آزمایشگاه پردازش بی درنگ سیگنال های دیجیتال

گزاش کار جلسه اول

Up Sample & Down Sample

استاد شاه منصوری

فاطمه جلیلی

شماره دانشجویی : 810199398

تاریخ تحویل : 1402/8/12

توضیحات فایل ها :

کد متلب در فایل code1.mat قرار دارد

کد C در فایل main.c قرار دارد که فایل های upSampled_audio.txt و downSampled_audio.txt را خروجی می دهد

کد متلب که فایل های خروجی C را لود و play می کند در Read_C_outputs.m قرار دارد

توضیح کد ها:

- کد متلب : پس از خواندن کد ها در دو for loop جدا گانه برای قسمت upSample یکی در میان بین نمونه ها صفر اضافه می کند و برای قسمت downSample یکی در میان نمونه ها را حذف می کند. فایل audio را هم به صورت txt ذخیره می کند تا در کد C استفاده شود. یکی از توابع sound به دلخواه کامنت شود تا صدا ها ترکیب نشوند

```
%% DSP Lab - Sassion 1
%% Upsample
clear;
[audio, fs] = audioread('myAudio.aac');
save('fs.mat', 'fs')
writematrix(audio(:,1), 'audio.txt')
upSampled_audio = zeros(2 * size(audio, 1), 2);
for i = 1 : 2 : size(audio, 1)
    upSampled_audio(i, :) = audio((i+1)/2, :);
end
sound(upSampled_audio, fs);

%% Downsample
downSampled_audio = zeros(size(audio, 1) / 2, 2);
for i = 1 : size(audio, 1) / 2
    downSampled_audio(i, :) = audio(2*i, :);
end
%sound(downSampled_audio, fs);
```

- کد C : فایل audio.txt را می خواند و دو فایل upSampled_audio ، downSampled_audio را ایجاد می کند سپس تا زمانی که به انتهای فایل audio.txt نرسیده ایم خط به خط فایل را می خواند و برای قسمت upSample در هر دور حلقه درون فایل upSampled_audio خطی که این دور خوانده شده و یک 0 را می نویسد و برای قسمت downSample چک می کند اگر شماره حلقه زوج بود خطی که این دور خوانده شده را می نویسد.

```
#include <stdio.h>
int main()
{
    FILE *audioFile;
    FILE *upSampled;
    FILE *downSampled;
    char filename1[]="../audio.txt";
    audioFile = fopen( Filename: filename1, Mode: "r");
    char filename2[]="../upSampled_audio.txt";
    upSampled = fopen( Filename: filename2, Mode: "w");
    char filename3[]="../downSampled_audio.txt";
    downSampled = fopen( Filename: filename3, Mode: "w");
    char line[100];
    int numSamples=0;
    while (fgets( Buf: line, MaxCount: sizeof(line), File: audioFile) != NULL) {
        fprintf( stream: upSampled, format: "%s0\n", line);
        if (numSamples % 2 == 0) {
            fprintf( stream: downSampled, format: "%s", line);
        }
        numSamples++;
    }
    fclose( File: audioFile);
    fclose( File: upSampled);
    fclose( File: downSampled);
    printf( format: "lines: %d\n", numSamples);

    return 0;
}
```

- کد متلب خواندن فایل های خروجی C : برای خوانش درست صدا ها فرکانس مربوطه که در فایل اول متلب ذخیره کرده بودیم را load می کند و فایل های خروجی C را play می کند. یکی از توابع sound به دلخواه کامنت شود تا صدا ها ترکیب نشوند.

```
%% DSP Lab - Sassion 1
clear;
%% Read Upsampled C output .txt file
load('fs.mat', 'fs');
upSampled_audio = importdata('upSampled_audio.txt');
sound(upSampled_audio, fs);

%% Read Downsampled C output .txt file
downSampled_audio = importdata('downSampled_audio.txt');
%sound(downSampled_audio, fs);
%%
```

اثر Up Sample و Down Sample:

در up sample یکی در میان بین sample ها صفر اضافه می شود گویی تعداد sample های ریکورد شده در هر ثانیه دو برابر شده لذا انگار کیفیت صدا بالاتر است منتها چون هنگام sound از فرکانس اصلی صدا برای play کردن audio استفاده می کنیم صدا بم می شود زیرا در فرکانس نصف فرکانس اصلی خود play شده است ، اگر از 2*fs در هنگام sound استفاده کنیم صدای اصلی (از نظر بمی و زیری صدا) را می شنویم .

در down sample یکی در میان sample ها حذف می شوند گویی تعداد sample های ریکورد شده در هر ثانیه نصف شده است لذا کیفیت صدا کم تر است منتها چون هنگام sound از فرکانس اصلی صدا برای play کردن audio استفاده می کنیم صدا زیر می شود زیرا در فرکانس دوبرابر فرکانس اصلی خود play شده است ، اگر از fs/2 در هنگام sound استفاده کنیم صدای اصلی (از نظر بمی و زیری صدا) را می شنویم .

معایب و مزایا دو روش interrupt , polling:

در حالت polling، سیستم با استفاده از بررسی مکرر بیت آمادگی ارسال (XRDY) در ثبت کنترلی پورت سریال MCBSP (SPCR)، تشخیص می‌دهد که کدک آماده دریافت نمونه خروجی جدید است یا خیر. پردازنده به طور مکرر این بیت را بررسی می‌کند تا به نتیجه برسد که کدک آماده ارسال داده است. در صورتی که آمادگی کدک تأیید شود، نمونه خروجی جدید با استفاده از تابع MCBSP_write به کدک ارسال می‌شود. اگرچه حالت polling نسبت به روش interrupt ساده‌تر است، اما کارایی کمتری دارد. زیرا پردازنده بیشترین زمان خود را برای بررسی مکرر آمادگی کدک صرف می‌کند. به عبارت دیگر، حالت polling ممکن است منابع سیستم را بیش از حد استفاده کند و به بیهوده‌گرایی در استفاده از قدرت پردازشی پردازنده منجر شود.

از سوی دیگر، در حالت interrupt، سیستم از طریق استفاده از interrupt برای انجام وظایف مورد نیازی که توسط interrupt فعال می‌شوند، استفاده می‌کند. هنگامی که یک interrupt رخ می‌دهد، فرآیند فعلی پردازنده را متوقف می‌کند و آن را به یک روتین خدمات interrupt (ISR) هدایت می‌کند. این امکان را به ما می‌دهد تا بدون هدر دادن چرخه‌های پردازنده در بررسی مکرر، به سرعت واکنش نشان دهیم. استفاده از interrupt به ویژه در مواردی که واکنش به وقوع رویدادها به موقع ضروری است، مزیت دارد. با تعلیق فرآیند فعلی و اولویت‌بندی در انجام interrupt، سیستم می‌تواند وظایف زمان‌بندی‌شده را به طور کارآمد انجام دهد. با این حال، پیاده‌سازی سیستم‌های مبتنی بر interrupt نیازمند مدیریت دقیق اولویت‌بندی interrupt، مدیریت منابع مشترک و تأمین همگام‌سازی مناسب است.