گزارش پروژه مدارهای واسط

محمد شفیع‌زاده – ۴۰۱۱۱۰۳۸۶

سیدمحمدپویان شمس‌الدین - ۴۰۱۱۱۰۸۱۲

محمدفرحان بهرامی

در این گزارش قصد داریم درمورد نحوه کارکرد انکودینگ 128/130 و 5/6 صحبت کنیم. همچنین به توضیح الگوریتم CRC8 می‌پردازیم. در ادامه کدی که برای پیاده‌سازی زدیم را بهمراه خروجی شرح می‌دهیم. در نهایت در مورد تصحیح خطا و روش‌های آن صحبت می‌کنیم.

توضیح انکودینگ ۱۳۰/۱۲۸ و ٦/٥

نحوه کارکرد انکودینگ ۱۳۰/۱۲۸

انکودینگ ۱۳۰/۱۲۸ یک تکنیک کدگذاری خطی است که در سیستم‌های مخابراتی و انتقال داده برای بهبود یکپارچگی داده‌ها و کاهش احتمال خطا استفاده می‌شود. در این روش، هر ۱۲۸ بیت داده ورودی به ۱۳۰ بیت داده خروجی تبدیل می‌شود، که این افزایش بیت‌ها به منظور افزایش قابلیت شناسایی و تصحیح خطا انجام می‌گیرد.

اکنون به نحوه عملکرد انکودینگ می‌پردازیم. ابتدا ۱۲۸ بیت داده دریافت می‌شود. فرآیند کدگذاری انجام می‌شود که شامل افزودن ۲ بیت اضافی برای کنترل خطا و تضمین تعادل سیگنال است. داده کدگذاری‌شده به ۱۳۰ بیت تبدیل شده و ارسال می‌شود. در گیرنده، عملیات دیکدینگ انجام می‌شود و ۱۲۸ بیت اصلی بازسازی می‌شود.

نحوه کارکرد انکودینگ ۶/۵

انکودینگ ۶/۵ یکی دیگر از روش‌های کدگذاری است که در سیستم‌های انتقال داده استفاده می‌شود. در این روش، هر ۵ بیت داده ورودی به ۶ بیت داده خروجی تبدیل می‌شود. با تبدیل ۵ بیت به ۶ بیت، میزان اطلاعات افزوده افزایش می‌یابد که کمک می‌کند خطاهای احتمالی شناسایی و تصحیح شوند. یکی از مزایای این انگودینگ حذف توالی‌های طولانی صفر یا یک است. بسیاری از سیستم‌های انتقال داده، وجود توالی‌های طولانی از صفر یا یک باعث مشکلات همگام‌سازی می‌شود. این روش به کاهش این مشکل کمک می‌کند.

برای ارسال ابتدا داده‌ها به بلوک‌های ۵ بیتی تقسیم می‌شوند. این نحوه تبدیل ۵ بیت به ۶ بیت از طریق یک جدول تبدیل ثابت که در کد هم آوردیم انجام میشود.

کاربرد این دو انکودینگ

انکودینگ ۱۳۰/۱۲۸ بیشتر در پروتکل‌های سریالی پرسرعت مانند PCI Express، Ethernet و برخی از سیستم‌های ارتباطی نوری استفاده می‌شود. انکودینگ ۶/۵ در سیستم‌های ارتباطی مانند DVB (پخش دیجیتال) و برخی از استانداردهای مخابراتی بی‌سیم برای بهبود پایداری داده‌ها و تصحیح خطا به کار می‌رود.

الگوریتم CRC8

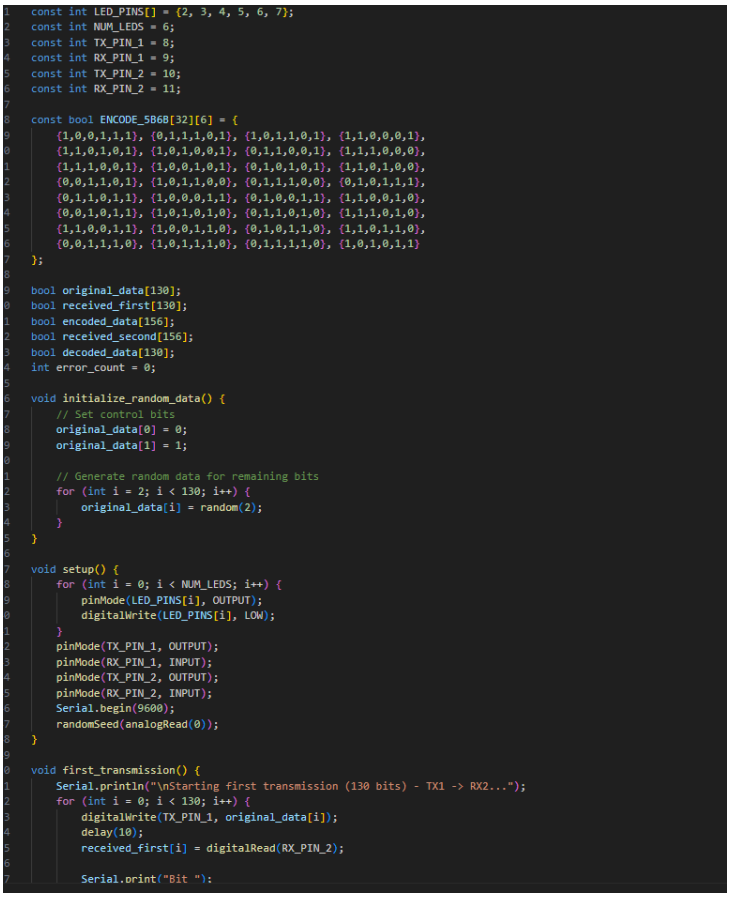
الگوریتم CRC یک روش بررسی خطا در داده‌های دیجیتال است که برای تشخیص تغییرات ناخواسته در داده‌های منتقل‌شده یا ذخیره‌شده استفاده می‌شود. CRC8 نوعی از این الگوریتم است که از یک مقدار ۸ بیتی به عنوان چک‌سام برای شناسایی خطاها استفاده می‌کند. الگوریتم Cyclic Redundancy Check **CRC8** با ۸ بیت یک روش بررسی خطا در داده‌های دیجیتال است که از تقسیم دودویی (بدون باقی‌مانده) استفاده می‌کند. در این الگوریتم، داده‌ی اصلی با یک چندجمله‌ای مولد (معمولاً به‌صورت یک مقدار هگزادسیمال مثل 0x07 تقسیم می‌شود و باقی‌مانده این تقسیم به‌عنوان **کد CRC** ذخیره می‌شود. هنگام دریافت داده، گیرنده همان عملیات تقسیم را انجام می‌دهد و اگر باقی‌مانده صفر باشد، داده صحیح فرض می‌شود؛ در غیر این صورت، خطا رخ داده است.

