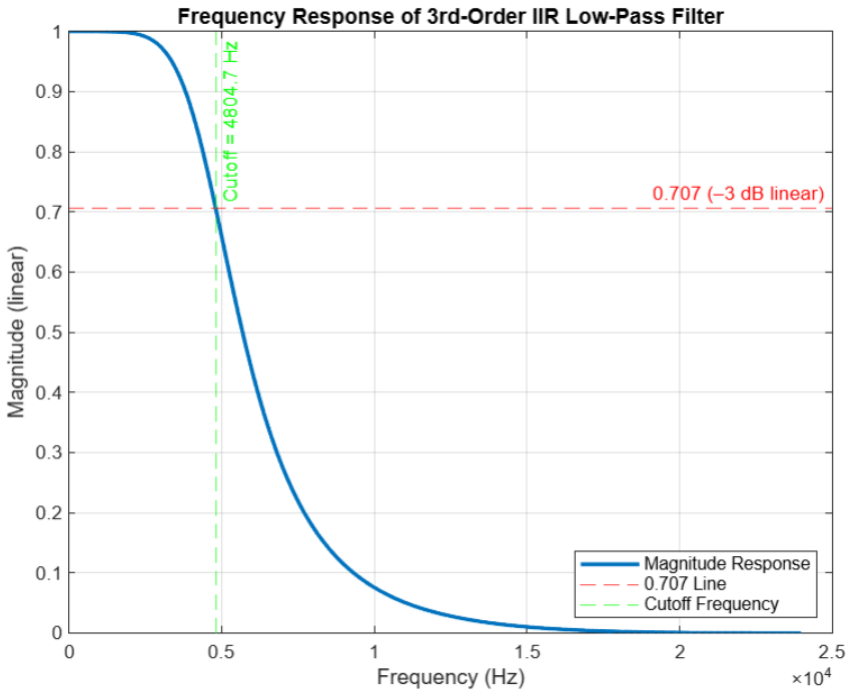

Figure 3: Frequency response of the filter used (Sampling frequency: 48 Khz)

This cutoff is a well-balanced choice for the following reasons:

- The majority of speech intelligibility lies within the 300 Hz to 4000 Hz range. By setting the cutoff just above this range, the filter allows the essential phonetic components (like formants and consonant clarity) to pass through, keeping the voice recognizable and clear.
- Both the ring modulator and bit crusher introduce high-frequency content—sidebands from modulation and sharp transitions from low-resolution quantization. The filter effectively attenuates these unwanted aliasing and distortion components above the useful range, contributing to a smoother and more pleasant sound.
- WALL·E's voice is robotic but also soft and warm. A lower cutoff frequency would risk making the voice too muffled, while a much higher one would let harsh digital noise through. The 4.8 kHz cutoff acts as a compromise between preserving clarity and maintaining a stylized, warm tone.

Thus, while the 4804.7 Hz cutoff may not be universally optimal, it is a contextually effective choice for achieving a robotic voice that feels synthetic, yet still articulate and emotionally expressive.

6.2 Minimalist Dark Phonk Voice Effect

This section presents the design and implementation of a **minimalist dark phonk voice effect**, built to emulate the sonic traits of lo-fi, moody vocals characteristic of the dark

phonk genre. The processing chain consists of three main stages: a low-pass filter, a tremolo modulator, and a short echo. These blocks are implemented efficiently to allow real-time performance while achieving the desired aesthetic.

6.2.1 Low-Pass Filtering

The first stage is a second-order Infinite Impulse Response (IIR) low-pass filter defined by the following coefficients:

$$b = \{0.0675,\ 0.1349,\ 0.0675\}$$
$$a = \{1.0,\ -1.1430,\ 0.4128\}$$

This filter yields a cutoff frequency approximately in the 4.5–5 kHz range, depending on the sampling rate. It attenuates high-frequency components while preserving the low- and mid-range vocal content, producing a warm, analog-like tone.

Why this filter?

- **Appropriate order**: A 2nd-order filter gives you 12 dB/octave attenuation beyond the cutoff. That's already enough to darken the signal and reduce high-frequency content significantly. A 3rd-order filter would give 18 dB/octave, which might be too aggressive, removing too much high-frequency detail and making the voice sound muffled or lifeless rather than dark and vibey.
- **Tonal shaping:** The filter removes sharp transients, typical of modern digital audio, to recreate the subdued vocal texture found in dark phonk.
- **Lo-fi aesthetic:** The upper harmonics are deliberately suppressed to achieve a vintage,"tape-recorded" sound.
- **Post-processing smoothing:** It also acts as a final-stage smoother, helping tame harsh edges introduced by modulation or quantization artifacts.

