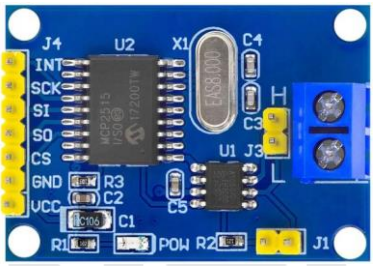
CAN’den LIN'e Veri Transferi ve RGB LED Kontrolü



Hazırlayan: Mesut ULUSOY

Temmuz, 2024
İSTANBUL

1. Giriş

Uygulamanın Amacı

Bu uygulamanın amacı, **PCAN-USB Pro** üzerinden alınan CAN (Controller Area Network) verilerini, **MCP2515** CAN denetleyicisi ve **Arduino Uno** kullanarak okumak ve bu verileri **TJA1020 LIN Modülü** yardımıyla **LIN Bus**'a iletmektir. LIN Bus üzerinden aktarılan veriler, RGB LED'in farklı renklerde yanmasını sağlar. Farklı CAN verileri, LED'in farklı renklerde yanmasına neden olacak şekilde ayarlanmıştır.

Kısaca LIN Bus Nedir?

LIN (Local Interconnect Network) Bus, düşük maliyetli bir iletişim protokolüdür ve genellikle otomotiv endüstrisinde kullanılır. LIN, araç içerisindeki elektronik kontrol üniteleri (ECU) arasında haberleşmeyi sağlamak için geliştirilmiş bir seri iletişim ağıdır. Özellikle düşük hız ve düşük maliyet gerektiren uygulamalar için tasarlanmıştır.

Seri Tek Yönlü ve Broadcast Mesajlaşma Sistemi: LIN Bus, seri tek yönlü ve yayın (broadcast) tabanlı bir mesajlaşma sistemi olarak çalışır. Bu sistemde, tek bir master cihazı (komutan) ve en fazla on altı adreslenebilir slave cihazı (cevaplayıcı) bulunur. Master cihazı, iletişimi kontrol eder ve mesajların zamanlamasını belirler. Slave cihazları ise sadece master cihazından gelen mesajlara yanıt verirler. Bu yapı, iletişimin basit ve düşük maliyetli olmasını sağlar.

İletişim, UART (Universal Asynchronous Receiver-Transmitter) tabanlı bir seri link katmanı kullanılarak gerçekleştirilir. Bu katman, start, veri ve stop bitlerini içerir. LIN Bus, tek bir kablo (ve toprak) üzerinden iletişim sağlar ve 12V bus voltajı kullanır, bu da onu otomotiv uygulamaları için ideal kılar. Tüm iletişim master cihazı tarafından başlatılır ve tek bir slave cihazı yanıt verir, bu da çakışma riskini ortadan kaldırır ve arbiter ihtiyacını elimine eder.

Bit Rate Bilgisi: LIN Bus, 1 kbps ile 20 kbps arasında değişen bit hızlarına sahiptir. Bu düşük bit hızı, sistemin düşük maliyetli ve enerji verimli olmasını sağlar. Düşük bit hızları, yüksek veri hızlarının gerekmediği uygulamalar için yeterlidir, örneğin araç içerisindeki basit sensör ve aktüatör kontrolü gibi. Baud rate yapılandırılabilir ve uygulamanın gereksinimlerine göre ayarlanabilir.

Çatışma (Collision) ve Arbiter Yokluğu: LIN Bus'ta, bus arbiteri veya çakışma algılama mekanizması bulunmaz. Tüm iletişim master cihazı tarafından başlatılır ve tek bir slave cihazı yanıt verir. Bu, çakışma riskini sıfıra indirir ve iletişimi basitleştirir.

LIN Frame Formatı

LIN Bus'ta veri iletişimi, belirli bir formatta çerçeveler (frames) halinde gerçekleştirilir. Bir LIN çerçevesi, başlık (header) ve yanıt (response) olmak üzere iki ana bileşenden oluşur. LIN Frame formatı figüre 1’de gösterilmiştir.