نحوه پیاده‌سازی و خروجی کد

کد زده شده به صورت زیر است:

این کد آردوینو شامل توابعی است که برای ارسال و دریافت داده‌ها با استفاده از پروتکل 5B/6B طراحی شده‌اند. تابع initialize_random_data() داده‌های تصادفی را برای ارسال آماده می‌کند. در ()setup، پین‌ها تنظیم و ارتباط سریال آغاز می‌شود. first transmission داده‌های اولیه را به RX2 ارسال کرده و آن‌ها را دریافت می‌کند. سپس encode_received_data() داده‌های دریافتی را به فرمت B/6B۵ رمزگذاری می‌کند. second_transmission() داده‌های رمزگذاری شده را به RX1 ارسال و دریافت می‌کند. تابع find_5bit_value() برای جستجوی مقدار ۵ بیتی از یک توالی ۶ بیتی استفاده می‌شود. در decode_and_display() داده‌های دریافتی رمزگشایی شده و بر روی LEDها نمایش داده می‌شوند. در نهایت، check_errors() برای مقایسه داده‌های اصلی و رمزگشایی شده و شناسایی خطاها به کار می‌رود. حلقه loop() به طور این مراحل را تکرار کرده و وضعیت انتقال و خطاها را گزارش می‌کند.

بخشی از کد زده شده: (فایل کامل کد قرار داده شده)



```
const int LED_PINS[] = {2, 3, 4, 5, 6, 7};
const int NUM_LEDS = 6;
const int TX_PIN_1 = 8;
const int RX_PIN_1 = 9;
const int TX_PIN_2 = 10;
const int RX_PIN_2 = 11;

const bool ENCODE_5B6B[32][6] = {
    {1,0,0,1,1,1}, {0,1,1,1,0,1}, {1,0,1,1,0,1}, {1,1,0,0,0,1},
    {1,1,0,1,0,1}, {1,0,1,0,0,1}, {0,1,1,0,0,1}, {1,1,1,0,0,0},
    {1,1,1,0,0,1}, {1,0,0,1,0,1}, {0,1,0,1,0,1}, {1,1,0,1,0,0},
    {0,0,1,1,0,1}, {1,0,1,1,0,0}, {0,1,1,1,0,0}, {0,1,0,1,1,1},
    {0,1,1,0,1,1}, {1,0,0,0,1,1}, {0,1,0,0,1,1}, {1,1,0,0,1,0},
    {0,0,1,0,1,1}, {1,0,1,0,1,0}, {0,1,1,0,1,0}, {1,1,1,0,1,0},
    {1,1,0,0,1,1}, {1,0,0,1,1,0}, {0,1,0,1,1,0}, {1,1,0,1,1,0},
    {0,0,1,1,1,0}, {1,0,1,1,1,0}, {0,1,1,1,1,0}, {1,0,1,0,1,1}
};

bool original_data[130];
bool received_first[130];
bool encoded_data[156];
bool received_second[156];
bool decoded_data[130];
int error_count = 0;

void initialize_random_data() {
    // Set control bits
    original_data[0] = 0;
    original_data[1] = 1;

    // Generate random data for remaining bits
    for (int i = 2; i < 130; i++) {
        original_data[i] = random(2);
    }
}

void setup() {
    for (int i = 0; i < NUM_LEDS; i++) {
        pinMode(LED_PINS[i], OUTPUT);
        digitalWrite(LED_PINS[i], LOW);
    }
    pinMode(TX_PIN_1, OUTPUT);
    pinMode(RX_PIN_1, INPUT);
    pinMode(TX_PIN_2, OUTPUT);
    pinMode(RX_PIN_2, INPUT);
    Serial.begin(9600);
    randomSeed(analogRead(0));
}

void first_transmission() {
    Serial.println("\nStarting first transmission (130 bits) - TX1 -> RX2...");
    for (int i = 0; i < 130; i++) {
        digitalWrite(TX_PIN_1, original_data[i]);
        delay(10);
        received_first[i] = digitalRead(RX_PIN_2);

        Serial.print("Bit ");
```

به ترتیب خروجی برنامه را نشان و هرکدام را توضیح میدهیم:

خروجی اول: (first transmission)