The frequency response of this filter is also attached below:

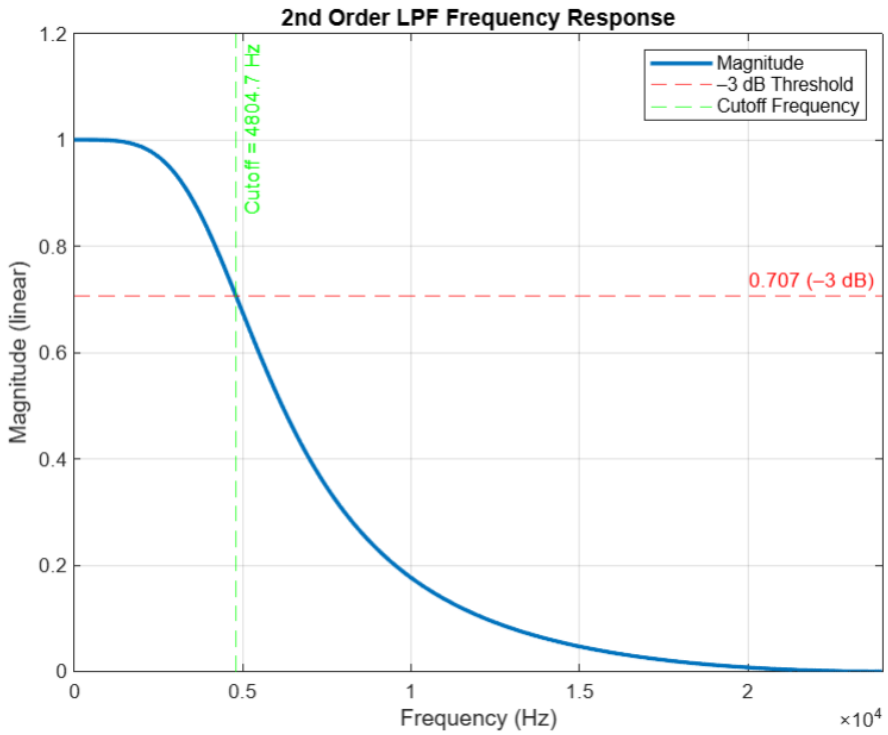

Figure 4: Frequency response of filter used

6.2.2 Tremolo Modulation (6 Hz)

Following filtering, the signal undergoes tremolo modulation — amplitude variation using a sinusoidal Low Frequency Oscillator (LFO). The modulation equation is:

$$y(t) = x(t) \cdot \left(\frac{1 + d \cdot \sin(2\pi \cdot 6 \cdot t)}{2} \right)$$

where d is the modulation depth and 6 Hz is the chosen tremolo rate.

Why 6 Hz?

- **Rhythmic enhancement:** At 6 Hz, the tremolo adds a perceptible but smooth pulsation that complements the beat without distracting from the lyrics.
- **Psychoacoustic impact:** This rate is musically effective for evoking a hypnotic, throbbing quality — ideal for slow, dark phonk instrumentals.
- **Analog character:** It mimics the flutter and instability of analog tape, reinforcing the lo-fi aesthetic.

6.2.3 Short Echo (Feedforward Delay)

The final stage is a short echo, implemented using a simple feedforward delay line with a time of 0.05 s and a decay factor of 0.5:

$$y[n] = x[n] + 0.5 \cdot x[n - D]$$

where $D = 2400$ samples at a 48 kHz sampling rate.

Why short echo?

- **Spatial depth:** Adds ambience without overlapping the dry signal too much, giving a subtle feeling of space and separation.
- **Vintage slapback:** Emulates slapback delay found in early analog recordings.
- **Retains intelligibility:** Unlike reverb, this does not overly diffuse the transients, preserving clarity in vocal articulation.

6.2.4 Combined Effect

The combined chain – low-pass → tremolo → echo – yields a dark, spacey, and rhythmically dynamic voice effect. The processing evokes the classic stylistic cues of dark phonk: *low-fidelity warmth*, *rhythmic modulation*, and *spatial ambience*. It is minimalist by design, requiring minimal computation, yet highly expressive and musically effective.

