پروژه پایانی
درس سیگنال ها و سیستم ها
رادیو دیجیتال

دکتر محمدی

امیرعلی پاکدامن دنیوی
۹۷۴۱۱۲۵۲
پارسا عیسی زاده
۹۷۴۱۲۳۶۴
یاسمین مدنی
۹۷۵۳۲۲۶۵

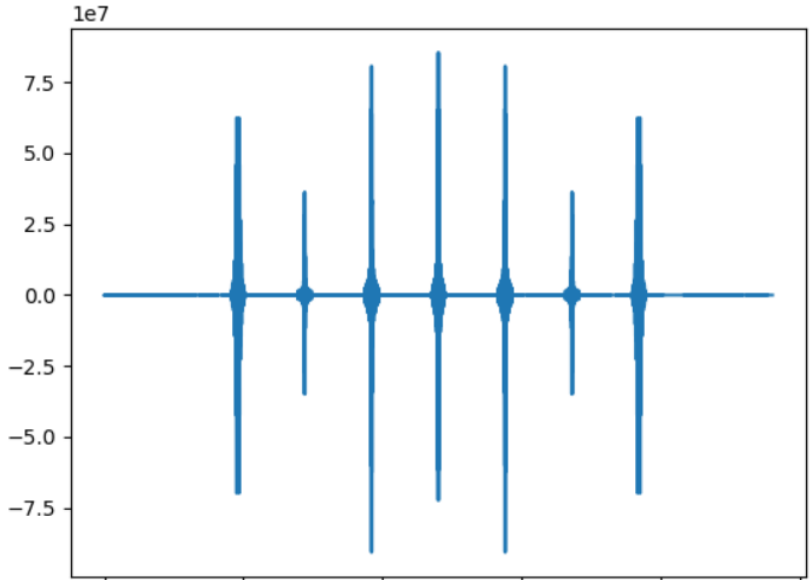
فهرست

روند کلی

ایده کلی در این پروژه به این صورت است که ابتدا یک فیلتر پایین گذر با استفاده از روش پنجره گذاری Kaiser طراحی کردیم که مشخصات این فیلتر با آزمون و خطا و با توجه به نمودار تبدیل فوریه سیگنال حاصله از فایل ورودی تعیین شد.

سپس، برای پیاده سازی به صورت Real-Time ، در هر سری، قسمتی از فایل ورودی خوانده میشود که در ابتدا، این سیگنال ورودی در حوزه فرکانس شیفت داده میشود به طوری که فرکانس مورد نظر کاربر، در محدوده شنوایی قرار گیرد.

سپس، با اعمال فیلتر پایین گذر طراحی شده، سایر شبکه ها حذف میشوند و در نهایت تکه پردازش شده از طریق جریان خروجی پخش میشود.



نمودار ۱- تبدیل فوریه سیگنال ورودی

تابع EXTRACTOR(FREQ)

این تابع، فرکانس درخواستی کاربر را به عنوان پارامتر ورودی freq دریافت کرده و با استفاده از فرمول $\omega = 2\pi \cdot \text{freq} \cdot T$ و $T = 1/480000$ ،امگای شبکه مورد نظر را محاسبه میکند. با استفاده از امگای بدست آمده (که در تابع پایتون به صورت *omega* تعریف شده)، تابع ریاضی *shift=cos(omega x n)* تعریف میشود که بعدا در سیگنال ورودی ضرب شده و فرکانس مورد نظر(*freq*) را روی صفر قرار داده تا در محدوده شنوایی قرار گیرد.

کار دیگر تابع *extractor* ، محاسبه پاسخ ضربه فیلتر *Kaiser* و *M* (طول فیلتر منهای یک) است که با استفاده از تابع *Kaiser* که داخل آن تعریف شده محاسبه میشود. مشخصات فیلتر هم با آزمون و خطا و با توجه به نمودار تبدیل فوریه سیگنال حاصله از فایل ورودی تعیین میشود.

این تابع *extractor* ، *omega, h, M, shift* را به عنوان خروجی میدهد.

تابع STARTRADIO(FILEPATH, FREQ)

این تابع که در واقع تابع اصلی برنامه است، آدرس فایل txt سیگنال ورودی را به عنوان ورودی filePath و فرکانس شبکه درخواستی را به عنوان ورودی freq میگیرد.

- ابتدا با صدا زدن تابع extractor(freq) ، مقادیر omega, h, M, shift محاسبه میگردد.