Başlık (Header)

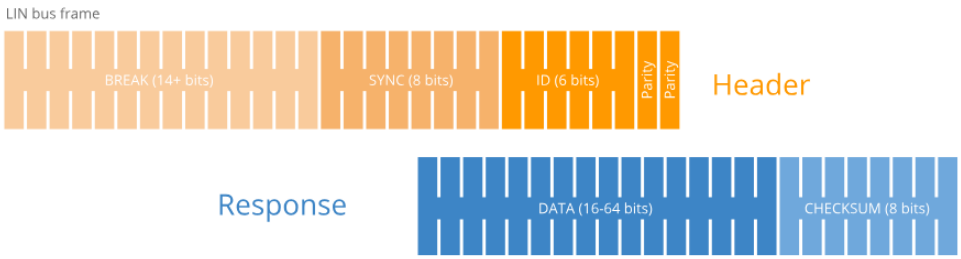
Başlık, genellikle LIN master tarafından LIN bus'a gönderilir. Bu, bir slave cihazını tetikler ve slave cihazı yanıt olarak veri gönderir. Başlık üç ana bölümden oluşur:

- **Break**: Senkronizasyon kırılma alanı (Sync Break Field, SBF) olarak da bilinen Break, minimum 13 + 1 bit uzunluğundadır (pratikte genellikle 18 + 2 bit). Break alanı, LIN bus üzerindeki tüm düğümlere (nodes) bir "çerçeve başlangıcı" bildirimi olarak işlev görür.
- **Sync**: 8 bitlik Sync alanı, 0x55 (ikilik olarak 01010101) önceden tanımlanmış bir değere sahiptir. Bu yapı, LIN düğümlerinin yükselen/düşen kenarlar arasındaki zamanı belirlemesine ve bu sayede master düğüm tarafından kullanılan baud hızını öğrenmesine olanak tanır. Bu, her bir düğümün senkronize kalmasını sağlar.
- **Identifier (Kimlik)**: Kimlik alanı, 6 bitlik kimlik ve 2 bitlik hata kontrol bitinden oluşur. Kimlik, gönderilen her LIN mesajının kimliğini belirler ve hangi düğümlerin başlığa tepki vereceğini belirler. Slave cihazları, kimlik alanının geçerliliğini (hata kontrol bitlerine dayanarak) belirler ve aşağıdaki şekillerde hareket eder:
    - Sonraki veri iletimini görmezden gelir.
    - Başka bir düğümden iletilen veriyi dinler.
    - Başlığa yanıt olarak veri yayınlar. Tipik olarak, bir slave cihazı bilgi almak için sorgulanır, bu da çarpışma riskini sıfıra indirir (bu nedenle arbiter ihtiyacı yoktur). 6 bitlik kimlik 64 kimlik oluşturulmasını sağlar, bunlardan kimlik 60-61 tanılama için kullanılır ve 62-63 ayrılmıştır.

Yanıt (Response)

Yanıt, bir LIN slave cihazı tarafından başlığa yanıt olarak gönderilen veri alanını içerir. Yanıt iki ana bölümden oluşur:

- **Data (Veri)**: Bir LIN slave, master tarafından sorgulandığında, 2, 4 veya 8 bayt veri ile yanıt verebilir. Veri uzunluğu özelleştirilebilir, ancak genellikle kimlik aralığına bağlıdır (kimlik 0-31: 2 bayt, 32-47: 4 bayt, 48-63: 8 bayt). Veri baytları, iletilen gerçek bilgiyi LIN sinyalleri şeklinde içerir. LIN sinyalleri, veri baytları içinde paketlenmiştir ve örneğin sadece 1 bit uzunluğunda veya birden fazla bayt olabilir.
- **Checksum (Hata Kontrol)**: CAN'de olduğu gibi, bir hata kontrol alanı (checksum) LIN çerçevesinin geçerliliğini sağlar. Klasik 8 bitlik hata kontrolü, yalnızca veri baytlarının toplanmasına dayanır (LIN 1.3), gelişmiş hata kontrol algoritması ise kimlik alanını da içerir (LIN 2.0).



Figür 1: LIN Frame

Kısaca CAN Bus Nedir?

Controller Area Network (CAN) Bus, yüksek hızlı ve güvenilir bir veri iletişim protokolüdür. İlk olarak Bosch tarafından 1980'lerin ortalarında geliştirilmiş ve otomotiv endüstrisinde yaygın olarak kullanılmaktadır. CAN Bus, mikrodenetleyiciler ve cihazlar arasında gerçek zamanlı veri iletimini sağlar. Figür 2'de 1 çerçevenin nasıl göründüğü gösterilmiştir.

Temel Özellikler

Çok Düşük Hata Oranı: CAN Bus, hata tespit ve düzeltme mekanizmaları sayesinde güvenilir veri iletimi sağlar.

Yüksek Hız: CAN Bus, 1 Mbps'ye kadar veri iletim hızlarına ulaşabilir.

Çoklu Ana Bilgisayar (Multimaster) İletişim: Ağda birden fazla cihaz aynı anda veri gönderebilir ve alabilir.