> This approach prioritizes emotional tone and genre authenticity over complexity, making it ideal for embedded or real-time DSP applications in music production.

6.3 Radio Voice Effect

To replicate the distinct, gritty, and narrowband quality of traditional radio communication, we developed a **Radio Voice** processing chain that emulates the analog characteristics of AM transmission. The overall methodology involves three core processing stages: *bandpass filtering*, *nonlinear saturation (distortion)*, and *dynamic noise gating*.

6.3.1 Bandpass Filtering with Biquad Sections

The defining tonal feature of radio speech lies in its limited bandwidth. To simulate this, we designed a narrow **bandpass filter** centered around the human speech formants. The filter used here has a sharper Q-factor to emulate the hollow, nasal timbre of radio voices. Its **magnitude response**, shown in Fig. 6, demonstrates clear roll-off outside the passband, with the lower and upper cutoff frequencies at approximately **8664.2 Hz** and **10476.6 Hz**, respectively.

Why Biquad Filters Were Used: Biquad (bi-quadratic) filters are second-order IIR filters that offer an excellent trade-off between performance and computational efficiency. They are especially well-suited for embedded DSP applications like ours, where

real-time processing is required on resource-constrained hardware. By implementing the bandpass filter as a *cascade of second-order sections (SOS)*, we achieve:

- **Stability**: Each SOS stage is more numerically stable than a higher-order filter implemented directly.
- **Precision**: Biquads allow precise placement of poles and zeros to sculpt the frequency response tightly around our target band.
- **Efficiency**: The computation per sample is low, which is ideal for real-time processing on the TMS320C6416 DSP.

This biquad-based bandpass filter forms the backbone of the radio effect, carving out the essential midrange frequencies while suppressing lows and highs that would otherwise add fullness or brightness to the signal — qualities that are undesirable for this effect.

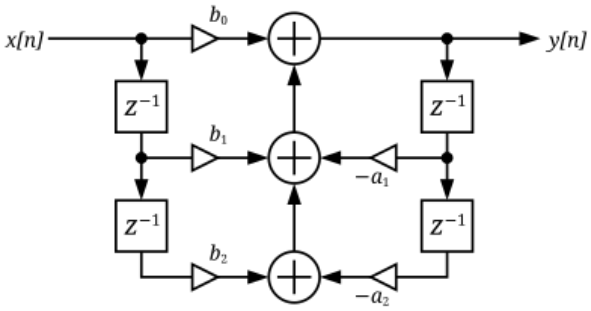

Figure 5: Digital Biquad Filter

6.3.2 Distortion for Grit and Presence

To further enhance the *character* of the signal and replicate the harmonic saturation that arises in analog radio transmitters, the filtered output is passed through a **nonlinear distortion** stage. A hyperbolic tangent function provides soft clipping and gentle saturation, introducing odd-order harmonics that add a gritty texture.

The distortion is modeled using the following equation:

$$y[n] = \tanh\left(G \cdot x[n]\right) \tag{1}$$

where $x[n]$ is the input sample, G is the gain factor (set to 3.0), and $y[n]$ is the saturated output.

This stage is crucial for imparting the perceived *presence* and *edge* often associated with walkie-talkie or vintage radio communication.

6.3.3 Noise Gate for Clean Silence Handling

Speech through radio is often punctuated by bursts of static or abrupt silences. To simulate this behavior and suppress low-level background noise, we implemented a **dynamic**

noise gate using an envelope follower with hysteresis.

First, the envelope $e[n]$ is tracked with separate attack and release rates:

$$e[n] = \begin{cases} e[n-1] + \alpha_a(|x[n]| - e[n-1]) & \text{if } |x[n]| > e[n-1] \\ e[n-1] + \alpha_r(|x[n]| - e[n-1]) & \text{otherwise} \end{cases} \tag{2}$$

where $\alpha_a = 0.1$ (fast attack) and $\alpha_r = 0.01$ (slow release).

