شبیه‌سازیِ Line Coder

تهیه و تنظیم: مبین خیری

شماره دانشجویی: 994421017

استاد راهنما: دکتر میرسامان تاجبخش

چکیده:

در گزارشِ پیش‌رو قصد داریم به شبیه‌سازی Line Coderهای مختلف بپردازیم. در واقع، هدف اصلی از از این تمرین، تولید رشته‌های تصادفی از بیت‌ها و نمایش آن‌ها در قالب سیگنال‌های دیجیتال با Line Codingهای مختلفِ قطبی است. این تمرین شامل سه بخشِ مختلف است:

1. تولید بیت و شبیه سازی تابع ارسال {Send() & Receive()}.
2. تولید یک زمان رندم برای برهم زدن synchronization و بررسی آن در روش‌های Line Codingهای مختلف.
3. نمایش سیگنالِ دیجیتال ارسالی و دریافتی به همراه خط‌چین‌های clock توسط یک گراف

همچنین در این گزارش -علاوه بر قسمت‌های یادشده - برای حلِ بخشِ اختیاریِ تمرین به پیوسته کردن تولید بیت‌ها و نمایش نمودارِ آن‌ها نیز خواهیم پرداخت. به عبارتی تلاش خواهیم کرد نموداری مانند گراف CPU یا RAM در Task Manager سیستم‌عاملِ ویندوز که در آن سیگنال‌های نمایش داده شده سیگنال‌های ارسالی و دریافتی هستند، ترسیم کنیم.

برای آنکه بتوانیم چالشِ پیش‌رو را حل کرده و روش‌های به‌کار رفته برای حل آن را پیاده‌سازی کنیم، ابتدا لازم است که درک درستی از دو مفهومِ **"انتقال دیجیتال در شبکه های کامپیوتری"** و نیز **"تکنیک‌های کدگذاری داده دیجیتال"** به‌دست آوریم.

انتقال دیجیتال در شبکه های کامپیوتری

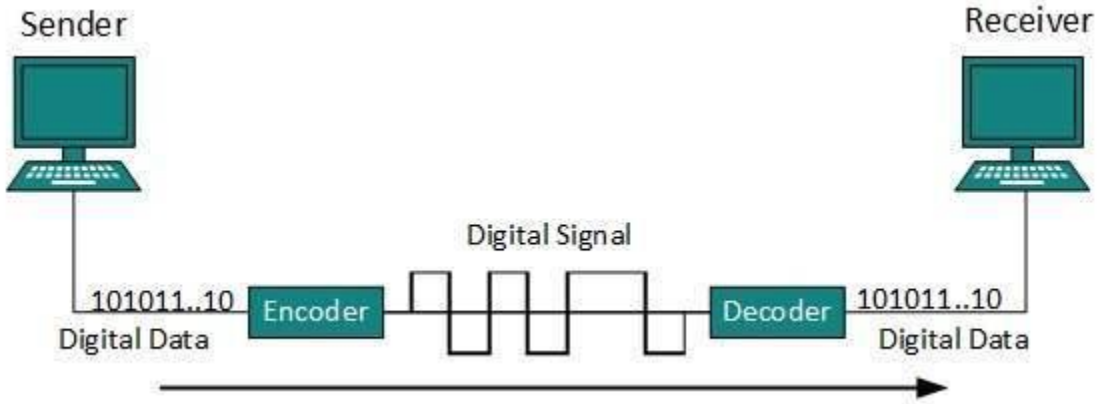
داده‌ها یا اطلاعات را می‌توان به دو روش آنالوگ یا دیجیتال ذخیره کرد. برای این که یک رایانه بتواند از داده‌ها استفاده کند، باید آن‌ها را به صورت دیجیتال ذخیره کنیم. سیگنال‌ها نیز همانند داده‌ها می‌توانند به شکل آنالوگ یا دیجیتال باشند. برای انتقال دیجیتالی داده‌ها باید ابتدا آن‌ها را به شکل دیجیتال تبدیل کنیم.

تبدیل دیجیتال به دیجیتال

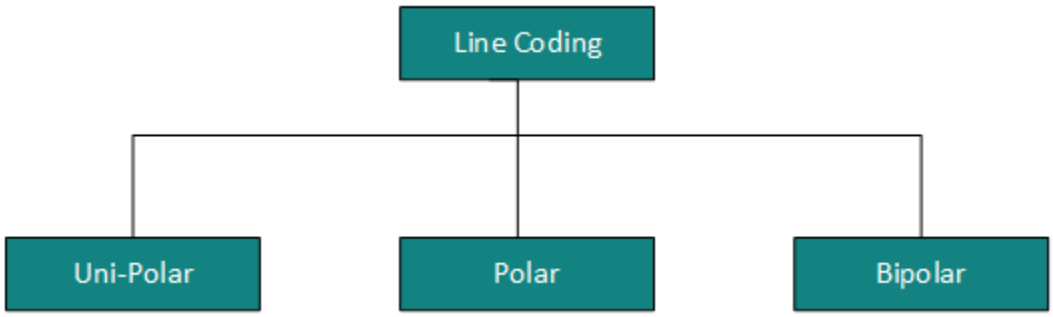
در این بخش شیوه تبدیل داده‌های دیجیتال به سیگنال‌های دیجیتال را توضیح می‌دهیم. این کار را می‌توان به دو روش کدینگ خط و کدینگ بلوک انجام داد .در همه انواع ارتباط‌ها کدینگ خط ضروری است؛ در حالی که کدینگ بلوک اختیاری است.

کدینگ خط

فرایند تبدیل داده‌های دیجیتال به سیگنال‌های دیجیتال به نام کدینگ خط نامیده می‌شود. داده‌های دیجیتال در قالب باینری هستند و به شکل داخلی به صورت یک سری از 0 و 1 نمایش می‌یابند.

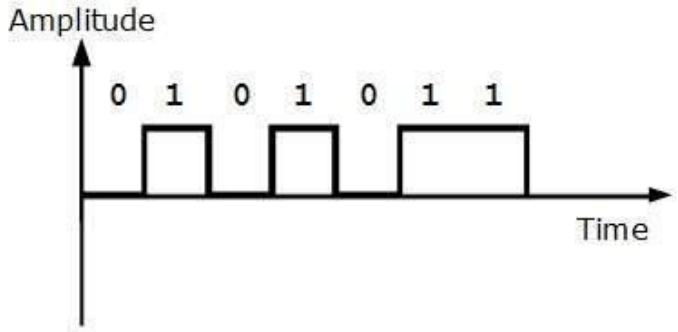


سیگنال دیجیتال به وسیله سیگنال گسسته نمایش می‌یابد که نماینده داده‌های دیجیتال است. سه نوع طرح‌بندی کدینگ خط به صورت زیر وجود دارند:



انکودینگ تک‌قطبی(Uni-Polar)

طرح‌بندی انکودینگ تک‌قطبی از سطح ولتاژ سیگنال برای نمایش داده‌ها استفاده می‌کند. در این حالت برای نمایش مقدار باینری 1، ولتاژ بالا انتقال می‌یابد و برای نمایش 0 هیچ ولتاژی انتقال نمی‌یابد. این حالت به نام «Unipolar-Non-return-to-zero»نیز نامیده می‌شود، زیرا هیچ حالت میانی وجود ندارد و در هر یک از حالت‌ها یا 0 و یا 1 نمایش می‌یابد.

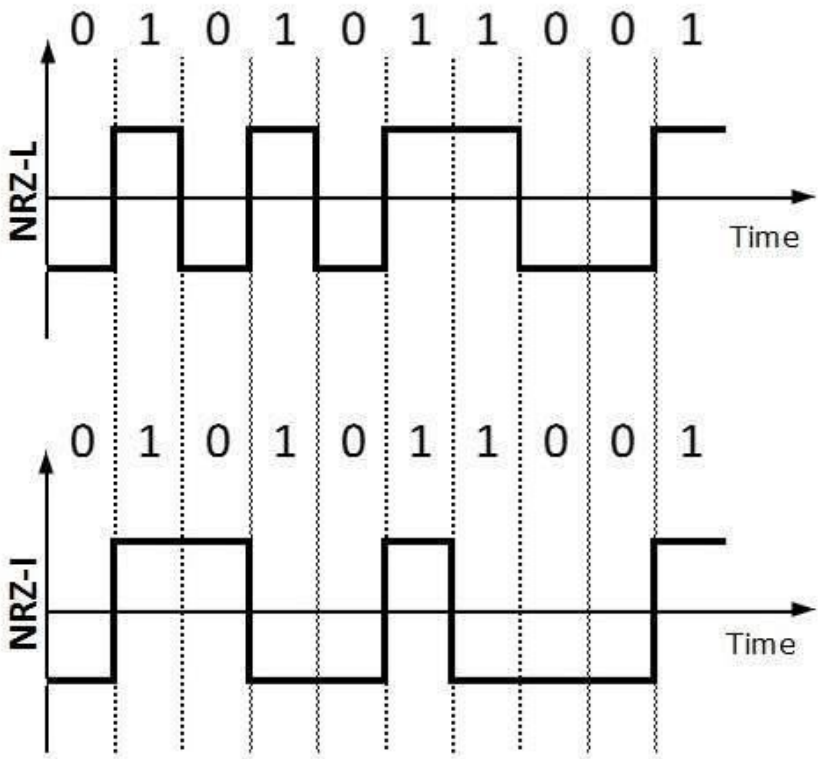


انکودینگ قطبی(Polar)

در طرح‌بندی انکودینگ قطبی از چندین سطح ولتاژ برای نمایش مقادیر باینری استفاده می‌شود. انکودینگ قطبی به چهار نوع امکان‌پذیر است:

Polar Non-Return to Zeroیا NRZ قطبی

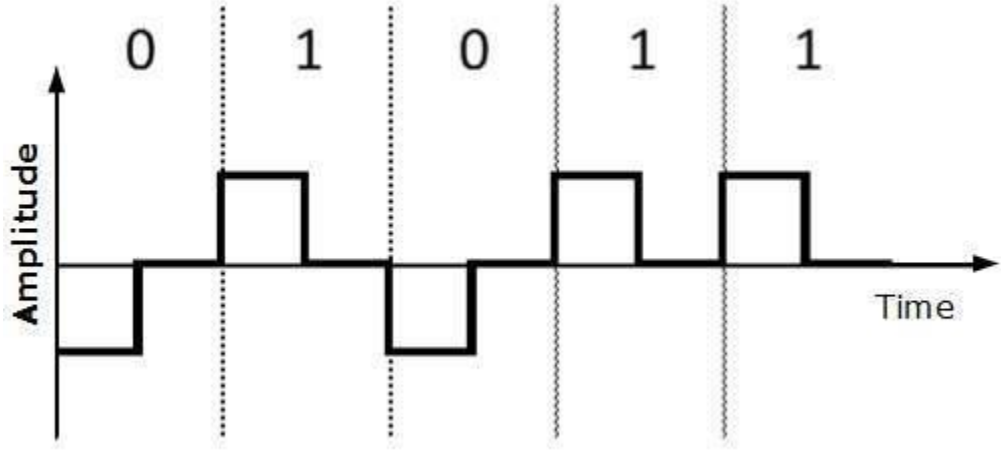
در این روش از دو سطح مختلف ولتاژ برای نمایش مقدارهای باینری استفاده می‌شود. به طور کلی ولتاژهای مثبت نماینده 1 و مقدار منفی نماینده 0 است .همچنین NRZ محسوب می‌شود، زیرا هیچ شرایط میانی وجود ندارد. طرح NRZ دو نسخه به صورت NRZ-L و NRZ-I دارد.