یک انتقال داده کامل و بدون خطا از TX1 به RX2 را نشان می دهد، با هر بیت که به درستی ارسال و دریافت شده است.
تمام ۱۳۰ بیت به درستی ارسال و دریافت شده اند

```
Starting first transmission (130 bits) - TX1 -> RX2...
Bit 0: Sent 0, Received 0
Bit 1: Sent 1, Received 1
Bit 2: Sent 0, Received 0
Bit 3: Sent 1, Received 1
Bit 4: Sent 0, Received 0
Bit 5: Sent 1, Received 1
Bit 6: Sent 0, Received 0
Bit 7: Sent 1, Received 1
Bit 8: Sent 0, Received 0
Bit 9: Sent 0, Received 0
Bit 10: Sent 0, Received 0
Bit 11: Sent 0, Received 0
Bit 12: Sent 0, Received 0
Bit 13: Sent 1, Received 1
Bit 14: Sent 1, Received 1
Bit 15: Sent 1, Received 1
Bit 16: Sent 0, Received 0
Bit 17: Sent 1, Received 1
Bit 18: Sent 1, Received 1
Bit 19: Sent 1, Received 1
Bit 20: Sent 0, Received 0
Bit 21: Sent 0, Received 0
Bit 22: Sent 1, Received 1
Bit 23: Sent 1, Received 1
Bit 24: Sent 0, Received 0
Bit 25: Sent 0, Received 0
Bit 26: Sent 1, Received 1
Bit 27: Sent 0, Received 0
Bit 28: Sent 0, Received 0
Bit 29: Sent 0, Received 0
Bit 30: Sent 1, Received 1
Bit 31: Sent 0, Received 0
```

خروجی دوم :

این خروجی نشان‌دهنده یک انتقال موفق و بدون خطا از TX1 به RX2 استفاده از کدگذاری 5B/6B است. در این روش، هر ۵ بیت داده‌ی ورودی به ۶ بیت تبدیل شده تا پایداری انتقال افزایش یابد و احتمال خطا کاهش پیدا کند. جدول نمایش داده‌شده تأیید می‌کند که هر بیت به درستی ارسال و دریافت شده و هیچ خطایی در مسیر انتقال رخ نداده است، که نشان‌دهنده عملکرد صحیح سیستم ارتباطی است.

```
Encoding received data into 5B/6B format...
Block 0 - Original 5b: 00010 -> Encoded 6b: 101101
Block 1 - Original 5b: 10100 -> Encoded 6b: 001011
Block 2 - Original 5b: 00011 -> Encoded 6b: 110001
Block 3 - Original 5b: 10111 -> Encoded 6b: 111010
Block 4 - Original 5b: 00110 -> Encoded 6b: 011001
Block 5 - Original 5b: 01000 -> Encoded 6b: 111001
Block 6 - Original 5b: 10000 -> Encoded 6b: 011011
Block 7 - Original 5b: 00010 -> Encoded 6b: 101101
Block 8 - Original 5b: 01011 -> Encoded 6b: 110100
Block 9 - Original 5b: 11010 -> Encoded 6b: 010110
Block 10 - Original 5b: 01011 -> Encoded 6b: 110100
Block 11 - Original 5b: 10101 -> Encoded 6b: 101010
Block 12 - Original 5b: 01010 -> Encoded 6b: 010101
Block 13 - Original 5b: 10101 -> Encoded 6b: 101010
Block 14 - Original 5b: 10000 -> Encoded 6b: 011011
Block 15 - Original 5b: 01001 -> Encoded 6b: 100101
Block 16 - Original 5b: 01010 -> Encoded 6b: 010101
Block 17 - Original 5b: 10000 -> Encoded 6b: 011011
Block 18 - Original 5b: 10101 -> Encoded 6b: 101010
Block 19 - Original 5b: 11111 -> Encoded 6b: 101011
Block 20 - Original 5b: 10010 -> Encoded 6b: 010011
Block 21 - Original 5b: 10000 -> Encoded 6b: 011011
Block 22 - Original 5b: 11011 -> Encoded 6b: 110110
Block 23 - Original 5b: 10001 -> Encoded 6b: 100011
Block 24 - Or Encoded 6b: 010111
Block 25 - Original 5b: 00010 -> Encoded 6b: 101101
```

خروجی سوم: (second transmission)

یک انتقال داده کامل و بدون خطا از TX2 به RX1 را نشان می دهد، با هر بیت فردی که به درستی ارسال و دریافت شده است.

هر خط نشان‌دهنده‌ی یک بیت ارسال‌شده و مقدار دریافتی آن است.
تمام ۱۵۶ بیت به درستی ارسال و دریافت شده اند.

```
Starting second transmission (156 encoded bits) - TX2 -> RX1...
Bit 0: Sent 1, Received 1
Bit 1: Sent 0, Received 0
Bit 2: Sent 1, Received 1
Bit 3: Sent 1, Received 1
Bit 4: Sent 0, Received 0
Bit 5: Sent 1, Received 1
Bit 6: Sent 0, Received 0
Bit 7: Sent 0, Received 0
Bit 8: Sent 1, Received 1
Bit 9: Sent 0, Received 0
Bit 10: Sent 1, Received 1
Bit 11: Sent 1, Received 1
Bit 12: Sent 1, Received 1
Bit 13: Sent 1, Received 1
Bit 14: Sent 0, Received 0
Bit 15: Sent 0, Received 0
Bit 16: Sent 0, Received 0
Bit 17: Sent 1, Received 1
Bit 18: Sent 1, Received 1
Bit 19: Sent 1, Received 1
Bit 20: Sent 1, Received 1
Bit 21: Sent 0, Received 0
Bit 22: Sent 1, Received 1
Bit 23: Sent 0, Received 0
Bit 24: Sent 0, Received 0
Bit 25: Sent 1, Received 1
Bit 26: Sent 1, Received 1
Bit 27: Sent 0, Received 0
Bit 28: Sent 0, Received 0
Bit 29: Sent 1, Received 1
Bit 30: Sent 1, Received 1
Bit 31: Sent 1, Received 1
Bit 32: Sent 1, Received 1
Bit 33: Sent 0, Received 0
```

خروجی چهارم:
داده های دریافت شده با کدگذاری ۶ بیتی به داده های اصلی ۵ بیتی رمزگشایی شده اند، بلوک به بلوک.