Then, the gate applies attenuation based on the envelope level:

$$y[n] = \begin{cases} x[n] & \text{if } e[n] > T_{\text{high}} \\ 0.5 \cdot x[n] & \text{if } T_{\text{low}} < e[n] \leq T_{\text{high}} \\ 0 & \text{if } e[n] \leq T_{\text{low}} \end{cases} \tag{3}$$

where $T_{\text{high}} = 0.02$ and $T_{\text{low}} = 0.015$ introduce hysteresis to prevent chattering.

This technique helps suppress environmental noise or microphone hiss when the speaker is silent, just like squelch control in analog radios.

6.3.4 Additional Enhancements

After passing through the bandpass filter, distortion, and noise gate, the output is **soft-limited** to a maximum amplitude of ± 0.95 to avoid clipping and ensure DAC safety. A gain boost factor is applied post-filtering to compensate for attenuation introduced by the bandpass response.

This full pipeline delivers a voice effect that is:

- **Narrowband and nasal**
- **Slightly distorted**
- **Noise-suppressed with gated silences**
- **Harmonically rich and vintage-sounding**

The result convincingly reproduces the aesthetic of **radio communication**, making it ideal for use in retro-futuristic sound design or stylized storytelling.

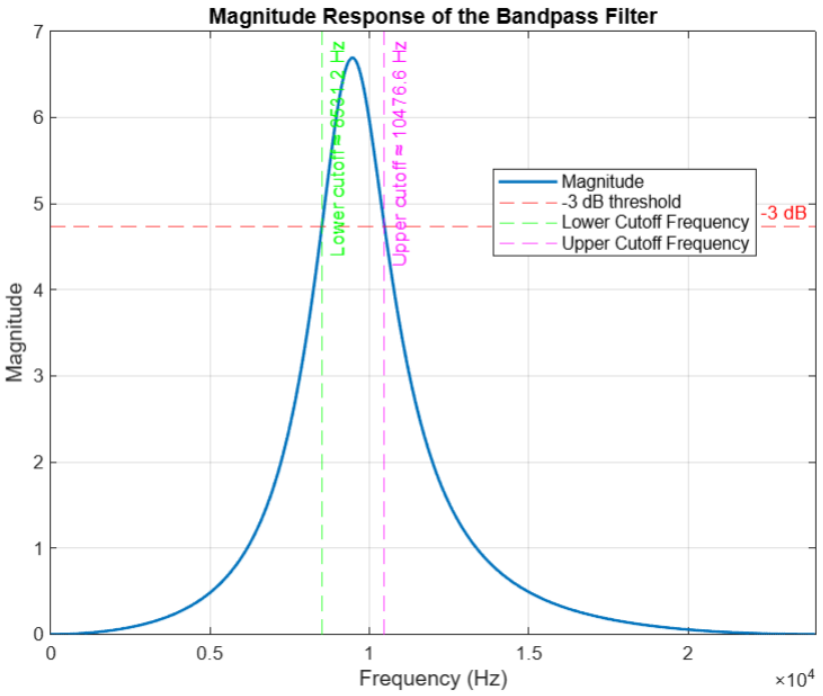

Figure 6: Magnitude response of the bandpass filter used in the Radio Voice effect. Cutoff frequencies are annotated at 8664.2 Hz and 10476.6 Hz.

6.4 3D Audio Effect

The **3D Audio** effect was designed to simulate spatial motion of sound between the left and right audio channels, creating a sense of immersion and movement. This was achieved by dynamically modulating the stereo volume levels over time, giving the impression that the audio source is moving in a circular or oscillatory pattern around the listener.

6.4.1 Methodology

To simulate the panning motion, the system gradually varies the left and right channel headphone volumes in opposite directions. The volume of the left channel is increased or decreased linearly within a predefined range, while the right channel mirrors this adjustment in the opposite direction. As a result, the perceived position of the sound continuously moves from left to right and back.

The modulation occurs periodically using a counter-based timing mechanism. Every fixed number of samples (defined by `ramp_interval`), the volume levels are recalculated and updated. The core idea is as follows:

- Maintain a midpoint volume: $V_{\text{mid}} = \frac{V_{\max}+V_{\min}}{2}$
- At each step, increment or decrement the left volume by one unit.
- Compute the right volume using $V_{\text{right}} = V_{\text{mid}} - (V_{\text{left}} - V_{\text{mid}})$

- When V_{left} reaches its bounds, the direction of increment reverses.

This approach creates a volume-based panning illusion without introducing interaural time differences (ITD), focusing purely on intensity-based localization. The effect loops infinitely, offering a smooth oscillating pan between the ears.