در روش NRZ-L هنگامی که با یک بیت متفاوت مواجه شویم، سطح ولتاژ تغییر می‌یابد؛ در حالی که در NRZ-I، ولتاژ زمانی تغییر پیدا می‌کند که با 1 مواجه شویم.

RZیاReturn to Zero

مشکل NRZ این است که وقتی کلاک گیرنده و فرستنده همگام نشده باشند، گیرنده نمی‌تواند بفهمد که یک بیت چه زمانی پایان یافته است و بیت بعدی چه زمانی آغاز شده است.



RZاز سه سطح ولتاژ به صورت ولتاژ مثبت برای نمایش 1، ولتاژ منفی برای نمایش 0 و ولتاژ صفر برای نمایش هیچ استفاده می‌کند. بدین ترتیب سیگنال‌ها درون بیت تغییر می‌یابند و نه بین بیت‌ها.

Manchester

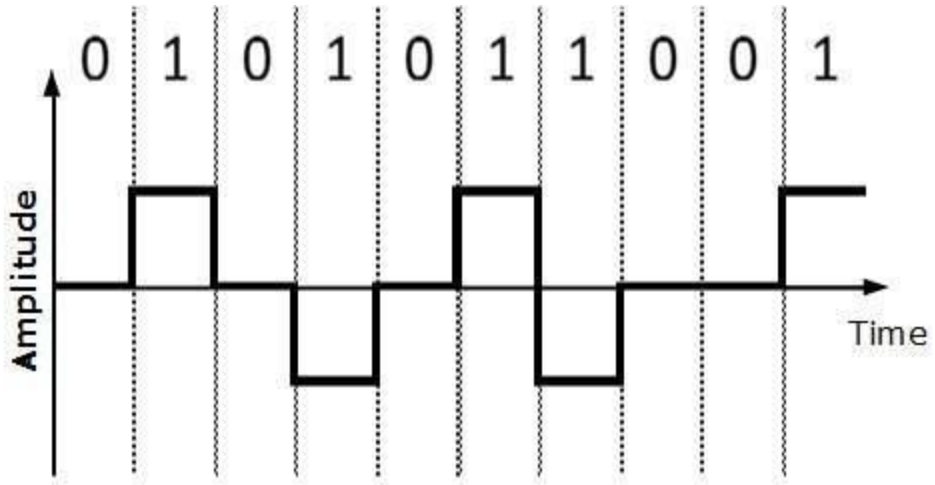
این طرح‌بندی انکودینگ ترکیبی از RZ و NRZ-L است. زمان بیت دو به نیمه تقسیم می‌شود. سیگنال در میانه بیت حمل می‌شود و هنگامی که با بیت متفاوتی مواجه شود فاز را تغییر می‌دهد.

Differential Manchester

این طرح‌بندی انکودینگ ترکیبی از RZ و NRZ-I است. در این روش نیز سیگنال در میانه بیت حمل می‌شود و فاز تنها در مواردی که با 1 مواجه شود تغییر پیدا می‌کند.

انکودینگ دوقطبی(Bipolar)

در روش انکودینگ دوقطبی از سه سطح ولتاژ به صورت مثبت، منفی و صفر استفاده می‌کنیم. ولتاژ صفر نماینده مقدار باینری 0 و بیت 1 نمایش‌دهنده تغییر ولتاژهای مثبت و منفی است.



کدینگ بلوک

برای اطمینان از صحت فریم داده‌های دریافتی، از بیت‌های تکراری (افزونگی) استفاده می‌شود. برای نمونه در توازن زوج (even-parity) یک بیت توازن اضافه می‌شود تا تعداد 1 ها در فریم داده زوج باشند. بدین ترتیب تعداد بیت‌های اولیه افزایش می‌یابد. این روش کدینگ بلوک نامیده می‌شود.

کدینگ بلوک به وسیله نمادگذاری ممیز (/) به صورت mB/nB نمایش می‌یابد .در مواردی که n < m باشد، بلوک m بیتی با بلوک n بیتی جایگزین می‌شود .کدینگ بلوک شامل سه مرحله است:

- تقسیم
- جایگزینی
- ترکیب

پس از اتمام کدینگ بلوک، از کدینگ خط برای انتقال استفاده می‌شود.

تبدیل آنالوگ به دیجیتال

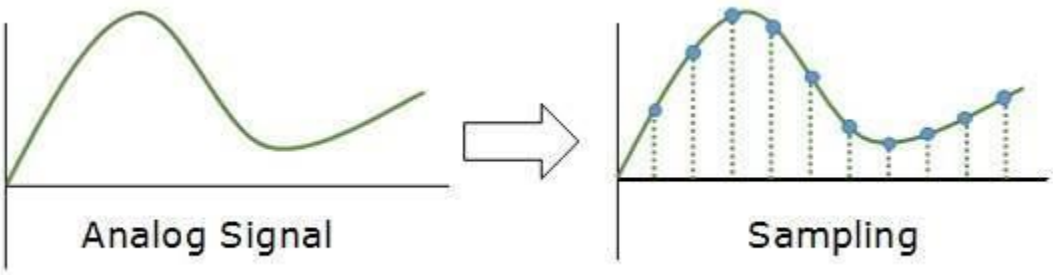
میکروفن‌ها صدا را به صورت آنالوگ تولید می‌کنند و دوربین‌ها نیز ویدئوهای آنالوگ تهیه می‌کنند که به صورت داده آنالوگ با آن برخورد می‌شود .برای انتقال داده‌های آنالوگ روی سیگنال‌های دیجیتال باید از تبدیل آنالوگ به دیجیتال استفاده کنیم.

داده‌های آنالوگ جریان پیوسته‌ای از داده‌ها به شکل موج هستند؛ در حالی که داده‌های دیجیتال گسسته هستند. برای تبدیل موج‌های آنالوگ به داده‌های دیجیتال، از «مدولاسیون کد پالس (PCM) «استفاده می‌کنیم.

PCMیکی از پراستفاده‌ترین روش‌ها برای تبدیل داده‌های آنالوگ به شکل دیجیتال است که در سه مرحله صورت می‌گیرد:

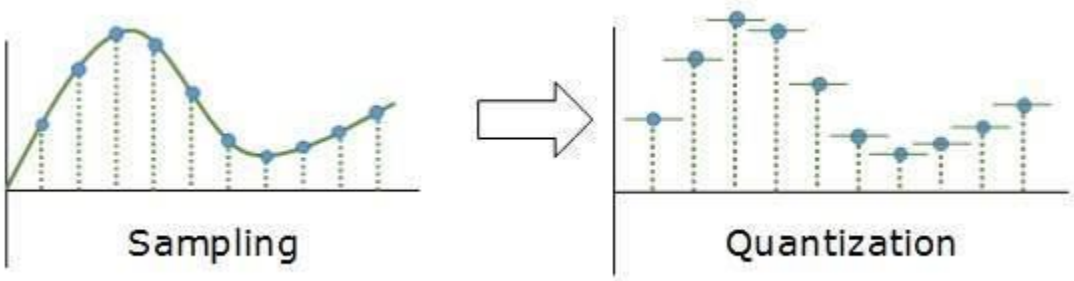
- نمونه‌برداری
- کمی سازی
- انکودینگ

نمونه‌برداری(Sampling)



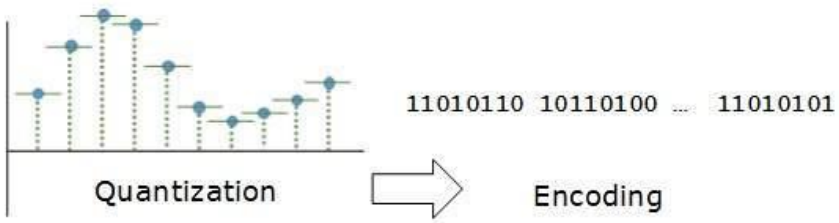
سیگنال آنالوگ در طی هر بازه T نمونه‌برداری می‌شود. مهم‌ترین عامل در نمونه‌برداری نرخ مربوط به نمونه‌برداری سیگنال آنالوگ است. بر اساس قضیه Nyquist، نرخ نمونه‌برداری باید دست‌کم دو برابر بالاترین فرکانس سیگنال باشد.

کمّی‌سازی(Quantization)



نمونه‌برداری شکل گسسته‌ای از سیگنال آنالوگ پیوسته است. هر الگوی گسسته شدت سیگنال آنالوگ را در آن لحظه نشان می‌دهد. کمّی‌سازی بین دو مقدار بیشینه شدت و کمینه شدت صورت می‌گیرد. کمی‌سازی تخمینی از مقدار آنالوگ لحظه‌ای به دست می‌دهد.

انکودینگ

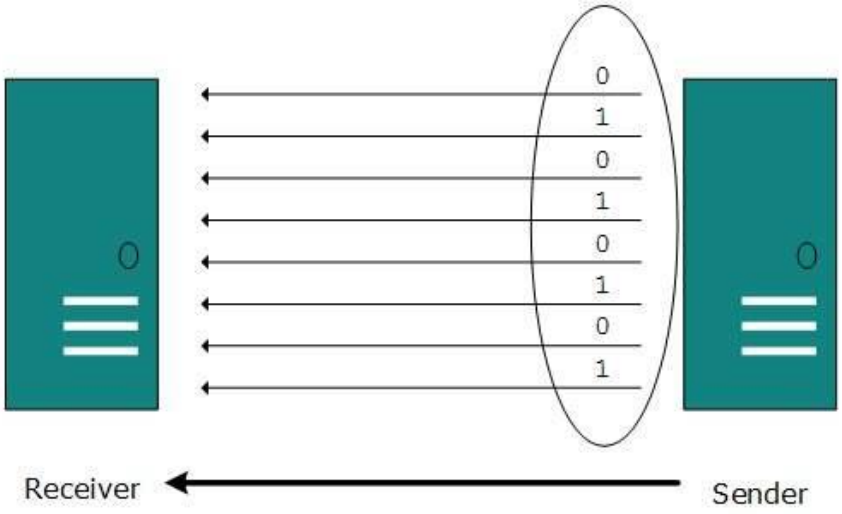


در مرحله انکودینگ هر مقدار تخمین زده شده به قالب باینری تبدیل می‌شود.