همانطور که در تصویر خروجی زیر مشخص است به عنوان مثال، در بلوک ۰، داده های ۶ بیتی دریافت شده "۱۰۱۱۰۱" به داده های ۵ بیتی "۰۰۰۱۰" رمزگشایی می شوند
این فرآیند رمزگشایی برای تمام ۲۵ بلوک داده تکرار می شود:

```
Decoding received data and displaying on LEDs...
Block 0 - Received 6b: 101101 -> Decoded 5b: 00010
Block 1 - Received 6b: 001011 -> Decoded 5b: 10100
Block 2 - Received 6b: 110001 -> Decoded 5b: 00011
Block 3 - Received 6b: 111010 -> Decoded 5b: 10111
Block 4 - Received 6b: 011001 -> Decoded 5b: 00110
Block 5 - Received 6b: 111001 -> Decoded 5b: 01000
Block 6 - Received 6b: 011011 -> Decoded 5b: 10000
Block 7 - Received 6b: 101101 -> Decoded 5b: 00010
Block 8 - Received 6b: 110100 -> Decoded 5b: 01011
Block 9 - Received 6b: 010110 -> Decoded 5b: 11010
Block 10 - Received 6b: 110100 -> Decoded 5b: 01011
Block 11 - Received 6b: 101010 -> Decoded 5b: 10101
Block 12 - Received 6b: 010101 -> Decoded 5b: 01010
Block 13 - Received 6b: 101010 -> Decoded 5b: 10101
Block 14 - Received 6b: 011011 -> Decoded 5b: 10000
Block 15 - Received 6b: 100101 -> Decoded 5b: 01001
Block 16 - Received 6b: 010101 -> Decoded 5b: 01010
Block 17 - Received 6b: 011011 -> Decoded 5b: 10000
Block 18 - Received 6b: 101010 -> Decoded 5b: 10101
Block 19 - Received 6b: 101011 -> Decoded 5b: 11111
Block 20 - Received 6b: 010011 -> Decoded 5b: 10010
Block 21 - Received 6b: 011011 -> Decoded 5b: 10000
Block 22 - Received 6b: 110110 -> Decoded 5b: 11011
Block 23 - Received 6b: 100011 -> Decoded 5b: 10001
Block 24 - Received 6b: 010111 -> Decoded 5b: 01111
Block 25 - Received 6b: 101101 -> Decoded 5b: 00010

Checking decode errors (original vs decoded):

Transmission complete. Total errors: 0
SUCCESS: All bits transmitted and decoded correctly!
```

قسمت دوم پروژه: (encoding error)

کد زده شده :

این کد به این صورت است که داده‌ها را با استفاده از پروتکل B/6B۵ و بررسی CRC برای اطمینان از صحت انتقال ارسال و دریافت می‌کند. تابع calculateCRC8 () برای محاسبه CRC8 داده‌ها استفاده می‌شود. در setup()، پین‌ها تنظیم و داده‌ها آماده می‌شوند. initialize_data() داده‌های اولیه را با یک بیت ثابت و بقیه به صورت تصادفی مقداردهی می‌کند. print_binary_array() برای نمایش آرایه‌های باینری استفاده می‌شود. در first_transmission()، داده‌های اصلی به RX2 ارسال و CRC محاسبه می‌شود.
encode_received_data() داده‌های دریافتی را به فرمت B/6B۵ رمزگذاری می‌کند.
second_transmission() داده‌های رمزگذاری شده را به RX1 ارسال و CRC آن‌ها را بررسی می‌کند.
decode_and_display() داده‌های دریافتی را رمزگشایی کرده و بر روی LEDها نمایش می‌دهد. در نهایت، loop() این مراحل را به طور مکرر اجرا کرده و تعداد خطاهای CRC را ثبت می‌کند.

قسمتی از کد زده شده:

(فایل کامل کد قرار داده شده)