6.4.2 Implementation Details

- **Platform:** TMS320C6416 DSK
- **Codec:** TLV320AIC23 configured in stereo mode at 48 kHz sampling rate
- **Control:** Volume registers (`0x02` and `0x03`) updated in real-time
- **Range:** Volume values were cycled between `V_MIN` and `V_MAX`

The left channel amplitude followed a sawtooth pattern, while the right channel simultaneously followed a complementary inverse pattern. This ensured continuous smooth motion across the stereo field.

6.4.3 Resulting Effect

The outcome is a "3D" auditory illusion where the listener perceives the sound as oscillating from one side of the head to the other. This effect is often used in binaural mixing or VR audio to simulate head motion or environmental movement. Though relatively simple in its approach, this dynamic stereo panning effectively adds dimensionality to mono input signals.

7 Implementation

This section outlines the hardware and software setup used to implement the described audio effects. The target hardware for this project is the Texas Instruments TMS320C6416 DSP Starter Kit (DSK), interfaced with the TLV320AIC23 audio codec. The DSP board was programmed using C, and the audio input/output was handled through the codec using its configurable registers.

7.1 Board and Codec Initialization

The DSP board was initialized using the Board Support Library (BSL) provided by Texas Instruments. Audio signals were routed from the Line-In jack to the codec's analog input and output through the headphone jack. The codec was configured to operate in stereo mode with a 48 kHz sampling rate and 16-bit resolution.

The configuration of the codec was handled via the `DSK6416_AIC23_Config` structure. Below is the code used to configure the codec:

Listing 1: Codec Initialization Configuration

```
#include "dsk6416.h"
#include "dsk6416_aic23.h"
#include <math.h>

#define V_MUTE 0x00B0    // Minimum volume level (Mute)
#define V_0db  0x00F9    // Default starting volume
#define V_MAX  0x00f0
#define V_MIN  0x00d0

#define BY_PASS 0x000a
#define MIC_IN  0x0015
#define LINE_IN 0x0012

// Codec configuration settings
DSK6416_AIC23_Config config = {
    0x0017,  // Left line input channel volume
    0x0017,  // Right line input channel volume
    V_MIN,   // Left headphone volume
    V_MIN,   // Right headphone volume
    MIC_IN,  // Analog audio path control: configure for Mic input
    0x0000,  // Digital audio path control
    0x0000,  // Power down control
    0x0043,  // Digital audio interface format (I2S mode)
    0x0081,  // Sample rate control (48KHz)
    0x0001   // Digital interface activation
};
```

7.2 Audio Interfacing Implementation

Once the codec was configured, real-time audio data exchange between the DSP and the codec was established. The audio interface implementation continuously samples analog input from the Line-In, processes it, and outputs it through the headphone port. The interfacing is handled in a polling-based loop where the processor waits for new input data, processes it, and sends it to both audio output channels.

The core routine is implemented in the `main()` function. The audio sample is read from the codec using `DSK6416_AIC23_read()`, converted to a floating-point format for processing, and the processed output is converted back to a 16-bit PCM format before being sent to the codec using `DSK6416_AIC23_write()`.

The following code snippet outlines the audio interfacing logic:

Listing 2: Audio Interfacing Logic

```c
void main(void)
{
    DSK6416_AIC23_CodecHandle hCodec;

    Uint32 sample;
    Int16 raw_sample; // 16-bit raw sample from the codec

    unsigned long sample_index = 0;

    float input_sample, processed_sample;
    Int16 out_sample;

    // Initialize the board support library
    DSK6416_init();

    // Open the codec using the configuration
    hCodec = DSK6416_AIC23_openCodec(0, &config);

    // Main loop: read from codec, process the sample,
    // and send it to output
    while (1)
    {
     // Wait for a sample to be available from the ADC
     while (!DSK6416_AIC23_read(hCodec, &sample));

     // Extract 16-bit sample and convert to float [-1.0, 1.0]
     raw_sample = (Int16)sample;
     input_sample = ((float)raw_sample) / 32767.0f;

     // Apply the desired audio effect
     processed_sample =
     processAudioSample(input_sample, sample_index);

     // Convert back to 16-bit integer PCM format
     out_sample = (Int16)(processed_sample * 32767.0f);

     //Move to the next sample
     sample_index++;

     // Output to both left and right channels
     while (!DSK6416_AIC23_write(hCodec, out_sample));
     while (!DSK6416_AIC23_write(hCodec, out_sample));
    }
    // Close the codec (not reached during execution)
    DSK6416_AIC23_closeCodec(hCodec);}
```

This structure supports real-time audio processing with minimal latency and sufficient precision for implementing various audio effects. Each effect is modularized through the `processAudioSample()` function, enabling easy integration and switching of different signal processing algorithms without altering the core audio interface loop.

