Digital Signal Processing LAB
LAB 3 (Report)

Mohammad Parsa Dini - std id: 400101204
Mohammad Hossein Momeni - std id: 99102359

Introduction

In this lab, we process an input voice signal by separating its left and right channels. After isolating the channels, we apply signal buffering using the first method to efficiently manipulate the audio data. Subsequently, we implement auditory effects such as echo and fading on the audio signals and analyze the impact of various parameters on the output signals.

Interrupt Service Routine

The `serialPortRcvISR()` function is an interrupt service routine that processes incoming audio data. It reads a sample from the codec, amplifies it by multiplying with `volumeGain`, separates it into left and right signals, applies echo and fading effects to both channels, and writes the modified sample back to the codec. This approach ensures efficient signal processing with minimal CPU usage. To adjust parameters dynamically, two Gel files are used to manually modify `volumeGain` and `alpha`.

```
interrupt void serialPortRcvISR(){
    // reading the inputs
    short int buffer_right[N];
    short int buffer_left[N];
    short int win_buffer_right[L];
    short int win_buffer_left[L];
    Uint32 temp_right, temp_left, temp;
    DSK6416_AIC23_read(hCodec, &temp);
    temp_right = (temp * volumeGain) & 0xFFFF;
    temp_left = (temp * volumeGain) & 0xFFFF0000;
    temp_left = (temp_left >> FRAME_LEN);
    // moving avg logic
    for (i=L-1; i>0;i= i-1){
        win_buffer_left[i] = win_buffer_left[i-1];
        win_buffer_right[i] = win_buffer_right[i-1];
    }
    win_buffer_left[0] = (short int)temp_left;
    win_buffer_right[0] = (short int)temp_right;
    avg_left = moving_avg(win_buffer_left, L);
    avg_right = moving_avg(win_buffer_right, L);
```

```
    // main buffering logic
    for (i=N-1; i>0;i= i-1){
        buffer_left[i] = buffer_left[i-1];
        buffer_right[i] = buffer_right[i-1];
    }
    // echo and fading logic without moving average buffer
    //buffer_left[0] = (short int)temp_left+(short int)(alpha*buffer_left[N-1]/10);
    //buffer_right[0] = (short int)temp_right+(short int)(alpha*buffer_right[N-1]/10);
    // echo and fading logic using moving average buffer
    buffer_left[0] = (short int)avg_left+(short int)(alpha*buffer_left[N-1]/10);
    buffer_right[0] = (short int)avg_right+(short int)(alpha*buffer_right[N-1]/10);
    temp = ((Uint32) buffer_left[0] << FRAME_LEN) |  ((Uint32) buffer_right[0] & 0xFFFF);
    DSK6416_AIC23_write(hCodec, temp);
}
```

Code Breakdown

1. **Reading Input Samples** using `DSK6416_AIC23_read(hCodec, &temp);`.

2. **Amplifying Audio Samples And Separating Stereo Signals.** The audio sample (`temp`) is amplified by multiplying it by `volumeGain`. Then The right channel is extracted by masking the lower 16 bits (`& 0xFFFF`), while the left channel is masked to get the upper 16 bits (`& 0xFFFF0000`). The left channel is then shifted by `FRAME_LEN` bits (16 bits) to match the correct format for processing.

3. **Moving Average Logic:** Create a buffer with length $L = 10$ using method 1 (as described in the lab instructions) to apply a moving average filter, which will smooth the input signal.

```
// moving avg logic
for (i=L-1; i>0;i= i-1){
    win_buffer_left[i] = win_buffer_left[i-1];
    win_buffer_right[i] = win_buffer_right[i-1];
}
win_buffer_left[0] = (short int)temp_left;
win_buffer_right[0] = (short int)temp_right;
avg_left = moving_avg(win_buffer_left, L);
avg_right = moving_avg(win_buffer_right, L);
```

   where the `moving_avg` function computes the average of a frame signal with length L:

```
short int moving_avg(short int *arr, int l) {
    long long int sum =0;
    for (j=0;j< l;j++){
        sum += arr[j];
    }
    return (short int)(sum/l);
}
```

4. **Echo and Fading Logic:** The echo effect is implemented by storing the previous values of the `buffer_left` and `buffer_right` signals and shifting them to the right. For the current sample (`buffer_left[0]` and `buffer_right[0]`), we add the moving average (`avg_left` and `avg_right`) along with a weighted version of the last sample (`buffer_left[N-1]` and `buffer_right[N-1]`), scaled by `alpha`. This introduces the fading effect as `alpha` controls the strength of the echo. Nota that `alpha` can take

```c
// main buffering logic
for (i=N-1; i>0;i= i-1){
    buffer_left[i] = buffer_left[i-1];
    buffer_right[i] = buffer_right[i-1];
}
// echo and fading logic without moving average buffer
//buffer_left[0] = (short int)temp_left+(short int)(alpha*buffer_left[N-1]/10);
//buffer_right[0] = (short int)temp_right+(short int)(alpha*buffer_right[N-1]/10);
// echo and fading logic using moving average buffer
buffer_left[0] = (short int)avg_left+(short int)(alpha*buffer_left[N-1]/10);
buffer_right[0] = (short int)avg_right+(short int)(alpha*buffer_right[N-1]/10);
```

Note that `alpha` can take values from 0 to 10, increasing in steps of 1. In the code, we divide it by 10, which results in an increment of 0.1 for `alpha`.

5. **Writing the Modified Samples:** The modified audio samples (`buffer_left[0]` and `buffer_right[0]`) are packed into a 32-bit value (`temp`), where the left channel is shifted by `FRAME_LEN` bits (16 bits), and the right channel is masked to ensure it fits into the lower 16 bits. This packed value is then written back to the codec using `DSK6416_AIC23_write()`, which sends the processed audio data to be output.

```c
temp = ((Uint32) buffer_left[0] << FRAME_LEN) |  ((Uint32) buffer_right[0] & 0xFFFF);
DSK6416_AIC23_write(hCodec, temp);
```

Implementation

In our implementation, we configure the parameters as follows (sampling rate is set to $8kHz$):

```c
    int FRAME_LEN = 16;
    #define N 2000
    #define L 10
    int volumeGain=1;   // initial volumeGain
    int alpha=0;    // initial alpha
```

The main function is as follows:

```c
int main(){
    MBZIO_init(DSK6416_AIC23_FREQ_8KHZ); // Initialize DSP system at 8 kHz sample rate
    MBZIO_set_inputsource_microphone();  // Set input source to microphone

    hook_int();  // Configure and enable interrupts

    while(1){ }  // Infinite loop (processor waits for interrupts)
}
```

Note that increasing the buffer size `N` results in a greater delay. As the value of `L` (the moving average buffer size) increases, the resulting signal becomes smoother and less noisy. If `alpha` is set to 1, the last signal remains in the buffer and repeats periodically.