حالت‌های انتقال

حالت انتقال به چگونگی انتقال داده‌ها بین دو رایانه گفته می‌شود .داده‌های باینری به شکل 1 و 0 را می‌توان به دو روش موازی و سریال ارسال کرد.

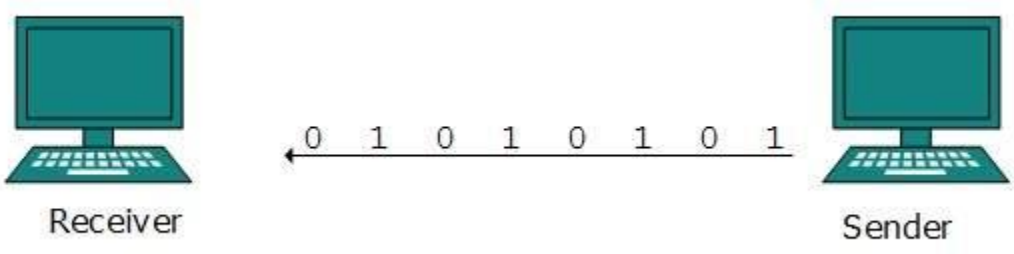
انتقال موازی(Parallel Transmission)



در این روش، بیت‌های باینری در گروه‌هایی با طول ثابت سازمان‌دهی می‌شوند. هم فرستنده و هم گیرنده به صورت موازی و با تعداد خطوط داده‌ای یکسانی به هم متصل هستند. هر دو رایانه بین خطوط داده‌ای با ترتیب بالا و پایین تمییز قائل شده‌اند. فرستنده همه بیت‌ها را روی همه خطوط به صورت یکجا ارسال می‌کند. از آنجا که خطوط داده‌ای با تعداد بیت‌ها در یک گروه فریم داده برابر است، یک گروه کامل از بیت‌ها (فریم داده) در هر بار ارسال می‌شود. مزیت انتقال موازی سرعت بالا و عیب آن هزینه سیم‌ها است، چون این تعداد باید برابر با تعداد بیت‌های ارسالی به صورت موازی باشد.

انتقال سریال

در روش انتقال سریال، بیت‌ها یکی پس از دیگری به صورت یک صف ارسال می‌شوند. انتقال سریال آن‌ها به یک کانال ارتباطی نیاز دارد.



انتقال سری می‌تواند به صورت همگام یا ناهمگام اجرا شود.

انتقال سریال ناهمگام

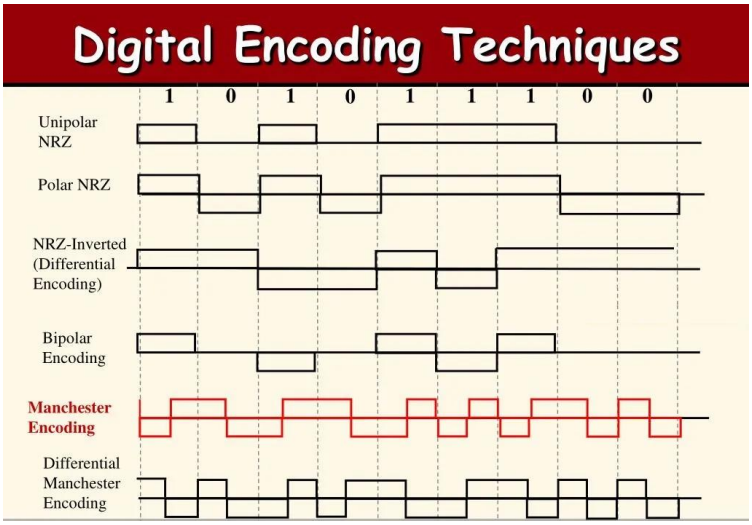
دلیل انتخاب چنین نامی برای این روش انتقال داده‌ها این است که زمان‌بندی هیچ اهمیتی ندارد و بیت‌های داده الگوی خاصی دارند. به این ترتیب گیرنده می‌تواند آغاز و پایان بیت‌های داده‌ای را تشخیص دهد. برای نمونه یک 0 به ابتدای هر بایت داده اضافه می‌شود و یک یا چند 1 در انتهای آن قرار می‌گیرد. دو قاب داده‌ای (بایت) پیوسته می‌توانند بین خود شکافی داشته باشند.

انتقال سریال همگام

در انتقال سریال همگام بحث زمان‌بندی حائز اهمیت است، زیرا هیچ سازوکاری برای شناسایی ابتدا و انتهای بیت‌های داده‌ای وجود ندارد. همچنین از هیچ الگو یا پیشوند/پسوند استفاده نمی‌شود. بیت‌های داده‌ها در حالت پشت سر هم ارسال می‌شوند و هیچ شکافی بین بایت‌ها (8 بیت) وجود ندارد. یک مجموعه از بیت‌های داده‌ای می‌تواند شامل چند بایت باشد. از این رو زمان‌بندی بسیار مهم است.

در این روش شناسایی و تمییز بیت‌ها از هم و قرار دادن آن‌ها به صورت بایت بر عهده گیرنده است. مزیت این روش سرعت بالا است و همچنین به هیچ سربار اضافی به صورت بیت‌های هدر و فوتر اضافی مانند روش انتقال ناهمگام نیاز نداریم.

تکنیک‌های کدگذاری داده دیجیتال



»کدگذاری (Encoding) «به فرایند تبدیل داده‌ها یا رشته‌ای از حروف، سمبل‌ها و کاراکترها به یک فرمت خاص گفته می‌شود که برای امن کردن فرایند انتقال انجام می‌گیرد. «کدگشایی (Decoding) «نیز به عکس فرایند کدگذاری اطلاق می‌شود که برای استخراج اطلاعات اصلی از سیگنال با فرمت تبدیل یافته مورد استفاده قرار می‌گیرد. در ادامه قصد داریم به بررسی روش‌ها و تکنیک‌هایی بپردازیم که در کدگذاری داده دیجیتال از آن‌ها استفاده می‌شود.

تکنیک‌های کدگذاری داده دیجیتال

همان طور که اشاره کردیم، کدگذاری به فرایند استفاده از الگوهای مختلف از سطوح ولتاژ و جریان گفته می‌شود که با کمک آن می‌توان صفر و یک‌های سیگنال دیجیتال در لینک انتقال را نمایش داد. انواع مختلفی از تکنیک‌ها وجود دارند که در کدگذاری خط مورد استفاده قرار می‌گیرند، اما متداول‌ترین این تکنیک‌ها «تک قطبی(Unipolar) «، »قطبی(Polar) «، »دوقطبی (Bipolar) «و »منچستر (Manchester) « هستند. تکنیک‌های کدگذاری داده دیجیتال به انواع مختلفی تقسیم‌بندی می‌شوند که در اصل به نوع تبدیل داده بستگی دارند. در حالت کلی انواع تبدیل داده شامل ۴ گروه تبدیلات زیر است.

تبدیل داده آنالوگ به سیگنال آنالوگ :تکنیک‌های مدولاسیون مانند مدولاسیون دامنه یاAM ، مدولاسیون فرکانس یا FM و مدولاسیون فاز یا PM سیگنال‌های آنالوگ در گروه تبدیل داده آنالوگ به سیگنال آنالوگ قرار می‌گیرند.

تبدیل داده آنالوگ به سیگنال دیجیتال :این فرایند را می‌توان «دیجیتال‌سازی (Digitization) «نیز نامید که با تکنیک‌هایی مانند مدولاسیون کد پالس یا PCM انجام می‌شود. در واقع این تبدیل همان مدولاسیون دیجیتال است» .نمونه‌برداری (Sampling) «و» کوانتیزاسیون (Quantization) «دو فاکتور بسیار مهم در مدولاسیون دیجیتال هستند که در مباحث قبلی مجله فرادرس به بررسی نحوه انجام آن‌ها پرداختیم. البته باید توجه کرد که تکنیک «مدولاسیون دیجیتال دلتا (Delta Modulation) «خروجی بهتری را نسبت به PCM تولید می‌کند.

تبدیل داده دیجیتال به سیگنال آنالوگ :تکنیک‌های مدولاسیون مانند »کلیدزنی شیفت دامنه « (Amplitude Shift Keying)یاASK ، »کلیدزنی شیفت فرکانس (Frequency Shift Keying) «یا FSK و »کلیدزنی شیفت فاز (Phase Shift Keying) «یا PSK این نوع از تبدیل داده را انجام می‌دهند.

تبدیل داده دیجیتال به سیگنال دیجیتال :روش‌های مختلفی برای تبدیل داده‌های دیجیتال به سیگنال دیجیتال وجود دارند که در ادامه قصد داریم به بررسی این روش‌ها بپردازیم.

تکنیک کدگذاری داده دیجیتال غیربازگشتی به صفر

تکنیک کدگذاری «غیربازگشتی به صفر (Non Return to Zero) «یا NRZ برای سطوح بالای ولتاژ، مقدار ۱ و برای سطوح پایین ولتاژ، مقدار ۰ را اختصاص می‌دهد. مشخصه اصلی تکنیک کدگذاری NRZ این است که در طول بازه بیت، سطح ولتاژ ثابت باقی می‌ماند. شروع یا پایان یک بیت نشان داده نخواهد شد و اگر مقدار بیت قبلی و مقدار بیت کنونی ثابت باقی بماند، سطح ولتاژ نیز ثابت حفظ می‌شود. یک نمونه از تکنیک کدگذاری داده دیجیتال NRZ در تصویر زیر توضیح داده شده است.

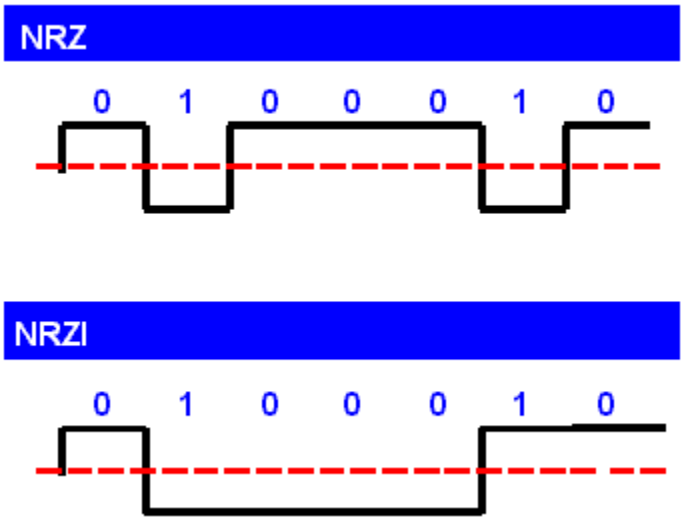
اگر به تصویر فوق دقت کنیم، مشاهده می‌کنیم که چون بازه‌های طولانی از سطوح ولتاژ ثابت وجود دارد و نیز به این دلیل که ممکن است سنکرون‌سازی کلاک به دلیل عدم وجود بازه بیت از بین برود، در نتیجه تمایز بین مقادیر ۰ و ۱ برای گیرنده دشوار می‌شود. به همین دلیل دو نوع دیگر از تکنیک NRZ نیز به وجود آمده‌اند.