Düşük Maliyet: CAN Bus, kablolama maliyetlerini azaltarak ekonomik bir çözüm sunar.

Çalışma Prensibi

CAN Bus, mesaj tabanlı bir iletişim protokolüdür. Her mesaj, bir identifier (tanımlayıcı) ile birlikte gönderilir. Bu identifier, mesajın önceliğini belirler ve hangi cihazların bu mesajı alacağını belirtir. CAN Bus, iki ana tel üzerinden (CAN_H ve CAN_L) iletişim sağlar ve bu sayede yüksek gürültü bağışıklığına sahiptir.

Veri Çerçevesi (Data Frame)

Bir CAN veri çerçevesi, aşağıdaki bileşenlerden oluşur:

Start of Frame (SOF): Veri çerçevesinin başlangıcını belirtir.

Identifier: Mesajın tanımlayıcısı ve önceliğini belirler.

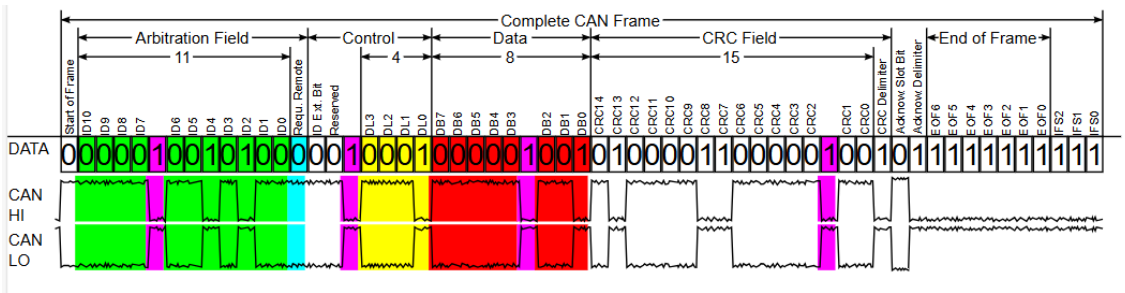
Control Field: Veri uzunluğunu belirtir.

Data Field: Gönderilen verileri içerir (0-8 byte).

CRC Field: Hata tespit için kullanılır.

ACK Field: Alıcının mesajı doğru aldığını onaylar.

End of Frame (EOF): Veri çerçevesinin sonunu belirtir.



Figür 2: CAN Frame

Donanım ve Bağlantılar

PCAN-USB Pro: CAN verilerini bilgisayardan MCP2515 modülüne aktarır.

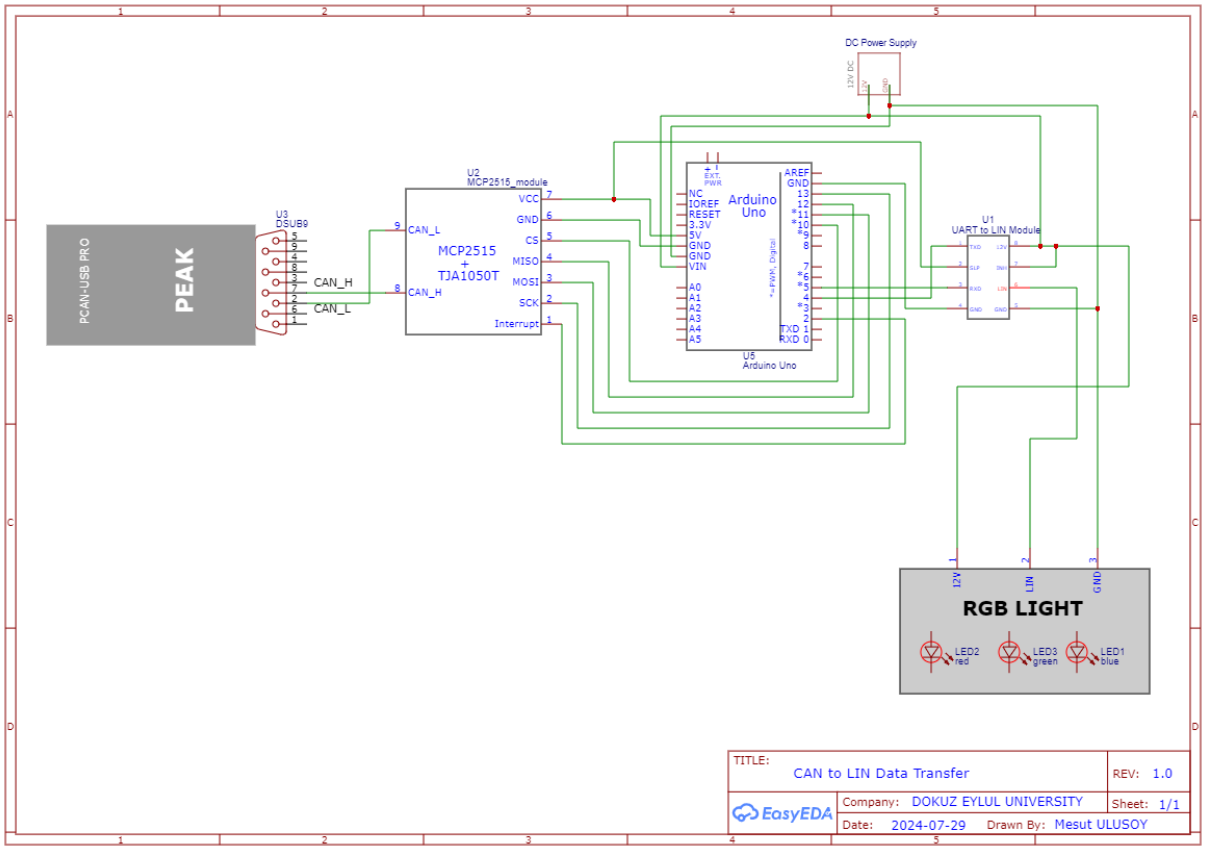
MCP2515 CAN Denetleyici: CAN verilerini alır ve SPI üzerinden Arduino UNO'ya iletir.

