

به نام خدا



گزارش پروژه درس مدارهای واسط

استاد:

دکتر امین فصحتی

اعضای گروه:

سهند اسماعیل زاده، شمیم رحیمی، امیرکسری احمدی

دانشگاه صنعتی شریف

پاییز ۱۴۰۳

عنوان پروژه

پیاده‌سازی انکودینگ NRZI و NRZ-L برای RS-232 و RS-422

شرح پروژه

در این پروژه، برنامه‌ای برای برد آردوینو توسعه داده خواهد شد که قابلیت تولید سیگنال‌های سریالی سازگار با پروتکل‌های RS-232 و RS-422 را داشته باشد و از دو روش انکودینگ داده‌ها، یعنی NRZ-L و NRZI، پشتیبانی کند. این پروژه شامل مراحل پیاده‌سازی، شبیه‌سازی، و بررسی عملکرد سیگنال‌های خروجی است تا تفاوت‌ها و مزایای هر روش انکودینگ به‌طور کامل تحلیل شود.

پیاده‌سازی

برای پیاده‌سازی این پروژه نیاز به تعریف فایل‌های متعدد داریم. ابتدا باید الگوریتم‌های Encode و Decode کردن را برای هر دو انکودینگ NRZI و NRZL پیاده کنیم. این تابع‌ها را در فایل NRZEncoding.cpp تعریف می‌کنیم.

۱. NRZEncoding.cpp:

تابع اول مربوط به انکودینگ NRZI است:

```
void encodeNRZI(byte* data, int length) {
    static bool lastBit = LOW;
    for (int i = 0; i < length; i++) {
        byte encodedByte = 0;
        for (int bit = 7; bit >= 0; bit--) {
            bool bitVal = (data[i] >> bit) & 0x01;
            lastBit = bitVal ? !lastBit : lastBit; // Toggle only on 1s
            encodedByte |= (lastBit << bit);
        }
        data[i] = encodedByte; // Store encoded value
    }
}
```

این تابع یک رشته‌ی ورودی و یک متغیر که نشان‌دهنده‌ی طول رشته به بایت است ورودی می‌گیرد. یک متغیر boolean تعریف می‌کنیم تا بتوانیم ۱ یا ۰ بودن آخرین بیت را داشته باشیم. حال باید هر بایت را

بیت به بیت انکود کنیم. به ازای هر بایت که متشکل از ۸ بیت است، بایت مربوطه را به اندازه‌ای شیفت می‌دهیم تا بیتی که می‌خواهیم انکود کنیم در LSB حاضر شود، سپس با AND کردن آن بایت با ۱، بیت مورد نظر را استخراج می‌کنیم. برای انکود کردن NRZI، اگر بیت مورد نظر ۱ باشد، toggle می‌کنیم و در غیر این صورت تغییری ایجاد نمی‌کنیم. در انتها بیت مورد نظر را به اندازه‌ای شیفت می‌دهیم تا به جایگاه خودش بازگردد.

تابع دوم مربوط به انکود کردن NRZL است:

```
void encodeNRZL(byte* data, int length) {
    for (int i = 0; i < length; i++) {
        byte encodedByte = 0;
        for (int bit = 7; bit >= 0; bit--) {
            bool bitVal = (data[i] >> bit) & 0x01;
            bitVal = !bitVal; // Invert bits
            encodedByte |= (bitVal << bit); // Directly map 0 -> LOW, 1 -> HIGH
        }
        data[i] = encodedByte; // Store encoded value
    }
}
```

این تابع هم یک رشته‌ی ورودی و یک متغیر که نشان‌دهنده‌ی طول رشته به بایت است ورودی می‌گیرد. در این تابع، استخراج کردن بیت مورد نظر برای انکود کردن مانند تابع قبلی است بنابراین از نوشتن توضیحات اضافه خودداری می‌کنیم. پس از استخراج بیت مورد نظر، باید آن را طبق قوانین NRZL انکود کنیم. NRZL بیت ۰ را با HIGH و بیت ۱ را با LOW نمایش می‌دهد. پس فقط کافی است بیت مورد نظر not کنیم. در انتها هم بیت انکود شده را شیفت می‌دهیم تا به جایگاه خودش در بایت مورد نظر بازگردد.