7.3 C Codes for implementations of all the effects used

7.3.1 WALL-E

Listing 3: WALL-E Effect

```
// Processing Constants
#define PI           3.14159265358979323846f
#define SAMPLE_RATE  48000.0f   // Sampling frequency in Hz
#define MOD_FREQ     200.0f     // Ring modulation frequency in Hz
#define BIT_DEPTH    5         // Bit depth for bit-crushing
#define FILTER_ORDER 3          // 3rd-order IIR filter
#define N_COEFF (FILTER_ORDER + 1)

float ring_modulation(float input_sample, unsigned long sample_index,
    float fs, float mod_freq)
{
    float t = sample_index / fs;
    float modulator = sinf(2.0f * PI * mod_freq * t);
    return input_sample * modulator;
}


float bit_crush(float input_sample) {
    // Triangular dither (-0.5LSB to +0.5LSB)
    float dither = (float)rand()/RAND_MAX - (float)rand()/RAND_MAX;
    dither = dither * 0.5f/(1 << BIT_DEPTH);
    input_sample += dither;
    int levels = 1 << BIT_DEPTH;
    return floorf(input_sample * levels) / levels;
}

// Low-Pass Filter
float b[N_COEFF] = {0.0181f, 0.0543f, 0.0543f, 0.0181f};  // Numerator
    coefficients
float a[N_COEFF] = {1.0f, -1.7600f, 1.1829f, -0.2781f};    //
    Denominator coefficients

float x_history[N_COEFF] = {0};  // Input sample history for filter
float y_history[N_COEFF] = {0};  // Output sample history for filter

float low_pass_filter(float input_sample)
{
```

```
    int i;
    // Shift histories: newest sample goes to index 0.
    for (i = FILTER_ORDER; i > 0; i--)
    {
        x_history[i] = x_history[i - 1];
        y_history[i] = y_history[i - 1];
    }
    x_history[0] = input_sample;

    // Compute filter output using the Direct Form I structure.
    float output = b[0] * x_history[0];
    for (i = 1; i < N_COEFF; i++)
    {
        output += b[i] * x_history[i] - a[i] * y_history[i];
    }
    y_history[0] = output;

    return output;
}

// Processing Chain:Combines ring modulation, bit-crushing, and low-pass filtering. input_sample is assumed to be a float in the range [-1.0, 1.0].

float processAudioSample(float input_sample, unsigned long sample_index)
{
    float processed_sample;
    processed_sample = input_sample;
    processed_sample = ring_modulation(processed_sample, sample_index, SAMPLE_RATE, MOD_FREQ;
    processed_sample = bit_crush(processed_sample);
    processed_sample = low_pass_filter(processed_sample);
    return processed_sample;
}
```

7.3.2 Dark Phonk Voice effect

Listing 4: Dark Phonk Voice

```
#define FILTER_ORDER 2
#define N_COEFF (FILTER_ORDER + 1)

float b[N_COEFF] = {0.0675f, 0.1349f, 0.0675f};
float a[N_COEFF] = {1.0f, -1.1430f, 0.4128f};

#define TREMOLO_RATE 6.0f
#define TREMOLO_DEPTH 1.0f
```

```
float process_tremolo(float input) {
    float t = sample_index / FS;
    float mod = (1.0f + TREMOLO_DEPTH * sinf(2.0f * PI * TREMOLO_RATE *
        t)) / 2.0f;
    return input * mod;
}

// Echo parameters: delay time = 0.05 s, decay factor = 0.5
#define ECHO_DELAY_TIME 0.05f
#define ECHO_DECAY_FACTOR 0.5f
#define ECHO_DELAY_SAMPLES  2400  //FS * ECHO_DELAY_TIME

// Circular buffer to store delayed samples
float echo_buffer[ECHO_DELAY_SAMPLES];
int echo_index = 0;
// Process one sample with echo (feed-forward delay effect)
float process_echo(float input)
{
    float delayed = echo_buffer[echo_index];
    float output = input + ECHO_DECAY_FACTOR * delayed;
    echo_buffer[echo_index] = input;
    echo_index = (echo_index + 1) % ECHO_DELAY_SAMPLES;
    return output;
}
float processAudioSample(float input_sample) {
    float processed_sample = input_sample;
    processed_sample = low_pass_filter(processed_sample); //SAME AS THE
        ONE IN WALL-E
    processed_sample = process_tremolo(processed_sample);
    processed_sample = process_echo(processed_sample);
    return processed_sample;
}
```