Arduino UNO: MCP2515'ten aldığı verileri işler ve LIN Bus üzerinden TJA1021 transceiver aracılığıyla iletir.

TJA1020 LIN Modülü: Arduino UNO'dan gelen verileri LIN Bus'a iletir.

RGB LED: LIN Bus'tan gelen verilere göre farklı renklerde yanar.

Bağlantı Şeması



Figür 3: Bağlantıların Şematik gösterimi

Yazılım Bileşenleri

1. **Arduino IDE**: Arduino Uno'yu programlamak için kullanılan geliştirme ortamıdır. Arduino kodlarının yazılması, derlenmesi ve karta yüklenmesi için kullanılır.
2. **LIN Kütüphanesi**: LIN Bus iletişimini yönetmek ve LIN protokolüne uygun çerçeveler oluşturmak için kullanılan kütüphanedir. Bu kütüphane, LIN Bus üzerinden veri gönderme ve alma işlemlerini kolaylaştırır. Buradaki linkten GitHub sayfasına giderek gerekli kütüphaneyi indirebilirsiniz.
3. **MCP2515 CAN Arayüz Kütüphanesi:** MCP2515 CAN denetleyici modülünü Arduino ile kullanmak için geliştirilmiş kütüphanedir. CAN Bus üzerinden veri gönderme ve alma işlemlerini kolaylaştırır. Kütüphaneyi buradaki linkten GitHub üzerinden indirebilir ve Arduino IDE'ye ekleyebilirsiniz.
4. **PCAN-USB Pro FD kullanıcı arayüzü:** Bu arayüz sayesinde bilgisayardan istediğimiz CAN verilerini ayarlayabiliriz. Buradaki linke giderek indirmeler kısmından PCAN-USB Pro' yu bilgisayarda kullanmak için gerekli olan yazılımı indirebilirsiniz.
5. **Özel Yazılım Kodları**: CAN bus'tan gelen verilerin işlenip LIN bus'a aktarılması için yazılmış özel kodlar. Bu uygulamada Arduino MCP2515 ve LIN kütüphaneleri kullanılmıştır. Bu uygulamada kullanılan kodlara ekler kısmından ulaşabilirsiniz.

2. Yazılım Geliştirme

Bu bölümde, CAN verilerinin MCP2515 modülü aracılığıyla alınması ve bu verilerin TJA1021 transceiver yardımıyla LIN bus'a aktarılması için kullanılan kodun genel yapısını bulabilirsiniz. Ekler kısmındaki kodu aşağıdaki açıklamayla birlikte inceleyebilirsiniz.

Kodun Açıklaması

1. Başlangıç ve Kütüphane Dahil Edilmesi:

Kodun başında gerekli kütüphaneler dahil edilir. **LIN_master_SoftwareSerial.h** ve **mcp2515.h** kütüphaneleri LIN ve CAN iletişimi için gereklidir.

2. Pin Tanımlamaları:

Arduino UNO için kullanılacak pinler tanımlanır. **PIN_TOGGLE, PIN_ERROR, PIN_LIN_RX ve PIN_LIN_TX** gibi pinler, LIN ve CAN haberleşmesi için kullanılır.