تابع سوم مربوط به دیکود کردن NRZI است:

```
void decodeNRZI(byte* data, int length) {
    static bool lastBit = LOW;
    for (int i = 0; i < length; i++) {
        byte decodedByte = 0;
        for (int bit = 7; bit >= 0; bit--) {
            bool bitVal = (data[i] >> bit) & 0x01;
            bool originalBit = (bitVal == lastBit) ? 0 : 1; // If no change, it was 0; if toggled, it was 1
            lastBit = bitVal; // Update last bit
            decodedByte |= (originalBit << bit);
        }

        data[i] = decodedByte; // Store decoded value
    }
}
```

این تابع هم یک رشته‌ی ورودی و یک متغیر که نشان‌دهنده‌ی طول رشته به بایت است ورودی می‌گیرد. در اینجا یک رشته از بیت‌های انکود شده داریم که نیاز داریم بیت‌های اولیه را از آنها استخراج کنیم. مراحل ابتدایی مانند تابع انکود کردن است. پس از اینکه بیت مورد نظر را استخراج کردیم، چک می‌کنیم، اگر این بیت با بیت قبلی تفاوت داشت یعنی بیت اولیه ۱ بوده است و در غیر این صورت یعنی در صورت تغییر نکردن بیت دریافتی، یعنی بیت اولیه ۰ بوده است. به این صورت بیت واقعی را استخراج می‌کنیم و آن را به جایگاه اصلی خودش در داده بازمی‌گردانیم.

تابع آخر مربوط به دیکود کردن NRZL است:

```
void decodeNRZL(byte* data, int length) {
    for (int i = 0; i < length; i++) {
        data[i] = ~data[i]; // Simply invert the bits back
    }
}
```

در این تابع، صرفاً باید هر بیت را not کنیم تا داده‌ی اصلی استخراج شود.

حال برای اینکه این توابع را در کتابخانه تعریف کنیم، یک فایل NRZEncoding.h می‌سازیم تا headerهای این توابع را در آن قرار دهیم. همین‌طور این فایل جدید را در ابتدای NRZEncoding.cpp نیز

include می‌کنیم:

```
#include "NRZEncoding.h"
```

فایل NRZEncoding.h را به این صورت تعریف می‌کنیم تا تعریف توابع و نوع ورودی و خروجی‌های آنها مشخص شوند:

```
#ifndef NRZ_ENCODING_H
#define NRZ_ENCODING_H

#include <Arduino.h>

void encodeNRZI(byte* data, int length);
void encodeNRZL(byte* data, int length);
void decodeNRZI(byte* data, int length);
void decodeNRZL(byte* data, int length);

#endif // NRZ_ENCODING_H
```

در انتها کد مربوط به ارسال و دریافت داده در برد آردوینو در فایل main.ino را می‌نویسم و در آن از کتابخانه‌ی نوشته شده استفاده می‌کنیم.

ابتدا در فایل main.ino کتابخانه‌ای که در بخش قبل ساختیم را include می‌کنیم:

```
#include "NRZEncoding.h"
```

سپس در تابع setup که تنها یک بار اجرا می‌شود، نرخ انتقال داده (baud rate) پورت‌های سریال ورودی/خروجی که می‌خواهیم استفاده کنیم، مقداردهی می‌شود تا بتوانیم از آنها برای برقراری ارتباط با یکدیگر بهره‌برداری کنیم. سریال با پورت شماره‌ی صفر تنها برای debug و تست عملکرد مقداردهی می‌شود. در این کد، randomSeed برای تنظیم مقدار اولیه تولید اعداد تصادفی استفاده می‌شود. این

مقدار از خواندن سیگنال آنالوگ پین A0 گرفته می‌شود که معمولاً تصادفی است، به همین دلیل اعداد تصادفی تولید شده هر بار متفاوت خواهند بود.

```
void setup() {  
    Serial.begin(9600);  
    Serial1.begin(9600);  
    Serial2.begin(9600);  
    randomSeed(analogRead(A0));  
}
```

همچنین یک متغیر global تعریف کردیم که مشخص می‌کند انکودینگ مورد استفاده به صورت NRZI است یا NRZL.

```
static bool useNRZI = false;
```

حال به بررسی تابع loop می‌پردازیم که به‌طور مداوم اجرا می‌شود. در ابتدا یک متغیر به نام lastSend به صورت استاتیک تعریف می‌کنیم که تنها یک‌بار مقداردهی صفر را انجام می‌دهد. این متغیر برای ذخیره زمان آخرین ارسال داده استفاده می‌شود، به‌طوری‌که داده‌ها هر یک ثانیه یک‌بار ارسال شوند. همچنین یک متغیر به نام turn به صورت مشابه تعریف می‌شود تا مشخص کند کدام یک از پورت‌ها قصد ارسال داده دارد. در صورتی که یک ثانیه گذشته باشد، قصد داریم داده‌ی جدیدی ارسال کنیم که این موضوع با استفاده از یک دستور if بررسی می‌شود. در داخل این if، متغیر lastSend به‌روزرسانی شده و یک عدد تصادفی جدید برای ارسال تولید می‌شود.

```
void loop() {  
    static unsigned long lastSend = 0;  
    static bool turn = false;  
  
    if (millis() - lastSend >= 1000) {  
        lastSend = millis();  
        byte data = random(256);
```