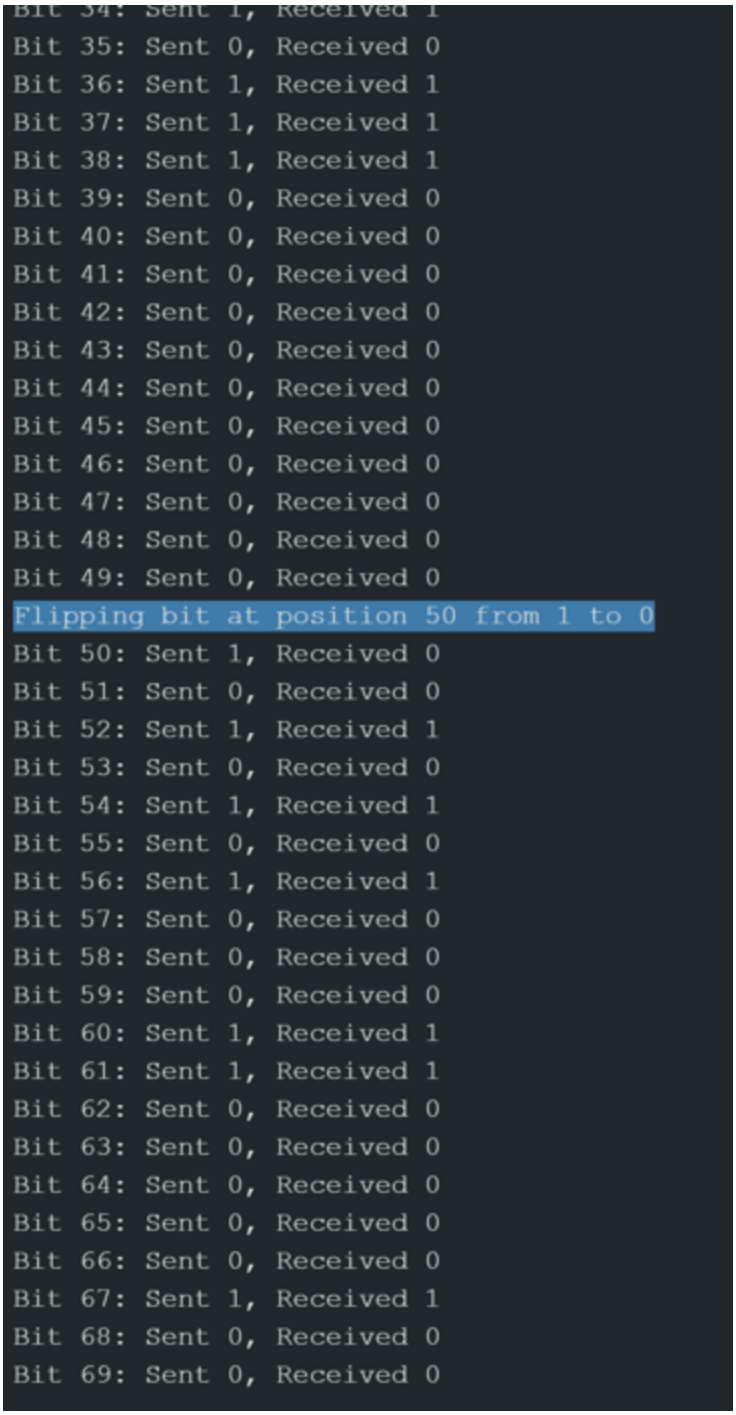
```
    }
}

void encode_received_data() {
    Serial.println("\n=== Encoding Stage ===");

    if (received_first[0] != 0 || received_first[1] != 1) {
        Serial.println("ERROR: First two bits must be 0 and 1");
        while(1);
    }

    received_first[0] = 0;
    received_first[1] = 0;

    for (int block = 0; block < 26; block++) {
        Serial.print("Block ");
        Serial.print(block);
        Serial.print(" (5b->6b): ");

        for (int j = 0; j < 5; j++) {
            Serial.print(received_first[(block * 5) + j]);
        }
        Serial.print(" -> ");

        int index = 0;
        for (int j = 0; j < 5; j++) {
            index = (index << 1) | received_first[(block * 5) + j];
        }
        for (int j = 0; j < 6; j++) {
            encoded_data[(block * 6) + j] = ENCODE_5B6B[index][j];
            Serial.print(encoded_data[(block * 6) + j]);
        }
        Serial.println();
    }

    print_binary_array("Complete encoded data (156b): ", encoded_data, 156);
}

void second_transmission() {
    Serial.println("\n=== Second Transmission Stage (TX2 -> RX1) ===");

    for (int block = 0; block < 26; block++) {
        Serial.print("\nBlock ");
        Serial.print(block);
        Serial.println(":");

        byte original_block_crc = calculateCRC8(&encoded_data[block * 6], 6);
        byte received_block_crc = 0;
        int bit_pos = random(6);
        for (int i = 0; i < 6; i++) {
            if(i == bit_pos){
              Serial.print(" from block: ");
              Serial.print(block);
              Serial.print(" from ");
              Serial.print(encoded_data[(block * 6) + i]);
              Serial.print(" to ");
             bool new_bit = (encoded_data[(block * 6) + i] == 0 ? 1 : 0);
             Serial.println(new_bit);
             digitalWrite(TX_PIN_2, new_bit);
            }
            else {
                digitalWrite(TX_PIN_2, encoded_data[(block * 6) + i]);
            }
            delay(10);
            received_second[(block * 6) + i] = digitalRead(RX_PIN_1);
```

خروجی اول:

این تصویر مرحله اول انتقال داده (TX1 -> RX2) را نشان می‌دهد. شامل داده اصلی ۱۳۰ بیتی و وضعیت ارسال و دریافت هر بیت است.

```
=== First Transmission Stage (TX1 -> RX2) ===
Original Data (130b):  01010000 01110000 01110101 10011100 01101110 00000000 00101010 10001100 00010001 11100101 01110001 10000101 10111010 00110100 01011000 11000010 11
Bit 0: Sent 0, Received 0
Bit 1: Sent 1, Received 1
Bit 2: Sent 0, Received 0
Bit 3: Sent 1, Received 1
Bit 4: Sent 0, Received 0
Bit 5: Sent 0, Received 0
Bit 6: Sent 0, Received 0
Bit 7: Sent 0, Received 0
Bit 8: Sent 0, Received 0
Bit 9: Sent 1, Received 1
Bit 10: Sent 1, Received 1
Bit 11: Sent 1, Received 1
Bit 12: Sent 0, Received 0
Bit 13: Sent 0, Received 0
Bit 14: Sent 0, Received 0
Bit 15: Sent 0, Received 0
Bit 16: Sent 0, Received 0
Bit 17: Sent 1, Received 1
Bit 18: Sent 1, Received 1
Bit 19: Sent 1, Received 1
Bit 20: Sent 0, Received 0
Bit 21: Sent 1, Received 1
Bit 22: Sent 0, Received 0
Bit 23: Sent 1, Received 1
Bit 24: Sent 1, Received 1
Bit 25: Sent 0, Received 0
Bit 26: Sent 0, Received 0
Bit 27: Sent 1, Received 1
Bit 28: Sent 1, Received 1
Bit 29: Sent 1, Received 1
Bit 30: Sent 0, Received 0
Bit 31: Sent 0, Received 0
```