3. LIN Node Ayarı:

LIN node'u oluşturulur ve **Rx** ile **Tx** pinleri atanır. Bu node, LIN iletişimini yönetir.

4. CAN İçin Ayarlar:

MCP2515 CAN modülü resetlenir ve baud rate ayarlanır. Normal moda alınarak CAN mesajları dinlenir.

5. **loop()** Fonksiyonu:

Bu fonksiyon, sürekli olarak çalışır ve CAN mesajlarını okur.

Gelen CAN verileri, LIN verilerine dönüştürülür ve LIN bus üzerinden gönderilir.

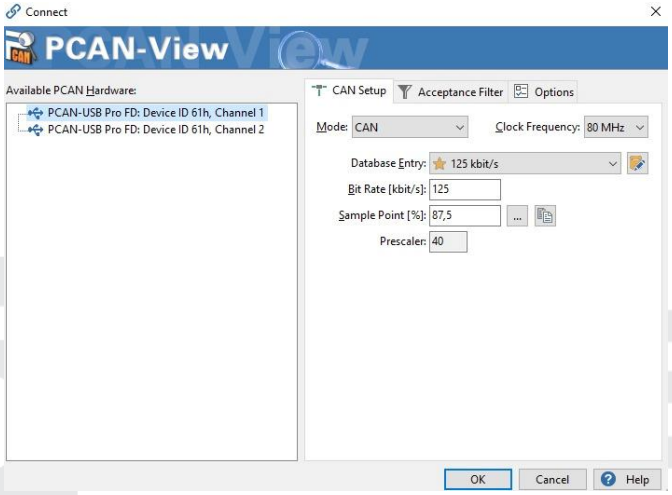
6. CAN Mesajlarının LIN'e Dönüştürülmesi:

CAN bus'tan alınan veriler **Tx_n** adlı bir diziye kopyalanır. Bu dizi, LIN bus üzerinden gönderilecek verileri içerir.

LIN üzerinden veriler gönderildikten sonra, sistem durumu ve hata durumları resetlenir.

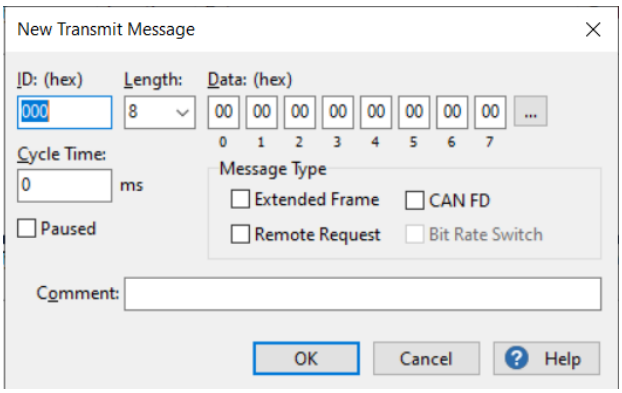
3. Test ve Sonuçlar

Öncelikle bağlantı şemasına göre bağlantılarıları yapıyoruz. Ardından PCAN-View için clock frequency ve data bit rate ayarlarını aşağıdaki gibi yapıyoruz.