- سپس، متغیر File برای مدیریت فایل ورودی و استریم های p و player برای ساختن جریان صوتی خروجی ساخته میشوند.
- مقدار CHUNK (تعداد خط هایی که در هر سری، از فایل ورودی خوانده میشود) نیز برابر با ۴۸۰۰۰ قرار داده شده که با توجه به نرخ فایل ورودی (۴۸۰،۰۰۰) تاخیر ۰.۱ ثانیه ای برای سیستم در نظر گرفته میشود که در مجموع با تاخیر اعمال فیلتر به مقدار بیشینه ۰.۲ ثانیه میرسد.
- یک آرایه(حافظه) به نام buffer نیز در نظر گرفته شده تا M مقدار نهایی قسمت خوانده شده از ورودی در مرحله قبل را در خود نگه میدارد. زیرا در زمان اعمال فیلتر که با عملگر کانولوشن صورت میگیرد، هر کدام از داده های ورودی به M مقدار قبلی وابسته است.

در ادامه یک حلقه while وجود دارد که تا زمانی که فایل ورودی تمام نشده تکرار میشود. در این حلقه ابتدا به تعداد CHUNK خط از فایل خوانده میشود، سپس در shift ضرب میشود تا فرکانس مورد نظر به محدوده شنوایی منتقل شود. سپس با buffer ادغام شده تا M مقدار قبلی برای کانولوشن موجود باشد. سپس سیگنال حاصل با h فیلتر که توسط extractor(freq) خروجی داده شده، با مود valid کانوالو میشود. نتیجه کانولوشن که همان تکه پردازش شده هست، نرخ آن با ضریب ۲ کاهش داده میشود تا توسط استریم قابل پخش باشد. سپس سیگنال نهایی در استریم خروجی player نوشته شده و صدایش پخش میشود. در انتها نیز M مقدار آخر سیگنال پخش شده برای استفاده در تکرار حلقه بعدی، در buffer ذخیره میشود و حلقه تا تمام شدن فایل تکرار میشود.

در صورت اتمام حلقه نیز تمامی فایل ها و استریم های ایجاد شده بسته میشوند.

محیط گرافیکی TKINKER (امتیازی)

در نهایت خواستیم تا رابط گرافیکی طراحی کنیم.از انجایی که یک رادیو را نوشتیم سعی کردیم تا رابط گرافیکی را شبیه یک رادیو طراحی کنیم .

برای نوشتن رابط گرافیکی با کاربر از کتاب خانه tkinter استفاده کردیم .به وسیله تابع ()Tk یک پنجره تعریف می کنیم .عکسی از یک رادیو را با تابع PhotoImage در پس زمینه برنامه میگذاریم .

سپس بر روی آن ۵ دکمه با تابع button ، دو ورودی با تابع Entry و یک نمایشگر با تابع Textمیگذاریم . ۴ دکمه برای شبکه های رادیو است . یک ورودی برای ادرس فایل ، یک ورودی برای فرکانس رادیو (بهKHz)، یک دکمه برای پخش فرکانس ورودی و در نهایت نمایشگر برای نام شبکه و فرکانس .

با تابع place موقعیت این ها را روی صفحه مشخص میکنیم. هر کدام از توابع گرافیکی که ذکر شد یک attribute برای صفحه ای که در آن قرار می- گیرند دارند و می توان رنگ پس زمینه ، رنگ فونت اندازه و ... را تعیین کرد .

در تابع button یک پارامتر به نام command داریم که تابعی که با کلیک روی دکمه اجرا می شود را به عنوان ورودی میگیرد . ما برای هر دکمه یک تابع به صورت زیر تعریف کردیم که ابتدا تابع fileButton را صدا میزند ، سپس اگر ورودی فایل خالی نبود ، نوشته نمایشگر را تغییر می دهد و تابع ()StartRadio را برای فرکانس شبکه دکمه فشرده شده صدا میزند .

در تابع fileButton ، ابتدا stream های قبلی متوقف می شوند (تا اگر هنگامیکه صدا پخش میشد شبکه عوض شد صدا ها تداخل نکنند .) و سپس چک میکند که

ورودی فایل صحیح باشد (اگر فایل فرمت مناسبی نداشت یا فایل وجود ندارد یا ورودی خالی است روی نمایشگر خطا را نشان میدهد .)

Command دکمه پخش فرکانس تابع PlayButton است که اگر ورودی ها مقدار نادرستی داشتند خطا نمایش میدهد و در نهایت تابع startradio را برای فرکانس ورودی صدای میزند .

برای تغییر نوشته نمایشگر ، نوشته قبلی را با delete پاک میکنیم و مقدار جدی را با insert وارد میکنیم . با تابع config حالت ویرایش را غیر فعال میکنیم و در نهایت پنجره را update میکنیم.

در نهایت محیط گرافیکی برنامه :