خروجی دوم:
مرحله دوم انتقال داده (TX1 -> RX2) نمایش داده شده است. در این مرحله، بیت ۵۰ از حالت ۱ به ۰ تغییر یافته است. بقیه بیت‌ها نیز به همراه وضعیت ارسال و دریافت آن‌ها نشان داده شده است



خروجی سوم:

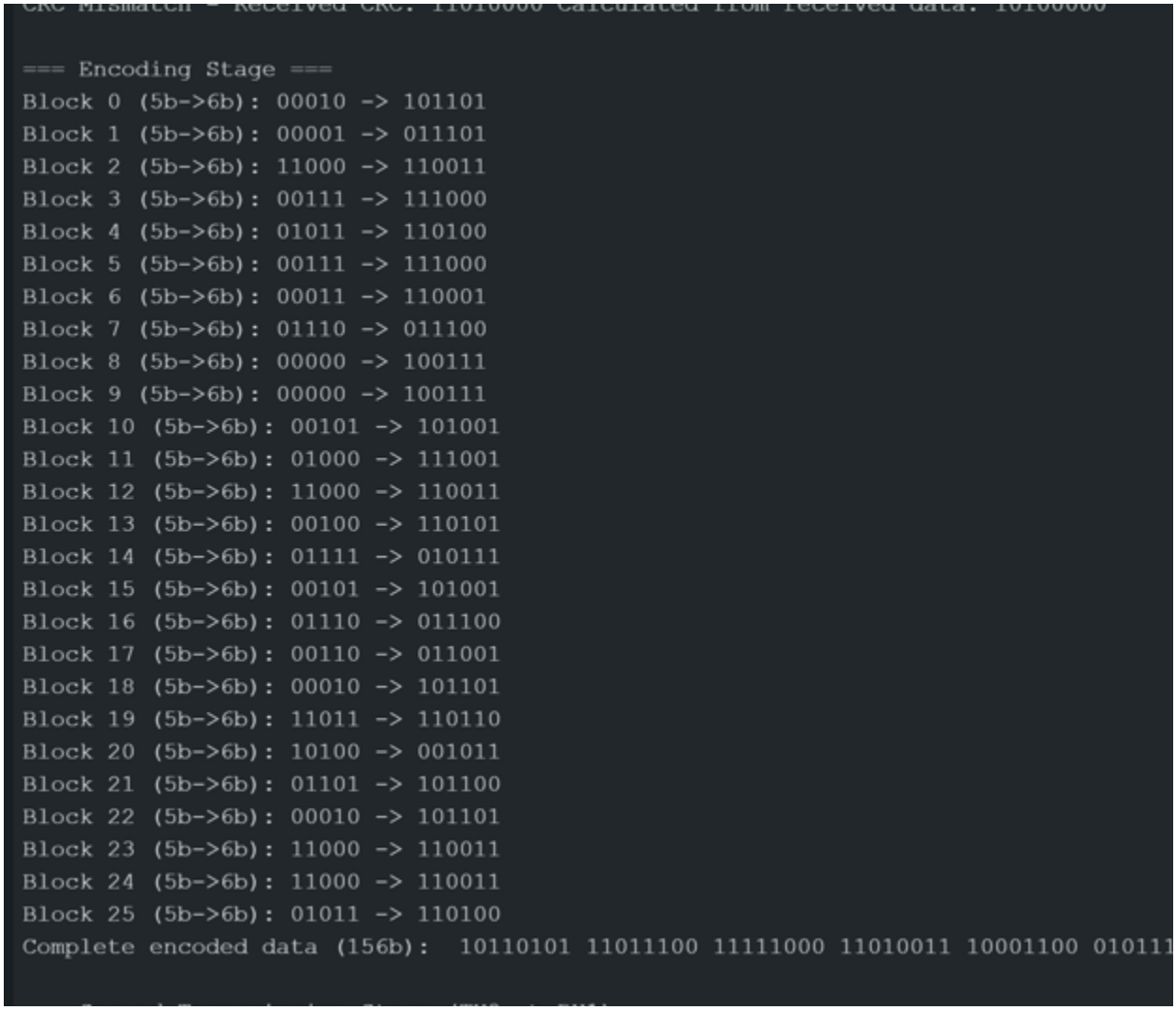
CRC دریافت شده ۱۱۰۱۰۰۰۰، در حالی که محاسبه شده از داده دریافت شده ۱۰۱۰۰۰۰۰ است.