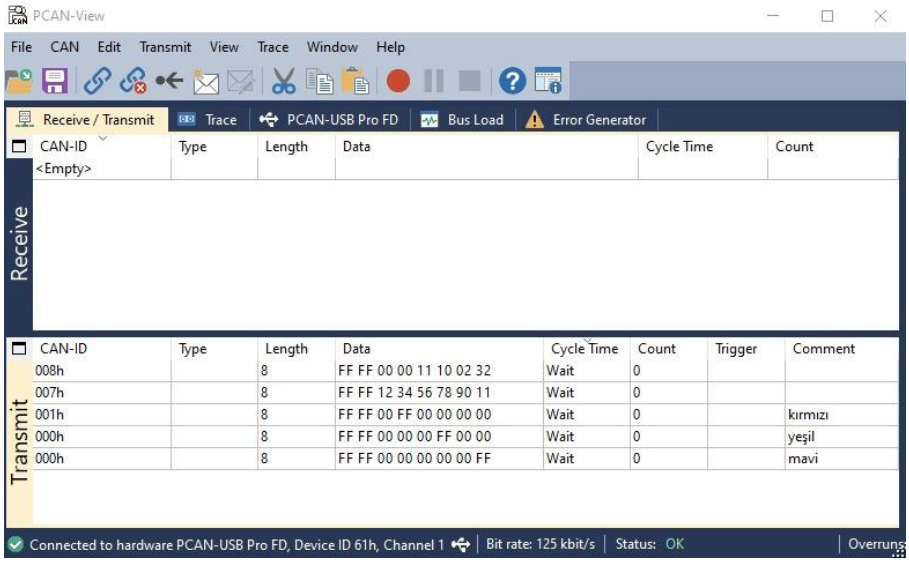


Figür 4: PCAN- View ayarları

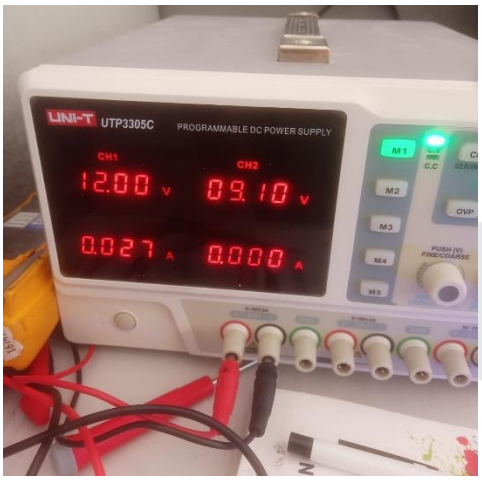
Yollamak istediğimiz mesajları ayarlıyoruz. Figür 6' da kırmızı, yeşil ve mavi için uygun olan mesaj çerçeveleri gösterilmiştir.



Figür 5: Mesaj ayarları

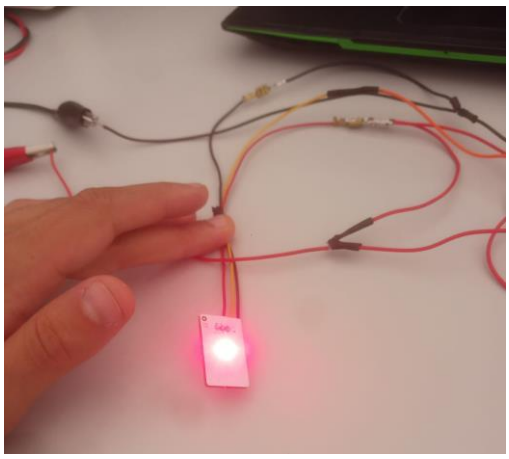


Figür 6: Kırmızı, yeşil ve mavi için CAN verileri



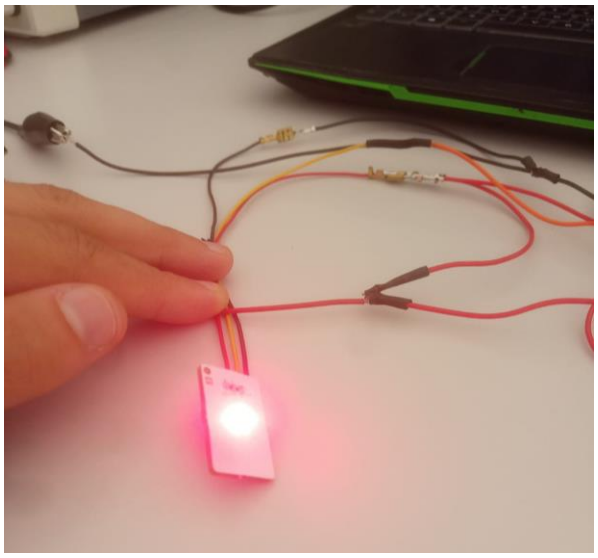
Figür 7: DC Kaynak ayarı

PCAN-View üzerinde **kırmızı** (comment yerinde belirtilmiş) için ayarlanmış kodun üzerine tıklayıp ardından boşluk (space) tuşuna tıkladıktan sonra kırmızı için olan CAN verisi gönderilir ve ledimiz figür 8'de gösterildiği gibi kırmızı yanar.