در این بخش، آماده‌ی ارسال داده‌ها می‌شویم و برای اطمینان از درستی داده‌ها و انکودینگ آن‌ها، از پورت ۰ با استفاده از دستورهای Serial اقدام به چاپ عبارت‌هایی برای بررسی صحت داده‌ها می‌کنیم. سپس در یک دستور if بررسی می‌کنیم که باید از کدام انکودینگ استفاده کنیم و برای هرکدام، با فراخوانی API مربوطه، داده‌ها را به شکل موردنظر انکود می‌کنیم. سپس با توجه به متغیر turn تصمیم می‌گیریم کدام یک از پورت‌ها باید داده را ارسال نماید. با توجه به شماره‌ی پورت از Serial#.write استفاده می‌کنیم و داده‌ی انکود شده را به شکل سریالی ارسال می‌کنیم. همچنین در نهایت turn را عوض می‌کنیم تا در نوبت بعد از طریق پورت دیگر داده ارسال شود.

```
Serial.print("Original: ");
Serial.println(data, BIN);

if (useNRZI) {
    encodeNRZI(&data, 1);
    Serial.print("NRZI Encoded: ");
} else {
    encodeNRZL(&data, 1);
    Serial.print("NRZL Encoded: ");
}
Serial.println(data, BIN);

if (!turn) {
    Serial1.write(data);
    Serial.print("Sent from 1: ");
} else {
    Serial2.write(data);
    Serial.print("Sent from 2: ");
}
Serial.println(data, BIN);
turn = !turn;
}
```