در این مرحله، داده دریافت شده صحیح نبوده و CRC آن نیز با داده محاسبه شده مطابقت ندارد
.

```
Bit 111: Sent 0, Received 0
Bit 112: Sent 0, Received 0
Bit 113: Sent 1, Received 1
Bit 114: Sent 0, Received 0
Bit 115: Sent 1, Received 1
Bit 116: Sent 1, Received 1
Bit 117: Sent 0, Received 0
Bit 118: Sent 0, Received 0
Bit 119: Sent 0, Received 0
Bit 120: Sent 1, Received 1
Bit 121: Sent 1, Received 1
Bit 122: Sent 0, Received 0
Bit 123: Sent 0, Received 0
Bit 124: Sent 0, Received 0
Bit 125: Sent 0, Received 0
Bit 126: Sent 1, Received 1
Bit 127: Sent 0, Received 0
Bit 128: Sent 1, Received 1
Bit 129: Sent 1, Received 1
Received Data (130b):  01010000 01110000 01110101 10011100 01101110 00000000 00001010 10001100 00010001 11100101 01110001 10000101 10111010 00110100 01011000 11000010 11
CRC Mismatch - Received CRC: 11010000 Calculated from received data: 10100000

=== Encoding Stage ===
Block 0 (5b->6b): 00010 -> 101101
Block 1 (5b->6b): 00001 -> 011101
Block 2 (5b->6b): 11000 -> 110011
Block 3 (5b->6b): 00111 -> 111000
Block 4 (5b->6b): 01011 -> 110100
Block 5 (5b->6b): 00111 -> 111000
Block 6 (5b->6b): 00011 -> 110001
Block 7 (5b->6b): 01110 -> 011100
Block 8 (5b->6b): 00000 -> 100111
Block 9 (5b->6b): 00000 -> 100111
Block 10 (5b->6b): 00101 -> 101001
Block 11 (5b->6b): 01000 -> 111001
Block 12 (5b->6b): 11000 -> 110011
Block 13 (5b->6b): 00100 -> 110101
```

خروجی چهارم:

در این مرحله، داده کدگذاری شده به صورت کامل ارائه شده و مراحل کدگذاری و بازکدگذاری هر بلوک ۵ بیتی به ۶ بیتی مشخص است.



خروجی پنجم :

در این مرحله انتقال داده، CRC برای بلوک‌های دریافت شده با محاسبات انجام شده مطابقت ندارد. این نشان می‌دهد که داده‌های دریافت شده دچار خطا شده‌اند

خروجی ششم :

مرحله رمزگشایی نشان می‌دهد که اکثر بلوک‌ها (۰-۶، ۱۱، ۱۵، ۱۶، ۲۰، ۲۳) به عنوان "INVALID" علامت‌گذاری شده‌اند، به این معنی که داده‌های آن بلوک‌ها به درستی رمزگشایی نشده‌اند.

برخی بلوک‌ها (۷-۱۰، ۱۲-۱۴، ۱۷-۱۹، ۲۱-۲۲، ۲۴-۲۵) با موفقیت رمزگشایی شده‌اند و داده نهایی رمزگشایی شده در انتها نمایش داده شده است.

```
Bit 2: Sent 0, Received 0
Bit 3: Sent 1, Received 1
 from block: 25 from 0 to 1
Bit 4: Sent 0, Received 1
Bit 5: Sent 0, Received 0
CRC Mismatch in block 25 - Received CRC: 1110000, Calculated from received data: 11000110

=== Decoding Stage ===
Block 0: 101111 -> INVALID
Block 1: 111101 -> INVALID
Block 2: 111011 -> INVALID
Block 3: 011000 -> INVALID
Block 4: 100100 -> INVALID
Block 5: 111100 -> INVALID
Block 6: 100001 -> INVALID
Block 7: 011110 -> 11110
Block 8: 100110 -> 11001
Block 9: 100110 -> 11001
Block 10: 101101 -> 00010
Block 11: 111101 -> INVALID
Block 12: 110001 -> 00011
Block 13: 100101 -> 01001
Block 14: 010101 -> 01010
Block 15: 101000 -> INVALID
Block 16: 011000 -> INVALID
Block 17: 011101 -> 00001
Block 18: 100101 -> 01001
Block 19: 100110 -> 11001
Block 20: 000011 -> INVALID
Block 21: 101101 -> 00010
Block 22: 100101 -> 01001
Block 23: 111011 -> INVALID
Block 24: 110010 -> 10011
Block 25: 110110 -> 11011

Final decoded data (130b):  00000000 00000000 00000000 00000000 00011110 11001110 01000100
```

خروجی هفتم :

این تصویر نشان می‌دهد که اکثر بیت‌ها و بلوک‌ها به درستی منتقل شده‌اند، اما تعدادی از بلوک‌ها دارای خطای CRC بوده‌اند. همچنین، تعداد خطاهای CRC در انتقال دوم افزایش یافته است