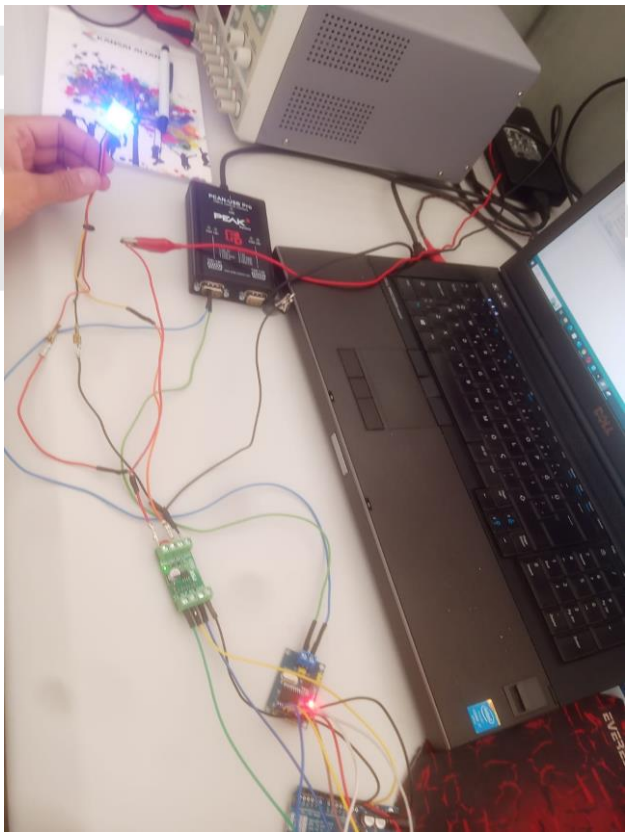


Figüre 8: Kırmızı renk kodu sonrası

Daha sonra klavyeden **ESC tuşuna** bastıktan sonra yeşil koduna tıklayıp klavyeden **boşluk tuşuna** tıklayıp ledimizi yeşil olarak yakıyoruz.

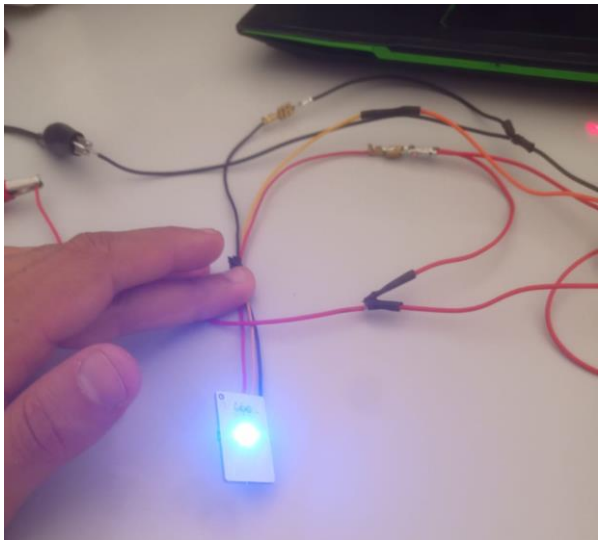


Figüre 8: Yeşil renk kodu sonrası



Figüre 9: Kurulmuş devrenin görüntüsü

Yukarıda belirtilen adımları tekrar ederek mavi şekilde yakabilirsiniz.



Figüre 10: Mavi renk kodu sonrası

SANEL

4. Sonuç ve Değerlendirme

Bu uygulama, CAN Bus verilerini LIN Bus üzerinden ileterek bir RGB LED'in istenilen renklerde yanması sağlanmıştır. Test aşamasında, PCAN-USB Pro FD arayüzü ile gönderilen CAN mesajları, MCP2515 CAN denetleyici modülü ve Arduino Uno aracılığıyla başarılı bir şekilde LIN Bus'a dönüştürülmüş ve TJA1020 LIN modülü üzerinden iletilmiştir. RGB LED'in renk değiştirme fonksiyonu, beklenen şekilde çalışmıştır.

1. **CAN Bus'ın Temel Prensiplerinin Anlaşılması:** CAN Bus protokolünün temel prensipleri, çalışma şekli ve bileşenleri başarıyla öğrenilmiştir.
2. **LIN Bus'ın Temel Prensiplerinin Anlaşılması:** LIN Bus protokolünün temel prensipleri, çalışma şekli ve bileşenleri başarılı bir şekilde kavranmıştır.
3. **Donanım Kurulumu:** Arduino UNO, MCP2515 CAN denetleyici, TJA1020 LIN modülü ve RGB LED gibi bileşenlerin bağlantıları doğru bir şekilde yapılmış ve LIN Bus iletişimi için gerekli donanım altyapısı başarıyla oluşturulmuştur. Donanımın doğru bir şekilde yapılandırılması, sistemin güvenilirliğini ve performansını artırmıştır.
4. **Yazılım Geliştirme:** LIN Bus üzerinden veri göndermek için gerekli olan kütüphaneler kullanılmış ve örnek kodlar bu uygulama için yeniden düzenlenmiştir. Arduino IDE kullanılarak, CAN'den LIN'e veri transferini sağlamak için gereken yazılım kodları başarıyla geliştirilmiş ve Arduino UNO'ya yüklenmiştir.
5. **Veri Gönderme İşlemi:** CAN Bus üzerinden gelen veriler başarılı bir şekilde okunduktan sonra CAN verileri LIN formatına dönüştürülmüş ve RGB LED'in farklı renklerde yanması sağlanmıştır.