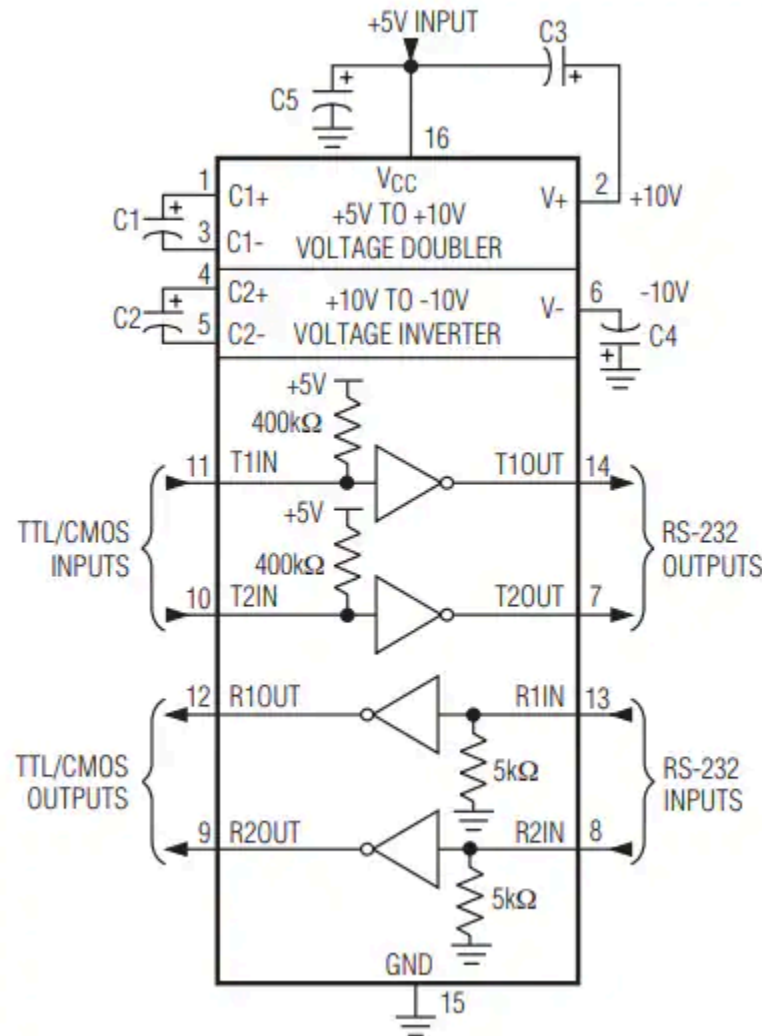

در انتهای این کد از دو تابع SerialEvent1 و SerialEvent2 استفاده می‌کنیم که در صورتی که تغییری در پورتهای ورودی آنها رخ دهد، فراخوانی می‌شوند. درون این تابع‌ها بررسی می‌کنیم که آیا داده‌ی جدیدی روی لینک قرار گرفته است یا خیر و در صورتی که داده‌ی جدیدی موجود باشد، آن را دریافت کرده و دیکود می‌کنیم.

```
void serialEvent1() {
    while (Serial1.available()) {
        byte receivedData = Serial1.read();
        Serial.print("Received on Serial1 (Encoded): ");
        Serial.println(receivedData, BIN);

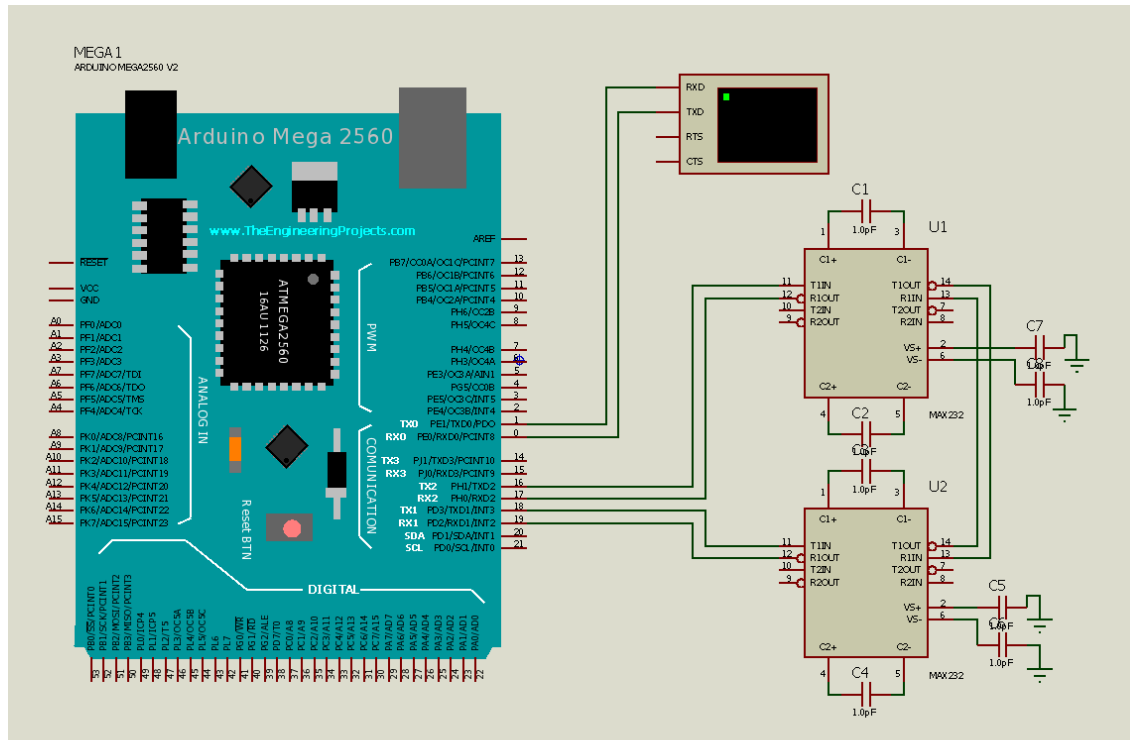
        byte decodedData = receivedData;
        if (useNRZI) {
            decodeNRZI(&decodedData, 1);
            Serial.print("Decoded (NRZI): ");
        } else {
            decodeNRZL(&decodedData, 1);
            Serial.print("Decoded (NRZL): ");
        }
        Serial.println(decodedData, BIN);
    }
}
```

شبیه‌سازی پروتئوس

در این بخش پروژه را درون پروتئوس شبیه‌سازی کردیم. برای شبیه‌سازی این پروژه نیاز به یک برد arduino mega، دو عدد ic max232 و ۸ عدد خازن داریم. در ابتدا دیتا شیت ic max232 را بررسی می‌کنیم که آن را می‌توانید در تصویر زیر ببینید.