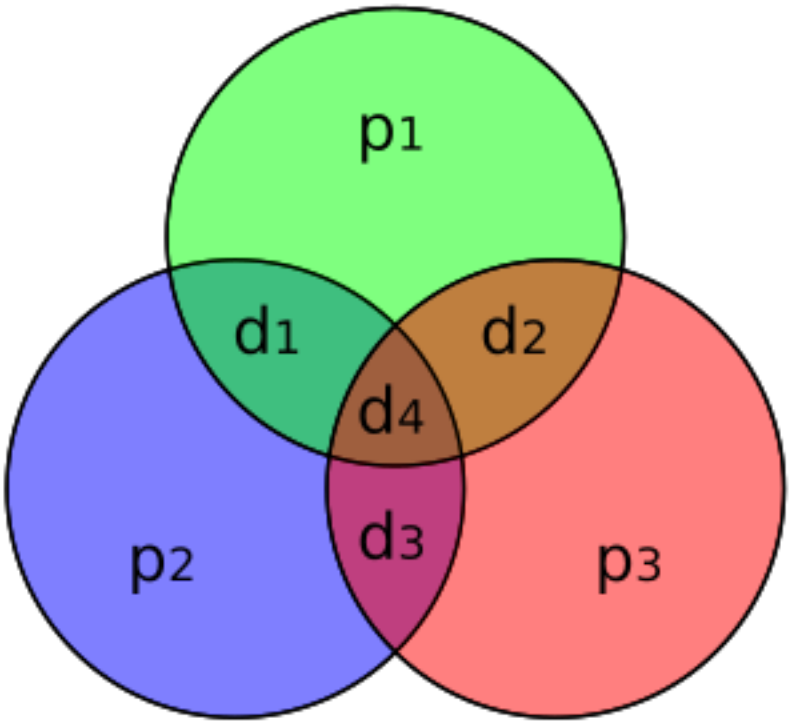
```
Block 8: 100110 -> 11001
Block 9: 100110 -> 11001
Block 10: 101101 -> 00010
Block 11: 111101 -> INVALID
Block 12: 110001 -> 00011
Block 13: 100101 -> 01001
Block 14: 010101 -> 01010
Block 15: 101000 -> INVALID
Block 16: 011000 -> INVALID
Block 17: 011101 -> 00001
Block 18: 100101 -> 01001
Block 19: 100110 -> 11001
Block 20: 000011 -> INVALID
Block 21: 101101 -> 00010
Block 22: 100101 -> 01001
Block 23: 111011 -> INVALID
Block 24: 110010 -> 10011
Block 25: 110110 -> 11011

Final decoded data (130b):  00000000 00000000 00000000 00000000 00011110 11001110 01000100 000000

Total CRC errors - First transmission: 1, Second transmission: 26

=== Starting new transmission cycle ===

=== First Transmission Stage (TX1 -> RX2) ===
Original Data (130b):  01011101 00111010 11010111 01011010 11111000 10100101 10101001 11110011 01
Bit 0: Sent 0, Received 0
Bit 1: Sent 1, Received 1
Bit 2: Sent 0, Received 0
Bit 3: Sent 1, Received 1
Bit 4: Sent 1, Received 1
Bit 5: Sent 1, Received 1
Bit 6: Sent 0, Received 0
Bit 7: Sent 1, Received 1
Bit 8: Sent 0, Received 0
Bit 9: Sent 0, Received 0
```

تصحیح خطا

در نهایت می‌خواهیم به تصحیح خطا بپردازیم. با توجه به قسمت‌های قبل ما الان می‌توانیم متوجه وجود خطا در داده ارسالی شویم. سوال این است که چگونه می‌توانیم داده خطادار با به حالت درست خود برگردانیم؟ ساده‌ترین روش درخواست مجدد داده است. در ادامه دو روش برای تصحیح خطا ارائه می‌دهیم.

نحوه عملکرد کدHamming ،افزودن بیت‌های توازن(Parity Bits)

کد Hamming یکی از اولین روش‌های تصحیح خطا است که توسط Richard Hamming معرفی شد. این روش برای تصحیح تک‌بیتی و دو بیتی (Double-bit error detection) مناسب است.
در یک داده‌ی k بیتی، تعداد اضافی r بیت به داده اضافه می‌شود که برای شناسایی و تصحیح خطا به کار می‌رود. یک استانداردی برای تعداد بیت‌های اضافی هست. به عنوان مثال برای یک پیام ۴ بیتی ۳ بیت توازن نیاز داریم.
هر بیت توازن یک ترکیب خاص از بیت‌های داده را بررسی می‌کند و مقدار آن بر اساس XOR سایر بیت‌ها تنظیم می‌شود. گیرنده با استفاده از همان XOR بررسی می‌کند که آیا خطایی رخ داده است یا نه. اگر خطایی رخ دهد، محل دقیق آن را تشخیص داده و تصحیح می‌کند.



BCH (Bose-Chaudhuri-Hocquenghem)الگوریتم

کدهای BCH یک دسته از کدهای تصحیح خطا هستند که برای تشخیص و تصحیح چندین بیت خطا در داده‌های دیجیتال طراحی شده‌اند. این کدها در سیستم‌های مخابراتی، ذخیره‌سازی داده‌ها مانند حافظه‌های NAND Flash، و ارتباطات بی‌سیم مورد استفاده قرار می‌گیرند.
برای یک عدد اول q و دو عدد صحیح مثبت m, d به طوریکه

$$d \leq q^m - 1 = n$$

کد BCH با ساخت یکسری چندجمله‌ای و ک.م.م گرفتن آنها، چندجمله‌ای مولد را می‌سازد که می‌تواند برای y خطا در n-y بیت داده، تصحیح خطا انجام دهد. برای مثال اگر فرض کنیم

$$q = 2\, m = 4 \Rightarrow n = 15$$

با گرفتن

$$g(x) = x^4 + x + 1$$

می‌تواند تا یک خطا را تشخیص دهد. چون درجه آن ۴ است، به ۴ بیت برای اضافه شدن به هر ۱۱ بیت داده نیاز داریم. برای اطلاعات بیشتر به اینجا مراجعه کنید.

پیاده سازی در عمل

ابتدا ۶ لامپ میگذاریم و از پین‌های ۲ تا ۷ وصل می‌کنیم به سر مثبت هر لامپ. از طرف دیگر GND آردیونو را بهشان با مقاومت ۱ کیلو اهمی وصل می‌کنیم (بصورت متوالی). پین ۸ و ۹ می‌شوند RX_1, Tx_1 و پین ۱۰و ۱۱ می‌شوند Rx_2, Tx_2. ما در شکل زیر برای اتصال، میایم Rx_1 را به Tx_2 و Tx_1 را به Rx_2 وصل می‌کنیم.