تکنیک کدگذاری داده غیربازگشتی به صفر-سطح(NRZ – L)

در تکنیک کدگذاری داده غیربازگشتی به صفر-سطح یا NRZ – L فقط زمانی در پلاریته سیگنال تغییر به وجود می‌آید که سیگنال ورودی از ۱ به ۰ و یا از ۰ به ۱ تغییر کند. این تکنیک بسیار شبیه به NRZ است، اما بیت اول از سیگنال ورودی باید یک تغییر در پلاریته داشته باشد.

تکنیک کدگذاری داده غیربازگشتی به صفر معکوس(NRZ – I)

اگر در سیگنال ورودی بیت بعدی برابر با ۱ باشد، آن‌گاه در ابتدای بازه بیت یک، گذاری اتفاق خواهد افتاد. اما اگر بیت بعدی در سیگنال ورودی برابر با ۰ باشد، هیچ گذاری در ابتدای بازه بیت صفر به وجود نمی‌آید. کدهای NRZ دارای این عیب بزرگ هستند که هنگامی که یک رشته از صفرها و یک‌ها وجود داشته باشد، همگام‌سازی کلاک فرستنده با کلاک گیرنده کاملا مختل می‌شود و بنابراین لازم است یک خط کلاک جداگانه فراهم شود. روش کدگذاری NRZ – I در تصویر زیر نشان داده شده است.



کدگذاری دو فاز

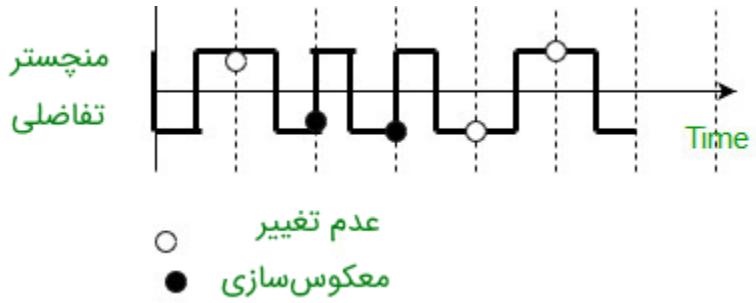
سطح سیگنال برای هر بیت دو بار، هم در ابتدا و هم در میانه، کنترل می‌شود. به همین دلیل نرخ کلاک دو برابر نرخ تبادل داده است و در نتیجه نرخ مدولاسیون نیز دو برابر است. سیگنال کلاک از خود سیگنال داده گرفته می‌شود. پهنای باند مورد نیاز برای این تکنیک کدگذاری داده بزرگ تر از سایر روش‌ها است .در حالت کلی دو نوع کدگذری داده دو فاز وجود دارد که عبارتند از: کدگذاری منچستر دو فاز و کدگذاری منچستر تفاضلی.

کدگذاری منچستر دو فاز

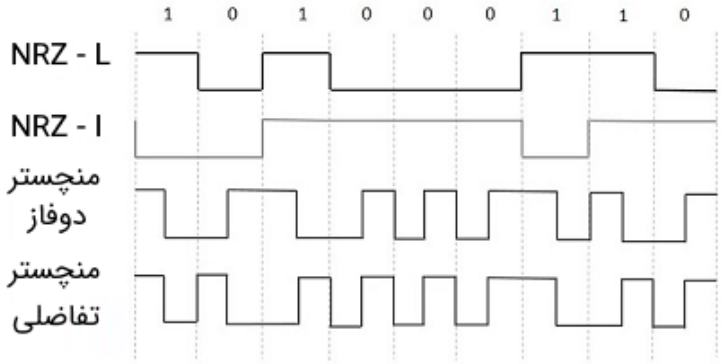
در این تکنیک کدگذاری، گذار حالت در وسط بازه بیت انجام می‌شود. در کدگذاری دو فاز منچستر، گذار در شکل موج پالس نهایی از سطح HIGH به سطح LOW و به ازای بیت ورودی یک رخ می‌دهد. در حالی که گذار از سطح LOW به سطح HIGHبرای بیت ورودی ۰ اتفاق می‌افتد.

کدگذاری منچستر تفاضلی

در این نوع از کدگذاری داده، همواره یک گذار حالت در وسط بازه بیت اتفاق می‌افتد. همچنین اگر یک گذار در ابتدای بازه بیت رخ دهد، آن‌گاه بیت ورودی برابر با صفر در نظر گرفته می‌شود. اگر هیچ گذار حالتی در ابتدای بازه بیت اتفاق نیفتد، بیت ورودی یک است. در تصویر زیر نحوه اعمال روش کدگذاری منچستر تفاضلی نشان داده شده است.



در تصویر زیر مقایسه‌ای بین تکنیک‌های کدگذاری دادهNRZ-L ، NRZ-I، منچستر دو فاز و منچستر تفاضلی برای ورودی‌های دیجیتال مختلف نشان داده شده است.



تکنیک کدگذاری داده بلوکی(Block)

در بین تکنیک‌های مختلف کدگذاری داده بلوکی، متداول‌ترین انواع کدگذاری 4B/5B و B/6T 8هستند. در هر دو این روش‌ها، تعداد بیت‌ها از راه‌های مختلفی پردازش می‌شوند که ما در این گزارش به توضیح این دسته از تکنیک‌ها نخواهیم پرداخت.

نکته: هنگام جست‌وجو برای منابع مختلفی که می‌توانستند در پیاده‌سازیِ این پروژه به من کمک کنند، تصمیم گرفتم علاوه بر الگوریتم‌های لازم، به پیاده‌سازیِ تکنیک کدینگِ NRZ-Unipolar که تفاوت جزئی با الگوریتم‌های NRZ-L و NRZ-I دارد، و نیز تکنیکِ AMI که در کلاس درباره‌ی آن بحث شده بود نیز بپردازم.

پیاده‌سازی

برای پیاده‌سازیِ راحت‌تر و همچنین سرراست‌تر شدنِ پروسه‌ی عیب‌یابیِ برنامه تصمیم گرفتم که برای هرکدام از تکنیک‌های یادشده یک تابع تعریف کرده و رفتارِ هرکدام از الگوریتم‌ها را در قبالِ سیگنال‌های دریافتی شبیه‌سازی کنم.

در اولین مرحله بعد از اجرای برنامه، تکه‌کدِ زیر به کاربر نمایش داده شده و از او خواسته می‌شود که الگوریتمِ دلخواهِ خود راانتخاب کرده و دنباله‌ی رشته‌هایی از 0 و 1 را به‌عنوانِ ورودی به برنامه بدهد.

```
print("+++++++++++++++ LINE CODING SCHEMES  ++++++++++++++++++")
choice = -1
while choice != 0:
    print("Select Line Coding Scheme\n1.NRZ-Unipolar\n2.NRZ-L\n3.NRZ-I\n4.RZ\n5.Manchester
\n6.Differential Manchester\n7.AMI\n8.2B1Q\nPress 0 to exit the program")
    choice = int(input("Enter your choice: "))
    if choice == 1:
        NRZ_Unipolar()
    elif choice == 2:
        NRZ_L()
    elif choice == 3:
        NRZ_I()
    elif choice == 4:
        RZ()
    elif choice == 5:
        manchester()
    elif choice == 6:
        differentialManchester()
    elif choice == 7:
        ami()
    elif choice == 8:
        TwoB1Q()
    elif choice == 0:
        exit
```

حاصلِ اجرای این کد، مطابقِ شکل زیر خواهد بود. همان‌طور که در تصویر نشان داده شده، کاربر می‌تواند از میانِ 7 الگوریتمِ کدینگ یکی را انتخاب کرده و رشته‌ی دلخواهِ خود را به‌عنونِ ورودی در اختیار برنامه قرار دهد.

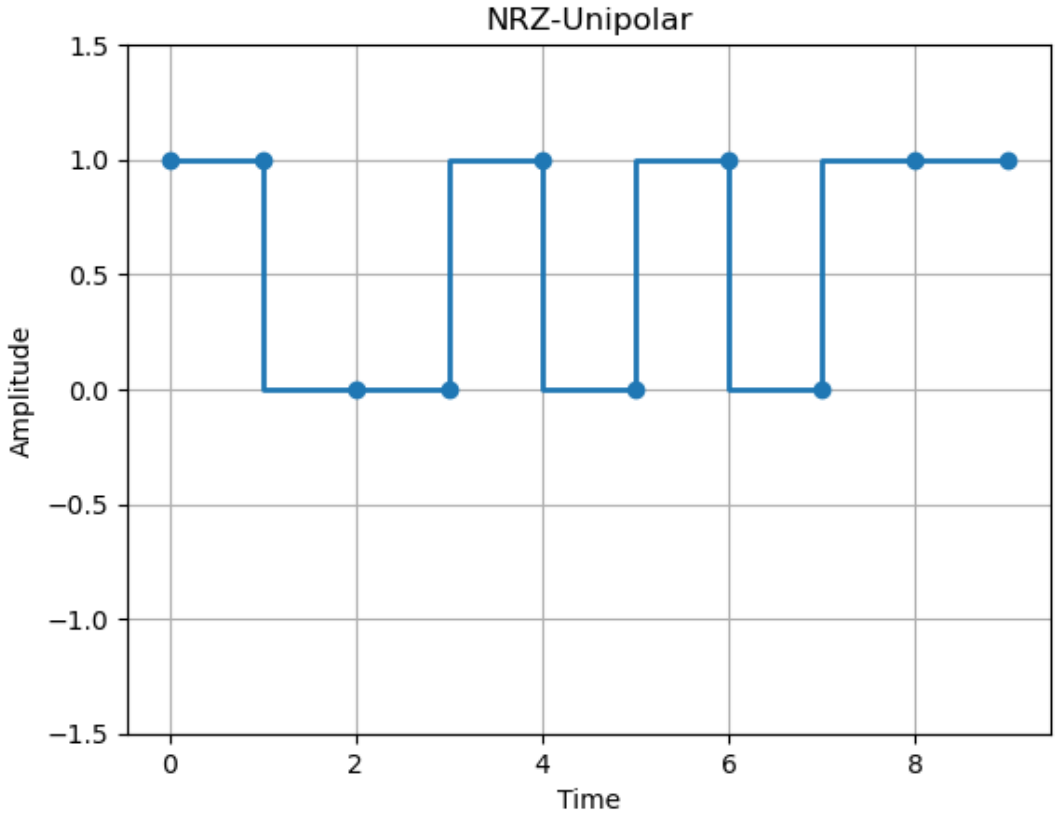
```
++++++++++++++ LINE CODING SCHEMES  +++++++++++++++++
Select Line Coding Scheme
1.NRZ-Unipolar
2.NRZ-L
3.NRZ-I
4.RZ
5.Manchester
6.Differential Manchester
7.AMI
Press 0 to exit the program
```

در ادامه، تک‌تکِ توابعِ به‌کاررفته در پیاده‌سازیِ پروژه را نشان داده و شیوه‌ی عملکردِ آن‌ها روی رشته‌ی 1100101011 را نمایش خواهیم داد.