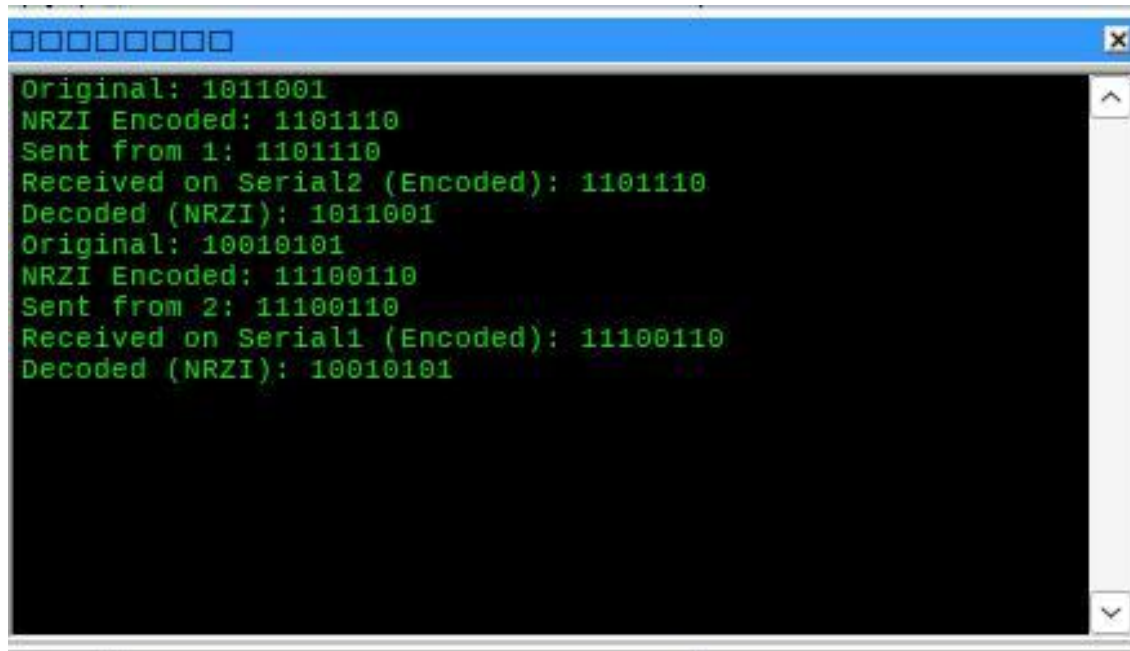
همان‌طور که مشاهده می‌کنید، در این مدار باید خروجی TX از برد را به T1IN این قطعه وصل کنیم و همچنین R1OUT آن را به ورودی RX برد متصل نماییم. علاوه بر این، باید T1OUT یکی از تراشه‌های MAX232 را به R1IN دیگری متصل کنیم و بالعکس. این پروژه با استفاده از دو پورت ورودی/خروجی برد آردوینو مگا شبیه‌سازی شده است، به گونه‌ای که مشابه ارتباط سریال بین دو دستگاه مجزا عمل می‌کند. می‌توانید شبیه‌سازی آن را در تصویر زیر مشاهده نمایید.



مدار را با کدهای نوشته شده تست می‌کنیم. قسمت virtual monitor را بررسی می‌کنیم و داده‌های ارسالی و دریافتی را مشاهده می‌کنیم:

```
Original: 1011001
NRZL Encoded: 10100110
Sent from 1: 10100110
Received on Serial2 (Encoded): 10100110
Decoded (NRZL): 1011001
Original: 10010101
NRZL Encoded: 1101010
Sent from 2: 1101010
Received on Serial1 (Encoded): 1101010
Decoded (NRZL): 10010101
Original: 101011
NRZL Encoded: 11010100
Sent from 1: 11010100
Received on Serial2 (Encoded): 11010100
Decoded (NRZL): 101011
```

این بخش مربوط به تست NRZL است. بیت اول داده (MSB) صفر بوده است به همین دلیل نمایش داده نشده است. مشاهده می‌شود که داده‌ی انکود شده نات بیت‌های اصلی است و به درستی انکود شده است. همچنین این داده به درستی روی پورت مربوطه ارسال شده است و در انتها نیز دیکود شده است.



```
Original: 1011001
NRZI Encoded: 1101110
Sent from 1: 1101110
Received on Serial2 (Encoded): 1101110
Decoded (NRZI): 1011001
Original: 10010101
NRZI Encoded: 11100110
Sent from 2: 11100110
Received on Serial1 (Encoded): 11100110
Decoded (NRZI): 10010101
```

این بخش مربوط به تست NRZI است. جاهایی که داده‌ی ورودی ۱ می‌شود، داده‌ی انکود شده تغییر بیت داده است. مثل قسمت قبلی به درستی ارسال و دیکود شده است.

پیاده‌سازی مدار فیزیکی

برای اجرای پروژه به صورت عملی، مدار فیزیکی مطابق شبیه‌سازی پروتئوس طراحی و اجرا کردیم. در این بخش، اجزای مدار، نحوه اتصال آن‌ها، و نتایج عملی را بررسی می‌کنیم.

اجزای مورد نیاز

- برد آردوینو مگا ۲۵۶۰
- دو عدد آی‌سی MAX232
- هشت عدد خازن ۱ میکروفاراد
- سیم‌های اتصال و برد بورد
- کابل‌های سریال برای ارتباط با رایانه
- مولتی‌متر برای بررسی سیگنال‌های خروجی

نحوه اتصال مدار

۱. اتصال آردوینو به MAX232

- پایه TX آردوینو را به پایه T1IN آی‌سی اول MAX232 متصل کردیم.
- پایه R1OUT آی‌سی را به RX آردوینو متصل کردیم.
- پایه T1OUT آی‌سی اول را به R1IN آی‌سی دوم متصل کردیم.
- پایه R1OUT آی‌سی دوم را به RX پورت سریال دوم آردوینو متصل کردیم.

۲. اتصال خازن‌ها

- چهار خازن ۱ میکروفاراد را برای تثبیت ولتاژ و عملکرد صحیح MAX232 مطابق با دیتاشیت آن متصل کردیم.