7.3.3 Radio Voice effect

Listing 5: Radio Voice

```

#define BIQUAD_STAGES 2
const float sos[BIQUAD_STAGES][6] = {
    { 0.23243f,  0.0f,   -0.23243f,  1.0f, -0.48949f,  0.53514f },
    { 0.90137f,  0.0f,   -0.90137f,  1.0f, -0.56619f,  0.73072f } };
typedef struct {
    float x[2], y[2];
} BiquadState;
BiquadState states[BIQUAD_STAGES] = {0};
float process_biquad(float in, int stage) {
    const float* s = sos[stage];
```

```
    BiquadState* st = &states[stage];
    float out = s[0]*in + s[1]*st->x[0] + s[2]*st->x[1]
                      - s[4]*st->y[0] - s[5]*st->y[1];
    st->x[1] = st->x[0]; st->x[0] = in;
    st->y[1] = st->y[0]; st->y[0] = out;
    return out;}
float bandpass_filter(float input) {
    float output = input;
    int i;
    for(i = 0; i < BIQUAD_STAGES; i++) {
        output = process_biquad(output, i);
    }
    return output * 3.5f;}
#define NOISE_THRESH 0.02f      // -40dB in linear scale
static float env_state = 0.0f;
#define HYSTERESIS   0.005f   // -46dB
float noise_gate(float input_sample) {
    float abs_val = fabsf(input_sample);
    // Envelope follower
    if(abs_val > env_state) {
        env_state += (abs_val - env_state) * 0.1f; // Fast attack
    } else {
        env_state += (abs_val - env_state) * 0.01f; // Slow release
    }
    // Hysteresis threshold
    if(env_state > NOISE_THRESH) {
        return input_sample;
    } else if(env_state > (NOISE_THRESH - HYSTERESIS)) {
        return input_sample * 0.5f; // Partial attenuation
    } else {
        return 0.0f;
    }
}
float distortion(float input_sample) {
    return tanhf(3.0f * input_sample);  // 3.0 = more grit
}
float processAudioSample(float input_sample, unsigned long sample_index
    ) {
  float processed = input_sample;
    processed = bandpass_filter(processed);
    processed = distortion(processed);
    processed = noise_gate(processed);
    if (processed > 0.95f) processed = 0.95f;
    if (processed < -0.95f) processed = -0.95f;
   return processed;
}
```

8 Results

The effects were successfully implemented and demonstrated live. Key observations include:

- Robotic voice had a clear, mechanical tone. It also created a wobbly effect as that of an extra-terrestrial being.
- Radio voice authentically simulated the characteristics of vintage broadcast. At stressed notes, the voice faced heavy distortion.
- Phonk effect was musically rhythmic and stylistically accurate. However, with slight changes in the Tremolo Rate, the output changes drastically, creating an almost ringing effect.
- 3D audio created a perceptible spatial depth using headphones.

9 Conclusion

The project successfully demonstrates the real-time processing capabilities of the TMS320C6416T DSP platform. Each effect highlighted different aspects of digital audio processing, from filtering to delay-based spatialization. Future work may include incorporating more complex effects or adapting the system for music production.
A pitch-shifting algorithm can also be employed which makes the voice either squeakier or denser. There exisiting algoritihms to achieve that including OLA (Overlap and Add) and WSOLA (Waveform Similarity Overlap Add) methods. However, the challenge persists in the implementation of such heavy effects in real-time with less CPU Load and Memory requirements.

10 References

1. Texas Instruments. *TMS320C6416 DSP Starter Kit Technical Reference*
2. Code Composer Studio User Guide, Texas Instruments.