الگوریتمِ NRZ Unipolar

```python
# NRZ unipolar
def NRZ_Unipolar():
    digitalData = list(input("Enter digital data elements: "))
    print(digitalData)
    y  = [int(i) for i in digitalData]
    print(y)
    x = []
    for i in range(len(y)):
        x.append(i)
    print(x)
    plt.step(x,y, 'o-'  ,linewidth = '2')
    plt.ylim(-1.5, 1.5)
    plt.title('NRZ-Unipolar')
    plt.xlabel('Time')
    plt.ylabel('Amplitude')
    plt.grid(0.2)
    plt.show()
```

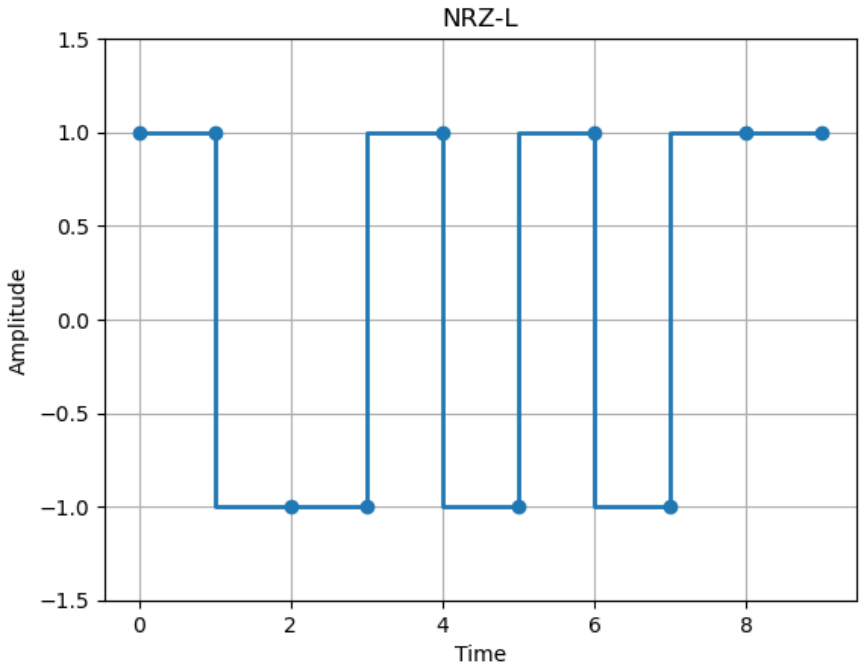
خروجیِ نشان‌داده‌شده:



الگوریتمِ NRZ-L

```python
# NRZ-L polar
def NRZ_L():
    digitalData = list(input("Enter digital data elements: "))
    print(digitalData)
    y = []
    for i in digitalData:
        if int(i) == 0:
            y.append(-1)
        else:
            y.append(int(i))
    print(y)
    x = []
    for i in range(len(y)):
        x.append(i)
    print(x)
    plt.step(x,y, 'o-'  ,linewidth = '2')
    plt.ylim(-1.5, 1.5)
    plt.title('NRZ-L')
    plt.xlabel('Time')
    plt.ylabel('Amplitude')
    plt.grid(0.2)
    plt.show()
```

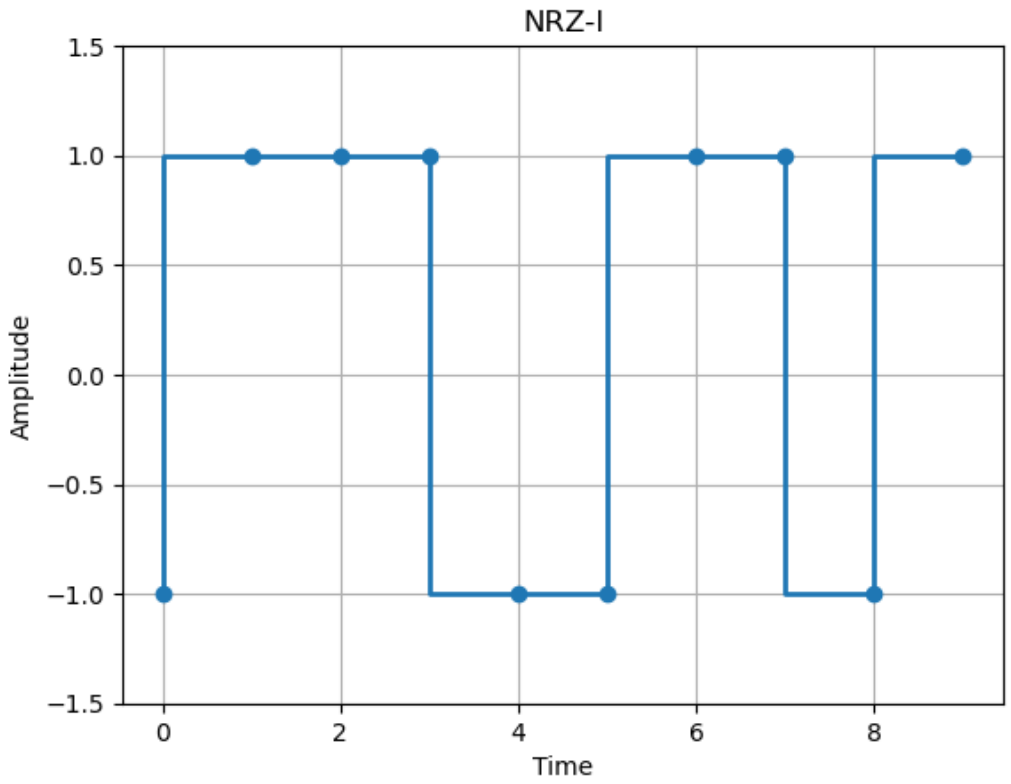
خروجِ نشان‌داده‌شده:



الگوریتمِ NRZ-I

```
# NRZ-I (polar)
def NRZ_I():
    digitalData = list(input("Enter digital data elements: "))
    print(digitalData)
    temp = []
    for i in digitalData:
        temp.append(int(i))

    if temp[0] == 1:
        y = [-1]
    else:
        y = [1]
    z = 1
    for i in range(len(temp)-1):
        if temp[i]+temp[z] == temp[i]:
            y.append(y[z-1])
        else:
            if y[z-1] == -1:
                y.append(1)
            else:
                y.append(-1)
        z = z + 1
    print(y)
    x = []
    for i in range(len(y)):
        x.append(i)
    print(x)
    plt.step(x,y, 'o-'  ,linewidth = '2')
    plt.ylim(-1.5, 1.5)
    plt.title('NRZ-I')
    plt.xlabel('Time')
    plt.ylabel('Amplitude')
    plt.grid(0.2)
    plt.show()
```

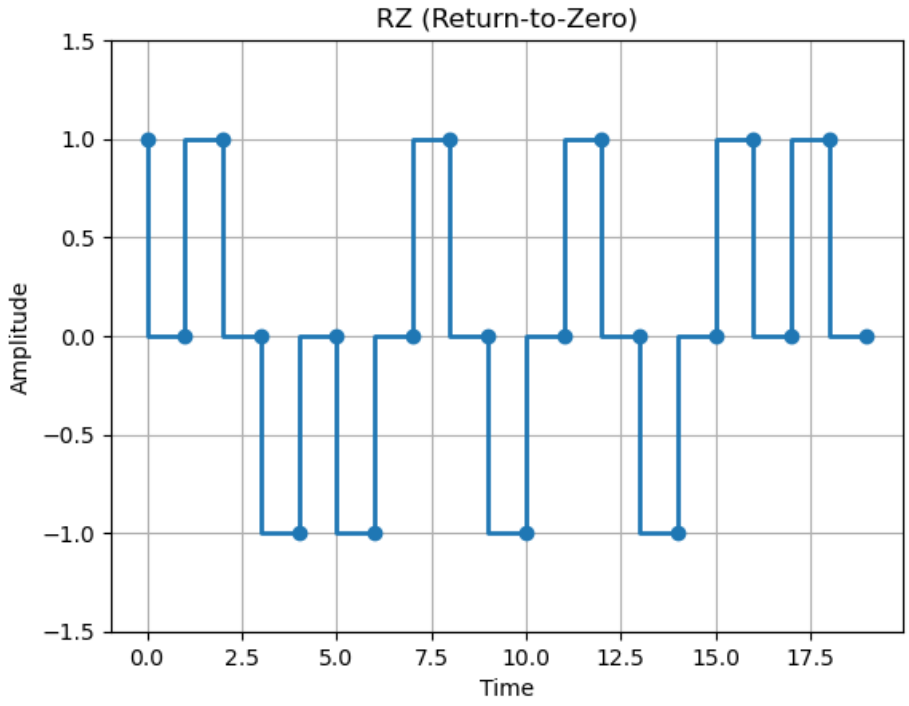
خروجیِ نشان‌داده‌شده:



الگوریتمِ RZ

```python
def RZ():
    digitalData = list(input("Enter digital data elements: "))
    print(digitalData)
    temp  = [int(i) for i in digitalData]
    print(temp)

    y = []
    for i in range(len(temp)):
        if temp[i] == 1:
            y.append(1)
            y.append(0)
        else:
            y.append(-1)
            y.append(0)

    x = []
    for i in range(len(y)):
        x.append(i)
    print(x)
    plt.step(x,y, 'o-', linewidth = '2')
    plt.ylim(-1.5, 1.5)
    plt.title('RZ (Return-to-Zero)')
    plt.xlabel('Time')
    plt.ylabel('Amplitude')
    plt.grid(0.2)
    plt.show()
```