۳. تأمین تغذیه مدار

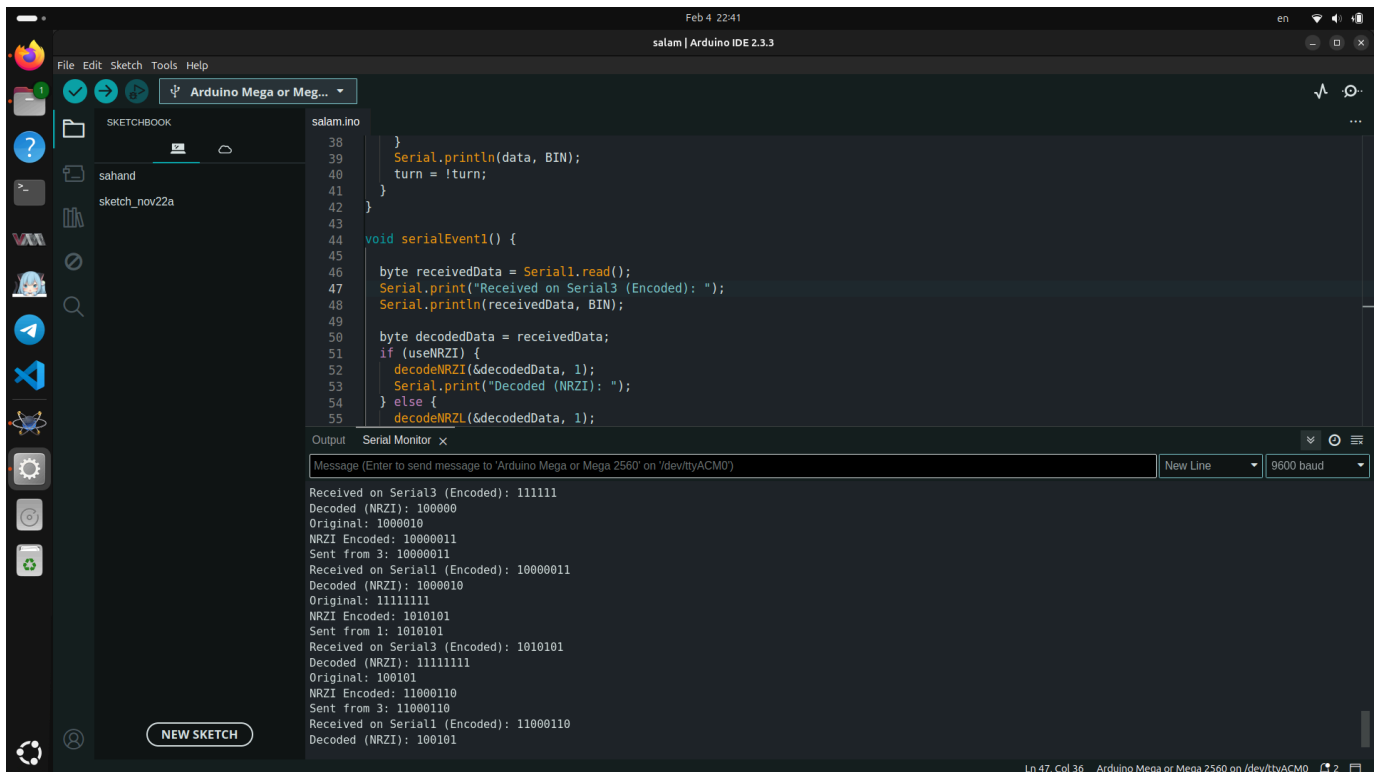
- تغذیه ۵ ولت را از آردوینو به آی‌سی‌های MAX232 متصل کردیم.
- اتصال زمین مشترک را بین تمامی قطعات برقرار کردیم.
- تست و بررسی عملکرد مدار:
- پس از اتصال مدار و آپلود کدهای نوشته شده روی آردوینو، داده‌های انکود شده را از طریق پورت سریال ارسال کردیم.
- برای بررسی صحت عملکرد، از نرم‌افزار Serial Monitor آردوینو استفاده کردیم.

- در حالت NRZ-L: مشاهده کردیم که بیت‌های صفر به سطح HIGH و بیت‌های یک به سطح LOW تبدیل شدند.
- در حالت NRZI: مشاهده کردیم که در صورت وجود یک، تغییر وضعیت در سطح سیگنال رخ داد، در حالی که برای مقدار صفر سطح سیگنال بدون تغییر باقی ماند.

نتایج عملی و تحلیل داده‌ها

- داده‌های ارسال‌شده از یک پورت سریال با موفقیت توسط پورت دیگر دریافت و دیکود شدند.
- تأخیر و نویز در سیگنال‌ها در محدوده‌ی قابل قبول برای ارتباط سریال استاندارد RS-232 و RS-422 قرار داشت.
- آزمایش‌های عملی نشان داد که MAX232 در تبدیل سطوح ولتاژ بین RS-232 و TTL به‌خوبی عمل می‌کند و ارتباط بین دستگاه‌ها پایدار است.