Bu aşamalar, uygulamanın temel amaçlarına ulaşıldığını göstermektedir. Özellikle, CAN ve LIN Bus protokollerinin başarılı bir şekilde entegre edilmesi ve LIN Bus iletişiminin sağlanması, uygulamanın en önemli hedefinin gerçekleştirildiğini ortaya koymaktadır.

EKLER

1- Düzenlenmiş kod

```
/*********************
CAN to LIN with PCAN-USB Pro FD, LIN Transceiver, Arduino Uno and MCP2515 CAN Controller

Author of the LIN libraries: Georg - https://github.com/gicking/LIN_master_portable_Arduino
Author of the CAN libraries: autowp- https://github.com/autowp/arduino-mcp2515

Edited by: Mesut - mesut.ulusoy@outlook.com.tr


Supported (=successfully tested) board:
 - Arduino UNO

**********************/

// include files for LIN

#include "LIN_master_SoftwareSerial.h"


// board pin definitions
#if defined(ARDUINO_AVR_UNO)
  #define PIN_TOGGLE    7         // pin to demonstrate background operation
  #define PIN_ERROR     6         // indicate LIN return status
  #define PIN_LIN_RX    4         // receive pin for LIN
  #define PIN_LIN_TX    5         // transmit pin for LIN
#else
  #error adapt parameters to board
#endif

// pause between LIN frames
#define LIN_PAUSE       100

// setup LIN node. Note: not all pins support Rx!
LIN_Master_SoftwareSerial   LIN(PIN_LIN_RX, PIN_LIN_TX, false, "LIN_SW");       // parameter: Rx, Tx, inverseLogic, name

  //uint8_t Txr[8] = {0xFF, 0xFF, 0x00, 0xFF, 0x00, 0x00, 0x00, 0x00};  // RED LED
  //uint8_t Txg[8] = {0xFF, 0xFF, 0x00, 0x00, 0x00, 0xFF, 0x00, 0x00};  // GREEN LED
  uint8_t Txb[8] = {0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF}; // BLUE LED
  uint8_t Tx_n[8]; // new Tx


// include files for CAN
```

```
#include <SPI.h>
#include <mcp2515.h>

struct can_frame canMsg;
MCP2515 mcp2515(10);


void setup() {

  // For CAN

  mcp2515.reset();
  mcp2515.setBitrate(CAN_125KBPS, MCP_8MHZ);
  mcp2515.setNormalMode();

  // For LIN

    // indicate background operation
  pinMode(PIN_TOGGLE, OUTPUT);

  // indicate LIN status via pin
  pinMode(PIN_ERROR, OUTPUT);

  // open LIN connection
  LIN.begin(19200);


}// setup()

void loop() {

  int dummy;

  // FOR CAN
  if (mcp2515.readMessage(&canMsg) == MCP2515::ERROR_OK) {
    /*Serial.print(canMsg.can_id, HEX); // print ID
    Serial.print(" ");
    Serial.print(canMsg.can_dlc, HEX); // print DLC
    Serial.print(" ");

    for (int i_2 = 0; i<canMsg.can_dlc; i_2++)  {  // print the data
      Serial.print(canMsg.data[i_2],HEX);
      Serial.print(" ");
    }

    Serial.println();  */
  }

  // CAN to LIN
    for (int i_2 = 0; i_2 < 8; i_2++) {
      Tx_n[i_2] = (i_2 < canMsg.can_dlc) ? canMsg.data[i_2] : 0x00;
    }
```

SANEL

```
      /* for (int b = 0; i<canMsg.can_dlc; b++)  {

       Tx_n[b] = canMsg.data[b];   // new Tx

     } */


  // FOR LIN
  static uint32_t       tStart;


  LIN_Master::frame_t   Type;
  uint8_t               Id;
  uint8_t               NumData;
  uint8_t               Data[8];
  LIN_Master::error_t   error;

  // send master request frame and get result immediately
  error = LIN.sendMasterRequestBlocking(LIN_Master::LIN_V2, 0x61, 8,
Tx_n); // new Tx

  // reset state machine & error
  LIN.resetStateMachine();
  LIN.resetError();

  // wait a bit. Toggle pin to show CPU load (100ms wait)
  tStart = millis();
  while (millis() - tStart < LIN_PAUSE)
  {
    dummy = dummy +1;;

  }

}
```

ISANEL