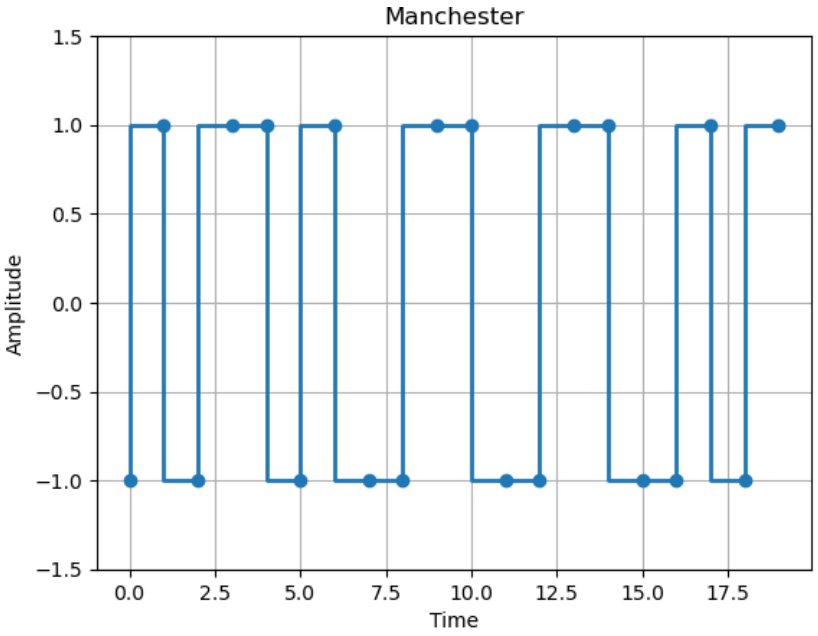
خروجیِ نشان‌داده‌شده:



الگوریتمِ Manchester

```python
# Manchester
def manchester():
    digitalData = list(input("Enter digital data elements: "))
    print(digitalData)
    temp  = [int(i) for i in digitalData]
    print(temp)

    y = []
    for i in range(len(temp)):
        if temp[i] == 1:
            y.append(-1)
            y.append(1)
        else:
            y.append(1)
            y.append(-1)

    x = []
    for i in range(len(y)):
        x.append(i)
    print(x)
    plt.step(x,y, 'o-', linewidth = '2')
    plt.ylim(-1.5, 1.5)
    plt.title('Manchester')
    plt.xlabel('Time')
    plt.ylabel('Amplitude')
    plt.grid(0.2)
    plt.show()
```

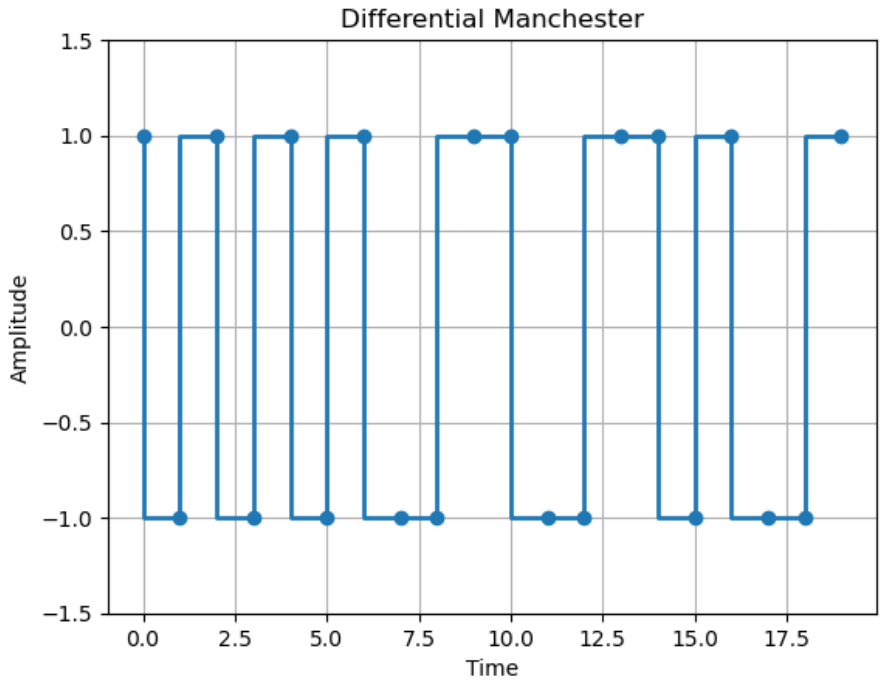
خروجیِ نشان‌داده‌شده:



الگوریتمِ Differential Manchester

```python
# Differential Manchester
def differentialManchester():
    digitalData = list(input("Enter digital data elements: "))
    print(digitalData)
    temp  = [int(i) for i in digitalData]
    print(temp)

    #y = [1,-1, 1,-1, -1,1, 1,-1, 1,-1, -1,1, -1,1]
    y = []

    for i in range(len(temp)):
        if temp[i] == 1:
            if len(y) == 0:
                y.append(1)
                y.append(-1)
            elif y[i] == 1:
                y.append(-1)
                y.append(1)
            else:
                y.append(1)
                y.append(-1)
        else:
            y.append(1)
            y.append(-1)

    print(y)
    x = []
    for i in range(len(y)):
        x.append(i)
    print(x)
    plt.step(x,y, 'o-', linewidth = '2')
    plt.ylim(-1.5, 1.5)
    plt.title('Differential Manchester')
    plt.xlabel('Time')
    plt.ylabel('Amplitude')
    plt.grid(0.2)
    plt.show()
```

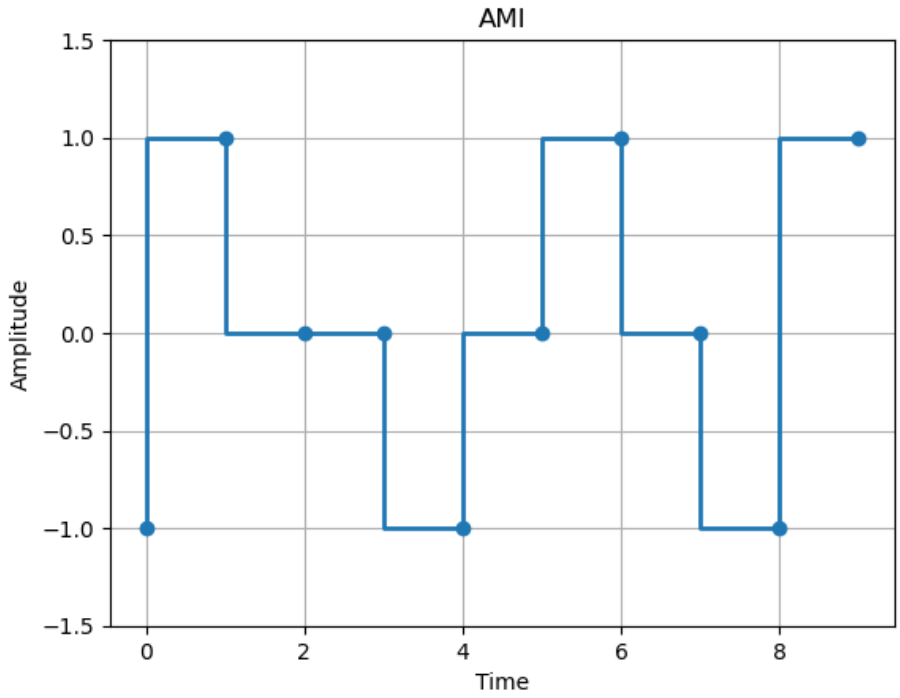
خروجیِ نشان‌داده‌شده:



الگوریتمِ AMI

```python
# AMI
def ami():
    digitalData = list(input("Enter digital data elements: "))
    print(digitalData)
    temp  = [int(i) for i in digitalData]
    print(temp)

    y = []
    # y = [1,0,-1,1,0,-1,0]
    # print(y)
    bList = [temp[0]]
    b = 0
    for i in temp:
        if int(i) == 0:
            y.append(0)
        else:
            if bList[b] == 1:
                y.append(-1)
                bList.append(-1)
                b = b + 1
            else:
                y.append(1)
                bList.append(1)
                b = b + 1

    print(y)
    x = []
    for i in range(len(y)):
        x.append(i)
    print(x)
    plt.step(x,y, 'o-', linewidth = '2')
    plt.ylim(-1.5, 1.5)
    plt.title('AMI')
    plt.xlabel('Time')
    plt.ylabel('Amplitude')
    plt.grid(0.2)
    plt.show()
```

خروجیِ نشان‌داده‌شده:



برای رسمِ تمامیِ این نمودارها از کتابخانه‌ی Matplotlib استفاده شده که امکانات زیادی را برای رسم انواعِ نمودارها در اختیارِ برنامه‌نویسانِ پایتون قرار می‌دهد.

علاوه بر این کد، نسخه‌ی مشابهی از آن را با تغییر دادنِ برخی جزئیات برای استفاده به‌صورتِ ماژول‌های مجزای یک کتابخانه پیاده‌سازی کرده‌ام که در بخش ضمائمِ این گزارش موجود است.

حال، بعد از پیاده‌سازیِ نسخه‌های ابتدایی و ساده‌ی این الگوریتم‌ها زمانِ آن فرا رسیده که کارکردهای مشابهی را با افزودنِ توابعِ ()Send و ()Receive و نیز تولید بیت رندُم و شبیه سازی ارسال و تولید زمان رندم در سمت گیرنده برای برهم زدن synchronization را به کدهای بالا بیفزاییم.

همچنین ذکرِ این نکته لازم به نظر می‌رسد که عملکردِ نمایش زنده‌ی ارسال و دریافتِ بیت‌ها نیز درقسمتِ پیش‌رو کامل‌تر شده و به‌نمایش گذاشته خواهد شد.

برای پیاده‌سازیِ امکاناتِ ذکرشده، در این قسمت علاوه بر امکاناتِ استانداردِ موجود در زبانِ برنامه‌نویسیِ پایتون، از امکانات و توابعِ موجود در کتابخانه‌های Numpy، Matplotlib و Random نیز بهره خوه برد.