تصاویر



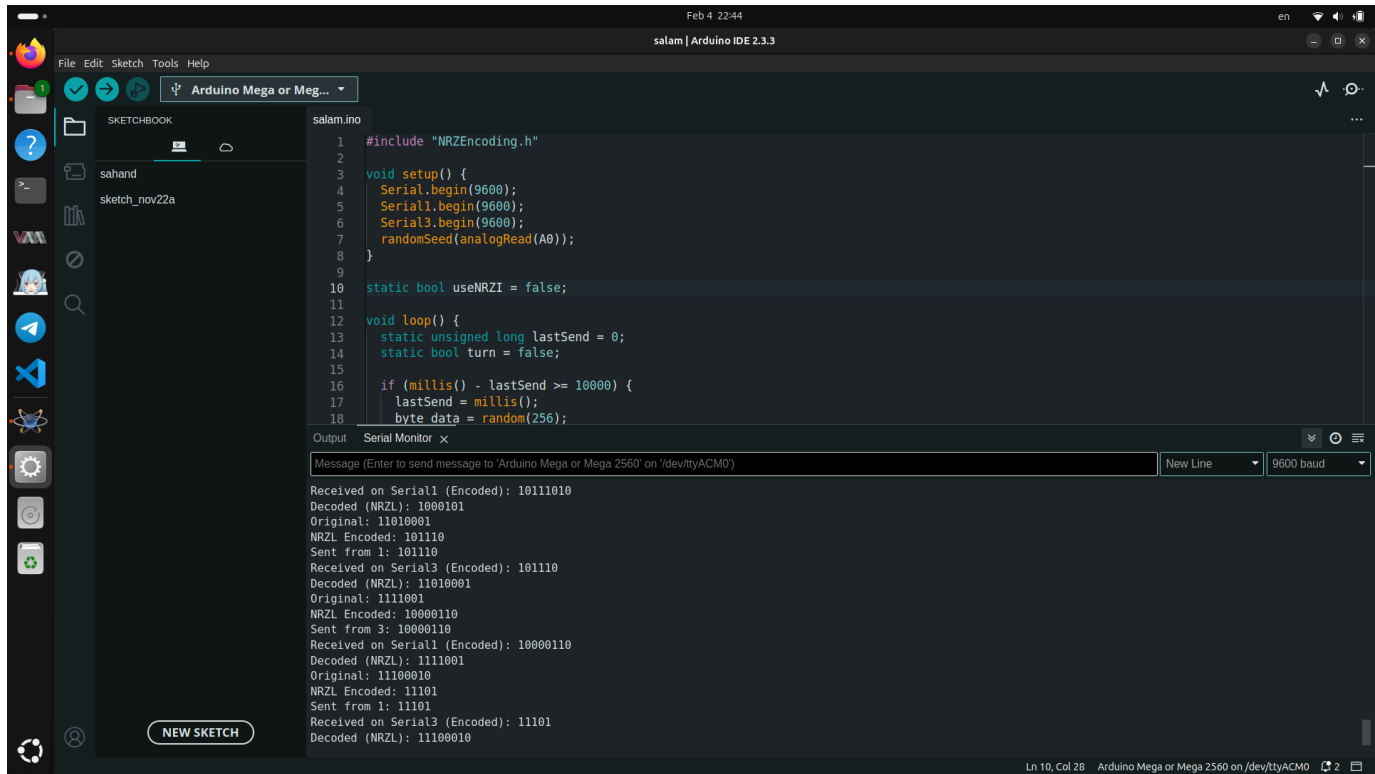
The screenshot shows the Arduino IDE 2.3.3 interface. The sketch 'salam.ino' is open, displaying the following code:

```
38  
39 Serial.println(data, BIN);  
40 turn = !turn;  
41 }  
42 }  
43  
44 void serialEvent1() {  
45  
46   byte receivedData = Serial1.read();  
47   Serial.print("Received on Serial3 (Encoded): ");  
48   Serial.println(receivedData, BIN);  
49  
50   byte decodedData = receivedData;  
51   if (useNRZI) {  
52     decodeNRZI(&decodedData, 1);  
53     Serial.print("Decoded (NRZI): ");  
54   } else {  
55     decodeNRZL(&decodedData, 1);
```

The Serial Monitor output shows the following data:

```
Received on Serial3 (Encoded): 111111  
Decoded (NRZI): 100000  
Original: 1000010  
NRZI Encoded: 10000011  
Sent from 3: 10000011  
Received on Serial1 (Encoded): 10000011  
Decoded (NRZI): 1000010  
Original: 11111111  
NRZI Encoded: 1010101  
Sent from 1: 1010101  
Received on Serial3 (Encoded): 1010101  
Decoded (NRZI): 11111111  
Original: 100101  
NRZI Encoded: 11000110  
Sent from 3: 11000110  
Received on Serial1 (Encoded): 11000110  
Decoded (NRZI): 100101
```