توابعِ ارسال و دریافتِ سیگنال (داده) توسطِ الگوریتمِ RZ

```python
import random
import matplotlib.pyplot as plt

def send_RZ(data):
    signal = []
    for bit in data:
        if bit == 1:
            signal.extend([1, 0])
        else:
            signal.extend([-1, 0])
    return signal

def receive_RZ(signal):
    data = []
    for i in range(0, len(signal), 2):
        if signal[i] == 1:
            data.append(1)
        else:
            data.append(0)
    return data
```

توابعِ ارسال و دریافتِ سیگنال (داده) توسطِ الگوریتمِ NRZ-I

```python
def send_NRZ_I(data):
    signal = []
    prev = 1
    for bit in data:
        if bit == 1:
            signal.extend([-1 * prev, prev])
            prev = -prev
        else:
            signal.extend([prev, prev])
    return signal

def receive_NRZ_I(signal):
    data = []
    prev = 1
    for i in range(0, len(signal), 2):
        if signal[i] == -prev:
            data.append(1)
            prev = -prev
        else:
            data.append(0)
    return data
```

توابعِ ارسال و دریافتِ سیگنال (داده) توسطِ الگوریتمِ NRZ-L

```python
def send_NRZ_L(data):
    signal = []
    for bit in data:
        if bit == 1:
            signal.extend([1, 1])
        else:
            signal.extend([-1, -1])
    return signal

def receive_NRZ_L(signal):
    data = []
    for i in range(0, len(signal), 2):
        if signal[i] == -1:
            data.append(1)
        else:
            data.append(0)
    return data
```

توابعِ ارسال و دریافتِ سیگنال (داده) توسطِ الگوریتمِ Manchester

```python
def send_manchester(data):
    signal = []
    for bit in data:
        if bit == 1:
            signal.extend([-1, 1])
        else:
            signal.extend([1, -1])
    return signal

def receive_manchester(signal):
    data = []
    for i in range(0, len(signal), 2):
        if signal[i] == -1 and signal[i+1] == 1:
            data.append(1)
        else:
            data.append(0)
    return data
```

توابعِ ارسال و دریافتِ سیگنال (داده) توسطِ الگوریتمِ Differential-Manchester

```python
def send_differential_manchester(data):
    signal = []
    prev = 1
    for bit in data:
        if bit == 1:
            signal.extend([-1 * prev, prev])
            prev = -prev
        else:
            signal.extend([prev, -prev])
    return signal

def receive_differential_manchester(signal):
    data = []
    prev = 1
    # skip any initial zero samples
    i = 0
    while i < len(signal) and signal[i] == 0:
        i += 1
    # decode the signal
    for j in range(i, len(signal), 2):
        if signal[j] == -prev and signal[j+1] == prev:
            data.append(1)
            prev = -prev
        else:
            data.append(0)
    return data
```

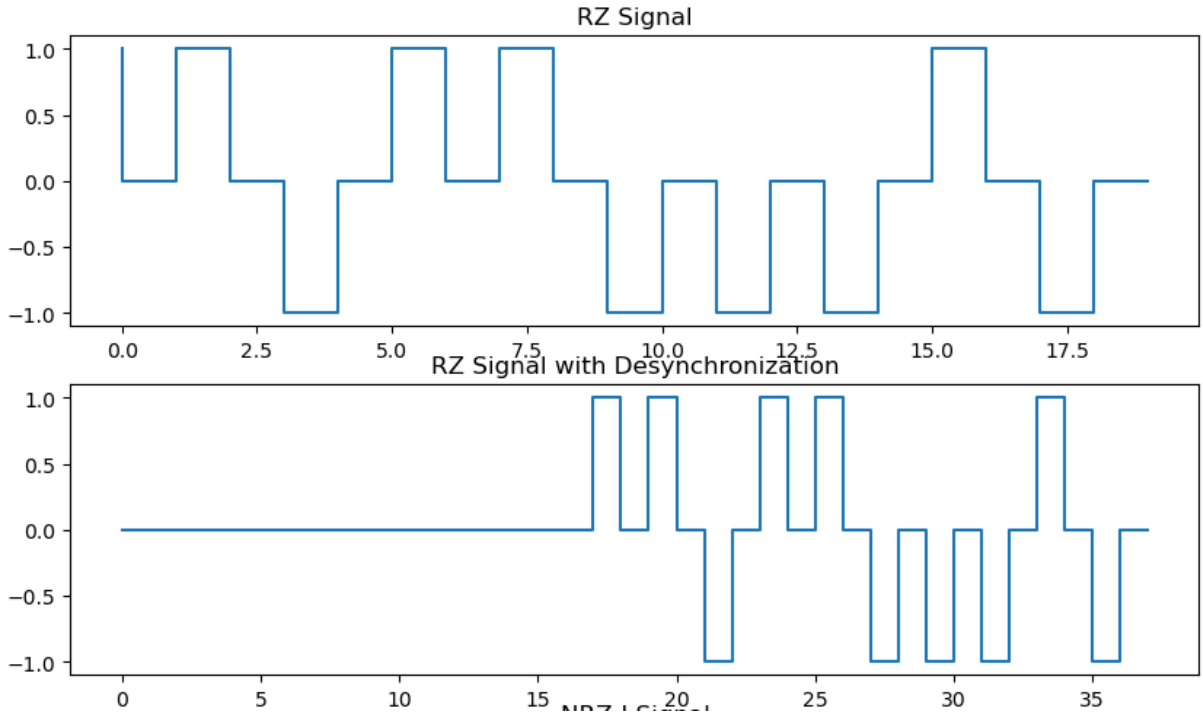
توابعِ تولیدِ پالس توسط تایمر و نیز پیاده‌سازیِ قسمتِ Desynchronization

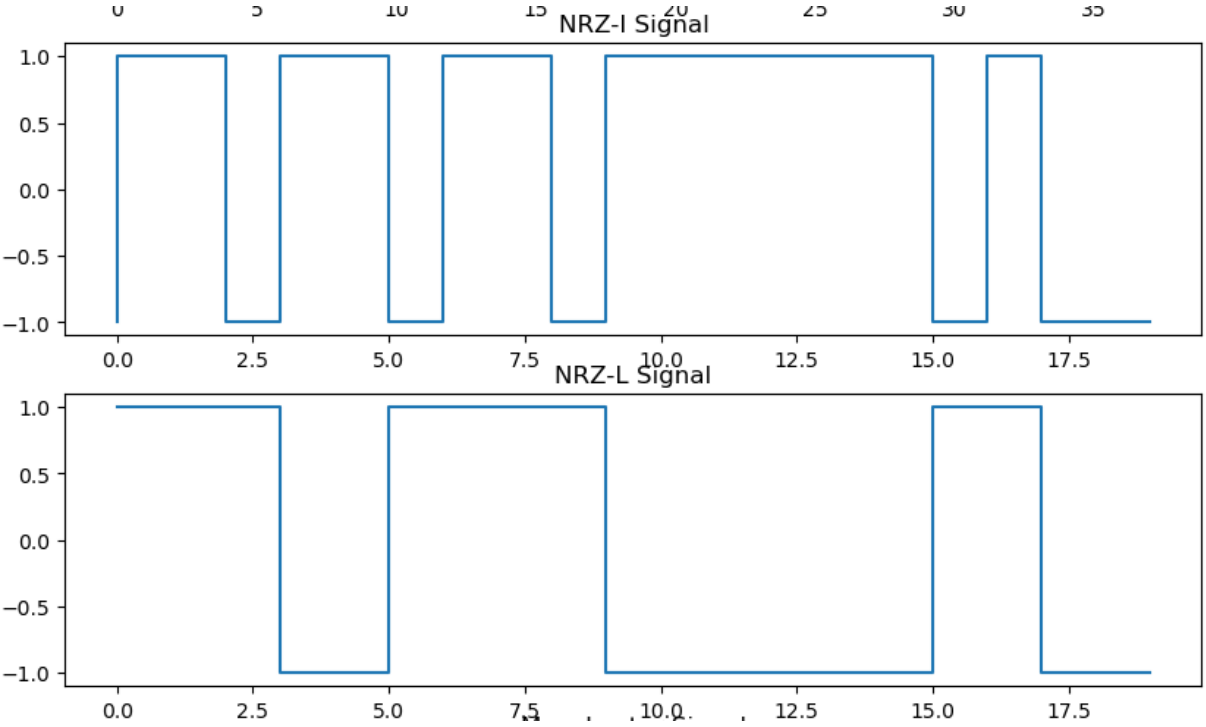
```python
# produce a random time offset for desynchronization
desync_time = random.uniform(0, 1)

# generate example data
data = [random.randint(0, 1) for i in range(10)]

# simulate transmission and reception for each line coding algorithm
rz_signal = send_RZ(data)
rz_signal_desync = [0] * int(desync_time * len(rz_signal)) + rz_signal
rz_data = receive_RZ(rz_signal_desync)

nrz_i_signal = send_NRZ_I(data)
nrz_i_signal_desync = [0] * int(desync_time * len(nrz_i_signal)) + nrz_i_signal
nrz_i_data = receive_NRZ_I(nrz_i_signal_desync)

nrz_l_signal = send_NRZ_L(data)
nrz_l_signal_desync = [0] * int(desync_time * len(nrz_l_signal)) + nrz_l_signal
nrz_l_data = receive_NRZ_L(nrz_l_signal_desync)

manchester_signal = send_manchester(data)
manchester_signal_desync = [0] * int(desync_time * len(manchester_signal)) + manchester_signal
manchester_data = receive_manchester(manchester_signal_desync)

differential_manchester_signal = send_differential_manchester(data)
differential_manchester_signal_desync = [0] * int(desync_time * len(differential_manchester_signal)) +
differential_manchester_signal
differential_manchester_data = receive_differential_manchester(differential_manchester_signal_desync)
```

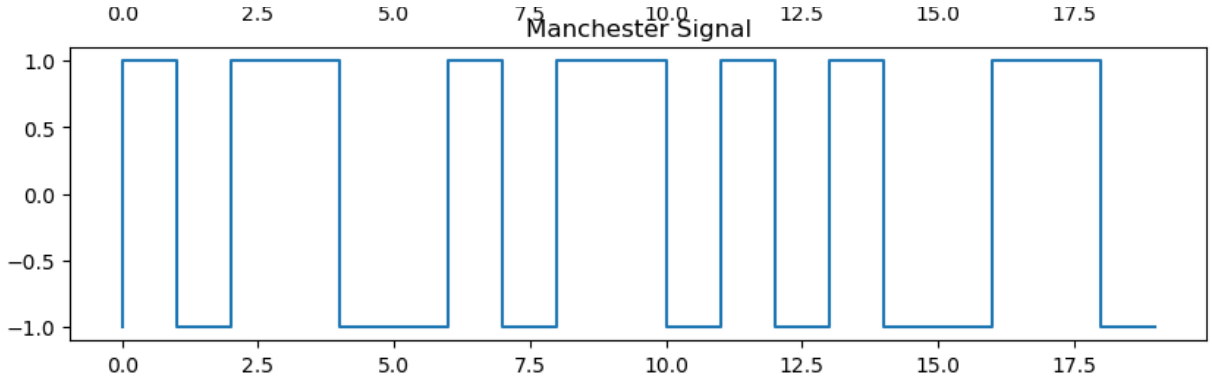
توابعِ مربوط به قسمتِ رمزگشایی یا Decode و رسمِ نمودارها