اجرای صحیح کد با انکودینگ NRZI بر روی برد فیزیکی



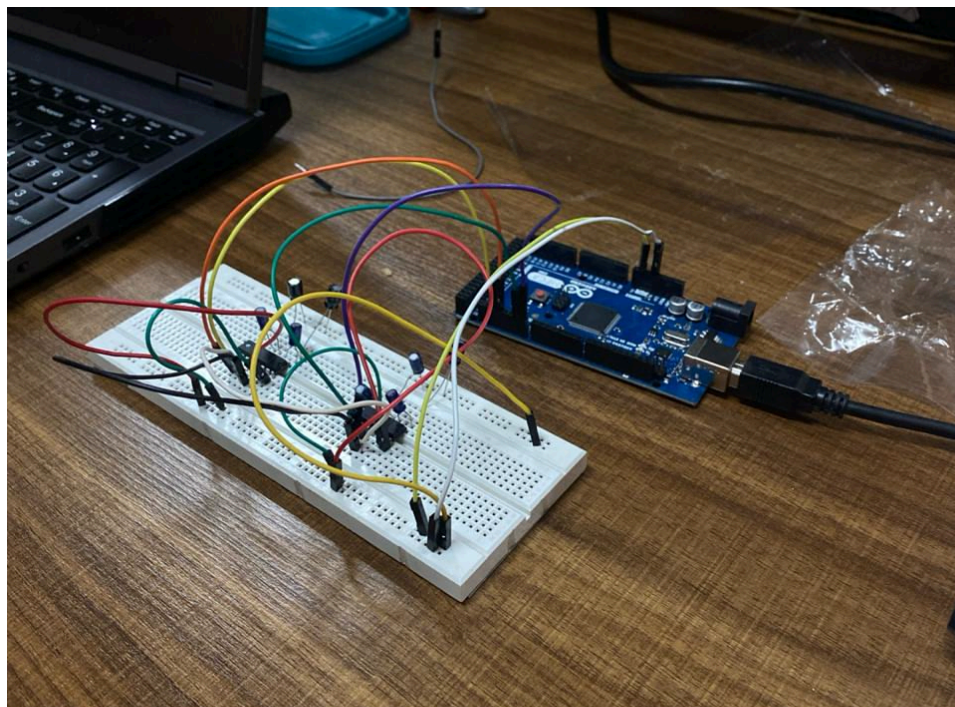
```
1 #include "NRZEncoding.h"
2
3 void setup() {
4   Serial.begin(9600);
5   Serial1.begin(9600);
6   Serial3.begin(9600);
7   randomSeed(analogRead(A0));
8 }
9
10 static bool useNRZI = false;
11
12 void loop() {
13   static unsigned long lastSend = 0;
14   static bool turn = false;
15
16   if (millis() - lastSend >= 10000) {
17     lastSend = millis();
18     byte_data = random(256);
19
20     if (turn) {
21       Serial1.write(byte_data);
22       Serial3.write(byte_data);
23       Serial.write(byte_data);
24       turn = false;
25     } else {
26       Serial1.write(~byte_data);
27       Serial3.write(~byte_data);
28       Serial.write(~byte_data);
29       turn = true;
30     }
31   }
32 }
```

Output Serial Monitor

Message (Enter to send message to 'Arduino Mega or Mega 2560' on '/dev/ttyACM0')

Received on Serial1 (Encoded): 10111010
Decoded (NRZL): 1000101
Original: 11010001
NRZL Encoded: 101110
Sent from 1: 101110
Received on Serial3 (Encoded): 101110
Decoded (NRZL): 11010001
Original: 1111001
NRZL Encoded: 10000110
Sent from 3: 10000110
Received on Serial1 (Encoded): 10000110
Decoded (NRZL): 1111001
Original: 11100010
NRZL Encoded: 11101
Sent from 1: 11101
Received on Serial3 (Encoded): 11101
Decoded (NRZL): 11100010

اجرای صحیح کد با انکودینگ NRZL بر روی برد فیزیکی



اتصالات فیزیکی و برد



جمع‌بندی

- پیاده‌سازی مدار فیزیکی با موفقیت انجام شد و داده‌ها با استفاده از انکودینگ‌های NRZ-L و NRZI ارسال و دریافت شدند.
- عملکرد مدار مطابق انتظار بود و تفاوت‌های هر دو روش انکودینگ را به‌صورت عملی بررسی کردیم.
- این پروژه نشان داد که می‌توانیم با استفاده از آردوینو و آی‌سی MAX232، سیگنال‌های سریال را مطابق با استانداردهای RS-232 و RS-422 تولید و پردازش کنیم.