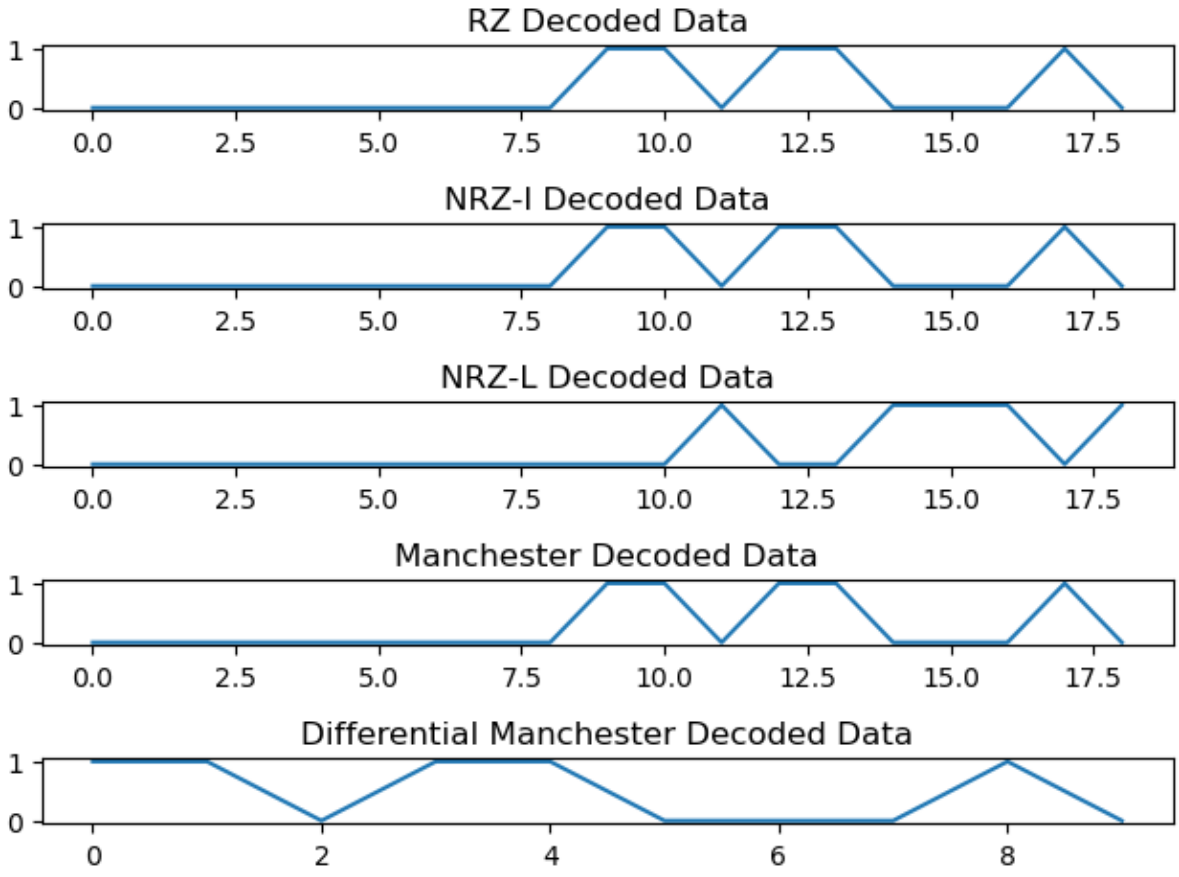
```
# plot the signals and decoded data
fig, axs = plt.subplots(5, 1, figsize=(10, 15))

axs[0].plot(rz_signal, drawstyle='steps-pre')
axs[0].set_title('RZ Signal')
axs[1].plot(rz_signal_desync, drawstyle='steps-pre')
axs[1].set_title('RZ Signal with Desynchronization')
axs[2].plot(nrz_i_signal, drawstyle='steps-pre')
axs[2].set_title('NRZ-I Signal')
axs[3].plot(nrz_l_signal, drawstyle='steps-pre')
axs[3].set_title('NRZ-L Signal')
axs[4].plot(manchester_signal, drawstyle='steps-pre')
axs[4].set_title('Manchester Signal')

fig.suptitle('Line Coding Algorithm Simulation')

plt.figure()

plt.subplot(511)
plt.plot(rz_data)
plt.title('RZ Decoded Data')

plt.subplot(512)
plt.plot(nrz_i_data)
plt.title('NRZ-I Decoded Data')

plt.subplot(513)
plt.plot(nrz_l_data)
plt.title('NRZ-L Decoded Data')

plt.subplot(514)
plt.plot(manchester_data)
plt.title('Manchester Decoded Data')

plt.subplot(515)
plt.plot(differential_manchester_data)
plt.title('Differential Manchester Decoded Data')

plt.tight_layout()
plt.show()
```

در ادامه‌ی این گزارش تصاویرِ نمونه‌ای از اجرای این کدها به‌ازای یک ورودیِ رندُم را بررسی خواهیم کرد...

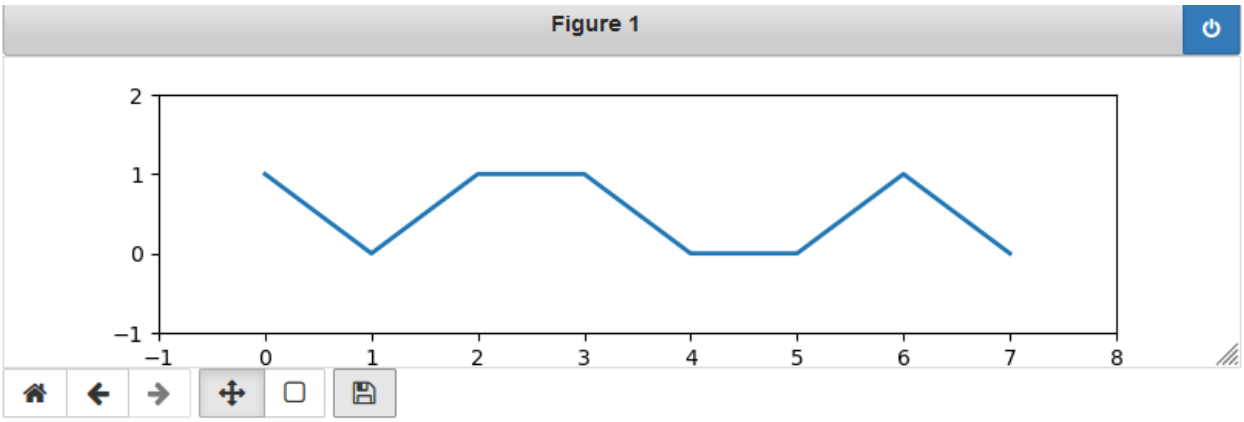

RZ Signal
1.0
0.5
0.0
−0.5
−1.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
RZ Signal with Desynchronization
1.0
0.5
0.0
−0.5
−1.0
0
5
10
15
20
25
30
35


NRZ-I Signal
1.0
0.5
0.0
−0.5
−1.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
NRZ-L Signal
1.0
0.5
0.0
−0.5
−1.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5





با فعال‌سازیِ یک حلقه و برنامه‌ریزیِ ارسالِ جزءبه‌جزء داده‌ها از سمتِ Sender و نیز با افزودنِ کمی تاخیر میانِ فریم‌های نمایش داده‌شده توسطِ کد، می‌توانیم نتایج را به‌صورتِ یک انیمیشن (زنده) به نمایش بگذاریم. این قابلیت در کدهای ضمیمه‌ی این گزارش گنجانده شده، اما متاسفانه بدلیلِ عدمِ امکانِ نمایش آن در فرمِ متنِ یا PDF، در این قسمت صرفا تصاویری از اجرای آن را در محیطِ Jupyter Notebook نشان می‌دهیم.



تصویرِ بالا حاصلِ اجرایِ قطعه‌کدِ زیر است که روندِ نمایشِ این انیمیشن و یا مصورسازیِ زنده‌ی ارسال و دریافتِ کد را برای الگوریتمِ منچستر شبیه‌سازی می‌کند.

```
%matplotlib notebook
import matplotlib.pyplot as plt
import matplotlib.animation as animation

# Define the Manchester encoding function
def manchester_encode(bits):
    encoding = []
    for bit in bits:
        encoding.append((bit, not bit))
    return encoding

# Define the bit sequence to encode
bits = [1, 0, 1, 1, 0, 0, 1, 0]

# Encode the bit sequence using Manchester encoding
encoding = manchester_encode(bits)

# Create the plot
fig, ax = plt.subplots(figsize=(8, 2))
ax.set_xlim(-1, len(encoding))
ax.set_ylim(-1, 2)

# Define the line objects to animate
line, = ax.plot([], [], lw=2)

# Define the initialization function for the animation
def init():
    line.set_data([], [])
    return (line,)

# Define the update function for the animation
def update(frame):
    x = [i for i in range(frame+1)]
    y = [encoding[i][0] for i in range(frame+1)]
    line.set_data(x, y)
    return (line,)

# Create the animation
ani = animation.FuncAnimation(fig, update, frames=len(encoding), init_func=init, blit=True)

# Display the animation
plt.show()
```

مشابهِ این کد برای تمامیِ الگوریتم‌ها نوشته‌شده و به‌کمکِ محیط برنامه‌نویسیِ Jupyter Notebook قابل اجراست. اما به‌جهتِ جلوگیری از طولانی‌تر شدنِ گزارش، در اینجا تنها به ذکرِ این نمونه بسنده خواهم کرد.

تمامیِ کدهای یادشده در فایلِ Optional (برای قسمتِ اختیاریِ پروژه) گنجانده شده و به‌ضمیمه‌ی گزارشِ اصلی در سامانه بارگذاری خواهند شد.

پایان.

برای تهیه و تنظیم قسمت‌های تئوریِ گزارشِ فوق، علاوه‌بر کتابِ مرجع از منابعِ زیر نیز استفاده شده:

i. https://blog.faradars.org/digital-transmission/
ii. https://blog.faradars.org/%D8%AA%DA%A9%D9%86%DB%8C%DA%A9%E2%80%8C%D9%87%D8%A7%DB%8C-%DA%A9%D8%AF%DA%AF%D8%B0%D8%A7%D8%B1%DB%8C-%D8%AF%D8%A7%D8%AF%D9%87-%D8%AF%DB%8C%D8%AC%DB%8C%D8%AA%D8%A7%D9%84/

همچنین برای پیاده‌سازیِ قسمت‌های عملیِ این پروژه – بدونِ کپی کردنِ یا بازنویسیِ مقلدانه‌ی حتی یک خط کد - از منابعِ زیر ایده و الهام گرفته شده:

i. https://github.com/OverPoweredDev/Line-Encoding-Plotter
ii. https://github.com/nur-zaman/digital-signal-visualizer
iii. https://github.com/akanshaAgarwal/Line-Coding-with-JOGL
iv. https://github.com/Sayansurya/Line-Coding
v. https://github.com/ekaksher/Line-Encoder-Decoder
vi. https://github.com/im-zshan/Line-Encoder-and-Scrambler
vii. https://github.com/mishal23/polar-bipolar-line-encoding